BIP Revisited: Enabling Efficient Dominance Pruning for Distributed Constraint Optimization

Xiangshuang Liu[1,2] , **Ziyu Chen**[1,*]
[1]College of Computer Science, Chongqing University, Chongqing, China
[2]Chongqing Chuanyi Automation Co.,Ltd. Measurement & Control Technology Branch, Chongqing, China
shuxliu21@163.com, chenziyu@cqu.edu.cn

Abstract

Bound-Independent Pruning (BIP) is a powerful technique for accelerating tree-based complete search algorithms for Distributed Constraint Optimization Problems (DCOPs) by pruning dominated regions of the search space using only local information. However, BIP fundamentally relies on constructing a context-dependent dominating space through joint optimization over local constraints, incurring exponential computational and storage complexity that limits its practical scalability. To address this limitation, we propose DR-BIP, a decoupled and sound reformulation of BIP that enables efficient identification of dominated subspaces without explicitly computing the dominating space. By reformulating dominated-subspace detection into pairwise dominance tests and decoupling them across independent constraints, DR-BIP eliminates joint optimization and reduces both computational and storage complexity from exponential to polynomial. The resulting dominance components can be precomputed and reused across context changes, further improving efficiency. We formally prove the correctness of DR-BIP and integrate it into several state-of-the-art tree-based complete DCOP algorithms. Extensive empirical evaluation on standard benchmarks confirms that DR-BIP consistently outperforms BIP, significantly reducing both computation time and communication overhead.

1 Introduction

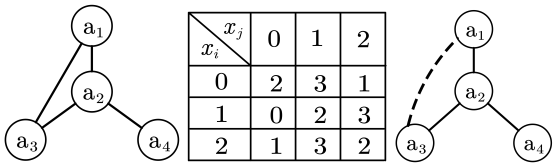
Distributed Constraint Optimization Problems (DCOPs) [Hirayama and Yokoo, 1997; Fioretto *et al.*, 2018] provide a foundational modeling framework for cooperative multi-agent systems with wide-ranging applications in areas such as sensor networks [Farinelli *et al.*, 2014], scheduling [Pertzovsky *et al.*, 2024] and smart grid [Fioretto *et al.*, 2017].

DCOP algorithms are broadly classified as incomplete or complete. Incomplete algorithms [Maheswaran *et al.*, 2006; Farinelli *et al.*, 2008; Ottens *et al.*, 2017; Nguyen *et al.*, 2019; Cohen *et al.*, 2020; Deng *et al.*, 2025] find good solutions efficiently, sacrificing optimality. Complete algorithms, in contrast, guarantee optimality through systematic inference or search. Inference-based complete algorithms like DPOP [Petcu and Faltings, 2005] and Action_GDL [Vinyals *et al.*, 2009] utilize dynamic programming to solve a DCOP optimally but require linear messages of exponential size. To address this limitation, subsequent improvements such as MB-DPOP and its variant [Petcu and Faltings, 2007; Chen *et al.*, 2020b] apply cycle-cutting techniques to partition messages into smaller components. Similarly, DPOP with function filtering [Brito and Meseguer, 2010] enhances efficiency by pruning suboptimal tuples through iterative refinement of cost bounds via agent communication.

Alternatively, search-based complete algorithms perform distributed backtracking with linear message sizes but exponential message counts. They rely on bound-dependent pruning, where lower bounds are tightened via communication to prune suboptimal regions. Numerous approaches have been developed to strengthen these bounds: BnB-ADOPT$^+$-AC/FDAC [Gutierrez and Meseguer, 2010] collects projected costs via multi-level arc consistency operations; PT-FB[Litov and Meisels, 2017] employs forward bounding, where agents communicate with unassigned neighbors to obtain bound estimates; ADOPT-BDP[Atlas *et al.*, 2008], DJAO [Kim and Lesser, 2014] and PT-ISABB [Deng *et al.*, 2019] utilize preprocessing inference to collect utility information; and subsequently, HS-CAI [Chen *et al.*, 2020a] further tightens bounds through iterative context-based inference. While these approaches improve pruning effectiveness, their reliance on iterative bound refinement necessitates substantial information exchange among agents, resulting in significant communication and coordination overhead.

To transcend these limitations, a distinct pruning paradigm has emerged. Bound-Independent Pruning (BIP) [Liu *et al.*, 2021] bypasses the reliance on dynamic bound propagation by exploiting intrinsic dominance relationships derived from local constraints. This approach enables agents to identify and prune provably suboptimal regions using only their local knowledge, while maintaining completeness guarantees. However, BIP requires the explicit construction of a context-dependent dominating space through joint optimization over all local constraints, incurring exponential computational and storage complexity.

*Corresponding author



x_i \ x_j	0	1	2
0	2	3	1
1	0	2	3
2	1	3	2

(a) Constraint graph (b) Constraint matrix (c) Pseudo tree

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

To address this challenge, we propose DR-BIP (Decoupled Reformulation of BIP), a novel and sound approach that enables efficient dominance pruning without constructing the dominating space. Our key contributions are: (1) we reformulate dominated-subspace detection into a set of pairwise dominance tests, decouple them across independent constraints into reusable components, and thereby reduce both computational and storage complexity from exponential to polynomial; (2) we formally prove the soundness of DR-BIP by showing its pruning is a subset of BIP's, ensuring compatibility with the BIP framework and enabling modular integration into state-of-the-art tree-based complete DCOP algorithms; (3) we demonstrate through extensive experiments that DR-BIP consistently and significantly outperforms both original methods and BIP-enhanced baselines, reducing both computation and communication overhead across diverse benchmark settings.

2 Background

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

2.1 Distributed Constraint Optimization Problems

A distributed constraint optimization problem [Modi *et al.*, 2005] can be defined by a tuple $\langle A, X, D, F\rangle$ where $A = \{a_1, a_2, ..., a_n\}$ 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 and $F = \{f_1, f_2, ..., f_q\}$ is a set of constraint functions. Each variable x_i takes a value from its domain D_i and each constraint $f_i : D_{i_1} \times \cdots D_{i_k} \rightarrow \mathbb{R}_{\geqslant 0}$ specifies a cost for each combination of $x_{i_1}, ..., x_{i_k}$. Here, we denote the maximal domain size as $d_{max} = \max_{x_i \in X} |D_i|$.

