Speeding Up Incomplete GDL-based Algorithms for Multi-agent Optimization with Dense Local Utilities

Yanchen Deng and **Bo An**
School of Computer Science and Engineering, Nanyang Technological University, Singapore
{ycdeng,boan}@ntu.edu.sg

Abstract

Incomplete GDL-based algorithms including Max-sum and its variants are important methods for multi-agent optimization. However, they face a significant scalability challenge as the computational overhead grows exponentially with respect to the arity of each utility function. Generic Domain Pruning (GDP) technique reduces the computational effort by performing a one-shot pruning to filter out suboptimal entries. Unfortunately, GDP could perform poorly when dealing with dense local utilities and ties which widely exist in many domains. In this paper, we present several novel sorting-based acceleration algorithms by alleviating the effect of densely distributed local utilities. Specifically, instead of one-shot pruning in GDP, we propose to integrate both search and pruning to iteratively reduce the search space. Besides, we cope with the utility ties by organizing the search space of tied utilities into AND/OR trees to enable branch-and-bound. Finally, we propose a discretization mechanism to offer a tradeoff between the reconstruction overhead and the pruning efficiency. We demonstrate the superiorities of our algorithms over the state-of-the-art from both theoretical and experimental perspectives.

1 Introduction

Distributed Constraint Optimization Problems (DCOPs) [Modi *et al.*, 2005; Fioretto *et al.*, 2018] are a fundamental model for multi-agent optimization and coordination, in which agents cooperatively find assignments to optimize a global objective. DCOPs have been successfully applied to model many real-world problems where information and controls are inherently distributed among agents such as distributed scheduling [Hirayama *et al.*, 2019; Li *et al.*, 2016], smart-grids [Fioretto *et al.*, 2017] and radio frequency allocation [Monteiro *et al.*, 2012].

Complete algorithms for DCOPs [Hirayama and Yokoo, 1997; Petcu and Faltings, 2005; Modi *et al.*, 2005; Yeoh *et al.*, 2010; Netzer *et al.*, 2012; Litov and Meisels, 2017] aim to find the optimal solution but incur exponential coordination overheads since solving DCOPs is NP-Hard. In contrast, incomplete algorithms [Maheswaran *et al.*, 2004; Zhang *et al.*, 2005; Okamoto *et al.*, 2016; Ottens *et al.*, 2017; Hoang *et al.*, 2018; Nguyen *et al.*, 2019] trade the solution quality for smaller computational efforts and can scale up to large problems.

Max-sum and its variants [Farinelli *et al.*, 2008; Rogers *et al.*, 2011; Zivan *et al.*, 2017; Chen *et al.*, 2018] are popular incomplete algorithms built upon the Generalized Distributive Law (GDL) [Aji and McEliece, 2000] and have been applied to many real-world domains due to their ability of directly handling n-ary constraints. However, these algorithms face a significant scalability challenge. In more detail, Max-sum implements belief propagation on a factor graph [Kschischang *et al.*, 2001] by optimizing the sum of local utility functions and corresponding query messages. As a result, the computational effort grows exponentially with respect to the arity of each utility function, which prohibits Max-sum from scaling up to large systems.

Therefore, a number of acceleration algorithms for GDL-based algorithms were proposed to improve their scalability and can be generally divided into BnB-based and sorting-based algorithms. BnB-based algorithms including BnB-MS [Stranders *et al.*, 2009] and BnB-FMS [Macarthur *et al.*, 2011] construct an estimation for each partial assignment and employ branch-and-bound to reduce the search space. Nevertheless, these algorithms compute estimations by either brute-force or domain-specific knowledge, which limits their generality. Recently, FDSP [Chen *et al.*, 2019] was proposed to implement generic branch-and-bound by using dynamic-programming to construct domain-agnostic estimations.

On the other hand, sorting-based algorithms including G-FBP [Kim and Lesser, 2013] and GDP [Khan *et al.*, 2018] require (partially) sorted local utilities to perform acceleration. Specifically, G-FBP only sorts for top $cd^{\frac{n-1}{2}}$ values of the search space and presumes that the highest utility can be found in the range. Here, c is a constant, d is the maximal domain size and n is the arity of a utility function, respectively. However, the algorithm has to perform an exhaustive traverse when the assumption fails. In contrast, GDP constructs a completely sorted local utility list for each assignment of each variable. Then it uses the entry with the highest local utility to compute a one-shot lower bound to prune suboptimal entries.

However, the existing methods could perform poorly when

dealing with dense local utilities. In more detail, FDSP would not be able to find a high-quality lower bound promptly when utility functions are highly structured since it sequentially exhausts the whole search space. On the other hand, although GDP attempts to find a more efficient one by considering the local utilities, the one-shot nature still cannot guarantee the quality. Additionally, the pruned range returned by GDP would contain many entries whose local utilities are close to each other, which also results in a poorly pruned rate. Unfortunately, dense utility functions and ties are very common in real-world scenarios. For example, in a NetRad system [Kim *et al.*, 2011] the utility of scanning a weather phenomenon does not increase linearly w.r.t. the scanning quality. As a result, the utility function could be extremely dense when the scanning quality is high.

