A Generic Approach to Accelerating Belief Propagation Based Incomplete Algorithms for DCOPs via a Branch-and-Bound Technique

Ziyu Chen,[1] **Xingqiong Jiang,**[1] **Yanchen Deng,**[2,*] **Dingding Chen,**[1] **Zhongshi He**[1]

College of Computer Science, Chongqing University, Chongqing, China

[1]{chenziyu, jxq, dingding, zshe}@cqu.edu.cn, [2]dyc941126@126.com

Abstract

Belief propagation approaches, such as Max-Sum and its variants, are important methods to solve large-scale Distributed Constraint Optimization Problems (DCOPs). However, for problems with n-ary constraints, these algorithms face a huge challenge since their computational complexity scales exponentially with the number of variables a function holds. In this paper, we present a generic and easy-to-use method based on a branch-and-bound technique to solve the issue, called Function Decomposing and State Pruning (FDSP). We theoretically prove that FDSP can provide monotonically non-increasing upper bounds and speed up belief propagation based incomplete DCOP algorithms without an effect on solution quality. Also, our empirically evaluation indicates that FDSP can reduce 97% of the search space at least and effectively accelerate Max-Sum, compared with the state-of-the-art.

Introduction

Distributed Constraint Optimization Problems (DCOPs) which require agents to coordinate their decisions to optimize a global objective, are a fundamental framework for modeling multi-agent coordination in multi-agent systems (Hirayama and Yokoo 1997). Thus, DCOPs are widely deployed in some real world coordination tasks including meeting scheduling (Enembreck and Barthès 2012), sensor networks (Farinelli, Rogers, and Jennings 2014), power networks (Fioretto et al. 2017), etc.

Algorithms for DCOPs can be classified into two categories: complete and incomplete, according to whether they guarantee to find the optimal solution. Complete algorithms (Hirayama and Yokoo 1997; Modi et al. 2005; Petcu and Faltings 2005; Yeoh, Felner, and Koenig 2008; Vinyals, Rodriguez-Aguilar, and Cerquides 2009; Gershman, Meisels, and Zivan 2009) can get the optimal solutions but incur exponential communication or computation overheads since DCOPs are NP-hard. In contrast, incomplete algorithms (Maheswaran, Pearce, and Tambe 2004; Arshad and Silaghi 2004; Zhang et al. 2005; Ottens, Dimitrakakis, and Faltings 2012; Nguyen, Yeoh, and Lau 2013;

Okamoto, Zivan, and Nahon 2016) trade accuracy for computation time and memory so that they can be applied to large-scale problems. As a kind of incomplete algorithms based on belief propagation, Max-Sum (Farinelli et al. 2008) and its variants (Rogers et al. 2011; Zivan and Peled 2012; Chen, Deng, and Wu 2017) have drawn a lot of attention since they can easily be deployed to any DCOP setting. Moreover, they can explicitly handle n-ary constraints and more variables per agent (Cerquides et al. 2014). In more detail, agents in Max-Sum propagate and accumulate beliefs through the whole factor graph. And each agent only holds its belief about the utility for each possible assignment and continuously updates its belief based on the messages received from its neighbors.

In spite of many advantages of belief propagation approaches, they suffer from a huge challenge in scalability. Specifically, they perform maximization operations repeatedly to locally accumulate beliefs for the involved variables, given the local utility function and a set of incoming messages. The computation complexity of this step grows exponentially as the number of constraint arities. In other words, when a constraint function holds n variables and the domain size of each variable is d, Max-Sum needs to perform d^n maximization operations to yield the best assignment for each variable.

To address the issue, two kinds of methods were proposed to improve the scalability of belief propagation approaches. The first kind is the algorithms based on a branch-and-bound technique including BnB-MS (Stranders et al. 2009) and BnB-FMS (Macarthur et al. 2011) which both consist of a preprocessing phase and a pruning phase. In the preprocessing phase, the two algorithms use localized message-passing to simplify DCOPs. Specifically, BnB-MS reduces the number of moves that each agent needs to consider in coordinating mobile sensors while BnB-FMS removes tasks that an agent should never perform in dynamic task allocations. In the pruning phase, both algorithms reduce the search space using a branch-and-bound technique to speed up maximization operations. Unfortunately, these algorithms require to exchange a lot of messages in their preprocessing phases. Moreover, the bounds in these algorithms are computed by either brute force or domain-specific knowledge, which limits their applicability.

The second kind of approaches is sorting based, such as

G-FBP(Kim and Lesser 2013) and GDP(Khan et al. 2018), which is applicable to all DCOP settings. G-FBP uses partially sorted lists to adapt FBP. Specifically, it selects and sorts the top $cd^{\frac{n-1}{2}}$ values of the search space, presuming that the maximum value can be found from the selected range. Here, c is a constant. However, G-FBP will incur additional computation once the maximum value cannot be found within the selected range. Different from G-FBP, the main idea of GDP is to explore only the rows that can cover the differences between the sum of the maximal utility of each message and the message utility corresponding to the assignment that produces the largest local utility. Thus, GDP needs to sort the local utilities of all function-nodes independently by each assignment of each variable in the preprocessing phase and $\mathcal{V}_i$ is the sorted result of each assignment i. Then, GDP returns a pruned range $[p, q]$ or $[p, q)$ according to whether $q == p - t$, where $p = \max(\mathcal{V}_i)$, $q = \max_c\{c \in \mathcal{V}_i : c \le (p - t)\}$ and $t = m - b$. Here, m is the summation of the maximum value for each received message from other variable-nodes and b is the summation for the corresponding values of p from the incoming messages of a function-node. However, GDP needs additional time to perform sorting operations in the preprocessing phase. More importantly, GDP is an one-shot pruning procedure that cannot use the learned experience from the assignment combinations explored to dynamically prune the search space.

Given the background, we devote to develop a generic and fast method for belief propagation based on a branch-and-bound technique, called Function Decomposing and State Pruning (FDSP). In more detail, we propose a domain-independent approach based on dynamic programming to effectively evaluate the upper bound of a given partial assignment, which overcomes the aforementioned drawbacks of BnB-MS and BnB-FMS. We further enforce the upper bounds by exploiting the fact that the assignment of the target variable is given. Finally, we prune the search space whenever the upper bound of a partial assignment is no greater than the best lower bound explored so far. The experimental results show the effectiveness of FDSP which can prune at least 97% of the search space when solving complex problems.

