Noname manuscript No.
(will be inserted by the editor)

Inference-based Complete Algorithms for Asymmetric Distributed Constraint Optimization Problems

Dingding Chen · Ziyu Chen · Yanchen Deng · Zhongshi He · Lulu Wang

Received: date / Accepted: date

Abstract Asymmetric Distributed Constraint Optimization Problems (ADCOPs) are an important framework for multiagent coordination and optimization, where each agent has its personal preferences. However, the existing inference-based complete algorithms that use local eliminations cannot be applied to ADCOPs, as the (pseudo) parents are required to transfer their private functions to their (pseudo) children to perform the local eliminations optimally. Rather than disclosing private functions explicitly to facilitate local eliminations, we solve the problem by enforcing delayed eliminations and propose the first inference-based complete algorithm for ADCOPs, named AsymDPOP. To solve the severe scalability problems incurred by delayed eliminations, we propose to reduce the memory consumption by propagating a set of smaller utility tables instead of a joint utility table, and the computation efforts by sequential eliminations instead of joint eliminations. To ensure the proposed algorithms can scale up to large-scale problems under the limited memory, we combine them with the memory-bounded inference by iteratively propagating the memory-bounded utility tables with the instantiation of cycle-cut (CC) nodes, where we propose to reduce the redundancy in bounded utility iterative propagation by enumerating CC nodes in different branches independently and propagating the utility tables within the memory limit only once. The empirical evaluation indicates that the proposed methods significantly outperform the state-of-the-art as well as the vanilla DPOP with PEAV formulation.

This paper is an extension of our IJCAI paper (Deng et al., 2019). Besides an extended description, examples, complete proofs, it includes two adaptations of the proposed algorithms to the memory-bounded inference, which were not included in the IJCAI version.

✉ Ziyu Chen
E-mail: chenziyu@cqu.edu.cn
College of Computer Science, Chongqing University, Chongqing 400044, China

Keywords DCOP · ADCOP · Inference-based Complete ADCOP Algorithm · Memory-bounded Inference

1 Introduction

Distributed Constraint Optimization Problems (DCOPs) (Modi et al., 2005; Yeoh and Yokoo, 2012; Fioretto et al., 2018) are a fundamental framework for multiagent system where multiple agents coordinate their decisions to optimize a global objective. DCOPs have been successfully applied to model a variety of real-world problems where information and controls are inherently distributed among agents, such as frequency allocation (Monteiro et al., 2012), distributed scheduling (Sultanik et al., 2007; Hirayama et al., 2019), smart grid (Fioretto et al., 2017) and many others.

Algorithms for DCOP can be classified into two categories, i.e., incomplete algorithms and complete algorithms. Incomplete algorithms for DCOPs focus on finding near-optimal solutions at small coordination overheads, and generally follow three main strategies, i.e., local search (Maheswaran et al., 2004a; Hirayama and Yokoo, 2005; Zhang et al., 2005; Okamoto et al., 2016; Hoang et al., 2018; Leite and Enembreck, 2019), inference (Farinelli et al., 2008; Rogers et al., 2011; Zivan et al., 2017; Chen et al., 2018; Cohen et al., 2020; Zivan et al., 2020a) and sampling (Ottens et al., 2017; Nguyen et al., 2019). In contrast, complete algorithms aim to find the optimal solution and are roughly divided into search-based algorithms and inference-based algorithms. Search-based complete algorithms (Hirayama and Yokoo, 1997; Modi et al., 2005; Gershman et al., 2009; Yeoh et al., 2010; Netzer et al., 2012; Litov and Meisels, 2017; Chen et al., 2020b) systematically explore the entire solution space by distributed backtrack search. Instead, inference-based complete algorithms (Petcu and Faltings, 2005b; Vinyals et al., 2011) employ a dynamic programming paradigm to solve DCOPs.

Unfortunately, DCOPs fail to capture the asymmetric structure which is ubiquitous in real-world problems (Burke et al., 2007; Maheswaran et al., 2004b; Ramchurn et al., 2011) since each constrained agent shares the same payoffs. PEAV (Private Events As Variables) (Maheswaran et al., 2004b) attempts to capture asymmetric constraint costs by introducing mirror variables and the consistency enforced by hard constraints. However, PEAV suffers from scalability problems since the number of variables significantly increases. Moreover, many classical DCOP algorithms perform poorly when applied to the formulation due to the presence of hard constraints (Grinshpoun et al., 2013). On the other side, ADCOPs (Grinshpoun et al., 2013) are another framework that captures asymmetry by explicitly defining the exact payoff for each participant of a constraint without introducing any variables, which has been intensively investigated in recent years.

Solving ADCOPs involves evaluating and aggregating the payoff for each constrained agent, which is more challenging in asymmetric settings due to a privacy concern. Complete algorithms for ADCOP (Grinshpoun et al., 2013;

Litov and Meisels, 2017) are nearly the variants of search-based complete DCOP solvers with consideration of the aggregation for two-side constraint costs. Namely, they rely on an exhaustive search to guarantee the optimality, which incurs exponential messages. On the other hand, although inference-based complete algorithms (e.g., DPOP (Petcu and Faltings, 2005b)) only require a linear number of messages of exponential size with respect to the induced width to solve DCOPs, they cannot be directly applied to solve ADCOPs without PEAV due to their requirement for complete knowledge of each constraint to facilitate variable eliminations. Accordingly, (pseudo) parents have to transfer their private constraint functions to their (pseudo) children, which leaks at least half of privacy.

To address the above issues, in this paper, we accommodate DPOP to ADCOPs for the first time by deferring the eliminations of variables. Specifically, we contribute to the state-of-the-art in the following aspects.

- We propose the first inference-based complete algorithm to solve ADCOPs, named AsymDPOP, via generalizing non-local elimination (Chen et al., 2020a). That is, for eliminating the variables optimally, the eliminations are postponed until their highest (pseudo) parent to void (pseudo) parents' private functions being exposed directly, so as to protect privacy through the solving process. Besides, we theoretically show that the space complexity of an agent in AsymDPOP.
- We scale up AsymDPOP by introducing a table-set propagation scheme (TSPS) to reduce memory consumption by discarding the unnecessary joint operations in the utility table calculation, and a mini-batch elimination scheme (MBES) to reduce the number of operations by distributing the elimination operators over each eliminated variable set. Moreover, we theoretically show that the memory consumption of AsymDPOP with TSPS is no greater than the one of AsymDPOP.
- To improve the scalability of AsymDPOP within the limited memory, we present RMB-AsymDPOP by combining the memory-bounded inference with AsymDPOP, where we propose to enumerate cycle-cut (CC) nodes in different branches independently instead of jointly to reduce the redundancy in the enumeration of CC nodes. Furthermore, we enhance RMB-AsymDPOP by combining it with TSPS, and propose to propagate the utility tables within the memory limit only once rather than iteratively to reduce the redundancy in the utility set propagation.
- We experimentally evaluate the proposed algorithms on various benchmarks. The experimental results show that the proposed algorithms are scalable and significantly outperform the state-of-the-art as well as the vanilla DPOP with PEAV formulation.

To sum up, we propose four inference-based complete algorithms (i.e, AsymDPOP and its variants) for ADCOPs. The relationship between them, as well as their relationship to existing inference-based complete DCOP algorithms including DPOP and its memory-bounded version (i.e., RMB-DPOP), can be found in Fig. **??**. The rest of this paper is organized as follows. We briefly

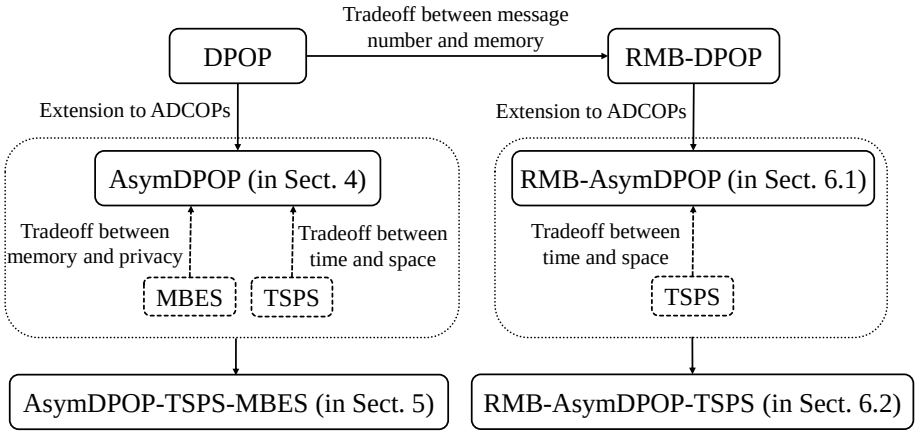


Fig. 1: The relation diagram of the proposed methods

review related work in Sect. 2. The preliminaries including DCOPs, ADCOPs, pseudo tree, DPOP, non-local elimination, MB-DPOP and RMB-DPOP are presented in Sect. 3. In Sect. 4, we describe the proposed inference-based complete algorithm for ADCOPs, named AsymDPOP. Then, we present two tradeoffs of AsymDPOP in Sect. 5 and its memory-bounded versions in Sect. 6. Next, a discussion about the privacy of the proposed methods is provided in Sect. 7. Finally, we present the empirical evaluation of our algorithms in Sect. 8 and the conclusion in Sect. 9.

2 Related work

Incomplete algorithms for DCOPs are broadly categorized into local search, inference-based and sampling-based algorithms. Local search algorithms including DBA (Hirayama and Yokoo, 2005), DSA (Zhang et al., 2005), MGM (Maheswaran et al., 2004a) and GDBA (Okamoto et al., 2016) are typical incomplete DCOP algorithms, where each agent keeps exchanging itself state with its neighbors and optimizes individual benefit in terms of the latest states of its neighbors. Inference-based incomplete algorithms like Max-sum (Farinelli et al., 2008) and its variants (Rogers et al., 2011; Zivan et al., 2017; Chen et al., 2018; Cohen et al., 2020; Zivan et al., 2020a) gather the global information through employing belief propagation on a factor-graph (Kschischang et al., 2001). Sampling-based algorithms including DUCT (Ottens et al., 2017) and D-Gibbs (Nguyen et al., 2019) sample the search space on a pseudo tree through the confidence bounds or the statistical inference.

Complete algorithms for DCOPs are classified as search-based or inference-based algorithms. SyncBB (Hirayama and Yokoo, 1997) is an early search-based algorithm built on a chain-based structure, but it incurs an unnecessary waste of computing resources due to the synchronous execution. To achieve

better concurrency, AFB (Gershman et al., 2009) and ConcFB (Netzer et al., 2012) were proposed. Since the algorithms with the chain-based structure could force unconstrained agents to communicate with each other and disallow parallel exploration of the solution space, a pseudo tree (Freuder and Quinn, 1985) is used to create communication links among constrained agents and parallelize the computation in different independent branches. ADOPT (Modi et al., 2005) is a representative asynchronous search-based algorithm operating on a pseudo tree and using a best-first search strategy. Afterward, many search algorithms based on ADOPT (Silaghi and Yokoo, 2006, 2009; Gutierrez and Meseguer, 2010) thrive. To improve the search efficiency of ADOPT, BnB-ADOPT (Yeoh et al., 2010) and its variants (Gutierrez and Meseguer, 2010; Gutierrez et al., 2013) employ depth-first search and branch-and-bound strategies instead. Different from ADOPT and BnB-ADOPT, PT-FB (Litov and Meisels, 2017) is a synchronous search-based algorithm that constructs lower bounds to speed up the search by performing forward bounding on a pseudo tree. Subsequently, HS-CAI (Chen et al., 2020b) was proposed to build tight lower bounds through executing context-based inference iteratively.

DPOP (Petcu and Faltings, 2005b) is a famous inference-based algorithm for DCOPs, which performs a dynamic programming strategy. In DPOP, the assignment combination utilities are forwarded bottom-up, and then the optimal decisions are propagated along the pseudo tree reversely. However, its memory consumption is exponential in the induced width of the pseudo tree. Therefore, ODPOP (Petcu and Faltings, 2006), MB-DPOP (Petcu and Faltings, 2007) and RMB-DPOP (Chen et al., 2020d) were proposed to trade the number of messages for smaller memory consumption. In addition, Action_GDL (Vinyals et al., 2011) was proposed to enhance the efficiency by performing dynamic programming on a distributed junction tree.

Existing algorithms for solving ADCOPs are almost the adaption of the DCOP algorithms by handling asymmetric constraint costs. Local search algorithms for ADCOPs including ACLS, MCS-MGM and GCA-MGM (Grinshpoun et al., 2013) are the asymmetric extensions of DSA and MGM, where each agent exchanges itself state with its neighbors and each constraint cost on its side with the neighbors that participate in the same constraint. Besides, Zivan et al. (Zivan et al., 2020b) proposed to adjust Max-sum and its variants to solve ADCOPs via formulating asymmetric factor-graphs.

Search-based complete algorithms for ADCOPs employ either a two-phase strategy or a one-phase strategy to aggregate double-side constraint costs. Specifically, the algorithms with a two-phase strategy first consider one-side constraint costs in the first phase and then gather the other side constraint costs in the second phase when reaching a complete assignment. While the algorithms with a one-phase strategy systematically check each side of the constraint costs before reaching a full assignment. SyncABB-2ph (Grinshpoun et al., 2013) is an asymmetric version of SyncBB with a two-phase strategy. Instead, SyncABB-1ph and ATWB (Grinshpoun et al., 2013) are asymmetric adaptations of SyncBB (Hirayama and Yokoo, 1997) and AFB (Gershman et al., 2009), using a one-phase strategy. To implement the 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 (Litov and Meisels, 2017) is the first tree-based algorithm for ADCOPs, which uses forward bounding to compute lower bounds and back bounding to achieve a one-phase check. Recently, PT-ISABB (Chen et al., 2020a) was proposed to improve a tree-based search process by implementing a non-local elimination version of ADPOP (Petcu and Faltings, 2005a) to provide much tighter lower bounds.

3 Background

In this section, we introduce the preliminaries including DCOPs, ADCOPs, pseudo tree, DPOP, non-local elimination, MB-DPOP and RMB-DPOP.

3.1 Distributed constraint optimization problems

A distributed constraint optimization problem (DCOP) (Modi et al., 2005) 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 a single 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 $x_i \in X$. Besides, we denote the maximal domain size as $d = \max_{x_i \in X} |D_i|$.
- $F = \{f_1, \ldots, f_m\}$ is a set of constraint functions. Each function $f_i : D_{i_1} \times \cdots \times D_{i_k} \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ specifies a non-negative cost to each value combination of the involved variables $x_{i_1}, \ldots, x_{i_k}$. Here, the constraints of an infinite cost are called hard constraints which represent the combinations of assignment that are strictly forbidden, and the constraints of a finite cost are called soft constraints.

For the sake of simplicity, we assume that each agent only controls a single variable (i.e., $q = n$) and all constraints are binary (i.e., $f_{ij} : D_i \times D_j \to \mathbb{R}_{\geq 0} \cup \{\infty\}$). Thus, the term *agent* and *variable* can be used interchangeably. These assumptions are commonly used in the DCOP literature (Litov and Meisels, 2017; Modi et al., 2005; Petcu and Faltings, 2005b; Yeoh et al., 2010). The objective of a DCOP is to find a joint assignment to all variables such that the total cost is minimized. That is,

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

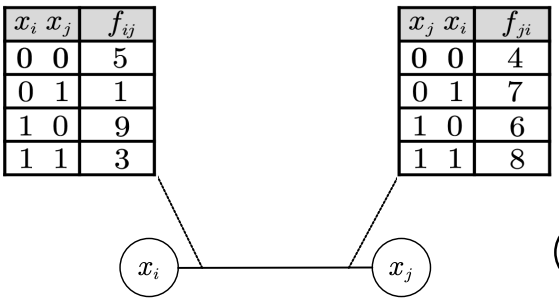


Fig. 2: An ADCOP with two variables and a constraint

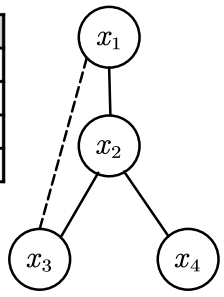


Fig. 3: Pseudo tree

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) (Grinshpoun et al., 2013) 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 assumptions in a DCOP, we also assume that each agent only controls a single variable and all the constraints are binary in an ADCOP. Therefore, an ADCOP can also be visualized by a constraint graph, where the vertexes denote variables and the edges denote constraints. Figure 2 presents an ADCOP with two variables x_i and x_j and a binary constraint between them, where the private constraint functions for x_i and x_j are denoted as f_{ij} and f_{ji}, respectively. Note that in the asymmetric setting, f_{ij} does not necessarily equal f_{ji}. The objective of an ADCOP is to find a joint assignment to all variables such that the aggregated constraint costs of each side are minimized. That is,

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

3.3 Pseudo tree

A pseudo tree (Freuder and Quinn, 1985) is an ordered arrangement of a constraint graph with the property that adjacent nodes from the original graph fall in the same branch of the tree, which is widely used to establish a communication structure and organize the solution space. A pseudo tree can be generated by a depth-first traversal of a constraint graph, categorizing constraints into tree edges and back edges (i.e., non-tree edges). Given a pseudo tree, the neighbors of a variable x_i (i.e., $N(x_i)$), the variables who share constraints with

x_i) can be categorized into its parent $P(x_i)$, pseudo parents $PP(x_i)$, children $C(x_i)$ and pseudo children $PC(x_i)$ according to their positions in the pseudo tree and the type of edges they connect through. Concretely, these notations can be formally defined by:

- $P(x_i)$ is an ancestor connecting with x_i through a tree edge.
- $PP(x_i)$ is the set of ancestors connecting with x_i through back edges.
- $C(x_i)$ is the set of descendants that connect with x_i through tree edges.
- $PC(x_i)$ is the set of descendants that connect with x_i through back edges.

We also adopt the following notations for succinctness.

