Auton Agent Multi-Agent Syst (2017) 31:1165–1207
DOI 10.1007/s10458-017-9360-1

Balancing exploration and exploitation in incomplete Min/Max-sum inference for distributed constraint optimization

Roie Zivan[1] · **Tomer Parash**[1] · **Liel Cohen**[1] · **Hilla Peled**[1] · **Steven Okamoto**[1]

Published online: 10 March 2017
© The Author(s) 2017

Abstract Distributed Constraint Optimization Problems (DCOPs) are NP-hard and therefore the number of studies that consider incomplete algorithms for solving them is growing. Specifically, the Max-sum algorithm has drawn attention in recent years and has been applied to a number of realistic applications. Unfortunately, in many cases Max-sum does not produce high-quality solutions. More specifically, Max-sum does not converge and explores solutions of low quality when run on problems whose constraint graph representation contains multiple cycles of different sizes. In this paper we advance the state-of-the-art in incomplete algorithms for DCOPs by: (1) proposing a version of the Max-sum algorithm that operates on an alternating directed acyclic graph (Max-sum_AD), which guarantees convergence in linear time; (2) solving a major weakness of Max-sum and Max-sum_AD that causes inconsistent costs/utilities to be propagated and affect the assignment selection, by introducing value propagation to Max-sum_AD (Max-sum_ADVP); and (3) proposing exploration heuristic methods that evidently improve the algorithms performance further. We prove that Max-

This paper is an extension of our AAMAS paper [43]. Besides an extended description and examples, it includes a proof of the monotonic improvement of Max-sum_ADVP and its cross-phase convergence, proposes two new classes of exploration heuristics, one inspired by simulated annealing and the other interleaving converging and non-converging versions of the algorithm. Furthermore, we present an extended empirical study that reveals the advantages in using the proposed exploration heuristics.

✉ Roie Zivan
zivanr@bgu.ac.il

Tomer Parash
parasht@bgu.ac.il

Liel Cohen
lielc@bgu.ac.il

Hilla Peled
hillapel@bgu.ac.il

Steven Okamoto
okamotos@bgu.ac.il

[1] Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer-Sheva, Israel

sum_ADVP converges to monotonically improving states after each change of direction, and that it is guaranteed to converge in pseudo-polynomial time to a stable solution that does not change with further changes of direction. Our empirical study reveals a large improvement in the quality of the solutions produced by Max-sum_ADVP on various benchmarks, compared to the solutions produced by the standard Max-sum algorithm, Bounded Max-sum and Max-sum_AD with no value propagation. It is found to be the best guaranteed convergence inference algorithm for DCOPs. The exploration methods we propose for Max-sum_ADVP improve its performance further. However, anytime results demonstrate that their exploration level is not as efficient as a version of Max-sum, which uses Damping.

Keywords DCOP · Incomplete inference algorithms

1 Introduction

The Distributed Constraint Optimization Problem (DCOP) is a general model for distributed problem solving that has a wide range of applications in multi-agent systems. Some examples are: Meeting scheduling [8,21], Sensor nets [3,44] and Disaster response [28].

Many algorithms for solving DCOPs have been proposed. Complete algorithms [9,22, 26] are guaranteed to find the optimal solution, but because DCOPs are NP-hard, to solve optimally, these algorithms require exponential time in the worst case. Thus, there is growing interest in incomplete algorithms, which may find suboptimal solutions but run quickly enough to be applied to large problems or real-time applications [14,20,35,41,42]. For the Meeting scheduling application, a suboptimal solution may include less meetings scheduled or meetings scheduled in less convenient time slots. However, a high quality solution will schedule the most important meetings and make compromises on the least important ones. A similar analogy is relevant in Disaster response scenarios where high quality solutions are the ones in which lives are saved but expandable equipment is lost (and not the other way around).

