# An Iterative Refined Max-sum_AD Algorithm via Single-side Value Propagation and Local Search

Ziyu Chen
College of Computer Science
Chongqing University
Chongqing, China
chenziyu@cqu.edu.cn

Yanchen Deng
College of Computer Science
Chongqing University
Chongqing, China
dyc941126@126.com

Tengfei Wu
College of Computer Science
Chongqing University
Chongqing, China
wutengfei0404@163.com

## ABSTRACT

Max-sum_ADVP is a message-passing algorithm for solving Distributed Constraint Optimization Problems (DCOPs) able to obtain a convergent solution in a cyclic factor graph. Nevertheless, the solution quality is closely related to the timing for starting value propagation in Max-sum_ADVP. In other words, low-quality initial assignments will lead to a poor result. In this paper, we illustrate that value propagation can eliminate the inconsistent contexts in Max-sum_AD and break ties among utilities, but it also restricts the exploration brought by Max-sum. For balancing between the exploration and the accuracy, we propose a new iterative refined Max-sum_AD algorithm with single-side value propagation, called Max-sum_ADSSVP. It performs two phases in every two convergences, one phase which enables the exploration to find high-quality initial assignments and the other phase which enables value propagation to guarantee solution quality. Max-sum_ADSSVP tackles the timing selection problem by iteratively refining initial assignments in every exploration phase. Besides, local search is introduced after the value propagation phase to speed up the convergence process. Our empirical evaluation indicates that our methods are independent of initial assignments and less likely to get stuck in local optima.

## Keywords

DCOP;Incomplete Algorithm;Max-sum;Value Propagation

## 1. INTRODUCTION

Distributed constraint optimization problems (DCOPs) [8] are a fundamental model for representing multi-agent systems (MAS) in which agents need to coordinate their decision-making to optimize a global objective. Due to their ability to capture essential MAS aspects, DCOPs have been successfully applied to various MAS applications such as sensor networks [23], meeting scheduling [21], power networks [16] and so on. A wide variety of algorithms have been investigated to solve DCOPs and are generally divided into two categories, i.e., complete and incomplete. Typical search-based complete algorithms include ADOPT [10] and its variants [22, 4, 7], ConFB [11], and so on. DPOP [13] and its variants [14, 15] are typical inference-based complete algorithms which use dynamic programming technique to solve DCOPs. Although they can find the optimal solution, complete algorithms exhibit an exponentially increasing coordination overhead, which prohibits them from scaling up to large real application problems. In contrast, incomplete algorithms require very little local computation and communication to find solutions at the cost of sacrificing optimality. For solving practical applications, considerable research efforts have been made to develop incomplete algorithms.

Local search algorithms are typical incomplete algorithms including DBA [5, 20], MGM [9], DSA [23], etc. In recent years, many mechanisms such as $k$-optimality [12] and anytime local search (ALS) framework [24] have been investigated in this field. In local search, agents keep exchanging self states (i.e., assignments or gains) with their neighbours, and then determine whether to replace their assignments based on the received states from their neighbours. The major difference among those algorithms lies on assignment replacement strategy. For example, agents in DSA decide to replace their assignments by making stochastic decisions every iteration, while agents who hold the maximal gain among their neighbours replace their assignments in MGM. However, local search algorithms often converge to poor-quality solutions since agents only communicate their preferred states based on current preferred states of their neighbours.

Max-sum [3] is another popular inference-based incomplete algorithm based on the Generalized Distributive Law [1]. Rather than exchanging self states with their neighbours in local search, agents in Max-sum propagate and accumulate utilities through the whole factor graph. Therefore, agents can obtain global utilities for each possible value assignment. Nevertheless, Max-sum only guarantees to converge in cycle-free factor graphs which are very rare in DCOPs. Bounded Max-sum (BMS) [17] was proposed to tackle this problem by removing dependencies, which have the least impact on the solution quality, between function-nodes and variable-nodes. Concretely, agents shrink those binary dependencies to unary functions by minimizing the dependencies. Consequently, the algorithm transforms a cyclic factor graph into a tree-structured problem, uses Max-sum to solve it, and provides a bound on the approximation of the optimal solution. However, a large approximation ratio in BMS reflects lack of confidence in the solution. Recently, Improved Bounded Max-sum (IBMS) [18] was proposed to provide a tighter approximation ratio. Instead of only solving a problem with minimized dependencies in BMS, IBMS also solves

**Appears in:** *Proc. of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), S. Das, E. Durfee, K. Larson, M. Winikoff (eds.), May 8–12, 2017, São Paulo, Brazil.*

the maximized one and selects the best result to calculate the approximation ratio. Besides, ED-IBMS [19] and AD-IBMS [19] improve IBMS in terms of binary dependencies decomposing.

