Autonomous Agents and Multi-Agent Systems (2018) 32:822–860
https://doi.org/10.1007/s10458-018-9395-y

A class of iterative refined Max-sum algorithms via non-consecutive value propagation strategies

Ziyu Chen[1] · Yanchen Deng[1] · Tengfei Wu[1] · Zhongshi He[1]

Published online: 25 September 2018

Abstract
As an important technique to solve distributed constraint optimization problems, Max-sum has drawn a lot of attention and successfully been deployed in real applications. Unfortunately, Max-sum fails to converge in cyclic problems and usually traverses states with low quality. Max-sum_AD and Max-sum_ADVP were proposed to guarantee the single phase convergence and the cross phase convergence respectively, and greatly improve the solution quality of Max-sum. However, 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 prove that value propagation could restrict the exploration ability brought by Max-sum and eventually makes Max-sum_ADVP equivalent to a sequential greedy local search algorithm. For getting a balance between exploration and exploitation, several non-consecutive value propagation strategies are proposed to relax the restriction caused by value propagation: single-side value propagation which executes value propagation and Max-sum_AD in an interleaved way, probabilistic value propagation which performs value propagation stochastically and hybrid belief/value propagation where agents perform Max-sum_AD and value propagation in one round. We illustrate that agents in our algorithms can make decisions beyond local functions. Our empirical evaluations demonstrate the superiority of our methods over Max-sum and its variants. It also can be found that

The paper is an extension to our AAMAS paper [3]. Beside execution examples and an extension to Max-sum_ADSSVP, we also present three algorithms that can substantially suppress the cost fluctuation.

This research is funded by Chongqing Research Program of Basic Research and Frontier Technology (No. cstc2017jcyjAX0030), Fundamental Research Funds for the Central Universities (No. 2018CDXYJSJ0026) and Graduate Research and Innovation Foundation of Chongqing, China (Grant No. CYS18047).

✉ Ziyu Chen
chenziyu@cqu.edu.cn

✉ Yanchen Deng
dyc941126@126.com

Tengfei Wu
wutengfei0404@163.com

Zhongshi He
zshe@cqu.edu.cn

[1] College of Computer Science, Chongqing University, Chongqing 400044, China

our methods are independent of the value propagation timing which is a major concern in Max-sum_ADVP.

Keywords DCOP · Incomplete algorithm · Max-sum · Value propagation

1 Introduction

Distributed constraint optimization problems (DCOPs) [12] are a fundamental model for multi-agent system (MAS), which requires agents to coordinate their decisions to optimize a global objective. Since they can capture essential MAS aspects, DCOPs are widely applied into various MAS applications such as task scheduling [6,31], sensor networks [36], power networks [26] and so on.

According to whether they guarantee to find the optimal solution, algorithms for DCOPs can generally be divided into complete algorithms and incomplete algorithms. Search-based complete algorithms [8,13,16,17,33] perform distributed searches to exhaust the entire solution space. On the other hand, complete inference-based algorithms [23–25,32] employ dynamic programming to solve DCOP. However, since DCOPs are NP-Hard, complete algorithms suffer from an exponentially increasing coordination overhead (e.g., message number or memory consumption) which prohibits them from scaling up to large real application problems. In contrast, incomplete algorithms usually require little computation and communication to find suboptimal solutions, which are successfully applied into practical applications. Thus, considerable research efforts have been made to develop incomplete algorithms.

Incomplete algorithms generally follow three strategies, i.e., local search, sampling and inference. Local search algorithms [9,10,14,30,36] usually perform in a synchronous way. Instead of systematically exploring the entire solution space, agents in local search algorithms usually try to optimize solutions via iterative local moves. Sampling-based algorithms [18,20] sample the search space to approximate a function as a product of statistical inference.

Max-sum [7] is an important inference-based algorithm that employs belief propagation to propagate and accumulate utilities through the whole factor graph. Specifically, each agent in Max-sum maintains belief about utility for each possible assignment, and keeps updating its beliefs based on the messages received from its neighbors. Agents in Max-sum make decisions by choosing the assignments with the highest utility. Unfortunately, Max-sum only guarantees to converge in cycle-free problems which are very rare in realistic applications.

Bounded Max-sum (BMS) [27] was proposed to overcome the pathology by removing a subset of edges from a cyclic factor graph to make it acyclic. And then Max-sum is used to solve the relaxed problem, providing a bound on the approximation of the optimal solution. Some improved algorithms [28,29] further enhance BMS by providing tighter approximation ratios.

Different than removing dependencies in BMS algorithms, Max-sum_AD [40] makes a factor graph acyclic by strictly controlling the direction of message-passing. That is, instead of sending messages to every neighbor in Max-sum, nodes in Max-sum_AD only send messages to ones ordered after them. And the message-passing direction is alternated after the algorithm converges. However, the algorithm still cannot guarantee the cross phase convergence and usually produces low-quality solutions.

Max-sum_ADVP [40] further enforces the cross phase convergence and monotonicity by performing value propagation on each variable nodes. That is, variable nodes propagate both their beliefs and the assignments they choose, and function nodes consider the assignments

they receive to produce messages. Nevertheless, the exploitative nature of Max-sum_ADVP restricts the solutions that are ultimately found. Thus, several value exploration methods and message exploration methods have been proposed in [39] to alleviate the problem. While value exploration methods attempt to expand the set of candidate solution, message exploration methods enable agents to take advantage of the differing balance of exploration and exploitation in each variant by executing different versions of Max-sum algorithms in an interlaced way. However, according to the experimental results in [39], only message exploration methods slightly improve the solution quality of Max-sum_ADVP.

Our study tries to balance exploration and exploitation in another perspective. We start by observing that agents in Max-sum_ADVP eventually get trapped in local optima, and further establish the equivalence between Max-sum_ADVP and greedy local search algorithms in terms of the decision-making strategy. Then instead of alternatively executing different versions of Max-sum algorithms in [39], we focus on how to effectively balance exploration and exploitation by providing fine-grained control on executing belief propagation and value propagation. Specifically, our work contributes to state-of-the-art Max-sum algorithms in the following aspects:

- To avoid Max-sum to be a pure exploitative one, we propose three Max-sum algorithms with novel non-consecutive value propagation strategies which can balance exploration and exploitation: Max-sum_ADSSVP, Max-sum_ADPVP and Max-sum_HBVP. Max-sum_ADSSVP performs a single-side value propagation phase every two convergences. That is, when the current message-passing order is backward direction, the algorithm behaves like Max-sum_AD to perform belief propagation to find potential optima, otherwise it enables value propagation to guarantee solution quality. Max-sum_ADPVP remedies the problem by perform value propagation stochastically, while belief propagation and value propagation are performed simultaneously in one round in Max-sum_HBVP.
- We empirically evaluate our algorithms on various DCOP benchmarks. Our study shows that our proposed algorithms outperform Max-sum and its variants, including Max-sum_AD, Max-sum_ADVP, Max-sum_ADVP with exploration methods and Damped Max-sum. It also can be observed that in Max-sum_ADSSVP, there are remarkable decreases of amplitudes of cost fluctuations in the transitions from value propagation phases to belief propagation phases, which indicates that the belief propagation phase and the value propagation phase can collaborate with each other to iteratively eliminate invalid assumptions. Moreover, it is worth mentioning that all our proposed algorithms have similar performances under different value propagation timings, which indicates that our methods are robust against the value propagation timing and thus overcome the drawback of timing selection in Max-sum ADVP [40].

The rest of the paper is organized as follows. In Sect. 2, we briefly review related work. The formal definition to DCOPs is presented in Sect. 3. The preliminary algorithms including Standard Max-sum, Max-sum_AD and Max-sum_ADVP can be found in Sect. 4. In Sect. 5, we analyse the limitation of value propagation and present the detail of our proposed algorithms: Max-sum_ADSSVP, Max-sum_HBVP and Max-sum_ADPVP. Section 6 presents the empirical evaluation to our proposed methods and Sect. 7 concludes the paper and gives the future work.

2 Related work

Complete algorithms for DCOP can be broadly divided into search-based algorithms and inference-based algorithms. Search-based complete algorithms including ADOPT [16], Bnb-ADOPT [33], AFB [8], ConFB [17], PT-FB [13], etc., perform distributed searches to explicitly enumerate all possible combinations to find the optimal solution, while inference-based algorithms employ dynamic programming to optimally solve DCOP. As a distributed implementation of bucket elimination algorithm [5], DPOP [23] is an important inference-based algorithm that performs dynamic programming on a pseudo tree, starting with a phase of utility propagation. In the phase, each agent joins the received utilities from its children with its local constraint utilities, eliminates its dimensionality from the joint utilities by calculating the optimal utility for each combination of the remaining dimensionalities, and propagates the reduced utilities to its parent. After that, a value propagation phase starts from the root agent. In the phase, each agent selects its optimal assignment based on the utilities calculated in the previous phase and the assignment received from its parent, and broadcasts its assignment to its children. The algorithm terminates when all agents have chosen their optimal assignments. DPOP requires a linear number of messages, but its memory consumption is still exponential in the induced width. Thus, Petcu and Faltings presented ODPOP [24] and MB-DPOP [25] to trade runtimes for smaller memory requirements. In addition, Action-GDL [32] was proposed to generalize DPOP by executing over a distributed junction tree.

Local search algorithms including DBA [9], DSA [36], MGM [14] are typical incomplete algorithms and usually perform in a synchronous way. In those algorithms, agents keep exchanging self states (i.e., assignments or gains) with their neighbors, and then determine whether to replace their assignments based on the received states from their neighbors. The difference among local search algorithms mainly lies on the assignment replacement strategy. For example, agents in DSA decide to replace their assignments by making stochastic decisions in every iteration, while only agents who hold the maximum gains among neighbors can replace their assignments in MGM. Recently, GDBA [30] was proposed to adapt DBA which is designed to solve distributed constraint satisfaction problems (DCSPs) [34], to general-valued DCOPs by formalizing the manner of computing effective costs based on original costs and modifiers, the definition of constraint violation, and the scope of changes to the modifiers during breakouts. To improve the solution of local convergence, KOPT [10] was proposed to produce local optimal solutions which are guaranteed to be within a predefined distance from the global optimal solution by coordinating the decisions of all agents within the k-size coalition. Specifically, agents in each coalition transfer their constraints to a mediator which performs a centralized complete search to find the best assignment for all agents within the k-size coalition. Consequently, the algorithm eventually converges to a status called k-optimal [21] ensuring that the solution quality cannot be improved if k agents or fewer change their assignments.

Unfortunately, since agents might have suboptimal local assignments when a system is in the optimal state, many local search algorithms cannot report the best solution they have visited (i.e., they are not anytime algorithms [37]). Thus, an anytime local search (ALS) framework [38] was proposed to cache the best solution explored so far by aggregating local costs through a breadth-first tree. Besides, local search algorithms usually converge quickly since agents only communicate their preferred states based on the current preferred states of their neighbors. Therefore, Yu et al. proposed a scheme for local search algorithms, called

partial decision scheme (PDS) [35] to help agents to escape from premature convergences by ignoring the assignments of their neighbors.

Sampling-based algorithms like DUCT [20] and D-Gibbs [18] are emerging incomplete approaches to solve DCOP in recent years. Given a context a, an agent in DUCT first constructs a confidence bound for each value k in its domain, which is an optimistic estimate of the optimal value for its subtree under context a and choice k, and samples the choice with the lowest bound. However, memory requirement per agent in DUCT is exponential in the number of agents, which makes it unsuitable for large real applications. Thus, Nguyen et al. proposed a memory-bounded DCOP algorithm called D-Gibbs that maps a DCOP to a maximum likelihood estimation (MLE) problem, and uses the distributed gibbs sampling algorithm to solve it.

Max-sum algorithms are popular inference-based incomplete algorithms in which agents communicate utilities instead of self-states. Bounded Max-sum [27], as a notable Max-sum variant, attempts to overcome the non-convergence of Max-sum by removing edges from a cyclic factor graph to make it acyclic. Specifically, the algorithm first shrinks the binary dependencies which have the least impacts on the solution to unary functions by minimizing the dependencies, which is called relaxation phase. Consequently, the algorithm transforms a cyclic factor graph into a tree-structured graph, 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) [28] was proposed to provide a tighter approximation ratio. Instead of only solving a problem with minimized dependencies in BMS, IBMS also solves the maximized one and selects the best result to calculate the approximation ratio. Nonetheless, all BMS algorithms introduce errors in their relaxation phase, which prevents the algorithms from computing the true optimal solution. ED-IBMS [29] and AD-IBMS [29] attempt to alleviate the pathology by decomposing binary dependencies into unary functions.

Recently, Cohen et al. [4] investigated the effect of using damping within Max-sum when applied into DCOPs. Damping is a technique to increase the chances for convergence of belief propagation by decreasing the effect of cyclic information propagation. It can be observed that the Damped Max-sum with a high damping factor under the anytime mechanism can outperform all other versions of Max-sum, as well as local search DCOP algorithms [4].

ADPOP [22] is an approximate version of DPOP, which allows the desired tradeoff between solution quality and computational complexity. Specifically, the algorithm imposes a limit $maxDims$ on the maximum number of dimensions in each message. When the dimensionality of an outgoing message exceeds the limit, the algorithm drops a set of dimensions to stay below the limit. That is, the algorithm first selects a set of dimensions to be dropped and then computes upper bound and lower bound respectively by eliminating those dimensions from the utility. It is worth noting that the approximation only occurs in high-width areas of a problem; for all the rest of the problem, where the dimensionality does not exceed $maxDims$, optimal sub-solutions are still found.

Okimoto et. al [19] proposed a new incomplete algorithm based on a total ordering on agents. The algorithm starts with generating a subgraph from an induced choral graph by removing edges so that the induced width of the induced chordal graph obtained from the subgraph is bounded by a parameter p. Then, a complete algorithm is used to solve the reduced problems, producing a solution that guarantees the p-optimality, i.e., the solution maximizes the rewards (or minimizes the cost) in the reduced problem.

3 Distributed constraint optimization problems

A distributed constraint optimization problem can be formalized with 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. Each domain D_i consists of a set of allowable values for variable x_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 cost 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 the variables that minimizes the total cost, which is the sum of all constraints:

$$X^* = \arg\min_{\mathbf{x}} \sum_{f_i \in F} f_i$$

where $\mathbf{x} = (x_1, \ldots, x_n)$. 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

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

A DCOP can be visualized by a constraint graph where the nodes represent agents and the edges represent constraints. Fig. 1a, b gives the constraint graph and the constraint matrices of a DCOP which comprises of four agents and four constraints.