Whether complete or incomplete, DCOP algorithms generally follow one of two broad approaches: distributed search [9,20,22,41] or inference [7,26,27,34]. In search algorithms, agents directly traverse the solution space by choosing value assignments and communicating these assignments to each other. Among the complete algorithms that implement this approach are ADOPT [22] and BnB-ADOPT [40], which use a pseudo-tree structure in order to increase concurrent actions, each traversing it according to a different scheme, AFB [9] in which a sequential assignment process is enhanced by asynchronous forward bounding, and ConcFB [23] in which concurrency is achieved by splitting the search space among sequential search processes. Incomplete search algorithms are commonly local search algorithms in which the agents hold a complete assignment and iteratively aim to improve it by reassigning their own (local) variables myopically [41,42].

By contrast, agents in inference algorithms traverse the solution space indirectly; each agent maintains *beliefs* about the best costs (or utilities) that can be achieved for each value assignment to its own variables, and selects value assignments that are optimal according to its beliefs. Agents calculate and communicate costs/utilities for each possible value assignment to neighboring agents' variables, and update their beliefs based on messages received from their neighbors. These update methods are specific realizations of the Generalized Distributive Law (GDL) algorithm [1], and hence inference-based algorithms are often referred to as GDL-based algorithms. Complete DCOP inference algorithms include DPOP [26], which is the

distributed implementation of the Bucket elimination algorithm [6] and Action GDL [37], which is a distributed implementation of the GDL algorithm, using a distributed Junction-tree structure.

The Max-sum algorithm [7] is an incomplete, inference, GDL-based algorithm that has drawn considerable attention in recent years, including being proposed for multi-agent applications such as sensor systems [34,35] and task allocation for rescue teams in disaster areas [28]. Max-sum is actually a version of the well known Belief propagation algorithm [39], adjusted to solve DCOPs. Agents in Max-sum propagate cost/utility information to all neighbors. This contrasts with other inference algorithms such as ADPOP [27], which only propagate costs up a pseudo-tree structure overlaid on the agents. As is typical of inference algorithms, Max-sum is purely exploitive both in the computation of its beliefs and in its selection of values based on those beliefs.

Although Max-sum is known to converge to the optimal solution for problems whose constraint graph is acyclic, there is no such guarantee for problems with cycles [7]. Furthermore, when the agents' beliefs fail to converge, the resulting assignments that are considered optimal under those beliefs may be of low quality. This occurs because cyclic information propagation leads to inaccurate and inconsistent information being computed by the agents. Previous empirical study has found this pathology to occur when the constraint graph of the problem contains cycles of various sizes [7]. Unfortunately, many DCOPs that were investigated in previous studies are dense and indeed include such cycles (e.g., [9,22]). Our own experimental study revealed that on various standard benchmark problem classes—random problems of different density parameters and problem sizes, scale-free networks, and problems with structured constraints (graph coloring and meeting scheduling)—Max-sum does not converge and explores low-quality solutions.

Thus, following the common assumption in belief propagation literature, that belief propagation algorithms perform best when they converge [7,39], researchers in the DCOP community joined the attempt to design versions of Max-sum that guarantee convergence. Bounded Max-sum [30] is an algorithm that was designed as part of this attempt. It guarantees convergence by eliminating some of the problem's constraints in order to reduce the DCOP to a tree-structured problem that can be solved in polynomial time by Max-sum. By quantifying the possible effects on solution quality of the removed constraints, it is even possible to provide a theoretical worst-case bound on the resulting solution. However, because most of the constraints may need to be removed, Bounded Max-sum may still find very low-quality solutions for the original problem.

Damping is a method that can be used in order to increase the probability that Max-sum will converge, though it does not offer guarantees for convergence [19]. In Max-sum with damping, agents use a damping factor $\lambda \in (0, 1]$. After performing the calculation of the costs that are intended to be sent to their neighbors, they multiply them by $1 - \lambda$ and add *lambda* times the costs in the messages sent in the previous iteration. For some reason this method was not mentioned in previous papers that presented Max-sum for DCOPs. Our results indicate that the main effect of using damping in Max-sum was not in increasing the probability of convergence, but rather in triggering efficient exploration of solutions of higher quality than standard Max-sum.