Background
Distributed Constraint Optimization Problems

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

- $A = \{a_1, a_2, \ldots, a_h\}$ is a set of agents.
- $X = \{x_1, x_2, \ldots, x_q\}$ is a set of variables.
- $D = \{D_1, D_2, \ldots, D_q\}$ is a set of finite and discrete domains, variable x_i taking an assignment value in D_i.
- $F = \{F_1, F_2, \ldots, F_r\}$ is a set of constraints, where each constraint $F_k : \mathbf{x_k} \to \mathbb{R}^+$ denotes how much utility is assigned to each possible combination of assignments of the involved variables $\mathbf{x_k} \subseteq X$.

Thus, a constraint function $F_k(\mathbf{x_k})$ denotes the utility for each possible assignment combination of the variables in $\mathbf{x_k}$,
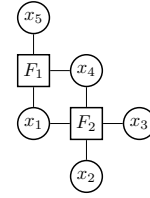

Figure 1: A DCOP instance using a factor graph

$n = |\mathbf{x_k}|$ represents the arity of F_k, and $d = |D_i|$ denotes the domain size of variable x_i. Note that the variables in $\mathbf{x_k}$ are ordered according to their own indices, where a variable $\mathbf{x_{k,i}}$ is ordered before a variable $\mathbf{x_{k,j}}$ if $i < j$.

Given this, the goal for a DCOP is to find the joint variable assignment X^* such that a given global objective function is maximal. Generally, the objective function is described as the sum over F:

$$X^* = \arg\max_X \sum_{F_k(\mathbf{x_k}) \in F, \mathbf{x_k} \subseteq X} F_k(\mathbf{x_k})$$

Max-Sum Algorithm

As a belief propagation approach, Max-Sum is a message-passing inference-based algorithm operating on factor graphs which comprise variable-nodes and function-nodes. Function-nodes which represent the constraints in a DCOP are connected to variable-nodes they depend on, while variable-nodes which represent the variables in a DCOP are connected to function-nodes they are involved in. As shown in Fig. 1, F_1 and F_2 are two function-nodes, x_1, x_2, x_3, x_4 and x_5 are variable-nodes, where x_1, x_2, x_3 and x_4 are connected to F_2, and x_1, x_4 and x_5 are connected to F_1. Here, F_1 (i.e., $F_1(\mathbf{x_1})$, where $\mathbf{x_1} = \{x_1, x_4, x_5\}$) is a 3-ary constraint and F_2 (i.e., $F_2(\mathbf{x_2})$, where $\mathbf{x_2} = \{x_1, x_2, x_3, x_4\}$) is a 4-ary one.

In Max-Sum, beliefs are propagated and accumulated through the whole factor graph via the messages exchanged between variable-nodes and function-nodes. The message from a variable-node x_i to a function-node $F_k(\mathbf{x_k})$, called the query message. It is defined by

$$Q_{x_i \to F_k}(x_i) = \alpha_{ik} + \sum_{F_j \in N_i \backslash F_k} R_{F_j \to x_i}(x_i) \tag{1}$$

where α_{ik} is a scalar set such that $\sum_{x_i} Q_{x_i \to F_k}(x_i) = 0$, $x_i \in \mathbf{x_k}$ and $N_i \backslash F_k$ is a set of neighbors of x_i except the target function-node F_k. The response message sent from a function-node $F_k(\mathbf{x_k})$ to a variable-node x_i is given by

$$R_{F_k \to x_i}(x_i) = \max_{\mathbf{x_k} \backslash x_i} \left(F_k(\mathbf{x_k}) + \sum_{x_j \in \mathbf{x_k} \backslash x_i} Q_{x_j \to F_k}(x_j) \right) \tag{2}$$

When a variable-node x_i makes its decision, it first accumulates the belief for each possible assignment from all messages it receives. Then, it selects a value to maximize the total utilities. The procedure can be formalized by:

$$x_i^* = \arg\max_{x_i} \sum_{F_k \in N_i} R_{F_k \to x_i}(x_i) \tag{3}$$

x_1	x_2	x_3	x_4	F_2
R	R	R	R	4
R	R	R	G	13
R	R	G	R	26
R	R	G	G	5
R	G	R	R	1
R	G	R	G	5
R	G	G	R	2
R	G	G	G	4
G	R	R	R	10
G	R	R	G	7
G	R	G	R	6
G	R	G	G	9
G	G	R	R	11
G	G	R	G	8
G	G	G	R	12
G	G	G	G	1
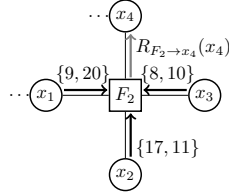
|(a)|(b)|

Figure 2: The utility matrix and message exchange for F_2

In this paper, we use the term "target variable" to denote the destination variable-node of the outgoing message being computed, and $\mathcal{M}_{\mathbf{x_{k,i}}}$ represents the message from variable $\mathbf{x_{k,i}}$.

Proposed Method

Motivation

The maximization operation in Eq. (2) is the most computationally expensive operation in Max-Sum. For a n-ary DCOP, the complexity of computing the response message is $O(d^n)$. Take Fig. 2 as an example. Assume that each variable takes a value in $\{R, G\}$ and the function-node F_2 has received the messages $\mathcal{M}_{x_1} = \{9, 20\}$, $\mathcal{M}_{x_2} = \{17, 11\}$ and $\mathcal{M}_{x_3} = \{8, 10\}$ from x_1, x_2 and x_3, respectively. Then, F_2 requires $d^n = 2^4 = 16$ operations to generate the message $R_{F_2 \to x_4}(x_4)$ since its domain size $d = 2$ and arity $n = 4$. Obviously, the complexity of this step grows exponentially as d and n scale up. Therefore, this is a huge challenge for scalability of belief propagation algorithms.