For the sake of simplicity, we follow the common assumptions that all constraints are binary[1] (i.e., $f_{ij} : D_i \times D_j \rightarrow \mathbb{R}_{\geqslant 0}$) and each agent controls only one variable (i.e., $n = m$). Thus, the term $agent$ and $variable$ can be used interchangeably. An optimal solution to a DCOP is an assignment to all the variables that minimizes the total cost. That is,

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

A DCOP can be visualized as 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 constraints are identical as shown in Fig. 1(b) where $i < j$.

[1]Non-binary constraints can be converted to binary ones via the dual or hidden variable transformation [Bacchus *et al.*, 2002].

2.2 Pseudo Tree

A pseudo tree is a partial ordered arrangement to a constraint graph and can be generated via depth-first search traversal, where different branches are independent of each other. It categorizes constraints into tree edges and pseudo edges (i.e., non-tree edges), thereby allowing the neighbors of an agent a_i to be classified, based on the connecting edge type and relative position, as its parent $P(a_i)$, pseudo parents $PP(a_i)$, children $C(a_i)$ and pseudo children $PC(a_i)$. We also denote all its (pseudo) parents as $AP\left(a_i\right) = PP\left(a_i\right) \cup \{P\left(a_i\right)\}$, all its (pseudo) children as $CD(a_i) = C(a_i) \cup PC(a_i)$. The set of ancestors that constrain a_i and its descendants is called the separator $Sep(a_i)$ [Petcu and Faltings, 2006]. A sample pseudo tree derived from Fig. 1(a) is shown in Fig. 1(c) where the dashed edge is a pseudo edge.

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 subspace by exploiting bounds. $Context_i$ is updated upon receiving a context message from a_i's (pseudo) parents. When such an update occurs, a_i first computes ancestor-constrained cost ap_cost_i for each $d_i \in D_i$, to initialize its lower bound estimates:

$$ap\_cost_i(d_i) = \sum_{a_j \in AP(a_i)} f_{ij}\left(d_i, Context_i(x_j)\right).$$

The search then proceeds in either a synchronous or asynchronous manner. In synchronous algorithms, agents must follow a fixed order: a_i and its descendants exhaustively explore the subproblem under $Context_i$ before a_i reports its final search results, including bounds, to its parent. In contrast, asynchronous algorithms allow agents to continuously update their assignments based on their local view and to continually report their bounds to their parents.

Upon receiving a search-result message from a child, agent a_i updates its own bounds by incorporating the received bounds. These updated bounds are then used to prune suboptimal subspaces. Through this iterative process of bound propagation and refinement, agents collaboratively converge to an optimal solution.

2.4 Bound-Independent Pruning Technique

Bound-Independent Pruning (BIP) accelerates tree-based complete search algorithms by pruning dominated regions of the search space using only local knowledge, without relying on bounds. Given the current context $Context_i$, agent a_i first joins all its local constraints involving descendant agents $a_c \in CD(a_i)$ together with the ancestor-constrained cost based on the context into a single cost table:

$$U_i\left(d_i, \boldsymbol{d}_{-i}\right) = \sum_{a_c \in CD(a_i)} f_{ic}\left(d_i, \boldsymbol{d}_{-i|x_c}\right) + ap\_cost_i\left(d_i\right), \quad \forall d_i \in D_i, \quad \boldsymbol{d}_{-i} \in D_{-i}^{U_i}, \tag{1}$$

where $D_{-i}^{U_i} = \times_{a_c \in CD(a_i)} D_c$, and $\boldsymbol{d}_{-i|x_c}$ denotes the component of $\boldsymbol{d}_{-i}$ corresponding to variable x_c.

The dominating space (DS) is then constructed by selecting, for each joint assignment $\boldsymbol{d}_{-i}$ of its descendants, the optimal value for its own variable:

$$S_i^{U_i} = \left\{ (d_i^*, \boldsymbol{d}_{-i}) \middle| d_i^* = \arg\min_{d_i \in D_i} U_i\left(d_i, \boldsymbol{d}_{-i}\right), \forall \boldsymbol{d}_{-i} \in D_{-i}^{U_i} \right\}. \quad (2)$$

Ties in the selection of d_i^* are broken alphabetically. The DS contains at least one assignment that can be extended to an optimal solution for the sub-problem rooted at a_i. Its complement, called the Dominated Subspace (DDS) and denoted by $D^{U_i} \setminus S_i^{U_i}$ (where $D^{U_i} = D_i \times D_{-i}^{U_i}$), can be pruned without losing completeness.

BIP prunes the DDS through a hierarchical process. First, for each value $d_i \in D_i$, the coarse subspace

$$D_{\langle d_i \rangle}^{U_i} = \{\boldsymbol{d} \in D^{U_i} \mid d_i \in \boldsymbol{d}\}$$

is considered. If the whole coarse subspace lies inside the DDS, i.e., $D_{\langle d_i \rangle}^{U_i} \subseteq D^{U_i} \setminus S_i^{U_i}$, d_i is removed from a_i's domain. Formally, the set of such dominated values is:

$$DV_i = \left\{ d_i \in D_i \middle| D_{\langle d_i \rangle}^{U_i} \subseteq D^{U_i} \setminus S_i^{U_i} \right\}. \quad (3)$$

Next, for each remaining value $d_i \in D_i \setminus DV_i$ and for each child $a_c \in C(a_i)$, BIP further examines finer subspaces obtained by also fixing the child's value d_c:

$$D_{\langle d_i, d_c \rangle}^{U_i} = \{\boldsymbol{d} \in D^{U_i} \mid d_i, d_c \in \boldsymbol{d}\}.$$

If $D_{\langle d_i, d_c \rangle}^{U_i} \subseteq D^{U_i} \setminus S_i^{U_i}$, value d_c can be pruned when a_i takes value d_i. The prunable child values are:

$$DV_i^c\left(d_i\right) = \left\{ d_c \in D_c \middle| D_{\langle d_i, d_c \rangle}^{U_i} \subseteq D^{U_i} \setminus S_i^{U_i} \right\}, \quad d_i \in D_i \setminus DV_i, a_c \in C\left(a_i\right). \quad (4)$$

Child a_c removes values in $DV_i^c(d_i)$ from its domain whenever the context assigns $x_i = d_i$. Thus, the domain of agent a_i after pruning is:

$$Dom_i = D_i \setminus \left(DV_i \cup DV_{P(a_i)}^i \left(Context_i(P(a_i)) \right) \right). \quad (5)$$

where $DV_{P(a_i)}^i(\cdot)$ is the set of prunable values computed by a_i's parent and piggy-backed through context messages.

