Autonomous Agents and Multi-Agent Systems (2020) 34:50
https://doi.org/10.1007/s10458-020-09476-5

A hybrid tree-based algorithm to solve asymmetric distributed constraint optimization problems

Dingding Chen[1] · Yanchen Deng[2] · Ziyu Chen[1] · Zhongshi He[1] · Wenxin Zhang[1]

Published online: 21 July 2020

Abstract
Asymmetric distributed constraint optimization problems (ADCOPs) have emerged as an important formalism in multi-agent community due to their ability to capture personal preferences. However, the existing search-based complete algorithms for ADCOPs only exploit local knowledge to calculate lower bounds, which leads to inefficient pruning and prohibits them from solving large scale problems. On the other hand, inference-based complete algorithms (e.g., DPOP) for distributed constraint optimization problems are able to aggregate the global cost promptly but cannot be directly applied into ADCOPs due to a privacy concern. Thus, in this paper, we investigate the possibility of combining inference and search to effectively solve ADCOPs at an acceptable loss of privacy. Specifically, we propose a hybrid complete ADCOP algorithm called PT-ISABB which uses a tailored inference algorithm to provide tight lower bounds and upper bounds, and a tree-based complete search algorithm to guarantee the optimality. Furthermore, we introduce two suboptimal variants of PT-ISABB based on bounded-error approximation mechanisms to enable trade-off between theoretically guaranteed solutions and coordination overheads. We prove the correctness of PT-ISABB and its suboptimal variants. Finally, the experimental results demonstrate that PT-ISABB exhibits great superiorities over other state-of-the-art search-based complete algorithms and its suboptimal variants can quickly find a solution within the user-specified bounded-error.

Keywords DCOP · ADCOP · Complete ADCOP algorithm · Search · Inference

Dingding Chen and Yanchen Deng have contributed equally to this work.

✉ Ziyu Chen
chenziyu@cqu.edu.cn

✉ Zhongshi He
zshe@cqu.edu.cn

[1] College of Computer Science, Chongqing University, Chongqing 400044, China

[2] School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798, Singapore

1 Introduction

Distributed constraint optimization problems (DCOPs) [10, 19, 41] are an elegant paradigm for modeling multi-agent system (MAS) where agents coordinate with each other to optimize a global objective. They have been successfully applied into various MAS applications including smart home [11], radio frequency allocation [27], task scheduling [18, 38] and many others.

Over the past decades, a wide variety of algorithms have been developed to solve DCOPs and can be divided into either complete or incomplete ones. Incomplete algorithms aim to produce near-optimal solutions at small computational efforts and generally follow three strategies, i.e., local search [24, 42], inference [5, 9, 37, 44] and sampling [29, 31]. On the contrary, complete algorithms guarantee to find the optimal solution and can be broadly classified into inference-based and search-based algorithms. Search-based complete algorithms [4, 13, 19, 23, 26, 28, 40] use distributed backtrack search to exhaust the search space, generally employing either a best-first or a depth-first branch-and-bound search strategy. These algorithms have a linear message size but an exponential number of messages. Different from search-based complete algorithms, inference-based complete algorithms [33, 39] use a dynamic programming to solve DCOPs. They require a linear number of messages, but the message size is exponential in the induced width [7, 35].

However, DCOPs fail to capture ubiquitous asymmetric structures in real world scenarios [3, 25, 36] since each constrained agent shares the same cost. Asymmetric distributed constraint optimization problems (ADCOPs) [15] are a notable extension to DCOPs, which captures asymmetry by explicitly defining the exact cost for each participant of a constraint and has been intensively investigated in recent years.

Solving ADCOPs is more challenging since algorithms must evaluate and aggregate the cost for each participant of a constraint. Complete algorithms for ADCOPs [15, 23] are nearly the variants of synchronous search-based complete DCOP solvers in consideration of the aggregation for the costs of each side. More specifically, they perform a depth-first branch-and-bound in virtue of either a two-phase strategy or a one-phase strategy [2] to accumulate the each side constraint cost at the expense of privacy. As known, the performance of the branch-and-bound search-based algorithms is heavily dependent on the tightness of bounds established according to the knowledge of each agent. Unfortunately, the existing search-based complete algorithms for ADCOPs only use local knowledge to construct the lower bounds, which leads to inefficient pruning and limits the scale of problems they can solve. On the other side, inference-based complete algorithms for DCOPs only require a linear number of messages to aggregate the global cost, but they cannot be directly applied to solve ADCOPs. That is partially because these algorithms require the total knowledge of each constraint in order to perform variable elimination optimally. In other words, parent and pseudo parents must transfer their private constraints to their children to perform variable eliminations, which is unacceptable in an asymmetric scenario.

In this paper, we consider the possibility of combining both inference and search to efficiently solve ADCOPs at an acceptable loss of privacy. Specifically, our main contributions are listed as follows.

- We propose a hybrid tree-based complete algorithm for ADCOPs, called PT-ISABB. The algorithm first establishes tight lower bounds and upper bounds by using a tailored version of ADPOP [32] (i.e., an approximated version of DPOP which is an inference-based complete algorithm for DCOPs) which can be performed at an acceptable loss of

privacy. Then, a variant of SyncABB-1ph [15] is implemented on a pseudo tree to find the optimal solution, where an estimation reporting mechanism is introduced to compute complete upper bounds and avoid directly disclosing the private constraint costs.
- We propose two suboptimal variants of PT-ISABB by using absolute error mechanism [26] and relative error mechanism [40] to allow users to specify an absolute error bound and a relative error bound, respectively. For adapting Absolute Error Mechanism on PT-ISABB, an absolute error bound allocation mechanism is proposed to allocate the user-specified absolute error bound for each subproblem.
- We theoretically show the correctness of our proposed algorithms including PT-ISABB and its two suboptimal variants. Moreover, we prove that the lower bounds in PT-ISABB are at least as tight as the ones in AsymPT-FB when its maximal dimension has no limit. Finally, our empirical evaluation results confirm the superiorities of our proposed algorithms.

The rest of the paper is organized as follows. In Sect. 2, we briefly review related work. Section 3 gives the preliminaries including DCOPs, ADCOPs, pseudo tree, DPOP and ADPOP. In Sect. 4, we describe the proposed hybrid complete algorithm for ADCOPs named PT-ISABB. The theoretical analysis of PT-ISABB can be found in Sect. 5. In Sect. 6, we introduce two suboptimal variants of PT-ISABB. Lastly, we present the empirical evaluation to our proposed methods in Sect. 7 and conclude the paper in Sect. 8.

2 Related work

Incomplete algorithms for DCOPs can be roughly categorized into local search, inference-based, and sampling-based algorithms. In local search algorithms such as DBA[20], DSA [42], MGM [24] and GDBA [30], agents exchange their own states with neighbors and optimize individual benefits in terms of the latest states of their neighbors. Inference-based incomplete algorithms like Max-sum [9] and its variants [5, 37, 44] employ belief propagation on the factor-graph [21] to gather the global information. Sampling-based algorithms including DUCT [31] and D-Gibbs [29] are the recent emerging incomplete algorithms which perform sampling on a pseudo tree according to the confidence bounds or the statistical inference, respectively.

Complete algorithms for DCOPs employ either systematic search or inference. SyncBB [19] is an early search-based algorithm, which performs a depth-first branch-and-bound search on a chain-based structure. AFB [13] was proposed to improve SyncBB for achieving better concurrent computation, where all agents execute the forward bounding concurrently and asynchronously to establish tight lower bound. ConcFB [28] came out to further enhance parallelism by performing multiple parallel versions of AFB concurrently. However, a chain-based structure could force unconstrained agents to communicate with each other and disallow parallel exploration of the search space. Alternatively, a pseudo tree [12] was proposed to create communication links among agents that share constraints and parallelize the computation in different branches. ADOPT [26] is a classical asynchronous algorithm operating on a pseudo tree, which uses a best-first search strategy. However, its search strategy incurs unnecessary reconstruction of abandoned solutions. Subsequently, BnB-ADOPT [40] was proposed to solve the issue by adopting a depth-first branch-and-bound search strategy. Different from BnB-ADOPT, NCBB [4] and PT-FB [23] are the synchronous algorithms that employ a depth-first branch-and-bound search on a pseudo

tree. For improving the lower bound tightness, NCBB adopts the eager propagation of lower bounds on the solution cost while PT-FB uses forward bounding on the independent branches of a pseudo tree in parallel.

DPOP is a representative inference-based complete algorithm for DCOPs, which performs dynamic programming on a pseudo tree. Specifically, agents in DPOP forward the assignment combination utilities to their parents and broadcast their optimal decisions to their children along the pseudo tree. However, its maximal message size is exponential in the induced width [35] of the pseudo tree. Accordingly, ODPOP [34] and MB-DPOP [35] were proposed to trade the number of messages for the maximal message size. Besides, Action_GDL [39] was proposed to generalize DPOP by performing dynamic programming on a distributed junction tree to enhance its efficiency.

Most of the algorithms for ADCOPs adapt the above DCOP algorithms to handle asymmetric constraint costs. Local search algorithms for ADCOPs like ACLS, MCS-MGM and GCA-MGM [15] are the asymmetric extensions of DSA and MGM, where each agent exchanges its state and its side constraint cost to its neighbour involved in the same constraint. Additionally, Zivan et al. [45] proposed to apply Max-sum and its variants to solve ADCOPs on asymmetric factor-graphs.

Search-based complete algorithms for ADCOPs adopt either a two-phase strategy or a one-phase strategy to aggregate constraint costs of each side. More specifically, the algorithms with a two-phase strategy consider only one-side constraint costs in the first phase and gather the other side constraint costs in the second phase once a complete assignment is reached, while the algorithms with a one-phase strategy systematically check each side of the constraints before reaching a full assignment. SyncABB-2ph [15] is the asymmetric adaption of SyncBB with a two-phase strategy. Instead, SyncABB-1ph and ATWB [15] are the asymmetric versions of SyncBB and AFB based on a one-phase strategy, respectively. To achieve a one-phase check, SyncABB-1ph uses a sequence of back checking processes while ATWB performs backward bounding. As the asymmetric adaptation of PT-FB, AsymPT-FB [23] is the first tree-based algorithm for ADCOPs with a one-phase strategy. Just like ATWB, AsymPT-FB executes forward bounding to facilitate the lower bound accumulation by forwarding copies of current partial assignment (*Cpa*) to their constrained descendants, and back bounding to inquire the constraint costs on the other side by sending copies of *Cpa* backwards to their constrained ancestors in the pseudo tree.

3 Background

In this section, we introduce the preliminaries including DCOPs, ADCOPs, pseudo tree, DPOP and ADPOP.

3.1 Distributed constraint optimization problems

A distributed constraint optimization problem [26] can be defined by a tuple $\langle A, X, D, F\rangle$ such that:

- $A = \{a_1, \ldots, a_q\}$ is a set of agents.
- $X = \{x_1, \ldots, x_n\}$ is a set of variables. Each variable x_i is only controlled by an agent.
- $D = \{D_1, \ldots, D_n\}$ is a set of finite variable domains. Each domain $D_i \in D$ consists of a set of finite allowable values for variable $x_i \in X$.

- $F = \{f_1, \ldots, f_m\}$ is a set of constraint functions. Each function $f_i : D_{i_1} \times \cdots \times D_{i_k} \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$ specifies a non-negative costs to each value combination of the involved variables $x_{i_1}, \ldots, x_{i_k}$. Here, the constraints of an infinite cost are so called hard constraints which represent the combinations of assignments that are strictly forbidden, and the constraints of a finite cost are called soft constraints.

To facilitate understanding, we assume that each agent only controls a single variable (i.e., $q = n$) and all constraints are binary (i.e., $f_{ij} : D_i \times D_j \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$). Thus, the term *agent* and *variable* can be used interchangeably. These assumptions are commonly used in the DCOP literature [23, 26, 33, 40]. A solution to a DCOP is an assignment to all variables such that the total cost is minimized. That is,

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

3.2 Asymmetric distributed constraint optimization problems

While DCOPs assume an equal cost for each participant of each constraint, asymmetric distributed constraint optimization problems (ADCOPs) [15] explicitly define the exact cost for each constrained agent. In other words, a constraint function $f_i : D_{i_1} \times \cdots \times D_{i_k} \rightarrow \prod_{j=1}^{k} (\mathbb{R}_{\geq 0} \cup \{\infty\})$ in an ADCOP specifies a cost vector for each possible combination of involved variables. Following the assumption in DCOPs, we also assume that each agent only controls a single variable and all the constraints are binary in ADCOPs. For a binary constraint between x_i and x_j, we denote the private cost functions for x_i and x_j as f_{ij} and f_{ji}, respectively. Note that in the asymmetric setting, f_{ij} does not necessarily equal f_{ji}. And the goal of solving an ADCOP is to find a solution which minimizes the aggregated constraint costs of each side. That is,

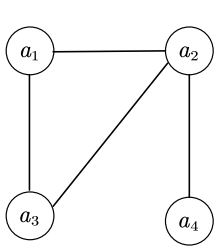
$$X^* = \underset{d_i \in D_i, d_j \in D_j}{\arg\min} \sum_{f_{ij}, f_{ji} \in F} f_{ij}(x_i = d_i, x_j = d_j) + f_{ji}(x_j = d_j, x_i = d_i).$$

An ADCOP can be visualized by a constraint graph where a node represents an agent and an edge represents a binary constraint. Figure 1a gives the constraint graph of an ADCOP with four agents and four binary constraints whose private cost functions can be found in Fig. 1c. In the example, each agent has a single variable with domain $\{0, 1\}$, and the optimal solution is $\{(x_1, 0), (x_2, 0), (x_3, 0), (x_4, 0)\}$ with the cost of 25.

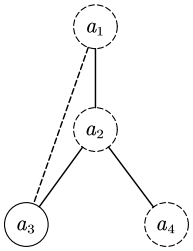
3.3 Pseudo tree

Definition 1 (*Pseudo tree* [12]) A pseudo tree arrangement of a constraint graph is a tree with the same nodes and edges as the original graph, and with the property that adjacent nodes from the original graph fall in the same branch of the tree.

Therefore, the search process can be performed on the independent branches in a pseudo tree in parallel. A pseudo tree can be generated by a depth-first traversal of a constraint graph, which categorizes the constraints into tree edges and pseudo edges (i.e., non-tree edges). Figure 1b presents a possible pseudo tree deriving from Fig. 1a where tree edges and pseudo edges are shown as solid lines and dashed lines, respectively.



(a) Constraint graph



(b) An example of pseudo tree

x_1 \ x_3	0	1
0	2	7
1	6	9

x_3 \ x_1	0	1
0	4	5
1	8	3

x_2 \ x_3	0	1
0	5	1
1	3	5

x_3 \ x_2	0	1
0	4	5
1	3	2

x_1 \ x_2	0	1
0	1	1
1	4	7

x_2 \ x_1	0	1
0	4	5
1	2	8

x_4 \ x_2	0	1
0	3	7
1	5	9

x_2 \ x_4	0	1
0	2	3
1	1	2

(c) Constraint functions

Fig. 1 An ADCOP instance

For an agent a_i, its neighbors can be categorized into its parent $P(a_i)$, pseudo parent $PP(a_i)$, children $C(a_i)$ and pseudo children $PC(a_i)$ according to their positions in the pseudo tree and the type of edges they connect through. More precisely, they can be formally defined as follows:

- $P(a_i)$ is the ancestor connecting with a_i through a tree edge (e.g., $P(a_2) = a_1$ in Fig. 1b).
- $PP(a_i)$ is the set of ancestors that connect with a_i through pseudo edges (e.g., $PP(a_3) = \{a_1\}$ in Fig. 1b).
- $C(a_i)$ is the set of descendants connecting with a_i through tree edges (e.g., $C(a_2) = \{a_3, a_4\}$ in Fig. 1b).
- $PC(a_i)$ is the set of the descendants that connect with a_i through pseudo edges (e.g., $PC(a_1) = \{a_3\}$ in Fig. 1b).

For succinctness, we also adopt the following notations.

- $AP(a_i)$ is the set of a_i's ancestors connecting with a_i, i.e., $AP(a_i) = PP(a_i) \cup \{P(a_i)\}$ (e.g., $AP(a_3) = \{a_1, a_2\}$ in Fig. 1b).
- $AC(a_i)$ is the set of a_i's descendants connecting with a_i, i.e., $AC(a_i) = PC(a_i) \cup C(a_i)$ (e.g., $AC(a_1) = \{a_2, a_3\}$ in Fig. 1b).
- $Desc(a_i)$ is the set of descendants of a_i (e.g., $Desc(a_1) = \{a_2, a_3, a_4\}$ in Fig. 1b).
- $Sep(a_i)$[34] is the separator set of a_i, comprising the ancestors that are constrained with agents in $\{a_i\} \cup Desc(a_i)$ (e.g., $Sep(a_3) = \{a_1, a_2\}$ in Fig. 1b).

Additionally, we define w^*, the induce width of a pseudo tree, as the size of the largest separator set of any node in the pseudo tree, i.e., $w^* = \max_{a_i \in A} |Sep(a_i)|$.

3.4 DPOP and ADPOP

Before diving into the details of DPOP and ADPOP, let's first consider the following definitions.

Definition 2 (*dims*) Let U be a function, $dims(U)$ is the dimensions of U.

