AsymDPOP: Complete Inference for Asymmetric Distributed Constraint Optimization Problems

Yanchen Deng , **Ziyu Chen**∗ , **Dingding Chen** , **Wenxin Zhang** and **Xingqiong Jiang**

College of Computer Science, Chongqing University

dyc941126@126.com, {chenziyu,dingding}@cqu.edu.cn, wenxinzhang18@163.com, jxq@cqu.edu.cn

Abstract

Asymmetric distributed constraint optimization problems (ADCOPs) are an emerging model for co-ordinating agents with personal preferences. However, the existing inference-based complete algorithms which use local eliminations cannot be applied to ADCOPs, as the parent agents are required to transfer their private functions to their children. Rather than disclosing private functions explicitly to facilitate local eliminations, we solve the problem by enforcing delayed eliminations and propose AsymDPOP, the first inference-based complete algorithm for ADCOPs. 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 to reduce the computation efforts by sequential optimizations instead of joint optimizations. The empirical evaluation indicates that AsymDPOP significantly outperforms the state-of-the-art, as well as the vanilla DPOP with PEAV formulation.

1 Introduction

Distributed constraint optimization problems (DCOPs) [Modi *et al.*, 2005; Fioretto *et al.*, 2018] are a fundamental framework in multi-agent systems where agents coordinate their decisions to optimize a global objective. DCOPs have been adopted to model many real world problems including radio frequency allocation [Monteiro *et al.*, 2012], smart grid [Fioretto *et al.*, 2017] and distributed scheduling [Maheswaran *et al.*, 2004b; Li *et al.*, 2016].

Most of complete algorithms for DCOPs employ either distributed search [Hirayama and Yokoo, 1997; Modi *et al.*, 2005; Gershman *et al.*, 2009; Yeoh *et al.*, 2010] or inference [Petcu and Faltings, 2005b; Vinyals *et al.*, 2011] to optimally solve DCOPs. However, since DCOPs are NP-hard, complete algorithms cannot scale up due to exponential overheads. Thus, incomplete algorithms [Maheswaran *et al.*, 2004a; Zhang *et al.*, 2005; Okamoto *et al.*, 2016; Rogers *et al.*, 2011; Zivan and Peled, 2012; Chen *et al.*, 2018; Ottens *et al.*, 2017;

∗Corresponding Author

Fioretto *et al.*, 2016] are proposed to trade optimality for smaller computational efforts.

Unfortunately, DCOPs fail to capture the ubiquitous asymmetric structure [Burke *et al.*, 2007; Maheswaran *et al.*, 2004b; Ramchurn *et al.*, 2011] since each constrained agent shares the same payoffs. PEAV [Maheswaran *et al.*, 2004b] attempts to capture the asymmetric costs by introducing mirror variables and the consistency is 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, AD-COPs [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 challenging in asymmetric settings due to a privacy concern. SyncABB and ATWB [Grinshpoun *et al.*, 2013] are asymmetric adaption of SyncBB [Hirayama and Yokoo, 1997] and AFB [Gershman *et al.*, 2009], using an one-phase strategy to aggregate the individual costs. That is, the algorithms systematically check each side of a constraint before reaching a complete assignment. Besides, 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 one-phase check. Recently, PT-ISABB [Deng *et al.*, 2019] was proposed to improve the tree-based search by implementing a non-local elimination version of ADPOP [Petcu and Faltings, 2005a] to provide much tighter lower bounds. However, since it relies on an exhaustive search to guarantee the optimality, the algorithm still suffers from exponential communication overheads. On the other hand, although complete inference algorithms (e.g., DPOP [Petcu and Faltings, 2005b]) only require a linear number of messages to solve DCOPs, they cannot be directly applied to ADCOPs without PEAV due to their requirement for complete knowledge of each constraint to facilitate variable elimination. Accordingly, the parents have to transfer their private cost functions to their children, which leaks at least a half of privacy.

In this paper, we adapt DPOP for solving 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 AsymDPOP, the first complete inference-based algorithm to solve ADCOPs, by generalizing non-local elimination [Deng *et al.*, 2019]. That is, instead of eliminating variables at their parents, we postpone the eliminations until their highest neighbors in the pseudo tree. In other words, an agent in our algorithm may be responsible for eliminating several variables.

- We theoretically analyze the complexity of our algorithm where the space complexity of an agent is not only exponential in the number of its separators but also the number of its non-eliminated descendants.

- We scale up our algorithm by introducing a table-set propagation scheme to reduce the memory consumption and a mini-batch scheme to reduce the number of operations when performing eliminations. Our empirical evaluation indicates that our proposed algorithm significantly outperforms the state-of-the-art, as well as the vanilla DPOP with PEAV formulation.