- $AP(x_i)$ is the set of x_i's ancestors connecting with x_i, i.e., $AP(x_i) = PP(x_i) \cup \{P(x_i)\}$.
- $AC(x_i)$ is the set of x_i's descendants connecting with x_i, i.e., $AC(x_i) = PC(x_i) \cup C(x_i)$.
- $Desc(x_i)$ is the set of descendants of x_i.
- $Anc(x_i)$ is the set of ancestors of x_i.
- B_i^c is the subset of $AC(x_i)$ that belongs to $Desc(x_c) \cup \{x_c\}$, i.e., $B_i^c = AC(x_i) \cap (Desc(x_c) \cup \{x_c\})$.
- $Sep(x_i)$ (Petcu and Faltings, 2006) is the separator set of x_i, comprising the ancestors that are constrained with the variables in $\{x_i\} \cup Desc(x_i)$, i.e., $Sep(x_i) = \{x_j \in Anc(x_i) | \exists x_k \in \{x_i\} \cup Desc(x_i), s.t., x_j \in AP(x_k)\}$. It can also be defined recursively as $Sep(x_i) = AP(x_i) \cup (\cup_{x_c \in C(x_i)} Sep(x_c)) \backslash \{x_i\}$. Besides, we denote the induced width of x_i as $|Sep(x_i)|$ and the induced width of the pseudo tree as $w^* = \max_{x_i \in X} |Sep(x_i)|$.

Figure 3 presents a pseudo tree where the solid edges and the dashed edges are tree edges and back edges, respectively. In the pseudo tree, x_2's neighbors (i.e., $N(x_2) = \{x_1, x_3, x_4\}$) are classified into $P(x_2) = x_1, PP(x_2) = \emptyset, C(x_2) = \{x_3, x_4\}$, $PC(x_2) = \emptyset$. Accordingly, $AP(x_2) = \{x_1\}$ and $AC(x_2) = \{x_3, x_4\}$. Besides, we have $Desc(x_2) = \{x_3, x_4\}$ and $Anc(x_2) = \{x_1\}$. Since $Desc(x_3) = Desc(x_4) = \emptyset$, we have $B_2^3 = \{x_3\}$ and $B_2^4 = \{x_4\}$. Particularly, since x_1 is constrained with both x_2 and x_3, we have $Sep(x_2) = \{x_1\}$. And we have $w^* = 2$ since $|Sep(x_1)| = 0$, $|Sep(x_2)| = |Sep(x_4)| = 1$ and $|Sep(x_3)| = 2$.

3.4 DPOP and non-local elimination

DPOP is an important inference-based complete algorithm for DCOPs, which implements the distributed bucket elimination scheme (Dechter et al., 2003) in a distributed manner. It performs two phases of propagation on a pseudo tree: a utility propagation phase to eliminate local variables and propagate utility tables from bottom to top, and a value propagation phase to select the optimal value for each variable along the pseudo tree reversely. Formally, the related operations of functions (e.g., constraint functions or utility tables) can be defined as follows.

Definition 1 (dims) Let f be a function, then $f.dims$ is its dimensions, i.e., the variables involved in f.

Definition 2 (Slice) Let P be a set of key-value pairs. $P_{[K]}$ is a slice of P over K such that:

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

Definition 3 (Join (Petcu and Faltings, 2005a)) Let f, f' be two functions and $D_f = \Pi_{x_i \in f.dims} D_i$,$D_{f'} = \Pi_{x_i \in f'.dims} D_i$ be their domain spaces. $f \otimes f'$ is the join of f and f' over $D_{f \otimes f'} = \Pi_{x_i \in f.dims \cup f'.dims} D_i$ such that:

$$(f \otimes f')(p) = f(p_{[f.dims]}) + f'(p_{[f'.dims]}), \forall p \in \{((f \otimes f').dims, V) | \forall V \in D_{f \otimes f'}\}$$

Definition 4 (Elimination (Petcu and Faltings, 2005a)) Let f be a function and S be a subset of the dimensions to f (i.e., $S \subset f.dims$). $f' = \min_S f$ is the minimal elimination of f along the S axis respectively, i.e.,

$$f'(p) = \min_{p_s \in \{(S,V) | \forall V \in D_S\}} f\left(p \cup p_s\right), \forall p \in \{(f'.dims, V) | \forall V \in D_{f'}\}$$

where $D_S = \Pi_{x_i \in S} D_i$. For succinctness, we denote $\min_{\{x_i\}} f$ as $\min_{x_i} f$.

Taking functions f_{ij} and f_{ji} in Fig. (2) as an example, we have the dimensions of f_{ij} is $f_{ij}.dims = \{x_i, x_j\}$, the join of f_{ij} and f_{ji} (i.e., $(f_{ij} \otimes f_{ji})$) and the elimination of f_{ij} along x_j (i.e., $\min_{x_j} f_{ij}$) can be computed by:
$(f_{ij} \otimes f_{ji})(x_i = d_i, x_j = d_j) = f_{ij}(x_i = d_i, x_j = d_j) + f_{ji}(x_j = d_j, x_i = d_i), \forall d_i, d_j \in \{0, 1\}$,
$(\min_{x_j} f_{ij})(x_i = d_i) = \min\{f_{ij}(x_i = d_i, x_j = 0), f_{ij}(x_i = d_i, x_j = 1)\}, \forall d_i \in \{0, 1\}$

Specifically, during the utility propagation phase, each variable x_i joins its local constraint functions related to $AP(x_i)$ with its received utility tables. Afterward, it eliminates its dimension from the joint utility table by computing the optimal value for each assignment combination of $Sep(x_i)$, and then propagates the eliminated result $u_{i \to p}$ to its parent. That is,

$$u_{i \to p} = \min_{x_i} \left(\left(\bigotimes_{x_j \in AP(x_i)} f_{ij} \right) \otimes \left(\bigotimes_{x_c \in C(x_i)} u_{c \to i} \right) \right)$$

where $u_{c \to i}$ is the utility table received from its child $x_c \in C(x_i)$.

In the value propagation phase, x_i determines its optimal assignment d_i^* based on the received utility tables $u_{c \to i}, \forall x_c \in C(x_i)$ and the optimal assignment PA^* received from its parent. That is,

$$d_i^* = \arg\min_{x_i} \left(\sum_{x_j \in AP(x_i)} f_{ij}(PA^*_{[x_j]}) + \sum_{x_c \in C(x_i)} u_{c \to i}(PA^*_{[u_{c \to i}.dims]}) \right)$$

Then, it extends PA^* with its optimal assignment and forwards the extended assignment to its children. It is noteworthy that the assignment sent to its child $x_c \in C(x_i)$ is $PA^*_{[Sep(x_c)]} \cup \{(x_i, d_i^*)\}$.

However, DPOP cannot be directly applied to asymmetric settings. That is because the total knowledge of each constraint function related to the variable should be aggregated for eliminating that variable optimally. Recently, PT-ISABB (Chen et al., 2020a) was proposed to apply an approximated version of

DPOP (i.e., ADPOP) to solve an ADCOP. Specifically, the algorithm performs variable eliminations only to a subset of constraints to build lookup tables for lower bounds, and then uses a tree-based search to find the optimal solution. It further proposed to reinforce the bounds by a non-local elimination scheme. That is, instead of performing variable eliminations locally, the elimination of a variable is postponed to its parent to include the private function enforced in its parent's side and promote the integrity of the utility table.

3.5 MB-DPOP and RMB-DPOP

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 problems with the large induced width. Consider the scenario where 8 agents with the domain size of 10 are fully-constrained. DPOP would require nearly 400MB of memory to perform variable elimination, which is impractical for typical DCOP applications (e.g., coordinating low-powered embedded devices). However, these problems can be solved by the search-based DCOP algorithms (e.g., PT-FB).

Subsequently, MB-DPOP (Petcu and Faltings, 2007), a memory-bounded algorithm (i.e., the algorithm with the maximal message below the user-specified memory budget k_{mb} in dimension size) was proposed to trade the number of messages for smaller memory consumption by using the cycle-cut idea (Dechter et al., 2003). To implement the cycle-cut idea, MB-DPOP introduces two additional phases, including a labeling phase and a bounded utility propagation phase. The labeling phase is used to group the variables with the induced width higher than k_{mb} into clusters, and determine cycle-cut (CC) nodes such that the width of each cluster is no greater than k_{mb} once CC nodes are removed. While the bounded utility propagation phase is implemented to iteratively propagate the utility tables with some dimensions fixed by the instantiation of CC nodes. Formally, a cluster and the CC nodes of that cluster can be defined by:

Definition 5 (Cluster node) Given a pseudo tree and a value of k_{mb}, a variable x_i in the pseudo tree is called a cluster node if $|Sep(x_i)| > k_{mb}$.

Definition 6 (Cluster root (CR) node) Given a pseudo tree and a value of k_{mb}, a variable x_i in the pseudo tree is called cluster root node if $\exists x_c \in C(x_i), s.t., |Sep(x_c)| > k_{mb}$ and $|Sep(x_i)| \leq k_{mb}$.

Definition 7 (Cluster of width greater than k_{mb}) Given a pseudo tree and a value of k_{mb}, a cluster C_r of width greater than k_{mb} is a set of nodes that are all labeled as cluster node or cluster root node, and there is a tree path between any two nodes in C_r that go only through cluster nodes in C_r.

Definition 8 (Cycle-cut (CC) nodes of a cluster) The cycle-cut (CC) nodes of a cluster C_r (i.e., $CClist_r$) is a subset of $\cup_{x_i \in C_r} Sep(x_i)$ such that for each variable $x_i \in C_r$ we have $|Sep(x_i) \setminus CClist_r| \leq k_{mb}$.

Specifically, for a cluster C_r, the CC nodes for C_r (i.e., $CClist_r$) are selected and aggregated in a bottom-up fashion during the labeling phase. In the bounded utility propagation phase, the CR node enumerates the instantiation of $CClist_r$ and propagates the instantiation iteratively to the nodes in C_r. Then, these nodes condition their utility tables on the received instantiation, and then forward these bounded utility tables to their parents. After exhausting all the instantiations of $CClist_r$, the CR node eliminates its dimension and also propagates its utility table to its parent.

For the nodes outside the clusters, they execute canonical utility propagation. After the bounded utility propagation and utility propagation phases end, a value propagation phase starts. Different from DPOP which only requires a round of utility propagation, the cluster nodes in MB-DPOP require an additional bounded utility propagation to re-drive the bounded utilities corresponding to the instantiation of CC nodes to obtain the optimal assignment. That is because each cluster node in the cluster only caches the utility table for the latest instantiation of CC nodes.

However, MB-DPOP suffers from a severe redundancy in memory-bounded inference, as it does not exploit the structure of a problem. Thus, three mechanisms including a distributed enumeration mechanism (DEM), an iterative selection mechanism (ISM) and a caching mechanism (CACHE) were proposed in RMB-DPOP (Chen et al., 2020d). Among them, DEM is adopted in each cluster to perform asynchronous memory-bounded inference by factorizing the instantiation. Specifically, for a cluster C_r that has CC nodes of $CClist_r$, instead of enumerating $CClist_r$ only at the CR node in MB-DPOP, the cluster nodes are eligible to enumerate $CClist_r$ in DEM. Namely, the CC nodes enumerated at $x_i \in C_r$ (i.e., CC_i) in DEM can be defined by:

$$CC_i = \begin{cases} CClist_r \cap (Sep(x_i) \cup \{x_i\}) & x_i \text{ is a CR node} \\ \{x_i\} \cap CClist_r & \text{otherwise} \end{cases} \tag{1}$$

ISM refines the CC node selection such that the number of CC nodes can be reduced. Concretely, for a cluster C_r, instead of determining $CClist_r$ in a bottom-up fashion by all the cluster nodes in MB-DPOP, each CC node in $CClist_r$ is determined iteratively only by the CR node in ISM such that it can cover a maximum number of the active nodes (i.e., the cluster nodes whose induced width is still greater than k_{mb} once the selected CC nodes are removed). And CACHE utilizes the historical inference results to further avoid unnecessary inferences.

4 AsymDPOP

In this section, we present a privacy-protecting inference-based complete algorithm for ADCOP, named AsymDPOP. We describe the motivation of the proposed algorithm in Sect. 4.1, and then elaborate on its utility propagation and value propagation phases in Sect. 4.2 and 4.3, respectively. Finally, a complexity analysis of AsymDPOP is provided in Sect. 4.4.

4.1 Motivation

The existing complete algorithms for ADCOPs are unsuitable for large-scale applications, since they require exponential messages to systematic search the solution space. In fact, these search-based complete algorithms cannot solve the random sensor networks with more than 16 sensors and the domain size of 8, as shown in the experimental results of Sect. 8.4.

On the other hand, inference-based complete algorithms (e.g., DPOP) which use local eliminations can solve DCOPs with only a linear number of messages. Unfortunately, they are not applicable for handling ADCOPs without PEAV since they require the total knowledge of each constraint to perform variable eliminations optimally. Concretely, when applying DPOP to solve a DCOP, once all the UTIL messages from its children have arrived, a variable can be eliminated locally without loss of completeness since all the functions related to it have been aggregated. However, this conclusion does not hold for ADCOPs as the functions related to the eliminated variable contain two-side private functions (i.e., the private functions on its side and the ones on its (pseudo) parents' side). As a result, the local elimination to a variable does not ensure the completeness without consideration of the private functions on its (pseudo) parents' sides.

Taking Fig. 3 as an example, since the private functions f_{13} and f_{23} are held by x_1 and x_2, respectively, x_3 has no knowledge about them when performing the local elimination. Actually, the functions involving x_3 include f_{13}, f_{23}, f_{32} and f_{31}. Thus, eliminating x_3 locally could lead to overestimated bias and offer no guarantee on the completeness. A naïve solution can be that (pseudo) parents transfer their private functions to their (pseudo) children to facilitate the local variable eliminations (e.g., x_1 and x_2 forward f_{13} and f_{23} to x_3 to eliminate x_3 locally), but this would incur an unacceptable loss of privacy.

To address the aforementioned issues, we propose the first complete inference algorithm for ADCOPs, named AsymDPOP. Next, we will detail the privacy-protecting utility propagation and the corresponding value propagation phases of AsymDPOP, respectively.

4.2 Utility propagation phase

As mentioned in Sect. 4.1, in the asymmetric setting, the local elimination to a variable does not ensure the completeness as the private functions on its (pseudo) parents' sides cannot be considered. On the other hand, the non-local elimination (Chen et al., 2020c) that defers the elimination to a variable at its parent also does not ensure the completeness since only the private function on its parent side is included. Therefore, we propose to postpone the elimination of a variable to its highest (pseudo) parent to aggregate all the private functions on its (pseudo) parents' sides, and name this elimination scheme as generalized non-local elimination (GNLE). Thus, a variable x_i in GNLE is responsible for eliminating the set of variables EV_i whose highest

Algorithm 1: Utility Propagation Phase in AsymDPOP (GNLE) for x_i

```
When Initialization:
  if x_i is a leaf then
    PropUtil()
When received UTIL(u_c) from x_c ∈ C(x_i):
  u_{c→i} ← u_c
  if x_i has received all UTIL from C(x_i) then
    if x_i is the root then
      start Value propagation phase
    else
      PropUtil()
Function PropUtil():
  compute u_{i→p} by Eq. (6)
  send UTIL(u_{i→p}) to P(x_i)
```

(pseudo) parent is x_i. Formally,

$$EV_i = \{x_j \in AC(x_i) | x_k \notin AP(x_j), \forall x_k \in Sep(x_i)\} \tag{2}$$

That is, a (pseudo) child x_j of x_i is eliminated at x_i if there is no (pseudo) parent of x_j ordered above x_i. Therefore, each variable in EV_i can be optimally eliminated from the aggregated utility table of x_i (i.e., u_i) since all the functions involving it have been aggregated. Here, u_i is the join of the local private functions of x_i and the received utility tables $u_{c\to i}, \forall x_c \in C(x_i)$, i.e.,

$$u_i = \left(\bigotimes_{x_j \in N(x_i)} f_{ij} \right) \otimes \left(\bigotimes_{x_c \in C(x_i)} u_{c\to i} \right)$$

Then, the utility table forwarded to its parent $u_{i\to p}$ can be obtained by:

$$u_{i\to p} = \min_{EV_i} u_i \tag{3}$$

Since $u_{i\to p}$ is a summation of multiple utility tables (according to Eq. (3)), $Anc(x_i)$ can hardly infer the private functions of x_i from $u_{i\to p}$ even though x_i does not eliminate its dimension (see Sect. 7 for a detailed analysis).

However, computing $u_{i\to p}$ by Eq. (3) incurs huge computational overhead, i.e., $O(d^{|UD_i \cup EV_i|})$, where $UD_i = u_{i\to p}.dims$. In fact, since $AP(x_i) \cap EV_i = \emptyset$ (according to Eq. (2)) and $AP(x_i) \cup AC(x_i) = N(x_i)$, the private functions of x_i related to $AP(x_i)$ can be moved to the outside of the min operation to optimize Eq. (3), which results in a significant reduction in computational overhead. That is,

$$u_{i\to p} = \left(\bigotimes_{x_j \in AP(x_i)} f_{ij} \right) \otimes \min_{EV_i} \left(\left(\bigotimes_{x_j \in AC(x_i)} f_{ij} \right) \otimes \left(\bigotimes_{x_c \in C(x_i)} u_{c\to i} \right) \right) \tag{4}$$

By this means, the time complexity is dominated by the maximum overhead of computing the local joint utility table (i.e., $O(d^{|UD_i|})$ or the elimination of

all the received utility tables (i.e., $O(d^{|\cup_{x_c \in C(x_i)} UD_c|})$). Therefore, Eq. (4) is a more efficient implementation for Eq. (3) except the case that $EV_i = \emptyset$.