In this paper, we aim to cope with dense local utilities from the perspectives of both bound quality and search space organization and develop more efficient sorting-based acceleration algorithms. To ensure bound quality, we integrate both search and pruning by iteratively updating the lower bound. Then we overcome the inability of pruning tied entries in domain pruning techniques by organizing the search space of these entries into AND/OR trees to enable efficient branch-and-bound. Finally, we discretize the utility range to balance the reconstruction overhead and the pruning efficiency and to further reduce the sorting overhead. We theoretically show our algorithms are sound and outperform GDP in terms of pruning efficiency. Our extensive empirical evaluations also confirm the great superiorities over the state-of-the-art.

2 Backgrounds

In this section, we review preliminaries including DCOPs, Max-sum and GDP.

2.1 Distributed Constraint Optimization Problems

A Distributed Constraint Optimization Problem (DCOP) [Modi *et al.*, 2005] can be defined by a tuple $\langle A, X, D, F\rangle$ where $A = \{a_1, \ldots, a_p\}$ is the set of agents, $X = \{x_1, \ldots, x_q\}$ is the set of variables, $D = \{D_1, \ldots, D_q\}$ is the set of discrete domains and $F = \{f_1, \ldots, f_m\}$ is the set of utility functions. Each variable x_i takes a value from domain D_i and each function $f_j : \mathbf{x_j} \to \mathbb{R}_{\geq 0}$ specifies a utility for each possible combination of involved variables $\mathbf{x_j} \subseteq X$. For the sake of simplicity, we assume that each agent controls a variable (i.e., $p = q$). The objective of a DCOP is to find an assignment for each variable to maximize the global utility. That is,

$$X^* = \arg\max_X \sum_{f_j(\mathbf{x_j}) \in F} f_j(\mathbf{x_j})$$

2.2 Max-sum

Max-sum [Farinelli *et al.*, 2008] is a GDL-based message-passing incomplete algorithm for DCOPs operating on a factor graph. A factor graph [Kschischang *et al.*, 2001] is a bipartite graph representation to a DCOP, which consists of variable nodes representing variables and function nodes representing utility functions in the DCOP, respectively. Fig.1

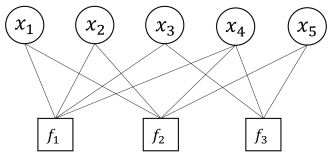


Figure 1: A factor graph

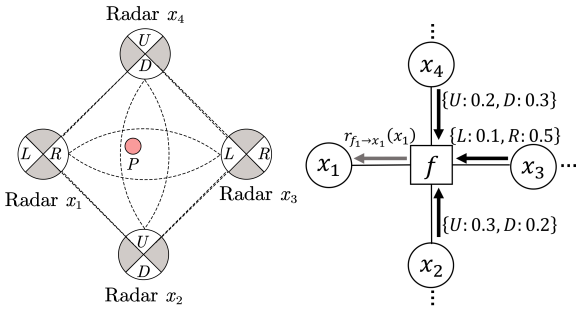


Figure 2: A part of a NetRad system and local interactions

presents a factor graph consisting of 3 function nodes and 5 variable nodes.

Max-sum implements belief propagation via query messages and response messages. Formally, the query message from variable node $x_i \in \mathbf{x_j}$ to function node f_j is given by

$$q_{x_i \to f_j}(x_i) = \sum_{f_k \in N_{x_i} \setminus \{f_j\}} r_{f_k \to x_i}(x_i) - \alpha \qquad (1)$$

where N_{x_i} is neighboring function nodes of x_i, $r_{f_k \to x_i}(x_i)$ is the response message from f_k and α is a normalization term to control the magnitude of each entry in the message. The response message from function node f_j to variable node $x_i \in \mathbf{x_j}$ is given by

$$r_{f_j \to x_i}(x_i) = \max_{\mathbf{x_j} \setminus \{x_i\}} \left(f_j(\mathbf{x_j}) + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(x_k) \right) \qquad (2)$$

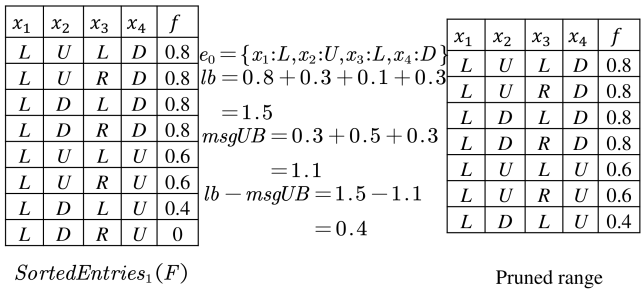
A variable node x_i makes a decision by choosing the assignment with the highest utility under the current belief. That is,