Definition 3 (*Slice*) Let P be a set of key-value pairs. $P_{[K]}$ is a slice of P over K such that:

$$P_{[K]} = \{(k, v) | (k, v) \in P, \forall k \in K\}. \tag{1}$$

Definition 4 (*Join* [32]) Let U, U' be two cost tables and $D_U = \Pi_{x_i \in dims(U)} D_i, D_{U'} = \Pi_{x_i \in dims(U')} D_i$ be their domain spaces. $U \otimes U'$ is the join of U and U' over $D_{U \otimes U'} = \Pi_{x_i \in dims(U) \cup dims(U')} D_i$ such that:

$$(U \otimes U')(p) = U(p_{[dims(U)]}) + U'(p_{[dims(U')]}), \forall p \in \left\{(dims(U \otimes U'), V) | \forall V \in D_{U \otimes U'}\right\}. \tag{2}$$

Definition 5 (*Projections* [32]) Let U be a cost table and S be a subset of the dimensions to U (i.e., $S \subset dims(U)$). $U\perp_S^+$ and $U\perp_S^-$ are the maximal and minimal projections of U along the S axis respectively, which can be defined by:

$$(U\perp_S^+)(p) = \max_{p_s \in P_{[S]}} U(p \cup p_s), \forall p \in P_{[dims(U) \setminus S]} \tag{3}$$

$$(U\perp_S^+)(p) = \min_{p_s \in P_{[S]}} U(p \cup p_s), \forall p \in P_{[dims(U) \setminus S]} \tag{4}$$

where $P = \{(dims(U), V) | \forall V \in \Pi_{x_i \in dims(U)} D_i\}$. For simplicity, we denote $U\perp_S$ as the abbreviation for $U\perp_S^+$ and $U\perp_S^-$. That is,

$$U\perp_S = \{U\perp_S^-, U\perp_S^+\} \tag{5}$$

Similarly, we also denote $U^{\pm} = U'^{\pm} \otimes U''^{\pm}$ as the abbreviation for $U^+ = U'^+ \otimes U''^+$ and $U^- = U'^- \otimes U''^-$.

DPOP[1] [33] is an important inference-based complete algorithm for DCOPs, which performs a distributed bucket elimination scheme [8] on a pseudo tree. DPOP comprises the following phases.

- *Pseudo tree construction phase* executes a distributed depth-first traversal on the constraint graph of a DCOP to produce a pseudo tree. After this phase, each agent a_i knows its parent $P(a_i)$, pseudo parents $PP(a_i)$, children $C(a_i)$ and pseudo children $PC(a_i)$.
- *COST propagation phase* performs variable eliminations and cost propagation from leaf agents to the root agent along the pseudo tree. In this phase, each agent a_i joins the received cost tables from its children with its local cost table, eliminates its variable from the joint cost table by calculating the optimal value for each assignment combination of its separator. That is,

[1] We describe DPOP as a minimization version to cope with the objective of DCOPs.

$$util_i = \left(\left(\bigotimes_{a_c \in C(a_i)} util_i^c \right) \otimes local\_util_i \right) \perp_{x_i}^{-} \tag{6}$$

where $util_i^c$ is the cost table received from its child $a_c \in C(a_i)$ (i.e., $util_i^c = util_c$) and $local\_util_i$ is a_i's local cost table combining the constraints between a_i and $AP(a_i)$. That is,

$$local\_util_i = \bigotimes_{a_j \in AP(a_i)} f_{ij} \tag{7}$$

and then propagates $util_i$ to its parent $P(a_i)$ via a COST message.

- *VALUE propagation phase* implements to assign the optimal value for each variable vice versa along the pseudo tree. In the phase, each agent a_i selects its optimal assignment d_i^* based on the cost tables received during the COST propagation phase and the optimal assignment PA_i^* received from its parent. That is,

$$d_i^* = \arg\min_{d_i \in D_i} \sum_{a_c \in C(a_i)} util_i^c(PA_i^*, d_i) + local\_util_i(PA_i^*, d_i) \tag{8}$$

extends PA_i^* with its optimal assignment (i.e., $PA_i^* \cup \{(x_i, d_i^*)\}$) and broadcasts the extended assignment to its children via VALUE messages. It is noteworthy that the assignment sent to its child $a_c \in C(a_i)$ is $PA^*_{i[Sep(a_c)]} \cup \{(x_i, d_i^*)\}$ where $Sep(a_c) = dims(util_i^c)$.

Although DPOP only requires a linear number of messages to solve a DCOP, its maximal message size is exponential in the induced width of the pseudo tree, which prohibits it from solving more complex problems. Thus, ADPOP[32] was proposed to allow the desired trade-off between solution quality and the maximal message size. Specifically, ADPOP imposes a limit k on the maximum number of dimensions in each COST message.[2] When the dimension size of its outgoing COST message exceeds the limit, a_i drops a set of dimensions S to stay below the limit. That is, it computes the upper bound and lower bound of the original cost table (i.e., the maximal cost table and minimal cost table) by eliminating $\{x_i\} \cup S$ according to Eq. (9).

$$util_i^{\pm} = \left(\left(\bigotimes_{a_c \in C(a_i)} util_i^{c\pm} \right) \otimes local\_util_i \right) \perp_S \perp_{\{x_i\}}^{-} \tag{9}$$

where $util_i^{c\pm}$ is the maximal and minimal cost tables received from its child a_c, and $S \subset dims\left(\left(\bigotimes_{a_c \in C(a_i)} util_i^{c-} \right) \otimes local\_util_i \right)$ is the dimension set selected to make the dimension size of the outgoing cost table below the limit k (i.e., $|dims(util_i^-)| = |dims(util_i^+)| \leq k$). During the VALUE propagation phase, each agent a_i makes a decision according to the partial assignment PA_i^* received from its parent, its local cost table and the received maximal cost table or minimal cost table $u \in \{util_i^{c-}, util_i^{c+}\}$. That is,

[2] For a coherent presentation, we denote the maximum dimension limit as k instead of *maxDims* in ADPOP [32].

$$d_i = \arg\min_{d_i \in D_i} \sum_{a_c \in C(a_i)} u(PA_i^*, d_i) + local\_util_i(PA_i^*, d_i). \tag{10}$$

4 PT-ISABB

In this section, we present our proposed PT-ISABB, a two-phase hybrid complete algorithm for ADCOPs. We describe the motivation of PT-ISABB in Sect. 4.1, and then elaborate on its inference phase and search phase in Sects. 4.2 and 4.3, respectively. Next, a discussion about the privacy in PT-ISABB is provided in Sect. 4.5. Finally, a trace of PT-ISABB for our example DCOP is shown in Sect. 4.5.

4.1 Motivation

A common drawback of the existing search-based complete ADCOP algorithms is that they only use *local* knowledge to compute lower bounds, which could lead to inefficient pruning. Taking AsymPT-FB for example, unassigned agents report the best *local costs* under the given partial assignment to compute lower bounds by forward bounding. Take Fig. 1b as an example. a_2 requires its parent a_1 to directly expose a_1's private constraint cost (i.e., f_{12} under the assignment of x_1 and x_2), and its children a_3 and a_4 to report the lower bounds of f_{21} and f_{31} via **LB_Report** messages. Accordingly, a_2 has no knowledge about the remaining constraints (i.e., the constraints between a_1 and a_3).

On the other hand, inference-based complete algorithms (e.g., DPOP) which use local eliminations are able to propagate and aggregate the *global* cost, but they are not applicable to solve ADCOPs due to a privacy concern. For example, a_3 needs to know both f_{13} and f_{23} to optimally eliminate its variable x_3, which violates the privacy of a_1 and a_2.

Thus, to overcome the downsides, we propose a novel hybrid scheme to solve ADCOPs, which combines the advantages of both inference and search. Specifically, the scheme consists of the following phases.

- *Inference phase* performs bottom-up cost propagation with respect to a subset of constraints to build both minimal and maximal look-up tables to establish lower bounds and upper bounds for the search process.
- *Search phase* uses a tree-based complete search algorithm for ADCOPs to exhaust the search space, where an estimation reporting mechanism is introduced to avoid the direct exposure of private constraint costs and compute complete upper bounds.

More specifically, we propose a tailored version of ADPOP for the inference phase to avoid the exponential memory consumption and severe privacy loss in DPOP. For the search phase, we implement SyncABB-1ph on a pseudo tree and propose an algorithm called PT-ISABB. Although they both operate on a pseudo tree, our algorithm excels AsymPT-FB twofold. The lower bounds in PT-ISABB are at least as tight as the ones in AsymPT-FB when there is no memory limit (see Property 1 for detail). Moreover, our algorithm avoids performing forward bounding which could be expensive during the search phase.

4.2 Inference phase: a tailored version of ADPOP for ADCOPs

As mentioned earlier, solving ADCOPs with exact local eliminations leads to an unacceptable loss of privacy. Therefore, we do not require parent and pseudo parents to disclose their private functions to perform inference exactly. However, ignoring the private constraints of parent and pseudo parents would lead to inconsistencies when performing local eliminations. In other words, we actually trade the bound tightness for privacy. We try to alleviate the problem by performing *non-local* elimination which is elaborated as follows.

When receiving the maximal and minimal cost tables $util_c^{\pm}$ from its child $a_c \in C(a_i)$, an agent a_i joins the received cost tables with its constraint function f_{ic} and eliminates x_c to improve the completeness of $util_i^{c\pm}$. To further enhance the completeness of $util_i^{c\pm}$ to compute tighten bounds for the search phase, we also consider the remaining constraints in a_c's branch. That is,

$$util_i^{c\pm} = (util_c^{\pm} \otimes f_{ic}) \perp_{x_c} \otimes \left(\bigotimes_{a_j \in PC(a_i) \cap Desc(a_c)} f_{ij} \perp_{x_j} \right) \tag{11}$$

Here, $Desc(a_c)$ can be propagated from a_c to a_i along with the cost table propagation which allows an agent a_i to know $Desc(a_c)$ in computing the most right part of Eq. (11).

Compared to DPOP and ADPOP, the elimination of each variable is postponed to its parent rather than performed by itself. Taking Fig. 1b as an example, the COST message sent from a_3 to a_2 is given by:

$$f_{31} \otimes f_{32}$$

if the limit $k \geq 3$. The elimination of x_3 is actually performed by a_2. That is,

$$util_2^{3\pm} = (f_{23} \otimes f_{32} \otimes f_{31}) \perp_{x_3}.$$

After all the COST messages from its children have arrived, a_i disposes the received cost tables according to Eq. (11), joins them with its local cost table and drops S (e.g., the dimensions of its highest ancestors[3] to stay below the limit k) when the dimension size of the outgoing cost tables exceeds the memory budget, and sends a COST message to its parent along with $util_i^{\pm}$.

$$util_i^{\pm} = \left(\left(\bigotimes_{a_c \in C(a_i)} util_i^{c\pm} \right) \otimes local\_util_i \right) \perp_S \tag{12}$$

Here, the local cost table $local\_util_i$ denotes the combination of the constraints between agent a_i and $AP(a_i)$ enforced in a_i's side. That is,

$$local\_util_i = \bigotimes_{a_j \in AP(a_i)} f_{ij} \tag{13}$$

[3] In our implementation, this is achieved by agents propagating their levels in the pseudo tree to their children and pseudo children, which allows an agent a_i knows which agent is its highest ancestor.

Algorithm 1: Inference phase for a_i

When `Initialization()`**:**

1 $util_i^{\pm} \leftarrow copy(local\_util_i)$
2 **if** a_i *is a leaf* **then**
3 **SendUtil**()

When `received UTIL(`$util_c^{\pm}$`)` *from* $a_c \in C(a_i)$**:**

4 compute $util_i^{c\pm}$ according to Equation (11)
5 $util_i^{\pm} \leftarrow util_i^{\pm} \otimes util_i^{c\pm}$
6 **if** a_i *have received all UTIL from* $C(a_i)$ **then**
7 **if** a_i *is the root* **then**
8 start **Search** phase
9 **else**
10 **SendUtil**()

Function `SendUtil()`**:**

11 **if** $|dims(util_i^-)| > k$ **then**
12 select $S \subset dims(util_i^-), s.t. |S| = |dims(util_i^-)| - k$
13 $util_i^{\pm} \leftarrow util_i^{\pm} \perp_S$
14 send UTIL($util_i^{\pm}$) to $P(a_i)$

Fig. 2 Pseudo code of inference phase

It can be seen from Eqs. (12) and (13) that a_i only considers its private functions when performing variable elimination and thus the privacy of $AP(a_i)$ are guaranteed.

Figure 2 presents the pseudo code of the inference phase for PT-ISABB. The phase begins with leaf agents sending their local cost tables to their parents via COST messages (lines 1–3, 11–14). Particularly, if the dimension size of the outgoing cost tables exceeds the limit k (line 11), a_i drops the exceeded dimensions with a maximal and minimal projection (lines 12–13). When receiving a COST message from its child a_c, a_i deals with the received cost tables according to Eq. (11) (line 4). After all the COST messages from its children have arrived, a_i propagates the joint maximal and minimal cost tables to its parent if it is not the root agent (lines 9–10, 11–14). Otherwise, it starts the search phase (lines 7–8).

4.3 Search phase: a variant of SyncABB-1ph on a pseudo tree

The search phase performs a depth-first branch-and-bound search and a one-phase strategy on a pseudo tree to exhaust the search space. More specifically, each branching agent decomposes the problem into several subproblems and thus its children can solve their subproblems in parallel. To discover the optimal solution and discard the suboptimal solution promptly, each agent a_i needs to maintain the lower bound LB_i and upper bound UB_i for its subproblem. To calculate UB_i, a_i needs to access the complete information about its subproblem including the private constraints on the sides of $Sep(a_i)$. Thus, we introduce an *estimation reporting* mechanism where agents in $Sep(a_i)$ report the optimistic and pessimistic accumulated estimations of their private constraints involved in a_i's subproblem. Here, the optimistic and pessimistic estimation of a binary function $f_{ji}(x_j, x_i)$ with a given (x_j, d_j) are the lower bound and upper bound of the unitary function $f_{ji}(d_j, x_i)$, respectively (i.e., $\min_{d_i \in D_i} f_{ji}(d_j, d_i)$ and $\max_{d_i \in D_i} f_{ji}(d_j, d_i)$). Further, for f_{ji} with a given $\{(x_j, d_j), (x_i, d_i)\}$,

we introduce the residual constraint cost as the difference between its actual cost and its optimistic estimation. Consequently, to accumulate the two-side constraint costs, parent and pseudo parents only need to forward their residual constraint costs instead of directly disclosing their private constraint costs in AsymPT-FB.

Specifically, each agent a_i needs to maintain the following data structures.

- Cpa_i refers to the current partial assignment (Cpa), which contains all the assignments to $Sep(a_i)$.
- $Srch\_val_i^c$ records the assignment currently being explored by the agents in the sub-tree rooted at its child $a_c \in C(a_i)$. The data structure is necessary since asynchronous search is carried out concurrently in sub-trees based on different possible values of x_i.
- $EstRep_i$ contains all the optimistic and pessimistic accumulated estimations of a_i and its descendants. $EstRep_i(x_j) \in EstRep_i$ is the accumulated estimations of $a_j \in \{a_i\} \cup Desc(a_i)$, which can be formalized as:

$$EstRep_i(x_j) = \bigotimes_{a_l \in Sep(a_i) \cap AP(a_j)} f_{lj}(Cpa_i(x_l), x_j) \perp_{x_j} \tag{14}$$

  It is worth noting that calculating $EstRep_i(x_j)$ is actually an incremental process in the proposed estimation reporting mechanism and does not require parent and pseudo parents to transfer their private cost functions as Eq. (14) does. Here, $EstRep_i(x_j) = \{EstRep_i(x_j)^-, EstRep_i(x_j)^+\}$ where $EstRep_i(x_j)^-$ and $EstRep_i(x_j)^+$ are the optimistic and pessimistic accumulated estimation of x_j, respectively.
- $high\_cost_i(d_i)$ is the sum of the two-side costs of constraints between a_i and $AP(a_i)$ under Cpa_i and its assignment (x_i, d_i) for $d_i \in D_i$. It is initialized to the sum of the constraint costs enforced in a_i's side and the optimistic accumulated estimation of a_i. That is,

$$high\_cost_i(d_i) = \sum_{a_j \in AP(a_i)} f_{ij}(d_i, Cpa_i(x_j)) + EstRep_i(x_i)^- \tag{15}$$

  Here, $EstRep_i(x_i)^-$ is included to avoid the direct exposure of parents' privacy constraint costs for obtaining the complete hight costs. Specifically, for obtaining the complete $high\_cost_i(d_i)$, a_i only needs to request all its parents for the corresponding residual constraint costs for d_i rather than the actual constraint costs. After receiving these residual constraint costs, $high\_cost_i(d_i)$ reflects the double-side cost between a_i and $AP(a_i)$ for d_i. That is,

$$\sum_{a_j \in AP(a_i)} f_{ij}(d_i, Cpa_i(x_j)) + f_{ji}(Cpa_i(x_j), d_i)$$

- $lb_i^c(d_i)$ is the lower bound of $a_c \in C(a_i)$ for $d_i \in D_i$, which is initially set to $util_i^{c-}$ under $Cpa_i \cup \{(x_i, d_i)\}$ plus the optimistic accumulated estimations of agents in $Desc(a_c) \cup \{a_c\}$. That is,

$$lb_i^c(d_i) = util_i^{c-}(Cpa_{i[Sep(a_c)]}, d_i) + \sum_{a_j \in Desc(a_c) \cup \{a_c\}} EstRep_i(x_j)^- \tag{16}$$

- $ub_i^c(d_i)$ is the upper bound of $a_c \in C(a_i)$ for $d_i \in D_i$, which is initially set to $util_i^{c+}$ under $Cpa_i \cup \{(x_i, d_i)\}$ plus the pessimistic accumulated estimations of agents in $Desc(a_c) \cup \{a_c\}$. That is,

$$ub_i^c(d_i) = util_i^{c+}(Cpa_{i[Sep(a_c)]}, d_i) + \sum_{a_j \in Desc(a_c) \cup \{a_c\}} EstRep_i(x_j)^+ \tag{17}$$

Since obtaining a complete $ub_i^c(d_i)$ needs to consider the double-side costs of all constraints related to $\{a_c\} \cup Desc(a_c)$ according to Lemmas 2 and 3, we include $EstRep_i(x_j)^+, \forall a_j \in Desc(a_c) \cup \{a_c\}$ in the calculation of Eq. (17) and separates the join computations of all constraints f_{ij} with $PC(a_i)$ by each a_c in Eq. (11) rather than join all constraints f_{ij} with $PC(a_i)$ in Eq. (12).