In this paper we make three major contributions to the development of incomplete inference algorithms for solving DCOPs.

1. We propose Max-sum_AD, a new version of the Max-sum algorithm that avoids cyclic information propagation while still considering all constraints. Agents in Max-sum_AD behave like those in Max-sum, except that they limit their communication to an alternating

directed acyclic graph (DAG) that forms a spanning subgraph of the communication in conventional Max-sum. Max-sum_AD uses a predefined ordering over the agents to enforce the desired DAG communication pattern, with agents only sending messages in a single direction according to the order. The order divides the set of neighbors of each agent into two disjoint subsets: neighbors who come before it in the order and from whom it receives messages; and neighbors who come after it in the order and to whom it sends messages. Notice that this is an order on the direction of messages and not on the agents' actions. Agents earlier in the order do not receive information from those later in the order, and hence do not consider all constraints in the DCOP. To remedy this, Max-sum_AD reverses the order after the agents' beliefs have converged, thereby reversing the communication flow on the DAG. We term the sequence of iterations in a specific order without change of direction a *phase*. We prove that the maximum number of iterations in a single phase required for the algorithm to converge is equal to the length l, the diameter of the DAG (the longest shortest path between two nodes in the DAG), which is linear in the worst case. Thus, by performing l iterations in each direction (i.e., run Max-sum_AD for two phases), agents converge to a solution that reflects all the constraints in the DCOP.

2. We solve a major weaknesses of Max-sum and Max-sum_AD: the possibility that the best costs/utilities computed for the same variable are based on different value assignments and thus, are not valid. We demonstrate that the propagation of such inconsistent information may result in a distorted selection of assignments by the agents. We solve this weakness with our second proposed algorithm, Max-sum_ADVP, which uses value propagation (VP). After two phases of the algorithm, converging once in each direction and hence considering all constraints in the problem, we require that agents modify their messages to include the value assignment they have selected and consider only constraint costs that are consistent with these selections. Thus, we prevent the possibility that different agents consider costs/utilities that are based on conflicting value assignments. We note that VP has been used in previous studies for complete GDL algorithms [26,37]) for breaking ties. However, to best of our knowledge, our use of VP in [43] was the first time it was applied to GDL algorithms operating on cyclic constraint graphs (e.g., Max-sum).
   We prove that after the second VP phase, Max-sum_ADVP converges to (weak) monotonically improving solutions and that this implies that it converges to a constant solution (i.e. a solution that is not changed after direction changes) in pseudo-polynomial time.
3. We propose two classes of exploration methods that further improve the performance of Max-sum_ADVP. In one class, inspired by simulated annealing [29], agents select their value assignments in a stochastic manner. The second class of exploration methods we propose interleaves the monotonic (exploitive) execution of Max-sum_ADVP with non-monotonic (explorative) versions of the algorithm, e.g., standard Max-sum and Max-sum_AD (without VP). Such balanced combinations allow the algorithm to converge to locally optimal solutions, escape them and converge again.

Our empirical study demonstrates the success of Max-sum_ADVP in comparison with the standard Max-sum algorithm and with Bounded Max-sum when solving random DCOPs, graph coloring problems, meeting scheduling problems and scale-free networks. It also outperformed linear programming relaxation and ADPOP on most benchmarks, thus, Max-sum_ADVP was found to be the best among the incomplete inference algorithms that guarantee convergence.

Some of the exploration methods we proposed were found to improve the results of the Max-sum_ADVP algorithm further, but only specific members of the second class of

exploration methods produced results with a statistically significant advantage over Max-sum_ADVP. On the other hand, Max-sum with damping, although in most scenarios failed to converge, has explored solutions with much higher quality than standard Max-sum, and in some cases (surprisingly on dense random problems) it explored solutions with higher quality than the ones Max-sum_ADVP converged to. When combined with an anytime mechanism [42] on random uniform problems, it produced results that outperformed significantly the exploration methods we proposed. On realistic structure problems this advantage was less conclusive.