3 Motivation

BIP suffers from exponential computational and storage overhead because it must explicitly construct the DS through joint optimization over all local constraints. For instance, consider agent a_2 in Fig. 1 with $Context_2 = \{(x_1, 1)\}$. To apply BIP, a_2 must first compute $S_2^{U_2}$ by jointly optimizing $f_{12}(x_1 = 1)$, f_{23}, and f_{24} over their shared variable x_2. This requires exhaustively enumerating all combinations in U_2 (illustrated in Fig. 2, where the elements of $S_2^{U_2}$ are highlighted in bold). Subsequently, a_2 derives the dominated subspaces $D_{\langle x_2=2 \rangle}^{U_2}$, $D_{\langle x_2=1, x_3=2 \rangle}^{U_2}$, and $D_{\langle x_2=1, x_4=2 \rangle}^{U_2}$ (shown as gray regions in Fig. 2). Clearly, all local descendant constraints and the context-dependent ancestor cost are computationally coupled through this joint optimization, incurring an exponential computational overhead of $O(d_{\max}^{|CD(a_i)|+1})$ per context change. Besides, to avoid recomputation upon context changes, BIP precomputes and stores the combination of all local descendant constraints, leading to an exponential storage overhead of $O(d_{\max}^{|CD(a_i)|+1})$.

x_2	x_3	x_4	U
0	0	0	4
0	0	1	5
0	**0**	**2**	3
0	1	0	5
0	**1**	**1**	6
0	**1**	**2**	4
0	**2**	**0**	3
0	**2**	**1**	4
0	**2**	**2**	2

(a) $x_2 = 0$.

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

(b) $x_2 = 1$.

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

(c) $x_2 = 2$.

Figure 2: U_2 under $Context_2 = \{(x_1, 1)\}$.

The exponential complexity of BIP limits its use in dense, large-scale problems. To mitigate this, BIP introduces a parameter k that restricts its application to agents with $|CD(a_i)|+1 \leq k$. This restriction, however, severely curtails pruning power. Therefore, we aim to reformulate the identification of dominated subspaces to circumvent the expensive construction of the DS, thereby unleashing the pruning potential of BIP while avoiding exponential overhead.

4 Proposed Method

In this section, we introduce DR-BIP, a decoupled and sound reformulation for efficiently identifying dominated subspaces in DCOP search. Unlike BIP, DR-BIP directly identifies dominated subspaces through a series of decoupled pairwise dominance tests based on formal dominance properties (defined in Section 4.1). Consequently, the complexity of dominated subspace identification drops from exponential (in BIP) to polynomial.

4.1 Dominance and Conditional Dominance

We begin by formalizing the notion of dominance between values of a variable. Intuitively, a value dominates another if it always yields no worse cost, regardless of how the remaining variables are assigned.

Definition 1 (Dominance)**.** *Let U_i be a cost table with scope containing x_i, and let d_i, $d_i' \in D_i$ be two values of x_i. Value d_i' dominates d_i with respect to U_i, denoted by $d_i' \prec_{U_i} d_i$, iff*

$$U_i\left(d_i', \boldsymbol{d}_{-i}\right) \leq U_i\left(d_i, \boldsymbol{d}_{-i}\right), \quad \forall \boldsymbol{d}_{-i} \in D_{-i}^{U_i}, \quad (6)$$

where $D_{-i}^{U_i}$ is the joint domain of all variables in the scope of U_i except x_i. If equality holds for some $\boldsymbol{d}_{-i}$, ties are broken alphabetically in favor of d_i'.

A direct consequence of dominance is that the entire subspace corresponding to a dominated value can be pruned.

***Property* 1.** *Given a cost table U_i and a value $d_i \in D_i$, if there exists a value $d_i' \in D_i$ such that $d_i' \prec_{U_i} d_i$, then $D_{\langle d_i \rangle}^{U_i} \subseteq D^{U_i} \setminus S_i^{U_i}$.*

Proof. For any $\boldsymbol{d} = (d_i, \boldsymbol{d}_{-i}) \in D_{\langle d_i \rangle}^{U_i}$, define $m = \min_{d \in D_i} U_i(d, \boldsymbol{d}_{-i})$. We have $U_i(d_i', \boldsymbol{d}_{-i}) \leq U_i(d_i, \boldsymbol{d}_{-i})$

from $d'_i \prec_{U_i} d_i$. If $U_i(d_i, \boldsymbol{d}_{-i}) = m$, then $U_i(d'_i, \boldsymbol{d}_{-i}) = m$ by the inequality. In this case, d'_i precedes d_i alphabetically and is chosen by tie-breaking. If $U_i(d_i, \boldsymbol{d}_{-i}) > m$, d_i is not a minimizer for $\boldsymbol{d}_{-i}$. In both cases $(d_i, \boldsymbol{d}_{-i}) \notin S_i^{U_i}$. Hence $D^{U_i}_{\langle d_i \rangle} \cap S_i^{U_i} = \emptyset$, i.e., $D^{U_i}_{\langle d_i \rangle} \subseteq D^{U_i} \setminus S_i^{U_i}$. □

Property 1 reduces the problem of detecting a (potentially large) dominated subspace to verifying whether a single value d_i is dominated by some other value d'_i. We extend this idea to conditional dominance, where the comparison is restricted to a specific slice of the cost table obtained by fixing a conditional variable x_c to a value d_c.

Definition 2 (Conditional Dominance)**.** *Let U_i be a cost table whose scope includes variables x_i and x_c. For d_i, $d'_i \in D_i$ and a fixed value $d_c \in D_c$. d'_i conditionally dominates d_i given $x_c = d_c$, denoted by $d'_i \prec_{U_i(d_c)} d_i$, if f*

$$U_i\left(d'_i, d_c, \boldsymbol{d}_{-(i,c)}\right) \leq U_i\left(d_i, d_c, \boldsymbol{d}_{-(i,c)}\right), \ \forall \boldsymbol{d}_{-(i,c)} \in D^{U_i}_{-(i,c)}, \tag{7}$$

where $D^{U_i}_{-(i,c)}$ is the joint domain of all variables in the scope of U_i except x_i and x_c. If equality holds for some $\boldsymbol{d}_{-(i,c)}$, ties are broken alphabetically in favor of d'_i.

Analogously, conditional dominance implies that a finer subspace can be pruned.