Besides, EV_i can be divided into multiple disjoint subsets since the different branches are independent of each other in a pseudo tree. That is, EV_i can be partitioned into $EV_i^c = EV_i \cap B_i^c, \forall x_c \in C(x_i)$, such that $EV_i = \cup_{x_c \in C(x_i)} EV_i^c$ and $EV_i^c \cap EV_i^{c'} = \emptyset, \forall x_c, x_{c'} \in C(x_i), c \neq c'$.[1] Formally,

$$EV_i^c = \{x_j \in B_i^c | x_k \notin AP(x_j), \forall x_k \in Sep(x_i)\} \tag{5}$$

Accordingly, Eq. (3) can be further optimized by eliminating EV_i^c independently in different branches. That is,

$$u_{i \to p} = \left(\bigotimes_{x_j \in AP(x_i)} f_{ij} \right) \otimes \left(\bigotimes_{x_c \in C(x_i)} u_{i \to p}^c \right) \tag{6}$$

where $u_{i \to p}^c$ is the component of $u_{i \to p}$ regarding x_c, i.e.,

$$u_{i \to p}^c = \min_{EV_i^c} \left(\left(\bigotimes_{x_j \in B_i^c} f_{ij} \right) \otimes u_{c \to i} \right) \tag{7}$$

The time complexity of Eq. (6) is dominated by the maximum overhead of computing the local joint utility table (i.e., $O(d^{|UD_i|})$) or the elimination of each received utility table (i.e., $O(d^{\max_{x_c \in C(x_i)} |UD_c|})$), which is no greater than the one of Eq. (4) except the case that $EV_i = \emptyset$.

Algorithm 1 presents the sketch of the utility propagation phase of AsymDPOP (GNLE)[2]. The algorithm begins with leaf variables sending their utility tables computed by Eq. (6) to their parents via UTIL messages (line 1-2, 9-10). When receives a UTIL message from its child $x_c \in C(x_i)$, x_i stores the received utility table (line 3). Once all the UTIL messages from its children have arrived, x_i computes and propagates the utility table $u_{i \to p}$ to its parent if it is not the root (line 7-8, 9-10). Otherwise, the value propagation phase starts (line 5-6).

4.3 Value propagation phase

Contrary to the one in vanilla DPOP which determines the optimal assignment locally for each eliminated variable, the value propagation phase in AsymDPOP is specialized to accommodate GNLE. Specifically, since a variable x_i has

[1] It is noteworthy that computing EV_i^c does not require variables to exchange their relative positions in a pseudo tree. Specifically, each variable is associated with a counter which is initially set to the number of its (pseudo) parents. Then, the counter of a variable is propagated to its (pseudo) parents via UTIL messages. When its (pseudo) parent x_i receives the UTIL message containing it from $x_c \in C(x_i)$, the variable's counter decreases. Once its counter equals zero, that variable is added to EV_i^c.

[2] An example for AsymDPOP can be found in Appendix A.1.

Algorithm 2: Value Propagation in AsymDPOP for x_i

```
   When Initialization:
11 |  if x_i is the root then
12 |  |  compute u_i by Eq. (6)
13 |  |  v_i^* ← arg min_{x_i} u_i
14 |  |  PropValue({(x_i = v_i^*)})
   When received VALUE(PA^*) from P(x_i):
15 |  PropValue(PA^*)
   Function PropValue(PA^*):
16 |  foreach x_c ∈ C(x_i) do
17 |  |  PA_i^c ← PA^*_{[Sep(x_c)∪{x_c}∪ID_c]}
18 |  |  if EV_i^c ≠ ∅ then
19 |  |  |  compute V_i^{c*} for EV_i^c by Eq. (8)
20 |  |  |  PA_i^c ← PA_i^c ∪ {(x_j = V^{c*}_{i[x_j]})|∀x_j ∈ EV_i^c}
21 |  |  send VALUE(PA_i^c) to x_c
```

eliminated all the variables in $EV_i^c, \forall x_c \in C(x_i)$ during the utility propagation, it is responsible for selecting the optimal assignment to these eliminated variables. The optimal assignment to EV_i^c can be determined by the optimal assignment PA^* received from its parent and the received utility table $u_{c\to i}$. That is,

$$X_i^{c*} = \arg\min_{EV_i^c} \left(\left(\bigotimes_{x_j \in B_i^c} f_{ij}(PA^*_{[f_{ij}.dims]}) \right) \otimes u_{c\to i}(PA^*_{[u_{c\to i}.dims]}) \right) \quad (8)$$

Besides, since a subset of $Desc(x_i)$ is already eliminated by the variables in $Sep(x_i)$ (and hence their assignment is also determined), we need to forward the corresponding assignment of these variables as well. We formalize these variables as interface descendants of x_i, (i.e., ID_i). Formally,

$$ID_i = \{x_j \in Desc(x_i) | \exists x_k \in Sep(x_i), s.t., x_k \in AP(x_j)\} \quad (9)$$

Thus, the value message forwarded from x_i to its child $x_c \in C(x_i)$ in our algorithm would contain not only the assignment to $Sep(x_c)$ but also the assignment to the variables eliminated at x_i and $Sep(x_c)$ (i.e, $EV_i^c \cup ID_c$).

Algorithm 2 presents the sketch of the value propagation phase. The phase is initiated by the root variable selecting the optimal assignment for itself (line 12-13). Given the assignment either determined by its parent (line 15) or computed locally (line 14), a variable x_i selects the optimal assignment to $EV_i^c, \forall x_c \in C(x_i)$ by Eq. (8) (line 19), and propagates the assignment together with the determined assignment to x_c (line 20-21). The algorithm terminates when each leaf variable receives a VALUE message.

4.4 Complexity analysis

In this subsection, we theoretically analyze the complexity of AsymDPOP. We begin by showing the relationship between ID_i and EV_i, then we detail the complexity analysis.

Lemma 1 $ID_i = \left(C(x_i) \cup \left(\cup_{x_c \in C(x_i)} ID_c\right)\right) \backslash EV_i$

Proof We will first prove $ID_i \cap EV_i = \emptyset$. Then, we will prove $ID_i \cup EV_i = C(x_i) \cup (\cup_{x_c \in C(x_i)} ID_c)$.

According to Eq. (2), we can conclude

$$\begin{aligned} EV_i &= \{x_j \in AC(x_i) | x_k \notin AP(x_j), \forall x_k \in Sep(x_i)\} \\ &= \{x_j \in Desc(x_i) | x_i \in AP(x_j), x_k \notin AP(x_j), \forall x_k \in Sep(x_i)\} \end{aligned}$$

Since EV_i is a subset of $Desc(x_i)$ that does not connect with the variable in $Sep(x_i)$ while $ID(x_i)$ is a subset of $Desc(x_i)$ that connects with $Sep(x_i)$ (according to Eq. (9)), we have $ID_i \cap EV_i = \emptyset$.

On the basis of the above analysis, we can also draw the conclusion that

$$\begin{aligned} &ID_i \cup EV_i = \{x_j \in Desc(x_i) | \exists x_k \in Sep(x_i) \cup \{x_i\}, s.t., x_k \in AP(x_j)\} \\ &= \{x_j \in Desc(x_i) | \exists x_k \in Anc(x_i) \cup \{x_i\}, s.t., x_k \in AP(x_j)\} \\ &= \{x_j \in C(x_i) \cup (\cup_{x_c \in C(x_i)} Desc(x_c)) | \exists x_k \in Anc(x_i) \cup \{x_i\}, s.t., x_k \in AP(x_j)\} \\ &= \{x_j \in C(x_i) | \exists x_k \in Anc(x_i) \cup \{x_i\}, s.t., x_k \in AP(x_j)\} \\ &\quad \cup \left(\cup_{x_c \in C(x_i)} \{x_j \in Desc(x_c) | \exists x_k \in Anc(x_i) \cup \{x_i\}, s.t., x_k \in AP(x_j)\}\right) \\ &= C(x_i) \cup \left(\cup_{x_c \in C(x_i)} \{x_j \in Desc(x_c) | \exists x_k \in Anc(x_i) \cup \{x_i\}, s.t., x_k \in AP(x_j)\}\right) \\ &= C(x_i) \cup \left(\cup_{x_c \in C(x_i)} \{x_j \in Desc(x_c) | \exists x_k \in Anc(x_c), s.t., x_k \in AP(x_j)\}\right) \\ &= C(x_i) \cup \left(\cup_{x_c \in C(x_i)} \{x_j \in Desc(x_c) | \exists x_k \in Sep(x_c), s.t., x_k \in AP(x_j)\}\right) \\ &= C(x_i) \cup \left(\cup_{x_c \in C(x_i)} ID_c\right) \end{aligned}$$

The equation in the first to the second step and the sixth to the seven-step hold, since $Sep(x_i)$ composes of all the ancestors that share constraints with $Desc(x_i) \cup \{x_i\}$. According to Eq. (9), the equation in the seven to the last step holds. Thus, the lemma is proved.

Theorem 1 *The size of a UTIL message produced by a variable x_i is exponential in $|Sep(x_i) \cup \{x_i\} \cup ID_i|$. That is, $u_{i \to p}.dims = Sep(x_i) \cup \{x_i\} \cup ID_i$.*

Proof We will prove the theorem by induction.

Base case. When x_i is a leaf, the dimensions of the utility table $u_{i \to p}$ only consist of $AP(x_i) \cup \{x_i\}$ (line 1-3, 9-10 of Algo. 1). Since $Sep(x_i) = AP(x_i)$ and $ID_i = \emptyset$, the theorem holds for the base case.

Inductive hypothesis. Assume that the theorem holds for $x_c \in C(x_i)$, we now show the theorem holds for non-leaf variable x_i as well. According to Eqs. (2) and (3), we have

$$\begin{aligned} &u_{i \to p}.dims = \left(N(x_i) \cup \{x_i\} \cup \left(\cup_{x_c \in C(x_i)} u_{c \to i}.dims\right)\right) \backslash EV_i \\ &= \left(N(x_i) \cup \{x_i\} \cup \left(\cup_{x_c \in C(x_i)} Sep(x_c) \cup \{x_c\} \cup ID_c\right)\right) \backslash EV_i \\ &= \left(AP(x_i) \cup \left(\cup_{x_c \in C(x_i)} Sep(x_c)\right) \cup \{x_i\} \cup AC(x_i) \cup C\,(x_i) \cup \left(\cup_{x_c \in C(x_i)} ID_c\right)\right) \backslash EV_i \\ &= \left(AP(x_i) \cup \left(\cup_{x_c \in C(x_i)} Sep(x_c)\right) \cup \{x_i\} \cup AC(x_i) \cup \left(\cup_{x_c \in C(x_i)} ID_c\right)\right) \backslash EV_i \\ &= \left(AP(x_i) \cup \left(\cup_{x_c \in C(x_i)} Sep(x_c) \backslash \{x_i\}\right) \cup \{x_i\} \cup AC(x_i) \cup \left(\cup_{x_c \in C(x_i)} ID_c\right)\right) \backslash EV_i \\ &= \left(Sep(x_i) \cup \{x_i\} \cup AC(x_i) \cup \left(\cup_{x_c \in C(x_i)} ID_c\right)\right) \backslash EV_i \end{aligned}$$

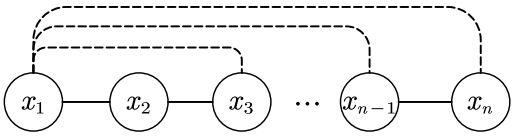


Fig. 4: A chain-like pseudo tree

The equation in the first to the second step holds due to the inductive hypothesis. Due to $(Sep(x_i) \cup \{x_i\}) \cap EV_i = \emptyset$, we can conclude

$$\begin{aligned}
&u_{i\to p}.dims = Sep(x_i) \cup \{x_i\} \cup \big(AC(x_i) \cup \big(\cup_{x_c \in C(x_i)} ID_c\big)\big) \backslash EV_i \\
&= Sep(x_i) \cup \{x_i\} \cup \big(PC(x_i) \cup C(x_i) \big(\cup_{x_c \in C(x_i)} ID_c\big)\big) \backslash EV_i \\
&= Sep(x_i) \cup \{x_i\} \cup (PC(x_i) \backslash EV_i) \cup \big(C(x_i) \big(\cup_{x_c \in C(x_i)} ID_c\big)\big) \backslash EV_i \\
&= Sep(x_i) \cup \{x_i\} \cup (PC(x_i) \backslash EV_i) \cup ID_i \\
&= Sep(x_i) \cup \{x_i\} \cup ID_i
\end{aligned}$$

According to Lemma 1, the equation in the third to the fourth step holds. Since the variables in $PC(x_i)$ must be eliminated at x_i or the variables in $Sep(x_i)$, we have $PC(x_i) \subseteq EV_i \cup ID_i$. Further, we have $PC(x_i) \backslash EV_i \subseteq ID_i$ based on $ID_i \cap EV_i = \emptyset$ proved in Lemma 1, and thus the equation in the fourth to the last step holds. Therefore, the theorem is proved.

5 Two tradeoffs for AsymDPOP

In this section, we first demonstrate that AsymDPOP suffers serious scalability problems in both memory and computation, and then propose two tradeoffs including a table-set propagation scheme in Sect. 5.1 and mini-batch elimination scheme in Sect. 5.2 to make AsymDPOP a practical algorithm. Finally, we discuss the relevance and difference between the proposed trade-offs and the related work in Sect. 5.3.

5.1 Table-set propagation scheme: A tradeoff between memory and privacy

As shown in Sect. 4.4, the size of a UTIL message forwarded by a variable x_i is exponential in $|Sep(x_i) \cup \{x_i\} \cup ID_i|$, which leads to unacceptable memory consumption when the built pseudo tree is poor. Consider the pseudo tree shown in Fig. 4. Since every variable has a constraint with the root variable, all the variables can only be eliminated at the root variable due to GNLE, which incurs a memory consumption of $O(d^n)$ due to the joint operation at each variable. Besides, a large utility table would also incur unacceptable computation overheads due to the joint operations in Eq. (6) (line 9 of Algo. 1).

Therefore, we consider reducing the dimension size of each propagated utility table by avoiding the unnecessary joint operations in Eq. (6), and hereby

each variable only needs to propagate a set of smaller utility tables rather than a joint and high-dimension utility table. However, completely discarding joint operations would incur privacy leakages. Taking Fig. 4 as an example again, if x_n chooses to propagate both $f_{n(n-1)}$ and f_{n1} without performing the joint operation to x_{n-1}, then x_{n-1} would know the private functions of x_n directly. Thus, we propose a compromise between memory consumption and privacy by introducing a parameter k_p to control the maximal number of dimensions of each local utility table. We refer to the tradeoff as a table-set propagation scheme (TSPS). In TSPS, the utility table set propagated by a variable x_i (i.e., $U_{i\to p}$) consists of its local utility tables (i.e., $U^i_{i\to p}$) and the result of handling the received utility tables (i.e., $U^c_{i\to p}, \forall x_c \in C(x_i)$), i.e.,

$$U_{i\to p} = \left\{ U^j_{i\to p} | \forall x_j \in \{x_i\} \cup C\left(x_i\right) \right\} \tag{10}$$

where $U^i_{i\to p}$ is computed by:

$$U^i_{i\to p} = \mathbf{PartF}\left(\underset{x_j \in AP(x_i)}{\cup} \{f_{ij}\}, k_p \right) \tag{11}$$

And $U^c_{i\to p}$ is calculated by:

$$U^c_{i\to p} = \mathbf{Elim}\left(\mathbf{JoinSet}\left(\underset{x_j \in B^c_i}{\cup} \{f_{ij}\}, U_{c\to i} \right), EV^c_i \right) \tag{12}$$

where Function **PartF** is used to implement TSPS by partitioning and joining the private functions of x_i related to $AP(x_i)$ into multiple local utility tables, such that the dimension size of each local utility table is no greater than k_p.[3] Functions **JoinSet** and **Elim** are applied to realize the join and elimination operations of the set of utility tables, respectively. It is noteworthy that the join of $\cup_{x_j \in B^c_i}\{f_{ij}\}$ and $U_{c\to i}$ does not increase the number of dimensions to $U_{c\to i}$. Additionally, propagating the utility table set $U_{i\to p}$ calculated by Eq. (10) can still guarantee the completeness of the algorithm since we can reconstruct $u_{i\to p}$ computed by Eq. (6) by joining all the utility tables in $U_{i\to p}$, i.e., $u_{i\to p} = \otimes_{u\in U_{i\to p}} u$.

Algorithm 3 presents the sketch of AsymDPOP-TSPS[4] (i.e., AsymDPOP with TSPS) which consists of a utility set propagation phase and a value propagation phase. Different from the utility propagation phase in AsymDPOP, the utility set propagation phase applies TSPS to propagate the set of utility tables calculated by Eq. (10) (line 12-13). In Eq. (10), x_i first joins its private functions w.r.t. its (pseudo) children with the received utility table set $U_{c\to i}$ (line 20-23), and then eliminates all the variables in EV^c_i from the joint utility table set (line 24-25). Finally, it obtains the propagated utility table set $U_{i\to p}$ by adding the eliminated result with its local utility tables obtained by

[3] It is worth noting that TSPS can only make sure that the size of each local utility table is not greater than k_p, and does not guarantee that the overall memory consumption is no greater than k_p.

[4] An example for AsymDPOP-TSPS can be found in Appendix A.2.

Algorithm 3: AsymDPOP-TSPS for x_i

```
   When Initialization:
1  |  if x_i is a leaf then
2  |  |  PropUtil()
   When received UTILSET(U_c) from x_c ∈ C(x_i):
3  |  U_{c→i} ← U_c
4  |  if x_i has received all UTILSET from C(x_i) then
5  |  |  if x_i is the root then
6  |  |  |  compute U_i by Eq. (10)
7  |  |  |  v_i^* ← arg min_{x_i}(⊗_{u∈U_i} u)
8  |  |  |  PropValue({(x_i = v_i^*)})
9  |  |  else
10 |  |  |  PropUtil()
   When received VALUE(PA^*) from P(x_i):
11 |  PropValue(PA^*)
   Function PropUtil():
12 |  compute U_{i→p} by Eq. (10)
13 |  send UTILSET(U_{i→p}) to P(x_i)
   Function PropValue(PA^*):
14 |  foreach x_c ∈ C(x_i) do
15 |  |  PA_i^c ← PA^*_{[Sep(x_c)∪{x_c}∪ID_c]}
16 |  |  if EV_i^c ≠ ∅ then
17 |  |  |  compute V_i^{c*} for EV_i^c by Eq. (13)
18 |  |  |  PA_i^c ← PA_i^c ∪ {(x_j = V^{c*}_{i[x_j]})|∀x_j ∈ EV_i^c}
19 |  |  send VALUE(PA_i^c) to x_c
   Function JoinSet(U, U_F):
20 |  foreach u_f ∈ U_F do
21 |  |  if ∃u ∈ U, s.t., u_f.dims ⊂ u.dims then
22 |  |  |  u ← u ⊗ u_f
23 |  return U
   Function Elim(U, EV):
24 |  U_EV ← {u|∀u ∈ U, u.dims ∩ EV ≠ ∅}
25 |  return (U\U_EV) ∪ {min_EV (⊗_{u∈U_EV} u)}
   Function PartF(U_F, k_p):
26 |  order U_F according to AP(x_i)'s levels
27 |  U ← ∅, u ← null
28 |  foreach u_f ∈ U_F do
29 |  |  if |u.dims| ≥ k_p then
30 |  |  |  U ← U ∪ {u},u ← null
31 |  |  u ← u ⊗ u_f
32 |  if u ≠ null then
33 |  |  U ← U ∪ {u}
34 |  return U
```

Function **PartF**. In the function, the private functions of x_i related to $AP(x_i)$ are partitioned into multiple small utility tables according to k_p (line 26-31). If there are any remaining functions, they would be joined into a utility table and then added into the local utility tables (line 32-33).

