A Bound-Independent Pruning Technique to Speeding up Tree-Based Complete Search Algorithms for Distributed Constraint Optimization Problems

Xiangshuang Liu
College of Computer Science, Chongqing University, China

Ziyu Chen[1]
College of Computer Science, Chongqing University, China

Dingding Chen
College of Computer Science, Chongqing University, China

Junsong Gao
College of Computer Science, Chongqing University, China

Abstract

Complete search algorithms are important methods for solving Distributed Constraint Optimization Problems (DCOPs), which generally utilize bounds to prune the search space. However, obtaining high-quality lower bounds is quite expensive since it requires each agent to collect more information aside from its local knowledge, which would cause tremendous traffic overheads. Instead of bothering for bounds, we propose a Bound-Independent Pruning (BIP) technique for existing tree-based complete search algorithms, which can independently reduce the search space only by exploiting local knowledge. Specifically, BIP enables each agent to determine a subspace containing the optimal solution only from its local constraints along with running contexts, which can be further exploited by any search strategies. Furthermore, we present an acceptability testing mechanism to tailor existing tree-based complete search algorithms to search the remaining space returned by BIP when they hold inconsistent contexts. Finally, we prove the correctness of our technique and the experimental results show that BIP can significantly speed up state-of-the-art tree-based complete search algorithms on various standard benchmarks.

2012 ACM Subject Classification Computing methodologies → Cooperation and coordination

Keywords and phrases DCOP, complete algorithms, search

Digital Object Identifier 10.4230/LIPIcs.CP.2021.41

Supplementary Material *Software (Source Code)*: `https://github.com/czy920/BIP`

Acknowledgements We would like to thank the reviewers for their valuable comments, and Dr. Ferdinando Fioretto and Dr. Yanchen Deng for their helpful suggestions.

1 Introduction

Distributed Constraint Optimization Problems (DCOPs) [14, 10] are a fundamental framework for coordinated and cooperative multi-agent systems. They have been widely deployed in many real applications such as sensor network [8], task scheduling [18, 26], smart grid [11] and many others.

[1] Corresponding author.

Incomplete algorithms for DCOPs [18, 28, 17, 9, 22, 21] aim to rapidly find a good solution at an acceptable overhead, while complete algorithms guarantee to find the optimal one by employing either inference or search to systematically explore the entire solution space. DPOP [23] and Action_GDL [27] are typical inference-based complete algorithms which perform dynamic-programming to solve a DCOP. However, they require a linear number of messages of exponential size. MB-DPOP and its variant [25, 5] proposed to trade the number of messages for smaller size of message. DPOP with function filtering [3] exploits utility bounds to reduce the size of message, where agents need to collect projected utilities to establish utility bounds.

On the other hand, search-based complete algorithms perform distributed backtrack searches and have a linear size of messages but an exponential number of messages. Tree-based complete search algorithms are the most popular ones among them and normally utilize bounds to prune the search space. Some work has been done to tailor centralized pruning techniques such as soft arc consistency [6] to tighten lower bounds in a distributed setting. BnB-ADOPT$^+$-AC/FDAC [12] proposed to get strong lower bounds via arc consistency (AC) and full directional arc consistency (FDAC) levels of soft arc consistency. However, stronger consistency levels require agents to know more information about other agents to plan sequences of soft arc consistency operations, which would compromise privacy and cause tremendous communication overheads. Besides, PT-FB [16] builds tight lower bounds via a forward bounding procedure which requires cost estimates from neighbors. ADOPT-BDP [1], DJAO [15] and PT-ISABB [7] came out to perform an approximation inference to acquire tighter lower bounds in the preprocessing phase. Recently, HS-CAI [4] was proposed to tighten lower bounds by executing the context-based inference iteratively. Like soft arc consistency, these methods also need to collect more information aside from local knowledge and thus lead to traffic overheads inevitably.

In a nutshell, the existing pruning techniques for DCOPs are bound-dependent and require collecting more information to obtain tight bounds. Different from them, we present a novel pruning technique independent of bounds and dispensing with information collection to accelerate existing tree-based complete search algorithms. More specifically, our contributions are listed as follows.

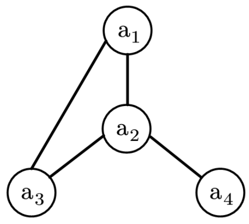
- We present a Bound-Independent Pruning (BIP) technique for existing tree-based complete search algorithms, which utilizes local constraints and running contexts to cut down the search space independently.
- We further introduce an acceptability testing mechanism (ATM) to filter out unacceptable search results produced by existing tree-based complete search algorithms when enforcing BIP under the inconsistent contexts.
- We theoretically show the correctness of BIP and ATM, and the experimental results demonstrate that BIP substantially improves state-of-the-art tree-based complete search algorithms on all the metrics in most cases.

2 Background

In this section, we introduce the preliminaries including DCOPs, pseudo tree and tree-based complete search algorithms.

2.1 Distributed Constraint Optimization Problems

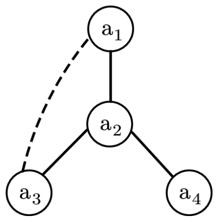
A distributed constraint optimization problem [19] can be formalized by a tuple $\langle A, X, D, F\rangle$ where



(a) Constraint graph.

x_i \ x_j	0	1	2
0	2	3	0
1	3	2	1
2	5	5	6

(b) Constraint matrix.



(c) Pseudo tree.

Figure 1 An example of a DCOP and its pseudo tree.

- $A = \{a_1, a_2, ..., a_n\}$ is a set of agents.
- $X = \{x_1, x_2, ..., x_m\}$ is a set of variables.
- $D = \{D_1, D_2, ..., D_m\}$ is a set of finite and discrete domains, where each variable x_i takes a value assignment in D_i. Here, we denote the maximal domain size as $d_{max} = \max_{a_i \in A} |D_i|$.
- $F = \{f_1, f_2, ..., f_q\}$ is a set of constraint functions, where each constraint $f_i : D_{i_1} \times \cdots D_{i_k} \rightarrow \mathbb{R}_{\geqslant 0}$ specifies the non-negative cost for each combination of $x_{i_1}, ..., x_{i_k}$.

For the sake of simplicity, we assume that each agent controls exactly one variable (i.e., $n = m$) and all constraint functions are binary (i.e., $f_{ij} : D_i \times D_j \rightarrow \mathbb{R}_{\geqslant 0}$). Thus, the term *agent* and *variable* can be used interchangeably. An optimal solution to a DCOP is an assignment to all the variables such that the total cost is minimized. That is,

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

A DCOP can be visualized by a constraint graph where the nodes denote agents and the edges denote constraints. Figure 1 (a) gives a DCOP with four variables and four constraints. For simplicity, the domain size of each variable is three and all the constraints are identical as shown in Fig. 1 (b) where $i < j$.

2.2 Pseudo Tree

A pseudo tree is a partial ordered arrangement to a constraint graph and can be generated by depth-first search traversal, where different branches are independent from each other and its constraints are categorized into tree edges and 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 its parent $P(a_i)$ and children $C(a_i)$, while the ones connected by pseudo edges are denoted as its pseudo parents $PP(a_i)$ and pseudo children $PC(a_i)$. For succinctness, we also denote all its (pseudo) parents as $AP(a_i) = PP(a_i) \cup \{P(a_i)\}$, all its (pseudo) children as $CD(a_i) = C(a_i) \cup PC(a_i)$, its ancestors as $Anc(a_i)$ and its descendants as $Desc(a_i)$. Besides, we denote the set of ancestors who share constraints with a_i and its descendants as $Sep(a_i)$ [24]. Figure 1 (c) gives a possible pseudo tree deriving from Fig. 1 (a) where tree edges and pseudo edges are denoted by solid and dashed lines, respectively.