4 Algorithm preliminaries

4.1 Standard Max-sum

Max-sum is a GDL (generalized distributive law) [1] based message-passing algorithm operating on factor graphs [11]. A factor graph is a bipartite graph representation to a DCOP,

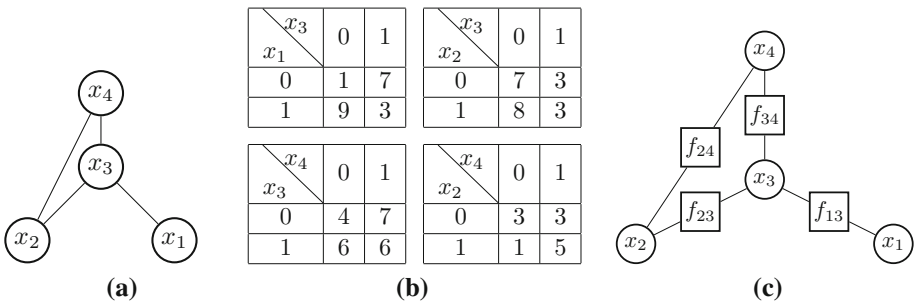


Fig. 1 A DCOP instance and its factor graph

which is composed of two types of nodes: variable nodes and function nodes. Function nodes which represent constraints in a DCOP are connected only to variable nodes they depend on, while variable nodes which represent variables in a DCOP are connected only to function nodes they are involved in Fig. 1c presents a factor graph deriving from Fig. 1a.

Beliefs are propagated and accumulated through the whole factor graph via two types of messages: query message (i.e., the message from a variable node to a function node) and response message (i.e., the message from a function node to a variable node). When computing a message to a function node f_j, variable node x_i sums up all the last messages received from its neighboring function nodes except the target function node f_j. Formally, the message from x_i to f_j is a function $q_{i\to j}(x_i) : D_i \to \mathbb{R}$ such that:

$$q_{i\to j}(x_i) = \alpha_{ij} + \sum_{n\in N_i\setminus j} r_{n\to i}(x_i) \tag{1}$$

where $N_i\setminus j$ is a set of neighbor indexes of x_i except the target function node f_j and $r_{n\to i}(x_i)$ is the message sent from function node f_n to x_i. In order to prevent entries in the message from endlessly increasing in a cyclic graph, a constant α_{ij} such that

$$\sum_{d_i\in D_i} q_{i\to j}(d_i) = 0 \tag{2}$$

is chosen to normalize the message.

The message from a function node f_j to a variable node x_i contains the best cost for each $d_i \in D_i$ under the current beliefs and the local function of f_j. That is, the function node f_j minimizes the sum of its local function and all the last messages received from its neighboring variable nodes except the target variable node x_i, in terms of variables other than x_i.[1] Formally, the message from f_j to x_i is a function $r_{j\to i}(x_i) : D_i \to \mathbb{R}$ such that:

$$r_{j\to i}(x_i) = \min_{\mathbf{x}_j\setminus x_i} \left[f_j(\mathbf{x}_j) + \sum_{n\in N_j\setminus i} q_{n\to j}(x_n) \right] \tag{3}$$

where N_j is a set of indexes of neighboring variable nodes connecting to f_j and $\mathbf{x}_j\setminus x_i = \{x_k : k \in N_j\setminus i\}$ and $\mathbf{x}_j$ is a vector of variables involved in f_j.

When a variable node x_i makes a decision, it first considers all messages it receives to calculate the belief for each possible assignment. That is,

$$z_i(x_i) = \sum_{n\in N_i} r_{n\to i}(x_i) \tag{4}$$

Then, x_i will choose the value with the lowest cost according to z_i. The procedure can be formalized by

$$x_i^* = \arg\min_{x_i} z_i(x_i) \tag{5}$$

4.2 Max-sum_AD

It is known that Standard Max-sum cannot guarantee to converge and usually produces low-quality solutions when factor graphs are cyclic. Max-sum on an alternating DAG (Max-sum_AD) was proposed to overcome the shortcoming by strictly controlling the nature of

[1] We describe Max-sum as a minimization version to cope with the objective of DCOPs.

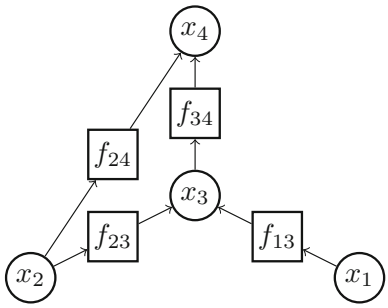


Fig. 2 An example of a directed acyclic factor graph

cyclic information propagation. Specifically, the algorithm makes a factor graph acyclic by transforming it to a directed acyclic graph according to a predefined order and only permits messages in the direction of each edge. In other words, nodes in Max-sum_AD only send messages to nodes which are after them according to the directions. The directions of all edges are periodically and synchronously reversed in order to make agents able to consider all constraints they are involved in. Figure 2 gives an example of a DAG deriving from Fig. 1b, and the pseudo code of Max-sum_AD can be found in Fig. 3.

The messages in Max-sum_AD are computed according to Eqs. (1) and (3), just like Standard Max-sum. It should be noted that those computations involve all messages received from neighbors rather than just upstream neighbors. In other words, a message is computed by considering the messages from upstream neighbors received in the current phase and the messages from downstream neighbors received in the previous phase, which enables cyclic information propagation across phases and allows agents to accumulate information based on all constraints in a DCOP.

An important property of Max-sum_AD is that it can guarantee the single phase convergence. That is, the messages of all the nodes and the beliefs of all the variable nodes do not change after a linear number of iterations in one direction where the number of iterations actually depends on the diameter of the DAG.

4.3 Max-sum_ADVP

Although it can guarantee the single phase convergence, Max-sum_AD suffers from belief ties and invalid assignment assumptions which prevent a variable node from making high-quality decisions. To illustrate the phenomenon, let's consider a graph-coloring problem whose initial message-passing order is shown in Fig. 4. According to Max-sum_AD, the messages sent in 4 iterations are shown as follows.[2]

Max-sum_AD

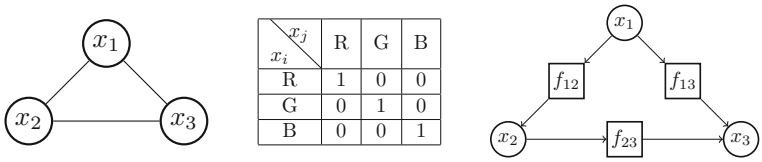
Iteration 1:	$x_1 \rightarrow f_{12} : [0, 0, 0]$	$x_1 \rightarrow f_{13} : [0, 0, 0]$	
Iteration 2:	$f_{12} \rightarrow x_2 : [0, 0, 0]$	$f_{13} \rightarrow x_3 : [0, 0, 0]$	
Iteration 3:	$x_2 \rightarrow f_{23} : [0, 0, 0]$		
Iteration 4:	$f_{23} \rightarrow x_3 : [0, 0, 0]$		
Beliefs:	$z_1 = [0, 0, 0]$	$z_2 = [0, 0, 0]$	$z_3 = [0, 0, 0]$
Assignments:	$x_1^* = \mathrm{R}$	$x_2^* = \mathrm{R}$	$x_3^* = \mathrm{R}$

[2] For sake of clarity and simplicity, we only show stable and unchanging outgoing messages here. In fact, nodes in Max-sum_AD perform concurrently and each node sends messages to its downstream neighbors every iteration. Besides, we also ignore the normalization to messages to make the trace more clear.