***Property* 2.** *Given a cost table U_i, a value $d_i \in D_i$, and a value $d_c \in D_c$ of a conditional variable x_c, if there exists a value $d'_i \in D_i$ such that $d'_i \prec_{U_i(d_c)} d_i$, then $D^{U_i}_{\langle d_i, d_c \rangle} \subseteq D^{U_i} \setminus S_i^{U_i}$.*

The proof follows a reasoning analogous to Property 1. Properties 1–2 establish pruning via pairwise dominance. Proposition 1 formalizes the relation to BIP: DR-BIP prunes a subset of BIP's subspaces, ensuring its soundness.

Proposition 1. *For any agent a_i and context Context$_i$,*

$$DV_i^{\text{DR-BIP}} \subseteq DV_i^{\text{BIP}}, \quad DV_i^c(d_i)^{\text{DR-BIP}} \subseteq DV_i^c(d_i)^{\text{BIP}}.$$

Proof. If $d_i \in DV_i^{\text{DR-BIP}}$, then by Def. 1 there exists d'_i such that $d'_i \prec_{U_i} d_i$. By Property 1, $D^{U_i}_{\langle d_i \rangle} \subseteq D^{U_i} \setminus S_i^{U_i}$, which by Eq. (3) implies $d_i \in DV_i^{\text{BIP}}$. If $d_c \in DV_i^c(d_i)^{\text{DR-BIP}}$, then by Def. 2 there exists d'_i such that $d'_i \prec_{U_i(d_c)} d_i$. By Property 2, $D^{U_i}_{\langle d_i, d_c \rangle} \subseteq D^{U_i} \setminus S_i^{U_i}$, which by Eq. (4) implies $d_c \in DV_i^c(d_i)^{\text{BIP}}$. Thus both subset relations hold. □

Illustrative Example. In Fig. 2, value 0 of x_2 dominates value 2 ($0 \prec_{U_2} 2$), which by Property 1 allows the entire subspace $D^{U_2}_{\langle x_2=2 \rangle}$ (the left gray columns in Fig. 2(c)) to be pruned; meanwhile, given $x_3 = 2$, value 0 of x_2 conditionally dominates value 1 ($0 \prec_{U_2(x_3=2)} 1$), which by Property 2 permits the finer subspace $D^{U_2}_{\langle x_2=1, x_3=2 \rangle}$ (gray cells in the bottom three rows of Fig. 2(b)) to be pruned.

4.2 Decoupled Dominance Evaluation

To efficiently test dominance, we reformulate the pairwise condition as an optimization query. From Def. 1, $d'_i \prec_{U_i} d_i$ holds iff

$$\max_{\boldsymbol{d}_{-i} \in D^{U_i}_{-i}} \left(U_i\left(d'_i, \boldsymbol{d}_{-i}\right) - U_i\left(d_i, \boldsymbol{d}_{-i}\right)\right) \leq 0, \tag{8}$$

with alphabetical precedence enforced when the maximum equals zero. The conditional version is:

$$\max_{\boldsymbol{d}_{-(i,c)} \in D^{U_i}_{-(i,c)}} \left(U_i\left(d'_i, d_c, \boldsymbol{d}_{-(i,c)}\right) - U_i\left(d_i, d_c, \boldsymbol{d}_{-(i,c)}\right)\right) \leq 0. \tag{9}$$

The key challenge is to evaluate these maximizations without enumerating the exponentially many assignments in $D^{U_i}_{-i}$. We achieve this by exploiting the structure of U_i.

Recall that U_i is defined as the sum of a_i's local constraints under the current context (Eq. 1):

$$U_i\left(d_i, \boldsymbol{d}_{-i}\right) = \sum_{a_c \in CD(a_i)} f_{ic}\left(d_i, \boldsymbol{d}_{-i|x_c}\right) + ap\_cost_i\left(d_i\right).$$

Substituting this expression into Eq. (8) yields:

$$\begin{aligned}
\overline{\Delta U_i}\left(d'_i, d_i\right) &= \max_{\boldsymbol{d}_{-i} \in D^{U_i}_{-i}} \left(U_i\left(d'_i, \boldsymbol{d}_{-i}\right) - U_i\left(d_i, \boldsymbol{d}_{-i}\right)\right) \\
&= \max_{\boldsymbol{d}_{-i} \in D^{U_i}_{-i}} \Bigg[\sum_{a_c \in CD(a_i)} f_{ic}(d'_i, \boldsymbol{d}_{-i|x_c}) + ap_cost_i\left(d'_i\right) \\
&\qquad - \sum_{a_c \in CD(a_i)} f_{ic}(d_i, \boldsymbol{d}_{-i|x_c}) - ap_cost_i\left(d_i\right) \Bigg] \\
&= \max_{\boldsymbol{d}_{-i} \in D^{U_i}_{-i}} \Bigg[\sum_{a_c \in CD(a_i)} \left(f_{ic}\left(d'_i, \boldsymbol{d}_{-i|x_c}\right) - f_{ic}\left(d_i, \boldsymbol{d}_{-i|x_c}\right)\right) \Bigg] \\
&\qquad + \left(ap_cost_i\left(d'_i\right) - ap_cost_i\left(d_i\right)\right).
\end{aligned} \tag{10}$$

The term $ap\_cost_i\left(d'_i\right) - ap\_cost_i\left(d_i\right)$ is independent of $\boldsymbol{d}_{-i}$ and can be pulled out of the maximization. The remaining maximization involves only the static constraints f_{ic}. Crucially, each f_{ic} depends only on x_i and its neighbor x_c; there is no coupling among different x_c's. Thus, the maximization over the joint domain $D^{U_i}_{-i} = \times_{a_c \in CD(a_i)} D_c$ can be decoupled into independent maximizations over each D_c:

$$\begin{aligned}
&\max_{\boldsymbol{d}_{-i} \in D^{U_i}_{-i}} \sum_{a_c \in CD(a_i)} \left(f_{ic}\left(d'_i, \boldsymbol{d}_{-i|x_c}\right) - f_{ic}\left(d_i, \boldsymbol{d}_{-i|x_c}\right)\right) \\
&= \sum_{a_c \in CD(a_i)} \max_{d_c \in D_c} \left(f_{ic}\left(d'_i, d_c\right) - f_{ic}\left(d_i, d_c\right)\right).
\end{aligned} \tag{11}$$

Define the static difference matrix for $a_c \in CD(a_i)$ as

$$\overline{\Delta f_{ic}}(d'_i, d_i) \quad = \max_{d_c \in D_c} \left(f_{ic}\left(d'_i, d_c\right) - f_{ic}\left(d_i, d_c\right)\right). \tag{12}$$