The rest of this paper is organized as follows: We present related work in Sect. 2. The DCOP formalism is presented in Sect. 3. Section 4 presents the standard Max-sum algorithm. The Max-sum_AD algorithm is presented in Sect. 5. Section 6 identifies the need for value propagation (VP) and describes how VP is combined with Max-sum_AD. It further includes the presentation of the proposed classes of exploration methods for Max-sum_ADVP. Section 8 includes an evaluation of the proposed algorithm in comparison with Max-sum, Max-sum with damping and Bounded Max-sum. It also presents experiments that compare the performance of Max-sum_ADVP with the exploration methods proposed. Our conclusions are presented in Sect. 9.

2 Related work

Many algorithms for solving DCOPs were proposed in the last decade. While some complete algorithms such as ADOPT [22], BnB-ADOPT [40], and AFB [9] perform distributed search, GDL-based complete algorithms implement a dynamic programming approach [1,26,37]. The first to apply this approach to DCOP were Petcu and Faltings by proposing the DPOP algorithm [26]. DPOP performs dynamic programming on a pseudo-tree structure, with agents starting with the leaves calculating tables of costs/utilities and propagating them up the pseudo-tree. The table computed by an agent stores the best sum of costs/utilities of all constraints involving agents in its subtree, with an entry for every joint value assignment for agents not in its subtree, which are involved in these constraints. DPOP requires a linear number of message-passing cycles, but the largest table size is exponential in the induced width and hence, so is the running time to compute the tables. Since pseudo-trees may have limited branching in dense problems, recent studies investigated alternative structures (e.g., junction trees) in order to increase the parallelism in GDL-based algorithms [4,37].

Incomplete DCOP search algorithms typically implement a synchronous local search. In each step of the algorithm an agent sends its value assignment to all its neighbors in the constraint graph and receives the assignments of all its neighbors. Local search algorithms differ in the method agents use to decide whether to replace their current value assignments to their variables. For example, the agent that can most improve in its neighborhood replaces its assignment in the maximum gain messages algorithm (MGM) [20], while every agent individually makes a stochastic decision to replace its assignment in the distributed stochastic algorithm (DSA) [41].

Maheswaran et al. [20] and Pearce and Tambe [25] used completely exploitive algorithms to converge to locally optimal solutions whose quality is guaranteed to be within a predefined distance from the quality of the global optimal solution. The approximation level is dependent on a parameter k, which defines the size of coalitions that agents can form. These k size coalitions transfer the problem data to a single agent, which performs a complete search procedure in order to find the best assignment for all agents within the k size coalition. As

a result, the algorithm converges to a state that is *k optimal* (k-opt) [25], i.e., no better state exists if k agents or fewer change their assignments.

The production of k-opt solutions may require solving an exponential number of problems of size k. To overcome this shortcoming, recent studies have proposed alternatives for the selection of small local environments that would be solved optimally in order to produce quality guarantees on the overall solution. One alternative, t-distance, generated environments dependent on the distance of nodes in the constraint graph [16]. While this alternative reduced the number of problems that need to be solved, it did not bound the size of the problems that are solved. The most recent approach included the generation of environments that were bounded both by distance and size [38]. Thus, the number of problems to solve is bounded by the number of agents and the magnitude of the problems by the predefined size.

Aggregation of agents' constraints was also used in an attempt to cope with Max-sum's failure to converge when solving cyclic problems [7]. It included the union of groups of agents to clusters of adjacent agents represented by a single agent in the cluster. The constraints between the agents in the cluster were aggregated and held by the agent representing the cluster. Thus, it required that some constraints would be revealed in a preprocessing phase to agents that are not included in the constraints. Furthermore, the amount of aggregated information was not limited and in dense problems could result in a single agent holding a large part of the problem's constraints (partial centralization). Another approach is to aggregate constraints and unite nodes in the constraint graph so that the resulting graph would be a tree [37]. However, the result of this rearrangement of the constraint graph is the need to perform exponential computation and transfer exponential communication that will result in a complete solution. In this paper we focus on incomplete GDL algorithms that avoid partial centralization and clustering of agents, and attempt to solve the original DCOPs as do standard complete algorithms (e.g., ADOPT and DPOP), *one-opt* distributed local search algorithms (e.g., DSA and MGM) and as the standard Max-sum algorithm does [30].