Different than removing dependencies in Bounded Max-sum, Max-sum_AD [25] makes a factor graph acyclic by converting it to a direct acyclic graph (DAG) according to a predefined order and performing message-passing on it. That is, instead of sending messages to every neighbour in Max-sum, nodes in Max-sum_AD only send messages to whom ordered after them. When the algorithm converges after a number of iterations, the order from which the direction of the DAG is derived is reversed. It has been proved that Max-sum_AD can converge after $l$ iterations, where $l$ is no less than the length of the longest path in a cyclic factor graph. However, ties among utilities and inconsistent contexts make Max-sum_AD traverse states with low quality. Max-sum_ADVP [25] was proposed to overcome the pathologies using a two-phase value propagation, where variable-nodes propagate both utilities and their values, and function-nodes produce messages by considering the value assignments received from variable-nodes. However, the timing for starting value propagation is a major concern in Max-sum_ADVP. If value propagation starts too early, variable-nodes don't have enough knowledge to produce high-quality assignments. As a result, those low-quality assignments will be propagated through the whole factor graph and lead to a poor result.

The main contributions of this paper are as follows:

- We analyze how value propagation affects Max-sum. We find that value propagation can restrict the exploration brought by Max-sum. Max-sum_ADVP will block utilities propagation and accumulation, which are essential in the GDL, and can only make use of local benefits to make decisions.
- We propose Max-sum_ADSSVP with a new value propagation schema to solve the problem of timing selection. It performs two phases in every two convergences, one phase which enables value propagation to guarantee solution quality and the other phase which enables the exploration to find potential optima. Our experimental results show that the value propagation phase can help the exploration phase to reduce the inconsistent contexts and the exploration phase with less inconsistent contexts will provide higher-quality assignments for the value propagation phase.
- We speed up the convergence process by introducing local search after the value propagation phase. Max-sum_ADSSVP requires more iterations to converge than Max-sum_ADVP due to the existence of inconsistent contexts in every exploration phase. Local search is performed after the value propagation phase to provide higher-quality solution for the exploration phase so as to eliminate more inconsistent contexts. Our experimental results also show that Max-sum_ADSSVP with local search can effectively suppress the cost fluctuation in the exploration phase.

The rest of the paper is organized as follow. The formal definition of DCOPs is presented in Section 2. Section 3 presents Max-sum, Max-sum_AD and Max-sum_ADVP. In Section 4, we analyze the limitation of Max-sum_ADVP and present the detail about our proposed Max-sum_ADSSVP.

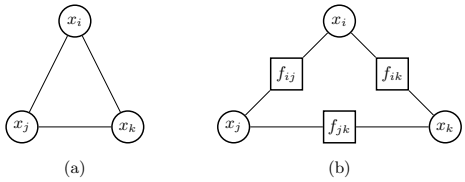


**Figure 1: DCOP and factor graph**

The evaluation of our methods in comparison with Max-sum and its variants is given in Section 5. Section 6 concludes the paper and gives our future research work.

# 2. DISTRIBUTED CONSTRAINT OPTIMIZATION PROBLEMS

A DCOP is defined as a tuple $\langle A, X, D, F\rangle$ such that:

- $A = \{a_1, \ldots, a_q\}$ is a set of agents.
- $X = \{x_1, \ldots, x_n\}$ is a set of variables. Each variable $x_i$ is controlled by an agent.
- $D = \{D_1, \ldots, D_n\}$ is a set of finite variable domains, variable $x_i$ taking a value in $D_i$.
- $F = \{f_1, \ldots, f_m\}$ is a set of constraints, where a constraint $f_i$ is any function with the scope $(x_{i_1}, \ldots x_{i_k})$, $f_i : D_{i_1} \times \cdots \times D_{i_k} \rightarrow \mathbb{R}^+$ which denotes how much utility is assigned to each possible combination of values of the involved variables.

Without loss of generality, a solution to a DCOP is an assignment to all variables that maximizes the total utilities, which is the sum of all constraints:

$$X^* = \arg\max_{X \in D} \sum_{f_i \in F} f_i$$

To facilitate understanding, we assume that each agent has a single variable and constraints are binary. Here, the term "agent" and "variable" can be used interchangeably. A binary constraint is a constraint involving exactly two variables defined as $f_{ij} : D_i \times D_j \rightarrow \mathbb{R}^+$. Consequently, a solution to a DCOP can be formalized as

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

A DCOP can be visualized by a constraint graph where the nodes represent agents and the edges represent constraints. Figure 1(a) gives a constraint graph of a DCOP which comprises of three agents and three constraints.

# 3. ALGORITHM PRELIMINARIES

## 3.1 Standard Max-sum

Max-sum is a message-passing inference-based algorithm operating on factor graphs. A factor graph [6] corresponds to a bipartite graph comprising two types of nodes: variable-nodes and function-nodes. Function-nodes which represent constraints in the original DCOP are connected to variable-nodes they depend on. Similarly, variable-nodes which represent variables in the original DCOP are connected to all function-nodes they are involved in.