$$x_i^* = \arg\max_{v_i \in D_i} \sum_{f_k \in N_{x_i}} r_{f_k \to x_i}(v_i)$$

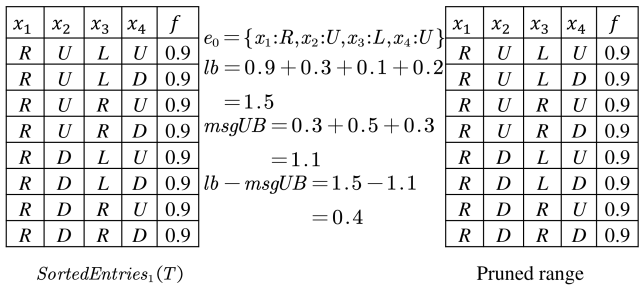
2.3 Generic Domain Pruning

It can be concluded that the computational overhead of Eq.(2) is exponential in the arity of utility function f_j, which prohibits Max-sum from scaling up to large systems. Generic Domain Pruning (GDP) [Khan *et al.*, 2018] is the state-of-the-art acceleration algorithm operating on completely sorted utility functions. In more detail, for each variable $x_i \in \mathbf{x_j}$ and assignment $v_i \in D_i$, f_j computes an ordered list $SortedEntries_i(v_i)$ according to its local utilities, where each entry $e \in SortedEntries_i(v_i)$ is an assignment to $\mathbf{x_j}$ such that $x_i = v_i$. When computing the utility for $x_i = v_i$, f_j constructs a one-shot lower bound lb and the message upper bound $msgUB$. That is,

$$lb = f_j(e_0) + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(e_0[x_k])$$



(a) The traces of GDP when computing for $x_1 = L$



(b) The traces of GDP when computing for $x_1 = R$

Figure 3: The traces of GDP when applied to the example in Fig.2

$$msgUB = \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} \max_{x_k} q_{x_k \to f_j}(x_k)$$

where e_0 is the first element in $SortedEntries_i(v_i)$ and $e_0[x_k]$ is the assignment of x_k in entry e_0. Then f_j returns a pruned range $P \subseteq SortedEntries_i(v_i)$ such that $f_j(e) + msgUB \geq lb, \forall e \in P$.

3 Motivation

In this section, we show GDP could fail to reduce the search space when there exist dense local utilities. Consider a part of a NetRad system [Kim *et al.*, 2011] shown in the left-hand side of Fig.2. There are four radars and a weather phenomenon P. For the sake of simplicity, we assume that each radar can only scan one of two sectors (i.e., $\{L, R\}$ for x_1 and x_3, $\{U, D\}$ for x_2 and x_4). The utility for a radar x_i scanning P depends on the distance between x_i and P. Assume that the utilities for scanning P by individual radars are 0.9, 0.6, 0.4 and 0.8, respectively. Finally, we consider the task as non-pinpointing. That is, given the scanning strategies of radars, the aggregated utility is the maximal one among the individual radars that scan P.

Given the incoming query messages shown in the right-hand side of Fig.2, we are going to compute the response message for radar x_1. It can be concluded from Fig.3 that GDP could perform poorly due to dense local utilities and ties. More specifically, when computing the utility for $x_1 = L$, the one-shot lower bound is 1.5 and the message upper bound is 1.1. As a result, any entry with local utility less than 0.4 is filtered out. However, since the local utilities are densely distributed in the range of $[0.4, 0.8]$, GDP can only prune one entry and result in a poorly pruned rate of 12.5%.

The pathology could be exacerbated when computing for $x_1 = R$ where all the local utilities are the same. In this case

Algorithm 1: GD^{2}P for function node f_j

```
When Initialization:
1   foreach x_i ∈ x_j do
2       foreach v_i ∈ D_i do
3           SortedEntries_i(v_i) ← arg sort
              f_j(x_i = v_i, ·) in non-increasing order
When computing a response message for x_i ∈ x_j:
4   msgUB ← Σ_{x_k ∈ x_j \ {x_i}} max_{v_k ∈ D_k} q_{x_k → f_j}(v_k)
5   foreach v_i ∈ D_i do
6       e ← the first element in SortedEntries_i(v_i)
7       util* ← −∞, lb ← −∞
8       while e≠NIL and f_j(e) ≥ lb do
9           u ← f_j(e) + Σ_{x_k ∈ x_j \ {x_i}} q_{x_k → f_j}(e[x_k])
10          util* ← max(util*, u)
11          lb ← util* − msgUB
12          e ← the next element in SortedEntries_i(v_i)
13      r_{f_j → x_i}(v_i) ← util*
14  return r_{f_j → x_i}(x_i)
```

GDP still cannot prune any entry even if we construct the most efficient lower bound according to $\{R, U, R, D\}$. That is because given a linearly structured search space GDP cannot discard suboptimal entries with the same local utility in advance.

In fact, dense local utilities are ubiquitous in real-world scenarios due to the law of diminishing marginal utility. Therefore, we aim to develop more efficient sorting-based acceleration algorithms for incomplete GDL-based algorithms by alleviating the negative effect of dense local utilities.

4 Proposed Methods

In this section, we first present our proposed algorithms for accelerating Max-sum. Then we briefly discuss the modifications when applying our methods to other versions of belief propagation.

4.1 GD^{2}P

As demonstrated earlier, the quality of the lower bound is the key of effective pruning, especially when local utilities are dense. Therefore, we propose to integrate both search and pruning to iteratively reduce the search space. Thus, the pruning is no longer a one-shot procedure and the scheme is referred as Generic Dynamic Domain Pruning (GD^{2}P). Alg.1 presents the sketch of GD^{2}P.

Similar to GDP, GD^{2}P also begins with complete sorting of each utility function (line 1-3). When computing a response message for a variable node $x_i \in \mathbf{x_j}$, it searches for the highest utility for each assignment $v_i \in D_i$ by exhausting the sorted entries whose local utility is no less than the running lower bound lb in a sequential order (line 8-13). Here, the lower bound is updated whenever a higher utility is found (line 10-11).