The value propagation phase is roughly the same as the one in AsymDPOP. Since the utility set propagation uses TSPS, there may have more than one utility table in $U_{c\to i}$. Thus, x_i must join all the tables in $U_{c\to i}$ and $\cup_{x_j\in B_i^c}\{f_{ij}\}$

before selecting the optimal assignment for EV_i^c (line 17). That is,

$$X_i^{c*} = \arg\min_{EV_i^c} \left(\left(\bigotimes_{x_j \in B_i^c} f_{ij}(PA^*_{[f_{ij}.dims]}) \right) \otimes \left(\bigotimes_{u \in U_{c \to i}} u(PA^*_{[u.dims]}) \right) \right) \tag{13}$$

Consider the pseudo tree shown in Fig. 4 again. Assume that $k_p = 3$, then variable x_n would propagate the utility set $u_{n \to n-1} = \{f_{n(n-1)} \otimes f_{n1}\}$ to x_{n-1}. Since there is no elimination at x_{n-1}, it is unnecessary to perform the joint operation. Thus, x_{n-1} would propagate the utility set $u_{n-1 \to n-2} = \{f_{(n-1)(n-2)} \otimes f_{(n-1)1}, f_{(n-1)n} \otimes u_{n \to n-1}\}$ to x_{n-2}. It can be concluded that TSPS in the example only requires $O(nd^3)$ space, which is much smaller than the one required by GNLE. Formally, we have the following theorem.

Theorem 2 *The size of each utility table of a variable x_i in AsymDPOP-TSPS is no greater than*

$$d^{\max(\min(|AP(x_i)|, k_p), \max_{x_c \in C(x_i)}(|Sep(x_c) \cup (ID_i \cap (ID_c \cup \{x_c\}))|)}$$

Proof According to Eq. (10), the set of utility tables propagated by x_i consists of the local utility tables of x_i (i.e., $U^i_{i \to p} = \mathbf{PartF}(\cup_{x_j \in AP(x_i)}\{f_{ij}\}, k_p)$) and the results obtained by handling the received utility tables (i.e., $U^c_{i \to p} = \mathbf{Elim}(\mathbf{JoinSet}(\cup_{x_j \in B_i^c}\{f_{ij}\}, U_{c \to i}), EV_i^c)), \forall x_c \in C(x_i)$). Since the private functions of x_i related to $AP(x_i)$ are partitioned into its local utility tables according to k_p, the maximal size of its local utility tables is

$$d^{\min(|AP(x_i)|, k_p)}$$

According to Theorem 1, the dimensions of each utility table received from child $x_c \in C(x_i)$ is a subset of $Sep(x_c) \cup \{x_c\} \cup ID_c$. Since TSPS omits the joint operation in Eq. (6), the dimensions of the utility tables in $U^c_{i \to p}$ is a subset of $(Sep(x_c) \cup \{x_c\} \cup ID_c) \setminus EV_i^c$. As $Sep(x_c) \cap EV_i^c = \emptyset$, we have $(Sep(x_c) \cup \{x_c\} \cup ID_c) \setminus EV_i^c = Sep(x_c) \cup ((\{x_c\} \cup ID_c) \setminus EV_i^c)$.

Besides, according to Lemma 1, we have

$$\begin{aligned} ID_i &= \left(C(x_i) \cup \left(\bigcup_{x_c \in C(x_i)} ID_c \right) \right) \setminus EV_i \\ &= \left(\bigcup_{x_c \in C(x_i)} (\{x_c\} \cup ID_c) \right) \setminus EV_i \\ &= \left(\bigcup_{x_c \in C(x_i)} (\{x_c\} \cup ID_c) \right) \setminus \left(\bigcup_{x_c \in C(x_i)} EV_i^c \right) \\ &= \bigcup_{x_c \in C(x_i)} ((\{x_c\} \cup ID_c) \setminus EV_i^c) \end{aligned}$$

Since $((\{x_c\} \cup ID_c) \backslash EV_i^c) \cap ((\{x_{c'}\} \cup ID_{c'}) \backslash EV_i^{c'}) = \emptyset, \forall x_c, x_{c'} \in C(x_i), c \neq c'$, we can conclude that

$$\begin{aligned}
&(\{x_c\} \cup ID_c) \backslash EV_i^c = ((\{x_c\} \cup ID_c) \backslash EV_i^c) \cap ID_i \\
&= ((\{x_c\} \cup ID_c) \cap ID_i) \setminus (EV_i^c \cap ID_i) \\
&= ((\{x_c\} \cup ID_c) \cap ID_i) \setminus \emptyset \\
&= (\{x_c\} \cup ID_c) \cap ID_i
\end{aligned}$$

Therefore, we have $Sep(x_c) \cup ((\{x_c\} \cup ID_c) \backslash EV_i^c) = Sep(x_c) \cup (ID_i \cap (\{x_c\} \cup ID_c))$. That is, the maximal size of the utility tables obtained by processing the received utility tables is

$$d^{\max\limits_{x_c \in C(x_i)}(|Sep(x_c) \cup (ID_i \cap (ID_c \cup \{x_c\}))|)}$$

Thus, the theorem holds.

Here, we only give an upper bound on the memory consumption of AsymDPOP-TSPS, because the space complexity of an agent in AsymDPOP-TSPS is not only related to the parameter of TSPS (i.e., k_p) but also to the constraint graph structure of the problem to be solved. For example, when the constraint graph is fully connected, all variables are eliminated at the root agent, and thus the space complexity of an agent a_i in AsymDPOP-TSPS is $d^{\min(k_p, |AP(a_i)|)}$ while that of a_i in AsymDPOP is d^n.

Theorem 3 *The memory consumption of AsymDPOP-TSPS is no greater than the one of AsymDPOP.*

Proof According to Theorems 1 and 2, to prove the theorem, we only need to prove that $\max(\min(|AP(x_i)|, k_p), \max_{x_c \in C(x_i)}(|Sep(x_c) \cup (ID_i \cap (ID_c \cup \{x_c\}))|) \leq |Sep(x_i) \cup \{x_i\} \cup ID_i|$.

Since $Sep(x_i) = AP(x_i) \cup (\cup_{x_c \in C(x_i)} Sep(x_c)) \backslash \{x_i\}$, we have $AP(x_i) \subseteq Sep(x_i)$ and $Sep(x_c) \subseteq Sep(x_i) \cup \{x_i\}, \forall x_c \in C(x_i)$. Moreover, due to $ID_i \cap (ID_c \cup \{x_c\}) \subseteq ID_i, \forall x_c \in C(x_i)$, we can conclude that $|AP(x_i)| \leq |Sep(x_i) \cup \{x_i\} \cup ID_i|$ and $\max_{x_c \in C(x_i)}(|Sep(x_c) \cup (ID_i \cap (ID_c \cup \{x_c\}))|) \leq |Sep(x_i) \cup \{x_i\} \cup ID_i|$. Thus, the theorem is proved.

5.2 Mini-batch elimination scheme: A tradeoff between time and space

In AsymDPOP-TSPS, a joint utility table could be factorized to a set of smaller utility tables, which allows us to reduce the computational efforts when performing the variable eliminations by a decrease-and-conquer strategy. Taking Fig. 4 as an example, to perform the elimination, x_1 in AsymDPOP-TSPS ($k_p = 2$) eliminates $x_2, \ldots x_n$ over a big utility table by Eq. (10), which requires $O(d^n)$ operations (i.e., $\min_{\{x_2,\ldots,x_n\}}(f_{12} \otimes f_{21} \otimes \cdots \otimes f_{1n} \otimes f_{n1} \otimes f_{n(n-1)} \otimes f_{(n-1)n})$). In fact, we could exploit the structure of each small utility

Algorithm 4: MBES

Function `Elim_MBES`(U, EV, k_e)**:**
- $EVGroup \leftarrow$ **GroupEV**(U, EV)
- $EVSet \leftarrow \cup_{EV \in EVGroup}$**PartEV**$(EV, k_e)$
- **foreach** $EV \in EVSet$ **do**
  - $U_{EV} \leftarrow \{u | \forall u \in U, u.dims \cap EV \neq \emptyset\}$
  - $U \leftarrow (U \backslash U_{EV}) \cup \{\min_{EV}(\otimes_{u \in U_{EV}} u)\}$
- **return** U

Function `GroupEV`(U, EV)**:**
- $G_{EV} \leftarrow \{u.dims \cap EV | \forall u \in U\}, RM_{EV} \leftarrow \emptyset$
- **while** $\exists EV', EV'' \in G_{EV}, EV', EV'' \notin RM_{EV}, s.t. EV' \cap EV'' \neq \emptyset$ **do**
  - $EV' \leftarrow EV' \cup EV''$,$RM_{EV} \leftarrow RM_{EV} \cup EV''$
- **return** $G_{EV} \backslash RM_{EV}$

Function `PartEV`(EV, k_e)**:**
- $EVSet \leftarrow \emptyset, EV' \leftarrow \emptyset$
- **foreach** $ev \in EV$ **do**
  - **if** $|EV'| \geq k_e$ **then**
    - $EVSet \leftarrow EVSet \cup \{EV'\}$, $EV' \leftarrow \{ev\}$
  - **else**
    - $EV' \leftarrow EV' \cup \{ev\}$
- **if** $EV' \neq \emptyset$ **then**
  - random select an element $EV'' \in EVSet$
  - $EV'' \leftarrow EV'' \cup EV'$
- **return** $EVSet$

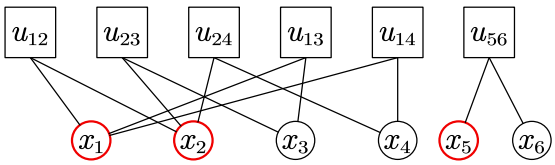


Fig. 5: A set of utility tables

table to reduce computational complexity by arranging the min operators among them. That is,

$$\min_{x_2}\left(f_{12} \otimes f_{21} \otimes \cdots \otimes \min_{x_n}\left(f_{1n} \otimes f_{n1} \otimes f_{n(n-1)} \otimes f_{(n-1)n}\right)\right)$$

which can be solved recursively from x_n to x_2 and the overall complexity is $O(nd^3)$. In other words, we reduce the computational complexity by exploiting the independence among utility tables to avoid unnecessary combinations of utility tables.

However, completely distributing the min operators into every variable would incur high memory consumption, as a min operator could implicitly join utility tables to a big and indivisible table. Although the problem can be alleviated by carefully arranging the min operators, it could be impossible to find the optimal sequence of eliminations in practice. Consider the utility tables shown as a factor graph in Fig. 5, where square nodes represent utility tables, circle nodes represent variables and the red circles represent the variables to be eliminated. Obviously, no matter how to arrange the elimination sequence,

a 3-ary utility table must appear when eliminating x_1 or x_2. Instead, if we jointly eliminate both x_1 and x_2, the maximal number of dimensions is 2. We thus overcome the issue by introducing a parameter k_e which specifies the minimal number of variables eliminated in a min operator (i.e., the size of a batch), and refer to the tradeoff as a mini-batch elimination scheme (MBES).

MBES can be deployed into AsymDPOP-TSPS by replacing Function **Elim** in Eq. (10) with **ElimMBES**, named AsymDPOP-TSPS-MEBS. Algorithm 4 gives the sketch of MBES which consists of the following three parts. The first part realized by Function **GroupEV** is used to divide the eliminated variables into disjoint variable groups such that the eliminated variables in each group share at least one common utility table (line 1, 7-10). This part is crucial, since eliminating independent variables jointly is equivalent to eliminating them individually. Taking Fig. 5 for example, if we set $k_e = 2$ and let x_2, x_5 be a batch, a 4-ary utility table over x_1, x_3, x_4 and x_6 still appear even if x_2 and x_5 are jointly eliminated. Then, the second part implemented by Function **PartEV** is applied to partition the eliminated variables in each group into several sets ($EVSet$) according to k_e (line 2, 11-20). The last part is carried out to traverse $EVSet$ to eliminate the utility tables (line 3-5). In detail, for each set in $EVSet$ (i.e., EV), a variable x_i performs variable eliminations to the utility tables related to EV and replaces these utility tables with the eliminated result. The process terminates when all the eliminated variable sets are exhausted.

5.3 Discussions

Although the proposed trade-offs share some similarities with the existing work, including mini-bucket elimination (Dechter and Rish, 2003) and super bucket (Kask et al., 2005), they are fundamentally different.

Both TSPS and mini-buckets elimination aim to reduce memory consumption by partitioning a set of constraint functions into multiple buckets. However, TSPS trades privacy loss for smaller memory consumption, while mini-buckets elimination trades solution quality for smaller memory consumption. Specifically, mini-bucket elimination performs variable elimination locally on these buckets whose dimensions contain the eliminated variable, and thus it can only get approximate solutions. As these buckets are directly propagated to the ancestors without eliminating local variables in TSPS, the ancestors can infer the private function costs of their (pseudo) children accordingly. Nonetheless, TSPS can guarantee the completeness of the algorithm. That is because the buckets involving the eliminated variables have included all the constraint functions of those variables and are joint into a utility table when executing the variable elimination.

Besides, MBES is different from super bucket although both of them aim to optimize the variable elimination procedure. Concretely, super bucket uses more computational efforts to save memory consumption by eliminating multiple variables at once rather than one at a time. While MBES uses more

memory consumption to reduce computational efforts by eliminating multiple variables sequentially rather than jointly. MBES is also different from mini-buckets elimination. Specifically, mini-buckets elimination partitions a set of constraint functions into multiple buckets and then performs variable elimination on these buckets. While MBES divides a set of eliminated variables into multiple batches and then eliminates these batches from the related buckets sequentially.

6 Memory-bounded inference for ADCOPs

Although TSPS tries to reduce memory consumption by limiting the joint operation to local constraint functions, the space complexity is still exponential according to Theorem 2. That is, both AsymDPOP and AsymDPOP-TSPS are impracticable for solving large-scale problems within the limited memory budget. Therefore, it is imperative to develop *memory-bounded* inference algorithms for ADCOPs. In this section, we extend RMB-DPOP, a state-of-the-art memory-bounded algorithm for DCOPs, to reduce the excessive memory consumption of AsymDPOP. We present the proposed algorithm named RMB-AsymDPOP in Sect. 6.1. Moreover, to improve the scalability of RMB-AsymDPOP, we propose RMB-AsymDPOP-TSPS by deploying TSPS on RMB-AsymDPOP in Sect. 6.2.

6.1 RMB-AsymDPOP

A major issue of directly applying RMB-DPOP to reduce the excessive memory consumption of AsymDPOP is that RMB-DPOP cannot handle the redundancy due to the non-local elimination. Specifically, in GNLE, the variables eliminated at a variable x_i are EV_i and the dimensions of the utility table propagated by x_i are $Sep(x_i) \cup \{x_i\} \cup ID_i$. Thus, based on the computation of CC nodes in Eq. (1), the CC nodes enumerated at x_i (i.e., CC_i) can be computed by:

$$CC_i = \begin{cases} CClist_r \cap \left(\underset{x_c \in C(x_i)}{\cup} (Sep(x_c) \cup \{x_c\} \cup ID_c) \right) & x_i \text{ is a CR node} \\ CClist_r \cap EV_i & \text{otherwise} \end{cases}$$

where $CClist_r$ is the CC nodes of the cluster containing x_i.

In fact, the elimination of CC nodes can be optimized by considering the branch independence of the variable eliminations (i.e., $EV_i = \cup_{x_c \in C(x_i)} EV_i^c$ and $EV_i^c \cap EV_i^{c'} = \emptyset, \forall x_c, x_{c'} \in C(x_i), c \neq c'$). That is, we propose to enumerate CC nodes in different branches independently and refer this enumeration scheme as branch-independent distributed enumeration mechanism (BI-DEM). Concretely, in BI-DEM, x_i is only responsible for enumerating the subset of CC_i that belongs to the branch x_c (i.e, CC_i^c) for its child $x_c \in C_i^{in}$. Here, C_i^{in}

Algorithm 5: RMB-AsymDPOP for x_i (message passing)

When `Initialization`**:**
start **Labeling phase**
When ***Labeling phase** finished*:
initialize $done_i$ and $u^c_{i\to p}, \forall x_c \in C(x_i)$
$C^{in}_i \leftarrow \{x_c | \forall x_c \in C(x_i), |Sep(x_c) \cup \{x_c\} \cup ID_c| > k_{mb}\}$
LabelItself() and compute CC^c_i by Eq. (14)
if $TYPE(x_i) = CR$ **then**
  PropInstantiation()
else if $TYPE(x_i) = NORMAL \wedge x_i$ *is a leaf* **then**
  PropUtil()
When `received NORMALUTIL` (u_c) *from* $x_c \in C(x_i) \backslash C^{in}_i$**:**
$u_{c\to i} \leftarrow u_c$ and compute $u^c_{i\to p}$ by Eq. (7)
if x_i *has received all* $NORMALUTIL$ *from* $C(x_i)$ **then**
  if x_i *is the root* **then**
    start **Value propagation phase**
  else if $TYPE(x_i) = NORMAL$ **then**
    PropUtil()
When `received INSTANTIATION` (Ins_i) *from* $P(x_i)$**:**
if $TYPE(x_i) = CL$ **then**
  PropUtil()
else
  PropInstantiation()
When `received BOUNDEDUTIL` (u_c) *from* $x_c \in C^{in}_i$**:**
$u_{c\to i} \leftarrow u_c$ and update $u^c_{i\to p}$ by Eq. (16)
if $Ins^c_i.next() \neq null$ **then**
  $Ins^c_i \leftarrow Ins^c_i.next()$
  send INSTANTIATION$(Ins^c_i \cup Ins_{i[u_{c\to i}.dims]})$ to x_c
else
  $done_i \leftarrow done_i \cup \{x_c\}$
  if $|done_i| = |C^{in}_i|$ **then**
    initialize $done_i$ and $u^c_{i\to p}, \forall x_c \in C^{in}_i$
    if x_i *is the root* **then**
      start **Value propagation phase**
    else
      PropUtil()

is a subset of $C(x_i)$, each of which is in the cluster. And CC^c_i can be calculated by:

$$CC^c_i = \begin{cases} CClist_r \cap (Sep(x_c) \cup \{x_c\} \cup ID_c) & x_i \text{ is a CR node} \\ CClist_r \cap EV^c_i & \text{otherwise} \end{cases} \tag{14}$$

Thus, we have $CC_i = \cup_{x_c \in C^{in}_i} CC^c_i$ based on Eq. (14). Next, we will detail the process of computing a bounded utility table in RMB-AsymDPOP.