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
```

Fig. 3 Sketch for Max-sum_AD



(a) **(b)** **(c)**

Fig. 4 An instance of graph-coloring

It can be easily observed that all the variable nodes will choose assignment R and therefore reach a total cost 3 after a phase of message-passing since there are ties among their beliefs. Concretely, variable nodes in Max-sum_AD always optimistically assume that their upstream variable nodes will choose the assignments minimizing costs according to Eq. (3), which could usually be failed. For example, according to its belief $z_2(x_2)$, x_2 can reach a cost with 0 if it choose assignment R. However, the cost can be achieved only when x_1 choose assignment G or B, which is impossible in this situation.

To give a better insight into the pathology, consider the following definitions.

Definition 1 (*Context*) A context of a message from a function node to a variable node specifies for each cost in the message, the assignments of all involved variables except the target to produce such cost.

For example, the context of a message from f_{ij} to x_i in Max-sum_AD can be specified by a function $c_{f_{ij}\to x_i}(x_i) : D_i \to D_j$ such that:

$$c_{f_{ij}\to x_i}(x_i) = \arg\min_{x_j}[f_{ij}(x_i, x_j) + q_{x_j\to f_{ij}}(x_j)]$$

Note that the context is a concept that helps us to analyze the algorithm, and Max-sum_AD itself is not required to propagate such a context.

Definition 2 (*Number of invalid assumptions*) The number of invalid assumptions of a variable node x_i in Max-sum algorithms on DAGs is a number IA_i such that:

$$IA_i = \sum_{n \in P_i} \mathbb{I}(c_{f_{ni} \to x_i}(k_i) \neq k_n)$$

where P_i is a set of indexes of neighboring agents whose variable nodes are before x_i in the current order, k_n is the assignment of x_n, and $\mathbb{I}$ is an indicator function.

It is worth mentioning that in the definition, only neighboring agents whose variable nodes are before the variable node in the current order are considered. That is because the assignments of downstream variable nodes do not converge before the assignment of the variable node converges. Thus, taking downstream variable nodes into consideration is meaningless.

Context functions and the number of invalid assumptions of the above example are shown as follows.

$c_{f_{12} \to x_2} = [\text{G,R,R}] \quad c_{f_{13} \to x_3} = [\text{G,R,R}] \quad c_{f_{23} \to x_3} = [\text{G,R,R}]$
$IA_1 = 0$
$IA_2 = \mathbb{I}(c_{f_{12} \to x_2}(R) \neq R) = 1$
$IA_3 = \mathbb{I}(c_{f_{13} \to x_3}(R) \neq R) + \mathbb{I}(c_{f_{23} \to x_3}(R) \neq R) = 2$

From the result, one can easily find that all the variable nodes except x_1 suffer from invalid assumptions, which eventually leads to the poorest solution.[3]

Max-sum_ADVP remedies the problem by performing value propagation after a certain timing. That is, after enabling value propagation, variable nodes in Max-sum_ADVP propagate both the beliefs they have accumulated and the assignments they have chosen. Function nodes produce messages by considering the received assignments. Specifically, instead of minimizing the function over all the variables except the target, a function node now fixes the received assignments of all the variables except the target to compute messages. In this way, the invalid assumptions are eliminated since the beliefs in a message from a function node are computed in terms of the assignments of upstream variable nodes.

Consider again the example shown in Fig. 2. When value propagation is enabled in the first iteration, the trace of Max-sum_ADVP is shown as follows.

Max-sum_ADVP

Iteration 1:	$x_1 \to f_{12} : [0,0,0], x_1 = \text{R}$	$x_1 \to f_{13} : [0,0,0], x_1 = \text{R}$	
Iteration 2:	$f_{12} \to x_2 : [1,0,0]$	$f_{13} \to x_3 : [1,0,0]$	
Iteration 3:	$x_2 \to f_{23} : [1,0,0], x_2 = \text{G}$		
Iteration 4:	$f_{23} \to x_3 : [0,1,0]$		
Beliefs:	$z_1 = [0,0,0]$	$z_2 = [1,0,0]$	$z_3 = [1,1,0]$
Assignments:	$x_1^* = \text{R}$	$x_2^* = \text{G}$	$x_3^* = \text{B}$

It can be observed that value propagation can help variable nodes to break the ties in their beliefs and reach a total cost 0 after a phase of message-passing. We also give an analysis on context functions and invalid assumptions during the execution as follows.

$c_{f_{12} \to x_2} = [\text{R,R,R}] \quad c_{f_{13} \to x_3} = [\text{R,R,R}] \quad c_{f_{23} \to x_3} = [\text{G,G,G}]$
$IA_1 = 0$

[3] In fact, Standard Max-sum without personal preferences cannot solve graph-coloring problem for the same reason.

$IA_2 = \mathbb{I}(c_{f_{12}\to x_2}(G) \neq R) = 0$
$IA_3 = \mathbb{I}(c_{f_{13}\to x_3}(B) \neq R) + \mathbb{I}(c_{f_{23}\to x_3}(B) \neq G) = 0$

It is noticeable that after value propagation is enabled, the contexts are identical to the assignments of upstream variable nodes and the number of invalid assumptions for each variable node is decreased to zero, which demonstrates the power of value propagation.

5 Proposed methods

5.1 Motivation

In this section, we will first theoretically show that, although Max-sum_ADVP can guarantee the cross phase convergence and greatly improve solution quality of Max-sum_AD, value propagation can block the belief propagation. As a result, agents fail to accumulate the global information and the algorithm will eventually behave like a sequential greedy local search algorithm. Then, we will give an example to explicitly show Max-sum_ADVP can eventually get stuck in a local optimum.

Lemma 1 *Function nodes will block the belief propagation, and can only propagate local functions when value propagation is enabled.*

Proof Without loss of generality, consider the part of a DAG which consists of two variable nodes and a function node, shown as Fig. 5. When value propagation is enabled, variable node x_i propagates its belief, as well as its assignment k_i, to its downstream function node f_{ij}. That is,

$$x_i \to f_{ij} : q_{x_i\to f_{ij}}(x_i) = \sum_{n\in N_i\setminus j} r_{f_{ni}\to x_i}(x_i), x_i = k_i \tag{6}$$

f_{ij} produces the message to x_j by considering the assignment it received, That is,

$$f_{ij} \to x_j : r_{f_{ij}\to x_j}(x_j) = f_{ij}(k_i, x_j) + q_{x_i\to f_{ij}}(k_i) \tag{7}$$

Since it is a constant and has no influence on the decision-making of x_j, the term $q_{x_i\to f_{ij}}(k_i)$ can be removed safely. And thus Eq. (7) can be simplified as

$$f_{ij} \to x_j : r_{f_{ij}\to x_j}(x_j) = f_{ij}(k_i, x_j) \tag{8}$$

Obviously, the right-hand side of Eq. (8) is a local function with a given assignment and the belief is eliminated, which concludes the lemma. □

The immediate corollary from Lemma 1 is that messages in a factor graph are all local functions after the first convergence of value propagation. At the same time, variable nodes also fail to collect and forward the global beliefs. In other words, value propagation restricts

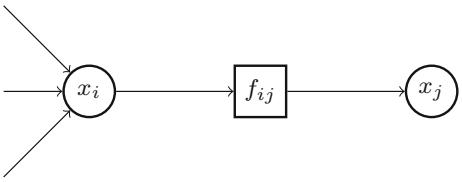


Fig. 5 Simple factor graph with two variable nodes and a function node

the exploration brought by Max-sum since agents no longer can utilize the global information to make decisions. Next, we will theoretically show that agents in Max-sum_ADVP eventually follow the decision-making strategy of a greedy local search algorithm.

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

Proof Since value propagation has converged in the previous phase, the beliefs from the downstream neighbors of a variable node x_i are local functions according to Lemma 1. Similarly, its upstream neighbors can only propagate local functions with respect to their assignments. Thus, the belief for x_i is

$$z_i(x_i) = \sum_{j \in P_i} f_{ij}(x_i, k_j^m) + \sum_{j \in S_i} f_{ij}(x_i, k_j^{m-1}) \tag{9}$$

where k_j^m is the assignment selected by variable node x_j in the mth phase and S_i denotes the set of indexes of the neighboring agents whose variable nodes are after x_i. Therefore x_i selects an assignment to minimize Eq. (9), which indicates that it always makes the best response in terms of the given assignments from its neighbors.

On the other hand, agent a_i in greedy local search algorithms (e.g., DSA, MGM, etc.) also makes decisions by considering all neighbors' assignments. That is:

$$\sum_{j \in N_i} f_{ij}(x_i, k_j) \tag{10}$$

And then a_i greedily selects an assignment to minimize Eq. (10). It can be concluded from Eqs. (9) and (10) that Max-sum_ADVP and greedy local search algorithm share the same decision-making strategy. So, Proposition 1 is proved. □

Note that Proposition 1 only establishes the equivalence between Max-sum_ADVP and greedy local search algorithms in terms of the decision-making strategy, and they are different in other aspects. For example, agents in DSA stochastically replace their assignments by the best responses in parallel, while variable nodes in Max-sum_ADVP make decisions sequentially. Proposition 1 indicates that agents in Max-sum_ADVP cannot utilize the global beliefs, and will eventually behave like the ones in a greedy local search algorithm. Thus, Max-sum_ADVP also suffers from getting stuck in local optima. Consider the DCOP instance shown in Fig. 1 and the initial message-passing order shown in Fig. 2. Assume that value propagation is enabled after the second convergence (i.e., at the beginning of the third phase). The trace of Max-sum_ADVP is shown as follows.[4]

Phase 1 (belief propagation)

Iteration 1:	$x_1 \to f_{13} : [0, 0]$	$x_2 \to f_{23} : [0, 0]$	$x_2 \to f_{24} : [0, 0]$	
Iteration 2:	$f_{13} \to x_3 : [1, 3]$	$f_{23} \to x_3 : [7, 3]$	$f_{24} \to x_4 : [1, 3]$	
Iteration 3:	$x_3 \to f_{34} : [8, 6]$			
Iteration 4:	$f_{34} \to x_4 : [12, 12]$			
Beliefs:	$z_1 = [0, 0]$	$z_2 = [0, 0]$	$z_3 = [8, 6]$	$z_4 = [13, 15]$
Assignments:	$x_1^* = 0$	$x_2^* = 0$	$x_3^* = 1$	$x_4^* = 0$

Phase 2 (belief propagation)

Iteration 1:	$x_4 \to f_{24} : [12, 12]$	$x_4 \to f_{34} : [1, 3]$

[4] Since Lemma 1 has demonstrated that the global beliefs have no influence on the algorithm after enabling value propagation, we hereafter drop the global beliefs in response messages and only present exactly local functions in value propagation phases.

Iteration 2: $f_{34} \rightarrow x_3 : [5, 7]$ $f_{24} \rightarrow x_2 : [15, 13]$
Iteration 3: $x_3 \rightarrow f_{13} : [12, 10]$ $x_3 \rightarrow f_{23} : [6, 10]$
Iteration 4: $f_{13} \rightarrow x_1 : [13, 13]$ $f_{23} \rightarrow x_2 : [13, 13]$
Beliefs: $z_1 = [13, 13]$ $z_2 = [28, 26]$ $z_3 = [13, 13]$ $z_4 = [13, 15]$
Assignments: $x_1^* = 0$ $x_2^* = 1$ $x_3^* = 1$ $x_4^* = 0$

Phase 3 (value propagation)

Iteration 1: $x_1 \rightarrow f_{13} : [0, 0], x_1 = 0$ $x_2 \rightarrow f_{23} : [8, 3], x_2 = 1$
$x_2 \rightarrow f_{24} : [1, 5], x_2 = 1$
Iteration 2: $f_{13} \rightarrow x_3 : [1, 7]$ $f_{23} \rightarrow x_3 : [8, 3]$ $f_{24} \rightarrow x_4 : [1, 5]$
Iteration 3: $x_3 \rightarrow f_{34} : [9, 10], x_3 = 0$
Iteration 4: $f_{34} \rightarrow x_4 : [4, 7]$
Beliefs: $z_1 = [13, 13]$ $z_2 = [28, 26]$ $z_3 = [14, 17]$ $z_4 = [5, 12]$
Assignments: $x_1^* = 0$ $x_2^* = 1$ $x_3^* = 0$ $x_4^* = 0$

Phase 4 (value propagation)

Iteration 1: $x_4 \rightarrow f_{24} : [4, 7], x_4 = 0$ $x_4 \rightarrow f_{34} : [1, 5], x_4 = 0$
Iteration 2: $f_{34} \rightarrow x_3 : [4, 6]$ $f_{24} \rightarrow x_2 : [3, 1]$
Iteration 3: $x_3 \rightarrow f_{13} : [12, 9], x_3 = 0$ $x_3 \rightarrow f_{23} : [5, 13], x_3 = 0$
Iteration 4: $f_{13} \rightarrow x_1 : [1, 9]$ $f_{23} \rightarrow x_2 : [7, 8]$
Beliefs: $z_1 = [1, 9]$ $z_2 = [10, 9]$ $z_3 = [13, 16]$ $z_4 = [5, 12]$
Assignments: $x_1^* = 0$ $x_2^* = 1$ $x_3^* = 0$ $x_4^* = 0$

Phase 5 (value propagation)

Iteration 1: $x_1 \rightarrow f_{13} : [0, 0], x_1 = 0$ $x_2 \rightarrow f_{23} : [3, 1], x_2 = 1$
$x_2 \rightarrow f_{24} : [7, 8], x_2 = 1$
Iteration 2: $f_{13} \rightarrow x_3 : [1, 7]$ $f_{23} \rightarrow x_3 : [8, 3]$ $f_{24} \rightarrow x_4 : [1, 5]$
Iteration 3: $x_3 \rightarrow f_{34} : [9, 10], x_3 = 0$
Iteration 4: $f_{34} \rightarrow x_4 : [4, 7]$
Beliefs: $z_1 = [13, 13]$ $z_2 = [28, 26]$ $z_3 = [14, 17]$ $z_4 = [5, 12]$
Assignments: $x_1^* = 0$ $x_2^* = 1$ $x_3^* = 0$ $x_4^* = 0$

It can be observed from the trace that after two phases of value propagation (i.e., Phase 3 and Phase 4), Max-sum_ADVP has converged since neither the response messages nor the beliefs in Phase 5 differ from the ones in Phase 3. Thus, the algorithm eventually converges to a suboptimal solution whose cost is 14, while the optimal solution is $\{x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 0\}$ whose cost is 13. Also, the trace demonstrates that Max-sum_ADVP heavily relies on the initial assignments, which is a major weakness of value propagation. Suppose that the assignment for each variable node at the end of Phase 4 is $\{x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0\}$, then one can easily verify that the algorithm will converge to the optimal solution after the message-passing of Phase 5.

Since Lemma 1 and Proposition 1 have demonstrate that Max-sum_ADVP can only propagate the local information and eventually behaves like a greedy local search algorithm, efficient explorative mechanisms must be introduced to make agents able to access the global information. Thus, in the following sections, we propose several Max-sum algorithms with novel non-consecutive value propagation strategies to balance exploration and exploitation.

5.2 Max-sum_AD with single-side value propagation

The first strategy we use to get a balance between exploration and exploitation is alternatively executing Max-sum_AD and Max-sum_ADVP. In this way, value propagation will be always

executed in the same direction and thus the algorithm is called Max-sum_AD with single-side value propagation (Max-sum_ADSSVP for short). Specifically, after enabling value propagation, the following phases will be alternatively in each round:

- *Value propagation phase* enabling value propagation to guarantee the solution quality when the current message-passing order is forward direction.
- *Belief propagation phase* performing like Max-sum_AD to find promising initial assignments for the next phase of value propagation when the current message-passing order is backward direction.

Here, the terms "forward direction" and "backward direction" refer to the initial message-passing order and the corresponding reverse message-passing order. To give a better explanation on the algorithm, consider again the DCOP instance shown in Fig. 1 and the initial message-passing order shown in Fig. 2. We also assume value propagation is enabled after the second convergence. The trace of Max-sum_ADSSVP is presented as follows (we omit the first two phases since they are exactly the same as the ones of Max-sum_ADVP):

Phase 3 (value propagation)

Iteration 1:	$x_1 \to f_{13} : [0, 0], x_1 = 0$	$x_2 \to f_{23} : [8, 3], x_2 = 1$		
	$x_2 \to f_{24} : [1, 5], x_2 = 1$			
Iteration 2:	$f_{13} \to x_3 : [1, 7]$	$f_{23} \to x_3 : [8, 3]$	$f_{24} \to x_4 : [1, 5]$	
Iteration 3:	$x_3 \to f_{34} : [9, 10], x_3 = 0$			
Iteration 4:	$f_{34} \to x_4 : [4, 7]$			
Beliefs:	$z_1 = [13, 13]$	$z_2 = [28, 26]$	$z_3 = [14, 17]$	$z_4 = [5, 12]$
Assignments:	$x_1^* = 0$	$x_2^* = 1$	$x_3^* = 0$	$x_4^* = 0$

Phase 4 (belief propagation)

Iteration 1:	$x_4 \to f_{24} : [4, 7]$	$x_4 \to f_{34} : [1, 5]$		
Iteration 2:	$f_{34} \to x_3 : [5, 7]$	$f_{24} \to x_2 : [7, 5]$		
Iteration 3:	$x_3 \to f_{13} : [13, 10]$	$x_3 \to f_{23} : [6, 14]$		
Iteration 4:	$f_{13} \to x_1 : [14, 13]$	$f_{23} \to x_2 : [13, 14]$		
Beliefs:	$z_1 = [14, 13]$	$z_2 = [20, 19]$	$z_3 = [14, 17]$	$z_4 = [5, 12]$
Assignments:	$x_1^* = 1$	$x_2^* = 1$	$x_3^* = 0$	$x_4^* = 0$

Phase 5 (value propagation)

Iteration 1:	$x_1 \to f_{13} : [0, 0], x_1 = 1$	$x_2 \to f_{23} : [7, 5], x_2 = 1$		
	$x_2 \to f_{24} : [13, 14], x_2 = 1$			
Iteration 2:	$f_{13} \to x_3 : [9, 3]$	$f_{23} \to x_3 : [8, 3]$	$f_{24} \to x_4 : [1, 5]$	
Iteration 3:	$x_3 \to f_{34} : [17, 6], x_3 = 1$			
Iteration 4:	$f_{34} \to x_4 : [6, 6]$			
Beliefs:	$z_1 = [14, 13]$	$z_2 = [20, 19]$	$z_3 = [22, 13]$	$z_4 = [7, 11]$
Assignments:	$x_1^* = 1$	$x_2^* = 1$	$x_3^* = 1$	$x_4^* = 0$

It can be seen from the trace that instead of continuously performing value propagation in Max-sum_ADVP, our algorithm performs belief propagation after the third convergence. Therefore, the algorithm has an opportunity to access the new promising assignments rather than monotonically optimizing the old ones like Max-sum_ADVP, and eventually breaks out of the local optima. Moreover, since every belief propagation phase can provide the new assignments, our algorithm is less sensitive to the value propagation timings, which overcomes the major weakness of Max-sum_ADVP.

Note that variable nodes in our algorithm can make decisions beyond local functions. Specifically, we have the following observation.

Observation 1 *Variable nodes in Max-sum_ADSSVP always make decisions in terms of local functions and accumulated beliefs after enabling value propagation. Specifically, the belief of a variable node x_i at the end of phase m is*

$$z_i(x_i) = \sum_{n \in P_i} f_{ni}(k_n^m, x_i) + \sum_{n \in S_i} r^{m-1}_{f_{ni} \to x_i}(x_i)$$

if the current phase is a value propagation phase. Here, $r^{m-1}_{f_{ni} \to x_i}(x_i)$ denotes the message from f_{ni} to x_i at the end of phase $m-1$. Similarly, if the current phase is a belief propagation phase, the belief for x_i is

$$z_i(x_i) = \sum_{n \in P_i} r^{m}_{f_{ni} \to x_i}(x_i) + \sum_{n \in S_i} f_{ni}(k_n^{m-1}, x_i)$$

Taking the DAG in Fig. 2 for an example, if the current phase is value propagation phase, variable node x_3 uses the local functions from its upstream neighbors f_{23}, f_{13} and the accumulated belief from its downstream neighbor f_{34}. Formally, the belief for x_3 at the end of the current phase m is:

$$z_3(x_3) = f_{13}(k_1^m, x_3) + f_{23}(k_2^m, x_3) + r^{m-1}_{f_{34} \to x_3}(x_3)$$

The property helps the algorithm to escape from local optima by considering both the immediate benefits (i.e., local functions) and the global benefits (i.e., accumulated beliefs). However, since variable nodes now do not entirely make decisions based on local functions, the algorithm no longer guarantees monotonicity and the cross phase convergence. In practice, the solution quality usually fluctuates wildly during the transitions from value propagation phases to belief propagation phases, and the algorithm needs a lot of iterations to suppress the fluctuations.

We notice that messages are computed based on the messages from all neighbors to facilitate cyclic information propagation across phases. Thus, the solution quality at the end of a value propagation phase has a critic impact on the next phase of belief propagation. Considering the monotonicity of value propagation, our first method to remedy the aforementioned problem is pretty straightforward. We parameterize the algorithm by the length of consecutive value propagation phases t, namely Max-sum_ADSSVP (t).[5] Specifically, in each round the algorithm first performs t phases of value propagation and then performs a phase of belief propagation. In this way, the solution quality is sufficiently optimized before switching to a belief propagation phase and fluctuations can be efficiently suppressed. It should be noted that the algorithm is substantially different from Max-sum_ADVP with message exploration methods [39]. The purpose of executing multiple phases of value propagation in our algorithm is to sufficiently optimize solutions, while [39] alternatively executes different versions of Max-sum algorithms to balance exploration and exploitation. Besides, in each round of our algorithm, belief propagation should be executed exactly one phase. If two consecutive phases of belief propagation are executed, then agents in the latter belief propagation phase compute messages entirely based on the messages in the first belief propagation phase and have nothing to do with the previous value propagation phases.

The second method we use to tackle the problem is to introduce a local search algorithm to improve the solution quality after value propagation phases, which yields an algorithm called

[5] We notice that in the method value propagation is no longer always performed in a direction, and thus the name is somewhat misleading. However, since the paper is an extension to our AAMAS paper [3], we inherit the naming convention from [3].

Max-sum_ADSSVP with local search. Note that although we have proved that value propagation will make Max-sum_ADVP eventually behave like a greedy local search algorithm, it is still necessary to perform a local search for the further optimization. That is because value propagation needs lots of iterations to refine the solutions, and it only guarantees to produce 1-Opt solutions at best. Thus, local search algorithms should be introduced for quick and substantial optimizations. Specifically, the following phases are performed after a value propagation phase:

- *Refining phase* performing 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 assignment generated by the refining phase to apply new assignments into a factor graph.

The sketch of the algorithm is presented in Fig. 6. The algorithm is parameterized by the adopted local search algorithm, the number of iterations k for a phase of value propagation or belief propagation and the number of iterations l for local search. An agent performs belief propagation when the current message-passing order is backward direction (line 5–7), otherwise it performs value propagation (line 8–11). If the current message-passing order is backward direction (i.e., agents are in a value propagation phase), the refining phase and modification phase are trigged consecutively after the first k iterations (line 13–19). Each agent takes its assignment generated by value propagation as its initial assignment and performs l iterations of local search (line 14). After that, each agent performs k iterations of 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 with local search. Actually, our purpose to perform local search after a value propagation phase is to provide a further optimization on solution quality and produce a positive impact on the

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 backward_direction then
6              perform belief propagation;
7          end
8          else if current_order is forward_direction then
9              x_i ← current optimal decision;
10             perform value propagation using assignment x_i;
11         end
12     end
13     if current_order is forward_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 value propagation using assignment x_i;
18         end
19     end
20     current_order ← reverse(current_order);
21 end
```

Fig. 6 Sketch for Max-sum_ADSSVP with local search

next phase of belief propagation. Max-sum_ADVP, however, is not necessary to be enhanced by local search since it is a monotonic algorithm and is proven to behave like a greedy local search algorithm after the second value propagation phase. Max-sum and Max-sum_AD, on the other hand, cannot be enhanced with local search algorithms since they do not have any value propagation phase. As a result, the assignments generated by local search cannot be applied into a factor graph and thus cannot affect the optimization processes.