Summing over all descendants gives the aggregated static difference:

$$\overline{\Delta cd\_cost_i}(d'_i, d_i) \quad = \sum\nolimits_{a_c \in CD(a_i)} \overline{\Delta f_{ic}}(d'_i, d_i). \tag{13}$$

Then

$$\begin{aligned}
\overline{\Delta U_i}\left(d'_i, d_i\right) &= \overline{\Delta cd_cost_i}(d'_i, d_i) \\
&\quad + \left(ap_cost_i\left(d'_i\right) - ap_cost_i\left(d_i\right)\right).
\end{aligned} \tag{14}$$

$\overline{\Delta cd\_cost_i}(d'_i, d_i)$ can be precomputed before search begins and reused throughout the search, because they do not depend on the dynamic context. Only the unary term ap_cost_i needs to be updated when the context change.

Similarly, for conditional dominance (Eq. (9)), a corresponding derivation for each child $a_c \in C(a_i)$ yields:

$$\overline{\Delta U_i^c}(d_i', d_i, d_c) = \overline{\Delta U_i}(d_i', d_i) + \Delta f_{ic}(d_i', d_i, d_c), \quad (15)$$

where

$$\Delta f_{ic}(d_i', d_i, d_c) = f_{ic}(d_i', d_c) - f_{ic}(d_i, d_c) - \overline{\Delta f_{ic}}(d_i', d_i). \quad (16)$$

Again, Δf_{ic} is static and can be precomputed. Thus, evaluating (conditional) dominance reduces to a lookup of precomputed tables plus simple arithmetic operations.

Illustrative Example. Take Fig. 2 as an example. To check if value 2 of x_2 is dominated, agent a_2 evaluates $\overline{\Delta U_2}(0,2) = \overline{\Delta cd\_cost_2}(0,2) + \big(ap\_cost_2(0) - ap\_cost_2(2)\big) = 2 + (-3) = -1 < 0$, allowing the entire subspace $D^{U_2}_{\langle x_2=2\rangle}$ to be pruned. To check if value 2 of x_3 is prunable when $x_2 = 1$, it computes $\overline{\Delta U_2^3}(0,1,2) = \overline{\Delta U_2}(0,1) + \Delta f_{23}(0,1,2) = 2 + (-4) = -2 < 0$, justifying the removal of the finer subspace $D^{U_2}_{\langle x_2=1,x_3=2\rangle}$. Both decisions are made directly via a few table lookups and simple arithmetic operations, without enumerating exponential subspaces.

4.3 Algorithm and Complexity

Algorithm. Algorithm 1 outlines the DR-BIP procedure integrated into a tree-based complete search algorithm. Each agent a_i executes two phases: a precomputation phase performed once before the search, and a runtime phase triggered whenever the agent's context changes (or when it is the root agent). In the precomputation phase (PRECOMPUTE()), agent a_i computes the static table $\overline{\Delta cd\_cost}_i$ and for each child $a_c \in C(a_i)$, the table Δf_{ic} (lines 1-2). These tables are context-independent and can be reused throughout the search.

The runtime phase (COMPUTE_DVS()) constructs the sets of prunable values: DV_i (for agent a_i itself) and, for each child a_c, $DV_i^c(d_i)$. First, a_i initializes DV_i and all $DV_i^c(d_i)$ to empty (lines 3-4). Then, for each value $d_i \in D_i$ that has not been pruned by its parent (via DR-BIP) nor by itself (via bound-dependent pruning) (lines 5-6), it calls COMPUTE_DV(d_i) to test whether d_i is (conditionally) dominated (line 7). COMPUTE_DV(d_i) first builds the set $D_i' \subseteq D_i$ of candidate dominating values (line 8), then tests dominance by evaluating $\overline{\Delta U_i}(d_i', d_i)$ for each $d_i' \in D_i'$ (lines 9-10). If a dominating value d_i' is found, d_i is added to DV_i and the subroutine returns (lines 11-12). If no dominating value is found, a_i proceeds to test conditional dominance through its children. Specifically, for each child $a_c \in C(a_i)$ and its value $d_c \in D_c$, it evaluates $\overline{\Delta U_i^c}(d_i', d_i, d_c)$ for each $d_i' \in D_i'$ (lines 13-16). If a dominating value d_i' exists for a given d_c, that d_c is added to $DV_i^c(d_i)$ (line 17).

If after processing a child a_c we have $DV_i^c(d_i) = D_c$ (line 18), then for every value d_c of a_c, there exists some (possibly different) value d_i' of a_i that conditionally dominates d_i. Consequently, d_i cannot lead to an optimal solution regardless of how a_c is assigned; it is therefore collectively dominated (by potentially different d_i' for each d_c) and added to DV_i (line 19). The evaluation for d_i terminates (line 20), avoiding redundant checks for the remaining children.

Algorithm 1: Calculating dominated values for a_i

```
Function PRECOMPUTE():
1   compute Δcd_cost_i(d_i', d_i) by Eq. (13), ∀d_i', d_i ∈ D_i
2   compute Δf_ic(d_i', d_i, d_c) by Eq. (16), ∀a_c ∈ C(a_i),
      ∀d_c ∈ D_c, d_i', d_i ∈ D_i
Function COMPUTE_DVS():
3   DV_i ← ∅
4   DV_i^c(d_i) ← ∅, ∀d_i ∈ D_i, a_c ∈ C(a_i)
5   foreach d_i ∈ D_i do
6       if d_i ∉ DV^i_{P(a_i)}(Context_i(P(a_i))) and d_i is not
          pruned by bounds then
7           COMPUTE_DV(d_i)
Function COMPUTE_DV(d_i):
8   D_i' ← D_i \ (DV_i ∪ {d_i})
9   compute ΔU_i(d_i', d_i) by Eq. (14), ∀d_i' ∈ D_i'
10  if ∃d_i' ∈ D_i' s.t. ΔU_i(d_i', d_i) ≤ 0 (with tie-breaking)
      then
11      DV_i ← DV_i ∪ {d_i}
12      return
13  foreach a_c ∈ C(a_i) do
14      foreach d_c ∈ D_c do
15          compute ΔU_i^c(d_i', d_i, d_c) by Eq. (15), ∀d_i' ∈ D_i'
16          if ∃d_i' ∈ D_i' s.t. ΔU_i^c(d_i', d_i, d_c) ≤ 0 (with
              tie-breaking) then
17              DV_i^c(d_i) ← DV_i^c(d_i) ∪ {d_c}
18      if DV_i^c(d_i) = D_c then
19          DV_i ← DV_i ∪ {d_i}
20          return
```