2.3 Tree-based Complete Search Algorithms

Tree-based complete search algorithms perform a systematic search on a pseudo tree. Specifically, each agent a_i traverses the subtree rooted at itself under the running context $Context_i$ (i.e., the assignment to $Sep(a_i)$) and avoids expanding suboptimal branches by exploiting the bounds including the lower and upper bounds of its subproblem (i.e., LB_i and UB_i), the ones of its subproblem given its value $d_i \in D_i$ (i.e., $LB_i(d_i)$ and $UB_i(d_i)$) and the ones of its subproblem given its value $d_i \in D_i$ for its child $a_c \in C(a_i)$ (i.e., $lb_i^c(d_i)$ and $ub_i^c(d_i)$).

According to the way that agents update their assignments, tree-based complete search algorithms can be classified as synchronous or asynchronous. Synchronous algorithms constrain the agents' decisions to follow a particular order. As a result, only when a_i and its descendants have thoroughly explored its subproblem given $Context_i$, it reports its search results including LB_i and UB_i (here, $LB_i = UB_i = opt_i$ if $Context_i$ is feasible and $LB_i = UB_i = \infty$ otherwise, and opt_i is the optimal solution cost of the subproblem) if it is a non-root agent; otherwise, it finds the optimal solution. In contrast, asynchronous ones allow agents to update their assignments solely based on their local view of their subproblems and report their search results including LB_i and UB_i (here, $LB_i \leq opt_i \leq UB_i$) continually. Once the root agent a_i has $LB_i = UB_i$, the optimal solution is found.

Next, we will take PT-FB and BnB-ADOPT for example to describe the concrete implementation of synchronous and asynchronous tree-based complete search algorithms, respectively. In PT-FB, each agent a_i explores its subproblem by sequentially expanding $Context_i$ via sending CPA messages[2] to its children, and reports its search results via a UB message[3] once exhausting its domain. When receiving a UB message including LB_c and UB_c from a_c, a_i updates $lb_i^c(d_i)$ and $ub_i^c(d_i)$ with LB_c and UB_c, respectively. As for BnB-ADOPT, each agent a_i explores its subproblem by constantly informing its current value to its constrained descendants via VALUE messages[4] and reporting its search results including LB_i and UB_i to its parent via a COST message[2]. Similarly to PT-FB, once receiving a COST message including LB_c and UB_c from a_c, a_i updates $lb_i^c(d_i)$ and $ub_i^c(d_i)$ with LB_c and UB_c, respectively. Afterwards, it updates $LB_i(d_i)$ and $UB_i(d_i)$, and then updates LB_i and UB_i according to the following equations.

$$LB_i = \min_{d_i \in D_i} \{LB_i(d_i)\} \tag{1}$$

$$UB_i = \min_{d_i \in D_i} \{UB_i(d_i)\} \tag{2}$$

3 Proposed Method

In this section, we propose a Bound-Independent Pruning (BIP) technique for existing tree-based complete search algorithms, where each agent solely exploits its local constraints and running contexts to confirm a subspace containing the optimal solution and thereby obtains pruned domains for itself and its children under the current context. We further introduce an acceptability testing mechanism to tailor existing tree-based complete search algorithms to match BIP when they hold inconsistent contexts. Before elaborating on BIP, we first introduce some terms, definitions and their related properties used in this paper.

▶ **Definition 1** (dims). *Let U be a cost table, $dims(U)$ is the set of variables involved in U.*

$D^U = \times_{x_i \in dims(U)} D_i$ is the set of all value combinations of $dims(U)$. Given $x_i \in dims(U)$, $D_{-i}^U = \times_{x_j \in dims(U) \setminus \{x_i\}} D_j$ is the set of all value combinations of $dims(U)$ except x_i. Particularly, we specify $D_{-i}^U = \{\emptyset\}$ when $dims(U) \setminus \{x_i\} = \emptyset$.

Take f_{12} in Fig. 1(b) for example. We have $dims(f_{12}) = \{x_1, x_2\}$ and $D_{-2}^{f_{12}} = \{((x_1, 0)), ((x_1, 1)), ((x_1, 2))\}$.

[2] A CPA message contains the Current Partial Assignment of the sending agent and all its ancestors.
[3] A UB (COST) message contains the results of a solution found to the sending agent's sub-problem.
[4] A VALUE message contains the value assignment of the sending agent.

▶ **Definition 2** (POS). *Given $\boldsymbol{d}_{-i} \in D^U_{-i}$, $d_i \in D_i$ is a Primary Optimal Support (POS) for $\boldsymbol{d}_{-i}$ on U, denoted by $pos_U(\boldsymbol{d}_{-i})$, iff $U(d_i, \boldsymbol{d}_{-i}) = \min_{d'_i \in D_i} U(d'_i, \boldsymbol{d}_{-i})$ (ties are broken alphabetically).*

Consider f_{12}. When $\boldsymbol{d}_{-2} = ((x_1, 2))$, we have $pos_{f_{12}}(\boldsymbol{d}_{-2})$=0 since $f_{12}(\boldsymbol{d}_{-2}, x_2 = 0) = f_{12}(\boldsymbol{d}_{-2}, x_2 = 1) < f_{12}(\boldsymbol{d}_{-2}, x_2 = 2)$ and 0 precedes 1.

According to Definition 2, we can obtain a subset of D^U, denoted by S^U_i, by Eq. (3).

$$S^U_i = \left\{(d_i, \boldsymbol{d}_{-i}) \,\middle|\, d_i = pos_U(\boldsymbol{d}_{-i}), \forall \boldsymbol{d}_{-i} \in D^U_{-i}\right\} \tag{3}$$

Considering f_{12} again, we have $S^{f_{12}}_2 = \{((x_2, 2), (x_1, 0)), ((x_2, 2), (x_1, 1)), ((x_2, 0), (x_1, 2))\}$.

▶ **Definition 3** (Join). *Let U, U' be two cost tables, the join of U and U', denoted by $U \otimes U'$, is a relation defined over $D^{U \otimes U'} = \times_{x_i \in dims(U) \cup dims(U')} D_i$ such that*

$$(U \otimes U')(\boldsymbol{d}) = U\left(\boldsymbol{d}_{[dims(U)]}\right) + U'\left(\boldsymbol{d}_{[dims(U')]}\right), \forall \boldsymbol{d} \in D^{U \otimes U'}$$

where $\boldsymbol{d}_{[dims(U)]}$ and $\boldsymbol{d}_{[dims(U')]}$ are slices of $\boldsymbol{d}$ along $dims(U)$ and $dims(U')$, respectively.

Next, we give the following properties of S^U_i, based on which BIP is presented.

▶ **Property 4.** *There exists at least one element in S^U_i leading to the optimal cost in U. That is, $\exists \boldsymbol{d} \in S^U_i$, s.t. $U(\boldsymbol{d}) = min_{\boldsymbol{d'} \in D^U} U(\boldsymbol{d'})$.*

Proof. Assume that $\forall \boldsymbol{d} \in S^U_i$, $U(\boldsymbol{d}) > U(\boldsymbol{d}^*)$ where $\boldsymbol{d}^* = (d^*_i, \boldsymbol{d}^*_{-i})$ is the optimal solution.

According to Definition 2, we have $\exists d'_i \in D_i$, *s.t.* $d'_i = pos_U(\boldsymbol{d}^*_{-i})$ and thus $U(d'_i, \boldsymbol{d}^*_{-i}) = U(\boldsymbol{d}^*)$. Furthermore, we have $(d'_i, \boldsymbol{d}^*_{-i}) \in S^U_i$ by Eq. (3). That is, $(d'_i, \boldsymbol{d}^*_{-i}) \in S^U_i$ and $U(d'_i, \boldsymbol{d}^*_{-i}) = U(\boldsymbol{d}^*)$ which contradicts the assumption. Thus, Property 4 is proved. ◀

▶ **Property 5.** *Given two cost tables U and U', $S^U_i = S^{U'}_i$ if*

$$U = U' \otimes U'' \tag{4}$$

where $dims(U'') \subseteq dims(U') \backslash \{x_i\}$ and $x_i \in dims(U')$.

Proof. Firstly, we prove that $U(d_i, \boldsymbol{d}_{-i}) - U(d'_i, \boldsymbol{d}_{-i}) = U'(d_i, \boldsymbol{d}_{-i}) - U'(d'_i, \boldsymbol{d}_{-i}), \forall d_i, d'_i \in D_i, \boldsymbol{d}_{-i} \in D^U_{-i}$.