The extra overhead of Max-sum_ADSSVP mainly lies on computation. Specifically, a function node in a value propagation phase requires $O(d)$ operations to compute a message by fixing the assignment of the upstream variable node, while a function node in a belief propagation phase requires $O(d^2)$ operations to compute a message by traversing the domain of the upstream variable node for each entry in the message. 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 computing messages to variable nodes in every belief propagation phase after enabling value propagation.

The computational overheads of Max-sum_ADSSVP (t) and Max-sum_ADSSVP with local search are even smaller. For Max-sum_ADSSVP (t), each round comprises several value propagation phases and a belief propagation phase. Thus, the number of belief propagation phases is much reduced if all the algorithms run for the same number of iterations. The same conclusion can be made in Max-sum_ADSSVP with local search due to the existence of refining phases and modification phases. Moreover, a refining phase usually incurs a minor overhead since it is very short, and a modification phase behaves just like a value propagation phase and only incurs $O(d)$ operations when producing a message from a function node. Thus, Max-sum_ADSSVP (t) and Max-sum_ADSSVP with local search have smaller computational overheads than Max-sum_ADSSVP in general.

5.3 Max-sum with hybrid belief/value propagation in a DAG

We notice that in our first algorithm agents can only perform exploration or exploitation in a phase, and thus considerable iterations need to be performed to suppress cost fluctuations during the transitions from value propagation phases to belief propagation phases. Moreover, it can be easily concluded from Observation 1 that variable nodes always make decisions based on the out-of-date assignment information in belief propagation phases after enabling value propagation. That is, variable nodes always take the assignment information received in the previous value propagation phase that (probably) is going to be overridden in this phase into the considerations, which also contributes to the cost fluctuations.

In this section, we try to mitigate the cost fluctuations by making the latest messages available to nodes with the least possible delay, and propose a novel algorithm called Max-sum with hybrid belief/value propagation (Max-sum_HBVP for short). Specifically, instead of executing belief propagation and value propagation alternatively every two phases, we execute value propagation and belief propagation simultaneously from two opposite ends of a DAG in one round. The sketch of the algorithm is presented in Fig. 7.

Each node in the algorithm checks whether it receives all messages from its upstream (or downstream) neighbors. If so, the node sends messages to its downstream nodes (or upstream nodes) provided it hasn't sent messages to the nodes (line 9–13, line 15–18). When the iteration number reaches k, all sending and receiving flags are cleared, and then a new round begins (line 19). The hybrid execution of belief propagation and value propagation is implemented by function nodes. That is, when a function node has received all messages from its upstream neighbors, it performs value propagation to produce messages to its downstream

Algorithm 3: Max-sum_HBVP(node n)

```
1  current_order ←select an order on all nodes in the factor graph;
2  N_n ← the set of indexes of n's neighboring nodes;
3  N_prev_n ← {n̂ ∈ N_n : n̂ is before n in current_order};
4  N_follow_n ← N_n\N_prev_n;
5  while no termination condition is met do
6      for k iterations do
7          collect messages from N_n;
8          if n is a variable node then
9              if n receives all messages from N_prev_n and hasn't sent
               messages to N_follow_n then
10                 compute x_n^* accroding to Eq. (5);
11                 produce the message {x_n^*} to n' using messages received
                   from N_n\{n'}, ∀n' ∈ N_follow_n;
12             if n receives all messages from N_follow_n and hasn't sent
               messages to N_prev_n then
13                 produce the message {q_{n→n'}} to n' using messages received
                   from N_n\{n'}, ∀n' ∈ N_prev_n;
14         else if n is a function node then
15             if n receives all messages from N_prev_n and hasn't sent
               messages to N_follow_n then
16                 produce the message to n' using the constraint,
                   assignments received from N_n\{n'}, ∀n' ∈ N_follow_n;
17             if n receives all messages from N_follow_n and hasn't sent
               messages to N_prev_n then
18                 produce the message to n' using the constraint, messages
                   received from N_n\{n'}, ∀n' ∈ N_prev_n;
19     clear all sending and receiving flags;
```

Fig. 7 Sketch for Max-sum_HBVP

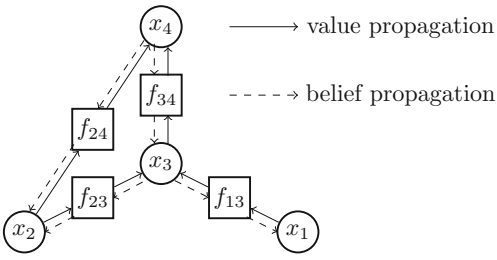


Fig. 8 An execution example of Max-sum_HBVP

neighbors (line 15–16), while it performs belief propagation to produce messages to its upstream neighbors when it has received all messages from its downstream neighbors (line 17–18). Moreover, variable nodes only make decisions when receiving messages from their upstream neighbors in a round (line 10). In this way, a variable node will be able to make its decision based on the latest assignment information in this round, which overcomes the aforementioned defect.

Taking the DCOP instance in Fig. 1 and the DAG in Fig. 2 for an example, Fig. 8 and the following trace demonstrate the execution of Max-sum_HBVP.

Round 1

Iteration 1: $z_1(x_1) = [0, 0] \Rightarrow x_1^* = 0$ $z_2(x_2) = [0, 0] \Rightarrow x_2^* = 0$
$x_1 \rightarrow f_{13} : [0, 0], x_1 = 0$ $x_2 \rightarrow f_{23} : [0, 0], x_2 = 0$
$x_2 \rightarrow f_{24} : [0, 0], x_2 = 0$ $x_4 \rightarrow f_{24} : [0, 0]$
$x_4 \rightarrow f_{34} : [0, 0]$

Iteration 2: $f_{13} \rightarrow x_3 : [1, 7]$ $f_{23} \rightarrow x_3 : [7, 3]$ $f_{24} \rightarrow x_4 : [3, 3]$
$f_{24} \rightarrow x_2 : [3, 1]$ $f_{34} \rightarrow x_3 : [4, 6]$

Iteration 3: $z_3(x_3) = [12, 16] \Rightarrow x_3^* = 0$
$x_3 \rightarrow f_{34} : [8, 10], x_3 = 0$ $x_3 \rightarrow f_{13} : [11, 9]$
$x_3 \rightarrow f_{23} : [5, 13]$

Iteration 4: $f_{34} \rightarrow x_4 : [4, 7]$ $f_{13} \rightarrow x_1 : [12, 12]$ $f_{23} \rightarrow x_2 : [12, 13]$

Iteration 5: $z_4(x_4) = [7, 10] \Rightarrow x_4^* = 0$

Round 2

Iteration 1: $z_1(x_1) = [12, 12] \Rightarrow x_1^* = 0$ $z_2(x_2) = [15, 14] \Rightarrow x_2^* = 1$
$x_1 \rightarrow f_{13} : [0, 0], x_1 = 0$ $x_2 \rightarrow f_{23} : [3, 1], x_2 = 1$
$x_2 \rightarrow f_{24} : [12, 13], x_2 = 1$ $x_4 \rightarrow f_{24} : [4, 7]$
$x_4 \rightarrow f_{34} : [3, 3]$

Iteration 2: $f_{13} \rightarrow x_3 : [1, 7]$ $f_{23} \rightarrow x_3 : [8, 3]$ $f_{24} \rightarrow x_4 : [1, 5]$
$f_{24} \rightarrow x_2 : [7, 5]$ $f_{34} \rightarrow x_3 : [7, 9]$

Iteration 3: $z_3(x_3) = [16, 19] \Rightarrow x_3^* = 0$
$x_3 \rightarrow f_{34} : [9, 10], x_3 = 0$ $x_3 \rightarrow f_{13} : [15, 12]$
$x_3 \rightarrow f_{23} : [8, 16]$

Iteration 4: $f_{34} \rightarrow x_4 : [4, 7]$ $f_{13} \rightarrow x_1 : [16, 15]$ $f_{23} \rightarrow x_2 : [15, 16]$

Iteration 5: $z_4(x_4) = [5, 12] \Rightarrow x_4^* = 0$

Round 3

Iteration 1: $z_1(x_1) = [16, 15] \Rightarrow x_1^* = 1$ $z_2(x_2) = [22, 21] \Rightarrow x_2^* = 1$
$x_1 \rightarrow f_{13} : [0, 0], x_1 = 1$ $x_2 \rightarrow f_{23} : [7, 5], x_2 = 1$
$x_2 \rightarrow f_{24} : [15, 16], x_2 = 1$ $x_4 \rightarrow f_{24} : [4, 7]$
$x_4 \rightarrow f_{34} : [1, 5]$

Iteration 2: $f_{13} \rightarrow x_3 : [9, 3]$ $f_{23} \rightarrow x_3 : [8, 3]$ $f_{24} \rightarrow x_4 : [1, 5]$
$f_{24} \rightarrow x_2 : [7, 5]$ $f_{34} \rightarrow x_3 : [5, 7]$

Iteration 3: $z_3(x_3) = [22, 13] \Rightarrow x_3^* = 1$
$x_3 \rightarrow f_{34} : [17, 6], x_3 = 1$ $x_3 \rightarrow f_{13} : [13, 10]$
$x_3 \rightarrow f_{23} : [14, 10]$

Iteration 4: $f_{34} \rightarrow x_4 : [6, 6]$ $f_{13} \rightarrow x_1 : [14, 13]$ $f_{23} \rightarrow x_2 : [13, 13]$

Iteration 5: $z_4(x_4) = [7, 11] \Rightarrow x_4^* = 0$

It can be seen from the trace that along with the DAG, the algorithm performs value propagation (e.g., $x_1 \rightarrow f_{13}, x_2 \rightarrow f_{23}, x_2 \rightarrow f_{24}$ in iteration 1, $f_{13} \rightarrow x_3$, $f_{23} \rightarrow x_3$, $f_{24} \rightarrow x_4$ in iteration 2, $x_3 \rightarrow f_{34}$ in iteration 3 and $f_{34} \rightarrow x_4$ in iteration 4 of each round), while the algorithm performs belief propagation simultaneously along with the reverse direction of the DAG (e.g., $x_4 \rightarrow f_{24}, x_4 \rightarrow f_{34}$ in iteration 1, $f_{24} \rightarrow x_2$, $f_{34} \rightarrow x_3$ in iteration 2, $x_3 \rightarrow f_{13}, x_3 \rightarrow f_{23}$ in iteration 3 and $f_{13} \rightarrow x_1$, $f_{23} \rightarrow x_2$ in iteration 4 of each round). Moreover, variable nodes who receive all messages from all its upstream neighbors can make decisions. For example, x_3 can only makes its decision in iteration 3 of each round since it hasn't received messages from f_{13} and f_{12} until the end of iteration 2. In this way, x_3 always makes its decision by considering the latest assignments of x_1 and x_2 in a round. As a result,

the optimization process is efficiently boosted and the optimal solution is produced within 3 rounds.

Since the decision-making timings are strictly controlled in Max-sum_HBVP, its belief now depends on its position in a DAG when a variable node makes a decision. Next, we will discuss the decision-making timings and the corresponding beliefs for variable nodes. To begin with, let us introduce some notations in DAGs. We denote the longest path from variable node x_i to x_j as $path_{i,j}$, the set of start nodes and end nodes (i.e., nodes who do not have any upstream or downstream neighbors) as V and the arc from x_i to neighboring function node f_{ij} as (i, ij) in a DAG. Particularly, if x_j is not reachable from node x_i, then $path_{i,j} = \emptyset$ and thus $|path_{i,j}| = 0$.

Proposition 2 *A variable node x_i makes a decision in the $h_i + 1$-th iteration in each round, where $h_i = \max_{v \in V: path_{v,i} \neq \emptyset} |path_{v,i}|$ is the length of the longest path among the paths from start nodes to x_i.*

Proof Let $v_i^* = \arg\max_{v \in V: path_{v,i} \neq \emptyset} |path_{v,i}|$. Without loss of any generality, assume that $(ij, i) \in path_{v_i^*,i}$. It is obvious that the decision-making cannot happen before the $h_i + 1$-th iteration since x_i receives the message from function node f_{ij} no earlier than the h_i-th iteration.

Suppose that the decision-making happens at the $h_i' + 1$-th iteration where $h_i' > h_i$, then there must exist (at least) a function node f_{ik} that sends a message to x_i in the $h_i' - 1$-th iteration. Thus there also must exist (at least) a variable node that sends a message to f_{ik} in the $h_i' - 2$-th iteration. The similar analysis can be applied recursively until the first iteration when (at least) a start node v_i' sends messages to its downstream neighbors. Therefore, there must exist a path from v_i' to x_i whose length is h_i', which is contradictory with the definition to h_i.

Thus, the proposition is proved. □

Since x_i needs to wait for messages from all its upstream neighbors to make a decision, it is always able to consider the latest assignment informations in this round. Moreover, it can be concluded from Proposition 2 that x_i also considers the messages sent in this round from downstream neighbors who send messages before the $h_i + 1$-th iteration in this round, and the messages sent in the last round from the remaining downstream neighbors. Formally, we have the following observation.