Complexity Analysis. In the precomputation phase, each agent a_i constructs two types of static tables. First, it builds the aggregated difference table $\overline{\Delta cd\_cost}_i$ of size $O(d_{\max}^2)$; each entry $\overline{\Delta f_{ic}}(d_i', d_i)$ for $a_c \in CD(a_i)$ requires maximizing over $d_{\max}$ values of d_c, so constructing $\overline{\Delta cd\_cost}_i$ takes $O(|CD(a_i)| \cdot d_{\max}^3)$. Second, for each child $a_c \in C(a_i)$, the agent builds a conditional difference table Δf_{ic} of size $O(d_{\max}^3)$, filled in $O(d_{\max}^3)$ time via Eq. (16). Hence the total precomputation per agent is $O((|CD(a_i)| + |C(a_i)|) \cdot d_{\max}^3)$.

In the runtime phase, each agent a_i computes DV_i and DV_i^c. During context processing, computing DV_i involves examining all $O(d_{\max}^2)$ value pairs via lookups in $\overline{\Delta U_i}$. For each child, computing DV_i^c requires inspecting the $O(d_{\max}^3)$ entries of $\overline{\Delta U_i^c}$. The overall runtime complexity per agent per context changes is thus $O\big(d_{\max}^2 + |C(a_i)| \cdot d_{\max}^3\big)$.

Memory consumption is dominated by storing $\overline{\Delta cd\_cost}_i$ (size $O(d_{\max}^2)$) and, for each child, Δf_{ic} (size $O(d_{\max}^3)$), leading to total storage $O(d_{\max}^2 + |C(a_i)| \cdot d_{\max}^3)$.

Discussion. While BIP provides a theoretically powerful pruning mechanism, its exponential complexity in $|CD(a_i)| + 1$ renders it impractical for agents with many children. In practice, BIP is often restricted to agents where $|CD(a_i)| + 1 \le k$, limiting its applicability to a small fraction of the search tree. To overcome this fundamental barrier, DR-BIP introduces a fully decoupled and sound reformulation that eliminates the exponential bottleneck entirely. Proposition 1 ensures that DR-BIP's pruning remains sound with respect to BIP. As a result, DR-BIP can be applied to all agents regardless of $|CD(a_i)|$, enabling systematic and scal-

able pruning across the entire pseudo-tree.

5 Empirical Evaluation

In this section, we first describe the experimental configuration. Then, we compare the DR-BIP-based algorithms with their originals, the corresponding BIP-enhanced versions, and RMB-DPOP [Chen *et al.*, 2020b] on benchmark problems.

5.1 Experimental Configuration

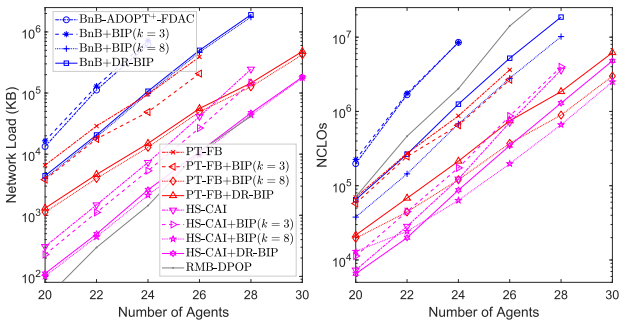
In order to demonstrate its effect on distributed search, DR-BIP is integrated into three state-of-the-art tree-based complete search algorithms: BnB-ADOPT$^+$-FDAC, PT-FB, and HS-CAI. Their DR-BIP-enhanced versions are respectively denoted as BnB+DR-BIP, PT-FB+DR-BIP, and HS-CAI+DR-BIP. We compare these DR-BIP-based algorithms with their original counterparts, the corresponding BIP-enhanced versions (i.e., BnB+BIP, PT-FB+BIP, and HS-CAI+BIP), as well as RMB-DPOP—the latest best-performing algorithm in the DPOP family—on two standard benchmark classes: random DCOPs and scale-free networks. We consider four problem configurations. The first two are random DCOPs with a fixed domain size of 3: a sparse setting with graph density 0.2 and the number of agents ranging from 20 to 30, and a dense setting with density 0.5 and the number of agents from 14 to 24. The third configuration also uses random DCOPs, with 22 agents, density 0.2, and domain sizes varying from 2 to 7. The fourth configuration is based on scale-free networks generated via BA model [Barabási and Albert, 1999], with 26 agents, domain size 3, initial clique size $m_0 = 10$, and attachment parameter m_1 ranging from 3 to 8. For each configuration, all constraint costs are integers uniformly sampled from $[0, 100]$.

To measure performance, we adopt two metrics: network load (NL), defined as the total volume of information exchanged among agents, and non-concurrent logical operations (NCLOs) [Netzer *et al.*, 2012], which capture hardware-independent runtime by counting logical accesses to utility and constraint checks. For DR-BIP–enhanced algorithms, the NCLOs counts encompass both the runtime accesses to $\overline{\Delta U_i}$ and $\overline{\Delta U_i^c}$ and the one-time precomputation of $\overline{\Delta cd\_cost_i}$ and Δf_{ic}. For each configuration, we generate 50 random instances and report the average over all successful runs. We set the memory-budget parameter $k = 3$ for HS-CAI, RMB-DPOP, and BIP to match DR-BIP's memory usage for a fair comparison. We also consider BIP with $k = 8$ to compare DR-BIP against BIP under abundant memory. All experiments ran on an i7-7820X workstation with 32 GB of RAM. We enforce a 30-min timeout per algorithm instance.