**Algorithm 1: Max-sum_AD(node $n$)**

```
current_order ← select an order on all nodes in the factor graph;
N_n ← all of n's neighbouring nodes;
while no termination condition is met do
    N_prev_n ← {n̂ ∈ N_n : n̂ is before n in current_order};
    N_follow_n ← N_n \ N_prev_n;
    for k iterations do
        collect messages from N_prev_n;
        foreach n' ∈ N_follow_n do
            if n is a variable-node then
                produce message m_{n'} using messages received from N_n \ {n'};
            end
            if n is a function-node then
                produce message m_{n'} using constraint and messages received from N_n \ {n'};
            end
            send m_{n'} to n';
        end
    end
    current_order ← reverse(current_order);
end
```

**Figure 2: Pseudo-code for Max-sum_AD**

When computing a message to a function-node $f$, variable-node $x_i$ sums up all the last messages received from neighbouring function-nodes except the target function-node $f$. When computing a message to a variable-node $x_j$, function-node projects out $x_j$ from the function that results by summing up the local function $f$ and the last messages received from neighbouring variable-nodes except the target variable-node $x_j$.

Figure 1(b) shows the factor graph deriving from Figure 1(a). The message sent from $x_i$ to $f_{ik}$ is defined by

$$q_{x_i \to f_{ik}}(x_i) = \alpha + \sum_{n \in N_i \backslash k} r_{f_{ni} \to x_i}(x_i)$$

where $N_i \backslash k$ is a set of neighbour indexes of $x_i$ except target node index $k$ (i.e., $N_i \backslash k = \{j\}$) and $r_{f_{ni} \to x_i}(x_i)$ is the message sent from function-node $f_{ni}$ to $x_i$. Besides, in order to avoid the entries in the messages uncontrollably growing in a cyclic factor graph, a scaler $\alpha$ such that

$$\sum_{x_i} q_{x_i \to f_{ik}}(x_i) = 0$$

is chosen to normalize utilities. The message sent from $f_{ij}$ to $x_j$ is defined by

$$r_{f_{ij} \to x_j}(x_j) = \max_{x_i}(f_{ij} + q_{x_i \to f_{ij}}(x_i))$$

where $f_{ij}$ is the local function that represents the constraint between $x_i$ and $x_j$.

When a variable-node is making decision, it accumulates all utilities it receives, and selects a value to maximize the total utilities. The procedure can be formalized by

$$x_i^* = \arg\max_{x_i} \sum_{n \in N_i} r_{f_{ni} \to x_i}(x_i)$$

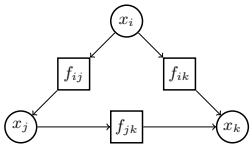


**Figure 3: A directed acyclic factor graph**

## 3.2 Max-sum_AD and Max-sum_ADVP

Max-sum_AD avoids cycles in a factor graph by performing message-passing on a DAG determined by a predefined order in the factor graph. Each node sends messages only to its neighbouring nodes that follow it in the selected order. After a linear number of iterations in which the algorithm is guaranteed to converge, the order is reversed and messages are sent in the opposite direction. The order is alternated following each convergence until termination. The pseudo-code of Max-sum_AD is shown in Figure 2.

It has been proved that Max-sum_AD can converge after $l$ iterations, where $l$ is no less than the length of the longest path in a cyclic factor graph.

Although Max-sum_AD can guarantee convergence in one direction, one variable-node may not make a high-quality decision when there are ties among utilities. Here, the utility ties refer to the phenomenon that several values correspond to the same utility. Besides, the inconsistent contexts can also lead to low quality solutions. Figure 3 gives an example of a DAG corresponding to Figure 1(b).

According to Max-sum_AD, the messages sent in 4 cycles are as follows[1]:

$$\begin{aligned} \text{Cycle 1:} & x_i \to f_{ik} : 0 \\ & x_i \to f_{ij} : 0 \end{aligned}$$

$$\begin{aligned} \text{Cycle 2:} & f_{ij} \to x_j : \max_{x_i} f_{ij} \\ & f_{ik} \to x_k : \max_{x_i} f_{ik} \end{aligned}$$

$$\text{Cycle 3:} x_j \to f_{jk} : \max_{x_i} f_{ij}$$

$$\text{Cycle 4:} f_{jk} \to x_k : \max_{x_j}(f_{jk} + \max_{x_i} f_{ij})$$

Eventually, variable-node $x_k$ uses $r_{f_{ik} \to x_k}$ and $r_{f_{jk} \to x_k}$ to select a value to maximize the total utilities. That is

$$\arg\max_{x_k} \max_{x_j}(f_{jk} + \max_{x_i} f_{ij} + \max_{x_i} f_{ik})$$

However, the optimal decision for $x_k$ is

$$\arg\max_{x_k}(\max_{x_i x_j}(f_{ij} + f_{jk} + f_{ik}))$$

which is equivalent to

$$\arg\max_{x_k}(\max_{x_j}(f_{jk} + \max_{x_i}(f_{ij} + f_{ik})))$$

Obviously, Max-sum_AD approximates $\max_{x_i}(f_{ik} + f_{ij})$ as $\max_{x_i} f_{ij} + \max_{x_i} f_{ik}$. In other words, it computes functions

[1]For sake of simplicity, we present the message-passing process in a sequential way. In fact, agents in Max-sum_AD perform concurrently. However, the messages from an agent keep changing until the agent receives all stable messages from its precursors. Thus, we present the messages when they converge.