Observation 2 *When a variable node x_i is able to make a decision in round m, its belief is given by*

$$z_i(x_i) = \sum_{n \in P_i} f_{ni}(k_n^m, x_i) + \sum_{\substack{n \in S_i: (i,ni) \in path_{i,v} \\ v \in V, |path_{i,v}| > h_i}} r_{f_{ni} \to x_i}^{m-1}(x_i) + \sum_{\substack{n \in S_i: (i,ni) \in path_{i,v} \\ v \in V, |path_{i,v}| \leq h_i}} r_{f_{ni} \to x_i}^{m}(x_i)$$

Consider the simple example shown in Fig. 9. It can be easily observed that the length of the longest path from the start nodes to variable node x_2 is 2. Thus, according to Proposition 2, x_2 makes a decision at the third iteration in round m. Up to the last iteration, x_2 has received the messages from f_{12} and f_{23} but hasn't received the message from f_{24} in this round. Thus, the belief for x_2 when making a decision is

$$z_2(x_2) = f_{12}(k_1^m, x_2) + r_{f_{23} \to x_2}^m(x_2) + r_{f_{24} \to x_2}^{m-1}(x_2)$$

It can be concluded from Observation 2 that similar to Max-sum_ADSSVP, Max-sum_HBVP enables variable nodes to escape from local optima by considering both local

functions and accumulated beliefs. Moreover, since variable nodes in Max-sum_HBVP always make decisions based on valid assignment information, the cost fluctuation is effectively suppressed. Variable nodes also have chances to consider the beliefs produced in the current round from their downstream neighbors, which is impossible in Max-sum_AD, Max-sum_ADVP and Max-sum_ADSSVP.

Max-sum_HBVP generally has smaller overheads than Max-sum_ADVP. Specifically, given the maximum iteration k for each phase or round, the total number of messages exchanged in Max-sum_ADVP is $2k|F|$ in each phase, where F is the set of constraints. However, since each node in Max-sum_HBVP only produces messages twice a round (one for each direction), the total number of messages exchanged in Max-sum_HBVP is just $4|F|$ in each round. Moreover, although computing belief propagation messages to upstream neighbors will incur an extra computational overhead for function nodes, our algorithm still has advantages since the computation is only executed by function nodes once a round.

5.4 Max-sum_AD with probabilistic value propagation

In this section, we balance exploration and exploitation in Max-sum_ADVP via probabilistic value propagation and present an algorithm called Max-sum_AD with probabilistic value propagation (Max-sum_ADPVP). That is, after enabling value propagation, each function node makes a stochastic decision about whether it performs value propagation in the current iteration. Thus, compared to the ones in Max-sum_ADVP, variable nodes in Max-sum_ADPVP have chances to consider beliefs beyond local functions, and hence our algorithm is less sensitive to the value propagation timing which is a major concern in Max-sum_ADPVP. The sketch of the algorithm is presented in Fig. 10.

Similar to Max-sum_ADVP, Max-sum_ADPVP also operates on alternating DAGs. The difference mainly lies on the computation of messages in each function node. Specifically, instead of always considering the assignments of its upstream neighbors in Max-sum_ADVP after enabling value propagation, a function node in our algorithm considers the assignments and performs value propagation when satisfying the probability p (line 12–13). Otherwise, it performs belief propagation (line 14–15).

In fact, the value propagation probability p determines the degree of exploitation. Particularly, when p is set to 1, the algorithm is entirely exploitative and is equivalent to Max-sum_ADVP. In contrast, when p is set to 0, the algorithm has the highest degree of exploration and is equivalent to Max-sum_AD. Thus, considering the impact of p, it is natural to extend the algorithm by adapting p during the optimizing process. In general, we want to encourage the algorithm to perform exploration at the beginning of the optimizing process to find potential promising assignments, while to make the algorithm more exploitative at

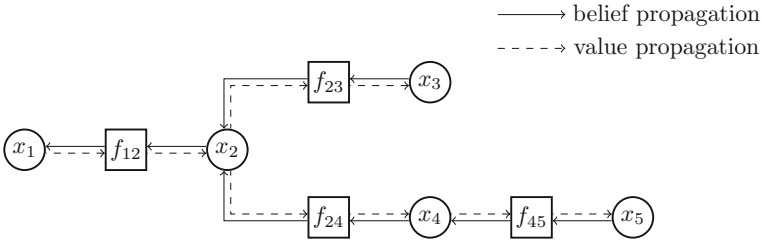


Fig. 9 A simple example of Max-sum_HBVP execution

Algorithm 4: Max-sum_ADPVP(node n, probability p)

```
1  current_order ←select an order on all nodes in the factor graph;
2  N_n ← the set of indexes of n's neighboring nodes;
3  while no termination condition is met do
4      N_{prev_n} ← {n̂ ∈ N_n : n̂ is before n in current_order};
5      N_{follow_n} ← N_n \ N_{prev_n};
6      for k iterations do
7          collect messages from N_n;
8          if n is a variable node then
9              compute x*_n accroding to Eq. (5);
10             produce the message {q_{n→n'}, x*_n} to n' using messages received
               from N_n \ {n'}, ∀n' ∈ N_{follow_n};
11         else if n is a function node then
12             if random()< p then
13                 produce the message to n' using the constraint,
                   assignments received from N_n \ {n'}, ∀n' ∈ N_{follow_n};
14             else
15                 produce the message to n' using the constraint, messages
                   received from N_n \ {n'}, ∀n' ∈ N_{prev_n};
16     current_order ← reverse(current_order);
```

Fig. 10 Sketch for Max-sum_ADPVP

the end of the optimizing process to guarantee solution qualities. Specifically, we propose the following four methods to adapt the value propagation probability.

– *Linear adaptation (LA)* increases the value propagation probability uniformly. Thus, it makes the algorithm transform from the least exploitative one to the most exploitative one uniformly during the whole optimizing process. Formally, the value propagation probability in iteration m is given by

$$p = \frac{m}{Max\_Iter}$$

where Max_Iter is the maximal iteration number.

– *Negative quadratic adaptation (NQA)* quadratically increases the value propagation probability with a monotonically decreased derivative. Thus, the probability increases dramatically at the beginning of the optimizing process and the algorithm behaves more exploitative in most of the iterations. Formally, the value propagation probability in iteration m is given by

$$p = -\left(\frac{m}{Max\_Iter}\right)^2 + \frac{2m}{Max\_Iter}$$

– *Positive quadratic adaptation (PQA)* quadratically increases the value propagation probability with a monotonically increased derivative. Thus, the probability increases slowly at the beginning of the optimizing process, which encourages the algorithm to perform exploration. Formally, the value propagation probability in iteration m is given by

$$p = \left(\frac{m}{Max\_Iter}\right)^2$$

– *Exponential adaptation (EA)* exponentially increases the value propagation probability. Thus, the probability remains at a low level for the most of iterations and grows

dramatically in few iterations at the end of the optimizing process. Formally, the value propagation probability in iteration m is given by

$$p = e^{\frac{m}{Max\_Iter}-1}$$

The overheads of Max-sum_ADPVP fall in between the one of Max-sum_ADVP and the one of Max-sum_AD. Since Max-sum_ADPVP also operates on alternating DAGs and follows the same message-passing scheme, the message number of Max-sum_ADPVP is identical to the ones of Max-sum_AD and Max-sum_ADVP, and thus the extra overheads mainly lie in computation. Specifically, function nodes in Max-sum_ADPVP with a small value propagation probability p are more likely to perform belief propagation in which $O(d^2)$ operations are required to compute a message. Function nodes in Max-sum_ADPVP with a large value propagation probability p are more likely to perform value propagation in which $O(d)$ operations are required to compute a message. Thus, Maxsum_ADPVP with a large value propagation probability usually has small overheads.

6 Experimental evaluations

In this section, we first empirically study how parameters affect our algorithms. Then, we present the solution qualities under different value propagation timings to explicitly demonstrate our algorithms are less sensitive to the timings. Finally, we present comparisons on the performances of our algorithms with DSA, ADPOP, and state-of-the-art Max-sum algorithms.

6.1 Experimental configurations

We benchmark algorithms with four types of problems, i.e., random DCOPs, scale-free networks, weighted graph coloring problems and random meeting scheduling problems.

- *Random DCOPs* are the general form of the distributed constraint optimization problems. In the experiments, we set the agent number to 120, the domain size to 10 and uniformly select costs from [1,100] for each constraint. We consider problems with the graph density 0.05 for the *sparse configuration* and 0.6 for the *dense configuration*.
- *Scale-free networks* [2] are networks whose degree distributions follow power laws. In the experiments, we use Barabási-Albert (BA) model to generate the constraint graph topology with an initial set of 15 agents. At each iteration of BA model procedure, a new agent is connected to 3 other agents (for the sparse configuration) or 10 other agents (for the 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 the ones in random DCOPs.
- *Weighted graph coloring problems* are problems in which each vertex should be colored and two adjacent vertexes should have different colors. In the experiments, we consider weighted graph coloring problems with 120 agents, 3 available color for each agent. The cost for each violation is uniformly selected from 1 to 100. The agent number and graph density are the same as the ones in the sparse configuration of random DCOPs.
- *Random meeting scheduling problems* [15,39] are problems in which agents are trying to schedule a set of meetings. For each pair of meetings, there is a randomly selected travel time. When the difference between the time-slots of two meetings with overlapping agent is less than the travel time, the agents in both meetings are considered to be overbooked

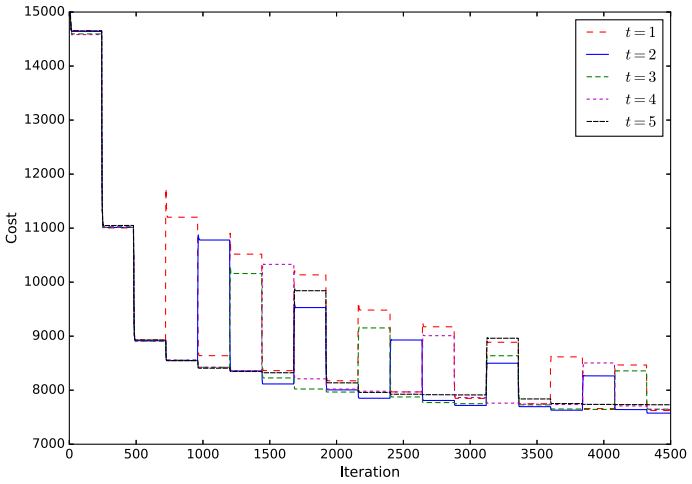


Fig. 11 Solution qualities of Max-sum_ADSSVP (t) under different t on sparse random DCOPs

and the cost is defined to be the number of the overbooked agents. In the experiments, we consider problems with 90 agents, 20 meetings and 20 available time-slots. Each agent randomly chooses two meetings to participate and travel times are uniformly selected from 6 to 10.

The competitors we consider in the experiments include Standard Max-sum, Max-sum_AD, Max-sum_ADVP, Max-sum_ADVP with exploration methods, Damped Max-sum, DSA-C ($p = 0.4$) and ADPOP. We have re-implemented all the algorithms and have checked their performances against the original papers.[6] For Max-sum_ADVP with exploration methods, we consider Max-sum_ADVP with the combination $ADVP\_AD\_ADVP$ which improves Max-sum_ADVP most according to [39] For Damped Max-sum, we set the damped factor to 0.9 according to [4]. For ADPOP, we set $maxDim = 3$. To guarantee convergence, we set the length of a phase (or a round) of Max-sum algorithms on DAGs to 40 (for random meeting scheduling problems) or 240 (for the others), and stop all algorithms after 750 iterations (for random meeting scheduling problems) or 4500 iterations (for the others). In all the Max-sum algorithms, ties are broken by random personal preferences according to [7]. In our experiments, we uniformly select the preferences from $[-0.5, 0.5]$. Besides, for the algorithms that use value propagation and operate on DAGs, we start value propagation at the beginning of the third phase unless otherwise specified. All experimental results are averaged over 50 independently generated problems that are each solved by each algorithm 30 times.

6.2 Parameters tuning

6.2.1 Suitable parameters for Max-sum_ADSSVP-based algorithms

In this section, we explore the relationships between the solution qualities and the parameters for Max-sum_ADSSVP-based algorithms, i.e., the length of consecutive value propagation phase t for Max-sum_ADSSVP (t) and the local search algorithm for Max-sum_ADSSVP with local search.

[6] https://github.com/czy920/DCOPSovler

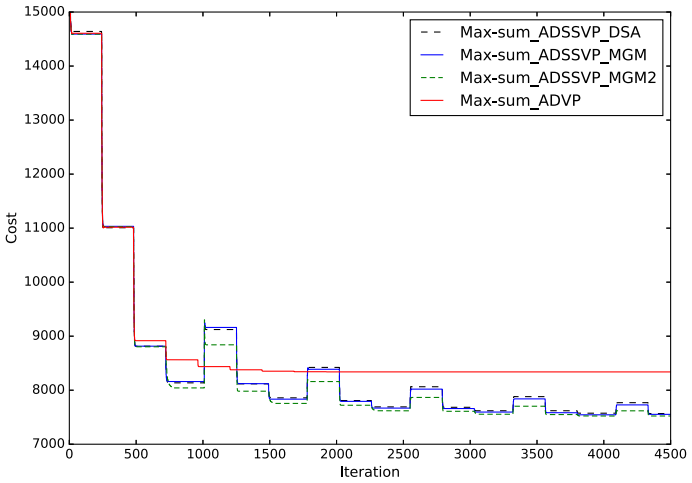


Fig. 12 Solution qualities of Max-sum_ADSSVP with different local search algorithms and Max-sum_ADVP on sparse random DCOPs

We first vary t from 1 to 5 and the corresponding solution qualities are shown in Fig. 11. It can be seen from the figure that the solution qualities of algorithms with small t oscillate wildly during the transitions from value propagation phases to belief propagation phases (e.g., the 720-th iteration and the 1200-th iteration for $t = 1$, the 960-th iteration for $t = 2$ etc.). The algorithms with large t (e.g., $t = 4$ or $t = 5$), on the other hand, usually have moderate transitions. For example, the first transition of Max-sum_ADSSVP ($t = 1$) occurs in the 720-th iteration and the corresponding amplitude is 14.6%, while the first transition of Max-sum_ADSSVP ($t = 5$) occurs in the 1680-th iteration and the corresponding amplitude is only 10.2%. However, Fig. 11 also indicates that the algorithms with large t usually could be dominated by the algorithms with smaller t. This is because the large t makes agents less opportunities to perform exploration. For example, agents in Max-sum_ADSSVP ($t = 5$) only perform 3 phases of belief propagation after enabling value propagation, while agents in Max-sum_ADSSVP ($t = 1$) perform 9 phases of belief propagation. Thus, according to the result, we choose $t = 2$ for Max-sum_ADSSVP (t).