Next, we theoretically show its correctness and superiority over GDP.

Theorem 1. *GD^{2}P guarantees the optimality of Eq.(2).*

Algorithm 2: ART-GD^{2}P for function node f_j

```
When Initialization:
1   foreach x_i ∈ x_j do
2       foreach v_i ∈ D_i do
3           SortedUtil_i(v_i) ← all distinct utilities of
              f_j(x_i = v_i, ·)
4           sort SortedUtil_i(v_i) in descending order
5           foreach u ∈ SortedUtil_i(v_i) do
6               trees_i(v_i, u) ← build an AND/OR tree
                  representing all entries E such that
                  f_j(e) = u, ∀e ∈ E, e[x_i] = v_i
  When computing a response message for x_i ∈ x_j:
7   msgUB ← Σ_{x_k ∈ x_j \ {x_i}} max_{v_k ∈ D_k} q_{x_k → f_j}(v_k)
8   foreach v_i ∈ D_i do
9       u ← the first element in SortedUtil_i(v_i)
10      util* ← −∞, lb ← −∞
11      while u≠NIL and u ≥ lb do
12          t ← trees_i(v_i, u)
13          util ← Branch-and-bound(t, x_i, util*)
14          util* ← max(util*, util)
15          lb ← util* − msgUB
16          u ← the next element in SortedUtil_i(v_i)
17      r_{f_j → x_i}(v_i) ← util*
18  return r_{f_j → x_i}(x_i)
```

Proof. Assume that the optimal entry e^* is pruned by GD^{2}P when $x_i = v_i$. According to line 4, 8 and 11 of Alg.1, we have

$$f_j(e^*) < lb = util^* - msgUB$$

where $util^*$ is the highest utility returned by GD^{2}P. Combing with query messages, we have

$$f_j(e^*) + Q_i(e^*) < util^* - msgUB + Q_i(e^*)$$

where $Q_i(e^*) = \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(e^*[x_k])$. Since

$$\sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(e^*[x_k]) \leq \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} \max_{v_k \in D_k} q_{x_k \to f_j}(v_k)$$

it must have

$$f_j(e^*) + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(e^*[x_k]) < util^*$$

which is contradictory to the assumption. Thus, GD^{2}P guarantees the optimality of Eq.(2). □

Theorem 2. *GD^{2}P never explores the entries pruned by GDP.*

Proof. Assume that GD^{2}P explores an entry e which is pruned by GDP. Denote the lower bound constructed by GDP as lb^{GDP} and the running lower bound in GD^{2}P as lb^{GD^2P}. According to line 8 of Alg.1, since e is explored by GD^{2}P but pruned by GDP, it must have

$$lb^{GDP} - \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} \max_{v_k \in D_k} q_{x_k \to f_j}(v_k) > f_j(e) \geq lb^{GD^2P}$$

which is contradictory to line 10-11. □

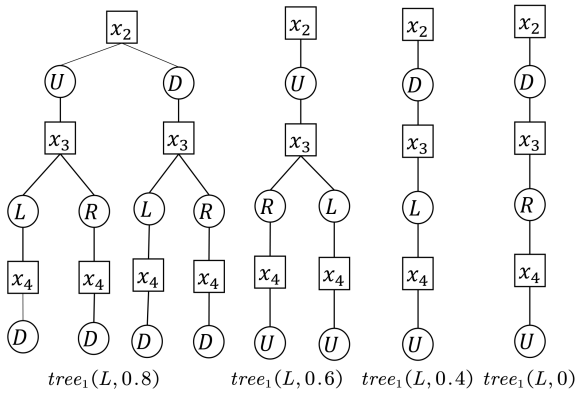


Figure 4: AND/OR trees for $x_1 = L$ when applied to Fig.2

4.2 ART-GD^{2}P

Both GDP and GD^{2}P perform poorly when there are ties in utility functions since they use a linear structure to organize the search space. As a result, they have to exhaust all the tied entries whose local utility is no less than the lower bound. Thus, we propose to organize the search space of tied entries into an AND/OR tree so as to enable branch-and-bound to reduce the search space. The scheme is referred as And/oR Tree-based GD^{2}P (ART-GD^{2}P) and Alg.2 presents the sketch.

Different than completely sorting a utility function in GDP and GD^{2}P, ART-GD^{2}P only sorts for *distinct* local utilities for each assignment of each variable in the preprocessing phase (line 1-4). After that, it builds an AND/OR tree for the entries with the same local utility (line 5-6). Note that the construction of AND/OR trees can be done by a sequential iteration over the utility function as the trees can be built incrementally. Similar to GD^{2}P, it maintains a running lower bound lb and terminates whenever the local utility is less than the lower bound (line 11-16).