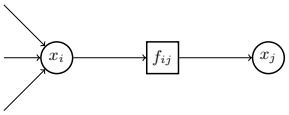


**Figure 4: Simple factor graph with two variable-nodes and a function-node**

in the different contexts rather than the identical context, which will lead to low quality solutions.

Max-sum_ADVP tackles those problems using value propagation. Specifically, variable-nodes enclose messages sent to function-nodes with the values they select. When function-nodes produce messages, they consider the received assignments. That is, instead of traversing domain of upstream variable-nodes to obtain the optimal utility for every target value, function-nodes in Max-sum_ADVP fix the values of upstream variable-nodes to the received assignments. As a result, function-nodes that have the same upstream variable-node now share the identical context. So, value propagation can eliminate the inconsistent contexts and break ties among utilities. However, the timing for starting value propagation becomes a major concern in value propagation.

# 4. ITERATIVE REFINED MAX-SUM_AD ALGORITHM

## 4.1 Motivation

Although it can greatly improve the solution quality of Max-sum_AD, value propagation restricts the exploration brought by Max-sum. Next, we prove that value propagation can block utilities propagation and accumulation, and will eventually make Max-sum_ADVP behave like a greedy local search algorithm.

LEMMA 1. *Function-nodes will block global utility propagation and propagate only local functions when value propagation is enabled.*

PROOF. Without loss of generality, let us focus on a part of a DAG which includes two variable-nodes and a function-node shown as Figure 4. Variable-node $x_i$ selects its value based on the utilities it receives. Assume that $x_i = k$ . According to Max-sum_ADVP, the message from $x_i$ to $f_{ij}$ is defined as

$$x_i \to f_{ij} : q_{x_i \to f_{ij}}(x_i) = \sum_{n \in N_i \setminus j} r_{f_{ni} \to x_i}(x_i), x_i = k \quad (1)$$

$f_{ij}$ produces the message sent to $x_j$ in terms of $x_i = k$. That is

$$f_{ij} \to x_j : r_{f_{ij} \to x_j}(x_j) = f_{ij}(k, x_j) + q_{x_i \to f_{ij}}(k) \quad (2)$$

Since it is a constant and has no influence on decision making of $x_j$, $q_{x_i \to f_{ij}}(k)$ can be removed safely. Consequently, formula(2) can be simplified as

$$f_{ij} \to x_j : r_{f_{ij} \to x_j}(x_j) = f_{ij}(k, x_j) \quad (3)$$

Obviously, formula(3) is equivalent to a local function with a given variable assignment. Besides, it can be concluded from formula(2) and formula(3) that value propagation blocks utility propagation by restricting a variable to a certain assignment, which degenerates global utility (i.e., $q_{x_i \to f_{ij}}(x_i)$ ) into a constant (i.e., $q_{x_i \to f_{ij}}(k)$ ). So, Lemma 1 is proved. □

The immediate corollary from Lemma 1 is that messages from function-nodes are all local functions after the first convergence when value propagation is enabled. At the same time, variable-nodes also fail to collect and forward global utility. In other words, value propagation restricts the exploration brought by Max-sum.

PROPOSITION 1. *Max-sum_ADVP shares the same decision-making strategy with greedy local search algorithms after the first convergence of value propagation.*

PROOF. Consider the factor graph shown in Figure 4. Assume that value propagation has been enabled and the algorithm has converged for the first time in the direction of right to left. According to Lemma 1, variable-node $x_i$ selects its value by

$$\arg\max_{x_i}( \sum_{n \in N_i \setminus j} f_{ni}(k_n^l, x_i) + f_{ij}(x_i, k_j^{l-1})) \quad (4)$$

where $k_n^l$ is the assignment of variable-node $x_n$ in the $l$-th convergence phase. Because $x_j$ is a downstream variable-node of $x_i$, $x_i$ makes decision based on the previous assignment of $x_j$. Thus, formula(4) indicates that $x_i$ produces the best response in terms of the given assignments from its neighbours. On the other hand, agents in greedy local search algorithms (e.g., DSA and MGM) also make optimal decisions by considering all neighbours' assignments. That is,

$$\arg\max_{x_i} \sum_{n \in N_i} f_{ni}(k_n, x_i) \quad (5)$$

It can be concluded from formula(4) and formula(5) that Max-sum_ADVP is equivalent to greedy local search algorithms in terms of decision-making strategy. So, Proposition 1 is proved. □

Proposition 1 also illustrates that agents in Max-sum_ADVP cannot make their decisions based on the global utilities and Max-sum_ADVP will eventually behave like a sequential local search algorithm.

## 4.2 Max-sum_ADSSVP

To solve the problem of value propagation, we propose a new iterative refined Max-sum_AD algorithm with single-side value propagation, called Max-sum_ADSSVP. It can keep the balance between the exploration brought by Max-sum and the accuracy brought by value propagation. Moreover, it can overcome the drawback of timing selection in Max-sum_ADVP.

For the sake of clarity, we use the term "forward direction" as initial message-passing order, and "backward direction" as reversed message-passing order. When value propagation is enabled, Max-sum_ADSSVP performs the following two phases in every two convergences.

- **Exploration Phase:** performing like standard Max-sum_AD to find higher-quality initial assignments for value propagation when the current message-passing order is forward direction.