Bounded Max-sum [30] is a version of Max-sum that both guarantees convergence and provides a theoretical bound on the quality of the solution found compared to the optimal. Bounded Max-sum eliminates some of the problem's constraints in order to reduce the DCOP to a tree-structured problem that can be solved in polynomial time. Then, the sum of the worst costs for all eliminated constraints serves as the bound on the approximation of the optimal solution. A later study proposed a version that produces an improved bound [31].

A different approach to cope with the non-convergence of Max-sum (or its equivalent belief propagation version Max-product) proposes algorithms that efficiently solve a linear program (LP) relaxation of the combinatorial problem, then select value assignments based on the solution to the LP. This approach has been intensively studied by the graphical models community in recent years and multiple methods have been developed, including Max-Product Linear Programming (MPLP) [10], Tree Reweighted Belief Propagation (TRBP) [39] and Norm-Product (NP) [12]. These all try to converge to the LP solution, but differ in their convergence rates and their guarantees on the ability to converge with respect to the size of the problem and the problem structure. This line of research draws much attention in the graphical model community because many of these methods, while incomplete, produce the optimal solution for many relevant applications. Optimal solutions are found more often when applying *tightening*, i.e., clustering of factors/constraints to higher-order (i.e., higher-arity) constraints [33]. LP relaxations were also applied to DCOPs and were found to improve on standard incomplete DCOP algorithms in some settings [11,36]. Our empirical study includes comparisons of the algorithms we propose with an algorithm solving the LP relaxations of the problems [39].

Approximate DPOP (ADPOP) [27] is an incomplete version of DPOP parameterized by an upper limit *maxDims* on the number of dimensions of cost/utility tables communicated by agents. When an agent would compute a table with more than *maxDims* dimensions under DPOP, it instead computes upper and lower bound projections to tables with *maxDims* dimensions. This can be done by selecting the set of dimensions to be included in the lower-dimensional table, then independently projecting tables received from children and constraints involving this constraint (but not its children) to tables that only involve the retained constraints. These are then combined as in DPOP to compute the bound projections to be sent to the agent's parent. In the worst case, computing the projection for each table received from a child requires finding the maximum and minimum over joint assignments to $maxDims - 1$ dimensions. Since this must be done for each of the joint assignments in the *maxDims* dimensions of the generated table and each child table, computing a table in ADPOP has complexity $\Theta\left(N \cdot D^{2 \cdot maxDims}\right)$ where N is the number of neighbors and D is the size of the domains.

Okimoto et al. [24] proposed an incomplete algorithm similar to both Bounded Max-sum and ADPOP. Like ADPOP, it uses a pseudo-tree, although one based on a variable ordering rather than depth-first search. It adds constraints to make the dependencies in the pseudo-tree explicit (i.e., the variable dimensions that would have to be included in tables sent in ADPOP). Similar to Bounded Max-sum, constraints are then eliminated to achieve the desired induced width while providing bounds on solution quality based on the eliminated constraints. The reduced problem is then solved using a complete algorithm.

The alternating direction approach we implement in this paper is inspired by algorithms for solving asymmetric distributed constraints problems [5]. However, unlike in the case of asymmetric problems where the motivation for this approach was preserving privacy, in this paper the motivation is strictly algorithmic. The alternating order approach was also used for directional arc consistency in complete search algorithms for solving weighted CSPs in [13]. The use of an alternating order was found to increase the effect of directional arc consistency and produce better results on some benchmarks. The usefulness of this approach for different types of algorithms and problems encourages future work investigating its usefulness in other scenarios.