When a cluster node x_i receives an instantiation (i.e., Ins_i) from its parent, the bounded utility table propagated by x_i can be computed by Eq. (15) after all the inference results from $C(x_i)$ have arrived.

$$u_{i\to p} = \left(\bigotimes_{x_j \in AP(x_i)} f_{ij}\left(Ins_i\right) \right) \otimes \left(\bigotimes_{x_c \in C(x_i)} u^c_{i\to p}\left(Ins_i\right) \right) \tag{15}$$

where $u^c_{i\to p}$ is the result of handling the received utility table $u_{c\to i}$. Specifically, if x_c is the node outside the cluster (i.e., $x_c \in C\left(x_i\right) \backslash C^{in}_i$), then $u^c_{i\to p}$ can be

Algorithm 5: RMB-AsymDPOP for x_i (auxiliary functions)

```
   Function PropUtil():
31 |  if TYPE(x_i) ∈ {NORMAL, CR} then
32 |  |  compute u_{i→p} by Eq. (6)
33 |  |  send NORMALUTIL(u_{i→p}) to P(x_i)
34 |  else
35 |  |  compute u_{i→p} by Eq. (15)
36 |  |  send BOUNDEDUTIL(u_{i→p}) to P(x_i)
   Function PropInstantiation():
37 |  foreach x_c ∈ C_i^in do
38 |  |  Ins_i^c ← the first instantiation of CC_i^c
39 |  |  if TYPE(x_i) = CR then
40 |  |  |  send INSTANTIATION(Ins_i^c) to x_c
41 |  |  else
42 |  |  |  send INSTANTIATION(Ins_i^c ∪ Ins_{i[u_{c→i}.dims]}) to x_c
   Function LabelItself ():
43 |  if |Sep(x_i) ∪ {x_i} ∪ ID_i| ≤ k_mb then
44 |  |  if C_i^in = ∅ then
45 |  |  |  TYPE(x_i) ← NORMAL
46 |  |  else
47 |  |  |  TYPE(x_i) ← CR
48 |  else
49 |  |  if C_i^in = ∅ then
50 |  |  |  TYPE(x_i) ← CL
51 |  |  else
52 |  |  |  TYPE(x_i) ← CN
```

computed by Eq. (7). Otherwise, $u^c_{i\to p}$ is calculated by:

$$u^c_{i\to p} = \mathbf{Update}\left(u^c_{i\to p}, \min_{EV^c_i \setminus CC^c_i}\left(\left(\bigotimes_{x_j\in B^c_i} f_{ij}\left(Ins_i \cup Ins^c_i\right)\right) \otimes u_{c\to i}\right)\right) \quad (16)$$

where Ins^c_i is the instantiation of CC^c_i.

Algorithm 5 gives the sketch of RMB-AsymDPOP[5]. After the labeling phase ends, each variable x_i obtains the CC nodes of the cluster containing x_i, labels its node type (line 4, 43-52) and computes the CC nodes CC^c_i for all the children in C^{in}_i by Eq. (14) (line 4). Then, a CR node x_i starts the memory-bounded inference by forwarding the first instantiation of CC^c_i to all the children in C^{in}_i (line 5-6, 37-42).

When receiving an instantiation Ins_i, x_i forwards the bounded utility table computed by Eq. (15) to its parent via a BOUNDEDUTIL message if it is a CL node (i.e., the cluster node with $C^{in}_i = \emptyset$) (line 15-16). Otherwise, it augments Ins_i by the first instantiation of CC^c_i (i.e., Ins^c_i) and propagates the extended instantiation to all the children in C^{in}_i (line 18, 37-42). Once x_i receives a BOUNDEDUTIL message from its child $x_c \in C^{in}_i$, it joins its private functions related to B^c_i, eliminates all the variables in EV^c_i and then updates the cache $u^c_{i\to p}$ with the eliminated result as shown in Eq. (16) (line 19). Afterward, it finds the next instantiation of CC^c_i for x_c. If the next instantiation exists, it replaces Ins^c_i with the new instantiation and propagates that instantiation to

[5] We omit the details of the value propagation phase due to its similarity to the one in MB-DPOP, and also omit the details of ISM and CACHE in RMB-AsymDPOP due to their similarity to the ones in RMB-DPOP.

x_c (line 20-22). Otherwise, it marks x_c as a node that completes the memory-bounded utility propagation under Ins_i (line 20). When all the children in C_i^{in} have completed, x_i sends the bounded utility table to its parent via a BOUNDEDUTIL message if it is a cluster node (line 25, 34-36).

6.2 RMB-AsymDPOP-TSPS

Although RMB-AsymDPOP can be applied in memory-limited cases, it is still unable to solve large-scale problems. In fact, it cannot solve the random sensor networks with more than 9 sensors and the domain size of 8 under the memory constraint (i.e., $k_{mb} = 5$), as shown in the experimental results of Sect. 8.4. Thus, we propose to improve the scalability of RMB-AsymDPOP by combining it with TSPS. We do not incorporate MBES into RMB-AsymDPOP-TSPS since MBES could increase the memory consumption of the algorithm and make it exceed the memory budget.

Since each agent propagates a utility table set during the utility set propagation phase in AsymDPOP-TSPS, there are two issues when deploying TSPS to RMB-AsymDPOP. The first is that the dimensions of each propagated utility table are only determined after the table is created, and thus are unknown before the utility set propagation phase and the labeling phase. However, these dimensions are indispensable for determining the clusters and CC nodes during the labeling phase. Therefore, we propose a preprocessing phase to simulate the utility table set propagation such that the dimensions of the propagated utility tables are known before the labeling phase.

The second is that there still exists redundancy when adopting the memory-bounded utility propagation of RMB-AsymDPOP into RMB-AsymDPOP-TSPS. Concretely, during the memory-bounded utility set propagation, each cluster node conditions its utility table set by the instantiation of CC nodes and then propagates the set of bounded utility tables to its parent iteratively. In fact, for the utility tables within the memory budget could be propagated only once, since they do not require any instantiation to limit their dimensions. Thus, we propose a two-phase utility set propagation scheme including a normal utility set propagation phase to forward the utility tables below the memory limit, and a bounded utility set propagation phase to iteratively propagate the remaining utility tables with some dimensions fixed by the instantiation of CC nodes. In the following, we will detail the preprocessing phase and the two-phase utility set propagation scheme, respectively.

Algorithm 6 presents the sketch of the preprocessing phase. The phase begins with leaf variables sending the dimensions of their local utility tables to their parents via DIMSET messages (line 2-3), where these dimensions are obtained by Function **PartD** to implement TSPS (line 1, 14-22). Once receiving a DIMSET message from its child $x_c \in C(x_i)$, x_i stores the received dimensions of each utility and attaches the private functions related to B_i^c into each received utility (line 4-7). Afterward, it applies Function **ElimD** to simulate the variable elimination process to obtain the dimensions of $U_{i\to p}^c$

Algorithm 6: Preprocessing Phase in RMB-AsymDPOP-TSPS for x_i

When `Initialization`**:**
1 $U^i_{i\to p} \leftarrow$ **PartD**$(\cup_{x_j \in AP(x_i)} \{f_{ij}\}, k_p)$
2 **if** x_i *is a leaf* **then**
3 send DIMSET$(U^i_{i\to p})$ to $P(x_i)$
When `received DIMSET` (U_c) *from* $x_c \in C(x_i)$**:**
4 $U_{c\to i} \leftarrow U_c$
5 **foreach** $x_j \in B^c_i$ **do**
6 **if** $\exists u \in U_{c\to i}, s.t., \{x_i, x_j\} \subset u.dims$ **then**
7 $u.joint \leftarrow u.joint \cup \{f_{ij}\}$
8 $U^c_{i\to p} \leftarrow$ **ElimD**$(U_{c\to i}, EV^c_i)$
9 **if** x_i *has received all DIMSET from* $C(x_i)$ **then**
10 **if** x_i *is not the root* **then**
11 send DIMSET$(\cup_{x_j \in \{x_i\}\cup C(x_i)} U^j_{i\to p})$ to $P(x_i)$
12 **else**
13 end **Perprocessing phase**
Function `PartD` (U_F, k_p)**:**
14 $U \leftarrow \emptyset$,$u \leftarrow$ *an empty bucket*
15 order U_F according to $AP(x_i)$'s levels
16 **foreach** $u_f \in U_F$ **do**
17 **if** $|u.dims| \geq k_p$ **then**
18 $U \leftarrow U \cup \{u\}$,$u \leftarrow$ *an empty bucket*
19 $u.dims \leftarrow u.dims \cup u_f.dims$
20 **if** $|u.dims| > 0$ **then**
21 $U \leftarrow U \cup \{u\}$
22 **return** U
Function `ElimD` (U, EV)**:**
23 $EVGroup^c_i \leftarrow$ **GroupEV**(U, EV)
24 **foreach** $EV \in EVGroup^c_i$ **do**
25 $U_{EV} \leftarrow \{u | \forall u \in U, u.dims \cap EV \neq \emptyset\}$
26 $u_{EV} \leftarrow$ *an empty bucket with dimensions of* $(\cup_{u\in U_{EV}} u.dims) \backslash EV$
27 $U \leftarrow (U \backslash U_{EV}) \cup \{u_{EV}\}$
28 **return** U
Function `GroupEV` (U, EV)**:**
29 $G_{EV} \leftarrow \{u.dims \cap EV | \forall u \in U\}$
30 **while** $\exists EV', EV'' \in G_{EV}, s.t.\ EV' \cap EV'' \neq \emptyset$ **do**
31 $EV' \leftarrow EV' \cup EV''$, $G_{EV} \leftarrow G_{EV} \backslash EV''$
32 **return** G_{EV}

(line 8, 23-28). Until receiving all the DIMSET messages from its children, x_i adds the dimensions of $U^c_{i\to p}, \forall x_c \in C(x_i)$ with the dimensions of its local utility tables, and then sends the joint dimensions to its parent if it is not the root (line 11). Otherwise, the preprocessing phase ends (line 13).

Algorithm 7 gives the sketch of the two-phase utility set propagation scheme. When the labeling phase ends, a variable x_i labels itself (line 36, 90-99) and computes the CC nodes for each child in C^{in}_i (line 36). Afterward, a leaf variable x_i forwards the local utility tables whose dimension size no greater than k_{mb} to its parent via a NORMALUTILSET message, and removes them from its local utility tables (line 37-38, 75-79).

When receiving a NORMALUTILSET message from $x_c \in C(x_i)$, x_i merges the received utility tables, and joins them with its corresponding private functions (line 39-41). Afterward, it uses Function **ElimNU** to perform the variable eliminations, and merges the eliminated result into the cache $U^c_{i\to p}$ (line 42). When all the NORMALUTILSET messages from its children have arrived, x_i

Algorithm 7: Two-phase utility set propagation scheme in RMB-AsymDPOP-TSPS for x_i (message passing)

When `Initialization`**:**
33 start **Labeling phase**
When ***Labeling phase*** *finished*:
34 initialize $done_i$
35 $C_i^{in} \leftarrow \{x_c | \forall x_c \in C(x_i), \exists u \in U_{c\rightarrow i}, s.t., |u.dims| > k_{mb}\}$
36 **LabelItself**() and compute CC_i^c by Eq. (14)
37 **if** x_i *is a leaf* **then**
38 **PropNormalUtil**()
When `received NORMALUTILSET` (U_c) *from* $x_c \in C(x_i)$**:**
39 $U_{c\rightarrow i} \leftarrow$ **Merge**$(U_{c\rightarrow i}, U_c)$
40 **foreach** $u \in U_{c\rightarrow i}, u.data \neq null, u_f \in u.joint$ **do**
41 $u.data \leftarrow u.data \otimes u_f$,$u.joint \leftarrow u.joint \backslash \{u_f\}$
42 $U_{i\rightarrow p}^c \leftarrow$ **Merge**$(U_{i\rightarrow p}^c$,**ElimNU**$(x_c))$
43 **if** x_i *has received all NORMALUTIL from* $C(x_i)$ **then**
44 **if** $TYPE(x_i) = CR$ **then**
45 **PropInstantiation**()
46 **else**
47 **if** x_i *is the root* **then**
48 start **Value propagation phase**
49 **else**
50 **PropNormalUtil**()
When `received INSTANTIATION` (Ins_i) *from* $P(x_i)$**:**
51 **if** $TYPE(x_i) = CL$ **then**
52 **foreach** $x_c \in C(x_i)$ **do**
53 $U_{i\rightarrow p}^c \leftarrow$ **ElimBU**(x_c)
54 **PropBoundedUtil**()
55 **else**
56 **PropInstantiation**()
When `received BOUNDEDUTILSET` (U_c) *from* $x_c \in C_i^{in}$**:**
57 $U_{c\rightarrow i} \leftarrow$ **Merge**$(U_{c\rightarrow i}, U_c)$
58 **foreach** $u \in U_{c\rightarrow i}, u_f \in u.joint$ **do**
59 $u.data \leftarrow u.data \otimes u_f(Ins_i \cup Ins_i^c)$
60 $U_{i\rightarrow p}^c \leftarrow$ **Update**$(U_{i\rightarrow p}^c$,**ElimBU**$(x_c))$
61 **if** $Ins_i^c.next() \neq null$ **then**
62 $Ins_i^c \leftarrow Ins_i^c.next()$
63 send INSTANTIATION$(Ins_i^c \cup Ins_{i[aug_dim_i^c]})$ to x_c
64 **else**
65 $done_i \leftarrow done_i \cup \{x_c\}$
66 **if** $|done_i| = |C_i^{in}|$ **then**
67 initialize $done_i$ and $U_{i\rightarrow p}^c, \forall x_c \in C_i^{in}$
68 **if** $TYPE(x_i) = CR$ **then**
69 **if** x_i *is the root* **then**
70 start **Value propagation phase**
71 **else**
72 **PropNormalUtil**()
73 **else**
74 **PropBoundedUtil**()

starts the iterative bounded utility set propagation phase if it is a CR node (line 44-45). Otherwise, it continues to propagate the joint utility tables to its parent if it is not the root (line 49-50).

The iterative bounded utility propagation phase begins with a CR node x_i forwarding the first instantiation of CC_i^c to all the children in C_i^{in} (line 44-45, 83-89). When receiving an instantiation from its parent, x_i calls Function **ElimBU** to eliminate the remaining eliminated variables (i.e., $EV_i^c \backslash CC_i^c$) from the received utility tables (line 52-53), and forwards the set of bounded

Algorithm 7: Two-phase utility set propagation scheme in RMB-AsymDPOP-TSPS for x_i (auxiliary functions)

Function `PropNormalUtil()`:
75 | **foreach** $u \in U^i_{i\to p}, |u.dims| \leq k_{mb}$ **do**
76 | | $u.data \leftarrow \otimes_{x_j \in u.dims, i \neq j} f_{ij}$
77 | $nU \leftarrow \{u | \forall u \in U^j_{i\to p}, u.data \neq null, \forall x_j \in \{x_i\} \cup C(x_i)\}$
78 | $U^j_{i\to p} \leftarrow U^j_{i\to p} \backslash (nU \cap U^j_{i\to p}), \forall x_j \in \{x_i\} \cup C(x_i)$
79 | send NORMALUTILSET(nU) to $P(x_i)$
Function `PropBoundedUtil()`:
80 | **foreach** $u \in U^i_{i\to p}$ **do**
81 | | $u.data \leftarrow \otimes_{x_j \in u.dims, i \neq j} f_{ij}(Ins_i)$
82 | send BOUNDEDUTIL($\cup_{x_j \in \{x_i\} \cup C(x_i)} U^j_{i\to p}$) to $P(x_i)$
Function `PropInstantiation()`:
83 | **foreach** $x_c \in C^{in}_i$ **do**
84 | | $Ins^c_i \leftarrow$ the first instantiation of CC^c_i
85 | | **if** $TYPE(x_i) = CR$ **then**
86 | | | send INSTANTIATION(Ins^c_i) to x_c
87 | | **else**
88 | | | $aug\_dim^c_i \leftarrow \cup_{u \in U_{c\to i}, |u.dims| > k_{mb}} u.dims$
89 | | | send INSTANTIATION($Ins^c_i \cup Ins_{i[aug\_dim^c_i]}$) to x_c
Function `LabelItself ()`:
90 | **if** $|u.dims| \leq k_{mb}, \forall u \in (\cup_{x_j \in \{x_i\} \cup C(x_i)} U^j_{i\to p})$ **then**
91 | | **if** $C^{in}_i = \emptyset$ **then**
92 | | | $TYPE(x_i) \leftarrow NORMAL$
93 | | **else**
94 | | | $TYPE(x_i) \leftarrow CR$
95 | **else**
96 | | **if** $C^{in}_i = \emptyset$ **then**
97 | | | $TYPE(x_i) \leftarrow CL$
98 | | **else**
99 | | | $TYPE(x_i) \leftarrow CN$
Function `ElimNU` (x_c):
100 | $U \leftarrow \{u | \forall u \in U_{c\to i}, u.data \neq null\}$
101 | **foreach** $EV \in EVGroup^c_i$ **do**
102 | | $U_{EV} \leftarrow \{u | \forall u \in U_{c\to i}, u.dims \cap EV \neq \emptyset\}$
103 | | $U \leftarrow U \backslash U_{EV}, d_{EV} \leftarrow \cup_{u \in U_{EV}} u.dims \backslash EV$
104 | | **if** $|d_{EV}| \leq k_{mb}, u.data \neq null, \forall u \in U_{EV}$ **then**
105 | | | $U_{c\to i} \leftarrow U_{c\to i} \backslash U_{EV}, EVGroup^c_i \leftarrow EVGroup^c_i \backslash \{EV\}$
106 | | | $U \leftarrow U \cup \{\min_{EV} (\otimes_{u \in U_{EV}} u)\}$
107 | $U_{c\to i} \leftarrow U_{c\to i} \backslash U$
108 | return U
Function `ElimBU` (x_c):
109 | $U \leftarrow \{u(Ins_i \cup Ins^c_i) | \forall u \in U_{c\to i}\}$
110 | **foreach** $EV \in EVGroup^c_i$ **do**
111 | | $U_{EV} \leftarrow \{u | \forall u \in U, u.dims \cap EV \neq \emptyset\}$
112 | | $U \leftarrow U \backslash U_{EV} \cup \{\min_{EV \backslash CC^c_i} (\otimes_{u \in U_{EV}} u)\}$
113 | return U

utility tables via a BOUNDEDUTILSET to its parent if it is a CL node (line 51-54, 80-82). Otherwise, x_i augments Ins_i by the first instantiation of CC^c_i (i.e., Ins^c_i) and propagates the extended instantiation to all the children in C^{in}_i (line 55-56, 83-89).