We compare the solution qualities of Max-sum_ADSSVP with DSA, MGM and MGM2. We set the length of the refining phase to 50 and the result is presented in Fig. 12. It can be seen from the result that all the three combinations outperform Max-sum_ADVP and the solution qualities are similar when DSA and MGM are applied into Max-sum_ADSSVP. MGM2, on the other hand, provides more substantial improvements every refining phase and yields more modest transitions, which also demonstrates that the solution quality at the end of a value propagation phase has a critic impact on the next phase of belief propagation. Thus, we choose MGM2 as the refiner for Max-sum_ADSSVP with local search in our experiments.

6.2.2 Suitable adaptation for Max-sum_ADPVP

In this section, we empirically study the solution qualities of Max-sum_ADPVP with different adaptations. To demonstrate the adaptation of the value propagation probability, we start value propagation at the first iteration and present the result in Fig. 13.

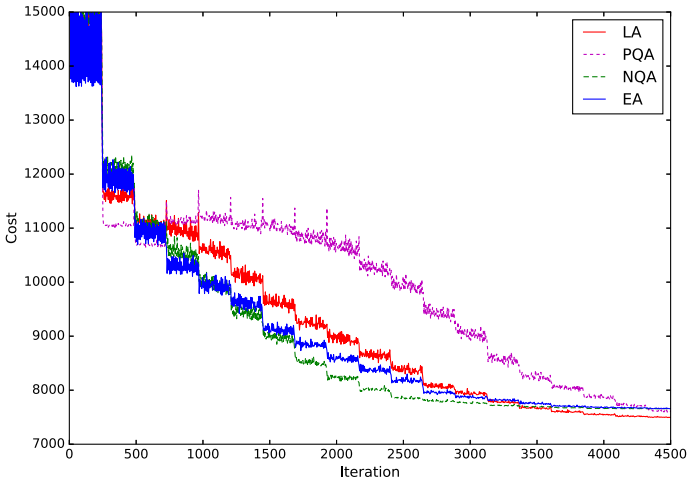


Fig. 13 Solution qualities of Max-sum_ADPVP with different adaptations on sparse random DCOPs

It can be observed that the solution qualities of adaptations except exponential adaptation oscillate modestly at the beginning of the execution. That is because the value propagation probabilities in those adaptations are very small at the beginning of execution. Since the value propagation probability in negative quadratic adaptation is strictly greater than the one in linear adaptation at any iteration other than the first and the last iteration, the algorithm with NQA is more exploitative and converges earlier than the one with LA. Similarly, the value propagation probability in positive quadratic adaptation is strictly less than the one in linear adaptation in most of the iterations, thus the algorithm with PQA performs more exploration and converges slowly than the one with LA. Exponential adaptation, on the other hand, retains the value propagation probability at a low level in most of the iterations and dramatically increases the probability at the end of the execution. The unbalance makes the solution quality oscillate wildly and eventually yields a poor solution. Thus, in our experiment we use linear adaptation to adapt Max-sum_ADPVP.

6.2.3 Value propagation timings for our proposed algorithms

In this section, we demonstrate our proposed algorithms (i.e., Max-sum_ADSSVP (t), Max-sum_ADSSVP with local search and Max-sum_ADPVP) are less sensitive to value propagation timings which is a critic impact on the solution qualities of Max-sum_ADVP. We do not include Max-sum_HBVP here since the algorithm requires value propagation from the first iteration and hence the value propagation timing is not adjustable. We vary the value propagation timing from the first phase to the fifth phase and present the anytime results in Fig. 14. All the parameters are set according to Sects. 6.2.1 and 6.2.2.

It can be seen from the figure that our algorithms have similar performances under different value propagation timings while the solution qualities of Max-sum_ADVP vary a lot, which indicates our algorithms are less sensitive to value propagation timings and thus overcome the drawback of timing selection in Max-sum ADVP. That is because agents in our algorithms still have opportunities to make decisions beyond local constraints after enabling value propagation. As a result, agents could break out of local optima and alleviate some negative impacts brought by the assignments of their neighbors. Max-sum_ADVP, on the other

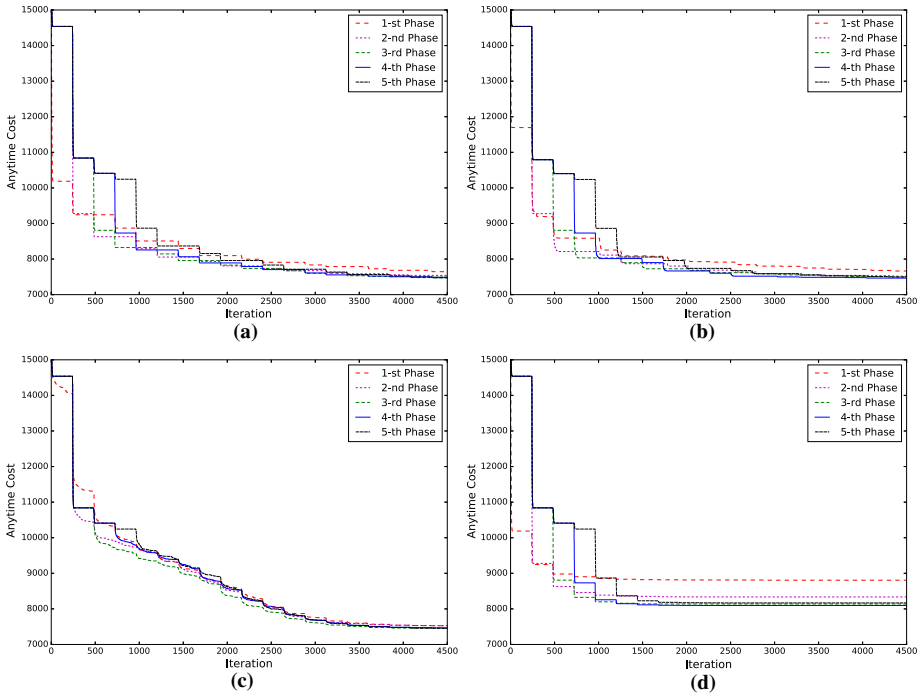


Fig. 14 Anytime solution qualities of the algorithms under different value propagation timings on sparse random DCOPs: **a** anytime solution qualities of Max-sum_ADSSVP($t = 2$), **b** anytime solution qualities of Max-sum_ADSSVP with MGM2, **c** anytime solution qualities of Max-sum_ADPVP with linear adaptation, **d** anytime solution qualities of Max-sum_ADVP

hand, monotonically optimizes solutions after the first convergence of value propagation. Thus, agents in Max-sum_ADVP now make decisions entirely based on local constraints and the assignments of their neighbors. Consequently, Max-sum_ADVP is sensitive to initial assignments, as well as the value propagation timings. Besides, it is interesting to find that all the algorithms report the best results when starting value propagation at the 4-th phase. But note that the best timing of starting value propagation is problem-specific and usually presented as an empirical value.

6.3 Performance comparisons

In this section, we consider the performances of the algorithms. Particularly, we consider the solution qualities and runtime when the algorithms terminate. Tables 1 and 2 give the solution qualities and runtime (in milliseconds) of these algorithms on different benchmarks, respectively.

Regarding the solution quality, our proposed Max-sum algorithms with non-consecutive value propagation strategies produce better solutions than the other algorithms in each benchmarks. Specifically, our algorithms improve Max-sum_ADVP by 9.1–10.5% and 10.1–12.0% when solving the sparse random DCOPs and scale-free networks. For the dense problems, however, the differences among these algorithms are narrowed. That is partially due to the high cost base in over-constrained problems. Despite the complexity of the dense problems,

our proposed algorithms still improve Max-sum_ADVP by 0.8–1.1% and 2.8–4.1% when solving the dense random DCOPs and scale-free networks.

Besides, our proposed algorithms exhibit excellent performances when solving problems with structured cost functions. It is noticeable that our algorithms outperform Max-sum_ADVP by 48.1–53.3% when solving weighted graph coloring problems. That is because Max-sum_ADVP is biased by local benefits which could severely conflict with the global benefit in those problems. It is worth mentioning that Max-sum_ADSSVP_MGM2 is slightly inferior to Max-sum_ADVP when solving random meeting scheduling problems. That is because Max-sum_ADSSVP_MGM2 was performing belief propagation before it terminated. As a result, it reports the solutions which are not optimized by value propagation phase and local search phase. Additionally, it is interesting to find that Max-sum_HBVP reports the best solutions among the competitors in the most benchmarks, which demonstrates the power of hybrid execution of value propagation and belief propagation. That is because agents in the algorithm always make decisions by considering the valid assignment information, and thus the optimization process is efficiently boosted.

It can be seen from Table 2 that Standard Max-sum and Damped Max-sum spend much more time than the other Max-sum algorithms in every benchmark. That is because the agents in these two algorithms send messages to all their neighbors. As a local search algorithm, DSA require less time than the other iterative algorithms. On the other hand, since ADPOP is a one-shot algorithm, it spends the least time in most of the benchmarks. However, ADPOP requires the most time when solving random meeting scheduling problems. That is because the domain size in the problems is relatively large (20 available time slots). As a consequence, the algorithm needs more operations to perform variable elimination. Additionally, it can be concluded that our algorithms excepts Max-sum_HBVP spend time as much as Max-sum_ADVP, which indicates that our algorithms incur modest extra overheads. Max-sum_HBVP requires little time even in the dense problems. That is because each node in Max-sum_HBVP only produces messages twice a round, and thus the total number of messages exchanged in a round is linear to the number of constraints.

It can be concluded from Tables 1 and 2 that our proposed algorithms are suitable for the scenarios where solution quality is an important concern but the computation and communication resources are limited (e.g., coordinating low-power embed devices). In more detail, although DSA and ADPOP require little time in the most of cases, they fail to produce high-quality solutions. Max-sum_ADVP and Max-sum_ADVP with the combination *ADVP_AD_ADVP*, on other hands, produce better solutions than DSA and ADPOP in some cases, but incur more computation overheads. In contrast, as we discussed earlier, our proposed algorithms have smaller overheads than Max-sum_ADVP and produce solutions which are significantly better than the ones produced by DSA and ADPOP. Besides, it is worth mentioning that Max-sum_HBVP actually has a smaller communication overhead than DSA in practice, since the number of messages exchanged in a *round* of Max-sum_HBVP is only a double of the one in an *iteration* of DSA.

6.4 Convergence analyses

In this section, we consider the convergence behavior of each algorithm on different benchmarks, i.e., solution quality per iteration and anytime result per iteration. Figures 15 and 16 present the comparisons of solution qualities and anytime results on the sparse random DCOPs. We omit the dense configuration for the similar trend.

Table 1 Solution qualities of algorithms on different benchmarks

Algorithms	Random DCOPs		Scale-free networks		Weighted graph coloring	Random meeting scheduling
	Sparse	Dense	Sparse	Dense		
Standard Max-sum	15317	209268	13873	49866	13946	1229
Max-sum_AD	11121	189040	9629	40434	2808	393
Max-sumADVP	8338	179325	7525	36136	812	245
ADVP_AD_ADVP	8408	180503	7220	35508	1290	373
Damped Max-sum	8663	182564	7989	38900	2239	350
DSA	8751	179348	7926	36275	891	263
ADPOP	9047	186444	7785	38169	1003	276
Max-sum_ADSSVP ($t = 2$)	7582	177814	6760	35118	413	216
Max-sum_ADSSVP_MGM2	7520	178017	6676	34982	385	247
Max-sum_HBVP	**7465**	**177317**	**6620**	**34638**	**379**	224
Max-sumADPVP	7475	177450	6635	34810	421	**212**

The best results are shown in bold

Table 2 Runtime (in milliseconds) of algorithms on different benchmarks

Algorithms	Random DCOPs		Scale-free networks		Weighted graph coloring	Random meeting scheduling
	Sparse	Dense	Sparse	Dense		
Standard Max-sum	8958	43262	9327	12211	8671	641
Max-sum_AD	8424	23908	8754	9892	8107	456
Max-sumADVP	7968	22178	8427	9750	8386	407
ADVP_AD_ADVP	8188	26930	8597	9511	8118	498
Damped Max-sum	8818	40598	9158	11206	8537	952
DSA	1887	2450	2021	2066	1838	351
ADPOP	603	1073	581	724	635	2712
Max-sum_ADSSVP ($t = 2$)	7936	22593	8576	9578	8506	436
Max-sum_ADSSVP_MGM2	8437	22984	8382	9725	7650	421
Max-sum_HBVP	7297	8999	7462	7713	7616	382
Max-sumADPVP	8213	22572	8604	9498	8089	469

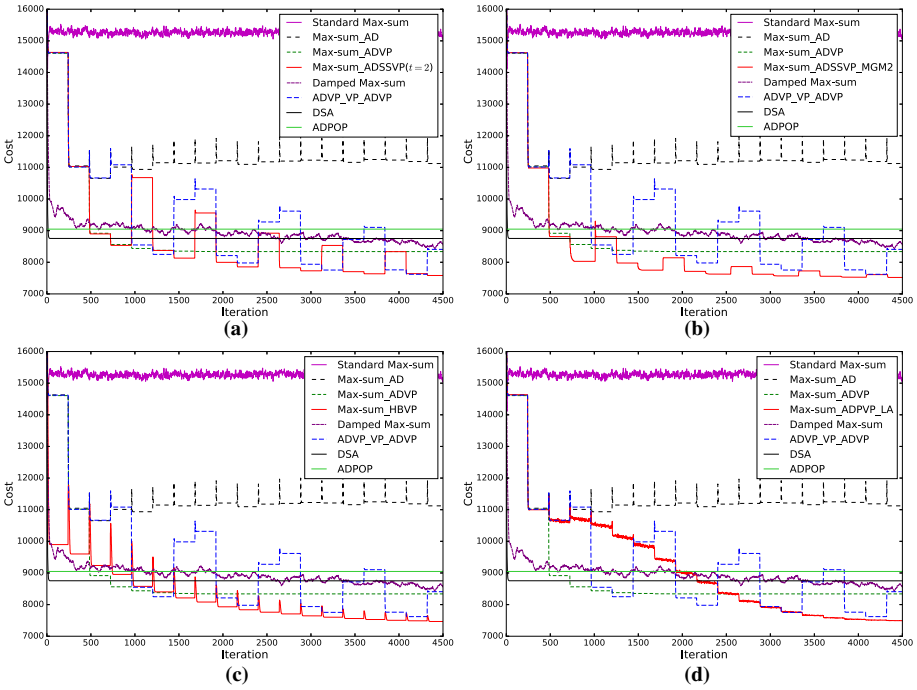


Fig. 15 Solution qualities on sparse random DCOPs

It can be seen from the results that Standard Max-sum does not converge and traverses states with low qualities. However, although it can guarantee the convergence within a single phase, Max-sum_AD also fails to produce a high quality solution. That is because neither Standard Max-sum nor Max-sum_AD can eliminate invalid assignment assumptions and break ties in beliefs. Max-sum_ADVP performs value propagation after the second convergence (i.e., the 481-st iteration) and monotonically optimizes the solution after the 721-st iteration. Since value propagation can eliminate invalid assignment assumptions and break ties in beliefs, Max-sum_ADVP eventually yields a relative higher quality solution. However, Max-sum_ADVP cannot perform any further optimization after the end of the 6-th phase (i.e., 1480-th iteration), which indicates that the algorithm has got trapped in local optima. Max-sum_ADVP with the combination $ADVP\_AD\_ADVP$ tries to mitigate the problem by taking advantage of the differing balance of exploration and exploitation in Max-sum_AD and Max-sum_ADVP. Unfortunately, the algorithm fails to suppress cost fluctuations effectively and the solution quality still oscillates wildly even at the end of the execution. That is due to the fact that the algorithm always performs two consecutive belief phases after two value propagation phases and thus agents cannot utilize any assignment information in the later belief propagation phase. Damped Max-sum attempts to improve Standard Max-sum by decreasing the effect of cyclic information propagation. However, the results indicate that Damped Max-sum is still inferior to Max-sum_ADVP. DSA converges very quickly since agents make decisions in parallel by considering only local information. As a result, DSA cannot further improve its solutions after 50 iterations. ADPOP finds the solutions that are better than the ones in Standard Max-sum and Max-sum_AD, but it is still inferior to DSA.