- **Value Propagation Phase:** enabling value propagation to guarantee solution quality when the current message-passing order is backward direction.

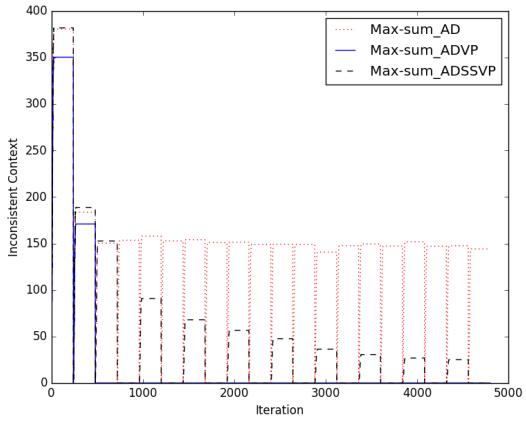


**Figure 5: Comparison of Max-sum_AD,Max-sum_ADVP,Max-sum_ADSSVP on inconsistent contexts**

The exploration phase and value propagation phase can collaborate with each other to iteratively refine the solution quality. Specifically, the value propagation phase can help the exploration phase to eliminate inconsistent contexts, meanwhile the exploration phase with less inconsistent contexts will provide higher-quality initial assignments for value propagation.

To demonstrate the above relationship between the two phases, the number of inconsistent contexts for Max-sum_AD, Max-sum_ADVP and Max-sum_ADSSVP is compared when applied to solve Random DCOPs where $n = 120$, $p = 0.05$ and domain size is 10. Here, $n$ represents the number of agents and $p$ refers to graph density. We average the experimental results over 25 DCOP instances with randomly generated constraints and randomly generated integer constraint costs in the range of 1 to 100. To make sure algorithms converge, the compared algorithms change their directions every 240 iterations and stop after 20 convergences. Moreover, value propagation starts after the second convergence for Max-sum_ADVP and Max-sum_ADSSVP to obtain the better performance.

The experimental results are shown as Figure 5. Obviously, the number of inconsistent contexts in Max-sum_ADVP decreases to zero after value propagation is enabled, which indicates that value propagation can eliminate inconsistent contexts. Since Max-sum_ADSSVP behaves the same as Max-sum_AD in the first three convergences, the number of the inconsistent contexts in Max-sum_ADSSVP is close to that of Max-sum_AD. When single-side value propagation is enabled, a remarkable decrease of inconsistent contexts appears in every exploration phase, which indicates that the value propagation phase can help the exploration phase to eliminate inconsistent contexts. In contrast, the number of inconsistent contexts fluctuates slightly in a high state in Max-sum_AD.

## 4.3 Max-sum_ADSSVP with Local Search

Although Max-sum_ADSSVP can obtain a balance between the exploration and the accuracy, it needs more iterations to converge due to the existence of inconsistent contexts in every exploration phase. We introduce local search after the value propagation phase to provide higher-quality solutions for the exploration phase to eliminate more inconsistent contexts. Specifically, the following two phases are performed after the value propagation phase:

**Algorithm 2: Max-sum_ADSSVP with local search(agent $i$, local search, $forward\_direction$,$k$,$l$)**

```
1  current_order ← forward_direction ;
2  backward_direction ← reverse(forward_direction) ;
3  while no termination condition is met do
4      for k iterations do
5          if current_order is forward_direction then
6              perform Max-sum_AD;
7          end
8          if current_order is backward_direction then
9              x_i ← current optimal decision;
10             perform Max-sum_ADVP using assignment
               x_i;
11         end
12     end
13     if current_order is backward_direction then
14         perform l iterations local search with initial
           assignment x_i;
15         x_i ← current optimal decision;
16         for k iterations do
17             perform Max-sum_ADVP using assignment
               x_i;
18         end
19     end
20     current_order ← reverse(current_order);
21 end
```

**Figure 6: Sketch of Max-sum_ADSSVP with local search**

- **Refining Phase:** performing short-range local search with the initial assignments generated by the value propagation phase to produce a higher-quality solution.
- **Modification Phase:** performing value propagation with the assignments generated by the refining phase to apply new assignments into a factor graph.

The sketch of the algorithm is presented in Figure 6. The algorithm has three parameters, i.e., local search algorithm, iterations for Max-sum_AD and Max-sum_ADVP, and iterations for local search. An agent performs the exploration phase (i.e., Max-sum_AD schema) when current message-passing order is forward direction (line 5 - 7), while agent performs the value propagation phase when current message-passing order is backward direction (line 8 - 11). If current message-passing order is backward direction, the refining phase and modification phase are trigged consecutively after the value propagation. Each agent takes its assignment generated by value propagation as its initial assignment and performs $l$-iteration local search (line 14). After that, each agent performs $k$-iteration value propagation with the new assignment generated by the refining phase (line 15 - 18).