As mentioned earlier, some efforts have been made to optimize this maximization operation. Nevertheless, the improved algorithms based on a branch-and-bound technique including BnB-MS and BnB-FMS require a number of messages to be passed in the preprocessing phase. And, these algorithms were proposed for the exact application, which makes it difficult to directly solve general DCOPs. Besides, the algorithms based on sorting, such as G-FBP and GDP, suffer from some drawbacks although they are generic. Specifically, G-FBP cannot guarantee that the maximum value can be found in the selected range, which can lead to a complete traverse to all the possible combinations in the worst case, while GDP requires sorting for each value in the domain of each variable in the preprocessing phase, which makes its use prohibitively expensive. Additionally, GDP cannot use the learned knowledge from the combinations explored to dynamically prune the search space. In other words, GDP is actually an one-shot pruning method. Taking Fig. 2 for example, according to GDP, the local utilities of F_2 are sorted independently by each value of the domain. When computing the pruned range of value G, there are $\mathcal{V}_G = \{13, 9, 8, 7, 5, 5, 4, 1\}$, $p = \max(\mathcal{V}_G) = 13$ and

the base case $t = (20+17+10)-(9+17+8) = 13$. Hence, $p - t = 0$. Accordingly, GDP returns a fixed pruned range $[13, 1]$ for value G. Obviously, the pruned range contains the entire search space of value G, and cannot be reduced in the subsequent search process.

Under such circumstances, we propose a generic, fast and easy-to-use approach based on a branch-and-bound technique, called FDSP that can use the learned experience from the combinations explored to dynamically prune the search space.

FDSP

FDSP generally consists of two components: estimation to provide upper bounds and pruning to reduce the search space. To provide the optimal upper bound for a partial assignment, the estimation must return the upper bounds for both the local function and the incoming messages. FDSP computes the function estimations in the preprocessing phase, called Function Decomposing (FD), while the message estimations are (re)constructed once the messages have changed. Pruning is implemented by a procedure called State Pruning (SP) which is based on a branch and bound technique. That is, the algorithm does not expand any partial assignment whose upper bound is less than the known lower bound. FDSP can be easily applied to any belief propagation based incomplete algorithms to deal with DCOPs with n-ary constraints.

Function Decomposing serves in a preprocessing phase to compute the function estimation for each variable of a function-node $F_k(\mathbf{x_k})$. Given a partial assignment $PA|_{\mathbf{x_{k,1}}}^{\mathbf{x_{k,i}}}$ to variables $\{\mathbf{x_{k,w}} \in \mathbf{x_k} | 1 \leq w \leq i\}$, the upper bound of the local function is maximization of $F_k(\mathbf{x_k})$ over the remaining unassigned variables. That is

$$FunEst_{\mathbf{x_{k,i}}}(PA|_{\mathbf{x_{k,1}}}^{\mathbf{x_{k,i}}}) = \max_{z=\{\mathbf{x_{k,j}}|j>i\}} F_k(PA|_{\mathbf{x_{k,1}}}^{\mathbf{x_{k,i}}}, z) \quad (4)$$

Here, $FunEst_{\mathbf{x_{k,i}}}(\cdot)$ is the *uninformed* function estimation for the i-th variable $\mathbf{x_{k,i}}$ in $\mathbf{x_k}$, which provides optimistic upper bounds on the utilities of the subsequent search spaces of $PA|_{\mathbf{x_{k,1}}}^{\mathbf{x_{k,i}}}$. Macarthur et al. (2011) tried to compute the estimation by using brute force, which incurs exponential operations for each partial assignment. They also suggested that the domain characteristics can be used to compute the estimation, which has a limited generalization and cannot guarantee the tightness. In contrast, our proposed FD is an one-shot preprocessing procedure that uses dynamic programming to compute the estimation, which can significantly reduce the computation efforts. Specifically, the estimations are computed recursively according to Eq. (5).

$$FunEst_{\mathbf{x_{k,i}}} = \begin{cases} F_k(\mathbf{x_k}) & i = n \\ \max_{\mathbf{x_{k,i+1}}} FunEst_{\mathbf{x_{k,i+1}}} & otherwise \end{cases} \quad (5)$$

That is, the estimation for a variable is maximization of the one for the next variable. Particularly, the estimation for the last variable is the function itself. Note that, compared to the exponential overhead for each partial assignment in BnB-MS, our proposed FD only requires $O(d^{i+1})$ operations to

Algorithm 1: Function Decomposing (FD) for each function-node $F_k(\mathbf{x_k})$

Function Decompose $(F_k(\mathbf{x_k}))$
1   $n \leftarrow |\mathbf{x_k}|$
2   **for** $i \leftarrow n$ **to** 1 **do**
3   compute $FunEst_{\mathbf{x_{k,i}}}$ by Eq. (5)
4   **for** $j \leftarrow n$ **to** 2 **do**
5   **foreach** $v_{k,j} \in D_{\mathbf{x_{k,j}}} \wedge i < j$ **do**
6   compute $FunEst_{\mathbf{x_{k,i}}}^{\mathbf{x_{k,j}}=v_{k,j}}$ by Eq. (6)

Figure 3: The sketch of function decomposing

compute the function estimation for each variable $\mathbf{x_{k,i}}$ in the preprocessing phase.

In fact, the uninformed function estimation $FunEst_{\mathbf{x_{k,i}}}$ for $\mathbf{x_{k,i}}$ could provide a tighter upper bound if we know the assignment of a variable $\mathbf{x_{k,j}}$ such that $j > i$. In this way, we can compute a tight upper bound even if there are many unassigned variables between the last assigned variable and the target variable in a partial assignment. By considering all the possible assignments of each variable ordered after $\mathbf{x_{k,i}}$, we further reinforce the upper bound and propose the *informed* function estimations. Eq. (6) gives the formal definition to the informed function estimation for $\mathbf{x_{k,i}}$ in terms of $\mathbf{x_{k,j}}$ where $j > i$.

$$FunEst_{\mathbf{x_{k,i}}}^{\mathbf{x_{k,j}}=v_{k,j}} = \begin{cases} FunEst_{\mathbf{x_{k,j}}}(v_{k,j}) & i = j-1 \\ \max_{\mathbf{x_{k,i+1}}} FunEst_{\mathbf{x_{k,i+1}}}^{\mathbf{x_{k,j}}=v_{k,j}} & otherwise \end{cases} \quad (6)$$

Similar to the uninformed function estimations, the informed function estimations are computed in a recursive fashion by maximizing the estimation for the next variable. And the estimation for the last variable before $\mathbf{x_{k,j}}$ is the corresponding uninformed function estimation with a given assignment $\mathbf{x_{k,j}} = v_{k,j}$.