According to Eq. (4), we have $dims(U') = dims(U)$ and $x_i \in dims(U)$. Thus, for all $d_i, d'_i \in D_i$ and $\boldsymbol{d}_{-i} \in D^U_{-i}$, we have

$$\begin{aligned} &U(d_i, \boldsymbol{d}_{-i}) - U(d'_i, \boldsymbol{d}_{-i}) \\ &= (U'(d_i, \boldsymbol{d}_{-i}) + U''(\boldsymbol{d}_{-i[dims(U'')]})) - (U'(d'_i, \boldsymbol{d}_{-i}) + U''(\boldsymbol{d}_{-i[dims(U'')]})) \\ &= U'(d_i, \boldsymbol{d}_{-i}) - U'(d'_i, \boldsymbol{d}_{-i}) \end{aligned} \tag{5}$$

Next, we prove that given $\boldsymbol{d}_{-i} \in D^U_{-i}$ and $d_i \in D_i$, $U(d_i, \boldsymbol{d}_{-i}) = \min_{d'_i \in D_i} U(d'_i, \boldsymbol{d}_{-i})$ if $U'(d_i, \boldsymbol{d}_{-i}) = \min_{d'_i \in D_i} U'(d'_i, \boldsymbol{d}_{-i})$.

Assume that $\exists d'_i \in D_i$, *s.t.* $U(d_i, \boldsymbol{d}_{-i}) > U(d'_i, \boldsymbol{d}_{-i})$. Since $U'(d_i, \boldsymbol{d}_{-i}) = \min_{d'_i \in D_i} U'(d'_i, \boldsymbol{d}_{-i})$, we have $U'(d_i, \boldsymbol{d}_{-i}) \leqslant U'(d'_i, \boldsymbol{d}_{-i})$, $\forall d'_i \in D_i$. Further, we have $U(d_i, \boldsymbol{d}_{-i}) \leqslant U(d'_i, \boldsymbol{d}_{-i})$, $\forall d'_i \in D_i$ by Eq. (5), which is contradictory to the assumption. Thus, the conclusion holds.

Based on the above conclusion and Definition 2, we have

$$pos_U(\boldsymbol{d}_{-i}) = pos_{U'}(\boldsymbol{d}_{-i}) \tag{6}$$

Therefore, we can conclude $S^U_i = S^{U'}_i$ based on Eqs. (3) and (6), and thereby Property 5 is proved. ◀

According to Property 5, we can readily obtain S_i^U from U' if U and U' satisfy Eq. (4). Namely, we can derive the desired subspace of a table from its subtable which includes its partial information if they satisfy Eq. (4). Look into a DCOP. Given its pseudo tree and a running context, if U' and U are respectively instantiated to the combination cost table of local constraints at an agent a_i and the combination cost table of all constraints in the subproblem rooted at itself after eliminating $Desc(a_i)\backslash CD(a_i)$, we find that they still satisfy Eq. (4) (see Lemma 7 for detail). Accordingly, each agent a_i can confirm the subspace containing the optimal solution like S_i^U from its local constraints under the running context. Hereby, we propose BIP for tree-based complete search algorithms.

3.1 Bound-Independent Pruning(BIP) Technique

BIP aims to enable each agent a_i to exclude some elements in $D^{U_i}\backslash S_i^{U_i}$ to prune the search space as possible under the current context in light of Property 4 and 5. Here, U_i is the combination of all local constraints at a_i under $Context_i$. Formally,

$$U_i = cd\_cost_i \otimes ap\_cost_i \tag{7}$$

$$cd\_cost_i = \otimes_{a_j \in CD(a_i)} f_{ij} \tag{8}$$

$$ap\_cost_i = \sum\nolimits_{a_j \in AP(a_i)} f_{ij}\left(Context_i(x_j)\right) \tag{9}$$

where cd_cost_i is the combination of all constraints between a_i and $CD(a_i)$ and ap_cost_i is the sum of all constraints between a_i and $AP(a_i)$ under the current context.

Theoretically, all the elements in $D^{U_i}\backslash S_i^{U_i}$ should be pruned. However, it is hard to do since pruning some elements requires the joint implementation by $CD(a_i)$ at different branches which search their subspace independently in existing tree-based complete search algorithms. Therefore, we choose to prune some elements from $D^{U_i}\backslash S_i^{U_i}$, which only involves a_i or the joint implementation by a_i and its child. Specifically, a_i removes DV_i computed by Eq. (10) from its domain (i.e., D_i). For each value $d_i \in D_i\backslash DV_i$, a_i suggests its child $a_c \in C(a_i)$ to remove $DV_i^c(d_i)$ computed by Eq. (11) from the domain of a_c (i.e., D_c).

$$DV_i = \{d_i \in D_i | (d_i, \boldsymbol{d}_{-i}) \in D^{U_i}\backslash S_i^{U_i}, \forall \boldsymbol{d}_{-i} \in D_{-i}^{U_i}\} \tag{10}$$

$$DV_i^c(d_i) = \{d_c \in D_c | (d_i, d_c, \boldsymbol{d}_{-(i,c)}) \in D^{U_i}\backslash S_i^{U_i}, \forall \boldsymbol{d}_{-(i,c)} \in D_{-(i,c)}^{U_i}\}, a_c \in C(a_i) \tag{11}$$

Here, $D_{-(i,c)}^{U_i} = \times_{x_j \in dims(U_i)\backslash\{x_i,x_c\}} D_j$.

Take a_2 in Fig. 1(c) for example. Given $Context_2 = \{(x_1, 0)\}$, we have $U_2 = f_{12}(x_1 = 0) \otimes f_{23} \otimes f_{24}$ as shown in Fig. 2 where all the elements in $S_2^{U_2}$ are highlighted in bold. We have $DV_2 = \{2\}$ according to Eq. (10), and $DV_2^3(1) = \{0, 2\}$ according to Eq. (11). Similarly, we have $DV_2^3(0) = \emptyset$, $DV_2^4(0) = \emptyset$ and $DV_2^4(1) = \{0, 2\}$. Accordingly, a_2 obtains all the removed elements shown in gray.

Since $DV_{P(a_i)}^i(Context_i(P(a_i)))$ can be piggybacked by a CPA or VALUE message from $P(a_i)$ and DV_i is computed by itself, a_i can obtain the pruned domain Dom_i by:

$$Dom_i = D_i\backslash(DV_i \cup DV_{P(a_i)}^i(Context_i(P(a_i)))) \tag{12}$$

Algorithm 1 gives the sketch of calculating Dom_i for both synchronous and asynchronous tree-based complete search algorithms when enforcing BIP. Each agent a_i computes cd_cost_i firstly (line 1). Afterwards, it calculates DV_i and $DV_i^c(d_i)$ by calling $Compute\_DVs()$ and

x_2	x_3	x_4	U
0	**0**	**0**	6
0	**0**	**1**	7
0	**0**	**2**	4
0	**1**	**0**	7
0	1	1	8
0	**1**	**2**	5
0	**2**	**0**	4
0	**2**	**1**	5
0	**2**	**2**	2

(a) $x_2 = 0$.

x_2	x_3	x_4	U
1	0	0	9
1	0	1	8
1	0	2	7
1	1	0	8
1	**1**	**1**	7
1	1	2	6
1	2	0	7
1	2	1	6
1	2	2	5

(b) $x_2 = 1$.

x_2	x_3	x_4	U
2	0	0	10
2	0	1	10
2	0	2	11
2	1	0	10
2	1	1	10
2	1	2	11
2	2	0	11
2	2	1	11
2	2	2	12

(c) $x_2 = 2$.

Figure 2 U_2 under $Context_2 = \{(x_1, 0)\}$.