It is worth mentioning that only Max-sum_ADSSVP can be (or needs to be) enhanced by local search. The aim of performing local search after the value propagation phase is to eliminate more inconsistent contexts. Since Max-sum_ADVP

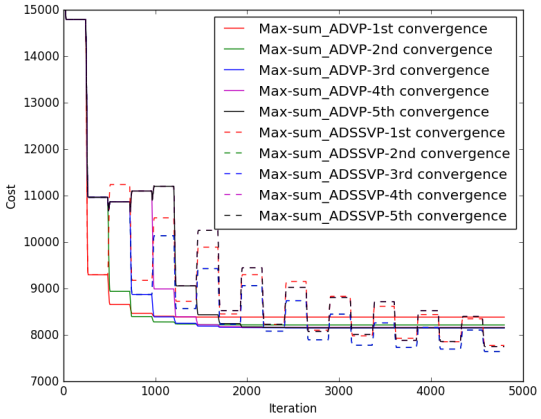


**Figure 7: Comparison of Max-sum_ADVP and Max-sum_ADSSVP on different value propagation timings**

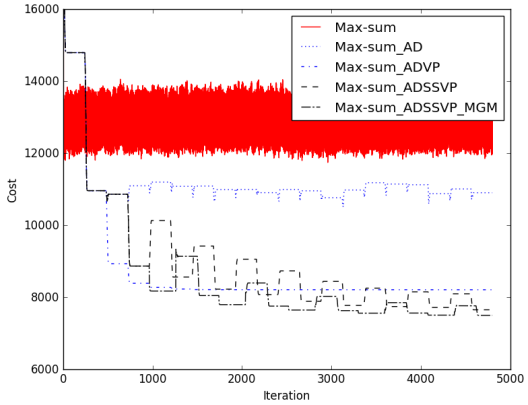


**Figure 8: Comparison of Max-sum variants on sparse configuration of random DCOPs**

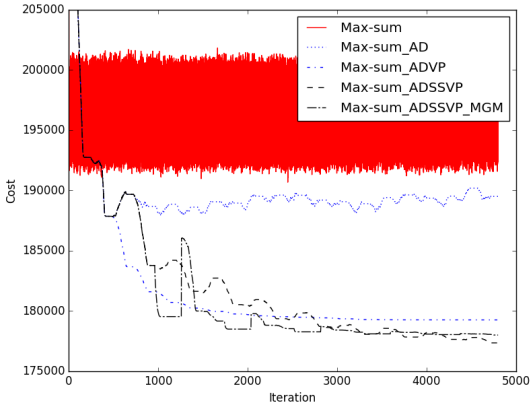


**Figure 9: Comparison of Max-sum variants on dense configuration of random DCOPs**

doesn't have any inconsistent contexts after value propagation is enabled, it is not necessary to be enhanced by local search. Max-sum and Max-sum_AD suffer from lots of inconsistent contexts, which leads them to traverse states with low quality. However, they don't have the value propagation phase, which indicates that the assignments of agents cannot affect the optimization process. As a result, Max-sum and Max-sum_AD cannot utilize the assignments produced by local search. In other words, Max-sum and Max-sum_AD cannot be enhanced by local search.

The extra overheads of our proposed methods mainly lie on the computation. Specifically, a function-node in value propagation phases requires $O(d)$ operations to produce a message by fixing the value of upstream variable-node, while a function-node in exploration phases requires $O(d^2)$ operations to produce a message by traversing the domain of upstream variable-node. Here, $d = \max_{D_i \in D} |D_i|$. Thus, compared to Max-sum_ADVP, Max-sum_ADSSVP will incur $O(d^2)$ operations when producing messages to variable-nodes in every exploration phase. Further, Max-sum_ADSSVP with local search has two additional phases, i.e., refining phase and modification phase. Since it is very short, the refining phase usually incurs a minor overhead. Besides, given the maximum iterations, the number of exploration phases is much reduced due to existence of the modification phase. Moreover, the modification phase behaves just like the value propagation phase and only incurs $O(d)$ operations when producing a message from a function-node. Thus, Max-sum_ADSSVP with local search has a smaller computational overhead than Max-sum_ADSSVP in general.

# 5. EMPIRICAL EVALUATION

## 5.1 Experimental Configurations

We use two DCOP problem types in our experiments, i.e., random DCOPs and scale-free networks [2]. For random DCOPs, we set the agent number to 120, the domain size to 10, graph density to 0.05 for sparse configuration and 0.6 for dense configuration, and uniformly select constraint costs from the range of 1 to 100. We use Barabási-Albert model to generate scale-free networks with an initial set of $m_0 = 15$ connected agents. At each iteration of the Barabási-Albert model procedure a new agent is connected to $m_1 = 3$ other agents (sparse configuration) or $m_1 = 10$ other agents (dense configuration) with a probability that is proportional to the number of links that the existing agents already have. The agent number, the domain size and the range of constraint costs in scale-free networks are the same as ones in random DCOPs.