2 Backgrounds

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

2.1 Distributed Constraint Optimization Problems

A distributed constraint optimization problem [Modi *et al.*, 2005] is defined by a tuple $\langle A, X, D, F \rangle$ where

- $A = \{a_1, \ldots, a_n\}$ is the set of agents
- $X = \{x_1, \ldots, x_m\}$ is the set of variables
- $D = \{D_1, \ldots, D_m\}$ is the set of domains. Variable x_i takes values from D_i
- $F = \{f_1, \ldots, f_q\}$ is the set of constraint functions. Each function $f_i : D_{i1} \times \cdots \times D_{ik} \to \mathbb{R}_{\geq 0}$ specifies the cost assigned to each combination of $x_{i1}, \ldots, x_{ik}$.

For the sake of simplicity, we assume that each agent controls a variable (and thus the term "agent" and "variable" could be used interchangeably) and all constraint functions are binary (i.e., $f_{ij} : D_i \times D_j \to \mathbb{R}_{\geq 0}$). A solution to a DCOP is an assignment to all the variables such that the total cost is minimized. That is,

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

2.2 Asymmetric Distributed Constraint Optimization Problems

While DCOPs assume an equal payoff for each participant of each constraint, asymmetric distributed constraint optimization problems (ADCOPs) [Grinshpoun *et al.*, 2013] explicitly define the exact payoff for each constrained agent. In other words, a constraint function $f_i : D_{i1} \times \cdots \times D_{ik} \to \mathbb{R}_{\geq 0}^k$ in an ADCOP specifies a cost vector for each possible combination
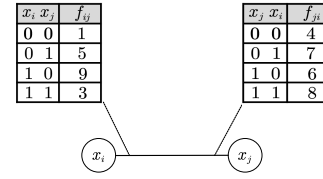

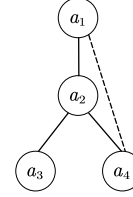Figure 1: An ADCOP with two variables and a constraint

Figure 2: A pseudo tree

of involved variables. And the goal is to find a solution which minimizes the aggregated cost. An ADCOP can be visualized by a constraint graph where the vertexes denote variables and the edges denote constraints. Fig. 1 presents an ADCOP with two variables and a constraint. Besides, for the constraint between x_i and x_j, we denote the private function for x_i and x_j as f_{ij} and f_{ji}, respectively.

2.3 Pseudo Tree

A pseudo tree [Freuder and Quinn, 1985] is an ordered arrangement to a constraint graph in which different branches are independent. A pseudo tree can be generated by a depth-first traverse to a constraint graph, categorizing constraints into tree edges and pseudo edges (i.e., non-tree edges). The neighbors of an agent a_i are therefore categorized into its parent $P(a_i)$, pseudo parents $PP(a_i)$, children $C(a_i)$ and pseudo children $PC(a_i)$ according to their positions in the pseudo tree and the types of edges they connect through. We also denote its parent and pseudo parents as $AP(a_i) = P(a_i) \cup PP(a_i)$, and its descendants as $Desc(a_i)$. Besides, we denote its separators, i.e., the set of ancestors which are constrained with a_i and its descendants, as $Sep(a_i)$ [Petcu and Faltings, 2006]. Finally, we denote a_i's interface descendants, the set of descendants which are constrained with $Sep(a_i)$, as $ID(a_i)$. Fig. 2 presents a pseudo tree in which the dotted edge is a pseudo edge.

2.4 DPOP and Non-local Elimination

DPOP [Petcu and Faltings, 2005b] is an inference-based complete algorithm for DCOPs based on bucket elimination [Dechter, 1999]. Given a pseudo tree, it performs a bottom-up utility propagation phase to eliminate variables and a value propagation phase to assign the optimal assignment for each variable. More specifically, in the utility propagation phase, an agent a_i eliminates its variables from the joint utility table by computing the optimal utility for each possible assignment to $Sep(a_i)$ after receiving the utility tables from its children, and sends the projected utility table to its parent. In the value propagation phase, a_i computes the optimal assignments for its variables by considering the assignments received from its

---

Algorithm 1: GNLE for a_i

  When `Initialization`:

1    $util_i \leftarrow \underset{a_j \in AP(a_i)}{\otimes} f_{ij}$

2   **if** a_i *is a leaf* **then**

3   send UTIL($util_i$) to $P(a_i)$

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

4    $util_i^c \leftarrow util_c$

5   **foreach** $a_j \in (PC(a_i) \cap Desc(a_c)) \cup \{a_c\}$ **do**

6        $util_i^c \leftarrow util_i^c \otimes f_{ij}$

7    $util_i \leftarrow util_i \otimes \underset{EV(a_i, a_c)}{\min} util_i^c$

8   **if** a_i *has received all UTIL from* $C(a_i)$ **then**

9   **if** a_i *is the root* **then**

10   start **Value Propagation** phase

11   **else**

12   send UTIL($util_i$) to $P(a_i)$

---

parent, and propagates the joint assignment to its children. Although DPOP only requires a linear number of messages to solve a DCOP, its memory consumption is exponential in the induced width. Thus, several tradeoffs including ADPOP [Petcu and Faltings, 2005a], MB-DPOP [Petcu and Faltings, 2007] and ODPOP [Petcu and Faltings, 2006] have been proposed to improve its scalability.