- $lb_i(d_i)$ is the lower bound for $d_i \in D_i$. That is,

$$lb_i(d_i) = high\_cost_i(d_i) + \sum_{a_c \in C(a_i)} lb_i^c(d_i) \tag{18}$$

- $ub_i(d_i)$ is the upper bound for $d_i \in D_i$ and initially set to

$$ub_i(d_i) = high\_cost_i(d_i) - EstRep_i(x_i)^- + EstRep_i(x_i)^+ + \sum_{a_c \in C(a_i)} ub_i^c(d_i) \tag{19}$$

After a_i receives all the residual constraint costs from $AP(a_i)$ for d_i, $ub_i(d_i)$ needs to be replaced with $high\_cost_i(d_i) + \sum_{a_c \in C(a_i)} ub_i^c(d_i)$.

- LB_i is the lower bound of its subproblem under Cpa_i. That is,

$$LB_i = \min_{d_i \in D_i} lb_i(d_i) \tag{20}$$

- UB_i is the upper bound of its subproblem under Cpa_i. That is,

$$UB_i = \min_{d_i \in D_i} ub_i(d_i) \tag{21}$$

To implement a depth-first branch-and-bound search and a one-phase strategy on a pseudo tree, four types of messages are used in the search phase.

- *CPA* is a message sent from agent a_i to $a_c \in C(a_i)$ and contains the partial assignment Cpa_c, threshold TH_c and accumulated estimations $EstRep_c$.

$$a_i \rightarrow a_c : CPA(Cpa_c, TH_c, EstRep_c), a_c \in C(a_i)$$

where Cpa_c is the partial assignment extended with the assignment (x_i, d_i) (i.e., $Cpa_c = Cpa_{i[Sep(a_c)]} \cup \{(x_i, d_i)\}$), and TH_i is the threshold of a_i's subproblem under Cpa_i. Particularly, $TH_i = \infty$ if a_i is the root agent. TH_c is the threshold of a_c's subproblem under Cpa_c, and can be computed by:

$$TH_c = \min(UB_i, TH_i) - high\_cost_i(d_i) - \sum_{a_j \in C(a_i) \wedge j \neq c} lb_i^j(d_i) \tag{22}$$

- *BACKTRACK* is a message sent from a_i to $P(a_i)$ and consists of the cost opt_i and corresponding partial assignment $Spa_i(d_i^*)$.

$$a_i \rightarrow P(a_i) : BACKTRACK(opt_i, Spa_i(d_i^*))$$

where opt_i is the optimal cost of a_i's subproblem under Cpa_i if Cpa_i is feasible (i.e., $opt_i \leq TH_i$) or ∞ otherwise, and $Spa_i(d_i^*)$ refers to the optimal assignment of its

subproblem under $Cpa_i \cup \{(x_i, d_i^*)\}$. Here, $d_i^* = \arg\min_{d_i \in D_i} ub_i(d_i)$. It is worth noting that $Spa_i(d_i)$ is initially set to $\{(x_i, d_i)\}$, and $Spa_i(d_i^*)$ is the optimal solution if a_i is the root agent.

- *RESD_REQ* is a message sent from a_i to $a_j \in AP(a_i)$ to request the residual constraint cost of f_{ji} under the assignment $\{(x_i, d_i), (x_j, d_j)\}$.

$$a_i \rightarrow a_j : RESD\_REQ(d_i, d_j), a_j \in AP(a_i)$$

- *RESD* is a message sent from a_i to $a_j \in C(a_i) \cup PC(a_i)$ to response the received RESD_REQ message with $\{(x_i, d_i), (x_j, d_j)\}$ from a_j.

$$a_i \rightarrow a_j : RESD(r_{ij}, d_j), a_j \in C(a_i) \cup PC(a_i)$$

Here, the residual constraint cost r_{ij} is calculated by:

$$r_{ij} = f_{ij}(d_i, d_j) - \min_{d_j' \in D_j} f_{ij}(d_i, d_j') \tag{23}$$

Figures 3 and 4 present the pseudo codes of the search phase for PT-ISABB. The phase begins with the root agent sending its first feasible assignment [i.e., the first assignment d_i such that $lb_i(d_i) < \min(UB_i, TH_i)$ (lines 67–70)] to its children via CPA messages (lines 15–18, 71–74). When receiving a CPA message from its parent, a_i stores the received Cpa_i, threshold and estimation reports, and then initializes its related variables according to Eqs. (15–21) (lines 19–20, 63–66). Afterwards, a_i backtracks to its parent with an infinity cost and an empty subproblem assignment if Cpa_i is infeasible (lines 21–22). Otherwise, it finds its first feasible assignment d_i and requests the residual constraint costs for that assignment via RESD_REQ messages (lines 23–26). Upon receipt of a RESD_REQ message from its (pseudo) child, a_i replies with the corresponding residual constraint cost via a RESD message (line 27).

When receiving a RESD message for d_i, a_i adds the reported residual constraint cost to its high cost for d_i (line 28). Upon receipt of all RESD messages for d_i from its parent and pseudo parents, a_i updates its bounds (lines 29–30), and backtracks to its parent with the found solution (i.e., the best cost and subproblem assignment when Cpa_i is feasible or ∞ and an empty assignment otherwise) via a BACKTRACK message if the backtrack condition is satisfied (lines 31–32, 75–79). Otherwise, a_i continues to explore the search space. For a leaf agent a_i, it just switches to its next feasible assignment d_i' and requests the residual constraint costs for d_i' (lines 34–36). If a_i is a non-leaf agent and d_i is still worth being explored, it extends Cpa_i and sends the extended Cpa_i to its children who are going to explore d_i (lines 43–44). Otherwise, d_i is proven to be suboptimal and thus a_i switches to its next feasible assignment d_i'[4] and requests residual constraint costs for d_i' (lines 38–42).

When receiving a BACKTRACK message for d_i from a_c, a_i updates its corresponding bounds with the reported cost opt_c and merges the received partial assignment Spa_c if $Cpa_i \cup \{(x_i, d_i)\}$ is feasible (otherwise, $opt_c = \infty$ and $Spa_c = \emptyset$) (line 45). Then, a_i determines whether it needs to backtrack (line 46). If it needs, a backtrack takes place if it is not the root agent (lines 50–51). Otherwise, it informs its children to terminate and terminates itself (lines 47–49). If it does not need to backtrack, a_i determines the next

[4] When the next feasible assignment d_i' does not exist, a_i stops here, does nothing and waits for BACKTRACK messages for the assignment d_i'' before d_i from its children.

Algorithm 1: Search phase for a_i (message passing)

```
    When Initialization():
15      if a_i is the root then
16          InitializeVariables()
17          d_i ← NextFeasibleAssignment(null)
18          SendCpa(d_i, a_c), ∀a_c ∈ C(a_i)
    When received CPA(Cpa_i, TH_i, EstRep_i) from P(a_i):
19      stores {Cpa_i, TH_i, EstRep_i}
20      InitializeVariables()
21      if LB_i ≥ min(UB_i, TH_i) then
22          send BACKTRACK(∞, ∅) to P(a_i)
23      else
24          d_i ← NextFeasibleAssignment(null)
25          Srch_val_i^c ← d_i, ∀a_c ∈ C(a_i)
26          send RESD_REQ(Cpa_i(x_j), d_i) to a_j, ∀a_j ∈ AP(a_i)
    When received RESD_REQ(d_i, d_c) from a_c ∈ C(a_i) ∪ PC(a_i):
27      send RESD(f_ic(d_i, d_c) − min_{d'_c ∈ D_c} f_ic(d_i, d'_c), d_c) to a_c
    When received RESD(r_ji, d_i) from a_j ∈ AP(a_i):
28      high_cost_i(d_i) ← high_cost_i(d_i) + r_ji
29      if a_i has received all RESD from AP(a_i) for d_i then
30          ub_i(d_i) ← Σ_{a_c ∈ C(a_i)} ub_i^c(d_i) + high_cost_i(d_i)
31          if LB_i ≥ min(UB_i, TH_i) then
32              SendBackTrack()
33          else
34              if a_i is a leaf then
35                  d'_i ← NextFeasibleAssignment(d_i)
36                  send RESD_REQ(Cpa_i(x_j), d'_i) to a_j, ∀a_j ∈ AP(a_i)
37              else
38                  if lb_i(d_i) ≥ min(UB_i, TH_i) then
39                      d'_i ← NextFeasibleAssignment(d_i)
40                      if d'_i ≠ null then
41                          Srch_val_i^c ← d'_i, ∀a_c ∈ C(a_i) ∧ Srch_val_i^c = d_i
42                          send RESD_REQ(Cpa_i(x_j), d'_i) to a_j, ∀a_j ∈ AP(a_i)
43                  else
44                      SendCpa(d_i, a_c), ∀a_c ∈ C(a_i) ∧ Srch_val_i^c = d_i
    When received BACKTRACK(opt_c, Spa_c) from a_c ∈ C(a_i):
45      d_i ← Srch_val_i^c, Spa_i(d_i) ← Spa_i(d_i) ∪ Spa_c,
        lb_i^c(d_i) ← max(lb_i^c(d_i), opt_c), ub_i^c(d_i) ← min(ub_i^c(d_i), opt_c)
46      if LB_i ≥ min(UB_i, TH_i) then
47          if a_i is the root then
48              send TERMINATE to a_c, ∀a_c ∈ C(a_i)
49              terminate
50          else
51              SendBackTrack()
52      else
53          d'_i ← NextFeasibleAssignment(d_i), Srch_val_i^c ← d'_i
54          if d'_i ≠ null then
55              if a_i has received all RESD from AP(a_i) for d'_i then
56                  SendCpa(d'_i, a_c)
57              else if a_i has not requested RESD for d'_i then
58                  send RESD_REQ(Cpa_i(x_j), d'_i) to a_j, ∀a_j ∈ AP(a_i)
    When received TERMINATE from P(a_i):
59      send TERMINATE to a_c, ∀a_c ∈ C(a_i)
60      terminate
    When received STOPEXPLORE from P(a_i):
61      send STOPEXPLORE to ∀a_c ∈ C(a_i) ∧ srch_val_i^c ≠ null
62      sleep and wait for a CPA message to wake it up
```

Fig. 3 Pseudo code of search phase (message passing)

feasible assignment d'_i for a_c (lines 52–58). Specifically, if d'_i exists and a_i has received all RESD messages from $AP(a_i)$ for d'_i, it informs a_c to explore d'_i (lines 53–56). Otherwise, a_i requests the residual constraint costs for d'_i if it has not done (lines 57–58).

Algorithm 1: Search phase for a_i (auxiliary functions)

Function `InitializeVariables`:
initialize $Srch\_val_i^c, \forall a_c \in C(a_i)$
initialize $lb_i^c(d_i), ub_i^c(d_i), \forall a_c \in C(a_i), d_i \in D_i$
initialize $high\_cost_i(d_i), Spa_i(d_i), lb_i(d_i), ub_i(d_i), \forall d_i \in D_i$
initialize LB_i, UB_i
Function `NextFeasibleAssignment`(d_i):
$d'_i \leftarrow$ the element next to d_i in D_i
while $d'_i \neq null \wedge lb_i(d'_i) \geq \min(UB_i, TH_i)$ **do**
  $d'_i \leftarrow$ the next element in D_i
return d'_i
Function `SendCpa`(d_i, a_c):
$Cpa_c \leftarrow Cpa_{i[Sep(a_c)]} \cup \{(x_i, d_i)\}, Srch\_val_i^c \leftarrow d_i$
$TH_c \leftarrow \min(UB_i, TH_i) - high\_cost_i(d_i) - \sum_{a_j \in C(a_i) \wedge j \neq c} lb_i^j(d_i)$
$EstRep_c \leftarrow$ **GetEstReports**(d_i, a_c)
send CPA$(Cpa_c, TH_c, EstRep_c)$ to a_c
Function `SendBackTrack()`:
if $LB_i = UB_i \wedge LB_i < TH_i$ **then**
  $d_i^* \leftarrow \arg\min_{d_i \in D_i} ub_i(d_i)$
  send BACKTRACK$(ub_i(d_i^*), Spa_i(d_i^*))$ to $P(a_i)$
else
  send BACKTRACK$(\infty, \emptyset)$ to $P(a_i)$
send STOPEXPLORE to $\forall a_c \in C(a_i) \wedge srch\_val_i^c \neq null$
sleep and wait for a CPA message to wake it up
Function `GetEstReports` (d_i, a_c):
$EstRep \leftarrow \emptyset$
foreach $a_j \in Desc(a_c) \cup \{a_c\}$ **do**
  if $x_j \in EstRep_i \wedge a_j \in AC(a_i)$ **then**
    $EstRep \leftarrow EstRep \cup \{(x_j, EstRep_i(x_j) \otimes f_{ij}(d_i)_{\perp_{x_j}})\}$
  else if $x_j \in EstRep_i \wedge a_j \notin AC(a_i)$ **then**
    $EstRep \leftarrow EstRep \cup \{(x_j, EstRep_i(x_j))\}$
  else if $x_j \notin EstRep_i \wedge a_j \in AC(a_i)$ **then**
    $EstRep \leftarrow EstRep \cup \{(x_j, f_{ij}(d_i)_{\perp_{x_j}})\}$
return $EstRep$

Fig. 4 Pseudo code of search phase (auxiliary functions)

4.4 Privacy of PT-ISABB

Privacy preserving is one of the main motivations for solving constraint problems in a distributed manner, since agents would not like their private information to be revealed. In [22], Faltings et al. distinguished among four notions of private information in distributed constraint reasoning. These four notions include agent privacy where each agent does not know the identity or even the existence of its non-neighbors, topology privacy where each agent has no idea of the topology of the constraint graph that it is not involved, constraint privacy where each agent has no knowledge about the constraints that it is not involved, and decision privacy where each agent does not know the final assignments to the variables that it does not control. Additionally, they also presented several versions of DPOP to provide strong privacy guarantees. Recently, several efforts have been made to provide strong privacy guarantees to existing DCOP and ADCOP algorithms by the means of cryptographic techniques. P-SyncBB [16] was proposed as a privacy-preserving SyncBB for DCOPs while respecting topology, constraint, and decision privacy. P-RODA [17] came out as a region optimal framework for local search algorithms for both DCOPs and ADCOPs to preserve constraint and partial decision privacy.

Instead of providing strong privacy guarantees, we devote to reduce the amount of constraint privacy loss when optimally solving ADCOPs with the search strategy. This is achieved by improving the pruning efficiency with initial tight bounds derived from the inference phase and an estimation reporting mechanism. For evaluating the performance of different ADCOP algorithms on the constraint privacy protecting, we use entropy as the measure metric to quantify the privacy constraints loss. In [2], the entropy was introduced to measure the privacy loss in DCSPs. Specifically, for each agent a_i, the entropy measures the amount of the missing information about its constraints shared with its neighbors in bits, and can be defined by:

$$H_i = -\sum_{a_j \in N(a_i)} \sum_{s \in S_{ij}} p_s \log_2 p_s \tag{24}$$

where $N(a_i)$ is the neighbors of a_i, S_{ij} is the set of possible states of the constraint between a_i and its neighbor a_j, and p_s is the probability of the state $s \in S_{ij}$. Assume that all the states $\forall s \in S_{ij}$ have equal probability for a simple case (i.e., $p_s = \frac{1}{|S_{ij}|}, \forall s \in S_{ij}$), then Eq. (24) can be rewritten as:

$$\begin{aligned} H_i &= -\sum_{a_j \in N(a_i)} \sum_{s \in S_{ij}} \frac{1}{|S_{ij}|} \log_2 \frac{1}{|S_{ij}|} \\ &= -\sum_{a_j \in N(a_i)} |S_{ij}| \frac{1}{|S_{ij}|} \log_2 \frac{1}{|S_{ij}|} \\ &= \sum_{a_j \in N(a_i)} \log_2 |S_{ij}| \end{aligned} \tag{25}$$

Thus, to quantify the amount of privacy violation of an algorithm, we only need to consider the different value between the entropy that measures their initial states before the algorithm starts and the entropy after the algorithm terminates for all agents.

Similar to [15], the privacy loss is computed for distributed asymmetric MaxCSPs. In an asymmetric MaxCSP, the number of possible states for the constraint between a_i and a_j is $2^{|D_i||D_j|}$ at the beginning of the algorithm and $2^{q_{ij}}$ at the end of the algorithm. Here, q_{ij} equals the number of value pairs $\{(x_i, d_i), (x_j, d_j)\}$ for which the constraint cost is still unknown after the algorithm terminates. Thus, the amount of privacy loss incurred by an ADCOP algorithm can be computed by:

$$\begin{aligned} &\frac{1}{|A|} \sum_{a_i \in A} \frac{\sum_{a_j \in N(a_i)} \log_2 2^{|D_i||D_j|} - \sum_{a_j \in N(a_i)} \log_2 2^{q_{ij}}}{\sum_{a_j \in N(a_i)} \log_2 2^{|D_i||D_j|}} \\ &= 1 - \frac{1}{|A|} \sum_{a_i \in A} \frac{\sum_{a_j \in N(a_i)} \log_2 2^{q_{ij}}}{\sum_{a_j \in N(a_i)} \log_2 2^{|D_i||D_j|}} \\ &= 1 - \frac{1}{|A|} \sum_{a_i \in A} \frac{\sum_{a_j \in N(a_i)} q_{ij}}{\sum_{a_j \in N(a_i)} |D_i||D_j|}. \end{aligned} \tag{26}$$

Next, we will analyze where the constraint privacy is leaked by the message passing during the running of PT-ISABB. A COST message from agent a_i to its parent $P(a_i)$ would cause a direct privacy loss on the constraint between them on a_i's side in the worst case.