It is noteworthy that instead of iterating over the sorted entries in GDP and GD^{2}P, ART-GD^{2}P iterates over the distinct sorted utilities and performs branch-and-bound to reduce the search space of tied entries. In more detail, for each distinct local utility u, ART-GD^{2}P exhausts the corresponding AND/OR tree $tree_i(v_i, u)$ by constructing an estimation ub^{PA} for each partial assignment PA and discarding the partial assignment whenever the estimation is less than the known highest utility (line 12-13). Formally, the estimation is given by

$$ub^{PA} = u + \sum_{x_k \in PA} q_{x_k \to f_j}(PA[x_k]) + \sum_{x_k \notin PA} \max_{v_k \in D_k} q_{x_k \to f_j}(v_k)$$

Take Fig.2 as an example. Given $x_1 = L$, ART-GD^{2}P first sorts distinct local utilities $\{0.8, 0.6, 0.4, 0\}$ and builds an AND/OR tree for each of them. Fig.4 presents the AND/OR trees. Here, we omit the AND/OR nodes for x_1. When computing utility for $x_1 = L$, ART-GD^{2}P first performs depth-first branch-and-bound to exhaust the AND/OR tree $tree_1(L, 0.8)$. It can be concluded that the algorithm terminates after reaching the first two complete assignments $\{L, U, L, D\}$ and $\{L, U, R, D\}$ since it finds the highest utility 1.9, which results in a pruned rate of 75%.

We now show its correctness and its superiority over GD^{2}P.

Theorem 3. *ART-GD^{2}P guarantees the optimality of Eq.(2).*

Proof. There are two lines that prune the search space, i.e., line 11 and line 13 of Alg.2. We have shown that line 11 cannot prune the highest utility in Theorem 1. Thus we only need to show that line 13 never prunes the optimal assignment. Assume that the optimal full assignment A^* is pruned by line 13. Thus it must exist a prefix $PA \subset A^*$ such that

$$ub^{PA} < util^*$$

where $util^*$ is the highest utility found by ART-GD^{2}P. In fact,

$$\begin{aligned} ub^{PA} &= f_j(A^*) + \Bigg(\sum_{x_k \in PA} q_{x_k \to f_j}(PA[x_k]) \\ &\quad + \sum_{x_k \notin PA} \max_{v_k \in D_k} q_{x_k \to f_j}(v_k) \Bigg) \\ &\geq f_j(A^*) + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(A^*[x_k]) \end{aligned}$$

Thus we have $f_j(A^*) + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(A^*[x_k]) < util^*$, which is contradictory to the definition of A^*. The optimality is hereby guaranteed. □

Theorem 4. *ART-GD^{2}P never explores the assignments pruned by GD^{2}P.*

Proof. Assume that ART-GD^{2}P explores a full assignment A which is pruned by GD^{2}P. Denote the lower bound of ART-GD^{2}P when exploring A as lb^{ART} and the lower bound of GD^{2}P when pruning A as lb^{GD^2P}, respectively. It must have

$$lb^{GD^2P} > f_j(A) \geq lb^{ART} \tag{3}$$

Since GD^{2}P sequentially exhausts the search space, it must exist an assignment $A' \prec A$ such that

$$\begin{aligned} lb^{GD^2P} &= f_j(A') \\ &\quad + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(A'[x_k]) - \max_{v_k \in D_k} q_{x_k \to f_j}(v_k) \\ &> f_j(A) \end{aligned} \tag{4}$$

which indicates that $f_j(A') > f_j(A)$. On the other hand, ART-GD^{2}P explores distinct local utilities in a descending order (line 4, 9 and 15 of Alg.2). According to Eqs.(3-4), A' produces a higher utility than A. Thus, ART-GD^{2}P must have explored an assignment A'' such that

$$\begin{aligned} f_j(A'') + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(A''[x_k]) &\geq f_j(A') \\ &\quad + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(A'[x_k]) \end{aligned}$$

which implies

$$\begin{aligned} lb^{ART} &\geq f_j(A') + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(A'_{x_k}) - \max_{v_k \in D_k} q_{x_k \to f_j}(v_k) \\ &= lb^{GD^2P} \end{aligned} \tag{5}$$

x_1	x_2	x_4	x_5	f_2
F	F	F	F	2
F	F	F	T	5
F	F	T	F	4.95
F	F	T	T	4.97
F	T	F	F	4.96
F	T	F	T	4.98
F	T	T	F	4.94
F	T	T	T	4.99

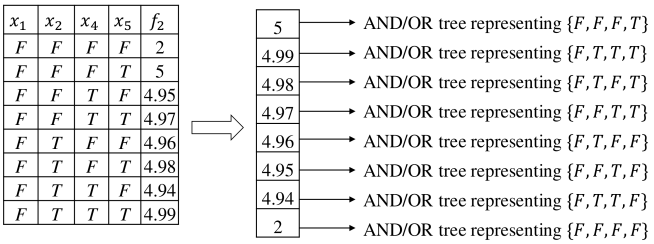


Figure 5: ART-GD^{2}P operating on extremely dense utilities

Therefore, Eq.(5) is contradictory to Eq.(3), which concludes the theorem. □

4.3 Discretization Mechanism

Since ART-GD^{2}P builds an AND/OR tree for each distinct local utility, each tree would correspond to a small search space when the local utilities are extremely dense, which would incur solution reconstructions. Consider the example in Fig.5. Assume that all the assignments with prefix $\{F, T\}$ are pruned by line 13 and the highest utility corresponds to assignment $\{F, F, F, F\}$. Obviously, ART-GD^{2}P has to visit the suboptimal partial assignment $\{F, T\}$ for 4 times in this case. Besides, a large number of distinct local utilities also result in a high sorting overhead.