Once receiving the bounded utility tables from its child $x_c \in C^{in}_i$, x_i handles the received utility tables by merging and joining them with its corresponding private functions (line 57-59). Then, it calls Function **ElimBU** to

perform the variable eliminations, and updates the cache $U^c_{i\to p}$ with the eliminated result (line 60). Afterward, it finds the next instantiation of CC^c_i for x_c to explore. If the next instantiation exists, x_i replaces Ins^c_i with the instantiation and propagates the new instantiation to x_c (line 61-63). Otherwise, it marks x_c as a node that completes the memory-bounded utility set propagation (line 65). After all the children in C^{in}_i have completed, x_i sends the bounded utility table set to its parent via a BOUNDEDUTILSET message (line 66, 74, 80-82).

7 Privacy of AsymDPOP and its variants

In this section, we focus on measuring the constraint privacy loss when solving an ADCOP with entropy, and then analyze the constraint privacy leaked out by the message-passing during the execution of the proposed algorithms.

Privacy protection 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 this paper, we aim to use entropy to measure the constraint privacy loss incurred by the ADCOP algorithms in Sect. 8.4. The entropy for each constraint $f_{ij} \in F$ is calculated by the ratio between the number of possible tables considering revealed information and the total number of possible tables (Grinshpoun et al., 2013; Zivan et al., 2020b). Without loss of generality, we assume that all costs in each constraint f_{ij} are positive integers and smaller than d_{max} (here, $d_{max} = \max_{x_i \in X, d_i \in D_i} d_i$), and then the total number of possible tables of f_{ij} is $d_{max}^{|D_i||D_j|}$. When k_{ij} entries in f_{ij} are revealed to x_j, and then the total number of possible tables of f_{ij} is reduced to $d_{max}^{|D_i||D_j|-k_{ij}}$. Therefore, the privacy loss of f_{ij} can be formally defined by:

$$1 - \frac{\log_2 d_{max}^{|D_i||D_j|-k_{ij}}}{\log_2 d_{max}^{|D_i||D_j|}}$$

By considering the amount of the missing information about all the constraints (i.e., $f_{ij}, \forall x_i \in X, x_j \in N(x_i)$), we can obtain the amount of privacy loss incurred by an ADCOP algorithm. That is,

$$\begin{aligned} &\frac{1}{|X|} \sum_{x_i \in X} \frac{1}{|N(x_i)|} \sum_{x_j \in N(x_i)} \left(1 - \frac{\log_2 d_{max}^{|D_i||D_j|-k_{ij}}}{\log_2 d_{max}^{|D_i||D_j|}}\right) \\ &= 1 - \frac{1}{|X|} \sum_{x_i \in X} \frac{1}{|N(x_i)|} \sum_{x_j \in N(x_i)} \frac{\log_2 d_{max}^{|D_i||D_j|-k_{ij}}}{\log_2 d_{max}^{|D_i||D_j|}} \end{aligned} \tag{17}$$

Similar to (Grinshpoun et al., 2013; Litov and Meisels, 2017; Chen et al., 2020a), to evaluate the performance of different ADCOP complete algorithms on the constraint privacy protection, we focus on measuring the constraint privacy loss when solving a distributed Asymmetric MaxCSP (Maximization Constraint Satisfaction Problems). Since there are only two values (i.e., zero

and non-zero) for the constraint cost of each assignment to two constrained variables in a distributed Asymmetric MaxCSP, we have $d_{max} = 2$, then Eq. (17) can be rewritten by:

$$
\begin{aligned}
&= 1 - \frac{1}{|X|} \sum_{x_i \in X} \frac{1}{|N(x_i)|} \sum_{x_j \in N(x_i)} \frac{\log_2 2^{|D_i||D_j|-k_{ij}}}{\log_2 2^{|D_i||D_j|}} \\
&= 1 - \frac{1}{|X|} \sum_{x_i \in X} \frac{1}{|N(x_i)|} \sum_{x_j \in N(x_i)} \frac{|D_i||D_j| - k_{ij}}{|D_i||D_j|} \\
&= \frac{1}{|X|} \sum_{x_i \in X} \frac{1}{|N(x_i)|} \sum_{x_j \in N(x_i)} \frac{k_{ij}}{|D_i||D_j|}
\end{aligned}
$$

Next, we will analyze the constraint privacy loss revealed by the message passing during the execution of the proposed algorithms. Since a VALUE message only contains the optimal assignment, it reveals no information. That is, all the leaked information comes from UTIL messages in AsymDPOP or UTILSET messages in AsymDPOP-TSPS-MBES. We do not discuss the constraint privacy loss incurred by BOUNDEDUTIL messages in RMB-AsymDPOP and BOUNDEDUTILSET messages in RMB-AsymDPOP-TSPS since they exploit an iterative propagation manner to propagate the same information in UTIL and UTILSET messages, respectively. Additionally, we do not consider the privacy loss incurred by the situation that a variable x_i can infer private functions from third-party variables (i.e., those variables that do not share constraints with x_i), since this problem can be solved by hiding variable names using code names (Léauté and Faltings, 2013).

A UTIL message forwarded from its child $x_c \in C(x_i)$ to x_i (i.e., $u_{c\to i}$) in AsymDPOP would cause the direct privacy loss on the constraints between x_i and B_i^c, i.e., a half of the constraints privacy divulged in the worst case. That is because $B_i^c \subset u_{c\to i}.dims$ according to Theorem 1, and thus x_i can infer the actual constraint costs of $f_{ji}, \forall x_j \in B_i^c$ by eliminating all the variables except x_i and x_j from $u_{c\to i}$.

Like a UTIL message in AsymDPOP, a UTILSET message propagated from $x_c \in C(x_i)$ to x_i (i.e., $U_{c\to i}$) in AsymDPOP-TSPS-MBES would also cause the direct privacy loss on the constraints between x_i and B_i^c. Specifically, for each variable $x_j \in B_i^c$, it is eliminated at the variable in $(\{x_i\} \cup Sep(x_i)) \cap AP(x_j)$ due to GNLE, and thus there must exist a utility table $u \in U_{c\to i}$ such that $f_{ji}.dims \subset u.dims$ due to $\otimes_{u \in U_{c\to i}} u = u_{c\to i}$ by Eq. (10). Therefore, x_i can still infer the actual constraint costs of $f_{ji}, \forall x_j \in B_i^c$ from the corresponding utility table in $U_{c\to i}$.

8 Experimental evaluations

In this section, we first compare AsymDPOP and AsymDPOP-TSPS-MBES and empirically study how the parameters affect AsymDPOP-TSPS-MBES. Then, we present a comparison of the performances of RMB-AsymDPOP

and RMB-AsymDPOP-TSPS under different memory budgets. Finally, we empirically evaluate the proposed algorithms with state-of-the-art complete algorithms for ADCOPs including PT-ISABB, AsymPT-FB, SyncABB-1ph, ATWB and the vanilla DPOP with PEAV formulation (PEAV_DPOP). We do not compare the proposed methods against the search-based DCOP algorithms with the PEAV formulation (e.g., PEAV_SyncBB, PEAV_AFB and PEAV_BnB-ADOPT) and SyncABB-2ph, since they are worse than SyncABB-1ph and ATWB as shown in the experimental results of (Grinshpoun et al., 2013). The experiments are conducted on an i7-7820x workstation, and we set the timeout to 30 minutes for each algorithm. Unless otherwise stated, we set the memory limit of each agent to 256KB to simulate the real-world scenarios, e.g., low-powered embedded devices (Duan et al., 2018). Therefore, for the experiments with the domain size of 8, we set $k_{mb} = 5$ to stay below the limit. Similarly, when varying domain size from 4 to 8, k_{mb} is selected from the range of 8 down to 4 accordingly. For each experiment, we generate 50 random instances and report the medians as the results.

8.1 Experimental configurations

In the experiments, the algorithms are benchmarked on four types of problems including random ADCOPs, scale-free networks, random sensor networks and random Asymmetric MaxDCSPs.

- *Random ADCOPs* are an asymmetric version of random DCOPs, a general form of the distributed constraint optimization problems in which a set of agents are randomly constrained with each other. In the experiment, the number of agents, graph density and domain size are varied to evaluate the performance of the algorithms. (see the detailed configurations in Sect. 8.3 and 8.4). In addition, the constraint costs are uniformly selected from $[0, 100]$.
- *Scale-free networks* are the networks whose degree distributions follow power laws (Barabási and Albert, 1999). In the experiment, Barabási-Albert (BA) model is used to generate the constraint graph topology where the domain size is set to 8, the number of agents varies from 10 to 14, and an initial number of agents to 8 (i.e., $m_0 = 8$). At each iteration of the BA model procedure, a new agent is added and connected to 4 (i.e., $m_1 = 4$) other agents with a probability proportional to the number of links that the existing agents already have. The range of constraint costs in scale-free networks is the same as the ones in random ADCOPs.
- *Random sensor network* (Nguyen et al., 2019) consists of a set of sensors arranged in a grid, where each sensor is constrained with all of its neighboring sensors and can move along its 2D plane or stay stationary. That is, the value of each sensor corresponds to the discrete motion of that sensor. For example, a sensor can move in two cardinal directions or stay stationary if it has three possible values. In the experiment, we fix the domain size of each sensor to 8, and vary the number of sensors in a square from 2×2

(i.e., $|A| = 4$) to 6×6 (i.e., $|A| = 36$). And the range of constraint costs in random sensor networks is the same as the ones in random ADCOPs.
- *Asymmetric MaxDCSPs* are an asymmetric version of MaxDCSPs, a subset of random DCOPs with all the constraint costs equal to one (Grinshpoun et al., 2013; Modi et al., 2005). Asymmetric MaxDCSPs are classified by the number of agents, domain size, graph density and constraint tightness (i.e., the probability that a cost of the assignment to two constrained variables is non-zero). In the experiment, we consider Asymmetric MaxDCSPs with 10 agents, the graph density of 0.4, the domain size of 10, and the tightness varying from 0.1 to 0.8.

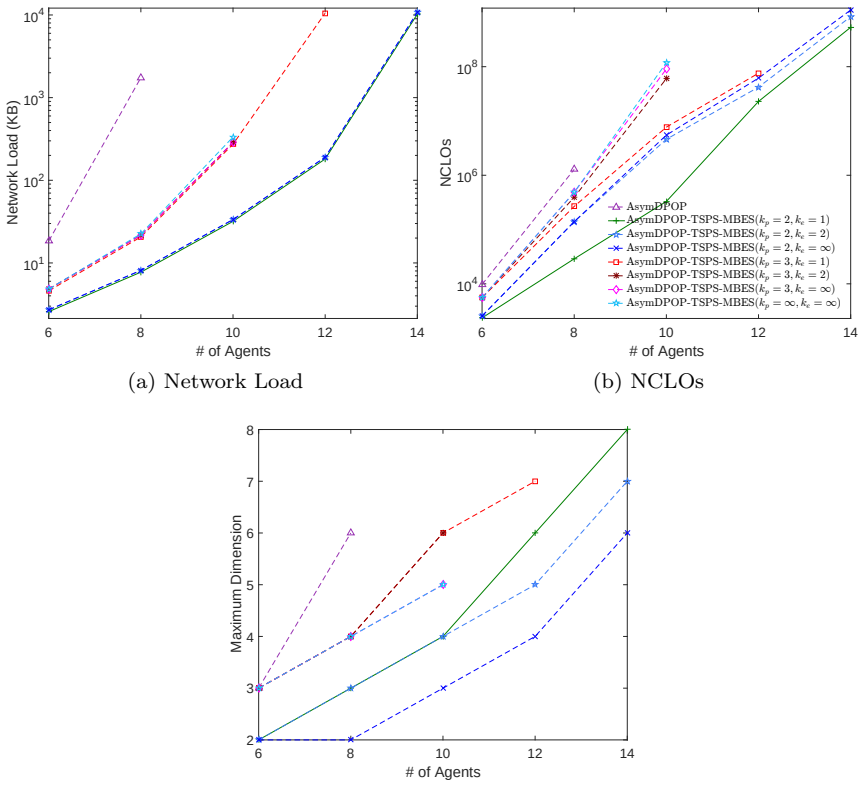
8.2 Evaluation metrics

- *The network load* is the total size of messages exchanged during the algorithm execution. Since search-based algorithms have an exponential number of messages with linear size, but inference-based algorithms require a linear number of messages of exponential size, the network load is used as a metric to evaluate the communication overheads of each algorithm in the experiments.
- *Non-Concurrent Logical Operations (NCLOs)* (Netzer et al., 2012) is a generalization of NCCCs (Zivan and Meisels, 2006) based on the concept of atomic operations (Gershman et al., 2008). In the experiments, NCLOs are used as a metric to evaluate hardware-independent runtime in which the logic operations in inference-based algorithms are access to the utility tables while the ones in search-based algorithms are constraint checks.
- *Entropy* (Brito et al., 2009) is used to measure the amount of the missing information of agents' private constraints during distributed constraint satisfaction problems solving. We adopt the method mentioned in Sect. 7 to calculate this metric when solving Asymmetric MaxDCSPs.
- *The maximal dimension size* is used to measure the maximal memory consumption of inference-based algorithms in the experiments.

8.3 Performance comparisons: AsymDPOP and its variants

To demonstrate the effects of MBES and TSPS, we benchmark AsymDPOP when combining MBES with different k_e and TSPS with different k_p on random ADCOPs with different agent numbers. Specifically, we consider the random ADCOPs with the domain size of 8, the density of 0.4, and the number of agents varying from 6 to 14. We do not limit the memory budget of each algorithm to better show the effectiveness of MBES.

Figure 6 presents the experimental results. Given enough memory, AsymDPOP can only solve the problems with less than 10 agents due to the runtime limit. Besides, its performance is far worse than AsymDPOP-TSPS-MBES even when $k_p = \infty$ on all the evaluation metrics. That is because the computation consumption of an agent in AsymDPOP is not only exponential in

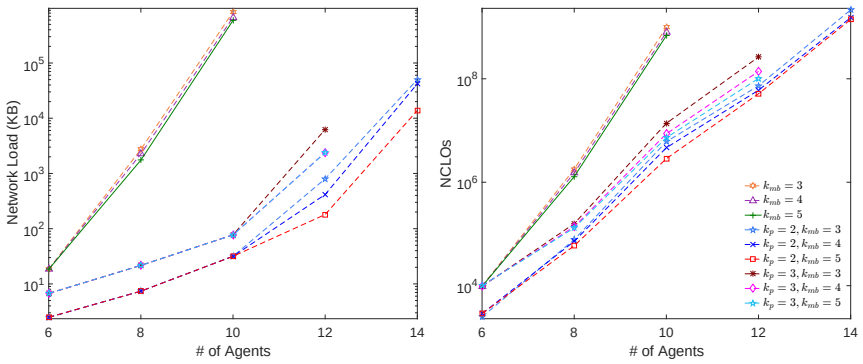


(a) Network Load (b) NCLOs

(c) The maximal dimension size

Fig. 6: Performance comparison for AsymDPOP and AsymDPOP-TSPS-MBES(k_p, k_e) on random ADCOPs ($6 \leq |A| \leq 14$, $|D_i| = 8$ and $p = 0.4$)

the number of its separators but also the number of its non-eliminated descendants as shown in Theorem 1. On the other hand, this phenomenon verifies that TSPS can reduce the huge computation overheads of AsymDPOP by avoiding unnecessary joint operations. In more detail, when solving the problems with 8 agents, the maximal dimension size of AsymDPOP is 6 and that of AsymDPOP-TSPS-MBES ($k_p = \infty$) is just 4, as shown in Fig. 6(c). Further, with the decrease of k_p, more joint operations are discarded and thus the dimension size of each utility table becomes smaller. It is worth noting that a small k_p can greatly improve the performance of AsymDPOP-TSPS-MBES but may lead to a high privacy loss as shown in Fig. 13(c). As for MBES, AsymDPOP-TSPS-MBES with a small k_e reduces NCLOs but produces larger intermediate utility tables, which indicates the necessity of the tradeoff. Besides, the performance of MBES dramatically degenerates when combined with TSPS given a larger k_p. That is because the utility tables contain more dimensions in the scenario, and a utility table would be traversed



(a) Network Load (b) NCLOs

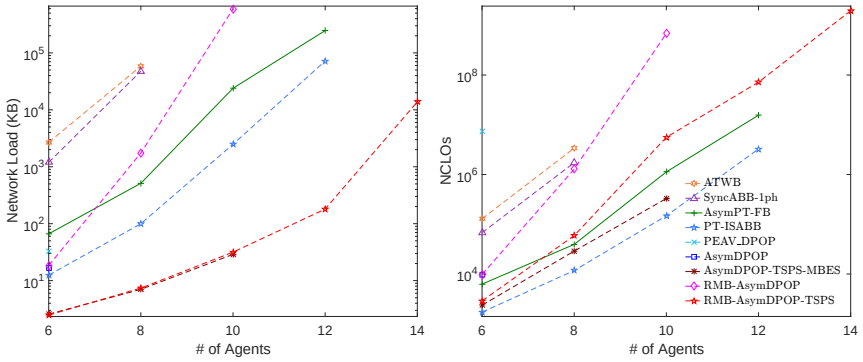
Fig. 7: Performance comparison for RMB-AsymDPOP(k_{mb}) and RMB-AsymDPOP-TSPS(k_p, k_{mb}) on random ADCOPs ($6 \leq |A| \leq 14$, $|D_i| = 8$ and $p = 0.4$)

more frequently when performing sequential variable eliminations. Thus, we set $k_p = 2$ and $k_e = 1$ for AsymDPOP-TSPS-MBES in the comparison experiments of Sect. 8.4.