We have applied DSA and MGM into Max-sum_ADSSVP, and found that Max-sum_ADSSVP with MGM works slightly better. The reason could be the monotonicity of MGM. Thus, in our experiments, we use MGM for Max-sum_ADSSVP with local search, named Max-sum_ADSSVP_MGM. We will compare Max-sum_ADSSVP and Max-sum_ADSSVP_MGM to standard Max-sum and its variants (i.e., Max-sum_AD and Max-sum_ADVP) for the above problems. To guarantee convergence, Max-sum_AD, Max-sum_ADVP and Max-sum_ADSSVP change their message-passing directions every 240 iterations and stop after 20 convergences. Besides, for obtaining the best results, we start value propagation for Max-sum_ADVP and Max-sum_ADSSVP after the second convergence. The iterations of each refining phase in Max-sum_ADSSVP_MGM are 30. Finally, we average the experimental results over 25 independently generated problems that are each solved by each algorithm 20 times.

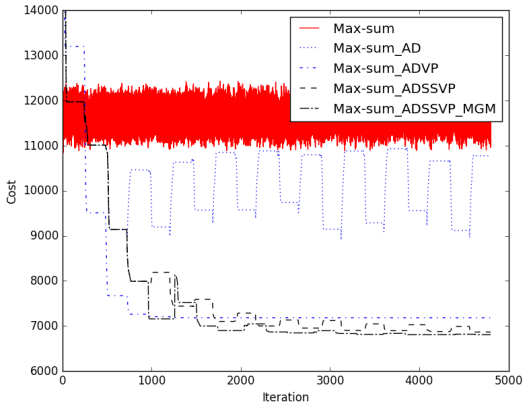


**Figure 10: Comparison of Max-sum variants on sparse configuration of scale-free networks**

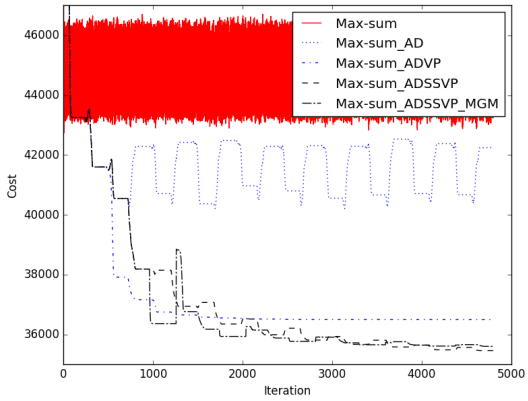


**Figure 11: Comparison of Max-sum variants on dense configuration of scale-free networks**

## 5.2 Experimental Results

To demonstrate how initial assignments affect Max-sum_ADVP and Max-sum_ADSSVP, we vary the timing for starting value propagation from the first to the fifth convergence. The results for sparse random DCOPs are shown in Figure 7.

It can be inferred from Figure 7 that Max-sum_ADVP is sensitive to the initial assignments since the results vary a lot among different value propagation timings. However, Max-sum_ADSSVP has similar performances for different value propagation timings, which indicates that our method is independent of initial assignments and thus overcomes the drawback of timing selection in Max-sum_ADVP. Besides, it is interesting to find that both algorithms report the best solution qualities when starting value propagation after the third convergence. But it is still worth mentioning that the best timing of starting value propagation is problem-specific and usually presented as an empirical value.

Figure 8 shows the comparison among Max-sum, Max-sum_AD, Max-sum_ADVP, Max-sum_ADSSVP and Max-sum_ADSSVP_MGM for random DCOPs under the sparse configuration. It can be concluded that Max-sum and Max-sum_AD cannot converge and traverse the states with high costs. In contrast, Max-sum_ADVP converges after the fifth convergence (1200 iterations). However, Max-sum_ADVP gets trapped in local optima. Max-sum_ADSSVP iteratively refines the solution after the second convergence (480 iterations) in every two convergences and eventually transcends Max-sum_ADVP (after 3600 iterations). Moreover, it can be observed from Figure 8 that Max-sum_ADSSVP needs lots of iterations to show its superiority since solutions are not sufficiently optimized in every two convergences. Max-sum_ADSSVP_MGM overcomes the pathology by performing short-range optimization to the solution after every value propagation phase (i.e., 960 iterations, 1740 iterations, 2520 iterations, etc.) and suppressing cost fluctuations efficiently in the exploration phases, and transcends Max-sum_ADVP after 2260 iterations. Compared to Max-sum_ADSSVP, Max-sum_ADSSVP_MGM has smoother cost fluctuations during the transitions from value propagation phases to exploration phases, which indicates that local search can help Max-sum_ADSSVP to suppress cost fluctuations. Besides, it is noticeable that the improvements of our proposed methods over Max-sum_ADVP are about 6.8% and 9% at the end of execution.

Figure 9 shows the similar trend for random DCOPs under the dense configuration. Max-sum_AD fluctuates wildly around the states with high costs because the factor graphs for the dense problems include more cycles which are prone to produce more inconsistent contexts. Conversely, Max-sum_ADSSVP exhibits obvious cost decreases in every exploration phase (i.e., 1200 iterations, 1680 iterations and so on), which indicates that the value propagation phase can help the exploration phase to eliminate inconsistence contexts. It is worth mentioning that Max-sum_ADSSVP_MGM suppresses the cost fluctuations more effectively for the dense problems and is superior to Max-sum_ADVP after 1300 iterations. Due to tremendous inconsistent contexts in dense problems, our proposed methods need more iterations to iteratively refine solutions. Although it only improves the Max-sum_ADVP by 1.1% at the end of execution, there is an apparent downtrend in Max-sum_ADSSVP, which indicates that our method still has great potential.