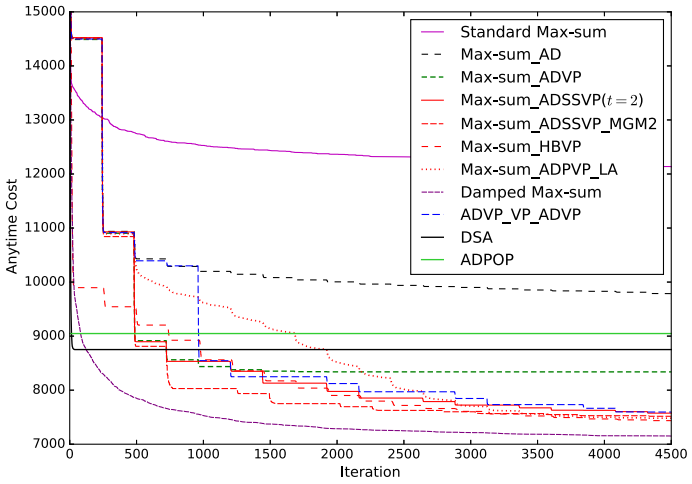


Fig. 16 Anytime solution qualities on sparse random DCOPs

In contrast, our algorithms achieve good balances between exploration and exploitation and substantially improve the solution quality of Max-sum_ADVP. It can be observed from Fig. 15a, b that Max-sum_ADSSVP ($t = 2$) and Max-sum_ADSSVP with MGM2 iteratively refine the solutions after enabling value propagation. It is notable that Max-sum_ADSSVP with MGM2 eventually outperforms all the competitors after about 1250 iterations. Moreover, it is worth mentioning that although both Max-sum_ADVP with the combination $ADVP\_AD\_ADVP$ and our proposed Max-sum_ADSSVP cannot guarantee the cross-phase convergence, the cost fluctuations in our algorithms are much smaller than the ones in Max-sum_ADVP with the combination $ADVP\_AD\_ADVP$. Besides, there are significant decreases of the amplitudes of the cost fluctuations, which also indicates that the belief propagation phases and the value propagation phases in our algorithms can collaborate with each other to effectively suppress the cost fluctuations. It can be seen from Fig. 15c that Max-sum_HBVP transcends all the competitors after about 2400 iterations. Besides, Fig. 15c also indicates that Max-sum_HBVP can guarantee the single-phase convergence, which is not surprising since we have imposed a constraint on the decision-making timing such that each variable node only makes a decision in a round. According to Fig. 15d, Max-sum_ADPVP_LA performs close to Max-sum_AD in the first 960 iterations and then improves dramatically and finally outperforms all the competitors after the 2600-th iteration. This is because the value propagation probability is very small in the beginning of the execution and the algorithm behaves like Max-sum_AD. With the probability growing, the algorithm becomes more and more exploitative and finally behaves like Max-sum_ADVP.

It can be concluded that Damped Max-sum outperforms all other algorithms when considering anytime results in Fig. 16. That is because damping triggers efficient exploration by Max-sum, and the good solutions are cached by the anytime mechanism in time. However, it does not mean that Damped Max-sum with anytime mechanism is suitable for all scenarios. In fact, anytime mechanism requires extra communication and storage overheads to construct BFS trees and retain the best assignments. Moreover, since a node in the algorithm produces messages to every neighbor, Damped Max-sum incurs much more overheads than Max-sum variants operating on DAGs. In other words, Damped Max-sum with anytime mechanism is not suitable for the scenarios where computation and communication resources are lim-

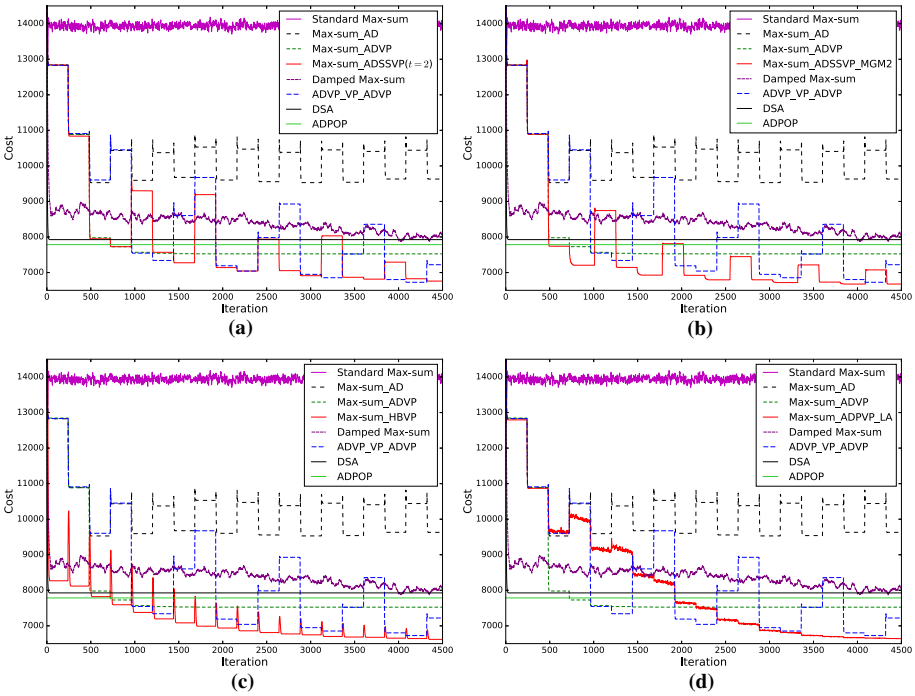


Fig. 17 Solution qualities on sparse scale-free networks

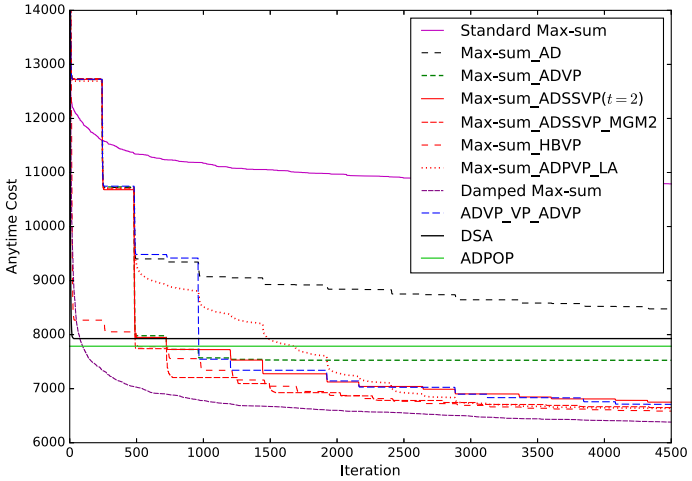


Fig. 18 Anytime solution qualities on sparse scale-free networks

ited. Figure 16 also shows that there are noticeable improvements every value propagation phase and refining phase, which indicates that our proposed Max-sum_ADSSVP algorithms achieve a good balance between exploration and exploitation. Moreover, It is worth mentioning that Max-sum_ADSSVP_MGM2 outperforms all the competitors except Damped Max-sum after 800 iterations, which demonstrates the power of hybrid execution of local search and value propagation.

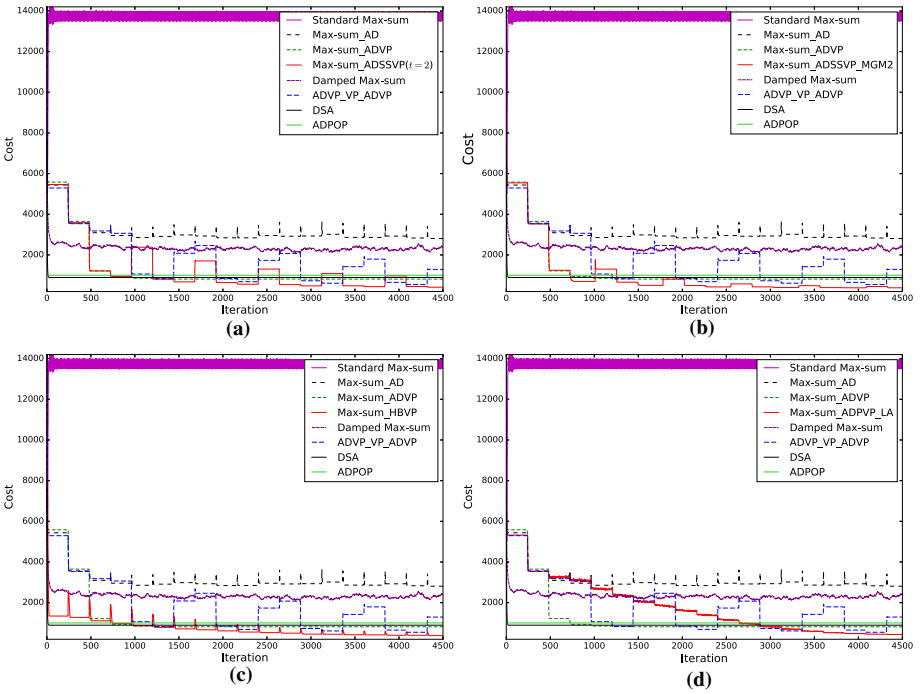


Fig. 19 Solution qualities on weighted graph coloring problems

Figures 17 and 18 present the solution qualities and corresponding anytime solution qualities when solving the sparse scale-free network problems. Here, we also omit the dense configuration for the similar trend. It can be seen from the figures that Standard Max-sum still performs poor due to the existences of invalid assumptions and ties in beliefs. Max-sum_AD finds the solutions with lower costs than Standard Max-sum in the first two convergences, but its solution qualities oscillate in a broad range in the third and the subsequent phases. DSA converges quickly and produces the solutions that are slightly inferior to ADPOP. Max-sum_ADVP finds the solutions with much higher qualities than Max-sum_AD, further provides monotonic optimizations to the solutions found in the end of the third phase, and outperforms DSA at the end of the execution. During the whole optimization process, Damped Max-sum strictly dominates all the algorithms except Max-sum_HBVP and DSA with the help of the anytime mechanism, but it is still inferior to Max-sum_ADVP in terms of the current costs. Max-sum_ADVP with the combination $ADVP\_AD\_ADVP$ finds better solutions than Max-sum_ADVP at the sixth phase and the subsequent ADVP phases, but it fails to suppress the cost oscillations in the transitions from ADVP phases to AD phases. Max-sum_ADSSVP ($t = 2$) and Max-sum_ADSSVP with MGM2 find the solutions which are superior to the ones of Max-sum_ADVP after the 1440-th iteration and the 770-th iteration under the anytime mechanism. Max-sum_HBVP transcends Max-sum_ADVP after 4 hybrid executions of belief propagation and value propagation. The stochastic nature of Max-sum_ADPVP allows variable nodes to consider information beyond local constraints, and thus Max-sum_ADPVP_LA finds better anytime solutions than Max-sum_ADVP after the 1920-th iteration.

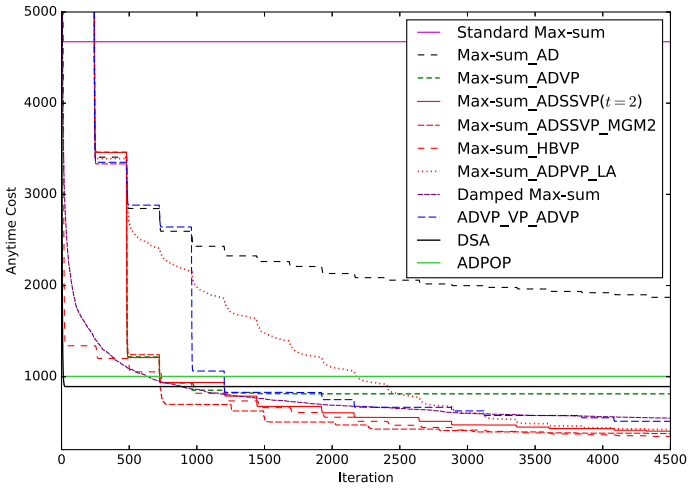


Fig. 20 Anytime solution qualities on weighted graph coloring problems

Figures 19 and 20 present the results and anytime results when solving weighted graph coloring problems. With the help of the random personal preferences, Standard Max-sum and Damped Max-sum can break ties and find better solutions than the other competitors that operate on alternating DAGs at the beginning of the execution. However, as we discuss in Sect. 4.3, agents in Standard Max-sum cannot propagate any useful information but the personal preferences. As a result, agents in Standard Max-sum arbitrarily make decisions and the algorithm performs poorly in the subsequent iterations. Although Max-sum_AD also uses the random personal preferences to break ties, it performs much better than Standard Max-sum, which indicates that Max-sum_AD is more exploitative than Standard Max-sum. Max-sum_ADVP excels Max-sum_AD after enabling value propagation, but soon gets stuck in local optima. It can be seen that Max-sum_ADVP produces the solutions which have the similar qualities with the ones in DSA at the end of the execution, which also demonstrates Max-sum_ADVP eventually behaves like a greedy local search algorithm. Max-sum_ADVP with combination $ADVP\_AD\_ADVP$ finds better solutions than Max-sum_ADVP at the tenth phase, but it fails to provide further optimization in the subsequent Max-sum_ADVP phases. Max-sum_ADSSVP ($t = 2$) and Max-sum_ADSSVP with MGM2 find the anytime results better than the other competitors after the 1440-th iteration and the 770-th iteration. Besides, it is worth mentioning that Max-sum_HBVP finds the anytime solutions better and more quickly than all the competitors except DSA and ADPOP at the beginning of the execution, which indicates that value propagation can break ties more effectively than the random personal preferences. Max-sum_ADPVP_LA optimizes the solutions by gradually increasing the value propagation probability, and outperforms all the competitors after the 3120-th iteration.

