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
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

\[ A = \{a_1, a_2, \ldots, a_n\} \]

\[ X = \{x_1, x_2, \ldots, x_m\} \]

\[ D = \{d_{i,j} : (a_i, x_j) \in D\} \]

\[ F = \{f_{i,j} : (a_i, x_j) \in F\} \]
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 \(d_{\text{max}} = \max_{a_i \in A} |D_i|\).
F = \{f_1, f_2, ..., f_q\} is a set of constraint functions, where each constraint \(f_i : D_{i_1} \times \cdots D_{i_k} \rightarrow \mathbb{R}_{\geq 0}\) 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., \(f_{ij} : D_i \times D_j \rightarrow \mathbb{R}_{\geq 0}\)). 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,

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

A DCOP can be visualized by a constraint graph 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., \(lb^c_i(d_i)\) and \(ub^c_i(d_i)\)).
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, \( LB_i = UB_i = \text{opt} \) if \( Context_i \) is feasible and \( LB_i = UB_i = \infty \) otherwise, and \( \text{opt} \) 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, \( LB_i \leq \text{opt} \leq UB_i \)) continually. Once the root agent \( a_1 \) has \( LB_1 = UB_1 \), 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** \((\text{dims})\). Let \( U \) be a cost table, \( \text{dims}(U) \) is the set of variables involved in \( U \).

\[
D_U = \times_{x_i \in \text{dims}(U)} D_i \text{ is the set of all value combinations of } \text{dims}(U). \text{ Given } x_i \in \text{dims}(U), D_{U \setminus x_i} = \times_{x_j \in \text{dims}(U) \setminus \{x_i\}} D_j \text{ is the set of all value combinations of } \text{dims}(U) \text{ except } x_i. \text{ Particularly, we specify } D_{U \setminus x_i} = \{\emptyset\} \text{ when } \text{dims}(U) \setminus \{x_i\} = \emptyset. \text{ Take } f_{12} \text{ in Fig. 1(b) for example. We have } \text{dims}(f_{12}) = \{x_1, x_2\} \text{ and } D_{f_{12}} = \{((x_1, 0)), ((x_1, 1)), ((x_1, 2))\}.
\]

\(^2\) 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.
Consider $f_{12}$. When $d_{-2} = ((x_1, 2))$, we have $\text{pos}_{f_{12}}(d_{-2}) = 0$ since $f_{12}(d_{-2}, x_2 = 1) < f_{12}(d_{-2}, x_2 = 2)$ and 0 precedes 1.

According to Definition 2, we can obtain a subset of $D^U$, denoted by $S_i^U$, by Eq. (3).
\[
S_i^U = \{(d_i, d_{-i}) \mid d_i = \text{pos}_{i}(d_{-i}), \forall d_{-i} \in D_{-i}^U\} \tag{3}
\]

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

**Definition 3 (Join).** Let $U, U'$ be two cost tables, the join of $U$ and $U'$, denoted by $U \odot U'$, is a relation defined over $D^U \odot D^U = \times_{x \in \text{dims}(U)}, d \in \text{dims}(U')D_i$ such that
\[
(U \odot U')(d) = U(d_{\text{dims}(U)}) + U'(d_{\text{dims}(U')}) \forall d \in D^U \odot D^U
\]

where $d_{\text{dims}(U)}$ and $d_{\text{dims}(U')}$ are slices of $d$ along dims$(U)$ and dims$(U')$, respectively.

Next, we give the following properties of $S_i^U$, based on which BIP is presented.

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

**Proof.** Assume that $\forall d \in S_i^U, U(d) > U(d')$ where $d^* = (d_i^*, d_{-i}^*)$ is the optimal solution.

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

**Property 5.** Given two cost tables $U$ and $U'$, $S_i^U = S_i^{U'}$ if
\[
U = U' \odot U''
\]

where $\text{dims}(U'') \subseteq \text{dims}(U') \backslash \{x_1\}$ and $x_1 \in \text{dims}(U')$.