Algorithm 1 Calculating Dom_i for a_i.

```
   When Initialization:
1      compute $cd\_cost_i$ according to Eq. (7)
2      if $a_i$ is the root then
3          Compute_DVs()
4          compute $Dom_i$ according to Eq. (12)
   When received a CPA or VALUE from $P(a_i)$:
5      Compute_DVs()
6      compute $Dom_i$ according to Eq. (12)
   When sending a CPA or VALUE to $a_c \in C(a_i)$:
7      attach $DV_i^c(d_i)$ to the CPA or VALUE message
   Function Compute_DVs():
8      $DV_i \leftarrow D_i$
9      $DV_i^c(d_i) \leftarrow D_c$ ,$\forall\ d_i \in D_i, a_c \in C(a_i)$
10     foreach $\boldsymbol{d}_{-i} \in D_{-i}^{U_i}$ do
11         $d_i = pos_{U_i}(\boldsymbol{d}_{-i})$
12         $DV_i \leftarrow DV_i \backslash \{d_i\}$
13         $DV_i^c(d_i) \leftarrow DV_i^c(d_i) \backslash \boldsymbol{d}_{-i[x_c]}, \forall a_c \in C(a_i)$
```

Dom_i according to Eq. (12) if it is the root (lines 2–4, 8–13) or receiving a CPA or VALUE message from $P(a_i)$ (line 5–6), and attaches $DV_i^c(d_i)$ to the message when forwarding a CPA or VALUE message to a_c (line 7). $Compute\_DVs()$ performs the following steps to obtain DV_i and $DV_i^c(d_i)$. Firstly, each agent a_i initializes DV_i to D_i and $DV_i^c(d_i)$ to D_c for each $d_i \in D_i$ and $a_c \in D_c$ (lines 8-9). Then, a_i traverses U_i to filter out elements from DV_i and $DV_i^c(d_i)$ if they do not satisfy Eq. (10) and (11) (lines 10–13), respectively.

3.2 An Example for BIP

We take Fig. 1 as an example to trace BIP runing on a tree-based synchronous search algorithm. Table 1 shows the variable update for BIP in the first three cycles. For the sake of simplicity, we omit the variable update for the search algorithm.

Cycle 1: After constructing a pseudo tree shown in Fig. 1(c), the root agent a_1 computes $U_1 = f_{12} \otimes f_{13}$, and then calls $Compute\_DVs()$ to get DV_1 and $DV_1^2(d_1)$ for itself and its child a_2, respectively. Afterwards, it computes $Dom_1 = D_1 \backslash DV_1 = \{0, 1\}$ and sends $\{(x_1, 0)\}$ and $DV_1^2(0) = \emptyset$ to its child a_2 via a CPA message. (Assume that a_1 takes its feasible assignment$(x_1, 0)$.)

$$a_1 \rightarrow a_2 : \text{CPA}(\{(x_1, 0)\}, DV_1^2(0))$$

Table 1 The trace of assignments to the variables of BIP.

$Cycle(a_i)$	$Contex_i$	DV_i	$DV_i^c(d_i)$	Dom_i	x_i
$1(a_1)$	$\emptyset$	$\{2\}$	$DV_1^2(0)=\emptyset$, $DV_1^2(1)=\{0,2\}$, $DV_1^2(2)=\{0,1,2\}$	$\{0,1\}$	0
$2(a_2)$	$\{(x_1,0)\}$	$\{2\}$	$DV_2^3(0)=DV_2^4(0)=\emptyset$, $DV_2^3(1)=DV_2^4(1)=\{0,2\}$, $DV_2^3(2)=DV_2^4(2)=\{0,1,2\}$	$\{0,1\}$	1
$3(a_3)$	$\{(x_1,0),$ $(x_2,1)\}$	$\{0,1\}$	——	$\emptyset$	——
$3(a_4)$	$\{(x_1,0),$ $(x_2,1)\}$	$\{0,1\}$	——	$\emptyset$	——

Cycle 2: When a_2 receives the CPA message from a_1, it computes $U_2 = f_{12}(x_1 = 0) \otimes f_{23} \otimes f_{24}$, calls $Compute\_DVs()$ to get DV_2, $DV_2^3(d_2)$ and $DV_2^4(d_2)$ for itself, a_3 and a_4, respectively, and then computes $Dom_2 = D_2 \backslash (DV_2 \cup DV_1^2(0)) = \{0,1\}$. Next, a_2 takes its feasible assignment$(x_2, 1)$ and sends a CPA message containing $\{(x_1,0),(x_2,1)\}$ and $DV_2^3(1)$ to a_3 and a CPA message containing $\{(x_1,0),(x_2,1)\}$ and $DV_2^4(1)$ to a_4.

$$a_2 \rightarrow a_3 : CPA(\{(x_1,0),(x_2,1)\}, DV_2^3(1))$$
$$a_2 \rightarrow a_4 : CPA(\{(x_1,0),(x_2,1)\}, DV_2^4(1))$$

Cycle 3: Upon receipt of the CPA message from a_2, a_3 computes DV_3 by calling $Compute\_DVs()$ [5], and gets $Dom_3 = D_3 \backslash (DV_3 \cup DV_2^3(1)) = \emptyset$ which means $Context_3$ can not lead to the optimal solution and should be changed. Therefore, it backtracks to its parent a_2 with an infinity cost. a_4 performs the same as a_3.

3.3 Acceptability Testing Mechanism(ATM)

For existing tree-based complete search algorithms, each agent a_i explores its subproblem conditioned on $Context_i$. When enforcing BIP, a_i needs to exploit Dom_i which is computed under the assignments to $AP(a_i)$ and $AP(P(a_i))$ according to Eqs. (10) - (12). However, the assignment to $AP(P(a_i)) \backslash Sep(a_i)$ is not contained in $Context_i$. Consequently, a_i is searching under its running context while $AP(P(a_i)) \backslash Sep(a_i)$ may change their values, which is very common in asynchronous algorithms. In the case, a_i's search results might be unacceptable. There exist two naïve solutions to the issue. The one is to expand the running context of a_i from the assignment of $Sep(a_i)$ to the ones of $Sep(a_i) \cup AP(P(a_i))$. The other is to remove $DV^i_{P(a_i)}(Context_i(P(a_i)))$ from Eq. (12). Unfortunately, the former could result in more frequent changes of running contexts and thus severely degrade the original algorithms while the latter could lead to missing opportunities to prune the search space.

Instead of changing the running context or BIP, we propose ATM to filter out unacceptable search results to ensure the completeness of the original algorithms when the inconsistent contexts happen. Specifically, for a_i and its child a_c, when $AP(a_i) \backslash Sep(a_c)$ change their values, a_c has to exploit new Dom_c, which puts a_c's search results under its old Dom_c at risk of unacceptability. For clarity, we denote the old Dom_c and the new Dom_c as $Dom_c(d_i)$ and $Dom'_c(d_i)$ for a given d_i, respectively. We introduce the following rules to determine if a_c's search results under $Dom_c(d_i)$ (i.e., $lb_i^c(d_i)$ and $ub_i^c(d_i)$) are acceptable or not.

[5] Note that $Compute\_DVs()$ is also applied to leaves since we specify $D^U_{-i} = \{\emptyset\}$ when $dims(U) \backslash \{x_i\} = \emptyset$.

- **Rule 1:** $lb_i^c(d_i)$ is acceptable under $Dom'_c(d_i)$ if $Dom'_c(d_i) \subseteq Dom_c(d_i)$; otherwise, discarded.
- **Rule 2:** $ub_i^c(d_i)$ is acceptable under $Dom'_c(d_i)$ if $Dom'_c(d_i) \supseteq Dom_c(d_i)$; otherwise, discarded.

Next, we will prove the correction of the two rules.

▶ **Proposition 6.** *Given d_i, $lb_i^c(d_i)$ produced under $Dom_c(d_i)$ is acceptable under $Dom'_i(d_i)$ if $Dom'_c(d_i) \subseteq Dom_c(d_i)$, and $ub_i^c(d_i)$ produced under $Dom_c(d_i)$ is acceptable under $Dom'_i(d_i)$ if $Dom'_c(d_i) \supseteq Dom_c(d_i)$.*

Proof. Let LB'_c and UB'_c be the search results of a_c under $Dom'_c(d_i)$, and opt'_c be the optimal cost of a_c's subproblem under $Dom'_c(d_i)$. To prove the proposition, we only need to prove that $lb_i^c(d_i) \leq LB'_c$ if $Dom'_c(d_i) \subseteq Dom_c(d_i)$ and $ub_i^c(d_i) \geq UB'_c$ if $Dom'_c(d_i) \supseteq Dom_c(d_i)$ since $LB'_c \leq opt'_c \leq UB'_c$. Next, we will firstly prove $lb_i^c(d_i) \leq LB'_c$ if $Dom'_c(d_i) \subseteq Dom_c(d_i)$.