However, DPOP cannot be directly applied to asymmetric settings as it requires the total knowledge of each constraint to perform optimal elimination locally. PT-ISABB [Deng *et al.*, 2019] applies (A)DPOP into solving ADCOPs by performing variable elimination only to a subset of constraints to build look-up tables for lower bounds, and uses a tree-based search to guarantee the optimality. The algorithm further reinforces the bounds by a non-local elimination scheme. That is, instead of performing elimination locally, the elimination of a variable is postponed to its parent to include the private function enforced in the parent's side and increase the integrity of the utility table.

3 Asymmetric DPOP

The existing complete algorithms for ADCOPs use complete search to exhaust the search space, which makes them unsuitable for large scale applications. In fact, as shown in our experimental results, these search-based algorithms can only solve the problems with the agent number less than 20. Hence, in this section, we propose a complete, privacy-protecting, and scalable inference-based algorithm for ADCOPs built upon generalized non-local elimination, called AsymDPOP. An execution example can be found in the appendix (https://arxiv.org/abs/1905.11828).

3.1 Utility Propagation Phase

In DPOP, a variable could be eliminated locally without loss of completeness after receiving all the utility messages from its children since all functions that involve the variable have been aggregated. However, the conclusion does not hold for ADCOPs. Taking Fig. 2 as an example, x_4 cannot be eliminated locally since the private functions f_{14} and f_{24} are not given. Thus, the local elimination to x_4 w.r.t. f_{41} and f_{42} would lead to overestimate bias and offer no guarantee on the completeness. A naïve solution would be that x_1 and x_2

transfer their private functions to their children, which would lead to an unacceptable privacy loss.

Inspired by non-local elimination, we consider an alternative privacy-protecting approach to aggregate constraint functions. That is, instead of deferring to its parent, we postpone the elimination of a variable to its highest (pseudo) parent. In this way, all the functions involving the variables have been aggregated and the variable can be eliminated from the utility table optimally. Note that the utility table is a summation of the utility tables from children and local constraints. As a result, although the utility table which contains the variable's private functions is propagated to its ancestors without elimination, the ancestors can hardly infer the exact payoffs in these private functions. We refer the bottom-up utility propagation phase as generalized non-local elimination (GNLE).

Before diving into the details of GNLE, let's first consider the following definitions.

Definition 1 ($dims$). *The $dims(\cdot)$ is a function which returns the set of dimensions of a utility table.*

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

$$S_{[K]} = \{(k = v) \in S | k \in K\}$$

Definition 3 (Join [Vinyals *et al.*, 2011]). *Let U, U' be two utility tables and $D_U = \times_{x_i \in dims(U)} D_i$, $D_{U'} = \times_{x_i \in dims(U')} D_i$ be their joint domain spaces. $U \otimes U'$ is the join of U and U' over $D_{U \otimes U'} = \times_{x_i \in dims(U) \cup dims(U')} D_i$ such that*

$$(U \otimes U')(V) = U(V_{[dims(U)]}) + U'(V_{[dims(U')]}), \forall V \in D_{U \otimes U'}$$

Algorithm 1 presents the sketch of GNLE. The algorithm begins with leaf agents sending their utility tables to their parents via UTIL messages (line 2-3). When an agent a_i receives a UTIL message from a child a_c, it joins its private functions w.r.t. its (pseudo) children in branch a_c (line 5-6), and eliminates all the belonging variables whose highest (pseudo) parents are a_i from the utility table (line 7). Here, $EV(a_i, a_c)$ is given by

$$EV(a_i, a_c) = PC(a_i) \cap Desc(a_c) \cup \{a_c\} \backslash ID(a_i)$$

Then a_i joins the eliminated utility table with its running utility table $util_i$. It is worth mentioning that computing the set of elimination variables $EV(a_i, a_c)$ does not require agents 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 parent and pseudo parents. When its (pseudo) parent receives the UTIL message containing it, the counter decreases. And the variable is added to the set of elimination variables as soon as its counter equals zero.

After receiving all the UTIL messages from its children, a_i propagates the utility table $util_i$ to its parent if it is not the root (line 12). Otherwise, the value propagation phase starts (line 10).

3.2 Value Propagation Phase

In contrary to the one in vanilla DPOP which determines the optimal assignment locally for each variable, the value assignment phase in AsymDPOP should be specialized to accommodate the non-local elimination. Specifically, since a

Algorithm 2: Value Propagation for a_i

```
    When Initialization:
13     if a_i is the root then
14         v_i* ← argmin util_i
                    x_i
15         PropagateValue({(x_i = v_i*)})
    When received VALUE(Assign) from P(a_i):
16     PropagateValue(Assign)
    Function PropagateValue(Assign):
17     foreach a_c ∈ C(a_i) do
18         Assign_i^c ← Assign_[Sep(a_c)∪{a_c}∪ID(a_c)]
19         if EV(a_i, a_c) ≠ ∅ then
20             V* ←    argmin    util_i^c(Assign_[dims(util_i^c)])
                    EV(a_i,a_c)
21             Assign_i^c ← Assign_i^c ∪ {(x_j = V*_[x_j])|∀x_j ∈
                   EV(a_i, a_c)}
22         send VALUE(Assign_i^c) to a_c
```
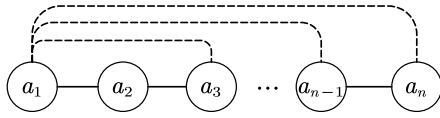

Figure 3: A chain-like pseudo tree

variable is eliminated at its highest (pseudo) parent, the parent is responsible for selecting the optimal assignment for that variable. Thus, the value messages in our algorithm would contain not only the assignments for ancestors, but also assignments for descendants. Algorithm 2 presents the sketch of value propagation phase.