Fig. 3 gives the sketch of FD. The procedure begins with computing the uninformed function estimation for each variable in F_k according to Eq. (5) from the last one to the first one (line 1-3). Then, for every possible assignment of each variable, we compute the informed function estimation for each variable whose index is smaller than the current variable according to Eq. (6) (line 4-6).

Taking Fig. 2 for example, we can compute the uninformed function estimations for variable x_1, x_2, x_3 and x_4 as follows:

$FunEst_{x_4} = F_2, FunEst_{x_3} = \max_{x_4} FunEst_{x_4}$
$FunEst_{x_2} = \max_{x_3} FunEst_{x_3}$
$FunEst_{x_1} = \max_{x_2} FunEst_{x_2}$

These estimations can provide the upper bounds for the partial assignments with respect to their variables. Besides, the informed function estimations in terms of $x_4 = R$ are computed as follows:

$FunEst_{x_3}^{x_4=R} = FunEst_{x_4}(x_4 = R)$
$FunEst_{x_2}^{x_4=R} = \max_{x_3} FunEst_{x_3}^{x_4=R}$
$FunEst_{x_1}^{x_4=R} = \max_{x_2} FunEst_{x_2}^{x_4=R}$

State Pruning is geared towards speeding up the computation of the messages from function-nodes to variable-nodes by branch and bound. That is, when the upper bound

Algorithm 2: State Pruning(SP) for each function-node $F_k(\mathbf{x_k})$

Function FDSP $(F_k(\mathbf{x_k}),\mathbf{x_{k,t}},\mathcal{M}_{\mathbf{x_k}\backslash\mathbf{x_{k,t}}})$
7   $n \leftarrow |\mathbf{x_k}|$
8   $Result \leftarrow \emptyset$
9   $lb \leftarrow -\infty$
10   $msgChanged \leftarrow false$
11   **for** $i \leftarrow n$ **to** 1 **do**
12   **if** $\mathcal{M}_{\mathbf{x_{k,i}}}$ *has changed* **then**
13       $msgChanged \leftarrow true$
14   **if** $msgChanged = true$ **then**
15       $\forall \mathbf{x_{k,i}} \in \mathbf{x_k}\backslash\mathbf{x_{k,t}}$, compute $MsgEst_{\mathbf{x_{k,i}}}$ by Equation 7
16   $start \leftarrow 1$
17   **if** $\mathbf{x_{k,start}} = \mathbf{x_{k,t}}$ **then**
18     $start \leftarrow 2$
19   **foreach** $v_{k,t} \in D_{\mathbf{x_{k,t}}}$ **do**
20     $Assign_{\mathbf{x_{k,t}}} \leftarrow v_{k,t}$
21     $util^* \leftarrow$ FDSPRec $(F_k(\mathbf{x_k}),\mathbf{x_{k,start}},\mathbf{x_{k,t}},$ $\mathcal{M}_{\mathbf{x_k}\backslash\mathbf{x_{k,t}}},0)$
22     $Result \leftarrow Result \cup \{(v_{k,t}, util^*)\}$
23   **return** $Result$

Function FDSPRec $(F_k(\mathbf{x_k}),\mathbf{x_{k,i}},\mathbf{x_{k,t}}, \mathcal{M}_{\mathbf{x_k}\backslash\mathbf{x_{k,t}}}, msgUtil)$
24   $next \leftarrow i + 1$
25   **if** $\mathbf{x_{k,next}} = \mathbf{x_{k,t}}$ **then**
26     $next \leftarrow next + 1$
27   $util^* \leftarrow -\infty$
28   **foreach** $v_{k,i} \in D_{\mathbf{x_{k,i}}}$ **do**
29     $Assign_{\mathbf{x_{k,i}}} \leftarrow v_{k,i}$
30     $msgUtil_{\mathbf{x_{k,i}}} = msgUtil + \mathcal{M}_{\mathbf{x_{k,i}}}(v_{k,i})$
31   compute $ub_{\mathbf{x_{k,i}}}$ by Eq. (8)
32   **if** $ub_{\mathbf{x_{k,i}}} > lb$ **then**
33   **if** $\mathbf{x_{k,next}} \neq null$ **then**
34         $retUtil \leftarrow$ FDSPRec $(F_k(\mathbf{x_k}),\mathbf{x_{k,next}},$ $\mathbf{x_{k,t}},\mathcal{M}_{\mathbf{x_k}\backslash\mathbf{x_{k,t}}},msgUtil_{\mathbf{x_{k,i}}})$
35         $util^* \leftarrow \max(util^*, retUtil)$
36   **else**
37         $retUtil \leftarrow F_k(Assign) + msgUtil_{\mathbf{x_{k,i}}}$
38         $util^* \leftarrow \max(util^*, retUtil)$
39         $lb \leftarrow \max(lb, util^*)$
40   **return** $util^*$

Figure 4: The sketch of state pruning

of a partial assignment is no greater than the lower bound, the search space corresponding to the partial assignment will be discarded. Fig. 4 gives the pseudo code of SP.