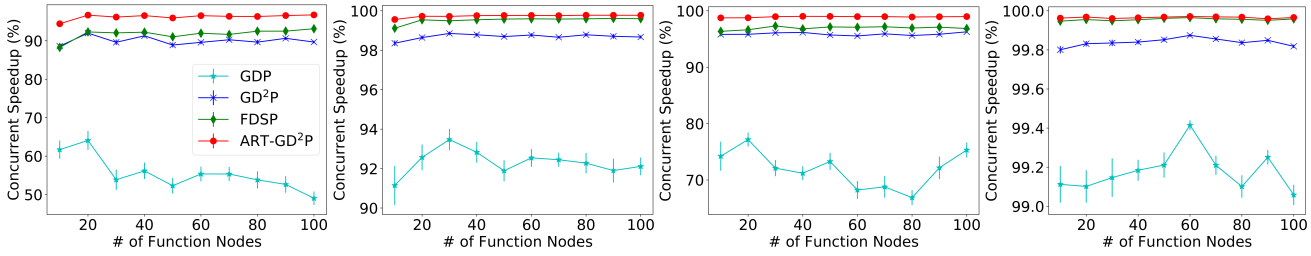
We overcome the problems by introducing a discretization mechanism. That is, instead of sorting and building AND/OR trees for distinct local utilities directly, we first group the utilities into several discrete slots. Then all the operations are applied to these slots. In more detail, we first divide the range of the local utilities into disjoint slots by a step size of t. Then we group all the utilities $\{u | val(s_i) - t < u \leq val(s_i)\}$ into slot $s_i \in S$. Here, $val(\cdot)$ returns the maximal utilities in a slot, which is used to perform pruning in line 11.

Finally, it is worth mentioning that ART-GD^{2}P with the discretization mechanism offers a tradeoff between domain pruning and reconstruction overhead. In more detail, a small t produces fine-grained slots and could prune the suboptimal utilities promptly (line 11). In contrast, a large t tends to build AND/OR trees corresponding to large search spaces, which reduces the reconstruction overhead.

4.4 Generalizing to Other GDL-based Algorithms

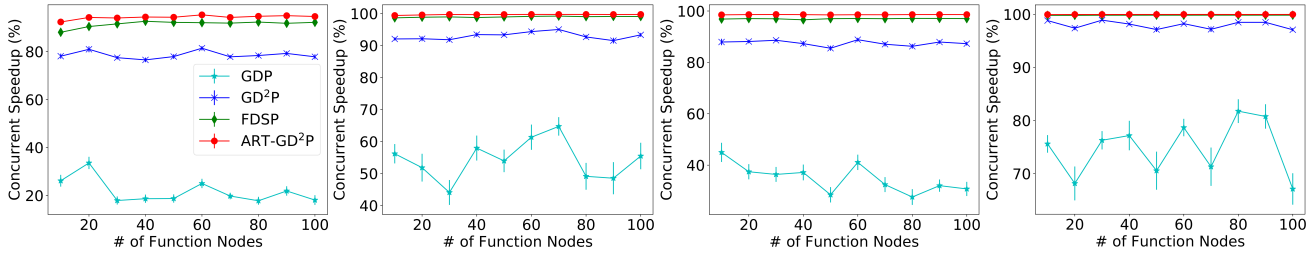
We note that our proposed algorithms can be easily adapted to the variants of Max-sum and other versions of belief propagation. For example, we could adapt our algorithms to speed up Max-product [Weiss and Freeman, 2001] by changing the summation operations to the product operations (i.e., line 4, 9 and 11 of Alg. 1, line 7, 13 and 15 of Alg. 2).

When combining with Max-sum variants like damped Max-sum [Cohen *et al.*, 2020], bounded Max-sum [Rogers *et al.*, 2011] and Max-sum_AD [Zivan *et al.*, 2017], our algorithms require few modifications since these variants still use Eq.(2) to compute response messages. However, there are variants which do not exactly follow Eq.(2). For example, a function node in Max-sum_ADSSVP [Chen *et al.*, 2018] needs to consider the assignments of some neighbors when computing the maximal utility. In this case, an additional



(a) Small domain, low arity (b) Small domain, high arity (c) Large domain, low arity (d) Large domain, high arity

Figure 6: Performance comparison on sparse problems



(a) Small domain, low arity (b) Small domain, high arity (c) Large domain, low arity (d) Large domain, high arity

Figure 7: Performance comparison on dense problems

verification procedure needs to be introduced to filter out the incompatible entries.

5 Experimental Evaluations

We empirically evaluate the performance of GDP, FDSP and our proposed algorithms when accelerating Max-sum on both random DCOPs and NetRad systems. The performance metrics we consider include pruned rate [Khan *et al.*, 2018], concurrent speedup, which is the improvement of the number of concurrent basic operations over vanilla Max-sum, and runtime. Here, basic operations are the access to query messages. For each of the configuration, we generate 50 random instances and terminate algorithms after 250 iterations, reporting the averaged metrics and standard errors as the results and the confidence intervals, respectively.