As LB'_c is the search result under $Dom'_c(d_i)$, according to Eq. (1), we have

$$LB'_c = \min_{d_c \in Dom'_c(d_i)} \{LB_c(d_c)\}$$

Since $lb_i^c(d_i) = LB_c$ (LB_c is actually obtained under $Dom_c(d_i)$) and $Dom'_c(d_i) \subseteq Dom_c(d_i)$, we have

$$\begin{aligned} lb_i^c(d_i) &= \min_{d_c \in Dom_c(d_i)} \{LB_c(d_c)\} \\ &= \min_{d_c \in (Dom_c(d_i) \setminus Dom'_c(d_i)) \cup Dom'_c(d_i)} \{LB_c(d_c)\} \\ &= \min(\min_{d_c \in Dom'_c(d_i)} LB_c(d_c), \min_{d_c \in Dom_c(d_i) \setminus Dom'_c(d_i)} LB_c(d_c)) \\ &= \min(LB'_c, \min_{d_c \in Dom_c(d_i) \setminus Dom'_c(d_i)} LB_c(d_c)) \leqslant LB'_c \end{aligned}$$

Similarly, we can conclude $ub_i^c(d_i) \geq UB'_c$ if $Dom_c(d_i) \subseteq Dom'_c(d_i)$. Thus, Proposition 6 is proved. ◀

To execute Rule 1 and 2, a_i needs to obtain $Dom_c(d_i)$ and $Dom'_c(d_i)$. Here, $Dom_c(d_i)$ can be piggybacked by a COST message from a_c and $Dom'_c(d_i)$ can be obtained by:

$$Dom'_c(d_i) = D_c \setminus (DV_c \cup DV_i^c(d_i))$$

where $DV_i^c(d_i)$ is computed by a_i based on Eq. (11) and DV_c can also be piggybacked by a COST message from a_c. When the search results are unacceptable, a_i attaches a Boolean variable $ReqCost_i^c(d_i)$ to the VALUE message to request a COST message from a_c.

$$\begin{gathered} \text{VALUE}(a_i, d_i, ID, TH, \boldsymbol{DV_i^c(d_i), ReqCost_i^c(d_i)}) \\ \text{COST}(a_i, context_i, LB_i, UB_i, ThReq, \boldsymbol{Dom_i, DV_i}) \end{gathered}$$

Figure 3 Messages of BnB-ADOPT$^+$ when enforcing BIP.

Algorithm 2 presents the sketch of acceptability testing mechanism for BnB-ADOPT$^+$ [13] (i.e., a version of BnB-ADOPT which removes most of the redundant messages) when enforcing BIP. Here, we attach Dom_i and DV_i to a COST message, and $DV_i^c(d_i)$ and $ReqCost_i^c(d_i)$ to a VALUE message. Figure 3 shows the modified messages where the attached items are bold. Accordingly, we make the following adjustment in processing VALUE and COST messages. Upon receipt of a COST message from its child or a VALUE message from its parent, a_i needs to check if the search results LB_c and UB_c are acceptable by Rule 1 and 2 (lines 14–24,

Algorithm 2 Acceptability testing mechanism for a_i.

```
   When received a COST from $a_c \in C(a_i)$:
14   if $Context_i$ is compatible with $Context_c$ then
15     $d_i = Context_c(a_i)$
16     if meet Rule 1 for $a_c$ then
17       $lb_i^c(d_i) \leftarrow max\{lb_i^c(d_i), LB_c\}$
18     if meet Rule 2 for $a_c$ then
19       if $Dom_c = \emptyset$ then
20         $ub_i^c(d_i) \leftarrow \infty$
21       else
22         $ub_i^c(d_i) \leftarrow min\{ub_i^c(d_i), UB_c\}$
23     if not meet Rule 1 or Rule 2 for $a_c$ then
24       $ReqCost_i^c(d_i) \leftarrow true$
   When received a VALUE from $P(a_i)$:
25   if $Context_i$ is compatible with $Context_c$ then
26     foreach $a_c \in C(a_i), d_i \in D_i$ do
27       if not meet Rule 1 for $a_c$ then
28         $lb_i^c(d_i) \leftarrow 0$,
29       if not meet Rule 2 for $a_c$ then
30         $ub_i^c(d_i) \leftarrow \infty$,
31       if not meet Rule 1 or Rule 2 for $a_c$ then
32         $ReqCost_i^c(d_i) \leftarrow true$
   When sending a VALUE to $a_c \in C(a_i)$:
33   if $ReqCost_i^c(d_i) = true$ then
34     attach $ReqCost_i^c(d_i)$ to the VALUE to request a COST from $a_c$
35     $ReqCost_i^c(d_i) \leftarrow false$
   When sending a COST to P($a_i$):
36   if $Dom_i = \emptyset$ then
37     $LB_i \leftarrow \infty$
38     $UB_i \leftarrow \infty$
```

25–32). If LB_c and UB_c are unacceptable, a_i sets $ReqCost_i^c(d_i)$ to true to request the latest search results of a_c by a VALUE message to a_c (lines 33–35). Besides, a_i sets its lower and upper bounds to infinity and sends them to its parent by a COST message if $Dom_i = \emptyset$ (lines 36-38).

3.4 Tradeoff

When deploying BIP into existing tree-based complete search algorithms, each agent a_i needs to store its cost table cd_cost_i which requires the memory consumption of $d_{max}^{|CD(a_i)|+1}$. Thus, we introduce a parameter k to specify the maximum memory budget (i.e., d_{max}^k) for each agent and only the agents with $|CD(a_i)| + 1 < k$ can perform BIP. Hereby, we trade pruning efficiency for memory consumption. In addition, we allocate the remaining memory (i.e., $d_{max}^{k-|CD(a_i)|-1}$) to store DV_i and $DV_i^c(d_i)$ to avoid repeated computation, where least recently used (LRU) policy is used to replace the old entry with the lasted one.

3.5 Complexity

When applied to existing tree-based complete search algorithms, BIP does not introduce any new messages, and only adds some extra attachments which only require linear memory to forwarding messages. Specifically, we attach $DV_i^c(d_i)$ to a CPA message for synchronous tree-based algorithms, $DV_i^c(d_i)$ and $ReqCost_i^c(d_i)$ to a VALUE message and Dom_i and DV_i to a COST message for asynchronous tree-based algorithms.

As for the memory consumption of each agent a_i, it is $O(|D_i|)$ if $|CD(a_i)| + 1 > k$ since a_i does not need to perform BIP and thus only stores Dom_i. Otherwise, it is $O(d_{max}^k)$ for synchronous tree-based algorithms and $O(d_{max}^k + |C(a_i)|d_{max}^2)$ for asynchronous tree-based algorithms. Here, d_{max}^k is the memory consumption as mentioned in Subsection 3.4 and the memory of $|C(a_i)|d_{max}^2$ is used for storing $Dom_c'(d_i)$ and $DV_c(d_i)$ when performing ATM.

For each agent a_i, it needs to traverse U_i whose size is $O(d_{max}^{|CD(a_i)|+1})$ to compute $S_i^{U_i}$. Then, it can obtain DV_i and $DV_i^c(d_i)$ for each $a_c \in C(a_i)$ by enumerating each element in $S_i^{U_i}$ whose size is $d_{max}^{|CD(a_i)|}$. Thus, the overall computational complexity of a_i is $O(d_{max}^{|CD(a_i)|+1} + (1 + |C(a_i)|)d_{max}^{|CD(a_i)|})$

4 Theoretical Results

In the section, we prove the correction of BIP. Firstly, we will prove Eq. (4) can be suitable for DCOPs.