The algorithm begins with calculating the message estimation for each variable $\mathbf{x_{k,i}} \in \mathbf{x_k}\backslash\mathbf{x_{k,t}}$, which gives the maximal message utility with respect to all non-target variables after it given these variables unassigned, according to Eq. (7) (line 7-15).

$$MsgEst_{\mathbf{x_{k,i}}} = \sum_{j>i \wedge j \neq t} \max(\mathcal{M}_{\mathbf{x_{k,j}}}) \quad (7)$$

Here, $\mathbf{x_{k,t}}$ is the target variable and $MsgEst_{\mathbf{x_{k,i}}}$ denotes the upper bounds on the received messages from the variables after $\mathbf{x_{k,i}}$ except $\mathbf{x_{k,t}}$. In order to reduce the unnecessary computation, F_k recomputes the message estimations for each variable ordered before $\mathbf{x_{k,i}}$ only when the message from $\mathbf{x_{k,i}}$ changes. Besides, instead of directly computing message estimations according to Eq. (7), F_k further reduces the computation efforts by recursively backing up the maximal message utilities from the last non-target variable

to the first one. That is, the message estimation of a variable is the sum of the maximal message utility and the message estimation of the non-target variable next to it. Consider the function-node F_2 in Fig. 2(b). When we are computing the message for x_4, the message estimations for x_1, x_2 and x_3 are computed as follows:

$$MsgEst_{x_3} = 0$$
$$MsgEst_{x_2} = MsgEst_{x_3} + \max \mathcal{M}_{x_3} = 10$$
$$MsgEst_{x_1} = MsgEst_{x_2} + \max \mathcal{M}_{x_2} = 10 + 17 = 27$$

Then, F_k computes the maximum utility $util^*$ of each assignment of the target variable $\mathbf{x_{k,t}}$ in $D_{\mathbf{x_{k,t}}}$ (line 16-22). Specifically, F_k assigns assignment $v_{k,t}$ to $\mathbf{x_{k,t}}$ according to the order of values in $D_{\mathbf{x_{k,t}}}$ (line 19). Thus, the current partial joint state $Assign = \{\emptyset, \ldots, v_{k,t}, \ldots, \emptyset\}$, where $\emptyset$ represents an unassigned variable (line 20). After that, **FD-SPRec** is called for the variable $\mathbf{x_{k,start}}$ which is the first unassigned variable to recursively expand the search space (line 21). Finally, F_k stores $util^*$ to $Result$ when $util^*$ for the current assignment $v_{k,t}$ is returned (line 22). The procedure (line 20 - 22) repeats until all the assignments of $\mathbf{x_{k,t}}$ have been visited.

In **FDSPRec**, F_k first finds $\mathbf{x_{k,next}}$ that is the unassigned variable next to $\mathbf{x_{k,i}}$ (line 24-26). Note that $\mathbf{x_{k,t}}$ is an assigned variable. Then, F_k decides to expand the search space or update the maximum utility and the lower bound (line 28-39). In more detail, F_k expands the search space by appending the assignment $v_{k,i}$ of $\mathbf{x_{k,i}}$ to the partial joint state (line 29). And then, F_k computes the utilities contributed (i.e., $msgUtil_{\mathbf{x_{k,i}}}$) by the incoming messages with respect to the current $Assign$ by summing the accumulated $msgUtil$ with the entry in terms of $\mathcal{M}_{\mathbf{x_{k,i}}}$ and assignment $v_{k,i}$ (line 30). Then, F_k computes the current upper bound $ub_{\mathbf{x_{k,i}}}$ according to Eq. (8) (line 31).

$$ub_{\mathbf{x_{k,i}}} = \begin{cases} msgUtil_{\mathbf{x_{k,i}}} + MsgEst_{\mathbf{x_{k,i}}} + FunEst_{\mathbf{x_{k,i}}}\left(Assign|_{\mathbf{x_{k,1}}}^{\mathbf{x_{k,i}}}\right) & i > t \\ msgUtil_{\mathbf{x_{k,i}}} + MsgEst_{\mathbf{x_{k,i}}} + FunEst_{\mathbf{x_{k,i}}}^{\mathbf{x_{k,t}}=v_{k,t}}\left(Assign|_{\mathbf{x_{k,1}}}^{\mathbf{x_{k,i}}}\right) & i < t \end{cases} \tag{8}$$

Specifically, if $i > t$(i.e., $\mathbf{x_{k,i}}$ is after $\mathbf{x_{k,t}}$), which means the assignments of all variables before $\mathbf{x_{k,t}}$ have been given, the current upper bound of the local function is provided by the uninformed function estimation of variable $\mathbf{x_{k,i}}$. Otherwise, the upper bound is computed by the informed function estimation. In other words, the informed function estimation is used to compute a tight upper bound whenever it is applicable.

Next, F_k decides whether to expand the search space according to the lower bound (line 32-39). If the upper bound $ub_{\mathbf{x_{k,i}}}$ is greater than the current lower bound lb and $\mathbf{x_{k,i}}$ is not the last variable, the algorithm proceeds by calling the recursive function **FDSPRec** to expand the search space (line 33-35). Otherwise, the search space corresponding to $Assign$ can be discarded. If $\mathbf{x_{k,i}}$ is the last non-target variable, i.e., the search space has been fully expanded, F_k computes the current utility $retUtil$ of the complete assignment by adding the local utility $F_k(Assign)$ and $msgUtil_{\mathbf{x_{k,i}}}$. Then, F_k updates the maximum utility $util^*$ and the lower bound lb (line 36-39). Finally, when all the assignments of $\mathbf{x_{k,i}}$ have been visited, the algorithm returns $util^*$ (line 40).
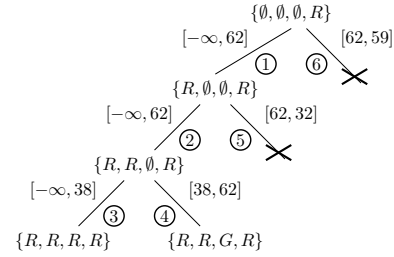