The phase is initiated by the root agent selecting the optimal assignment (line 14). Given the determined assignments either from its parent (line 16) or computed locally (line 15), agent a_i selects the optimal assignments for the eliminated variables in each branch $a_c \in C(a_i)$ by a joint optimization over them (line 17-20), and propagates the assignments together with the determined assignments to a_c (line 21-22). The algorithm terminates when each leaf agent receives a VALUE message.

3.3 Complexity Analysis

Theorem 1. *The size of a UTIL message produced by an agent is exponential in the number of its separator and its interface descendants.*

Proof. We prove the theorem by showing a UTIL message produced by agent a_i contains the dimensions of $Sep(a_i)$ and $ID(a_i) \cup \{a_i\}$. The UTIL message must contain the dimensions of $ID(a_i) \cup \{a_i\}$ since a_i is not the highest (pseudo) parent of $a_j, \forall a_j \in ID(a_i) \cup \{a_i\}$. On the other hand, according to the definition to $ID(a_i)$, the UTIL message cannot contain the dimensions of $Desc(a_i) \backslash ID(a_i)$ since for each $a_j \in Desc(a_i) \backslash ID(a_i)$ it must exist an agent $a_k \in Desc(a_i) \cup \{a_i\}$ such that a_k is the highest (pseudo) parent of a_j and thus the variable is eliminated at a_k (line 7). Finally, the UTIL message contains $Sep(a_i)$ according to [Petcu and Faltings, 2005b] (Theorem 1). Thus, the size of the UTIL message is exponential in $|Sep(a_i)| + |ID(a_i)| + 1$ and the theorem is concluded.

$\square$

4 Tradeoffs

As shown in Section 3.3, AsymDPOP suffers serious scalability problems in both memory and computation. In this section, we propose two tradeoffs which make AsymDPOP a practical algorithm.

4.1 Table-Set Propagation Scheme: A Tradeoff between Memory and Privacy

The utility propagation phase of AsymDPOP could be problematic due to the unacceptable memory consumption when the pseudo tree is poor. Consider the pseudo tree shown in Fig. 3. Since every agent is constrained with the root agent, according to the GNLE all the variables can only be eliminated at the root agent, which incurs a memory consumption of $O(d^n)$ due to the join operation in each agent. Here, $d = \max_i |D_i|$. Besides, a large utility table would also incur unacceptable computation overheads due to the join operations and the elimination operation (line 5-7).

We notice that utility tables are divisible before performing eliminations. Thus, instead of propagating a joint and high-dimension utility table to a parent agent, we propagate a set of small utility tables. In other words, we wish to reduce the unnecessary join operations (i.e., line 1 and line 7 in the case of $EV(a_i, a_c) = \emptyset$) which could cause a dimension increase during the utility propagation phase. On the other side, completely discarding join operations would incur privacy leakages. For example, if a_n chooses to propagate both $f_{n,n-1}$ and f_{n1} without performing the join operation to a_{n-1}, then a_{n-1} would know the private functions of a_n directly. Thus, we propose to compromise the memory consumption and the privacy by a parameter k_p controlling the the maximal number of dimensions of each local utility table. We refer the tradeoff as a table-set propagation scheme (TSPS).

Specifically, when an agent sends a UTIL message to its parent, it first joins its private functions w.r.t. its parents with the received utility tables whose dimensions contain the dimensions of these private functions. Notice that the first step does not incur a dimension increase, and can reduce the number of utility tables. Finally, it propagates the set of utility tables to its parent.

Consider the pseudo tree shown in Fig. 3 again. Assume that $k_p = 3$ and then agent a_n would propagate the utility set $util_n = \{f_{n,n-1} \otimes f_{n1}\}$ to a_{n-1}. Since there is no elimination in a_{n-1}, it is unnecessary to perform the join operation in line 7. Thus, a_{n-1} would propagate the utility set $util_{n-1} = \{f_{n-1,n-2} \otimes f_{n-1,1}, util_{n-1}^n\}$ to a_{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 an agent a_i in TSPS is no greater than*

$$d^{\max(\min(|AP(a_i)|, k_p), \max_{a_c \in C(a_i)}(|ID(a_i) \cap (ID(a_c) \cup \{a_c\})| + |Sep(a_c)|))}$$