5.1 Results on Random DCOPs

We generate high arity and low arity factor graphs by uniformly selecting an arity for each function node from [2,5] and [5,8], respectively. Further, we generate large domain problems and small domain problems by uniformly selecting a domain size from [5,8] and [2,5] for each variable, respectively. Finally, we consider the factor graphs with the variable tightness [Chen *et al.*, 2019] between 0.1 and 0.5 as sparse graphs and the ones with the variable tightness between 0.5 and 0.9 as dense graphs. The utilities of each function node are uniformly sampled from (0,1000). For ART-GD^{2}P, we use an empirically tuned step size $t = 50$.

Fig.6 presents the results when solving sparse problems. It can be seen that FDSP performs worse than ART-GD^{2}P. That is due to the fact that FDSP performs a depth-first search on a huge AND/OR tree without any *a priori* knowledge. On the other hand, our ART-GD^{2}P performs branch-and-bound on a sorted array of AND/OR trees and can find an efficient lower bound more promptly, reducing about 95% - 99.9% of concurrent operations, which demonstrates great superiorities over the other competitors. Besides, our GD^{2}P maintains a running lower bound and significantly improves the performance of GDP, which highlights the importance of tight lower bounds in domain pruning techniques.

Fig.7 presents the results when solving dense problems. Compared to the ones in sparse problems, the variable nodes in these problems are over-constrained due to high variable tightness. As a result, the query messages could be multimodal and the one-shot lower bound constructed by GDP is not necessarily efficient. Therefore, GDP performs poorly and is strictly dominated by other competitors. Besides, it is interesting to find that compared to the results on sparse problems, the gaps between GD^{2}P and FDSP are widened. In addition to the multimodality of query messages, another reason for this is that FDSP performs a depth-first search on AND/OR trees, which significantly reduces solution reconstructions. Finally, combing both domain pruning and branch-and-bound techniques, our proposed ART-GD^{2}P exhibits the best performance on all of the tasks, which demonstrates its merits on general n-ary DCOPs.

To examine the performance on the problems with dense local utilities, we consider the problems with 100 function nodes, low arity, large domain size and the variable tightness of 0.5. For each problem, there are a number of dense function nodes whose utilities are selected according to a power-law distribution. More specifically, the probability of selecting a utility $u \in (0, 1000)$ is proportional to $(1000 - u)^{-\alpha}$.

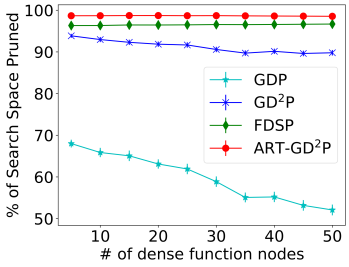
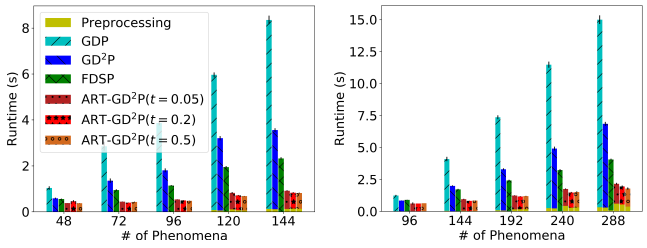


Figure 8: Performance comparison on the problems with dense local utilities



(a) 48 radars (b) 96 radars

Figure 9: Performance comparison on NetRad systems

In our experiments, we set $\alpha = 1.1$. Fig.8 presents the performance comparison in terms of the pruned rate.

It can be seen that the performance of GDP decreases by near 20% w.r.t. growing dense function nodes, which indicates that GDP is sensitive to dense utility functions. That is due to the fact that the pruned range could contain many entries since the utilities are close to each other in dense utility functions. On the other hand, although GD^{2}P also relies on domain pruning to reduce the search space, it is much more robust to the presence of dense function nodes due to the iteratively tighten lower bound. In fact, with the growing of dense function nodes its performance only drops by 4%. Finally, dense function nodes can hardly deteriorate the performance of ART-GD^{2}P since it can still perform effective branch-and-bound according to the query messages when local utilities are not distinguishable.

5.2 Results on NetRad Systems

We consider the NetRad systems with 48 and 96 radars which are arranged into 6×8 and 8×12 grids, respectively. Weather phenomena with different sizes, weights and types are randomly generated across grids. Each phenomenon is associated with a utility function defined according to [Pepyne *et al.*, 2007]. In this set of experiments, the maximal utility of a function is 1, the maximal arity is 4 and the domain size of a variable can go up to 15. Fig.9 gives the experimental results.

It can be concluded that GDP performs poorly and the runtime grows quickly w.r.t. the number of phenomena in both cases. In contrast, our proposed GD^{2}P and ART-GD^{2}P significantly outperform GDP, only requiring about a half and one-eighth of its runtimes, respectively. On the other hand, although FDSP outperforms domain pruning variants, it is still dominated by ART-GD^{2}P. This is due to the fact that with the sorted AND/OR trees our proposed ART-GD^{2}P can find a high-quality lower bound quickly despite of highly-structured utility functions. Finally, ART-GD^{2}P with a large t reduces the runtimes in both preprocessing phase and pruning phase, which demonstrates the necessity of the discretization mechanism.

6 Conclusion

In this paper, we demonstrate that the presence of dense utility functions or utility ties would deteriorate the performance of GDP. To alleviate the negative effect of densely distributed local utilities, we propose to iteratively update the lower bound and organize the tied entries to AND/OR trees. Finally, we present a discretization mechanism which offers a tradeoff between the reconstruction overhead and pruning efficiency. We theoretically show their correctness and superiorities over GDP. Our empirical evaluations also demonstrate their effectiveness on both synthetic and realistic benchmarks.