Given a DCOP and its pseudo tree, let us consider the following three cost tables regarding the subproblem rooted at a_i under the current context $Context_i$. U_i is the combination of all local constraints with a_i and computed by Eq. (7). U_i^{irr} is the combination cost table of all constraints without a_i in the subproblem. That is,

$$\begin{aligned} U_i^{irr} &= \otimes_{a_j \in Desc(a_i)} (\otimes_{a_k \in AP(a_j) \backslash (Sep(a_i) \cup \{a_i\})} f_{jk} \\ &\quad \otimes (\otimes_{a_k \in AP(a_j) \cap Sep(a_i)} f_{jk}(Context_i(x_k)))) \end{aligned} \tag{13}$$

U_i^{sub} is the combination cost table of all constraints in a_i's subproblem after eliminating $Desc(a_i) \backslash CD(a_i)$. That is,

$$U_i^{sub} = \min_{Desc(a_i) \backslash CD(a_i)} (U_i \otimes U_i^{irr}) \tag{14}$$

▶ **Lemma 7.** *$U'' = \min_{Desc(a_i) \backslash CD(a_i)} U_i^{irr}$ is a cost table such that $U_i^{sub} = U_i \otimes U''$ and $dims(U'') \subseteq dims(U_i) \backslash \{x_i\}$.*

Proof. According to Eq. (14), we have

$$\begin{aligned} U_i^{sub} &= \min_{Desc(a_i) \backslash CD(a_i)} (U_i \otimes U_i^{irr}) \\ &= U_i \otimes \min_{Desc(a_i) \backslash CD(a_i)} U_i^{irr} \end{aligned}$$

The equation from the first step to the second step holds since $dims(U_i) = CD(x_i) \cup \{x_i\}$ according to Eq. (7) and $(CD(x_i) \cup \{x_i\}) \cap (Desc(a_i) \backslash CD(a_i)) = \emptyset$. Further, according to Eq. (13), we have $dims(U_i^{irr}) = Desc(a_i)$ and thus $dims(U'') \subseteq dims(U_i) \backslash \{x_i\}$. Therefore, Lemma 7 is proved. ◀

▶ **Lemma 8.** *There exists at least one element in $S_i^{U_i}$ that can be extended to the optimal solution of a_i's subproblem.*

Proof. According to Property 4, we have $\exists \boldsymbol{d} \in S_i^{U_i^{sub}}$, *s.t.* $U_i^{sub}(\boldsymbol{d}) = \min_{\boldsymbol{d'} \in D^{U_i^{sub}}} U_i^{sub}(\boldsymbol{d'})$. Further, according to Lemma 7 and Property 5, we have $S_i^{U_i} = S_i^{U_i^{sub}}$ and $\exists \boldsymbol{d} \in S_i^{U_i}$, *s.t.* $U_i^{sub}(\boldsymbol{d}) = \min_{\boldsymbol{d'} \in D^{U_i^{sub}}} U_i^{sub}(\boldsymbol{d'})$. Therefore, we can conclude that the optimal solution of the subproblem rooted at a_i is the join of $\boldsymbol{d}$ and the optimal assignment to $Desc(a_i) \backslash CD(a_i)$ from Eq. (14). Thus, Lemma 8 is proved. ◀

▶ **Theorem 9.** *There exists an optimal solution in the remaining space returned by BIP.*

Proof. For any given context $Context_i$, there exists one element in $S_i^{U_i}$ that can be extended to the optimal solution of the subproblem rooted at a_i according to Lemma 8, and BIP does not prune any elements in $S_i^{U_i}$ according to Eqs. (10) and (11). Thus, Theorem 9 is proved. ◀

5 Empirical Evaluation

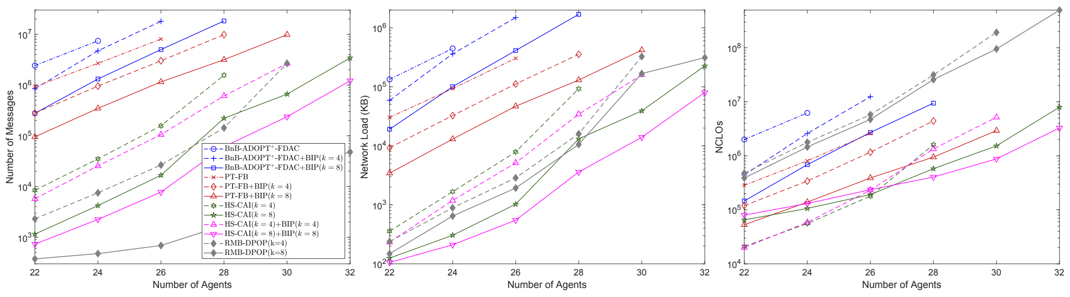
5.1 Experimental Configuration

In order to demonstrate its effect on distributed search, BIP is applied to BnB-ADOPT$^+$-FDAC, PT-FB and HS-CAI, named BnB-ADOPT$^+$-FDAC+BIP, PT-FB+BIP and HS-CAI+BIP, respectively. In our experiments, we will compare these BIP-based algorithms with their originals and RMB-DPOP [5] on two types of problems, i.e., random DCOPs and scale-free networks. RMB-DPOP is the latest best-performing algorithm in the DPOP family. We consider four configurations, and the first two are sparse and dense configurations for ramdom DCOPs. In more detail, we set the graph density to 0.2, the domain size to 3 and the number of agents varying from 22 to 32 for the sparse configuration, and the graph density to 0.5, the domain size to 3 and the number of agents varying from 14 to 24 for the dense configuration. The third configuration is the random DCOPs with 22 agents, the graph density of 0.2 and the domain size varying from 3 to 8. In the fourth configuration, we consider the scale-free networks whose degree distribution follows a power law. We generate the instances by BA model [2], where we set the number of agents to 26, the domain size to 3 and m_0 to 10, and vary m_1 from 2 to 8.

In our experiments, we use the number of messages (Msgs) and network load (NL, i.e., the size of total information exchanged) to measure the traffic overheads, and the NCLOs [20] to measure the hardware-independent runtime where the logical operations in the inference and the search are accesses to utilities and constraint checks, respectively. In order to capture the computation overhead introduced by BIP, the accesses to U_i, DV_i and $DV_i^c(d_i)$ are also counted into the NCLOs for the BIP-based algorithms. For each experiment, we generate 50 instances randomly with the integer constraint costs in the range of 0 to 100, and report the average over all instances. Moreover, we choose $k = 4$ and $k = 8$ as the low and high memory budget for HS-CAI, RMB-DPOP and BIP, respectively. For fairness, we set the memory for BIP to be the same as the one for HS-CAI (i.e., $O(|C(a_i)|d_{max}^k)$). The experiments are conducted on an i7-7820x workstation with 32GB of memory and we set the timeout to 30 minutes for each algorithm.

5.2 Experimental Results

Figure 4 presents the experimental results under different numbers of agents on the sparse configuration, and the corresponding improvement over the originals is displayed in the first two rows of Table 2 where the numbers greater than zero are shown in bold. It can be seen



(a) Number of Messages. **(b)** Network Load. **(c)** NCLOs.

Figure 4 Performance comparison under different numbers of agents on the sparse configuration.

Table 2 The improvement of the BIP-based algorithms over their respective originals.