Proof. According to Theorem 1, the dimension of each utility table from child $a_c \in C(a_i)$ is a subset of $Sep(a_c) \cup ID(a_c) \cup \{a_c\}$. Since TSPS omits the join operation in line 7, the maximal size of received utility tables of a_i is

$$d^{\max_{a_c \in C(a_i)}(|ID(a_i) \cap (ID(a_c) \cup \{a_c\})| + |Sep(a_c)|)}$$
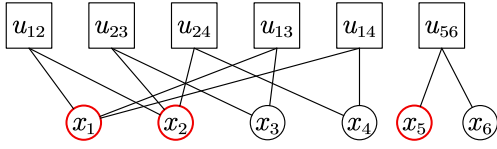
Figure 4: A set of utility tables

Since the local utility is partitioned into utility tables according to k_p, the maximal size of local utility tables of a_i is

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

Thus the theorem holds. □

4.2 Mini-batch Elimination Scheme: A Tradeoff between Time and Space

TSPS could factorize a big utility table to a set of smaller utility tables, which allows us to reduce the computational efforts when performing eliminations by a decrease-and-conquer strategy. Taking Fig. 3 as an example, to perform the elimination, a_1 in GNLE has no choice but to optimize a big utility table over variables $x_2, \ldots x_n$ (line 7), which requires $O(d^n)$ operations. Instead, combining with TSPS ($k_p = 2$) we could exploit the the structure of each small utility table by arranging the min operators among them to reduce computational complexity. That is, instead of performing

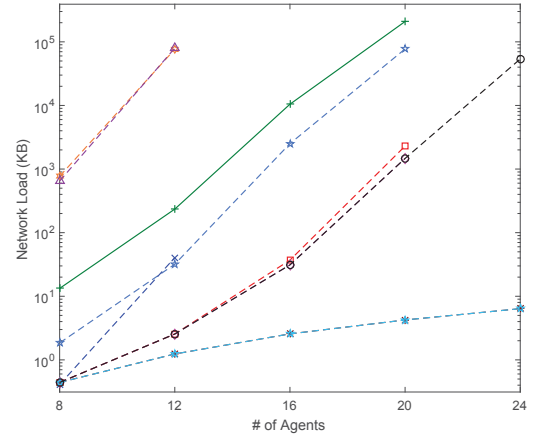$$\min_{x_2, \ldots x_n} u_{1,2,\ldots,n}$$

we perform

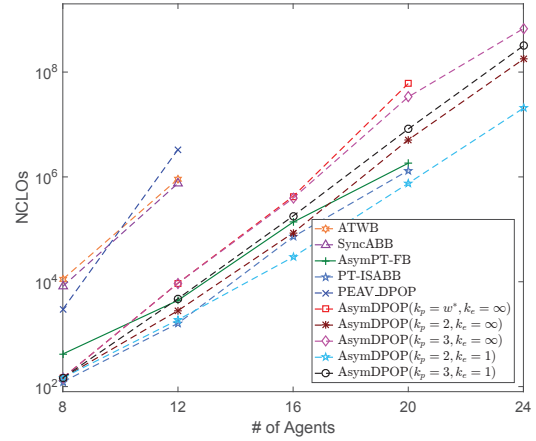$$\min_{x_2}(f_{12} + f_{21} + \cdots + \min_{x_n}(f_{1n} + f_{n1} + f_{n,n-1} + f_{n-1,n}) \cdots)$$

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 traverses.

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 is not easy to find the optimal sequence of eliminations in practice. Consider the utility tables shown as a factor graph in Fig. 4 where square nodes represent utility tables and 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 optimize both x_1 and x_2, the maximal number of dimensions are 2.

We thus overcome the problem by introducing a parameter k_e which specifies the minimal number of variables optimized in a min operator (i.e., the size of a batch), and refer the tradeoff as a mini-batch elimination scheme (MBES). Specifically, when performing an elimination operation, we first divide elimination variables into several groups whose variables share at least a common utility table. For each variable group, we divide the variables into batches whose sizes are at least k_e if it is possible. For each batch, we perform
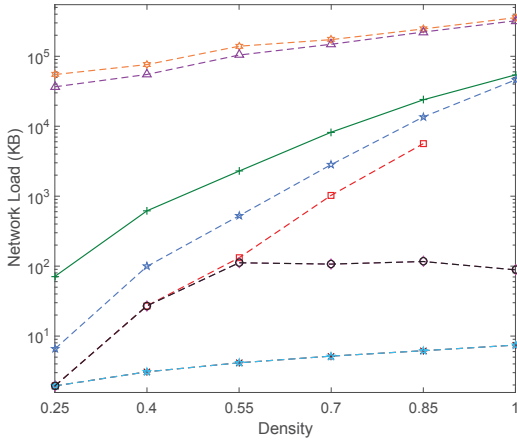