We consider two lines of future work. First, ART-GD^{2}P relies on an empirically tuned step size t and uninformed depth-first search, which could be undesirable in real-world applications. Therefore, heuristics for determining t and mechanisms for improving pruning ability need to be further investigated. Second, we plan to extend our work to cope with changing utility functions in a dynamic environment.

Acknowledgements

This research is supported by Singapore National Research Foundation projects AISG-RP2019-0013, NSOE-TSS2019-01, and NTU.

References

[Aji and McEliece, 2000] Srinivas M Aji and Robert J McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.

[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.

[Chen *et al.*, 2019] Ziyu Chen, Xingqiong Jiang, Yanchen Deng, Dingding Chen, and Zhongshi He. A generic approach to accelerating belief propagation based incomplete algorithms for DCOPs via a branch-and-bound technique. In *AAAI*, pages 6038–6045, 2019.

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

[Farinelli *et al.*, 2008] Alessandro Farinelli, Alex Rogers, Adrian Petcu, and Nicholas R Jennings. Decentralised coordination of low-power embedded devices using the Max-sum algorithm. In *AAMAS*, pages 639–646, 2008.

[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 *AAMAS*, pages 999–1007, 2017.

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

[Hirayama and Yokoo, 1997] Katsutoshi Hirayama and Makoto Yokoo. Distributed partial constraint satisfaction problem. In *CP*, pages 222–236, 1997.

[Hirayama *et al.*, 2019] K Hirayama, K Miyake, T Shiotani, and T Okimoto. DSSA+: Distributed collision avoidance algorithm in an environment where both course and speed changes are allowed. *International Journal on Marine Navigation and Safety of Sea Transportation*, 13(1):117–123, 2019.

[Hoang *et al.*, 2018] Khoi D Hoang, Ferdinando Fioretto, William Yeoh, Enrico Pontelli, and Roie Zivan. A large neighboring search schema for multi-agent optimization. In *CP*, pages 688–706, 2018.

[Khan *et al.*, 2018] Md Mosaddek Khan, Long Tran-Thanh, and Nicholas R Jennings. A generic domain pruning technique for GDL-based DCOP algorithms in cooperative multi-agent systems. In *AAMAS*, pages 1595–1603, 2018.

[Kim and Lesser, 2013] Yoonheui Kim and Victor Lesser. Improved Max-sum algorithm for DCOP with n-ary constraints. In *AAMAS*, pages 191–198, 2013.

[Kim *et al.*, 2011] Yoonheui Kim, Michael Krainin, and Victor Lesser. Effective variants of the Max-sum algorithm for radar coordination and scheduling. In *WI/IAT*, pages 357–364, 2011.

[Kschischang *et al.*, 2001] Frank R Kschischang, Brendan J Frey, Hans-Andrea Loeliger, et al. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

[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 ADCOPs. *Artificial Intelligence*, 252:83–99, 2017.

[Macarthur *et al.*, 2011] Kathryn S. Macarthur, Ruben Stranders, Sarvapali D. Ramchurn, and Nicholas R. Jennings. A distributed anytime algorithm for dynamic task allocation in multi-agent systems. In *AAAI*, pages 701–706, 2011.

[Maheswaran *et al.*, 2004] Rajiv T Maheswaran, Jonathan P Pearce, and Milind Tambe. Distributed algorithms for DCOP: A graphical-game-based approach. In *ISCA PDCS*, pages 432–439, 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 *WCNC*, 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.

[Nguyen *et al.*, 2019] Duc Thien Nguyen, William Yeoh, Hoong Chuin Lau, and Roie Zivan. Distributed Gibbs: A linear-space sampling-based DCOP algorithm. *Journal of Artificial Intelligence Research*, 64:705–748, 2019.

[Okamoto *et al.*, 2016] Steven Okamoto, Roie Zivan, and Aviv Nahon. Distributed breakout: Beyond satisfaction. In *IJCAI*, 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):69:1–69:27, 2017.

[Pepyne *et al.*, 2007] D Pepyne, D Westbrook, B Philips, E Lyons, M Zink, and J Kurose. A design for distributed collaborative adaptive sensing of the atmosphere. Technical report, University of Massachusetts, 2007.

[Petcu and Faltings, 2005] Adrian Petcu and Boi Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, pages 266–271, 2005.

[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.

[Stranders *et al.*, 2009] Ruben Stranders, Alessandro Farinelli, Alex Rogers, and Nicholas R. Jennings. Decentralised coordination of mobile sensors using the Max-sum algorithm. In *IJCAI*, pages 299–304, 2009.

[Weiss and Freeman, 2001] Yair Weiss and William T Freeman. On the optimality of solutions of the Max-product belief-propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, 47(2):736–744, 2001.

[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 *et al.*, 2017] Roie Zivan, Tomer Parash, Liel Cohen, Hilla Peled, and Steven Okamoto. Balancing exploration and exploitation in incomplete Min/Max-sum inference for distributed constraint optimization. *Autonomous Agents and Multi-Agent Systems*, 31(5):1165–1207, 2017.