Table 1 The cost tables propagated in the inference phase

x_2 \ x_4	0	1
0	2	1
1	3	2

$util^{-}_{4\to 2}$

x_2 \ x_4	0	1
0	2	1
1	3	2

$util^{+}_{4\to 2}$

(a) $util^{\pm}_{4\to 2}$

x_2 \ x_3	0	1
0	8	6
1	9	5

$util^{-}_{3\to 2}$

x_2 \ x_3	0	1
0	9	11
1	10	10

$util^{+}_{3\to 2}$

(b) $util^{\pm}_{3\to 2}$

x_1 \ x_2	0	1
0	16	20
1	17	26

$util^{-}_{2\to 1}$

x_1 \ x_2	0	1
0	26	28
1	27	34

$util^{+}_{2\to 1}$

(c) $util^{\pm}_{2\to 1}$

That is because the elimination of variable x_i is performed by its parent agent $P(a_i)$, and $P(a_i)$ can easily figure out which pairs of assignments in the received cost table are feasible. More specifically, as for the constraint enforced on a_i's side, $P(a_i)$ can directly deduce the feasible assignments from the zero entries of the cost table sent by a_i. A CPA message from $P(a_i)$ to a_i gives away a small amount of information since a_i can infer that the constraints between a_i and its direct ancestors are always feasible when the pessimistic accumulated estimation of x_i equals zero (i.e., $EstRep_i(x_i)^+ = 0$). Similarly, these constraints are proven to be infeasible when the optimistic accumulated estimation of x_i equals the number of its direct ancestors (i.e., $EstRep_i(x_i)^- = |AP(a_i)|$). A BACKTRACK message also leaks a negligible amount of information since the privacy loss happens only by the messages with the reported optimal cost equal to zero. A RESD_REQ message reveals no information as it only contains the assignment of the message sender and message receiver. Another significant information leak may be incurred by RESD messages. A RESD message from the (pseudo) parent $a_j \in AP(a_i)$ to a_i contains a direct revelation of the constraint cost for the specific assignment of x_i and x_j if the optimistic accumulated estimation for x_i equals zero (i.e., $EstRep_i(x_i)^- = 0$). Otherwise, the direct revelation happens when the received residual constraint costs are non-zero.

4.5 Trace

In this subsection, we will take Fig. 1 as an example to show the trace of PT-ISABB.

Cycle 1 After constructing a pseudo tree shown in Fig. 1b, given $k = 2$, the inference phase begins with a_3 and a_4 sending their local cost tables to a_2 via COST messages (lines 1–3, 11–14).

$$a_4 \to a_2 : \text{COST}(util^{\pm}_{4\to 2})$$
$$a_3 \to a_2 : \text{COST}(util^{\pm}_{3\to 2})$$

where $util^{\pm}_{4\to 2} = f_{42}$ and $util^{\pm}_{3\to 2} = (f_{31} \otimes f_{32})\perp_{x_1}$ which are presented in Table 1(a) and (b), respectively. Note that x_1 is the dimension dropped by a_3 since $dims(f_{31} \otimes f_{32}) > k$ when computing the cost tables $util^{\pm}_{3\to 2}$.

Table 2 The cost tables stored in a_1 and a_2

x_2	$util_2^{4-}$	$util_2^{4+}$
(a) $util_2^{4\pm}$		
0	5	8
1	8	11
x_2	$util_2^{3-}$	$util_2^{3+}$
(b) $util_2^{3\pm}$		
0	7	14
1	10	15
x_1	$util_1^{2-}$	$util_1^{2+}$
(c)$util_1^{2\pm}$		
0	19	36
1	27	50

Table 3 a_i's high costs and its own bounds

Cycle(a_i)	3(a_1)		4, 6(a_2)		7(a_3)		3(a_4)		9(a_3)		9(a_4)		11(a_3)	
d_i	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$high\_cost_i(d_i)$	0	0	5	3	11	14	5	4	15	14	5	4	15	19
$ub_i(d_i)$	36	50	34	36	20	23	9	8	15	23	5	8	15	19
$lb_i(d_i)$	19	27	19	23	11	14	5	4	15	14	5	4	15	19
UB_i	36		34		20		8		15		5		15	
LB_i	19		19		11		4		14		4		15	
Cycle(a_i)	11(a_4)		12, 14(a_2)		15(a_3)		15(a_4)		16(a_2)		17(a_1)		–	
d_i	0	1	0	1	0	1	0	1	0	1	0	1	–	–
$high\_cost_i(d_i)$	5	8	5	3	14	15	8	7	5	3	0	0	–	–
$ub_i(d_i)$	5	8	25	36	21	22	12	11	25	36	25	50	–	–
$lb_i(d_i)$	5	8	25	23	14	15	8	7	25	∞	25	27	–	–
UB_i	5		25		21		11		25		25		–	
LB_i	5		23		14		7		25		25		–	

Cycle 2 When receiving COST messages from a_3 and a_4, a_2 joins the received cost tables with f_{23} and f_{24}, and eliminates x_3 and x_4, respectively.

$$util_2^{4\pm} = (util_{4\to 2}^{\pm} \otimes f_{24})\perp_{x_4}, util_2^{3\pm} = (util_{3\to 2}^{\pm} \otimes f_{23})\perp_{x_3}$$

Afterwards, it computes the joint cost tables $util_{2\to 1}^{\pm} = f_{21} \otimes util_2^{3\pm} \otimes util_2^{4\pm}$ (line 5) which is presented in Table 1(c). Since $|dims(util_{2\to 1}^{\pm})| = k$, it directly sends $util_{2\to 1}^{\pm}$ to a_1 (lines 9–10, 11–14).

$$a_2 \to a_1 : COST((util_{2\to 1}^{\pm})$$

Table 4 a_i's bounds for its children

Cycle(a_i)	3(a_1)		4, 6(a_2)		12, 14(a_2)		16(a_2)		17(a_1)	
d_i	0	1	0	1	0	1	0	1	0	1
$ub_i^c(d_i)$	36	50	21	22	15	22	15	22	25	50
	–	–	8	11	5	11	5	11	–	–
$lb_i^c(d_i)$	19	27	9	12	15	12	15	∞	25	27
	–	–	5	8	5	8	5	8	–	–

Cycle 3 Once receiving a COST message from a_2, a_1 calculates $util_1^{2\pm}$ by:

$$util_1^{2\pm} = (util_{2\to 1}^{\pm} \otimes f_{12} \otimes f_{13}\bot_{x_3})\bot_{x_2}$$

Since a_1 has received all the COST messages from its children, the inference phase finishes and the inference results $util_2^{4\pm}$, $util_2^{3\pm}$ and $util_1^{2\pm}$ stored in the corresponding agents a_1 and a_2 are presented in Table 2.

Then, the search phase begins with a_1 who initializes its related variables according to Eqs. (15–21) as shown in Cycle 3 of Tables 3 and 4 (lines 7–8, 15–16, 63–66). Since $LB_1 < \min(TH_1, UB_1)$ (i.e., $19 < \min(\infty, 36)$), a_1 sends a CPA message with its first feasible assignment $(x_1, 0)$, threshold TH_2 and estimation report $EstRep_2$ to a_2 (lines 17–18).

$$a_1 \to a_2 : \text{CPA}(\{(x_1, 0)\}, TH_2, EstRep_2)$$

where $TH_2 = 36$ and $EstRep_2 = \{(\langle {x_2}^-, 1\rangle, \langle {x_2}^+, 1\rangle), (\langle {x_3}^-, 2\rangle, \langle {x_3}^+, 7\rangle)\}$ are computed according to Eqs. (22) and (14), respectively.

Cycle 4 When receiving a CPA message from a_1, a_2 stores the received $Cpa_2 = \{(x_1, 0)\}$, TH_2 and $EstRep_2$, and then initializes its related variables as shown in Cycle 4 of Tables 3 and 4 (lines 19–20). Since $LB_2 < \min(UB_2, TH_2)$ (i.e., $19 < \min(34, 36)$), a_2 finds its first feasible assignment $(x_2, 0)$ and requests the residual constraint cost of f_{12} via a RESD_REQ message to a_1 (lines 23–26).

$$a_2 \to a_1 : \text{RESD\_REQ}(0, 0)$$

Cycle 5 Upon receipt of a RESD_REQ message from a_2, a_1 calculates the residual cost $r_{12} = 0$ by Eq. (23) and sends that cost to a_1 via a RESD message (line 27).

$$a_2 \to a_1 : \text{RESD}(r_{12} = 0, 0)$$

Cycle 6 After receiving a RESD message along with $r_{12} = 0$, a_2 does not update its related variables and sends $Cpa_2 \cup \{(x_2, 0)\}$ to a_3 and a_4 via CPA messages.

$$a_2 \to a_3 : \text{CPA}(\{(x_1, 0), (x_2, 0)\}, TH_3 = 24, EstRep_3)$$
$$a_2 \to a_4 : \text{CPA}(\{(x_1, 0), (x_2, 0)\}, TH_4 = 20, EstRep_4)$$

Here, $EstRep_3 = \{\langle x_3^-, 3\rangle, \langle x_3^+, 12\rangle\}$ and $EstRep_4 = \{\langle x_4^-, 3\rangle, \langle x_4^+, 7\rangle\}$.

Cycle 7 When receiving CPA messages from a_2, a_3 and a_4 initialize their related variables as shown in Cycle 7 of Table 3, find their first feasible assignments ($(x_3 = 0)$ and $(x_4 = 0)$) and requests the residual constraint costs of f_{24}, f_{13} and f_{23}.

$$a_4 \rightarrow a_2 : \text{RESD\_REQ}(0, 0)$$
$$a_3 \rightarrow a_1 : \text{RESD\_REQ}(0, 0)$$
$$a_3 \rightarrow a_2 : \text{RESD\_REQ}(0, 0)$$

Cycle 8 When receiving RESD_REQ messages from a_4 and a_3, a_1 and a_2 reply with corresponding constraint residual costs just like a_1 in Cycle 5.

$$a_2 \rightarrow a_4 : \text{RESD}(r_{24} = 0, 0)$$
$$a_1 \rightarrow a_3 : \text{RESD}(r_{13} = 0, 0)$$
$$a_2 \rightarrow a_3 : \text{RESD}(r_{23} = 4, 0)$$

Cycle 9 After receiving the residual constraint costs, a_3 and a_4 update their related variables as shown in Cycle 9 of Table 3. Since $LB_4 < \min(UB_4, TH_4)$ and $LB_3 < \min(UB_3, TH_3)$, they switch to their next feasible assignment (($x_3 = 1$) and ($x_4 = 1$)) and request the residual constraint costs of f_{24}, f_{13} and f_{23}.

$$a_4 \rightarrow a_2 : \text{RESD\_REQ}(0, 1)$$
$$a_3 \rightarrow a_1 : \text{RESD\_REQ}(0, 1)$$
$$a_3 \rightarrow a_2 : \text{RESD\_REQ}(0, 1)$$

Cycle 10 When receiving RESD_REQ messages from a_4 and a_3, a_1 and a_2 response with corresponding constraint residual costs.

$$a_2 \rightarrow a_4 : \text{RESD}(r_{24} = 4, 1)$$
$$a_1 \rightarrow a_3 : \text{RESD}(r_{13} = 5, 1)$$
$$a_2 \rightarrow a_3 : \text{RESD}(r_{23} = 0, 1)$$

Cycle 11 After receiving the residual constraint costs, a_4 and a_3 update their related variables as shown in Cycle 11 of Table 3. Since $UB_3 = LB_3$ and $UB_4 = LB_4$, both a_4 and a_3 backtrack with their optimal costs and optimal partial assignments to a_2 (lines 31–32).

$$a_4 \rightarrow a_2 : \text{BACKTRACK}(opt_4 = ub_4(d_4^*) = 5, Spa_4 = \{(x_4, d_4^*)\})$$
$$a_3 \rightarrow a_2 : \text{BACKTRACK}(opt_3 = ub_3(d_3^*) = 15, Spa_3 = \{(x_3, d_4^*)\})$$

where $d_4^* = \arg\min_{d_4 \in D_4} ub_4(d_4) = 0$ and $d_3^* = \arg\min_{d_3 \in D_3} ub_3(d_3) = 0$.

Cycle 12 When receiving BACKTRACK messages for ($x_2 = 0$) from a_4 and a_3, a_2 updates the bounds for a_4 and a_3 with the reported costs (shown in Cycle 12 of Tables 3 and 4) and merges the received subproblem assignments (i.e., $Spa_2(0) \cup Spa_3 \cup Spa_4 = \{(x_2, 0), (x_3, 0), (x_4, 0)\}$) (line 45). Since $LB_2 < UB_2$, a_2 switches to its next feasible assignment ($x_2 = 1$) and requests the residual constraint cost of f_{12}.

$$a_2 \rightarrow a_1 : \text{RESD\_REQ}(0, 1)$$

Cycle 13 When receiving a RESD_REQ message from a_2, a_1 replies with r_{12} via a RESD message.

$$a_2 \rightarrow a_1 : \text{RESD}(r_{12} = 0, 1)$$

Cycle 14 After receiving $r_{12} = 0$, a_2 does not update its related variables and sends $Cpa_2 \cup \{(x_2, 1)\}$ to a_3 and a_4 via CPA messages.

$$a_2 \rightarrow a_3 : \text{CPA}(\{(x_1, 0), (x_2, 1)\}, TH_3 = 14, EstRep_3)$$
$$a_2 \rightarrow a_4 : \text{CPA}(\{(x_1, 0), (x_2, 1)\}, TH_4 = 10, EstRep_4)$$

where $EstRep_3 = \{\langle x_3^-, 5\rangle, \langle x_3^+, 12\rangle\}$ and $EstRep_4 = \{\langle x_4^-, 5\rangle, \langle x_4^+, 9\rangle\}$.

Cycle 15 When receiving CPA messages from a_2, a_3 and a_4 store the received contents and initialize their related variables as shown in Cycle 15 of Table 3. Since $LB_3 = TH_3$ (i.e., the received Cpa_3 is infeasible), a_3 backtracks to a_2 (lines 31–32, 78–79). a_4 finds its first feasible assignment $(x_4, 0)$ as $lb_4(0) < \min(TH_4, UB_4)$, and requests the residual constraint cost of f_{42} via a RESD_REQ message.

$$a_3 \rightarrow a_2 : \text{BACKTRACK}(opt_3 = \infty, Spa_3 = \emptyset)$$
$$a_4 \rightarrow a_2 : \text{RESD\_REQ}(1, 0)$$

Cycle 16 When receiving a BACKTRACK message for $(x_2, 1)$ from a_3 (assume that the BACKTRACK message from a_3 arrives earlier than the RESD_REQ message from a_4), a_2 merges Spa_3 and updates its bounds as shown in Cycle 16 of Tables 3 and 4. Since $LB_2 = UB_2$, a_2 sends a BACKTRACK message to a_1 and STOPEXPLORE messages to a_3 and a_4, sleeps and ignores the RESD_REQ message from a_4.

$$a_2 \rightarrow a_1 : \text{BACKTRACK}(opt_2 = 25, Spa_2 = \{(x_2, 0), (x_3, 0), (x_4, 0)\})$$
$$a_2 \rightarrow a_3 : \text{STOPEXPLORE}()$$
$$a_2 \rightarrow a_4 : \text{STOPEXPLORE}()$$

Cycle 17 When receiving STOPEXPLORE messages from a_2, a_3 and a_4 sleep and wait CPA messages to wake them up. When receiving a BACKTRACK message for $(x_1, 0)$ from a_2, a_1 merges Spa_2 (i.e., $Spa_1(0) \cup Spa_2 = \{(x_1, 0), (x_2, 0), (x_3, 0), (x_4, 0)\}$) and updates its bounds as shown in Cycle 17 of Tables 3 and 4. Then, it finds the optimal solution (i.e., the full assignment $\{(x_1, 0), (x_2, 0), (x_3, 0), (x_4, 0)\}$ with the cost to 25), terminates itself and informs a_2 to stop via a TERMINATE massage since $LB_1 = UB_1$.

Cycle 18–19 After receiving a TERMINATE message from a_1, a_2 stops itself and sends TERMINATE messages to a_3 and a_4. Once receiving TERMINATE messages from a_2, a_3 and a_4 stop themselves. Finally, the algorithm terminates since all the agents have stopped.

5 Theoretic results

In this section, we theoretically show the correctness and completeness of PT-ISABB in Sect. 5.1, and then prove that the lower bounds in PT-ISABB are at least as tight as the ones in AsymPT-FB when its maximal dimension limit $k = w^* + 1$ in Sect. 5.2. Finally, we analyze the complexity of PT-ISABB in Sect. 5.3.

5.1 Correctness

In this subsection, we first prove the termination and optimality, and further establish the completeness of PT-ISABB.

Lemma 1 *PT-ISABB will terminate after a finite number of iterations.*

Proof Directly from the pseudo codes, the inference phase will terminate since it only needs a linear number of messages. Thus, to prove the termination, it is enough to show that the same partial assignment cannot be explored twice in the search phase, i.e., an agent will not receive two identical *Cpa*s. Obviously, the claim holds for the root agent since it does not receive any CPA message. For an agent and a given *Cpa* from its parent, it will send several CPA messages to each child. Since each of them contains the different assignments of the agent (lines 24, 35, 39, 53, 67–70), the *Cpa*s sent to the child are all different. Therefore, the termination is hereby guaranteed. □

Lemma 2 *For an agent a_i and a given Cpa_i, $cost(Spa_i)$, the cost incurred by any assignment Spa_i to the sub-tree rooted at a_i with the assignment (x_i, d_i), is no smaller than the corresponding lower bound $lb_i(d_i)$. Here, $cost(Spa_i) = \sum_{a_l \in \{a_i\} \cup Desc(a_i)} \sum_{a_j \in AP(a_l)} f_{jl}(d_j, d_l) + f_{lj}(d_l, d_j)$ where d_l and d_j is the assignment of x_l and x_j in Cpa_i or Spa_i, respectively.*

Proof This lemma will be proved by the following two cases.