To analyze how the parameters for the memory-bounded algorithms (i.e., k_{mb} for RMB-AsymDPOP and k_p and k_{mb} for RMB-AsymDPOP-TSPS) affect their performance, we vary k_{mb} from 3 to 5 and k_p from 2 to 3, and benchmark them on the above configuration. Figure 7 shows the experimental results. It can be seen that RMB-AsymDPOP cannot solve the problems with more than 10 agents. Once combining RMB-AsymDPOP with TSPS, the scalability and performance have been improved dramatically. Specifically, when solving the problems with 10 agents, the network load and NCLOs of RMB-AsymDPOP are 18,000 and 110 times higher than that of RMB-AsymDPOP-TSPS, respectively. Moreover, RMB-AsymDPOP-TSPS with $k_p = 2$ can solve all the problems. Additionally, it can be concluded from the figure that a larger k_{mb} can enhance the performance of our algorithms on all evaluation metrics. This indicates that a greater memory budget results in fewer CC nodes and communication overheads. Therefore, we set $k_{mb} = 5$ for RMB-AsymDPOP, and $k_p = 2$ and $k_{mb} = 5$ for RMB-AsymDPOP-TSPS in the comparison experiments of Sect. 8.4.

8.4 Performance comparisons: AsymDPOP, its variants and its competitors

Figure 8 presents the comparison results on the configuration of Sect. 8.3. It can be observed that AsymDPOP can only solve the problems with 6 agents. The algorithm cannot solve the problems with more than 10 agents, even when combined with TSPS and MBES. That is because the memory consumption

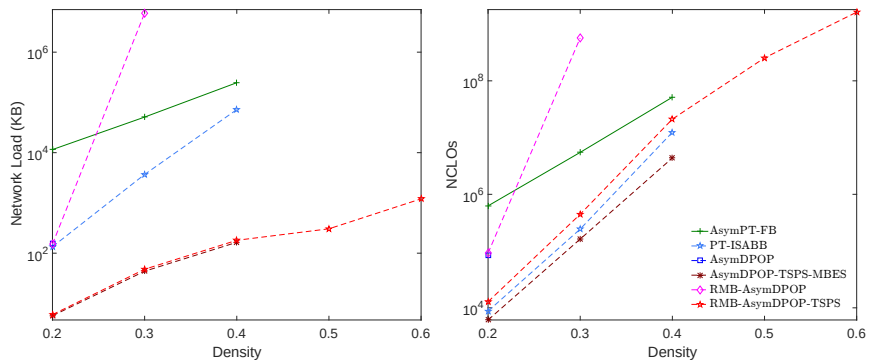


(a) Network Load (b) NCLOs

Fig. 8: Performance comparison on random ADCOPs ($6 \leq |A| \leq 14$, $|D_i| = 8$ and $p = 0.4$)

of AsymDPOP and AsymDPOP-TSPS-MBES exceeds the limit when handling larger problems. To be precise, the maximal dimension size of AsymDPOP when solving the problems with 8 agents and the one of AsymDPOP-TSPS-MBES when dealing with the problems with 12 agents are both 6, which is greater than the memory constraint (i.e., $k_{mb} = 5$), as shown in Fig. 6(c). This phenomenon states that the memory consumption of AsymDPOP even when combined with TSPS is exponential, as shown in Theorem 2. Their scalability is improved once combined with the memory-bounded inference, which demonstrates the indispensability of RMB-AsymDPOP and RMB-AsymDPOP-TSPS under the limited memory condition. Furthermore, compared to the search-based solvers, the proposed algorithms with TSPS exhibit great superiorities in terms of the network load. It is because the search-based algorithms use a message-passing manner to explicitly exhaust the solution space, which is quite expensive, especially when solving problems with numerous agents. In contrast, the proposed algorithms like AsymDPOP-TSPS-MBES and RMB-AsymDPOP-TSPS incur fewer communication overheads since they follow an inference protocol. On the other hand, although PEAV_DPOP also uses the inference protocol, it still suffers from a severe scalability problem and can only solve the problems with 6 agents. That is because DPOP with the PEAV formulation introduces the mirror variables to enforce the consistency of assignment to all variables, which significantly increases the complexity. More specifically, each utility table in PEAV_DPOP contains the dimensions of mirror variables, which markedly increases the memory consumption.

Figure 9 presents the performance comparison under different densities. Specifically, we consider the random ADCOPs with the number of agents of 12, the domain size of 8, and the density varying from 0.2 to 0.6. We do not include SynchABB-1ph and AWTB since they cannot solve all the problems

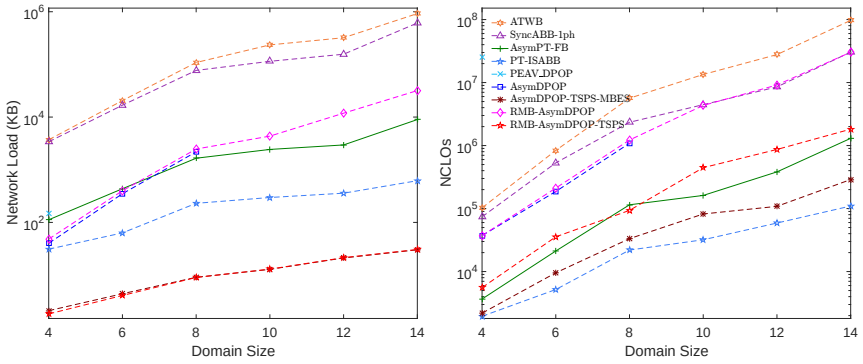


(a) Network Load (b) NCLOs

Fig. 9: Performance comparison on random ADCOPs ($|A| = 12$, $|D_i| = 8$ and $0.2 \leq p \leq 0.6$)

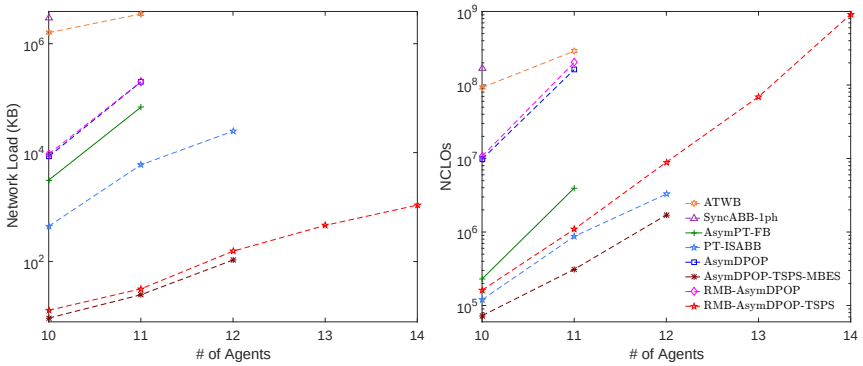
within 30 minutes, and PEAV_DPOP because it cannot solve the problems under the limited memory even when the density is 0.2. It can be concluded from the figure that AsymDPOP with TSPS (i.e., AsymDPOP-TSPS-MBES and RMB-AsymDPOP-TSPS) incurs significantly fewer communication overheads, which demonstrates the merit of avoiding unnecessary joint operations. Besides, it is interesting to find that compared to the one of AsymDPOP without TSPS (i.e., AsymDPOP and RMB-AsymDPOP), the network load of AsymDPOP with TSPS increases much slowly. That is because as the density increases, eliminations are more likely to happen at the top of a pseudo tree. On the other hand, since unnecessary joint operations are avoided in TSPS, eliminations are the major source of the dimension increase. As a result, agents propagate low dimension utility tables most of the time.

Figure 10 presents the performance comparison under different domain sizes. Specifically, we consider the random ADCOPs with the number of agents of 8, the density of 0.4, and the domain size varying from 4 to 14. It can be observed that the communication and computation overheads of all the algorithms increase exponentially as the domain size grows. Among them, AsymDPOP cannot solve the problems with a domain size greater than 8. This is due to the fact that the maximal message size in inference-based complete algorithms is exponential based on the domain size. As a result, it incurs huge coordination overheads and runs out of memory when facing complex problems. Once incorporating TSPS, its scalability and performance are greatly improved. Besides, AsymDPOP with TSPS also exhibits great advantages over state-of-the-art search-based algorithms on the network load, although the algorithm always generates larger messages than the search-based solvers. More specifically, the network load required by AsymDPOP-TSPS-MBES is at most five percent of the one produced by PT-ISABB. These phenomenons demonstrate the necessity of TSPS for the proposed algorithms. Besides, we



(a) Network Load (b) NCLOs

Fig. 10: Performance comparison on random ADCOPs ($|A| = 8$, $4 \leq |D_i| \leq 14$ and $p = 0.4$)

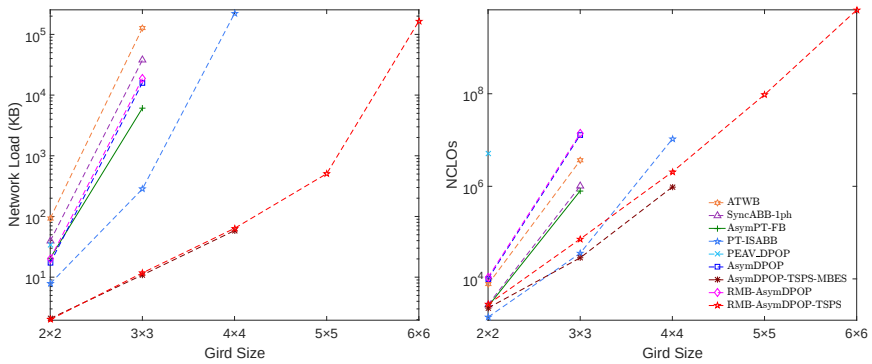


(a) Network Load (b) NCLOs

Fig. 11: Performance comparison on scale-free networks ($10 \leq |A| \leq 14$, $|D_i| = 8$, $m_0 = 8$ and $m_1 = 4$)

can see that the network load of AsymDPOP-TSPS-MBES is approximately the same as that of its memory-bounded counterpart without MBES. The reason could be that the utility tables propagated by these two algorithms are the same. In fact, the average induced width in the experiment is only 3, which is smaller than the memory constraint (i.e., $k_{mb} \geq 4$).

Figure 11 shows the performance comparison on the scale-free networks with different agent numbers. PEAV_DPOP is not included since it cannot solve the problems under the limited memory budget even when $|A| = 10$. It can be seen from the figure that AsymDPOP and its memory-bounded version (i.e., RMB-AsymDPOP) suffer from a serious scalability problem. Concretely, AsymDPOP and RMB-AsymDPOP can only solve the problems when



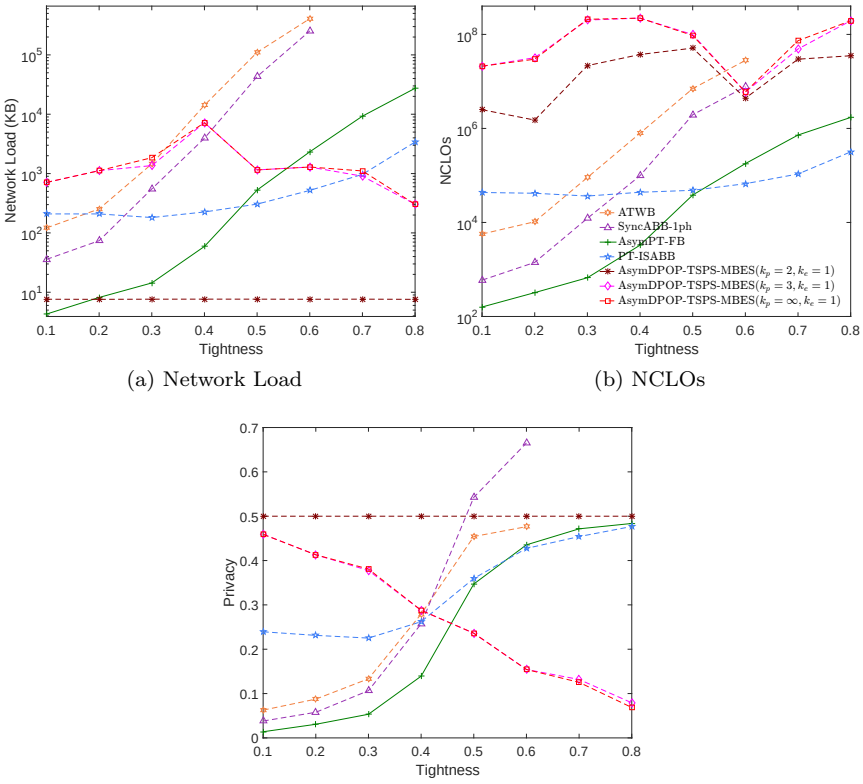
(a) Network Load (b) NCLOs

Fig. 12: Performance comparison on random sensor networks ($9 \leq |A| \leq 36$ and $|D_i| = 8$)

$|A| \leq 11$. Besides, they are inferior to the latest search-based algorithms like AsymPT-FB and PT-ISABB on all the evaluation metrics. That is because AsymDPOP requires tremendous communication and computation overheads, as stated in Theorem 1. Once incorporating TSPS, their scalability and performance have improved. Specifically, AsymDPOP-TSPS-MBES can scale up to the problems with 12 agents and RMB-AsymDPOP-TSPS can solve all the problems in this configuration. When solving the problems with 11 agents, the network load and NCLOs of AsymDPOP without TSPS are about 6,300 and 180 times higher than that of AsymDPOP with TSPS, respectively. These experimental results demonstrate the indispensability of TSPS for the proposed methods when handling structured problems.

Figure 12 presents the performance comparison on the random sensor networks with different sensor numbers. In this configuration, each sensor only shares constraints with its four neighbor sensors, therefore the density of the problems decreases with the increase of the number of sensors. However, as the number of sensors grows, the search space becomes larger and thus both the communication and computation overheads of all the algorithms increase exponentially as shown in the figure. Among them, our algorithms with TSPS exhibit significant superiority on all the metrics, which indicates the merit of the dynamic programming strategy combined with the table set propagation scheme for ADCOPs when the density is relatively low. Without TSPS, their scalability declines dramatically. For example, AsymDPOP and RMB-AsymDPOP cannot solve the problems with more than 9 sensors. These results indicate that TSPS is indispensable for AsymDPOP and RMB-AsymDPOP when dealing with real-world problems.

In the last experiment, we consider the privacy loss of each algorithm when solving Asymmetric MaxDCSPs with different tightness. For the proposed algorithms, we only consider AsymDPOP-TSPS-MBES with $k_e = 1$ for MBES,



(a) Network Load (b) NCLOs

(c) Privacy Loss

Fig. 13: Performance comparison on Asymmetric MaxDCSPs ($|A| = 10$, $|D_i| = 10$, $p_1 = 0.4$ and $0.1 \leq p_2 \leq 0.8$)

since the propagated utility tables are only related to the value of k_p in TSPS. We do not include AsymDPOP since it runs out of memory on all the problems, RMB-AsymDPOP because it cannot solve the problems within 30 minutes, and RMB-AsymDPOP-TSPS since it incurs the same privacy loss with AsymDPOP-TSPS-MBES given a fixed k_p. Figure 13 presents the experimental results. It can be seen that all the search-based algorithms suffer from exponential overheads, but AsymDPOP-TSPS-MBES seems to be unaffected. This is because all the search-based complete algorithms need to exhaust the solution space by a systematic search, and it is becoming more difficult for them to prune the solution space as the tightness grows. However, the performance of AsymDPOP-TSPS-MBES does not depend on constraint tightness, as it uses a dynamic programming paradigm to solve the problems. As for the privacy loss, it can be concluded from Fig. 13(c) that as the tightness grows, the search-based algorithms leak more privacy while the inference-based algorithms leak less privacy. It is because these search-based algorithms mainly

rely on a direct disclosure mechanism to aggregate the private costs and need to traverse more proportions of the search space when solving the problems with high tightness. In contrast, the inference-based algorithms accumulate utility through the pseudo tree, and a variable x_i could infer the private costs of its (pseudo) child x_c when the received utility table involving both x_i and x_c contains zero entries or is a binary table which is not a result of eliminations. Thus, AsymDPOP-TSPS-MBES ($k_p = 2$) leaks almost half of privacy. On the other hand, AsymDPOP-TSPS-MBES ($k_p \geq 3$) incurs much lower privacy loss when solving the high tightness problems, since the number of assignment combinations with non-zero cost grows as the tightness increases.

9 Conclusion

In this paper, we present the first inference-based complete algorithm for ADCOPs, namely AsymDPOP, by using generalized non-local elimination which facilitates the aggregation of utility in an asymmetric environment. To enhance the scalability of AsymDPOP, we introduce a table-set propagation scheme (TSPS) to reduce the memory consumption and a mini-batch elimination scheme (MBES) to reduce the computation operations in the utility propagation phase. Furthermore, to ensure the proposed algorithms can still scale up to large-scale problems under the limited memory budget, we adapt the memory-bounded inference to AsymDPOP and AsymDPOP-TSPS, where a branch-independent distributed enumeration mechanism and a two-phase utility set propagation scheme are introduced to reduce the redundancy in memory-bounded inference. Finally, we theoretically show the complexity of the proposed AsymDPOP and AsymDPOP-TSPS. And our empirical evaluation demonstrates the superiority of AsymDPOP and its variants over the state-of-the-art as well as the vanilla DPOP with PEAV formulation. Additionally, when incorporating TSPS and the memory-bounded inference, the scalability and performance of AsymDPOP are dramatically improved under the limited memory cases.