(a) network load

(b) NCLOs

Figure 5: Performance comparison under different agent numbers

optimization to the functions that are related to the batch and replace these functions with the results. The process terminates when all the variable groups are exhausted.
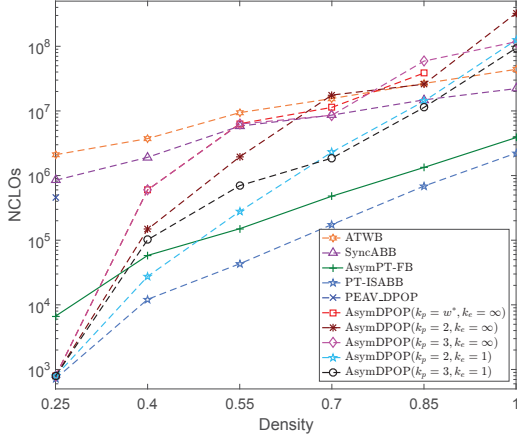
Note that dividing variables into disjoint variable groups in the first step is crucial since optimizing independent variables jointly is equivalent to optimizing them individually. Taking Fig. 4 for example, if we set $k_e = 2$ and let x_2, x_5 be a batch, a 4-ary utility table over x_1, x_3, x_4 and x_6 still appear even if x_2 and x_5 is jointly optimized.

5 Experimental Results

We empirically evaluate the performances of AsymDPOP and state-of-the-art complete algorithms for ADCOPs including PT-ISABB, AsymPT-FB, SyncABB, ATWB and the vanilla DPOP with PEAV formulation (PEAV_DPOP) in terms of the number of basic operations, network load and privacy leakage. For inference-based algorithms, we consider the maximal number of dimensions during the utility propagation phase as an additional metric. Specifically, we use NCLOs [Netzer *et al.*, 2012] to measure the hardware-independent runtime in which the logic operations in inference-based algorithms are accesses to the utility tables while the ones in
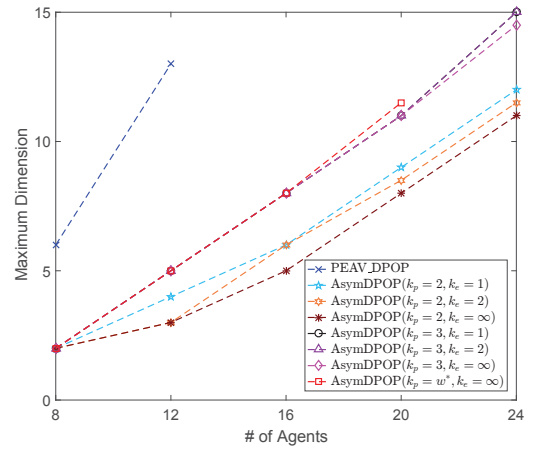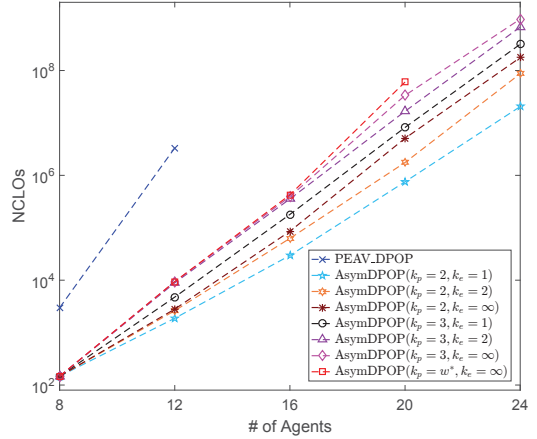
(a) network load

(b) NCLOs

Figure 6: Performance comparison under different densities

(a) maximum dimensions

(b) NCLOs

Figure 7: Performance comparison of different batch sizes

search-based algorithms are constraint checks. For the network load, we measure the size of information during an execution. Finally, we use entropy [Brito *et al.*, 2009] to quantify the privacy loss [Litov and Meisels, 2017; Grinshpoun *et al.*, 2013]. For each experiment, we generate 50 random instances and report the medians as the results.

In our first experiment, we consider the ADCOPs with the domain size of 3, the density of 0.25 and the agent number varying from 8 to 24. Fig. 5 presents the experimental results under different agent numbers. It can be concluded that compared to the search-based solvers, our AsymDPOP algorithms exhibit great superiorities in terms of network load. That is due to the fact that search-based algorithms explicitly exhaust the search space by message-passing, which is quite expensive especially when the agent number is large. In contrast, our proposed algorithms incur few communication overheads since they follow an inference protocol and only require a linear number of messages. 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 the agent number less than 12. The phenomenon is due to the mirror variables introduced by PEAV formulation, which significantly increases the complexity. More specifi-

cally, a UTIL message in PEAV_DPOP contains the dimensions of mirror variables, which significantly increases the memory consumption.