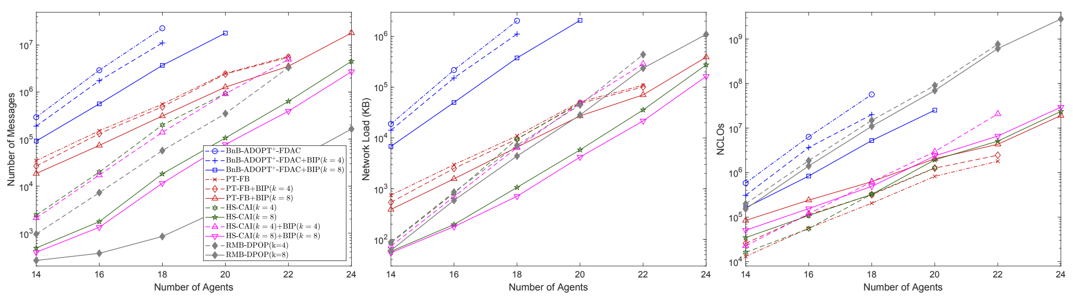
Configuration	k	BnB − ADOPT$^+$ − FDAC + BIP			PT − FB + BIP			HS − CAI + BIP		
		Msgs (%)	NL (%)	NCLOs (%)	Msgs (%)	NL (%)	NCLOs (%)	Msgs (%)	NL (%)	NCLOs (%)[6,7]
Sparse	4	**36 ∼ 65**	**29 ∼ 55**	**58 ∼ 77**	**62 ∼ 69**	**63 ∼ 69**	**54 ∼ 58**	**28 ∼ 61**	**30 ∼ 63**	−29 ∼ **17**
	8	**82 ∼ 88**	**77 ∼ 85**	**89 ∼ 92**	**85 ∼ 90**	**84 ∼ 88**	**81 ∼ 85**	**37 ∼ 73**	**16 ∼ 73**	−23 ∼ **59**
Dense	4	**35 ∼ 51**	**25 ∼ 45**	**43 ∼ 65**	**3 ∼ 23**	**7 ∼ 23**	−101 ∼ −36	**2 ∼ 30**	**3 ∼ 32**	−131 ∼ −40
	8	**69 ∼ 84**	**64 ∼ 81**	**72 ∼ 91**	**38 ∼ 51**	**35 ∼ 47**	−542 ∼ −140	**38 ∼ 40**	**7 ∼ 41**	−51 ∼ −22
Varying Domain Size	4	**57 ∼ 65**	**46 ∼ 55**	**72 ∼ 80**	**49 ∼ 70**	**50 ∼ 70**	**30 ∼ 66**	**19 ∼ 32**	**17 ∼ 35**	−86 ∼ **5**
	8	**85 ∼ 89**	**82 ∼ 86**	**90 ∼ 93**	**73 ∼ 90**	**89 ∼ 72**	**47 ∼ 81**	**19 ∼ 27**	**0 ∼ 16**	−25 ∼ −11
Scale − Free Networks	4	**52 ∼ 61**	**21 ∼ 43**	**56 ∼ 72**	**35 ∼ 87**	**37 ∼ 86**	−2 ∼ **86**	**28 ∼ 35**	**32 ∼ 36**	−39 ∼ −19
	8	**68 ∼ 84**	**62 ∼ 67**	**53 ∼ 90**	**71 ∼ 94**	**71 ∼ 93**	**57 ∼ 87**	**30 ∼ 67**	**28 ∼ 59**	**12 ∼ 28**

that the BIP-based algorithms exhibit a great advantage on all the metrics in most cases and the advantage expands as k increases. This is because more agents performing BIP can lead to better pruning efficiency under larger k. Moreover, the BIP-based algorithms can scale up to larger problems when $k = 4$ and their scalability is further enhanced when $k = 8$. In more detail, BnB-ADOPT$^+$-FDAC can only solve problems with the number of agents no greater than 24, and ADOPT$^+$-FDAC+BIP can scale up to 26 and 28 when $k = 4$ and $k = 8$, respectively. The similar phenomenon can be found from PT-FB+BIP and HS-CAI($k = 4$)+BIP($k = 4$). In addition, RMB-DPOP has a great advantage over the search algorithms on the number of messages, but performs worse than all the search algorithms except BnB-ADOPT$^+$-FDAC and BnB-ADOPT$^+$-FDAC+BIP($k = 4$) in terms of the NCLOs. Besides, when $k = 8$, HS-CAI is superior to RMB-DPOP in terms of the network load in most cases and HS-CAI+BIP greatly expands the superiority of HS-CAI over RMB-DPOP.

Figure 5 presents the experimental results under different numbers of agents on the dense configuration, and the third and fourth rows of Table 2 show the corresponding improvement over the originals. It can be seen that the BIP-based algorithms also perform better than their originals in terms of both the number of messages and network load. However, the

[6] In the sparse configuration, HS-CAI+BIP is superior to HS-CAI on the NCLOs and the superiority expands as the number of agents increases when the number of agents is greater than 26.

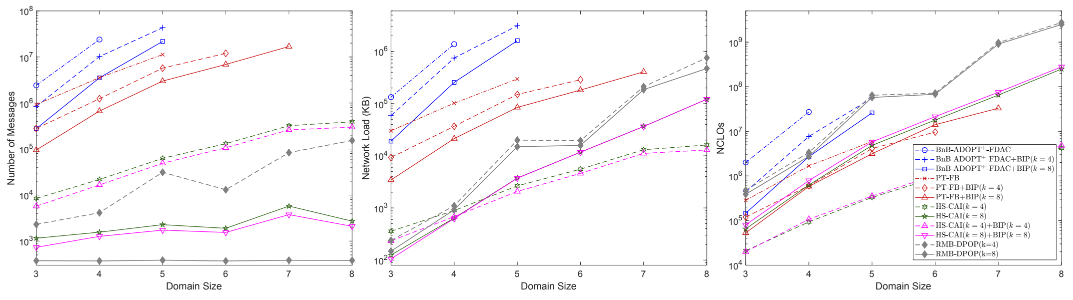
[7] In the configuration of varying domain size, we set the number of agents to 22 and HS-CAI+BIP is inferior to HS-CAI on the NCLOs, while HS-CAI+BIP will be superior to HS-CAI on this metric when the number of agents is greater than 26, which can be seen from Figure 4(c).



(a) Number of Messages. **(b)** Network Load. **(c)** NCLOs.

Figure 5 Performance comparison under different numbers of agents on the dense configuration.

gaps narrow compared to the ones on the sparse configuration. This is because less agents at the high positions of the pseudo-tree perform BIP on the dense configuration, which impairs pruning efficiency. In addition, BIP does not always perform well in terms of the NCLOs on the dense configuration. That is because the computational consumption of BIP is exponential to the number of (pseudo) children of an agent and there are more agents with a large number of (pseudo) children in the dense configuration. Compared to the search algorithms, the performance of RMB-DPOP is similar to the one on the sparse configuration.

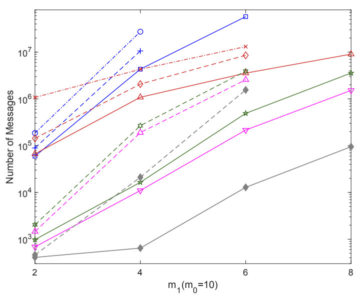


(a) Number of Messages. **(b)** Network Load. **(c)** NCLOs.

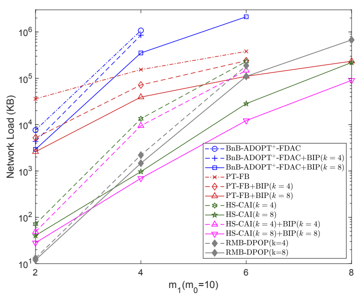
Figure 6 Performance comparison under different domain sizes.

Figure 6 presents the experimental results under different domain sizes, and the corresponding improvement over the originals can be found in the fifth and sixth rows of Table 2. We can see that BIP still works well when facing the problems with larger domain size. However, the improvement gaps narrow as the domain size increases. This is because the proportion of values pruned out by BIP reduces at large domain size. In addition, the BIP-based algorithms can solve the problems with larger domain size than their originals. In more detail, PT-FB can not solve the problems with the domain size greater than 5, while PT-FB+BIP can scale up to the ones with the domain size of 6 when $k = 4$, and further to the ones with the domain size of 7 when $k = 8$. When facing the problems with larger domain size, the performance of RMB-DPOP is similar to the one in the first two configurations. It is worth noting that the number of messages of RMB-DPOP($k = 8$) holds steady as the domain size increases. That is because under this configuration, it performs just like DPOP where the number of messages is linear to the number of agents.

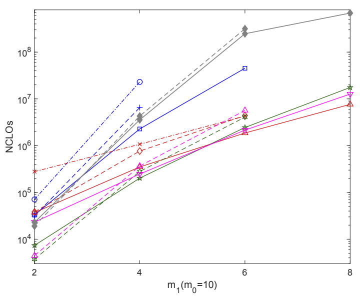
Figure 7 presents the experimental results on scale-free networks and the seventh and eighth rows of Table 2 show the corresponding improvement over the originals. It can be seen that the BIP-based algorithms exhibit a great advantage over their originals on all the metrics in most cases and the advantage expands as k increases, which is similar to



(a) Number of Messages.