3 Distributed constraint optimization

To avoid confusion, and without loss of generality, in the rest of this paper we will assume all problems are minimization problems as presented in the early DCOP papers (e.g., [22]). Thus, we assume that all constraints define costs and not utilities. The GDL algorithm for minimization problems is actually a *Min*-sum GDL algorithm. However, we will continue to refer to it as Max-sum since this name is widely accepted. Our description of a DCOP is also consistent with the definitions in many DCOP studies, e.g., [9,22,26].

A $DCOP$ is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$. $\mathcal{A}$ is a finite set of agents $A_1, A_2, \ldots, A_n$. $\mathcal{X}$ is a finite set of variables $X_1, X_2, \ldots, X_m$. Each variable is held by a single agent (an agent may hold more than one variable). $\mathcal{D}$ is a set of domains $D_1, D_2, \ldots, D_m$. Each domain D_i contains the finite set of values that can be assigned to variable X_i. We denote an assignment of value $d \in D_i$ to X_i by an ordered pair $\langle X_i, d \rangle$. $\mathcal{R}$ is a set of relations (constraints). Each constraint $C \in \mathcal{R}$ defines a non-negative *cost* for every possible value combination of a set of variables, and is of the form $C : D_{i_1} \times D_{i_2} \times \ldots \times D_{i_k} \rightarrow \mathbb{R}^+ \cup \{0\}$. A *binary constraint* refers to

exactly two variables and is of the form $C_{ij} : D_i \times D_j \rightarrow \mathbb{R}^+ \cup \{0\}$. A *binary DCOP* is a DCOP in which all constraints are binary.

A *partial assignment* (PA) is a set of value assignments to variables, in which each variable appears at most once. The set of all variables that appear in PA is $vars(PA) = \{X_i : \exists d \in D_i \wedge \langle X_i, d\rangle \in PA\}$. A constraint $C \in \mathcal{R}$ of the form $C : D_{i_1} \times D_{i_2} \times \ldots \times D_{i_k} \rightarrow \mathbb{R}^+ \cup \{0\}$ is *applicable* to PA if $X_{i_1}, X_{i_2}, \ldots, X_{i_k} \in vars(PA)$. If C is applicable to PA with $\{\langle X_{i_1}, d_{i_1}\rangle, \langle X_{i_2}, d_{i_2}\rangle, \ldots, \langle X_{i_k}, d_{i_k}\rangle\} \subseteq PA$, we may also write $C(PA)$ in place of $C(d_{i_1}, d_{i_2}, \ldots, d_{i_k})$. We denote the partial assignment PA with the value assignment for a specific variable X omitted as $PA^{-X} = \{\langle X', d_{X'}\rangle \in PA : X' \neq X\}$.

The *cost of a partial assignment* PA is the sum of all applicable constraints to PA over the assignments in PA. A *complete assignment* or a *solution* is a partial assignment that includes all the variables ($vars(PA) = \mathcal{X}$). An *optimal solution* is a complete assignment/solution with minimum cost.

For simplicity we make the standard assumptions made in DCOP studies: All DCOPs considered in this paper are binary DCOPs, each agent holds exactly one variable and all the constraints this variable is involved in. Thus, the number of agents is equal to the number of variables and we use n for both.

4 Standard Max-sum

The Max-Sum algorithm [7] is a GDL algorithm [1] that operates on a *factor graph* [17] that is a bipartite graph whose nodes represent both variables and constraints.[1] We overload the notation and use X to denote the variable-node representing variable X. The set of function-nodes is denoted by $\mathcal{F}$, and for a function-node $F \in \mathcal{F}$, we denote the constraint it represents as C_F. Each variable-node X is connected to all function-nodes F where X is involved in C_F. Similarly, F is connected to all X that represent variables in the original DCOP that are involved in C_F. The set of neighbors in the factor graph of a variable-node X or a function-node F is denoted by N_X and N_F, respectively; note that N_X contains only function-nodes and N_F contains only variable-nodes. Variable-nodes and function-nodes are considered "agents" in Max-sum, i.e., they can perform computation and send and receive messages. The DCOP agents perform the roles of different nodes in the factor graph. We assume that the role of each variable-node is performed by the DCOP agent that holds the variable, and the role for each function-node is performed by one of the agents whose variable is involved in the constraint it represents.