Figures 21 and 22 present the results and anytime results when solving random meeting scheduling problems. Since the cost functions in the problems are also highly structured, belief ties make Standard Max-sum barely optimize the solutions. In contrast, Max-sum_AD produces much better solutions by strictly controlling the loopy information propagation. Max-sum_ADVP finds better solutions than Standard Max-sum, Max-sum_AD and Damped Max-sum at the third phase, slowly optimizes the solutions in the subsequent phases, and eventually outperforms DSA and ADPOP. $ADVP\_VP\_ADVP$ tries to improve Max-sum_ADVP by combining Max-sum_AD and Max-sum_ADVP. However, it can be seen

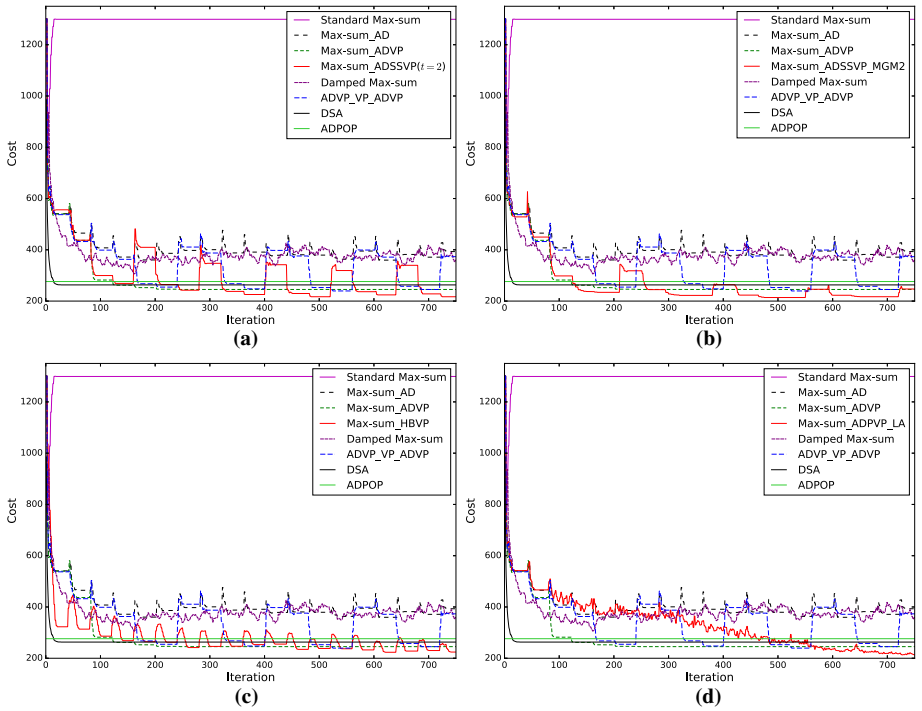


Fig. 21 Solution qualities on random meeting scheduling problems

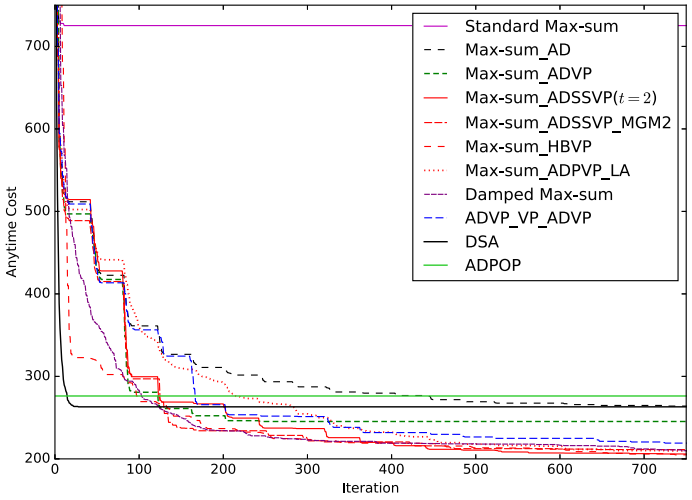


Fig. 22 Anytime solution qualities on random meeting scheduling problems

from Fig. 21 that the algorithm only outperforms Max-sum_ADVP at the 6-th VP phase (i.e., from the 520-th iteration to the 560-th iteration). Our proposed Max-sum_ADSSVP algorithms iteratively refine the solutions after enabling value propagation, and excel Max-sum_ADVP in terms of anytime results after 240 iterations and 150 iterations, respectively.

Additionally, Max-sum_HBVP outperforms all Max-sum algorithms at the beginning of the execution, which shows that value propagation can effectively break ties. As the algorithm proceeds, Max-sum_ADPVP_LA uniformly transforms from the most explorative one to the most exploitative one, and finally outperforms Max-sum_ADVP after 580 iterations.

7 Conclusion

In this paper, we first theoretically analyze how value propagation affects the Max-sum_AD algorithm. We prove that although value propagation can greatly improve the solution qualities of Max-sum_AD, it blocks belief propagation and thus agents in Max-sum_ADVP can only consider local constraints and follow the decision-making strategy of a greedy local search algorithm. As a result, Max-sum_ADVP becomes entirely exploitative after the second phase of value propagation. To overcome the defect, we propose several Max-sum algorithms with novel non-consecutive value propagation strategies which can balance exploration and exploitation.

Our first attempt is to alternatively execute Max-sum_AD and Max-sum_ADVP and we propose an algorithm called Max-sum_ADSSVP. Different from the two-phase execution in Max-sum_ADVP, value propagation in Max-sum_ADSSVP is always executed in the same direction. Unfortunately, we notice that the algorithm no longer guarantees monotonicity and the cross phase convergence, and the solution qualities oscillates wildly during the transitions from value propagation phases to belief propagation phases in practice. We alleviate the pathology by providing higher-quality solutions before switching to belief propagation phases, and propose two algorithms called Max-sum_ADSSVP (t) and Max-sum_ADSSVP_LS. They improve the solution quality by multiple phases of value propagation and local search, respectively. Besides, we find that the out-of-date assignment information also contributes to the oscillation. We overcome the defect by proposing a novel algorithm called Max-sum_HBVP in which value propagation and belief propagation are executed simultaneously from two opposite ends of a DAG and each variable node only makes a decision when receiving value propagation messages from all its upstream neighbors in a round. We finally use stochasticity to balance exploration and exploitation, and present an algorithm called Max-sum_ADPVP. In the algorithm, each function node stochastically decides to perform value propagation or belief propagation according to a probability p. Besides, we further extend the algorithm by adapting p along with the optimization process via four adaptations. Our empirical evaluations have demonstrated the superiorities of our proposed algorithms on various benchmarks.

We notice that personal preferences and value propagation are two main techniques to break ties in Max-sum algorithms. However, as we have demonstrated earlier, personal preferences cannot effectively break ties in most cases. Value propagation, on the other hand, could block belief propagation and make agents consider only local functions. In the future, we will explore other methods to effectively break ties and still keep a balance between exploration and exploitation.

References

1. Aji, S. M., & McEliece, R. J. (2000). The generalized distributive law. *IEEE Transactions on Information Theory*, *46*(2), 325–343.

2. Barabási, A. L., & Albert, R. (1999). Emergence of scaling in random networks. *Science*, *286*(5439), 509–512.
3. Chen, Z., Deng, Y., & Wu, T. (2017). An iterative refined Max-sum_ad algorithm via single-side value propagation and local search. In *Proceedings of the 16th conference on autonomous agents and multiagent systems* (pp. 195–202). International Foundation for Autonomous Agents and Multiagent Systems.
4. Cohen, L., & Zivan, R. (2017). Max-sum revisited: The real power of damping. In *Proceedings of the 16th conference on autonomous agents and multiagent systems* (pp. 1505–1507). International Foundation for Autonomous Agents and Multiagent Systems.
5. Dechter, R. (1999). Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, *113*(1), 41–85.
6. Enembreck, F., & Barths, J. P. A. (2012). Distributed constraint optimization with mulbs: A case study on collaborative meeting scheduling. *Journal of Network and Computer Applications*, *35*(1), 164–175.
7. Farinelli, A., Rogers, A., Petcu, A., & Jennings, N. R. (2008). 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* (Vol. 2, pp. 639–646). International Foundation for Autonomous Agents and Multiagent Systems.
8. Gershman, A., Meisels, A., & Zivan, R. (2009). Asynchronous forward bounding for distributed cops. *Journal of Artificial Intelligence Research*, *34*, 61–88.
9. Hirayama, K., & Yokoo, M. (2005). The distributed breakout algorithms. *Artificial Intelligence*, *161*(1), 89–115.
10. Katagishi, H., & Pearce, J. P. (2007). Kopt: Distributed dcop algorithm for arbitrary k-optima with monotonically increasing utility. In *Ninth DCR workshop* (CP-07).
11. Kschischang, F. R., Frey, B. J., & Loeliger, H. A. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, *47*(2), 498–519.
12. Leite, A. R., Enembreck, F., & Barthès, J. P. A. (2014). Distributed constraint optimization problems: Review and perspectives. *Expert Systems with Applications*, *41*(11), 5139–5157.
13. Litov, O., & Meisels, A. (2017). Forward bounding on pseudo-trees for dcops and adcops. *Artificial Intelligence*, *252*, 83–99.
14. Maheswaran, R. T., Pearce, J. P., & Tambe, M. (2004). Distributed algorithms for dcop: A graphical-game-based approach. In *ISCA PDCS* (pp. 432–439).
15. Meisels, A., & Lavee, O. (2004). Using additional information in discsp search. In *Distributed constraint reasoning workshop (DCR)*.
16. Modi, P. J., Shen, W. M., Tambe, M., & Yokoo, M. (2005). Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, *161*(1), 149–180.
17. Netzer, A., Grubshtein, A., & Meisels, A. (2012). Concurrent forward bounding for distributed constraint optimization problems. *Artificial Intelligence*, *193*, 186–216.
18. Nguyen, D. T., Yeoh, W., & Lau, H. C. (2013). Distributed gibbs: A memory-bounded sampling-based dcop algorithm. In *Proceedings of the 12th international conference on autonomous agents and multi-agent systems* (pp. 167–174). International Foundation for Autonomous Agents and Multiagent Systems.
19. Okimoto, T., Joe, Y., Iwasaki, A., Yokoo, M., & Faltings, B. (2011). Pseudo-tree-based incomplete algorithm for distributed constraint optimization with quality bounds. In *International conference on principles and practice of constraint programming* (pp. 660–674). Springer.
20. Ottens, B., Dimitrakakis, C., & Faltings, B. (2017). Duct: An upper confidence bound approach to distributed constraint optimization problems. *ACM Transactions on Intelligent Systems and Technology*, *8*(5), 69.
21. Pearce, J. P., & Tambe, M. (2007). Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In *International joint conference on artifical intelligence* (pp. 1446–1451)
22. Petcu, A., & Faltings, B. (2005). Approximations in distributed optimization. In *International conference on principles and practice of constraint programming* (pp. 802–806). Springer.
23. Petcu, A., & Faltings, B. (2005). A scalable method for multiagent constraint optimization. In *Proceedings of the 19th international joint conference on artificial intelligence* (pp. 266–271).
24. Petcu, A., & Faltings, B. (2006). Odpop: An algorithm for open/distributed constraint optimization. In *Proceedings of the 21st national conference on artificial intelligence* (pp. 703–708). AAAI Press.
25. Petcu, A., & Faltings, B. (2007). Mb-dpop: A new memory-bounded algorithm for distributed optimization. In *Proceedings of the 20th international joint conference on artifical intelligence* (pp. 1452–1457). Morgan Kaufmann Publishers Inc.
26. Petcu, A., & Faltings, B. (2008). Distributed constraint optimization applications in power networks. *International Journal of Innovations in Energy Systems and Power*, *3*(1), 1–12.
27. Rogers, A., Farinelli, A., Stranders, R., & Jennings, N. R. (2011). Bounded approximate decentralised coordination via the Max-sum algorithm. *Artificial Intelligence*, *175*(2), 730–759.

28. Rollon, E., & Larrosa, J. (2012). Improved bounded Max-sum for distributed constraint optimization. In *Proceedings of the 18th international conference on principles and practice of constraint programming* (Vol. 7514, pp. 624–632). Berlin, Heidelberg: Springer.
29. Rollon, E., & Larrosa, J. (2014). Decomposing utility functions in bounded Max-sum for distributed constraint optimization. In *International conference on principles and practice of constraint programming* (pp. 646–654). Springer.
30. Steven, O., Roie, Z., & Aviv, N. (2016). Distributed breakout: Beyond satisfaction. In *Proceedings of the twenty-fifth international joint conference on artificial intelligence* (pp. 447–453).
31. Sultanik, E., Modi, P. J., & Regli, W. C. (2007). On modeling multiagent task scheduling as a distributed constraint optimization problem. In *IJCAI* (pp. 1531–1536).
32. Vinyals, M., Rodriguez-Aguilar, J. A., & 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* (Vol. 2, pp. 1239–1240). International Foundation for Autonomous Agents and Multiagent Systems.
33. Yeoh, W., Felner, A., & Koenig, S. (2010). Bnb-adopt: An asynchronous branch-and-bound dcop algorithm. *Journal of Artificial Intelligence Research*, *38*, 85–133.
34. Yokoo, M., Durfee, E. H., Ishida, T., & Kuwabara, K. (1998). The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on knowledge and data engineering*, *10*(5), 673–685.
35. Yu, Z., Chen, Z., He, J., & Deng, Y. (2017). A partial decision scheme for local search algorithms for distributed constraint optimization problems. In *Proceedings of the 16th conference on autonomous agents and multiagent systems* (pp. 187–194). International Foundation for Autonomous Agents and Multiagent Systems.
36. Zhang, W., Wang, G., Xing, Z., & Wittenburg, L. (2005). Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, *161*(1), 55–87.
37. Zilberstein, S. (1996). Using anytime algorithms in intelligent systems. *Ai Magazine*, *17*(3), 73–83.
38. Zivan, R., Okamoto, S., & Peled, H. (2014). Explorative anytime local search for distributed constraint optimization. *Artificial Intelligence*, *212*, 1–26.
39. Zivan, R., Parash, T., Cohen, L., Peled, H., & Okamoto, S. (2017). Balancing exploration and exploitation in incomplete min/max-sum inference for distributed constraint optimization. *Autonomous Agents and Multi-Agent Systems*,. https://doi.org/10.1007/s10458-017-9360-1.
40. Zivan, R., & 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* (Vol. 1, pp. 265–272). International Foundation for Autonomous Agents and Multiagent Systems.