Fig. 6 presents the performance comparison under different densities. Specifically, we consider the ADCOPs with the agent number of 8, the domain size of 8 and the density varying from 0.25 to 1. It can be concluded from Fig. 6(a) that AsymDPOP with TSPS (i.e., $k_p < w^*$ where w^* is the induced width) incurs significantly less communication overheads when solving dense problems, which demonstrates the merit of avoiding unnecessary join operations. Besides, it is interesting to find that compared to the one of AsymDPOP without TSPS (i.e., $k_p = w^*$), the network load of AsymDPOP($k_p < w^*, \cdot$) increases much slowly. That is due to the fact that as the density increases, eliminations are more likely to happen at the top of a pseudo tree. On the other hand, since unnecessary join operations are avoided in TSPS, eliminations are the major source of the dimension increase. As a result, agents propagate low dimension utility tables in most of the time. For NCLOs, search-based algorithms like AsymPT-FB and PT-ISABB outperform AsymDPOP algorithms when solving dense problems due to their effective pruning.
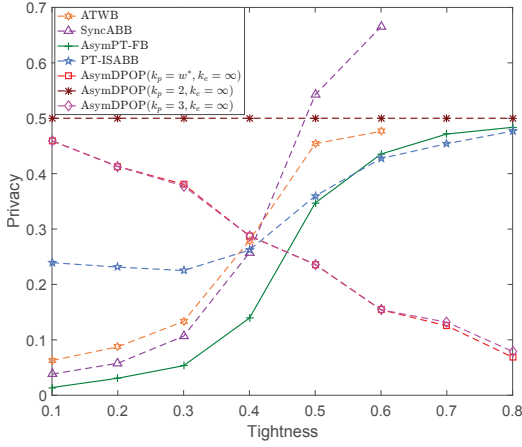
Figure 8: Privacy loss under different tightness

To demonstrate the effects of different batch sizes, we benchmark MBES with different k_e when combining with TSPS with different k_p on the configuration used in the first experiment. Specifically, we measure the maximal number of dimensions generated in the utility propagation phase, including the intermediate results of MBES. Fig. 7 presents the experimental results. It can be seen that AsymDPOP with a small batch size significantly reduces NCLOs but produces larger intermediate results, which indicates the necessity of the tradeoff. Besides, the performance of MBES significantly degenerates when combining with TSPS with a larger k_p. That is because the utility tables contain more dimensions in the scenario, and a utility table would be traversed more frequently when performing eliminations.

In the last experiment, we consider the privacy loss of different algorithms when solving asymmetric MaxDCSPs with different tightness. In particular, we consider the asymmetric MaxDCSPs with 10 agents, the domain size of 10, the density of 0.4 and the tightness varying from 0.1 to 0.8. Fig. 8 presents the results. It can be concluded from the figure that as the tightness grows, the search-based algorithms leak more privacy while the inference-based algorithms leaks less privacy. That is due to the fact that search-based algorithms rely on a direct disclosure mechanism to aggregate the private costs. Thus, search-based algorithms would leak more privacy when solving the problems with high tightness as they need to traverse more proportions of the search space. In contrast, inference-based algorithms accumulate utility through the pseudo tree, and an agent a_i could infer the private costs of its (pseudo) child a_c when the utility table involving both x_i and x_c is a binary table which is not a result of eliminations or contains zero entries. Thus, AsymDPOP($k_p = 2, \cdot$) leaks almost a half of privacy. On the other hand, since the number of prohibit combinations grows as the tightness increases, AsymDPOP($k_p \geq 3, \cdot$) incurs much lower privacy loss when solving high tightness problems.

6 Conclusion

In this paper we present AsymDPOP, the first complete inference algorithm for ADCOPs. The algorithm incorporates three ingredients: generalized non-local elimination which facilitates the aggregation of utility in an asymmetric environment, table-set propagation scheme which reduces the memory consumption and mini-batch elimination scheme which reduces the operations in the utility propagation phase. We theoretically show its complexity and our empirical evaluation demonstrates its superiorities over the state-of-the-art, as well as the vanilla DPOP with PEAV formulation.

References

[Brito *et al.*, 2009] Ismel Brito, Amnon Meisels, Pedro Meseguer, and Roie Zivan. Distributed constraint satisfaction with partially known constraints. *Constraints*, 14(2):199–234, 2009.

[Burke *et al.*, 2007] David A Burke, Kenneth N Brown, Mustafa Dogru, and Ben Lowe. Supply chain coordination through distributed constraint optimization. In *The 9th International Workshop on DCR*, 2007.

[Chen *et al.*, 2018] Ziyu Chen, Yanchen Deng, Tengfei Wu, and Zhongshi He. A class of iterative refined max-sum algorithms via non-consecutive value propagation strategies. *Autonomous Agents and Multi-Agent Systems*, 32(6):822–860, 2018.

[Dechter, 1999] Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.

[Deng *et al.*, 2019] Yanchen Deng, Ziyu Chen, Dingding Chen, Xingqiong Jiang, and Qiang Li. PT-ISABB: A hybrid tree-based complete algorithm to solve asymmetric distributed constraint optimization problems. In *Proceedings of 18th International Conference on Autonomous Agents and Multiagent Systems*, pages 1506–1514, 2019.