**Proof.** Firstly, we prove that $U(d_i, d_{-i}) - U(d_i', d_{-i}) = U'(d_i, d_{-i}) - U'(d_i', d_{-i}); \forall d_i, d_i' \in D_i, d_{-i} \in D_{-i}$.

According to Eq. (4), we have $\text{dims}(U'') = \text{dims}(U)$ and $x_1 \in \text{dims}(U)$. Thus, for all $d_i, d_i' \in D_i$ and $d_{-i} \in D_{-i}$, we have
\[
\begin{aligned}
&U(d_i, d_{-i}) - U(d_i', d_{-i}) \\
&= (U'(d_i, d_{-i}) + U''(d_{-i}[\text{ dims}(U'')])) - (U'(d_i', d_{-i}) + U''(d_{-i}[\text{ dims}(U'')])) \\
&= U'(d_i, d_{-i}) - U'(d_i', d_{-i})
\end{aligned}
\tag{5}
\]

Next, we prove that given $d_{-i} \in D_{-i}$ and $d_i \in D_i$, $U(d_i, d_{-i}) = \min_{d_i' \in D_i} U(d_i', d_{-i})$ if $U'(d_i, d_{-i}) = \min_{d_i' \in D_i} U'(d_i', d_{-i})$.

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

Based on the above conclusion and Definition 2, we have
\[
\text{pos}_U(d_{-i}) = \text{pos}_{U'}(d_{-i})
\tag{6}
\]

Therefore, we can conclude $S_i^U = S_i^{U'}$ based on Eqs. (3) and (6), and thereby Property 5 is proved.
A Bound-Independent Pruning Technique

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 $\text{Desc}(a_i) \setminus CD(a_i)$, 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 $D^{U_i \setminus S_i^U}$ 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,

$$U_i = cd_{-}\text{cost}_i \otimes ap_{-}\text{cost}_i \tag{7}$$

$$cd_{-}\text{cost}_i = \otimes_{a_j \in CD(a_i)} f_{ij} \tag{8}$$

$$ap_{-}\text{cost}_i = \sum_{a_j \in AP(a_i)} f_{ij} (Context_i(x_j)) \tag{9}$$

where $cd_{-}\text{cost}_i$ is the combination of all constraints between $a_i$ and $CD(a_i)$ and $ap_{-}\text{cost}_i$ is the sum of all constraints between $a_i$ and $AP(a_i)$ under the current context.

Theoretically, all the elements in $D^{U_i \setminus S_i^U}$ 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 $D^{U_i \setminus S_i^U}$, 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 $d_i \in D_i \setminus DV_i$, $a_i$ suggests its child $d_{c} \in C(a_i)$ to remove $DV_i^c(d_c)$ computed by Eq. (11) from the domain of $d_{c}$ (i.e., $D_{c}$).

$$DV_i = \{d_i \in D_i | (d_i, d_{-i}) \in D^{U_i \setminus S_i^U}, \forall d_{-i} \in D_i \} \tag{10}$$

$$DV_i^c(d_c) = \{d_{c} \in D_{c} | (d_{c}, d_{-c}) \in D^{U_i \setminus S_i^U}, \forall d_{-c} \in D_{c} \} \tag{11}$$

Here, $D_i^\text{Union}(c,i) = \times_{x_j \in \text{dom}(U_j)}\{x_j, x_{-j}\}D_{j}$.

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 $DV_2^2(1) = \{0, 2\}$ according to Eq. (11). Similarly, we have $DV_2^2(0) = \emptyset$, $DV_2^2(0) = \emptyset$ and $DV_2^2(1) = \{0, 2\}$. Accordingly, $a_2$ obtains all the removed elements shown in gray.