(b) Network Load.



(c) NCLOs.

Figure 7 Performance comparison on scale-free networks.

the results on random DCOPs. In addition, when $k = 8$, both BnB-ADOPT$^+$-FDAC+BIP and PT-FB+BIP can scale up to the problems with larger m_1 than their originals. The performance of RMB-DPOP is similar to the one on random DCOPs.

From Table 2, we can see that BIP can improve the tree-based complete search algorithms in terms of both the number of messages and network load in all the experimental configurations. This is because BIP can significantly reduce the search space without introducing any new messages, and only adds some extra attachments which only require linear memory to forwarding messages. Thus, BIP is well suited for some real-world applications that are equipped with devices with the limited memory and desire for lower communication overheads. In addition, BIP can greatly improve the search-based algorithms under the sparse configuration on all the metrics. Thus, BIP is also well suited for solving the real-world applications with low graph density.

6 Conclusion

Complete search algorithms for DCOPs depend solely on bounds to prune the search space. However, obtaining strong lower bounds come at a high price. The paper presents a novel pruning technique named BIP which can independently reduce the search space only by means of local knowledge and running contexts. To the best of our knowledge, BIP is the first pruning technique independent of bounds for tree-based complete search algorithms to solve a DCOP. Moreover, our proposed BIP can be easily applied to any existing tree-based complete search algorithms for DCOPs with minor modifications. We theoretically prove the correctness of our technique and our empirical evaluation confirms its great superiority. It is worth noting that our proposed BIP is not specific to tree-based complete search algorithms for DCOPs and can be easily adapted to other backtracking search algorithms for distributed and centralized optimization problems.

References

1 James Atlas, Matt Warner, and Keith Decker. A memory bounded hybrid approach to distributed constraint optimization. In *Proceedings 10th International Workshop on DCR*, pages 37–51, 2008.

2 Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999. `doi:10.1126/science.286.5439.509`.

3 Ismel Brito and Pedro Meseguer. Improving DPOP with function filtering. In *Proceedings of the 9th AAMAS*, pages 141–148, 2010.

4 Dingding Chen, Yanchen Deng, Ziyu Chen, Wenxin Zhang, and Zhongshi He. HS-CAI: A hybrid dcop algorithm via combining search with context-based inference. In *Proceedings of the 34th AAAI*, pages 7087–7094, 2020.

5 Ziyu Chen, Wenxin Zhang, Yanchen Deng, Dingding Chen, and Qiang Li. RMB-DPOP: Refining MB-DPOP by reducing redundant inference. In *Proceedings of the 19th AAMAS*, pages 249–257, 2020.

6 Martin C Cooper, Simon De Givry, Martı Sánchez, Thomas Schiex, Matthias Zytnicki, and Tomas Werner. Soft arc consistency revisited. *Artificial Intelligence*, 174(7-8):449–478, 2010. doi:10.1016/j.artint.2010.02.001.

7 Yanchen Deng, Ziyu Chen, Dingding Chen, Xingqiong Jiang, and Qiang Li. PT-ISABB: A hybrid tree-based complete algorithm to solve asymmetric distributed constraint optimization problems. In *Proceedings of the 18th AAMAS*, pages 1506–1514, 2019.

8 Alessandro Farinelli, Alex Rogers, and Nick R Jennings. Agent-based decentralised coordination for sensor networks using the max-sum algorithm. *Autonomous agents and multi-agent systems*, 28(3):337–380, 2014. doi:10.1007/s10458-013-9225-1.

9 Alessandro Farinelli, Alex Rogers, Adrian Petcu, and Nicholas R Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proceedings of the 7th AAMAS*, volume 2, pages 639–646, 2008.

10 Ferdinando Fioretto, Enrico Pontelli, and William Yeoh. Distributed constraint optimization problems and applications: A survey. *Journal of Artificial Intelligence Research*, 61:623–698, 2018. doi:10.1613/jair.5565.

11 Ferdinando Fioretto, William Yeoh, Enrico Pontelli, Ye Ma, and Satishkumar J Ranade. A distributed constraint optimization (DCOP) approach to the economic dispatch with demand response. In *Proceedings of the 16th AAMAS*, pages 999–1007, 2017.

12 Patricia Gutierrez and Pedro Meseguer. BnB-ADOPT+ with several soft arc consistency levels. In *Proceedings of the 19th ECAI*, pages 67–72, 2010.

13 Patricia Gutierrez and Pedro Meseguer. Saving redundant messages in BnB-ADOPT. In *Proceedings of the 24th AAAI*, pages 1259–1260, 2010.

14 Katsutoshi Hirayama and Makoto Yokoo. Distributed partial constraint satisfaction problem. In *International Conference on Principles and Practice of Constraint Programming*, pages 222–236, 1997. doi:10.1007/BFb0017442.

15 Yoonheui Kim and Victor Lesser. DJAO: A communication-constrained DCOP algorithm that combines features of ADOPT and Action-GDL. In *Proceedings of the 28th AAAI*, pages 2680–2687, 2014.

16 Omer Litov and Amnon Meisels. Forward bounding on pseudo-trees for DCOPs and ADCOPs. *Artificial Intelligence*, 252:83–99, 2017. doi:10.1016/j.artint.2017.07.003.

17 Rajiv T Maheswaran, Jonathan P Pearce, and Milind Tambe. A family of graphical-game-based algorithms for distributed constraint optimization problems. In *Coordination of large-scale multiagent systems*, pages 127–146. Springer, 2006. doi:10.1007/0-387-27972-5_6.

18 Rajiv T Maheswaran, Milind Tambe, Emma Bowring, Jonathan P Pearce, and Pradeep Varakantham. Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *Proceedings of the 3rd AAMAS*, volume 1, pages 310–317, 2004.

19 Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149–180, 2005. doi:10.1016/j.artint.2004.09.003.

20 Arnon Netzer, Alon Grubshtein, and Amnon Meisels. Concurrent forward bounding for distributed constraint optimization problems. *Artificial Intelligence*, 193:186–216, 2012. doi:10.1016/j.artint.2012.09.002.

21 Duc Thien Nguyen, William Yeoh, Hoong Chuin Lau, and Roie Zivan. Distributed Gibbs: A linear-space sampling-based dcop algorithm. *Journal of Artificial Intelligence Research*, 64:705–748, 2019. doi:10.1613/jair.1.11400.

22 Brammert Ottens, Christos Dimitrakakis, and Boi Faltings. DUCT: An upper confidence bound approach to distributed constraint optimization problems. *ACM Transactions on Intelligent Systems and Technologyc*, 8(5):69, 2017. doi:10.1145/3066156.

23 Adrian Petcu and Boi Faltings. Approximations in distributed optimization. In *International Conference on Principles and Practice of Constraint Programming*, pages 802–806, 2005. doi:10.1007/11564751_68.

24 Adrian Petcu and Boi Faltings. ODPOP: An algorithm for open/distributed constraint optimization. In *Proceedings of the 21th AAAI*, pages 703–708, 2006.

25 Adrian Petcu and Boi Faltings. MB-DPOP: A new memory-bounded algorithm for distributed optimization. In *Proceedings of the 20th IJCAI*, pages 1452–1457, 2007.

26 Evan A Sultanik, Pragnesh Jay Modi, and William C Regli. On modeling multiagent task scheduling as a distributed constraint optimization problem. In *Proceedings of the 20th IJCAI*, pages 1531–1536, 2007.

27 Meritxell Vinyals, Juan A Rodriguez-Aguilar, and Jesús Cerquides. Generalizing DPOP: DPOP, a new complete algorithm for DCOPs. In *Proceedings of the 8th AAMAS*, pages 1239–1240, 2009.

28 Weixiong Zhang, Guandong Wang, Zhao Xing, and Lars Wittenburg. 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. doi:10.1016/j.artint.2004.10.004.