In the future, we will try to improve the proposed algorithms in the following two aspects. First, we plan to investigate the possibility of providing strong privacy guarantees on the proposed algorithms through cryptographic techniques (Léauté and Faltings, 2013; Grinshpoun and Tassa, 2016; Tassa et al., 2017; Grinshpoun et al., 2019). Then, we will probe into combing GNLE with other dynamic programming techniques (e.g., function filtering (Sánchez-Fibla et al., 2005; Brito and Meseguer, 2010)) to solve larger-scale ADCOPs.

References

Barabási AL, Albert R (1999) Emergence of scaling in random networks. Science 286(5439):509–512

Brito I, Meseguer P (2010) Improving DPOP with function filtering. In: AAMAS, vol 1435, pp 141–148

Brito I, Meisels A, Meseguer P, Zivan R (2009) Distributed constraint satisfaction with partially known constraints. Constraints 14(2):199–234

Burke DA, Brown KN, Dogru M, Lowe B (2007) Supply chain coordination through distributed constraint optimization. In: AAMAS Workshop on Distributed Constraint Reasoning

Chen D, Deng Y, Chen Z, He Z, Zhang W (2020a) A hybrid tree-based algorithm to solve asymmetric distributed constraint optimization problems. Autonomous Agents and Multi-Agent Systems 34(2):1–42

Chen D, Deng Y, Chen Z, Zhang W, He Z (2020b) HS-CAI: A hybrid DCOP algorithm via combining search with context-based inference. In: AAAI, pp 7087–7094

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

Chen Z, Liu L, He J, Yu Z (2020c) A genetic algorithm based framework for local search algorithms for distributed constraint optimization problems. Autonomous Agents and Multi-Agent Systems 34(2):41

Chen Z, Zhang W, Deng Y, Chen D, Li Q (2020d) RMB-DPOP: Refining MB-DPOP by reducing redundant inferences. In: AAMAS, pp 249–257

Cohen L, Galiki R, Zivan R (2020) Governing convergence of max-sum on DCOPs through damping and splitting. Artificial Intelligence 279:103212

Dechter R, Rish I (2003) Mini-buckets: A general scheme for bounded inference. Journal of the ACM (JACM) 50(2):107–153

Dechter R, et al. (2003) Constraint processing. Morgan Kaufmann

Deng Y, Chen Z, Chen D, Zhang W, Jiang X (2019) AsymDPOP: Complete inference for asymmetric distributed constraint optimization problems. In: IJCAI, pp 223–230

Duan P, Zhang C, Mao G, Zhang B (2018) Applying distributed constraint optimization approach to the user association problem in heterogeneous networks. IEEE transactions on cybernetics 48(6):1696–1707

Farinelli A, Rogers A, Petcu A, Jennings NR (2008) Decentralised coordination of low-power embedded devices using the max-sum algorithm. In: AAMAS, pp 639–646

Fioretto F, Yeoh W, Pontelli E, Ma Y, Ranade SJ (2017) A distributed constraint optimization (DCOP) approach to the economic dispatch with demand response. In: AAMAS, pp 999–1007

Fioretto F, Pontelli E, Yeoh W (2018) Distributed constraint optimization problems and applications: A survey. Journal of Artificial Intelligence Research 61:623–698

Freuder EC, Quinn MJ (1985) Taking advantage of stable sets of variables in constraint satisfaction problems. In: IJCAI, pp 1076–1078

Gershman A, Zivan R, Grinshpoun T, Grubshtein A, Meisels A (2008) Measuring distributed constraint optimization algorithms. In: AAMAS Workshop on Distributed Constraint Reasoning

Gershman A, Meisels A, Zivan R (2009) Asynchronous forward bounding for distributed COPs. Journal of Artificial Intelligence Research 34:61–88

Grinshpoun T, Tassa T (2016) P-SyncBB: A privacy preserving branch and bound DCOP algorithm. Journal of Artificial Intelligence Research 57:621–660

Grinshpoun T, Grubshtein A, Zivan R, Netzer A, Meisels A (2013) Asymmetric distributed constraint optimization problems. Journal of Artificial Intelligence Research 47:613–647

Grinshpoun T, Tassa T, Levit V, Zivan R (2019) Privacy preserving region optimal algorithms for symmetric and asymmetric DCOPs. Artificial Intelligence 266:27–50

Gutierrez P, Meseguer P (2010) Saving messages in ADOPT-based algorithms. In: AAMAS Workshop on Distributed Constraint Reasoning, pp 53–64

Gutierrez P, Lee JH, Lei KM, Mak TW, Meseguer P (2013) Maintaining soft arc consistencies in BnB-ADOPT$^+$ during search. In: CP, pp 365–380

Hirayama K, Yokoo M (1997) Distributed partial constraint satisfaction problem. In: CP, pp 222–236

Hirayama K, Yokoo M (2005) The distributed breakout algorithms. Artificial Intelligence 161(1-2):89–115

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

Hoang KD, Fioretto F, Yeoh W, Pontelli E, Zivan R (2018) A large neighboring search schema for multi-agent optimization. In: CP, pp 688–706

Kask K, Dechter R, Larrosa J, Dechter A (2005) Unifying tree decompositions for reasoning in graphical models. Artificial Intelligence 166(1-2):165–193

Kschischang FR, Frey BJ, Loeliger HA (2001) Factor graphs and the sum-product algorithm. IEEE Transactions on information theory 47(2):498–519

Léauté T, Faltings B (2013) Protecting privacy through distributed computation in multi-agent decision making. Journal of Artificial Intelligence Research 47:649–695

Leite AR, Enembreck F (2019) Using collective behavior of coupled oscillators for solving DCOP. Journal of Artificial Intelligence Research 64:987–1023

Litov O, Meisels A (2017) Forward bounding on pseudo-trees for DCOPs and ADCOPs. Artificial Intelligence 252:83–99

Maheswaran RT, Pearce JP, Tambe M (2004a) Distributed algorithms for DCOP: A graphical-game-based approach. In: ISCA PDCS, pp 432–439

Maheswaran RT, Tambe M, Bowring E, Pearce JP, Varakantham P (2004b) Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In: AAMAS, pp 310–317

Modi PJ, Shen WM, Tambe M, Yokoo M (2005) ADOPT: Asynchronous distributed constraint optimization with quality guarantees. Artificial Intelligence 161(1-2):149–180

Monteiro TL, Pujolle G, Pellenz ME, Penna MC, Enembreck F, Souza RD (2012) A multi-agent approach to optimal channel assignment in WLANs. In: WCNC, pp 2637–2642

Netzer A, Grubshtein A, Meisels A (2012) Concurrent forward bounding for distributed constraint optimization problems. Artificial Intelligence 193:186–216

Nguyen DT, Yeoh W, Lau HC, Zivan R (2019) Distributed Gibbs: A linear-space sampling-based DCOP algorithm. Journal of Artificial Intelligence Research 64:705–748

Okamoto S, Zivan R, Nahon A (2016) Distributed breakout: beyond satisfaction. In: IJCAI, pp 447–453

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

Petcu A, Faltings B (2005a) Approximations in distributed optimization. In: CP, pp 802–806

Petcu A, Faltings B (2005b) A scalable method for multiagent constraint optimization. In: IJCAI, pp 266–271

Petcu A, Faltings B (2006) ODPOP: An algorithm for open/distributed constraint optimization. In: AAAI, pp 703–708

Petcu A, Faltings B (2007) MB-DPOP: A new memory-bounded algorithm for distributed optimization. In: IJCAI, pp 1452–1457

Ramchurn SD, Vytelingum P, Rogers A, Jennings N (2011) Agent-based control for decentralised demand side management in the smart grid. In: AAMAS, pp 5–12

Rogers A, Farinelli A, Stranders R, Jennings NR (2011) Bounded approximate decentralised coordination via the max-sum algorithm. Artificial Intelligence 175(2):730–759

Sánchez-Fibla M, Larrosa J, Meseguer P (2005) Improving tree decomposition methods with function filtering. In: IJCAI, pp 1537–1538

Silaghi MC, Yokoo M (2006) Nogood based asynchronous distributed optimization (ADOPT-ng). In: AAMAS, pp 1389–1396

Silaghi MC, Yokoo M (2009) ADOPT-ing: Unifying asynchronous distributed optimization with asynchronous backtracking. Autonomous Agents and Multi-Agent Systems 19(2):89–123

Sultanik EA, Modi PJ, Regli WC (2007) On modeling multiagent task scheduling as a distributed constraint optimization problem. In: IJCAI, pp 1531–1536

Tassa T, Grinshpoun T, Zivan R (2017) Privacy preserving implementation of the Max-Sum algorithm and its variants. Journal of Artificial Intelligence Research 59:311–349

Vinyals M, Rodriguez-Aguilar JA, 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

Yeoh W, Yokoo M (2012) Distributed problem solving. AI Magazine 33(3):53

Yeoh W, Felner A, Koenig S (2010) BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. Journal of Artificial Intelligence Research 38:85–133

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

Zivan R, Meisels A (2006) Message delay and DisCSP search algorithms. Annals of Mathematics and Artificial Intelligence 46(4):415–439

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

Zivan R, Lev O, Galiki R (2020a) Beyond trees: Analysis and convergence of belief propagation in graphs with multiple cycles. In: AAAI, pp 7333–7340

Zivan R, Parash T, Cohen-Lavi L, Naveh Y (2020b) Applying Max-sum to asymmetric distributed constraint optimization problems. Autonomous Agents and Multi-Agent Systems 34(1):1–29

Cycle	x_i	$x_c \in C(x_i)$	B_i^c	EV_i^c	$u_{i\to p}^c$	$AP(x_i)$	$\bigotimes_{x_j \in AP(x_i)} f_{ij}$	$u_{i\to p}$
1	x_3	–	–	–	–	$\{x_2\}$	f_{32}	$u_{3\to 2} = f_{32}$
	x_4	–	–	–	–	$\{x_1, x_2\}$	$f_{41} \otimes f_{42}$	$u_{4\to 2} = f_{41} \otimes f_{42}$
2	x_2	x_3	$\{x_3\}$	$\{x_3\}$	$\min_{x_3}(f_{32} \otimes f_{23})$	$\{x_1\}$	f_{21}	$u_{2\to 1} = (\min_{x_3}(f_{32} \otimes f_{23}))$
		x_4	$\{x_4\}$	$\emptyset$	$f_{41} \otimes f_{42} \otimes f_{24}$			$\otimes(f_{41} \otimes f_{42} \otimes f_{24}) \otimes f_{21}$
3	x_1	x_2	$\{x_2, x_4\}$	$\{x_2, x_4\}$	$\min_{\{x_2,x_4\}}((f_{12} \otimes f_{14}) \otimes (\min_{x_3}(f_{32} \otimes f_{23}) \otimes f_{41} \otimes f_{42} \otimes f_{24} \otimes f_{21})$	–	–	–

Fig. 14: The utility tables propagated in the utility propagation phase of AsymDPOP

A Appendix

In this subsection, we will take the variable x_2 in Fig. 3 as an example to show how to compute the utility table propagated by a variable x_i (i.e., $u_{i\to p}$) in AsymDPOP, and the utility table set propagated by x_i (i.e., $U_{i\to p}$) in AsymDPOP-TSPS when $k_p = 3$. We do not give the full trace of the execution of these algorithms since the other variables do a similar computation like x_2. The utility tables propagated in the utility propagation phase of AsymDPOP and the utility table set propagated in the utility set propagation phase of AsymDPOP-TSPS are shown in Fig. 14 and Fig. 15, respectively.

A.1 An example for AsymDPOP

When receiving the utility tables (i.e., $u_{3\to 2} = f_{32}$ and $u_{4\to 2} = f_{41} \otimes f_{42}$) from x_3 and x_4, x_2 computes the utility table $u_{2\to 1}$ (according to Eq. (6)) which is the join of the result of handling all the received utility tables (i.e., $u_{2\to 1}^3$ and $u_{2\to 1}^4$) and its local utility table (i.e., $\otimes_{x_j \in AP(x_2)} f_{2j}$). That is,

$$u_{2\to 1} = u_{2\to 1}^3 \otimes u_{2\to 1}^4 \otimes \left(\bigotimes_{x_j \in AP(x_2)} f_{2j} \right)$$

where $u_{2\to 1}^3$ and $u_{2\to 1}^4$ are calculated by Eq. (7). Specifically, for computing $u_{2\to 1}^3$, x_2 first joins the received utility table $u_{3\to 2} = f_{32}$ with its local utility table $\otimes_{x_j \in B_2^3} f_{2j} = f_{23}$, and then eliminates all the variables in $EV_2^3 = \{x_3\}$ from the joint utility table. That is,

$$\begin{aligned} u_{2\to 1}^3 &= \min_{EV_2^3} \left(\left(\bigotimes_{x_j \in B_2^3} f_{2j} \right) \otimes u_{3\to 2} \right) \\ &= \min_{x_3} \left(\left(\bigotimes_{x_j \in \{x_3\}} f_{2j} \right) \otimes f_{32} \right) \\ &= \min_{x_3} (f_{32} \otimes f_{23}) \end{aligned}$$

Similarly, we have $u_{2\to 1}^4 = f_{41} \otimes f_{42} \otimes f_{24}$, and thus $u_{2\to 1} = (\min_{x_3}(f_{32} \otimes f_{23})) \otimes (f_{41} \otimes f_{42} \otimes f_{24}) \otimes f_{21}$ since $AP(x_2) = \{x_1\}$.

Cycle	x_i	$x_c \in C(x_i)$	B_i^c	EV_i^c	$U_{i\to p}^c$	$AP(x_i)$	$U_{i\to p}^i$	$U_{i\to p}$
1	x_3	–	–	–	–	$\{x_2\}$	$\{f_{32}\}$	$U_{3\to 2} = \{f_{32}\}$
	x_4	–	–	–	–	$\{x_1, x_2\}$	$\{f_{41} \otimes f_{42}\}$	$U_{4\to 2} = \{f_{41} \otimes f_{42}\}$
2	x_2	x_3	$\{x_3\}$	$\{x_3\}$	$\{\min_{x_3}(f_{32} \otimes f_{23})\}$	$\{x_1\}$	$\{f_{21}\}$	$U_{2\to 1} = \{\min_{x_3}(f_{32} \otimes f_{23}),$
		x_4	$\{x_4\}$	$\emptyset$	$\{f_{41} \otimes f_{42} \otimes f_{24}\}$			$f_{41} \otimes f_{42} \otimes f_{24}, f_{21}\}$
3	x_1	x_2	$\{x_2, x_4\}$	$\{x_2, x_4\}$	$\{\min_{\{x_2,x_4\}}((f_{12} \otimes f_{14}) \otimes (\min_{x_3}(f_{32} \otimes f_{23}) \otimes f_{41} \otimes f_{42} \otimes f_{24} \otimes f_{21})\}$	–	–	–

Fig. 15: The utility table sets propagated in the utility set propagation phase of AsymDPOP-TSPS

A.2 An example for AsymDPOP-TSPS

After the utility table sets (i.e., $U_{3\to 2} = \{f_{32}\}$ and $U_{4\to 2} = \{f_{41} \otimes f_{42}\}$) from its child x_3 and x_4 have arrived, x_2 calculates the utility table set $U_{2\to 1}$ (according to Eq. (10)) which consists of the result of handling all the received utility table sets (i.e., $U_{2\to 1}^3$ and $U_{2\to 1}^4$) and its local utility table set (i.e., $U_{2\to 1}^2$). That is,

$$U_{2\to 1} = U_{2\to 1}^3 \cup U_{2\to 1}^4 \cup U_{2\to 1}^2$$

where $U_{2\to 1}^3$ and $U_{2\to 1}^4$ are calculated by Eq. (12). Concretely, for obtaining $U_{2\to 1}^3$, x_2 first joins the received utility set $U_{3\to 2} = \{f_{32}\}$ with its constraint functions related to B_2^3 (i.e., $\cup_{x_j \in B_2^3}\{f_{2j}\} = \{f_{23}\}$), and then eliminates $EV_2^3 = \{x_3\}$ from the set joint utility tables related to EV_2^3. That is,

$$\begin{aligned} U_{2\to 1}^3 &= \mathbf{Elim}\left(\mathbf{JoinSet}\left(\bigcup_{x_j \in B_2^3}\{f_{2j}\}, U_{3\to 2}\right), EV_2^3\right) \\ &= \mathbf{Elim}\left(\mathbf{JoinSet}\left(\bigcup_{x_j \in \{x_3\}}\{f_{2j}\}, \{f_{32}\}\right), \{x_3\}\right) \\ &= \mathbf{Elim}\left(\{f_{32} \otimes f_{23}\}, \{x_3\}\right) \\ &= \{\min_{x_3}(f_{32} \otimes f_{23})\} \end{aligned}$$

Similarly, we have $U_{2\to 1}^4 = \{f_{41} \otimes f_{42} \otimes f_{24}\}$. And $U_{2\to 1}^2$ is obtained by Eq. (11). In Eq. (11), x_2 first groups its constraint functions related to $AP(x_i)$ (i.e., $\cup_{x_j \in AP(x_2)}\{f_{2j}\} = \{f_{21}\}$) into a function set as the dimension of $\{f_{21}\}$ is no greater than $k_p = 3$. Afterward, x_2 obtains a 2-ary local utility table f_{21} by joining the functions in $\{f_{21}\}$. That is,

$$\begin{aligned} U_{2\to 1}^2 &= \mathbf{PartF}\left(\bigcup_{x_j \in AP(x_2)}\{f_{2j}\}, k_p\right) \\ &= \mathbf{PartF}\left(\bigcup_{x_j \in \{x_1\}}\{f_{2j}\}, 3\right) \\ &= \{f_{21}\} \end{aligned}$$

Thus, we have $U_{2\to 1} = \{\min_{x_3}(f_{32} \otimes f_{23}), f_{41} \otimes f_{42} \otimes f_{24}, f_{21}\}$.