Since $DV_{P(a_i)}^2(Context_2(P(a_i)))$ 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 $\text{Dom}_i$ by:

$$\text{Dom}_i = D_{i} \setminus (DV_i \cup DV_{P(a_i)}^2(Context_2(P(a_i)))) \tag{12}$$

Algorithm 1 gives the sketch of calculating $\text{Dom}_i$ for both synchronous and asynchronous tree-based complete search algorithms when enforcing BIP. Each agent $a_i$ computes $cd_{-}\text{cost}_i$ firstly (line 1). Afterwards, it calculates $DV_i$ and $DV_i^c(d_i)$ by calling $\text{Compute}_{-}\text{DV}^i$() and
\begin{abstract}

We take Fig. 1 as an example to trace BIP running 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 $\text{Compute}_DVs()$ to get $DV_1$ and $DV_1^c(d_1)$ for itself and its child $a_2$, respectively. Afterwards, it computes $Dom_1 = D_1 \setminus DV_1 = \{0, 1\}$ and sends $\{(x_1, 0)\}$ and $DV_1^c(0) = \emptyset$ to its child $a_2$ via a CPA message. (Assume that $a_1$ takes its feasible assignment $\{(x_1, 0)\}$.)

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

\end{abstract}
Table 1: The trace of assignments to the variables of BIP.

<table>
<thead>
<tr>
<th>Cycle(a)</th>
<th>Context</th>
<th>$DV_1$</th>
<th>$DV_2(3)$</th>
<th>Dom</th>
<th>$a_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1(a)</td>
<td>$\emptyset$</td>
<td>(2)</td>
<td>$DV_1(0) = \emptyset$</td>
<td>(0, 1)</td>
<td>0</td>
</tr>
<tr>
<td>2(a)</td>
<td>${(x, 0)}$</td>
<td>(2)</td>
<td>$DV_1(0) = DV_2^3(0) = \emptyset$, $DV_1(1) = DV_2^3(1) = (0, 2)$, $DV_2^3(2) = (0, 1, 2)$</td>
<td>(0, 1)</td>
<td>1</td>
</tr>
<tr>
<td>3(a)</td>
<td>${(x, 0), (x, 1)}$</td>
<td>(0, 1)</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>3(a)</td>
<td>${(x, 0), (x, 1)}$</td>
<td>(0, 1)</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

Cycle 2: When $a_2$ receives the CPA message from $a_1$, it computes $U_2 = f_{12}(x_1 = 0) \perp f_{23} \otimes f_{24}$, calls Compute-$DV_3()$ to get $DV_2$, $DV_2^3(d_2)$ and $DV_2^3(d_4)$ for itself, $a_3$ and $a_4$, respectively, and then computes $Dom_{a_2} = D_2 \setminus (DV_2 \cup DV_2^3(0)) = \{0, 1\}$. 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 $DV_2^3(1)$ to $a_4$.

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

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

Cycle 3: Upon receipt of the CPA message from $a_2$, $a_3$ computes $DV_3$ by calling Compute-$DV_3()$ $^5$, and gets $Dom_{a_3} = D_3 \setminus (DV_3 \cup DV_3^3(1)) = \emptyset$ which means Context$_3$ cannot 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 $AP(P(a_i)) \setminus Sep(a_i)$ is not contained in Context$_i$. Consequently, $a_i$ is searching under its running context while $AP(P(a_i)) \setminus Sep(a_i)$ 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 naive 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 $DV_3^3(P(a_i))(Context_i(P(a_i)))$ 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 $AP(a_i) \setminus Sep(a_c)$ 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., $lb(c)(d_i)$ and $ub(c)(d_i)$) are acceptable or not.

---

$^5$ Note that Compute-$DV_3()$ is also applied to leaves since we specify $DV_3^U = \emptyset$ when $dims(U) \setminus \{x_i\} = \emptyset$. 
Rule 1: $lb(i)(d_i)$ is acceptable under $Dom^c(i)(d_i)$ if $Dom^c(i)(d_i) \subseteq Dom_c(d_i)$; otherwise, discarded.