Figure 5: Calculating $R_{F_2 \to x_4}(x_4 = R)$ using FDSP

Fig. 5 shows an example for calculating the message from function F_2 to variable x_4 when $x_4 = R$ (i.e., $R_{F_2 \to x_4}(x_4 = R)$) in Fig. 2, where the numbers with circles represent the trace of SP. Since x_4 is fixed to assignment R, F_2 needs to compute the maximum utility $util^*$ by extending the partial assignment $Assign = \{\emptyset, \emptyset, \emptyset, R\}$. Firstly, F_2 visits the first assignment R of x_1 and computes $ub_{x_1} = 9 + 27 + 26 = 62$ by Eq. (8). Then, it expands $Assign = \{R, \emptyset, \emptyset, R\}$ by visiting the first assignment R of x_2 since $ub_{x_1} > lb(= -\infty)$. Similarly, F_2 expands $Assign = \{R, R, \emptyset, R\}$. At this point, since $ub_{x_3} > lb$ and $Assign$ is fully assigned, F_2 computes the utility $retUtil$ of $Assign = \{R, R, R, R\}$: $retUtil = F_2(Assign) + msgUtil_{x_3} = 4 + (9 + 17 + 8) = 38$, and updates $util^* = 38$ and $lb = 38$. Next, F_2 visits the next assignment G of x_3. Similarly, F_2 computes the current upper bound $ub_{x_3} = 62$ and the current utility $retUtil = 62$ corresponding to $Assign = \{R, R, G, R\}$, and updates lb and $util^*$. After that, F_2 visits the second assignment G of x_2 since all assignments of x_3 have been exhausted. And, F_2 computes $ub_{x_2} = 32$ which is less than lb, so $Assign = \{R, G, \emptyset, R\}$ is discarded. Similarly, $Assign = \{G, \emptyset, \emptyset, R\}$ is also discarded. Finally, F_2 finds the maximum utility $util^* = 62$, i.e., $R_{F_2 \to x_4}(x_4 = R) = 62$.

As seen from the example, FDSP can prune at least 75% of the search space during computing the message from the function-node F_2 to variable-node x_4, where $d = 2$ and $n = 4$.

Theoretical Analysis

In this section, we will theoretically prove that FDSP can speed up belief propagation based incomplete algorithms without an effect on solution quality, i.e., FDSP can provide monotonically non-increasing upper bounds and never prunes the optimal assignment with the maximum utility $util^*$.

Lemma 1. *For a function-node $F_k(\mathbf{x_k})$ and a given partial assignment PA with $(\mathbf{x_{k,t}} = v_{k,t})$ in which $(\mathbf{x_{k,i}} = v_{k,i})$ is the last non-target entry, the upper bound of any direct subsequent partial assignment $PA' = PA \cup (\mathbf{x_{k,j}} = v_{k,j})$ is at least as low as the one of PA, where $\mathbf{x_{k,j}}$ is the variable next to $\mathbf{x_{k,i}}$ such that $j \neq t$.*

Proof. Recall that the upper bound of a given partial assignment is computed according to either the uninformed function estimation or the informed function estimation, depending on the index of the target variable. Thus, three cases

need to be discussed: 1) all the upper bounds are computed according to the uninformed function estimations; 2) the upper bound of PA is computed according to the uninformed function estimation, while the one of PA' is computed according to the informed function estimation; 3) all the upper bounds are computed according to the informed function estimations. Here, we only give the prove for case 2) (i.e., $i + 1 = t, t + 1 = j$) due to the limited space. Similar analysis can be applied to case 1) and 3).

$$
\begin{aligned}
ub_{\mathbf{x_{k,i}}}(PA) &= FunEst^{\mathbf{x_{k,t}}=v_{k,t}}_{\mathbf{x_{k,i}}}(PA) + msgUtil_{\mathbf{x_{k,i}}} + MsgEst_{\mathbf{x_{k,i}}} \\
&= FunEst_{\mathbf{x_{k,i}}}(PA) + \sum_{l \leq i} \mathcal{M}_{\mathbf{x_{k,i}}}(v_{k,l}) + \sum_{l > i \wedge l \neq t} \max(\mathcal{M}_{\mathbf{x_{k,i}}}) \\
&\geq FunEst_{\mathbf{x_{k,i}}}(PA') + \sum_{l \leq i} \mathcal{M}_{\mathbf{x_{k,i}}}(v_{k,l}) + \sum_{l > i \wedge l \neq t} \max(\mathcal{M}_{\mathbf{x_{k,i}}}) \\
&\geq FunEst_{\mathbf{x_{k,i}}}(PA') + \sum_{l \leq j \wedge l \neq t} \mathcal{M}_{\mathbf{x_{k,i}}}(v_{k,l}) + \sum_{l > j} \max(\mathcal{M}_{\mathbf{x_{k,i}}}) \\
&= ub_{\mathbf{x_{k,i}}}(PA')
\end{aligned}
$$

Here, the second step holds since $i = t - 1$. Thus, according to Eq. (6) we have $FunEst^{\mathbf{x_{k,t}}=v_{k,t}}_{\mathbf{x_{k,t}}}(PA) = FunEst_{\mathbf{x_{k,t}}}(PA)$. Besides, the third step and the fourth step hold since $FunEst_{\mathbf{x_{k,t}}}(PA) = \max_{\mathbf{x_{k,j}}} FunEst_{\mathbf{x_{k,j}}}(PA, \mathbf{x_{k,j}}) \geq FunEst_{\mathbf{x_{k,j}}}(PA')$(Eq. (5)) and $\max(\mathcal{M}_{\mathbf{x_{k,j}}}) \geq \mathcal{M}_{\mathbf{x_{k,j}}}(v_{k,j})$, respectively.

Thus the lemma is proved. $\qquad\square$

Theorem 1. *FDSP does not affect the optimality of Eq. (2).*

Proof. Prove by contradiction. For a function-node $F_k(\mathbf{x_k})$, assume that the optimal assignment of Eq. (2) is $Assign^*$, and the corresponding utility value is $val(Assign^*)$. Assume that FDSP has missed that assignment. Thus, there must exist a partial assignment $PA \subset Assign^*$ such that $ub(PA) < lb \leq val(Assign^*)$. According to Lemma 1, the upper bound is monotonically non-increasing, i.e., $ub(PA) \geq ub(Assign^*)$. Note that

$$
\begin{aligned}
ub_{\mathbf{x_{k,n}}}(Assign^*) &= FunEst_{\mathbf{x_{k,n}}}(Assign^*) + msgUtil_{\mathbf{x_{k,n}}} + MsgEst_{\mathbf{x_{k,n}}} \\
&= F_k(Assign^*) + \sum_{l \leq n \wedge l \neq t} \mathcal{M}_{\mathbf{x_{k,l}}}(v_{k,l}) \\
&= val(Assign^*)
\end{aligned}
$$

Here, $n = |\mathbf{x_k}|$. Thus, we have $ub(PA) \geq ub(Assign^*) = val(Assign^*)$, which is contradict to the assumption. Therefore, the upper bound of a partial assignment cannot be less than the value of any subsequent full assignment and the optimality is hereby guaranteed. $\qquad\square$

Complexity Analysis

Each variable $\mathbf{x_{k,i}}$ needs to compute and stores an uninformed function estimation and $d(n - i)$ informed function estimations in the preprocessing phase. Thus, the time and space of each variable require $O([1 + d(n - i)]d^{i+1})$ and $O([1 + d(n - i)]d^i)$, respectively, where the value of i becomes smaller as FD performs. Thus, FDSP in the preprocessing phase needs a small overhead.