Figure 1 demonstrates the transformation of a DCOP to a factor graph. On the left we have a DCOP with three agents, each holding a single variable. All variables are connected by binary constraints. On the right we have a factor graph. Each agent takes the role of the node representing its own variable and the role of one of the function-nodes representing a constraint it is involved in, e.g., in this factor graph agent A_1 takes the role of function-node F_1, which represents the constraint between its own variable X_1 and variable X_3 held by agent A_3.

Figure 2 presents a sketch of the Max-sum algorithm.[2] The pseudo-code for variable-nodes and function-nodes is similar apart from the computation of the content of messages

[1] Following [7] we use the terms "variable-node" and "function-node" to refer to nodes in the factor graph corresponding to variables and constraints, respectively.

[2] In contrast to previous papers on Max-sum, we present it using pseudo-code. This is following standard DCOP literature, e.g., [22,26,41]. Nevertheless, only the presentation is different; the algorithm itself is identical to the algorithm presented in [7,30].

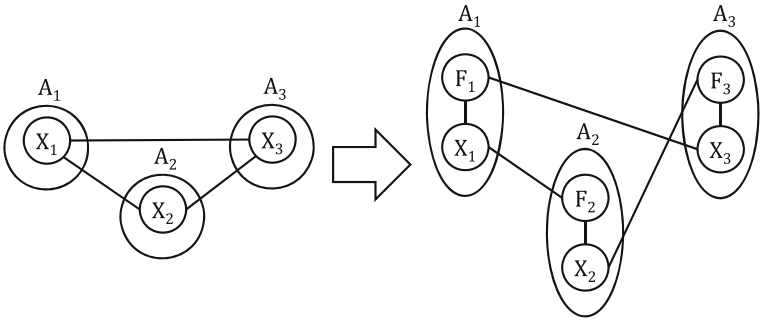

Fig. 1 Transformation of a DCOP to a factor graph

Max-sum (node v)
1. initialize all received messages to 0
2. $i \leftarrow 0$
3. **while** (no termination condition is met)
4. $\quad i \leftarrow i + 1$
5. $\quad$ **if** (not first iteration) collect messages from N_v
6. $\quad$ **for each** $v' \in N_v$
7. $\quad\quad$ **if** (v is a variable-node)
8. $\quad\quad\quad$ produce message $Q^i_{v \to v'}$ using messages $\{R^{i-1}_{F \to v} : F \in N_v \setminus \{v'\}\}$ by Eq. (1)
9. $\quad\quad\quad$ send message $Q^i_{v \to v'}$ to v'
10. $\quad\quad$ **if** (v is a function-node)
11. $\quad\quad\quad$ produce message $R^i_{v \to v'}$ using constraint and messages $\{Q^{i-1}_{X \to v} : X \in N_v \setminus \{v'\}\}$ by Eq. (2)
12. $\quad\quad\quad$ send message $R^i_{v \to v'}$ to v'

Fig. 2 Standard Max-sum

to be sent. Messages are only sent from variable-nodes to neighboring function-nodes and from function-nodes to neighboring variable-nodes. The message sent by each variable-node X to a neighboring function-nodes F at iteration i is denoted by $Q^i_{X \to F}$ and is based solely on data received from neighboring function nodes. The message sent by each function-node F to a neighboring variable-node X at iteration i is denoted by $R^i_{F \to X}$ and is based on data received from neighbors as well as the original constraint represented by the function-node.