Case 1 If a_i is a leaf agent, $cost(Spa_i)$ can be rewritten by:

$$\begin{aligned} cost(Spa_i) &= \sum_{a_j \in AP(a_i)} f_{ij}(d_i, d_j) + f_{ji}(d_j, d_i) \\ &\geq \sum_{a_j \in AP(a_i)} f_{ij}(d_i, d_j) + \sum_{a_j \in AP(a_i)} \min_{x_i} f_{ji}(d_j, x_i) \\ &= \sum_{a_j \in AP(a_i)} f_{ij}(d_i, d_j) + EstRep_i(x_i)^- \\ &\leq lb_i(d_i) \end{aligned}$$

Case 2 If it is a non-leaf agent, a_i will replace the original lower bound with the actual cost reported by a_c after receiving a BACKTRACK message for d_i from $a_c \in C(a_i)$ (line 45). Thus, to prove the lemma, it is sufficient to show that the initial lower bound $lb_i^c(d_i)$ is no greater than the actual cost of $Spa_c, \forall a_c \in C(a_i)$, where $Spa_c \subset Spa_i$ is the assignment of the sub-tree rooted at a_c.

Consider the induction basis, i.e., a_i's children are leaf agents. For each child $a_c \in C(a_i)$, we have

$$
\begin{aligned}
cost(Spa_c) &= \sum_{a_j \in AP(a_c)} f_{jc}(d_j, d_c) + f_{cj}(d_c, d_j) \\
&\geq \min_{x_c} \left(\sum_{a_j \in AP(a_c)} f_{jc}(d_j, x_c) + \sum_{a_j \in AP(a_c)} f_{cj}(x_c, d_j) \right) \\
&= \min_{x_c} \left(f_{ic}(d_i, x_c) + \sum_{a_j \in AP(a_c)} f_{cj}(x_c, d_j) + \sum_{a_j \in PP(a_c)} f_{jc}(d_j, x_c) \right) \\
&\geq \min_{x_c} \left(f_{ic}(d_i, x_c) + \sum_{a_j \in AP(a_c)} f_{cj}(x_c, d_j) \right) + \sum_{a_j \in PP(a_c)} \min_{x_c} f_{jc}(d_j, x_c) \\
&\geq util_i^{c-}(Cpa_{i[Sep(a_c)]}, d_i) + \sum_{a_j \in PP(a_c)} \min_{x_c} f_{jc}(d_j, x_c) \\
&= util_i^{c-}(Cpa_{i[Sep(a_c)]}, d_i) + EstRep_i(x_c)^- \\
&= lb_i^c(d_i)
\end{aligned}
$$

The equation in the fourth to the fifth step holds when the maximal dimension limit $k \geq |Sep(a_i)| + 1$. Since a_c is a leaf agent, we have $Desc(a_c) = \emptyset$, and hence the equation in the sixth to the last step holds. Thus, the lemma holds for the basis.

Assume that the lemma holds for all $a_c \in C(a_i)$. Next, we are going to show the lemma holds for a_i as well. For each child $a_c \in C(a_i)$, we have

$$
\begin{aligned}
cost(Spa_c) &= \sum_{a_k \in \{a_c\} \cup Desc(a_c)} \sum_{a_j \in AP(a_k)} f_{jk}(d_j, d_k) + f_{kj}(d_k, d_j) \\
&= \sum_{a_j \in AP(a_c)} f_{cj}(d_c, d_j) + f_{jc}(d_j, d_c) + \sum_{a_k \in Desc(a_c)} \sum_{a_j \in AP(a_k)} f_{jk}(d_j, d_k) + f_{kj}(d_k, d_j) \\
&= \sum_{a_j \in AP(a_c)} f_{cj}(d_c, d_j) + f_{jc}(d_j, d_c) \\
&\quad + \sum_{a_{c'} \in C(a_c)} \sum_{a_k \in Desc(a_{c'}) \cup \{a_{c'}\}} \sum_{a_j \in AP(a_k)} f_{jk}(d_j, d_k) + f_{kj}(d_k, d_j) \\
&= \sum_{a_j \in AP(a_c)} f_{cj}(d_c, d_j) + f_{jc}(d_j, d_c) + \sum_{a_{c'} \in C(a_c)} cost(Spa_{c'}) \\
&\geq \sum_{a_j \in AP(a_c)} f_{cj}(d_c, d_j) + f_{jc}(d_j, d_c) + \sum_{a_{c'} \in C(a_c)} lb_c^{c'}(d_c) \\
&\geq f_{ic}(d_i, d_c) + \sum_{a_j \in AP(a_c)} f_{cj}(d_c, d_j) + \sum_{a_{c'} \in C(a_c)} lb_c^{c'}(d_c) + \sum_{a_j \in PP(a_c)} \min_{x_c} f_{jc}(d_j, x_c) \\
&= f_{ic}(d_i, d_c) + \sum_{a_j \in AP(a_c)} f_{cj}(d_c, d_j) + \sum_{a_{c'} \in C(a_c)} lb_c^{c'}(d_c) + EstRep_i(x_c)^- \\
&= f_{ic}(d_i, d_c) + \sum_{a_j \in AP(a_c)} f_{cj}(d_c, d_j) + \sum_{a_{c'} \in C(a_c)} \left(util_c^{c'-}(Cpa_{i[Sep(a_{c'})]}, d_c) \right. \\
&\quad \left. + \sum_{a_l \in Desc(a_{c'}) \cup \{a_{c'}\}} EstRep_i(x_l)^- \right) + EstRep_i(x_c)^-
\end{aligned}
$$

Since $Desc(a_c) = \{a_j | \forall a_j \in Desc(a'_c) \cup \{a'_c\}, \forall a_{c'} \in C(a_c)\}$, all the terms about the optimistic accumulated estimations can be combined. That is,

$$\begin{aligned} cost(Spa_c) &\geq f_{ic}(d_i, d_c) + \sum_{a_j \in AP(a_c)} f_{cj}(d_c, d_j) + \sum_{a_{c'} \in C(a_c)} util_c^{c'-}(Cpa_{i[Sep(a_{c'})]}, d_c) \\ &\quad + \sum_{a_l \in Desc(a_c) \cup \{a_c\}} EstRep_i(x_l)^- \\ &\geq \min_{x_c} \left(f_{ic}(d_i, d_c) + \sum_{a_j \in AP(a_c)} f_{cj}(x_c, d_j) + \sum_{a_{c'} \in C(a_c)} util_c^{c'-}(Cpa_{i[Sep(a_{c'})]}, x_c) \right) \\ &\quad + \sum_{a_l \in Desc(a_c) \cup \{a_c\}} EstRep_i(x_l)^- \\ &= util_i^{c-}(Cpa_{i[Sep(a_c)]}, d_i) + \sum_{a_l \in Desc(a_c) \cup \{a_c\}} EstRep_i(x_l)^- \\ &= lb_i^c(d_i) \end{aligned}$$

which establishes the lemma. □

Lemma 3 *For an agent a_i and a given Cpa_i, the cost incurred by any assignment Spa_i to the sub-tree rooted at a_i with the assignment (x_i, d_i), is no greater than the corresponding upper bound $ub_i(d_i)$.*

Proof To avoid redundancy, we omit the full proof for this lemma, which is very similar to the proof of Lemma 2. The main difference is that the minimal cost tables and optimistic accumulated estimations are replaced by the maximal cost tables and pessimistic accumulated estimations. □

Proposition 1 *For an agent a_i and a given Cpa_i, the cost incurred by the optimal sub-problem assignment $Spa_i(d_i^*)$ equals a_i's lower bound LB_i and upper bound UB_i, i.e., $LB_i = cost(Spa_i(d_i^*)) = UB_i$.*

Proof Based on line 75, the condition for a_i reporting $Spa_i(d_i^*)$ is $LB_i = UB_i$. Further, according to lines 76–77 and Eq. (21), we can conclude that $ub_i(d_i^*) = UB_i = LB_i$. Since $lb_i(d_i^*) \geq \max_{d_i \in D_i} lb_i(d_i) = LB_i$, $ub_i(d_i^*) \leq \min_{d_i \in D_i} ub_i(d_i) = UB_i$ and $lb_i(d_i^*) \leq ub_i(d_i^*)$ by Eqs. (18–21), we have $ub_i(d_i^*) = lb_i(d_i^*) = LB_i = UB_i$. Besides, it has been proved in Lemmas 2 and 3 that $lb_i(d_i^*) \leq cost(Spa_i^*)$ and $cost(Spa_i^*) \leq ub_i(d_i^*)$. Thus, we have $LB_i = cost(Spa_i(d_i^*)) = UB_i$ and the proposition is proved. □

Lemma 4 *For an agent a_i and a given Cpa_i, any assignment to the sub-tree rooted at a_i with cost greater than the minimal of TH_i and UB_i cannot be a part of a solution with cost smaller than the global upper bound.*

Proof We will prove recursively by showing that for a partial assignment Spa_i to the sub-tree rooted at a_i with $cost(Spa_i) > \min(TH_i, UB_i)$, any partial assignment $Spa_j \supset Spa_i$ to the sub-tree rooted at a_j will have $cost(Spa_j) > \min(TH_j, UB_j)$ where $a_j = P(a_i)$. Note that TH_i is a threshold received from a_j via a CPA message (line 19) and it has been proved in Proposition 1 that $UB_i = cost(Spa_i)$ when a_i backtracks to a_j. Thus, a_i cannot backtrack by

reporting Spa_i with its cost greater than TH_i since there must exist a better partial assignment whose cost smaller than TH_i. If TH_i is received from a_j, according to line 72, we have

$$\min(TH_j, UB_j) = TH_i + high\_cost_j(d_j) + \sum_{a_c \in C(a_j) \wedge c \neq i} lb_j^c(d_j)$$

Thus, $cost(Spa_i) > \min(TH_i, UB_i)$ necessarily means that any partial assignment $Spa_j \supset Spa_i$ will have $cost(Spa_j) > \min(TH_j, UB_j)$. □

Theorem 1 *PT-ISABB is complete.*

Proof Immediately from Lemmas 1–4, the algorithm will terminate and all pruned assignments are suboptimal. Thus, PT-ISABB is complete. □

5.2 Lower bound tightness

Property 1 *For an agent a_i and a given Cpa_i, the initial lower bound $lb_i^c(d_i)$ of $a_c \in C(a_i)$ for d_i is at least as tight as the one in AsymPT-FB when the maximal dimension limit equals $w^* + 1$. Here, w^* is the induced width of the pseudo tree.*

Proof In AsymPT-FB, the lower bound for a_c after receiving all the **LB_Report** messages from agents in the subtree rooted at a_c is given by the sum of the best single-side local costs of a_c's descendants under Cpa_i. That is,

$$SubtreeLB_i^c(d_i) = \sum_{a_j \in Desc(a_c)} \min_{x_j} \sum_{a_l \in Sep(a_c) \cap PP(a_j)} f_{jl}(x_j, d_l) + \min_{x_c} \sum_{a_l \in AP(a_c)} f_{cl}(x_c, d_l)$$

For the sake of clarity, we denote the vector of x_c and its descendant variables as $\mathbf{x_c}$. Next, we will show $lb_i^c(d_i) \geq SubtreeLB_i^c(d_i)$. Since $k = w^* + 1$, the inference phase does not drop any dimension. Thus, we have

$$\begin{aligned}
lb_i^c(d_i) &= util_i^{c-}(Cpa_{i[Sep(a_c)]}, d_i) + \sum_{a_j \in Desc(a_c) \cup \{a_c\}} EstRep_i(x_j)^- \\
&\geq util_i^{c-}(Cpa_{i[Sep(a_c)]}, d_i) \\
&= \min_{\mathbf{x_c}} \sum_{a_j \in Desc(a_c)} \left(\sum_{a_l \in AP(a_j) \cap Sep(a_c)} f_{jl}(x_j, d_l) + \sum_{a_l \in AP(a_j) \cap Desc(a_i)} f_{jl}(x_j, x_l) \right. \\
&\quad \left. + \sum_{a_l \in C(a_j)} f_{jl}(x_j, x_l) + \sum_{a_l \in PC(a_j)} \min_{x_l} f_{jl}(x_j, x_l) \right) + \sum_{a_l \in AP(a_c)} f_{cl}(x_c, d_l) \\
&\quad + f_{ic}(d_i, x_c) + \sum_{a_l \in C(a_c)} f_{cl}(x_c, x_l) + \sum_{a_l \in PC(a_c)} \min_{x_l} f_{cl}(x_c, x_l) \\
&\geq \min_{\mathbf{x_c}} \sum_{a_j \in Desc(a_c)} \sum_{a_l \in AP(a_j) \cap Sep(a_c)} f_{jl}(x_j, d_l) + \sum_{a_l \in AP(a_c)} f_{cl}(x_c, d_l)
\end{aligned}$$

Since $x_c \in \mathbf{x_c}$ and $AP(a_j) \supset PP(a_j)$, the right-hand side of the inequality in the last step can be further reduced. Thus, we have

$$\begin{aligned} lb_i^c(d_i) &\geq \min_{\mathbf{x_c} \setminus x_c} \sum_{a_j \in Desc(a_c)} \sum_{a_l \in AP(a_j) \cap Sep(a_c)} f_{jl}(x_j, d_l) + \min_{x_c} \sum_{a_l \in AP(a_c)} f_{cl}(x_c, d_l) \\ &\geq \min_{\mathbf{x_c} \setminus x_c} \sum_{a_j \in Desc(a_c)} \sum_{a_l \in PP(a_j) \cap Sep(a_c)} f_{jl}(x_j, d_l) + \min_{x_c} \sum_{a_l \in AP(a_c)} f_{cl}(x_c, d_l) \\ &\geq \sum_{a_j \in Desc(a_c)} \min_{x_j} \sum_{a_l \in Sep(a_c) \cap PP(a_j)} f_{jl}(x_j, d_l) + \min_{x_c} \sum_{a_l \in AP(a_c)} f_{cl}(x_c, d_l) \\ &= SubtreeLb_i^c(d_i) \end{aligned}$$

which concludes the property. □

5.3 Complexity

Since an agent a_i stores $util_i^{c\pm}$, $lb_i^c(d_i)$ and $ub_i^c(d_i)$ for each child and the bounds including $lb_i(d_i), ub_i(d_i), LB_i$ and UB_i, the overall space complexity is $O(|C(a_i)|d_{max}^{\min(k,|Sep(a_i)|+1)} + (|C(a_i)|+1)|D_i|)$ where $d_{max} = \max_{a_j \in Sep(a_i)} |D_j|$. Since the dimension size of each outgoing COST message is no greater than k, the size of an COST message from a_i is $O(d_{max}^{\min(k,|Sep(a_i)|+1)})$. For a CPA message, it consists of the assignment of each agent, a threshold and all the optimistic and pessimistic accumulated estimations of the descendants of that agent. Thus, the size of a CPA message is $O(|A|)$. Other messages including RESD_REQ, RESD, BACKTRACK, STOPEXPLORE and TERMINATE carry several scalars and thus they only require $O(1)$ space.

Different from standard DPOP/ADPOP, PT-ISABB only requires $|A|-1$ messages in the inference phase since it does not have the value propagation phase. Like any other search-based complete algorithms, the search phase produces $d_{max}^{|A|}$ messages where $d_{max} = \max_{a_j \in A} |D_j|$.

6 Bounded error approximations

In this section, we present two suboptimal variants of PT-ISABB to trade solution quality for coordination overheads, where the solution cost is bounded by a user-specified error. More specifically, we adapt PT-ISABB as its suboptimal versions with the Absolute Error Mechanism of ADOPT [26] and Relative Error Mechanism of BnB-ADOPT [40]. Here, the Absolute Error Mechanism allows users to specify an absolute error bound on the solution cost while the Relative Error Mechanism allows users to specify a relative error bound.

To implement the Absolute Error Mechanism and Relative Error Mechanism on top of asynchronous search-based complete algorithms (e.g., ADOPT and BnB-ADOPT), only the root agent makes the change to relax the termination condition. However, this strategy is not suitable for our case since the root agent cannot update its lower bounds and upper bounds quickly and each agent backtracks only if it has found the solution to its subproblem. Thus, to adapt these two error mechanisms to PT-ISABB, each agent $a_i \in A$ needs to maintain the error bound gap_i to relax its backtracking condition (lines 21, 31 and 46). That is,

$$LB_i \geq \min(UB_i - gap_i, TH_i) \tag{27}$$

Besides, several necessary modifications have to be made in the BACKTRACK message propagation and disposition.

- When a_i sends a **BACKTRACK**, the condition that claims Cpa_i to be feasible changes from $LB_i = UB_i \wedge LB_i < TH_i$ to $LB_i \geq UB_i - gap_i \wedge LB_i < TH_i$. Besides, the content forwarded to its parent changes from $(ub_i(d_i^*), Spa_i(d_i^*))$ to $(lb_i(d_i^*), ub_i(d_i^*), Spa_i(d_i^*))$ if Cpa_i is feasible. Otherwise, the context changes from $(\infty, \emptyset)$ to $(\infty, \infty, \emptyset)$ (line 75-79).
- When a_i receives a **BACKTRACK** from a_c, the bound update process changes from $lb_i^c(d_i) \leftarrow \max(lb_i^c(d_i), opt_c)$ and $ub_i^c(d_i) \leftarrow \min(ub_i^c(d_i), opt_c)$ to $lb_i^c(d_i) \leftarrow \max(lb_i^c(d_i), lb_c)$ and $ub_i^c(d_i) \leftarrow \min(ub_i^c(d_i), ub_c)$ (line 45).

Besides, it is noteworthy that the two suboptimal version of PT-ISABB could terminate without reaching any complete solution when the gap is sufficiently large (i.e., $gap_i \geq UB_i - LB_i$). The issue can be voided by assuming that each agent $a_j \in A$ holds a default value d_j^0 for its variable x_j at the beginning of the algorithm. When a very large gap_i is assigned for a non-root agent a_i, a_i backtracks with the assignment Spa_i^0 where $Spa_i^0 = \bigcup_{a_j \in Desc(a_i)} \{x_j, d_j^0\} \cup \{x_i, d_i^*\}$ and $d_i^* = \arg\min_{d_i \in D_i} ub_i(d_i)$.