Besides, since each function-node needs to explore the search space with respect to the target variable, the time complexity in the worst case is $O(d^n)$. However, with SP, only the small search space needs to be explored. Therefore, the overall overhead is small. For this point, our empirical evaluation also verifies that FDSP only requires little time to run.

Empirical Evaluation

We empirically evaluate the performances of FDSP and GDP which are both applied to Max-Sum on four configurations of n-ary random DCOPs. Since BnB-MS and BnB-FMS are not generic algorithms and G-FBP is inferior to GDP (Khan et al. 2018), we do not include them for comparison. The complexity of a n-ary DCOP can be quantified by the number of function-nodes, the average/maximal arity and the domain size (Kim and Lesser 2013; Khan et al. 2018). In addition to these parameters, we also find the number of variable-nodes can affect the complexity. Intuitively, given the number of function-nodes and the average arity per function-node, the graph density is actually determined by the number of function-nodes. Therefore, we introduce a new parameter called variable tightness (denoted as var_T) to depict the complexity from another perspective, which is defined as follows.

$$
\text{var\_T} = 1 - \frac{\text{number of variable-nodes}}{\text{total number of arities}}
$$

It can be concluded that given the function-node number and total arity number, the number of variable-nodes decreases as var_T increases, which will generate a denser and more complex problem since each variable-node has to connect more function-nodes.

For each configuration other than the first one, we generate sparse factor graphs and dense factor graphs by randomly selecting var_T from [0.1, 0.5] and (0.5, 0.9], respectively. In the first configuration, we set the number of function-nodes to 100 and the minimal arity to 2, and uniformly select the costs, the domain size and the maximal arity from [1, 100], [2, 10] and [2, 7], respectively. And, var_T varies from 0.1 to 0.9. In the second one, we vary the maximal arity from 2 to 7. In the third configuration, we vary the number of function-nodes from 10 to 100. In the last one, we set the number of function-nodes to 50 and vary the domain size from 2 to 7. Also, we benchmark Max-Sum_ADVP+FDSP to demonstrate the generalization of FDSP. To guarantee Max-Sum_ADVP to converge, we alternate its directions every 100 iterations. All the omitted parameters except var_T in each configuration are the same as the ones in the first configuration. For each of the setting, we generate 25 random instances and the results are averaged over all instances. The algorithms terminate after 200 iterations for each instance.

Fig. 6 gives the comparison under different var_T. It can be observed that FDSP clearly outperforms GDP under different var_T, and the gap is widen as var_T grows. Concretely, FDSP can prune at least 97% of the search space while GDP only prunes at most 87% of the search space when computing Eq. (2). Moreover, FDSP performs similarly as var_T grows, which indicates FDSP is less sensitive to the complexity of problems. On the other hand, the performance of GDP decreases as var_T increases, and GDP per-
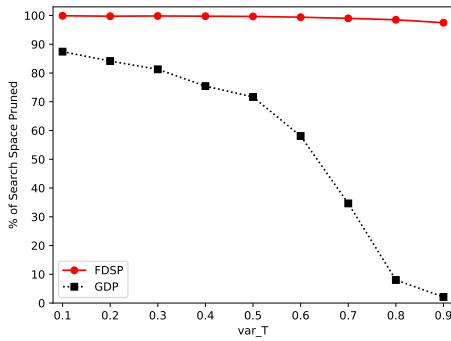
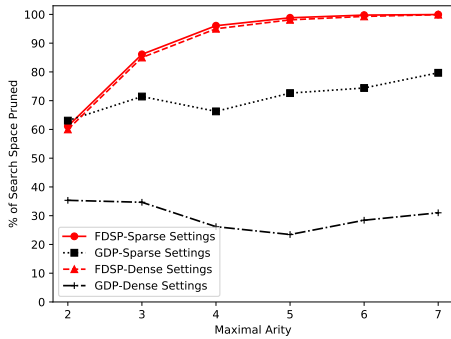Figure 6: Performance comparison on different var_T

Figure 8: Performance comparison on different function-node numbers

Figure 7: Performance comparison on different arities

Figure 9: Runtime on different domain sizes

forms poorly when solving the problems with var_T$>= 0.5$. This is because the sum of the difference between the maximal value and the value corresponding to the maximal local utility in each message will increase when the graph density increases as var_T grows. As a result, GDP provides a large pruned range so as to prune only a small proportion of the search space.

Fig. 7 shows the performance comparison on different maximal arities. It can be concluded that FDSP outperforms GDP in both sparse and dense factor graphs, especially in dense factor graphs. FDSP prunes around 60%-99% of the search space in both sparse and dense factor graphs, while GDP can only prune at most 80% and 36%, respectively. That is because FDSP provides tighter bounds to make Max-Sum explore fewer combinations.

Fig. 8 presents the results under different number of function-nodes. Similar to the first configuration, FDSP prunes at least 97% of the search space in both sparse and dense factor graphs, while GDP can only prune at most 88% and 55% of the search space in sparse and dense factor graphs, respectively. This is because GDP is an one-shot pruning procedure and cannot use the learned experience from the assignment combinations explored to dynamically prune the search space.