5.2 Experimental Results

Figure 3 illustrates the performance under varying numbers of agents in the sparse configuration, with the corresponding normalized metrics (NL and NCLOs) presented as percentages relative to the original algorithms in the first two columns of Table 1. Values below 100%, indicating improved performance, are highlighted in bold. Under BIP with $k = 3$, a clear trend emerges: while the BIP-enhanced algorithms show only marginal improvement or even degradation in some metrics compared to their originals, the DR-BIP-enhanced versions consistently and substantially outperform both the original and BIP-enhanced baselines. For instance, the normalized NL and NCLOs of BnB+BIP($k = 3$) remain near 100%, whereas BnB+DR-BIP achieves only 15%–33% of the original NL and NCLOs. This pattern holds across PT-FB and HS-CAI, confirming that DR-BIP provides significantly stronger pruning with lower computational overhead than BIP. Furthermore, DR-BIP significantly enhances the scalability of the original algorithms. For instance, while BnB+BIP($k = 3$) is limited to problems with at most 24 agents, the DR-BIP-enhanced version of this algorithm scales to 28 agents. Similar scalability gains are observed for PT-FB+DR-BIP and HS-CAI+DR-BIP. When BIP is given a larger memory budget ($k = 8$), its scalability and NL reduction become comparable to those of DR-BIP, and it achieves even lower NCLOs than DR-BIP because more memory is available for caching reusable pruning sets, thereby avoiding redundant computation. Although RMB-DPOP maintains an advantage in NL over most search-based algorithms, it is outperformed in terms of NCLOs by almost all others.

Figure 4 presents the results on dense configurations, with the corresponding normalized metrics shown in the third and fourth columns of Table 1. The BIP-enhanced variants with $k = 3$ exhibit only marginal improvements over their orig-



(a) Network Load (b) NCLOs

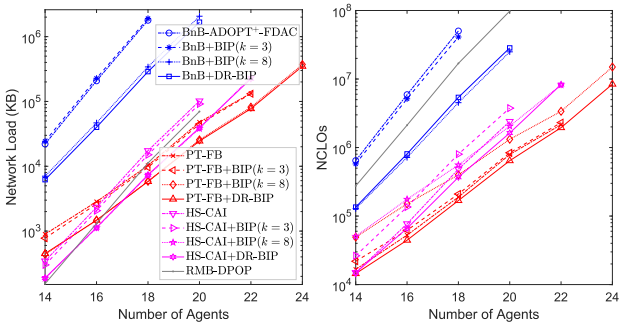
Figure 3: Performance comparison under different numbers of agents on the sparse configuration.

Algorithm	Sparse DCOPs		Dense DCOPs		Varying Domain Size		Scale − Free Networks	
	NL (%)	NCLOs (%)	NL (%)	NCLOs (%)	NL (%)	NCLOs (%)	NL (%)[2]	NCLOs (%)[2]
BnB + BIP ($k = 3$)	116 ~ 124	**99 ~ 114**	107 ~ 110	**82 ~ 90**	116 ~ 121	101 ~ 105	119	**98**
BnB + BIP ($k = 8$)	**16 ~ 30**	**8 ~ 19**	**19 ~ 32**	**10 ~ 20**	**17 ~ 29**	**9 ~ 18**	**20**	**10**
BnB + DR - BIP	**15 ~ 33**	**15 ~ 33**	**16 ~ 28**	**11 ~ 21**	**18 ~ 27**	**16 ~ 28**	**23**	**20**
PT-FB + BIP ($k = 3$)	**51 ~ 62**	**73 ~ 91**	**87 ~ 99**	106 ~ 132	**51 ~ 62**	**73** ~ 104	**36 ~ 75**	**49 ~ 98**
PT-FB + BIP ($k = 8$)	**13 ~ 17**	**10 ~ 28**	**49 ~ 61**	157 ~ 296	**14 ~ 19**	**16 ~ 22**	**10 ~ 22**	**10 ~ 24**
PT-FB + DR - BIP	**14 ~ 20**	**21 ~ 31**	**51 ~ 58**	**81 ~ 90**	**14 ~ 24**	**20 ~ 40**	**10 ~ 26**	**15 ~ 39**
HS - CAI + BIP ($k = 3$)	**60 ~ 75**	111 ~ 158	**86 ~ 90**	156 ~ 177	**59 ~ 90**	121 ~ 170	**65 ~ 74**	130 ~ 137
HS - CAI + BIP ($k = 8$)	**18 ~ 33**	18 ~ 179	**37 ~ 54**	85 ~ 339	**30 ~ 48**	**85** ~ 270	**26 ~ 34**	**30 ~ 76**
HS - CAI + DR - BIP	**19 ~ 36**	**36 ~ 89**	**39 ~ 55**	**68 ~ 99**	**33 ~ 58**	**70** ~ 158	**25 ~ 35**	**46 ~ 67**

Table 1: Normalized performance of BIP-based and DR-BIP-based algorithms relative to their originals[3].

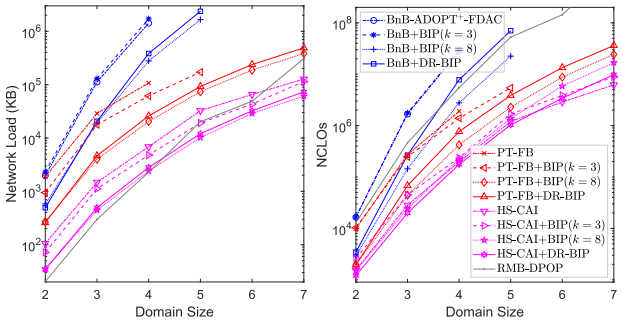
[2]BnB+BIP and BnB+DR-BIP yield only one normalized point, as the original succeeds solely on the smallest instance with $m_1 = 3$.

[3]All values are normalized against the performance of the corresponding original (unenhanced) algorithms, lower is better.



(a) Network Load (b) NCLOs

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

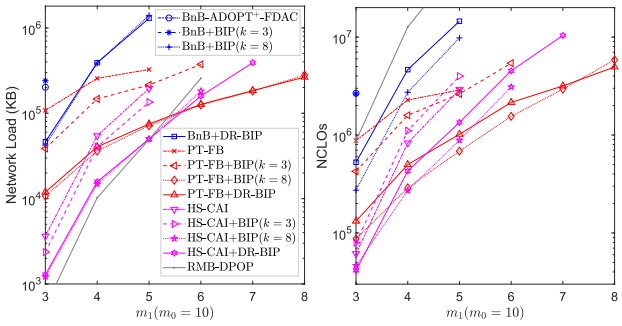


(a) Network Load (b) NCLOs

Figure 5: Performance comparison under different domain sizes.

inal counterparts, and in several cases even yield increased NL and NCLOs. In contrast, the DR-BIP-based algorithms maintain a clear advantage, consistently outperforming both the original algorithms and the BIP-enhanced versions across all metrics. However, the performance gaps between DR-BIP and the baselines are narrower than those in sparse settings. This is because denser graphs increase the dimensionality of both coarse and finer subspaces (especially for agents near the root of the pseudo-tree), making the dominance condition stricter and thereby reducing the number of prunable values. BIP-enhanced algorithms with $k = 8$ achieve NL reduction comparable to their DR-BIP counterparts, but for most algorithms, their NCLOs become substantially higher than those of DR-BIP due to exponential growth in computational cost.