Lemma 5 *For a non-root agent a_i, given $gap_i \geq UB_i - LB_i$, Spa_i^0 satisfies $LB_i \leq cost(Spa_i^0) \leq LB_i + gap_i$.*

Proof According to Lemmas 2 and 3, it can be concluded that $cost(Spa_i^0) \geq lb_i(d_i^*)$ and $cost(Spa_i^0) \leq ub_i(d_i^*)$ where $d_i^* = \arg\min_{d_i \in D_i} ub_i(d_i)$. Further, based on Eqs. (20) and (21), we have $lb_i(d_i^*) \geq LB_i$ and $ub_i(d_i^*) = UB_i$. Thus, it can be concluded that $LB_i \leq cost(Spa_i^0) \leq UB_i$. Since $gap_i \geq UB_i - LB_i$, we have $LB_i \leq cost(Spa_i^0) \leq LB_i + gap_i$. □

6.1 Absolute error Mechanism

The absolute error mechanism of ADOPT requires users to specify an absolute error bound b ($0 \leq b < \infty$) such that the difference between the solution cost and the optimal solution cost is no greater than b. However, for PT-ISABB, b cannot be directly applicable to relax the backtracking condition for non-root agents since their optimal subproblem solution costs are strictly less than the optimal solution cost. Thus, based on the inference results produced during the inference phase, we introduce an *absolute error bound allocation* mechanism to compute an absolute error bound b_i for each agent a_i. Specifically, b_i is equal to the user-defined bound b if a_i is the root agent. Otherwise, b_i is a partition of its parent's error bound $b_{P(a_i)}$ and received from its parent $P(a_i)$ along with the partial assignment Cpa_i via a CPA message. That is,

$$b_i = b_{P(a_i)} * \left(\frac{v_i}{\sum_{a_c \in C(P(a_i))} v_c} \right) \tag{28}$$

where $v_i \in \{util^{i-}_{P(a_i)}(Cpa_i), util^{i+}_{P(a_i)}(Cpa_i), util^{i+}_{P(a_i)}(Cpa_i) - util^{i-}_{P(a_i)}(Cpa_i)\}$. The intuition behind Eq. (28) is that the allocation of $b_{P(a_i)}$ to each of its children $a_j \in C(P(a_i))$ should be fair. This can be achieved by considering that the allocated error bounds of a_j should be

Table 5 The PT-ISABB variants

Algorithm	Variable elimination scheme	Cost table propagation scheme
PT-ISABB-A	Local elimination	Min cost table
PT-ISABB-B	Local elimination	Both Min and Max cost tables
PT-ISABB-C	Non-local elimination	Min cost table
PT-ISABB-D	Non-local elimination	Both Min and Max cost tables

proportional to the actual cost of a_j's subproblem. Here, we adopt the cost tables obtained in the inference phase to approximate the actual cost which is unknown before a_j reports that cost. Then, the absolute error mechanism can be adapted to PT-ISABB easily by setting a_i's error bound gap_i as follows:

$$gap_i = b_i. \tag{29}$$

We name the suboptimal variant of PT-ISABB with the absolute error mechanism PT-ISABB$_{AEM}$. In PT-ISABB$_{AEM}$, the root agent a_i terminates the search process once the gap between its upper bound and lower bound is no greater than the user-specified absolute bound (i.e., $UB_i - LB_i \leq b$). As a result, PT-ISABB$_{AEM}$ terminates with a solution cost ($cost_i$) that meets $LB_i \leq cost_i \leq LB_i + b$.

6.2 Relative error mechanism

Rather than specifying an absolute error bound on the solution cost, the relative error Mechanism of BnB-ADOPT allows users to specify a relative error bound ρ ($1 \leq \rho < \infty$) such that the solution cost should be at most ρ times larger than the optimal solution cost. This mechanism can be deployed to PT-ISABB by assigning the variable gap_i as follows:

$$gap_i = (\rho - 1) \cdot LB_i. \tag{30}$$

We name the suboptimal variant of PT-ISABB with the relative error mechanism PT-ISABB$_{REM}$. In PT-ISABB$_{REM}$, the root agent a_i terminates the search process once the gap between its upper bound and lower bound is no greater than the error bound gap_i (i.e., $UB_i - LB_i \leq gap_i = (\rho - 1) \cdot LB_i$). PT-ISABB$_{REM}$ terminates with a solution cost ($cost_i$) that satisfies $LB_i \leq cost_i \leq \rho \cdot LB_i$.

Proposition 2 *For an agent a_i and a given Cpa_i in the two suboptimal variants of PT-ISABB (PT-ISABB$_{AEM}$ and PT-ISABB$_{REM}$), the cost incurred by a complete subproblem assignment $Spa_i(d_i^*)$ is no smaller than LB_i and no greater than $LB_i + gap_i$. That is, $LB_i \leq cost(Spa_i(d_i^*)) \leq LB_i + gap_i$.*

Proof According to lines 76–77, Eqs. (20) and (21), we have $ub_i(d_i^*) = \min_{d_i \in D_i} ub_i(d_i) = UB_i$ and $lb_i(d_i^*) \geq \min_{d_i \in D_i} lb_i(d_i) = LB_i$. As it has been proved in Lemmas 2 and 3 that $lb_i(d_i^*) \leq cost(Spa_i(d_i^*))$ and $cost(Spa_i(d_i^*)) \leq ub_i(d_i^*)$ respectively, we have $LB_i \leq cost(Spa_i(d_i^*)) \leq UB_i$. Since $LB_i \geq \min(UB_i - gap_i, TH_i)$ and $LB_i < TH_i$ by Eq. (27) and the condition of finding $Spa_i(d_i^*)$ (line 75) respectively, we have $LB_i \geq UB_i - gap_i$. Thus, we can conclude that $LB_i \leq cost(Spa_i(d_i^*)) \leq LB_i + gap_i$. □

Theorem 2 *The two suboptimal variants of PT-ISABB (PT-*ISABB$_{AEM}$ *and PT-*ISABB$_{REM}$*) will terminate when* $LB_i \leq cost(Spa_i(d_i^*)) \leq LB_i + gap_i$ *for the root agent* a_i*, where* $cost(Spa_i(d_i^*))$ *is the cost incurred by the solution* $Spa_i(d_i^*)$*.*

Proof Immediately from Proposition 2, the solution $Spa_i(d_i^*)$ satisfies $LB_i \leq cost(Spa_i(d_i^*)) \leq LB_i + gap_i$ when meeting the termination condition. □

7 Experimental evaluations

In this section, to show the strength of non-local elimination and propagating both maximal and minimal cost tables, we first conduct an ablation study by comparing the four different optimal versions of PT-ISABB which are detailed in Table 5. Then, we empirically evaluate PT-ISABB and state-of-the-art search-based complete algorithms for ADCOPs including SyncABB-1ph, ATWB and AysmPT-FB on random ADCOPs, scale-free networks and asymmetric MaxDCSPs. To demonstrate the real power of tight bounds established by the inference phase, we also consider SyncABB-1ph on a pseudo-tree (PT-SABB). Finally, we compare the two suboptimal variants of PT-ISABB to each other. All the evaluated algorithms are implemented in DCOPSovler,[5] the DCOP simulator developed by ourselves. All the reported experimental results are averaged over 50 randomly generated instances. The experiments are conducted on an i7-7820x workstation with 32 GB of memory, and we set the timeout to 15 min for each algorithm.

7.1 Experimental configurations

We benchmark the algorithms with three types of problems, i.e., random ADCOPs, scale-free networks and random asymmetric MaxDCSPs.

- *Random ADCOPs* are an asymmetric version of random DCOPs, a general form of the distributed constraint optimization problems where a set of agents are constrained with each other randomly [6]. In the experiment, the number of agents and graph density are varied to evaluate the performance of the complete algorithms (see the detailed configurations in Sects. 7.4 and 7.5). Additionally, the constraint costs are uniformly selected from [0, 100].
- *Scale-free networks* [1] are networks whose degree distributions follow power laws. In the experiment, we use Barabási–Albert (BA) model to generate the constraint graph topology where we set the domain size to 3, the agent number to 16, and an initial agent number to 10 (i.e., $m_0 = 10$). At each iteration of BA model procedure, a new agent is connected to m_1 other agents with a probability proportional to the number of links that the existing agents already have, where m_1 varies from 2 to 10. The range of constraint costs in scale-free networks are the same as the ones in random ADCOPs.
- *Asymmetric MaxDCSPs* are an asymmetric version of MaxDCSPs, a subset of random DCOPs where all the constraint costs are equal to one [15, 26]. Asymmetric MaxDCSPs are classified by the agent number, domain size, graph density and constraint tightness (i.e., the probability for the emergence of a non-zero cost among two value assign-

[5] https://github.com/czy920/DCOPSovler.

Table 6 Performance comparison of the PT-ISABB variants given $k = 2$ force random ADCOPs ($8 \leq |A| \leq 18, |D_i| = 3$ and $p = 0.25$)

Agent number	Algorithm	Message number	Networkload (KB)	NCLOs	Runtime (ms)
8	A	370.35	3.73	237.19	26.62
	B	354.50	3.80	315.81	33.38
	C	216.92	2.16	118.81	15.58
	D	**56.00**	**0.55**	**85.65**	**3.46**
10	A	1157.69	13.01	656.15	91.42
	B	1135.31	13.18	812.54	102.35
	C	732.31	8.24	**367.58**	62.15
	D	**576.58**	**6.67**	371.27	**46.31**
12	A	3928.88	46.30	1690.88	312.42
	B	3893.31	46.56	1904.35	356.15
	C	2971.73	35.44	**1193.92**	246.50
	D	**2807.65**	**33.60**	1249.50	**232.35**
14	A	17,771.62	228.38	6851.31	1732.27
	B	17,768.42	228.89	7138.12	1887.35
	C	12,913.15	165.28	**4603.42**	1211.38
	D	**12,677.65**	**162.57**	4659.73	**1083.35**
16	A	205,709.62	2798.58	65,622.88	18,202.08
	B	205,654.12	2799.26	66,009.46	20,340.62
	C	172,231.27	2342.58	53,152.12	16,387.31
	D	**171,273.50**	**2327.88**	**53,150.50**	**14,779.04**
18	A	1,184,726.46	16,150.03	307,890.77	97,720.81
	B	1,184,047.15	16,143.35	308,208.92	98,058.62
	C	898,321.54	12,193.97	230,693.65	76,554.12
	D	**886,177.60**	**12,089.82**	**229,888.36**	**69,072.92**

The best results are shown in bold

ments). In the experiment, we consider asymmetric MaxDCSPs with 10 agents, the domain size of 10 and the graph density of 0.4, and the tightness varies from 0.1 to 0.8.

7.2 Performance metrics

- *Non-Concurrent Logical Operations (NCLOs)* [28] is a generalization of NCCCs [43] based on the concept of atomic operations [14], where a basic operation is not necessarily a constraint check in the search-based complete algorithms. In the experiment, we use NCLOs as a metric to evaluate hardware-independent runtime where the basic operations are accesses to cost tables for the inference phase and constraint checks for the search phase and other competitors.
- *The message number* is the total number of messages sent by all agents. In the experiment, we use the message number as a metric to measure the communication costs incurred by an algorithm.
- *The size of total information exchanged* is the total size of messages exchanged during the execution of an algorithm. Since the size of a message is linear to the agent number in the search phase and other competitors but exponential in its dimension

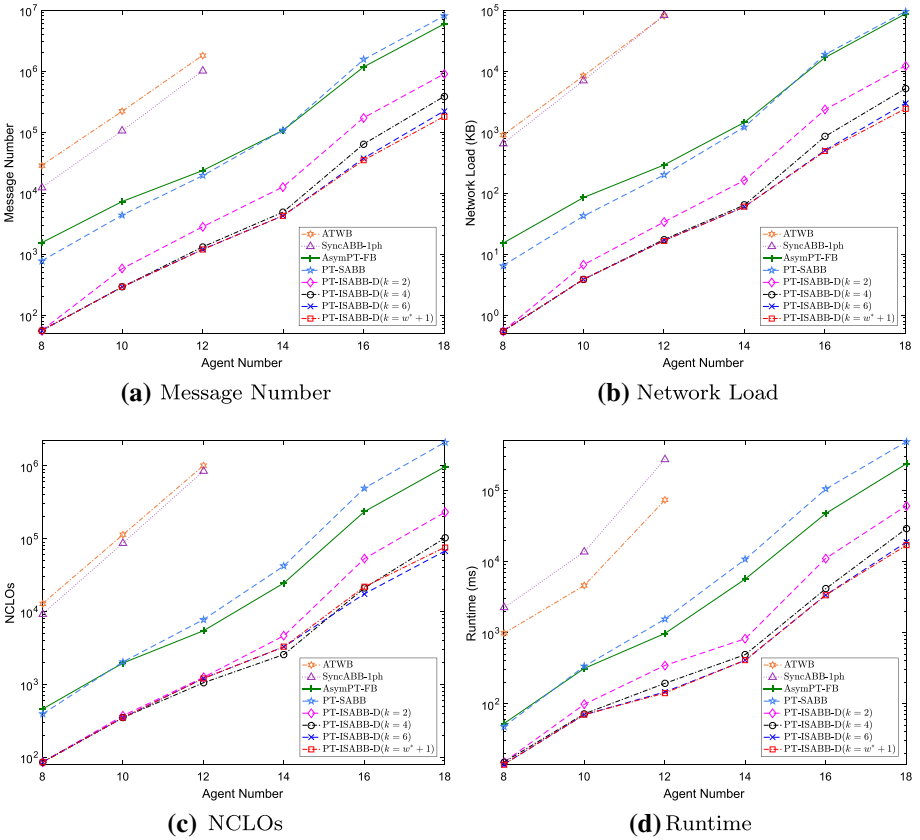
Table 7 Performance comparison of the PT-ISABB variants given $k = w^* + 1$ for random ADCOPs ($8 \leq |A| \leq 18, |D_i| = 3$ and $p = 0.25$)

Agent number	Algorithm	Message number	Networkload (KB)	NCLOs	Runtime (ms)
8	A	370.27	3.73	236.77	29.38
	B	358.88	3.80	315.85	35.50
	C	217.81	2.16	119.04	22.58
	D	**56.00**	**0.55**	**85.65**	**3.81**
10	A	1137.85	12.71	660.85	91.27
	B	1117.19	12.96	837.69	106.15
	C	556.35	6.39	**341.77**	54.85
	D	**307.35**	**4.14**	372.54	**24.12**
12	A	3409.92	39.34	1687.00	266.35
	B	3378.35	40.09	2135.46	316.42
	C	1570.69	19.22	**933.35**	139.58
	D	**1193.23**	**16.67**	1210.04	**101.62**
14	A	13,699.42	172.78	5890.38	1236.58
	B	13,576.15	174.28	6867.81	1497.62
	C	4649.85	59.97	**2450.23**	468.85
	D	**4266.62**	**59.50**	3279.46	**388.04**
16	A	101,986.73	1305.41	36,308.08	8671.31
	B	101,815.38	1312.97	42,172.08	9695.62
	C	35,529.04	471.18	**16,186.73**	3402.88
	D	**34,622.38**	**485.18**	21,888.38	**3144.19**
18	A	518,513.23	6,685.46	162,332.73	36,623.77
	B	518,332.46	6730.11	197,276.46	42,277.85
	C	173,486.27	**2338.11**	**77,594.15**	14,518.19
	D	**172,672.69**	2468.63	112,518.50	**13,977.42**

The best results are shown in bold

size in the inference, we use the size of total information exchanged as an additional metric to evaluate the communication costs in the experiment.

- *Runtime* measures the simulated execution time of a distributed algorithm. In more detail, we simulate the distributed environment by JAVA multi-thread mechanism, where each agent is simulated by a thread. The runtime is considered as the difference between the time stamp when the first agent starts and the one when the last agent terminates.
- *Entropy* [2] is a metric to quantify the privacy loss in distributed constraints satisfaction problems, measuring the amount of the missing information about each agent's local privacy constraints. In the experiment, we adopt the method in [15, 23] to calculate this metric when solving asymmetric MaxDCSPs.
- *The normalized cost* is the ratio of the solution cost to the optimal solution cost. We use the metric to quantify the solutions of the two suboptimal variants of PT-ISABB in the experiment.