Rule 2: $ub(i)(d_i)$ is acceptable under $Dom^c(i)(d_i)$ if $Dom^c(i)(d_i) \supseteq Dom_c(d_i)$; otherwise, discarded.

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

Proposition 6. Given $d_i$, $lb(i)(d_i)$ produced under $Dom_c(d_i)$ is acceptable under $Dom^c(i)(d_i)$ if $Dom^c(i)(d_i) \subseteq Dom_c(d_i)$, and $ub(i)(d_i)$ produced under $Dom_c(d_i)$ is acceptable under $Dom^c(i)(d_i)$ if $Dom^c(i)(d_i) \supseteq Dom_c(d_i)$.

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

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

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

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

$$lb(i)(d_i) = \min_{d_i \in Dom'(d_i)} \{LB_c(d_i)\} = \min_{d_i \in Dom'(d_i) \cup Dom'_c(d_i) \cup Dom'_c(d_i)} \{LB_c(d_i)\} = \min_{d_i \in Dom'_c(d_i)} \{LB_c(d_i)\} \leq LB'_c$$

Similarly, we can conclude $ub(i)(d_i) \geq UB'_c$ 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 $a_i$ and $Dom'_c(d_i)$ can be obtained by:

$$Dom'_c(d_i) = D_c \setminus (DV_c \cup DV'_c(d_i))$$

where $DV'_c(d_i)$ is computed by $a_i$ based on Eq. (11) and $DV_c$ can also be piggybacked by a COST message from $a_i$. When the search results are unacceptable, $a_i$ attaches a Boolean variable $ReqCost'_i(d_i)$ to the VALUE message to request a COST message from $a_i$.

VALUE($a_i, d_i, ID, TH, DV'_c(d_i), ReqCost'_i(d_i)$)

COST($a_i, context_i, LB_i, UB_i, ThReq, Dom_i, DV_i$)

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 $DV'_c(d_i)$ and $ReqCost'_i(d_i)$ 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,
A Bound-Independent Pruning Technique

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     $d_i = \text{Context}_c(a_i)$
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$_i^c(d_i) \leftarrow \text{true}$

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$_i^c(d_i) \leftarrow \text{true}$

When sending a VALUE to $a_c \in C(a_i)$:
33   if ReqCost$_i^c(d_i) = \text{true}$ then
34       attach ReqCost$_i^c(d_i)$ to the VALUE to request a COST from $a_c$
35       ReqCost$_i^c(d_i) \leftarrow \text{false}$

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 $cd_{\text{cost}}$, which requires the memory consumption of $d_{\text{max}}^{CD(a_i)}+1$. Thus, we introduce a parameter $k$ to specify the maximum memory budget (i.e., $d_{\text{max}}^{k}$) 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., $d_{\text{max}}^{k-|CD(a_i)|-1}$) 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 $DV_i^r(d_i)$ to a CPA message for synchronous tree-based algorithms, $DV_i^v(d_i)$ and $ReqCost_i^v(d_i)$ to a VALUE message and $Dom_i$ and $DV_i$ to a C 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 $O(d_{max}^i)$ for synchronous tree-based algorithms and $O(d_{max}^i + |C(a_i)|d_{max}^i)$ for asynchronous tree-based algorithms. Here, $d_{max}^i$ is the memory consumption as mentioned in Subsection 3.4 and the memory of $|C(a_i)|d_{max}^i$ is used for storing $Dom_i^r(d_i)$ and $DV_i^r(d_i)$ when performing ATM.