Figure 10 and Figure 11 show the performance comparisons among all 5 algorithms under the sparse configuration ($m_1 = 3$) and dense configuration ($m_1 = 10$) for scale-free networks, respectively. It can be seen that Max-sum_ADVP converges very fast for the sparse configuration (1200 iterations). This is because these sparse problems have the ultra-small diameters, which is a fundamental property of scale-free networks. We can also see that Max-sum_ADSSVP outperforms Max-sum_ADVP after the ninth direction change (2160 iterations) for the sparse configuration and the seventh direction change (1680 iterations) for the dense configuration, which demonstrates that the single-side value propagation schema has an advantage over the two-phase value propagation in Max-sum_ADVP.

# 6. CONCLUSION

In this paper, we analyze how value propagation affects the Max-sum process, and propose an iteratively refined Max-sum_AD algorithm which has both the exploration brought by Max-sum and the accuracy brought by value propagation. Also, our proposed algorithm solves the timing selection problem in Max-sum_ADVP, which is a major concern in value propagation. Besides, we speed up the converge pro-

cess by introducing local search after the value propagation phase. The experiment results demonstrate that our proposed algorithms are superior to Max-sum, Max-sum_AD and Max-sum_ADVP.

However, we also notice that the construction of a DAG in Max-sum_AD and its variants is problem-irrelevant since agent *id* which is independent of problem structure is used to determine the message-passing order. As a result, the algorithms cannot utilize structures of problems to eliminate potential inconsistent contexts. So, our future work is to explore various heuristics (such as the degree of each node) and provide a problem-specific method for DAG construction.

## REFERENCES


[1] S. M. Aji and R. J. McEliece. The generalized distributive law. *IEEE transactions on Information Theory*, 46(2):325–343, 2000.

[2] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[3] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 639–646. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[4] P. Gutierrez and P. Meseguer. Saving redundant messages in bnb-adopt. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 1259–1260. AAAI Press, 2010.

[5] K. Hirayama and M. Yokoo. The distributed breakout algorithms. *Artificial Intelligence*, 161(1):89–115, 2005.

[6] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519, 2001.

[7] K. M. Lei. Maintaining soft arc consistencies in bnb-adopt+ during search. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 3227–3228. AAAI Press, 2013.

[8] A. R. Leite, F. Enembreck, and J.-P. A. Barthès. Distributed constraint optimization problems: Review and perspectives. *Expert Systems with Applications*, 41(11):5139–5157, 2014.

[9] R. T. Maheswaran, J. P. Pearce, and M. Tambe. Distributed algorithms for dcop: A graphical-game-based approach. In *ISCA PDCS*, pages 432–439, 2004.

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

[11] A. Netzer, A. Grubshtein, and A. Meisels. Concurrent forward bounding for distributed constraint optimization problems. *Artificial Intelligence*, 193:186–216, 2012.

[12] J. P. Pearce and M. Tambe. Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 1446–1451. Morgan Kaufmann Publishers Inc., 2007.

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

[14] A. Petcu and B. Faltings. Odpop: An algorithm for open/distributed constraint optimization. In *Proceedings of AAAI Conference on Artificial Intelligence*, volume 21, pages 703–708. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[15] A. Petcu and B. Faltings. Mb-dpop: a new memory-bounded algorithm for distributed optimization. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 1452–1457. Morgan Kaufmann Publishers Inc., 2007.

[16] A. Petcu and B. Faltings. Distributed constraint optimization applications in power networks. *International Journal of Innovations in Energy Systems and Power*, 3(1):1–12, 2008.

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

[18] E. Rollon and J. Larrosa. Improved bounded max-sum for distributed constraint optimization. In *Principles and Practice of Constraint Programming*, pages 624–632. Springer, 2012.

[19] E. Rollon and J. Larrosa. Decomposing utility functions in bounded max-sum for distributed constraint optimization. In *International Conference on Principles and Practice of Constraint Programming*, pages 646–654. Springer, 2014.

[20] O. Steven, Z. Roie, and N. Aviv. Distributed breakout: Beyond satisfaction. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 447–453, 2016.

[21] E. A. Sultanik, P. J. Modi, and W. C. Regli. On modeling multiagent task scheduling as a distributed constraint optimization problem. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 1531–1536, 2007.

[22] W. Yeoh, A. Felner, and S. Koenig. Bnb-adopt: An asynchronous branch-and-bound dcop algorithm. *Journal of Artificial Intelligence Research*, 38:85–133, 2010.

[23] W. Zhang, G. Wang, Z. Xing, and L. Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, 161(1):55–87, 2005.

[24] R. Zivan, S. Okamoto, and H. Peled. Explorative anytime local search for distributed constraint optimization. *Artificial Intelligence*, 212:1–26, 2014.

[25] R. Zivan and H. 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-Volume 1*, pages 265–272. International Foundation for Autonomous Agents and Multiagent Systems, 2012.