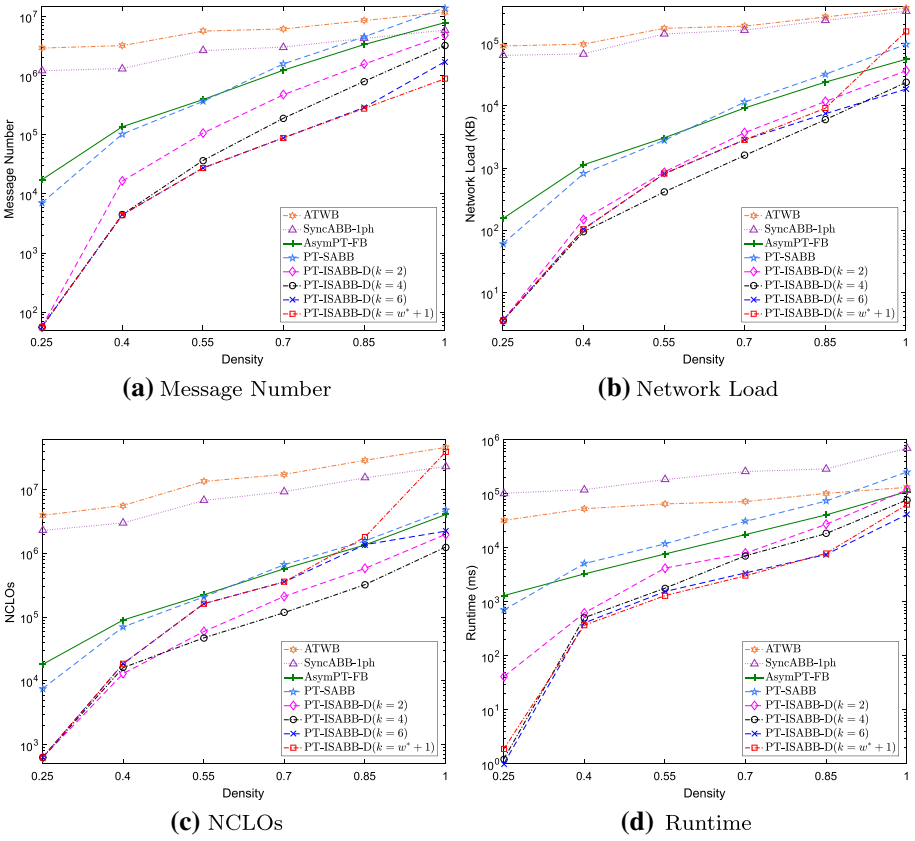
(a) Message Number **(b)** Network Load

(c) NCLOs **(d)** Runtime

Fig. 5 Performance comparison for random ADCOPs ($8 \leq |A| \leq 18, |D_i| = 3$ and $p = 0.25$)

7.3 Performance comparisons: the optimal variants of PT-ISABB

Tables 6 and 7 present the performance of the four optimal variants of PT-ISABB given $k = 2$ and $k = w^* + 1$ on the random ADCOPs where we set the graph density to 0.25, the domain size to 3 and vary the agent number from 8 to 18. It can be seen that given a fixed k, the variants with non-local elimination (i.e., PT-ISABB-C and PT-ISABB-D) exhibit great advantages over these variants with local elimination (i.e., PT-ISABB-A and PT-ISABB-B) on all the evaluation metrics. This is particularly prominent when there is no limit on the memory budget. In more detail, the improvement of PT-ISABB-C and PT-ISABB-D over PT-ISABB-A and PT-ISABB-B is about 20–40% when $k = 2$ while the improvement is widened to 50–65% when $k = w^* + 1$. These phenomena indicate that in PT-ISABB, the inference phase with non-local elimination can provide tighter lower bound, and hereby lead to efficient pruning and great reduction on the coordination overheads incurred by the search phase. Additionally, PT-ISABB-A share the similar performance with PT-ISABB-B while PT-ISABB-D outperforms PT-ISABB-C in terms of the message number, network load and runtime. That is because the gap in PT-ISABB-D between the upper bounds and lower bounds is smaller than the one in PT-ISABB-C. As a consequence, each agent in PT-ISABB-D can exploit the gap effectively to reduce the search efforts.



(a) Message Number **(b)** Network Load

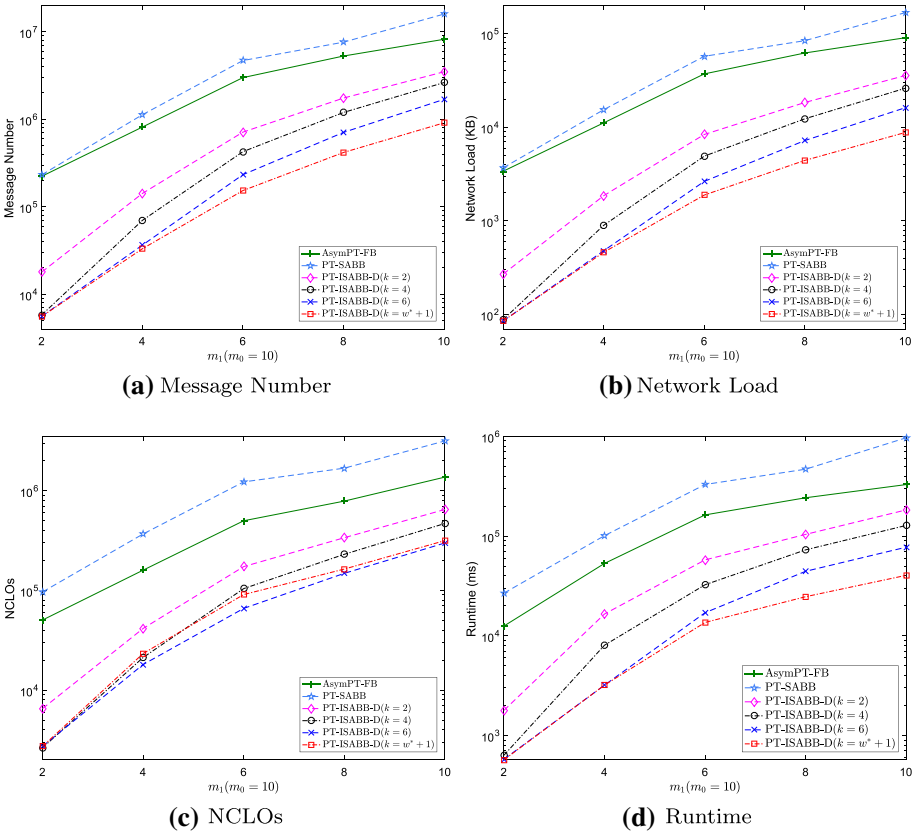
(c) NCLOs **(d)** Runtime

Fig. 6 Performance comparison for random ADCOPs ($|A| = 8, |D_i| = 8$ and $0.25 \leq p \leq 1$)

7.4 Performance comparisons: PT-ISABB and its competitors

For random ADCOPs, we firstly set the graph density to 0.25, the domain size to 3 and vary the agent number from 8 to 18. Figure 5 presents the performance comparison on different agent numbers. The average induced widths in the experiments are 1–6.84. It can be observed that as the agent number grows, both the communication and computation overheads of all the algorithms increase exponentially. Among them, our algorithms exhibit great superiorities on all the metrics, which indicates the merit of the hybrid execution of inference and search. Moreover, PT-ISABB-D produces significantly lower overheads than AsymPT-FB even with a small dimension limit k (e.g., $k = 2$). In more detail, PT-ISABB-D ($k = 2$) has an advantage over AsymPT-FB by about 88% in the message number and network load and 75% in the NCLOs and runtime. That is due to the fact that PT-ISABB-D does not rely on forward bounding which is expensive in message-passing to compute lower bounds. Also, this phenomenon implies that our algorithm can produce tighter lower bounds even if the memory budget is relatively low.

Figure 6 gives the results under different graph densities. Specifically, we consider the random ADCOPs with 8 agents, the domain size of 8 and the graph density varying from 0.25 to 1. The average induced widths here are 1–6. Note that in this configuration, the



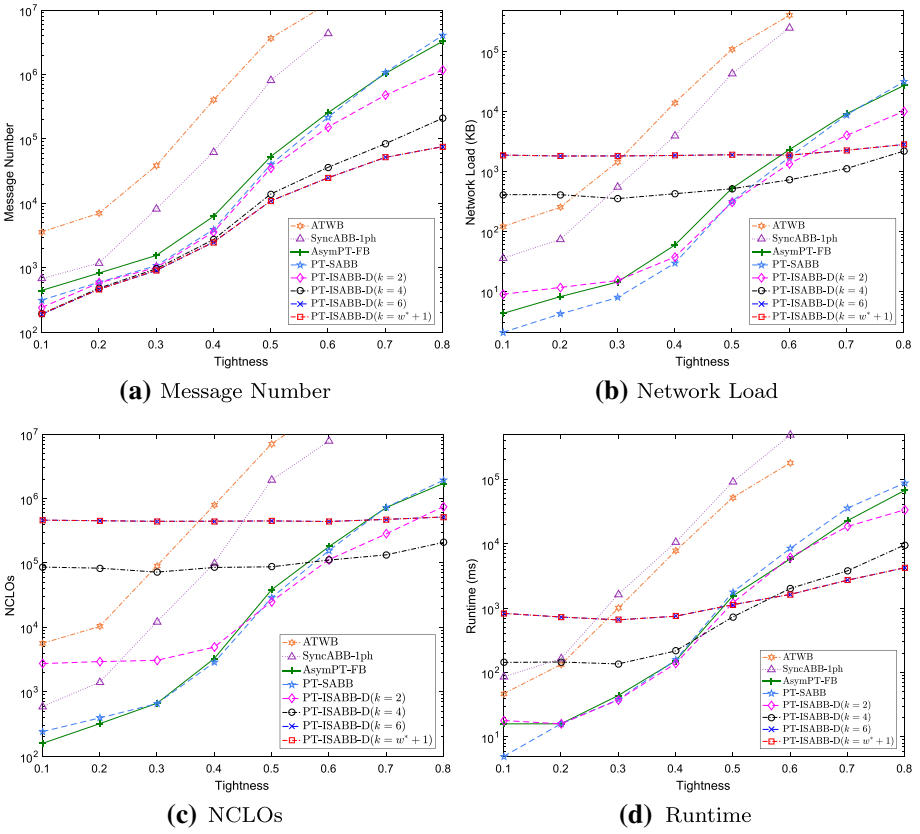
(a) Message Number **(b)** Network Load

(c) NCLOs **(d)** Runtime

Fig. 7 Performance comparison on scale-free networks ($|A| = 16, |D_i| = 3, m_0 = 10$ and $2 \leq m_1 \leq 10$)

size of the search space does not change and the complexity is reflected in the topologies. It can be seen from the figure that all the tree-based algorithms have good performance on the sparse problems, and the advantages gradually diminish as the density increases. That is because the density would affect the construction of a pseudo tree and thus influence the parallelism of search processes. Specifically, the dense problems usually result in pseudo trees with low branching factors, making those tree-based algorithms require more messages than SyncABB-1ph. Even so, our proposed PT-ISABB-D still outperforms SyncABB-1ph when the problems are fully connected, which demonstrates the necessity of tighter lower bounds. As for SynchABB-1ph, it outperforms over ATWB on all the metrics except the runtime. That is due to the fact that the two-side cost accumulation is achieved by sending back the *Cpa* to all the assigned agents sequentially in SynchABB-1ph, while it is implemented by sending the copy of *Cpa* to all the assigned agents simultaneously in ATWB. Additionally, PT-ISABB-D with different k performs similarly on the sparse problems ($p < 0.4$), but the performance varies a lot on the dense problems ($p \geq 0.4$). That is because the small induced width of the pseudo tree results in a small set of dimensions that PT-ISABB would drop during the inference phase when solving a sparse problem.

Figure 7 shows the results on the scale-free networks with different m_1 where the average induced widths in the experiments are 4.5–7.6. We do not include SynchABB-1ph and



(a) Message Number **(b)** Network Load

(c) NCLOs **(d)** Runtime

Fig. 8 Performance comparison for asymmetric MaxDCSPs ($|A| = 10$, $|D_i| = 10$, $p_1 = 0.4$ and $0.1 \leq p_2 \leq 0.8$)

AWTB here since they cannot solve the problems even when $m_1 = 2$ within 15 minutes. It can be concluded from the figure that PT-ISABB-D outperforms the other competitors, and the superiorities are widened as the memory budget grows. Besides, although AsymPT-FB is superior to PT-SABB over the NCLOs and runtime, they perform similarly in terms of the message number and network load. This phenomenon confirms the fact that forwarding bounding itself could be expensive in message-passing despite the ability of building tight lower bounds.

Figure 8 presents the results when solving asymmetric MaxDCSPs with different tightness as the average induced width is 3.92. This configuration neither increases the search space nor affects the topologies, but instead increases the difficulty of pruning. It can be seen that all the algorithms except ATWB produce few messages when the tightness is low. That is because the algorithms can find low upper bounds very quickly to prune most of the search space on these problems. With the growth of tightness, the number of prohibited combinations increases and the algorithms can no longer find high-quality upper bounds promptly. As a result, the algorithms require much more search efforts to exhaust the search space. Note that SyncABB-1ph and ATWB perform poorly and can only solve the problems with the tightness up to 0.6, which can attribute to their inability of accelerating

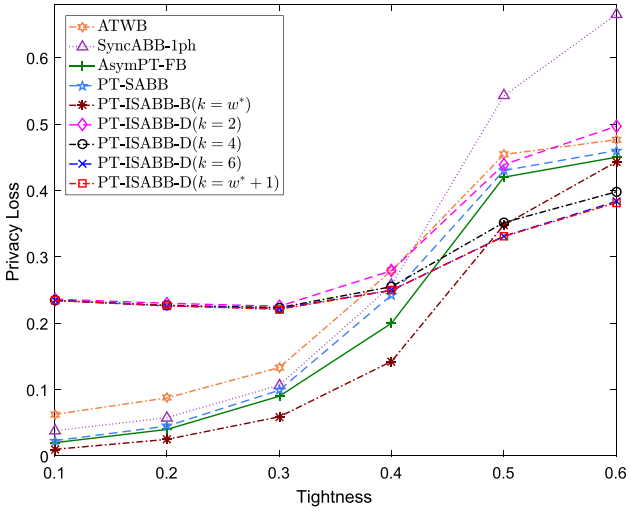
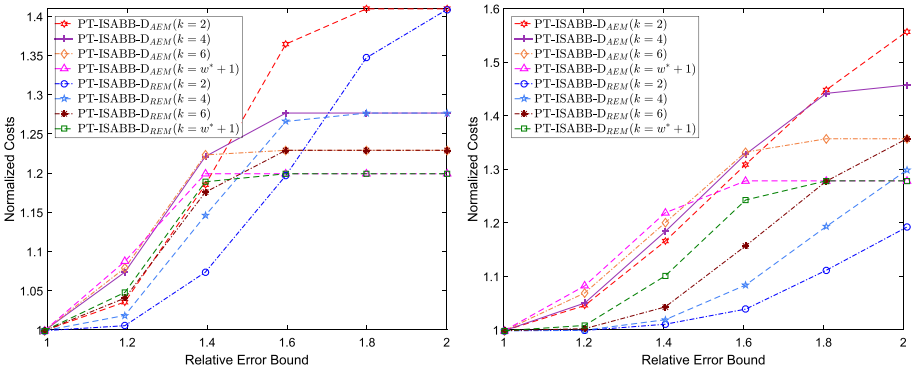


Fig. 9 Privacy loss for asymmetric MaxDCSPs ($|A| = 10, |D_i| = 10, p_1 = 0.4$ and $0.1 \leq p_2 \leq 0.8$)

the search process by exploiting topologies. On the other hand, the tree-based algorithms divide a problem to several smaller subproblems at each branching agent and search the subproblems in parallel. Among them, our proposed PT-ISABB-D with $k \geq 4$ incurs much smaller overheads, which demonstrates the effectiveness of the inference phase in computing tighter lower bounds. In other words, although the lower bounds produced by PT-ISABB-D are only proved to be as tight as the ones of AsymPT-FB when $k = w^* + 1$ according to Property 1, the memory consumption for computing such lower bounds is much less in practice. Additionally, it can be seen from the figure that PT-SABB-D incurs smaller communication overheads than AsymPT-FB when solving the problems with low tightness, which demonstrates forward bounding is expensive in message-passing again. And it is no surprise to find PT-ISABB with large k requires much more network load, NCLOs and runtime than the other competitors when solving problems with low tightness. The reason is that the inference on problems with large domain sizes would be quite expensive, whereas search-based algorithms can find a feasible solution very quickly even if the lower bounds are poor under the same circumstance.

Figure 9 gives the privacy losses of each algorithm under different tightness, where PT-ISABB-B (i.e., the local-elimination version of PT-ISABB) is considered in this experiment to compare the performance of the non local-elimination version and local-elimination version of PT-ISABB on the constraint privacy protecting. Different from the other competitors, the privacy loss in PT-ISABB-D comes from both the inference phase and the search phase. As described in Sect. 4.4, the inference phase would cause a half privacy loss on each tree edge in the worst case as the variable elimination is actually performed by parent agents. In spite of this, it is still a better choice than directly employing DPOP to solve the problems, which will leak at least half privacy. In the search phase, the proposed estimation reporting mechanism where parent and pseudo parents only need to forward their residual constraint costs to their (pseudo) children can reduce the privacy loss to some degree, but agents can still infer the other-side actual constraint costs. Fortunately, the loss could be much reduced by efficient pruning. So it is no surprise to see that our proposed algorithm leaks more privacy than the other competitors when solving



(a) Normalized costs of the sparse problems **(b)** Normalized costs of the dense problems

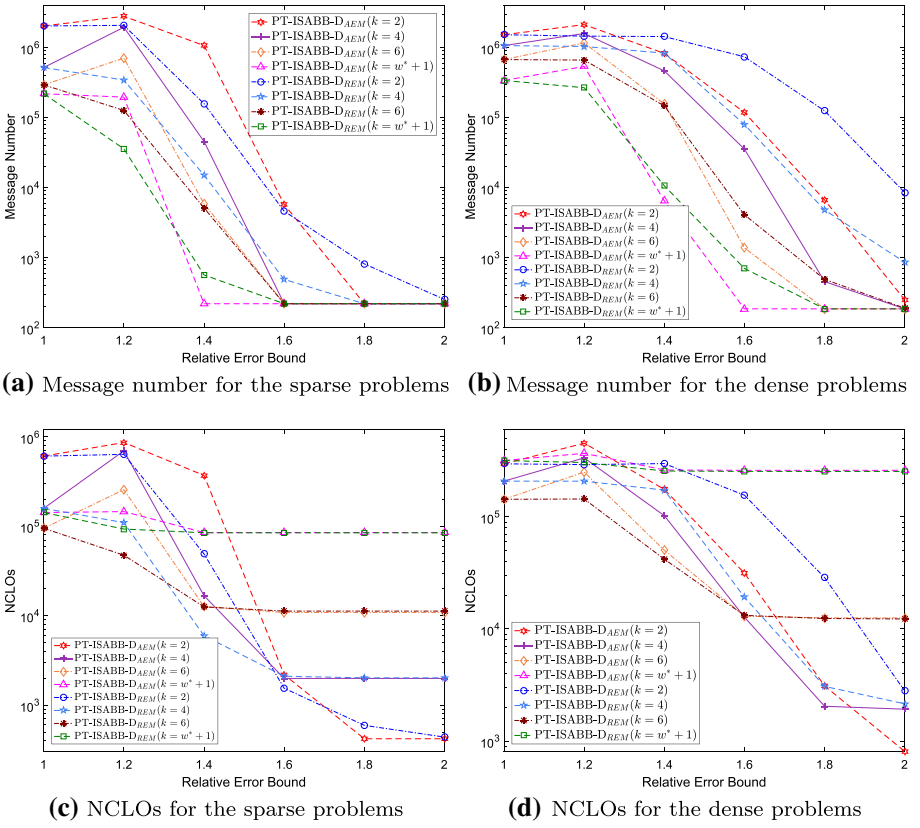
Fig. 10 Normalized costs of the sparse problems ($|A| = 24$, $|D_i| = 3$ and $p = 0.15$) and dense problems ($|A| = 14, |D_i| = 3$ and $p = 0.6$) under different relative error bound

the problems with low tightness. That is due to the fact that such problems usually have feasible solutions, and thus most of the entries in a cost table from a child are zero. However, the advantage of PT-ISABB-D is gradually emerged as the tightness grows. It can be seen that PT-ISABB-D with $k \geq 4$ leaks less privacy than the other competitors when the tightness is over 0.5. The reason is twofold: one is that with the increase of the tightness the feasible assignment pairs parents can infer are on the decrease, and the other one is that the tight lower bounds produced in the inference phase lead to the efficient pruning during the search phase. Besides, it is worth mentioning that PT-ISABB-B performs better in terms of privacy protecting when solving the problems with the tightness under 0.5. That is because variables are already eliminated before sending cost tables to their parents. Therefore, parents only know the best cost their children can achieve, but cannot figure out the corresponding assignments of their children.