Fig. 9 gives the runtime under different domain sizes. It can be seen that our FDSP exhibits great superiority over GDP and Max-Sum_ADVP when solving the problems with large domain sizes, which indicates that FDSP can scale up well and only requires few computation efforts. GDP would
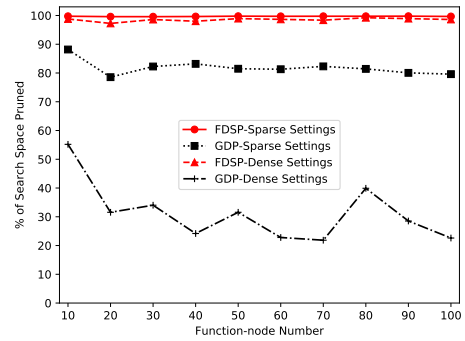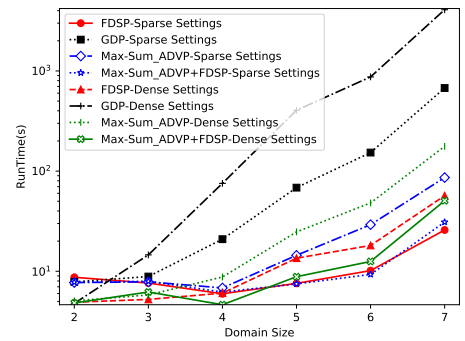
perform even worse in practice since the runtime presented in Fig. 9 actually does not take sorting, which is quite expensive when the domain size is large, into consideration. Besides, one can easily observe that Max-Sum_ADVP+FDSP is superior to Max-Sum_ADVP when solving the problems with large domain sizes in sparse and dense factor graphs, which indicates that FDSP can also effectively accelerate the variants of Max-Sum.

Conclusion

In this paper, we propose FDSP, a generic, fast and easy-to-use method based branch and bound, which significantly accelerates belief propagation based incomplete DCOP algorithms. Specifically, we first propose function decomposing (FD) to effectively compute the function estimation, which dramatically reduces the overheads in computing an upper bound of a partial assignment. Then, we further present state pruning (SP) based on branch and bound to reduce the search space. Besides, we theoretically prove that our bounds are monotonically non-increasing during the search process and FDSP never prunes the assignment with the maximum utility. Our experimental results clearly show that FDSP can prune around 97%-99% of the search space and only requires little time, especially for the large and complex problems.

References

Arshad, M., and Silaghi, M. C. 2004. Distributed simulated annealing. *Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems* 112.

Cerquides, J.; Farinelli, A.; Meseguer, P.; and Ramchurn, S. D. 2014. A tutorial on optimization for multi-agent systems. *Computer Journal* 57(6):799–824.

Chen, Z.; Deng, Y.; and Wu, T. 2017. An iterative refined max-sum_ad algorithm via single-side value propagation and local search. In *Proc. of the 16th Conference on AAMAS*, 195–202.

Enembreck, F., and Barthès, J. P. A. 2012. Distributed constraint optimization with mulbs: A case study on collaborative meeting scheduling. *Journal of Network & Computer Applications* 35(1):164–175.

Farinelli, A.; Rogers, A.; Petcu, A.; and Jennings, N. R. 2008. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proc. of the 7th Conference on AAMAS*, 639–646.

Farinelli, A.; Rogers, A.; and Jennings, N. R. 2014. Agent-based decentralised coordination for sensor networks using the max-sum algorithm. *Autonomous Agents and Multi-Agent Systems* 28(3):337–380.

Fioretto, F.; Yeoh, W.; Pontelli, E.; Ma, Y.; and Ranade, S. J. 2017. A distributed constraint optimization (dcop) approach to the economic dispatch with demand response. In *Proc. of the 16th Conference on AAMAS*, 999–1007.

Gershman, A.; Meisels, A.; and Zivan, R. 2009. Asynchronous forward bounding for distributed cops. *Journal of Artificial Intelligence Research* 34:61–88.

Hirayama, K., and Yokoo, M. 1997. Distributed partial constraint satisfaction problem. In *International Conference on Principles and Practice of Constraint Programming*, 222–236. Springer.

Khan, M.; Tran-Thanh, L.; Jennings, N.; et al. 2018. A generic domain pruning technique for gdl-based dcop algorithms in cooperative multi-agent systems. In *Proc. of the 17th Conference on AAMAS*, 1595–1603.

Kim, Y., and Lesser, V. 2013. Improved max-sum algorithm for dcop with n-ary constraints. In *International Conference on Autonomous Agents and Multi-Agent Systems*, 191–198.

Macarthur, K. S.; Stranders, R.; Ramchurn, S. D.; and Jennings, N. R. 2011. A distributed anytime algorithm for dynamic task allocation in multi-agent systems. In *AAAI Conference on Artificial Intelligence*, 701–706.

Maheswaran, R. T.; Pearce, J. P.; and Tambe, M. 2004. Distributed algorithms for dcop: A graphical-game-based approach. In *Proceedings of the International Conference on Parallel and Distributed Computing Systems (PDCS)*, 432–439.

Modi, P. J.; Shen, W.-M.; Tambe, M.; and Yokoo, M. 2005. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence* 161(1-2):149–180.

Nguyen, D. T.; Yeoh, W.; and Lau, H. C. 2013. Distributed gibbs: A memory-bounded sampling-based dcop algorithm. In *Proc. of the 12th Conference on AAMAS*, 167–174.

Okamoto, S.; Zivan, R.; and Nahon, A. 2016. Distributed breakout: beyond satisfaction. In *Proc. of the 25th IJCAI*, 447–453.

Ottens, B.; Dimitrakakis, C.; and Faltings, B. 2012. Duct: An upper confidence bound approach to distributed constraint optimization problems. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 528–534.

Petcu, A., and Faltings, B. 2005. A scalable method for multiagent constraint optimization. In *Proc. of the 19th IJCAI*, 266–271.

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

Stranders, R.; Farinelli, A.; Rogers, A.; and Jennings, N. R. 2009. Decentralised coordination of mobile sensors using the max-sum algorithm. In *IJCAI 2009, Proceedings of the International Joint Conference on Artificial Intelligence, Pasadena, California, Usa, July*, 299–304.

Vinyals, M.; Rodriguez-Aguilar, J. A.; and Cerquides, J. 2009. Generalizing dpop: Action-gdl, a new complete algorithm for dcops. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, 1239–1240. International Foundation for Autonomous Agents and Multiagent Systems.

Yeoh, W.; Felner, A.; and Koenig, S. 2008. Bnb-adopt: an asynchronous branch-and-bound dcop algorithm. In *International Joint Conference on Autonomous Agents and Multiagent Systems*, 591–598.

Zhang, W.; Wang, G.; Xing, Z.; and 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., and Peled, H. 2012. 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-Volume 1*, 265–272. International Foundation for Autonomous Agents and Multiagent Systems.