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

4 Theoretical Results

In the section, we prove the correctness 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). $U_i^{U_i}$ is the combination cost table of all constraints without $a_i$ in the subproblem. That is,

$$
U_i^{U_i} = \bigotimes_{a_j \in Desc(a_i)} \bigotimes_{a_k \in AP(a_i) \setminus (Sep(a_i) \cup \{a_i\})} f_{jk} \\
\otimes \bigotimes_{a_k \in AP(a_i) \cap Sep(a_i) \setminus \{\text{Context}_i(x_i)\}}
$$

(13)

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

$$
U_i^{sub} = \min_{Desc(a_i) \setminus CD(a_i)} (U_i \otimes U_i^{U_i})
$$

(14)

**Lemma 7.** $U_i'' = \min_{Desc(a_i) \setminus CD(a_i)} U_i^{U_i}$ is a cost table such that $U_i^{sub} = U_i \otimes U_i''$ and $\dim(U_i) \subseteq \dim(U_i')$.

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

$$
U_i^{sub} = \min_{Desc(a_i) \setminus CD(a_i)} (U_i \otimes U_i^{U_i}) = U_i \otimes \min_{Desc(a_i) \setminus CD(a_i)} U_i^{U_i}
$$

The equation from the first step to the second step holds since $\dim(U_i') = CD(x_i) \cup \{x_i\}$ according to Eq. (7) and $(CD(x_i) \cup \{x_i\}) \cap (Desc(a_i) \setminus CD(a_i)) = \emptyset$. Further, according to Eq. (13), we have $\dim(U_i^{U_i}) = Desc(a_i)$ and thus $\dim(U_i'') \subseteq \dim(U_i) \setminus \{x_i\}$. 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 \( \exists d \in S_{U_i}^{U_{\text{sub}}} \), s.t. \( U_{\text{sub}}^i(d) = \min_{d' \in D_{U_i}^{U_{\text{sub}}}} U_{\text{sub}}^i(d') \). Further, according to Lemma 7 and Property 5, we have \( S_{U_i}^{U_{\text{sub}}} = S_{U_i}^{U_{\text{sub}}} \) and \( \exists d \in S_{U_i}^{U_{\text{sub}}} \), s.t. \( U_{\text{sub}}^i(d) = \min_{d' \in D_{U_i}^{U_{\text{sub}}}} U_{\text{sub}}^i(d') \). Therefore, we can conclude that the optimal solution of the subproblem rooted at \( a_i \) is the join of \( d \) and the optimal assignment to \( \text{Desc}(a_i) \setminus \text{CD}(a_i) \) from Eq. (14). Thus, Lemma 8 is proved.

\[ \square \]

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

Proof. For any given context \( \text{Context}_i \), there exists one element in \( S_{U_i}^{U_{\text{sub}}} \) 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 \( S_{U_i}^{U_{\text{sub}}} \) according to Eqs. (10) and (11). Thus, Theorem 9 is proved.

\[ \square \]

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 random 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., \( O(|C(a_i)|d^{k}_{\text{max}}) \)). 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
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

---

**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.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>( k )</th>
<th>BnB-ADOPT(^+)-FDAC+BIP</th>
<th>PT-FB+BIP</th>
<th>HS-CAI+BIP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Maps(%)</td>
<td>NL(%)</td>
<td>NCLOs(%)</td>
</tr>
<tr>
<td>Sparse</td>
<td>4</td>
<td>36-65</td>
<td>29-55</td>
<td>58-77</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>82-88</td>
<td>77-85</td>
<td>89-92</td>
</tr>
<tr>
<td>Dense</td>
<td>4</td>
<td>35-51</td>
<td>25-45</td>
<td>43-65</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>69-84</td>
<td>64-81</td>
<td>72-91</td>
</tr>
<tr>
<td>Domain Size</td>
<td>8</td>
<td>85-89</td>
<td>82-86</td>
<td>90-93</td>
</tr>
<tr>
<td>Scale-Free</td>
<td>4</td>
<td>52-61</td>
<td>21-43</td>
<td>56-72</td>
</tr>
<tr>
<td>Networks</td>
<td>8</td>
<td>68-84</td>
<td>62-67</td>
<td>53-90</td>
</tr>
</tbody>
</table>

---

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.

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).
41:14 A Bound-Independent Pruning Technique

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.

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
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