7.5 Performance comparisons: the suboptimal variants of PT-ISABB-D

We compare the two suboptimal variants of PT-ISABB-D to each other on the sparse and dense random ADCOPs. Specifically, we consider the random ADCOPs with 24 agents, the density of 0.15 and the domain size of 3 as the sparse problems, and the ones with 14 agents, the density of 0.6 and the domain size of 3 as the dense problems. Besides, we vary the relative error bound from 1 to 2 to assess the performance of the two suboptimal variants. Here, the relative error bound is ρ for PT-ISABB-D$_{REM}$ or b divided by the optimal solution cost for PT-ISABB-D$_{AEM}$. For PT-ISABB-D$_{AEM}$, we select $v_i = util^{i+}_{P(a_i)}(Cpa_i) - util^{i-}_{P(a_i)}(Cpa_i)$ since three settings in Eq. (28) lead to the similar trend in the experiment, and pre-calculate the average optimal solution cost to set the value of b. In our experiments, the average optimal solution cost is 2741 for spare problems and 4234 for dense problems, respectively.

Figure 10 shows the normalized cost of each algorithm for the sparse and dense problems under different relative error bounds. Here, the average induced width is 6 for the sparse problems and 8 for the dense problems. It can be seen that the normalized solution costs of two suboptimal variants increase as the relative error bound waxes. However, the solution costs are still much smaller than the error bound. Furthermore, the normalized



(a) Message number for the sparse problems **(b)** Message number for the dense problems

(c) NCLOs for the sparse problems **(d)** NCLOs for the dense problems

Fig. 11 Performance comparison of the sparse problems ($|A| = 24, |D_i| = 3$ and $p = 0.15$) and dense problems ($|A| = 14, |D_i| = 3$ and $p = 0.6$) under different relative error bound

solution costs of PT-ISABB-D$_{AEM}$ are always larger than the ones of PT-ISABB-D$_{REM}$ with the same relative error bound. This is due to the termination condition for two suboptimal variants that the root agent a_i holds $LB_i \leq cost_i \leq LB_i + gap_i$ where $cost_i$ is the solution cost, $gap_i = (\rho - 1) * optCost_i$ in PT-ISABB-D$_{AEM}$ and $gap_i = (\rho - 1) * LB_i$ in PT-ISABB-D$_{REM}$, and $optCost_i$ is the optimal solution cost. Since $optCost_i \geq LB_i$, the absolute error bound (gap_i) of PT-ISABB-D$_{AEM}$ is no smaller than gap_i of PT-ISABB-D$_{REM}$. Besides, it can be noticed that PT-ISABB-D$_{AEM}$ with different k have little difference in the solution cost when the relative error bound is small, but the advantages of large k appear as the relative error bound waxes. The reason for this is that the gaps between lower bounds and upper bounds constructed by the inference phase with large k are close to the user-specified error bound when the specified error bound is very large. This also happens when using PT-ISABB-D$_{REM}$ with different k to solve the sparse problems, but there are some differences on the dense problems. This is because the heuristic bounds in PT-ISABB-D$_{REM}$ with $k \leq 6$ do less for the termination of PT-ISABB-D$_{REM}$ since the average induced width of the dense problems is greater than the one of the sparse problems. In addition, PT-ISABB-D$_{REM}$ with large k performs unsatisfactory on the dense problems since the inference phase with large k can produce tight lower bounds to terminate PT-ISABB-D$_{REM}$ quickly.

Figure 11 presents the performance comparison of two suboptimal variants for solving the sparse and dense problems under different relative error bounds in terms of the message number and NCLOs. It can be noticed that PT-ISABB-D$_{AEM}$ does not always decrease with the increase of the relative error bound and PT-ISABB-D$_{AEM}$ with the relative error bound to 1.2 produces the maximal number of messages and NCLOs. Also, PT-ISABB-D$_{REM}$ with $\rho = 1$ and PT-ISABB-D$_{REM}$ with $\rho = 1.2$ require the similar number of messages and NCLOs. That is due to inefficient pruning during the search phase when the relative error bound equals 1.2. More precisely, two suboptimal variants can not get low upper bounds promptly with a large relative error bound, but they still need to find a solution with the cost that is no greater than 1.2 times the optimal solution cost. Moreover, PT-ISABB-D$_{AEM}$ produces much more messages and NCLOs than PT-ISABB-D$_{REM}$ with the same k when the relative error bound is relatively small but this phenomenon changes as the relative error bound waxes. That is because the absolute error bound allocation mechanism in PT-ISABB-D$_{AEM}$ [i.e., Eq. (28)] does not allocate the absolute error bounds evenly when the relative error bound is relatively small, but the allocation gradually becomes even as the relative error bound waxes. Besides, it can be concluded from Figs. 11a and 10a that the suboptimal variants with $k \geq 4$ can quickly find a solution within the specified error bound for the sparse problems when the relative error bound is large enough.

8 Conclusion

It is known that DPOP/ADPOP for DCOPs cannot be directly applied to ADCOPs due to a privacy concern. In this paper, we take ADPOP into solving ADCOPs for the first time by combining with a tree-based variation of SyncABB-1ph, and present a two-phase complete algorithm called PT-ISABB. In the inference phase, a non-local elimination version of ADPOP is performed to solve a subset of constraints and build look-up tables for tight lower bounds and upper bounds. In the search phase, a tree-based variation of SyncABB-1ph is implemented to exhaust the search space and an estimation reporting mechanism is introduced to avoid directly disclosing the private constraint costs and compute complete upper bounds. Furthermore, the two suboptimal variants of PT-ISABB, named PT-ISABB$_{AEM}$ and PT-ISABB$_{REM}$, are proposed to allow a desired trade-off between solution quality and coordination overheads, where PT-ISABB$_{AEM}$ and PT-ISABB$_{REM}$ guarantee to find a solution within the user-specified absolute error bound and relative error bound, respectively. To implement PT-ISABB$_{AEM}$, we introduce an absolute error bound allocation mechanism to allocate the user-specified absolute error bound for all the non-root agents so as to relax their backtracking condition. The experimental results show that PT-ISABB is markedly superior to state-of-the-art search-based algorithms as well as its local elimination version and leaks less privacy when solving complex problems. It can also be seen from the empirical evaluation that the suboptimal variants of PT-ISABB can find a solution within the user-specified bounded-error with less coordination overheads when the relative error bound is greater than 1.2.

In the future, we plan to improve PT-ISABB in the following aspects: firstly, we will work for further enhancing the pruning efficiency by computing lower upper bounds in the search phase; secondly, we will devote to improving the parallelism by combining the algorithm with multi-search process [28]; finally, we will probe into rearranging the search space with the initial lower bounds and upper bounds so as to find a better global upper bound to speed up the systematic search.

Acknowledgements This research is funded by Chongqing Research Program of Basic Research and Frontier Technology (No. cstc2017jcyjAX0030), Fundamental Research Funds for the Central Universities (No. 2018CDXYJSJ0026) and Graduate Research and Innovation Foundation of Chongqing (No. CYS17023). We are also grateful to the reviewers of this article for their kind suggestions.

References

1. Barabási, A. L., & Albert, R. (1999). Emergence of scaling in random networks. *Science*, *286*(5439), 509–512.
2. Brito, I., Meisels, A., Meseguer, P., & Zivan, R. (2009). Distributed constraint satisfaction with partially known constraints. *Constraints*, *14*(2), 199–234.
3. Burke, D. A., Brown, K. N., Dogru, M., & Lowe, B. (2007). Supply chain coordination through distributed constraint optimization. In *Proceedings AAMAS workshop on distributed constraint reasoning (DCR-07)*.
4. Chechetka, A., & Sycara, K. (2006) No-commitment branch and bound search for distributed constraint optimization. In *Proceedings of the 5th international joint conference on autonomous agents and multiagent systems* (pp. 1427–1429).
5. Chen, Z., Deng, Y., Wu, T., & He, Z. (2018). A class of iterative refined Max-sum algorithms via non-consecutive value propagation strategies. *Autonomous Agents and Multi-Agent Systems*, *32*(6), 822–860.
6. Chen, Z., He, C., He, Z., & Chen, M. (2018). BD-ADOPT: A hybrid DCOP algorithm with best-first and depth-first search strategies. *Artificial Intelligence Review*, *50*(2), 161–199.
7. Chen, Z., Zhang, W., Deng, Y., Chen, D., & Li, Q. (2020). RMB-DPOP: Refining MB-DPOP by reducing redundant inference. In *Proceedings of the 19th international conference on autonomous agents and multiagent systems* (pp. 249–257).
8. Dechter, R., & Rish, I. (2003). Mini-buckets: A general scheme for bounded inference. *Journal of the ACM (JACM)*, *50*(2), 107–153.
9. Farinelli, A., Rogers, A., Petcu, A., & Jennings, N. R. (2008). Decentralised coordination of low-power embedded devices using the Max-sum algorithm. In *Proceedings of the 7th international joint conference on autonomous agents and multiagent systems* (pp. 639–646).
10. Fioretto, F., Pontelli, E., & Yeoh, W. (2018). Distributed constraint optimization problems and applications: A survey. *Journal of Artificial Intelligence Research*, *61*, 623–698.
11. Fioretto, F., Yeoh, W., & Pontelli, E. (2017) A multiagent system approach to scheduling devices in smart homes. In *Proceedings of the 16th international conference on autonomous agents and multiagent systems* (pp. 981–989).
12. Freuder, E. C., & Quinn, M. J. (1985). Taking advantage of stable sets of variables in constraint satisfaction problems. In *Proceedings of the 9th international joint conference on artificial intelligence* (pp. 1076–1078).
13. Gershman, A., Meisels, A., & Zivan, R. (2009). Asynchronous forward bounding for distributed COPs. *Journal of Artificial Intelligence Research*, *34*, 61–88.
14. Gershman, A., Zivan, R., Grinshpoun, T., Grubshtein, A., & Meisels, A. (2008) Measuring distributed constraint optimization algorithms. In *Proceedings AAMAS workshop on distributed constraint reasoning (DCR-08)*
15. Grinshpoun, T., Grubshtein, A., Zivan, R., Netzer, A., & Meisels, A. (2013). Asymmetric distributed constraint optimization problems. *Journal of Artificial Intelligence Research*, *47*, 613–647.
16. Grinshpoun, T., & Tassa, T. (2016). P-SyncBB: A privacy preserving branch and bound DCOP algorithm. *Journal of Artificial Intelligence Research*, *57*, 621–660.
17. Grinshpoun, T., Tassa, T., Levit, V., & Zivan, R. (2019). Privacy preserving region optimal algorithms for symmetric and asymmetric dcops. *Artificial Intelligence*, *266*, 27–50.
18. Hirayama, K., Miyake, K., Shiota, T., & Okimoto, T. (2019). DSSA+: Distributed collision avoidance algorithm in an environment where both course and speed changes are allowed. TransNav. *International Journal on Marine Navigation and Safety of Sea Transportation, 13*(1), 117–123.
19. Hirayama, K., & Yokoo, M. (1997). Distributed partial constraint satisfaction problem. In *International conference on principles and practice of constraint programming* (pp. 222–236).
20. Hirayama, K., & Yokoo, M. (2005). The distributed breakout algorithms. *Artificial Intelligence*, *161*(1–2), 89–115.
21. Kschischang, F. R., Frey, B. J., & Loeliger, H. A. (2001). Factor graphs and the Sum-product algorithm. *IEEE Transactions on Information Theory*, *47*(2), 498–519.

22. Léauté, T., & Faltings, B. (2013). Protecting privacy through distributed computation in multi-agent decision making. *Journal of Artificial Intelligence Research*, *47*, 649–695.
23. Litov, O., & Meisels, A. (2017). Forward bounding on pseudo-trees for DCOPs and ADCOPs. *Artificial Intelligence*, *252*, 83–99.
24. Maheswaran, R. T., Pearce, J. P., & Tambe, M. (2004). Distributed algorithms for DCOP: A graphical-game-based approach. In *Proceedings of ISCA PDCS'04* (pp. 432–439).
25. Maheswaran, R. T., Tambe, M., Bowring, E., Pearce, J. P., & Varakantham, P. (2004) Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *Proceedings of the 3rd international joint conference on autonomous agents and multiagent systems* (pp. 310–317).
26. Modi, P. J., Shen, W. M., Tambe, M., & Yokoo, M. (2005). ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, *161*(1–2), 149–180.
27. Monteiro, T. L., Pujolle, G., Pellenz, M. E., Penna, M. C., & Souza, R. D. (2012) A multi-agent approach to optimal channel assignment in WLANS. In *Wireless communications and networking conference (WCNC)* (pp. 2637–2642).
28. Netzer, A., Grubshtein, A., & Meisels, A. (2012). Concurrent forward bounding for distributed constraint optimization problems. *Artificial Intelligence*, *193*, 186–216.
29. Nguyen, D. T., Yeoh, W., Lau, H. C., & Zivan, R. (2019). Distributed Gibbs: A linear-space sampling-based DCOP algorithm. *Journal of Artificial Intelligence Research*, *64*, 705–748.
30. Okamoto, S., Zivan, R., & Nahon, A. (2016) Distributed breakout: Beyond satisfaction. In *Proceedings of the 25th international joint conference on artificial intelligence* (pp. 447–453).
31. Ottens, B., Dimitrakakis, C., & Faltings, B. (2017). DUCT: An upper confidence bound approach to distributed constraint optimization problems. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *8*(5), 69.
32. Petcu, A., & Faltings, B. (2005). Approximations in distributed optimization. In *International conference on principles and practice of constraint programming* (pp. 802–806). Berlin: Springer.
33. Petcu, A., & Faltings, B. (2005). A scalable method for multiagent constraint optimization. In *Proceedings of the 19th international joint conference on artificial intelligence* (pp. 266–271).
34. Petcu, A., & Faltings, B. (2006) ODPOP: An algorithm for open/distributed constraint optimization. In *Proceedings of the 21st AAAI conference on artificial intelligence* (pp. 703–708).
35. Petcu, A., & Faltings, B. (2007). MB-DPOP: A new memory-bounded algorithm for distributed optimization. In *Proceedings of the 20th international joint conference on artificial intelligence* (pp. 1452–1457).
36. Ramchurn, S. D., Vytelingum, P., Rogers, A., & Jennings, N. (2011). Agent-based control for decentralised demand side management in the smart grid. In *Proceedings of the 10th international conference on autonomous agents and multiagent systems* (pp. 5–12).
37. Rogers, A., Farinelli, A., Stranders, R., & Jennings, N. R. (2011). Bounded approximate decentralised coordination via the Max-sum algorithm. *Artificial Intelligence*, *175*(2), 730–759.
38. Sultanik, E. A., Modi, P. J., & Regli, W. C. (2007). On modeling multiagent task scheduling as a distributed constraint optimization problem. In *Proceedings of the 20th international joint conference on artificial intelligence* (pp. 1531–1536).
39. Vinyals, M., Rodriguez-Aguilar, J. A., & Cerquides, J. (2011). Constructing a unifying theory of dynamic programming DCOP algorithms via the generalized distributive law. *Autonomous Agents and Multi-Agent Systems*, *22*(3), 439–464.
40. Yeoh, W., Felner, A., & Koenig, S. (2010). BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. *Journal of Artificial Intelligence Research*, *38*, 85–133.
41. Yeoh, W., & Yokoo, M. (2012). Distributed problem solving. *AI Magazine*, *33*(3), 53.
42. Zhang, W., Wang, G., Xing, Z., & Wittenburg, L. (2005). Distributed stochastic search and distributed breakout: Properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, *161*(1–2), 55–87.
43. Zivan, R., & Meisels, A. (2006). Message delay and DisCSP search algorithms. *Annals of Mathematics and Artificial Intelligence*, *46*(4), 415–439.
44. Zivan, R., Parash, T., Cohen, L., Peled, H., & Okamoto, S. (2017). Balancing exploration and exploitation in incomplete Min/Max-sum inference for distributed constraint optimization. *Autonomous Agents and Multi-Agent Systems*, *31*(5), 1165–1207.
45. Zivan, R., Parash, T., Cohen-Lavi, L., & Naveh, Y. (2020). Applying Max-sum to asymmetric distributed constraint optimization problems. *Autonomous Agents and Multi-Agent Systems*, *34*(1), 1–29.