[Fioretto *et al.*, 2016] Ferdinando Fioretto, William Yeoh, and Enrico Pontelli. A dynamic programming-based MCMC framework for solving DCOPs with GPUs. In *Proceedings of 22nd International Conference on Principles and Practice of Constraint Programming*, pages 813–831, 2016.

[Fioretto *et al.*, 2017] Ferdinando Fioretto, William Yeoh, Enrico Pontelli, Ye Ma, and Satishkumar J Ranade. A distributed constraint optimization (DCOP) approach to the economic dispatch with demand response. In *Proceedings of the 16th International Conference on Autonomous Agents and MultiAgent Systems*, pages 999–1007, 2017.

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

[Freuder and Quinn, 1985] Eugene C. Freuder and Michael J. Quinn. Taking advantage of stable sets of variables in constraint satisfaction problems. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 1076–1078, 1985.

[Gershman *et al.*, 2009] Amir Gershman, Amnon Meisels, and Roie Zivan. Asynchronous forward bounding for distributed COPs. *Journal of Artificial Intelligence Research*, 34:61–88, 2009.

[Grinshpoun *et al.*, 2013] Tal Grinshpoun, Alon Grubshtein, Roie Zivan, Arnon Netzer, and Amnon Meisels. Asymmetric distributed constraint optimization problems. *Journal of Artificial Intelligence Research*, 47:613–647, 2013.

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

[Li *et al.*, 2016] Shijie Li, Rudy R. Negenborn, and Gabriël Lodewijks. Distributed constraint optimization for addressing vessel rotation planning problems. *Engineering Applications of Artificial Intelligence*, 48:159–172, 2016.

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

[Maheswaran *et al.*, 2004a] Rajiv T Maheswaran, Jonathan P Pearce, and Milind Tambe. Distributed algorithms for DCOP: A Graphical-Game-Based approach. In *Proceedings of ISCA PDCS*, pages 432–439, 2004.

[Maheswaran *et al.*, 2004b] Rajiv T. Maheswaran, Milind Tambe, Emma Bowring, Jonathan P. Pearce, and Pradeep Varakantham. Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *Proceedings of the 3rd International Conference on Autonomous Agents and Multiagent Systems*, pages 310–317, 2004.

[Modi *et al.*, 2005] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149–180, 2005.

[Monteiro *et al.*, 2012] Tânia L Monteiro, Guy Pujolle, Marcelo E Pellenz, Manoel C Penna, and Richard Demo Souza. A multi-agent approach to optimal channel assignment in WLANs. In *Wireless Communications and Networking Conference*, pages 2637–2642, 2012.

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

[Okamoto *et al.*, 2016] Steven Okamoto, Roie Zivan, and Aviv Nahon. Distributed breakout: Beyond satisfaction. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, pages 447–453, 2016.

[Ottens *et al.*, 2017] Brammert Ottens, Christos Dimitrakakis, and Boi Faltings. DUCT: An upper confidence bound approach to distributed constraint optimization problems. *ACM Transactions on Intelligent Systems and Technology*, 8(5):69, 2017.

[Petcu and Faltings, 2005a] Adrian Petcu and Boi Faltings. Approximations in distributed optimization. In *International conference on principles and practice of constraint programming*, pages 802–806, 2005.

[Petcu and Faltings, 2005b] Adrian Petcu and Boi Faltings. A scalable method for multiagent constraint optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 266–271, 2005.

[Petcu and Faltings, 2006] Adrian Petcu and Boi Faltings. ODPOP: An algorithm for open/distributed constraint optimization. In *Proceedings of the 21st national conference on Artificial intelligence*, pages 703–708, 2006.

[Petcu and Faltings, 2007] Adrian Petcu and Boi Faltings. MB-DPOP: A new memory-bounded algorithm for distributed optimization. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1452–1457, 2007.

[Ramchurn *et al.*, 2011] Sarvapali D Ramchurn, Perukrishnen Vytelingum, Alex Rogers, and Nick Jennings. Agent-based control for decentralised demand side management in the smart grid. In *The 10th International Conference on Autonomous Agents and Multiagent Systems*, pages 5–12, 2011.

[Rogers *et al.*, 2011] Alex Rogers, Alessandro Farinelli, Ruben Stranders, and Nicholas R Jennings. Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence*, 175(2):730–759, 2011.

[Vinyals *et al.*, 2011] Meritxell Vinyals, Juan A Rodriguez-Aguilar, and Jesús Cerquides. Constructing a unifying theory of dynamic programming DCOP algorithms via the generalized distributive law. *Autonomous Agents and Multi-Agent Systems*, 22(3):439–464, 2011.

[Yeoh *et al.*, 2010] William Yeoh, Ariel Felner, and Sven Koenig. BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. *Journal of Artificial Intelligence Research*, 38:85–133, 2010.

[Zhang *et al.*, 2005] Weixiong Zhang, Guandong Wang, Zhao Xing, and Lars Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, 161(1-2):55–87, 2005.

[Zivan and Peled, 2012] Roie Zivan and Hilla Peled. Max/min-sum distributed constraint optimization through value propagation on an alternating DAG. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, pages 265–272, 2012.