Figure 5 illustrates performance under varying domain sizes. The normalized metrics, given in the fifth and sixth columns of table 1, indicate that DR-BIP remains effective even as domain size grows, though its relative improvement narrows. This reduction occurs because larger domains enlarge both coarse and finer subspaces, which tightens the dominance condition and reduces the number of prunable values. Notably, the BIP-enhanced algorithms ($k = 3$) struggle with scalability: for example, PT-FB+BIP fails on domains larger than 5, while PT-FB+DR-BIP scales up to domain size 7. Besides, DR-BIP achieves scalability comparable to BIP with $k = 8$, but with far less memory.



(a) Network Load (b) NCLOs

Figure 6: Performance comparison on scale-free networks.

Finally, Fig. 6 presents results on scale-free networks. The corresponding normalized metrics, shown in the seventh and eighth columns of table 1, follow the same trend observed on random DCOPs: the DR-BIP-based algorithms significantly outperform both their originals and the BIP-enhanced versions with $k = 3$ across all metrics, and their NL reduction is comparable to that of BIP-enhanced versions with $k = 8$. Moreover, all the DR-BIP-enhanced algorithms scale to larger m_1 values than their BIP-enhanced counterparts with $k = 3$; for HS-CAI, this advantage persists even when BIP is allowed $k = 8$. These results further evidence the scalability benefits of the decoupled reformulation.

Summary. The experimental results consistently show that BIP with $k = 3$ provides limited and sometimes inconsistent improvement over the original algorithms, while DR-BIP delivers substantial, robust gains in both efficiency and scalability across diverse benchmark settings. Moreover, even when BIP is granted a much larger memory budget ($k = 8$), DR-BIP remains highly competitive: it achieves comparable pruning effectiveness (NL) across all problems, while its NCLOs are often lower than BIP's in dense problems. This confirms that DR-BIP retains BIP's core pruning power without its exponential overhead.

6 Conclusion

Building on the foundational principles of bound-independent pruning (BIP) in distributed constraint optimization, we propose DR-BIP, a decoupled reformulation that eliminates the exponential computational and storage complexity barrier of BIP. By decoupling dominated-subspace detection into independent and reusable dominance tests, DR-BIP achieves polynomial complexity while delivering sound and scalable pruning. Our approach is both formally proven and empirically demonstrated to significantly outperform BIP. Beyond DCOPs, the decoupled design of DR-BIP provides a more efficient approach to dominance pruning that can be adapted to other optimization settings where dominance-based reasoning is applicable.

In the future, we will explore extending dominance pruning to dynamic and uncertain DCOP settings, as well as integrating it with learning-based methods to further enhance pruning adaptability and efficiency.

References

[Atlas *et al.*, 2008] 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.

[Bacchus *et al.*, 2002] Fahiem Bacchus, Xinguang Chen, Peter Van Beek, and Toby Walsh. Binary vs. non-binary constraints. *Artificial Intelligence*, 140(1-2):1–37, 2002.

[Barabási and Albert, 1999] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

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

[Chen *et al.*, 2020a] 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.

[Chen *et al.*, 2020b] Ziyu Chen, Wenxin Zhang, Yanchen Deng, Dingding Chen, and Qing Li. RMB-DPOP: Refining MB-DPOP by reducing redundant inferences. pages 249–257, 2020.

[Cohen *et al.*, 2020] Liel Cohen, Rotem Galiki, and Roie Zivan. Governing convergence of max-sum on dcops through damping and splitting. *Artificial Intelligence*, 279:103212, 2020.

[Deng *et al.*, 2019] 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.

[Deng *et al.*, 2025] Yanchen Deng, Xinrun Wang, and Bo An. Gdba revisited: Unleashing the power of guided local search for distributed constraint optimization. *arXiv preprint arXiv:2508.06899*, 2025.

[Farinelli *et al.*, 2008] 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.

[Farinelli *et al.*, 2014] 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.

[Fioretto *et al.*, 2017] 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.

[Fioretto *et al.*, 2018] Ferdinando Fioretto, Enrico Pontelli, and William Yeoh. Distributed constraint optimization problems and applications: A survey. *Journal of Artificial Intelligence Research*, 61:623–698, 2018.

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

[Hirayama and Yokoo, 1997] Katsutoshi Hirayama and Makoto Yokoo. Distributed partial constraint satisfaction problem. In *International Conference on Principles and Practice of Constraint Programming*, pages 222–236, 1997.

[Kim and Lesser, 2014] 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.

[Litov and Meisels, 2017] Omer Litov and Amnon Meisels. Forward bounding on pseudo-trees for DCOPs and ADCOPs. *Artificial Intelligence*, 252:83–99, 2017.

[Liu *et al.*, 2021] Xiangshuang Liu, Ziyu Chen, Dingding Chen, and Junsong Gao. A bound-independent pruning technique to speeding up tree-based complete search algorithms for distributed constraint optimization problems. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, pages 41–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2021.

[Maheswaran *et al.*, 2006] 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. 2006.

[Modi *et al.*, 2005] 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.

[Netzer *et al.*, 2012] Arnon Netzer, Alon Grubshtein, and Amnon Meisels. Concurrent forward bounding for distributed constraint optimization problems. *Artificial Intelligence*, 193:186–216, 2012.

[Nguyen *et al.*, 2019] 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.

[Ottens *et al.*, 2017] 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.

[Pertzovsky *et al.*, 2024] Arseniy Pertzovsky, Roie Zivan, and Noa Agmon. Collision avoiding max-sum for mobile sensor teams. *Journal of Artificial Intelligence Research*, 79:1281–1311, 2024.

[Petcu and Faltings, 2005] Adrian Petcu and Boi Faltings. Approximations in distributed optimization. In *International Conference on Principles and Practice of Constraint Programming*, pages 802–806, 2005.

[Petcu and Faltings, 2006] Adrian Petcu and Boi Faltings. ODPOP: An algorithm for open/distributed constraint optimization. pages 703–708, 2006.

[Petcu and Faltings, 2007] 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.

[Vinyals *et al.*, 2009] 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.