The message sent from a variable-node X to a function-node F at iteration i contains, for each of the values $d \in D_X$, the sum of costs for d that was received from all function neighbors apart from F in iteration $i-1$. Formally, it is the function $Q^i_{X \to F} : D_X \to \mathbb{R}$ with

$$Q^i_{X \to F}(d) = \begin{cases} \sum_{F' \in N_X \setminus \{F\}} R^{i-1}_{F' \to X}(d) - \alpha^i_{XF} & \text{if } i > 1 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

for all $d \in D_X$. The α^i_{XF} term is a constant that is deducted in order to prevent the magnitudes of transmitted messages from growing arbitrarily. Selecting

$$\alpha^i_{XF} = \frac{1}{|D_X|} \sum_{d \in D_X} \sum_{F' \in N_X \setminus \{F\}} R^i_{F' \to X}(d)$$

so that $\sum_{d \in D_X} Q^i_{X \to F}(d) = 0$ is a reasonable choice for this purpose [7,30]. Note that as long as a constant amount is subtracted for all $d \in D_X$, the algorithm is not affected because only the differences between the costs for different values matter.

A message sent from a function-node F to a variable-node X in iteration i includes for each possible value $d \in D_X$ the minimal cost of any combination of assignments to the variables involved in F apart from X and the assignment of value d to variable X. Formally, the message from F (representing constraint C_F) to X is the function $R^i_{F\to X} : D_X \to \mathbb{R}$ with

$$R^i_{F\to X}(d) = \begin{cases} \min\limits_{\mathbf{d}\in \mathrm{Dom}(C_F):d_X=d} C_F(\mathbf{d}) + \sum\limits_{Y\in N_F\setminus\{X\}} Q^{i-1}_{Y\to F}(v_Y) & \text{if } i > 1 \\ \min\limits_{\mathbf{d}\in \mathrm{Dom}(C_F):d_X=d} C_F(\mathbf{d}) & \text{otherwise} \end{cases} \tag{2}$$

for all $d \in D_X$, where $\mathrm{Dom}(C_F)$ denotes the domain of the constraint function C_F.

Agents compute their beliefs over their variables according to the messages received by the corresponding variable-nodes. Formally, the beliefs for a variable-node X at iteration i is a function $b^i : D_X \to \mathbb{R}$ with

$$b^i_X(d) = \sum_{F\in N_X} R^{i-1}_{F\to X}(d) \qquad \forall d \in D_X. \tag{3}$$

The agent performing the role of X chooses the optimal value assignment $\widehat{d^i_X} \in D_X$ based on its beliefs, selecting

$$\widehat{d^i_X} = \arg\min_{d\in D_X} b^i_X(d). \tag{4}$$

Note that updating Q, R, and b for iteration i in Eqs. (1)–(3) requires only the messages generated by neighbors in the previous iteration. This facilitates a dynamic programming approach that is typical of GDL-family algorithms: in Max-sum the received messages are all initialized as 0, and in subsequent iterations only the most recently received messages are retained, overwriting those received in previous iterations. Thus, most descriptions of Max-sum do not index the messages or belief by iteration; we included them only to emphasize that new messages are synchronously generated based on messages generated by neighbors in the previous iteration. In following sections we will drop the iteration index from our notation for clarity because in Max-sum_AD and Max-sum_ADVP the most recently received message from a neighbor may not have been generated in the previous iteration.

5 Max-sum on an alternating DAG (Max-sum_AD)

Standard max-sum is known to perform poorly when there is cyclic propagation of information on multiple cycles in the factor graph [7]. To overcome this shortcoming, we strictly control the nature of cyclic information propagation. In particular, we direct each edge so that the factor graph at each iteration is a directed acyclic graph (DAG), and only permit messages in the direction of each edge. The directions of all edges (and hence the flows of messages) are periodically and synchronously reversed. We term the sequence of iterations between changes in direction a *phase*. Within each phase information propagation is purely acyclic; cyclic information propagation only occurs across phases. The resulting algorithm is Max-sum on an Alternating DAG (Max-sum_AD).

To define the DAG topology, we first select an order $\prec_{\mathcal{X}}$ on all variable-nodes in the factor graph. This order on variable-nodes is then extended to a partial order $\prec$ on all nodes in the factor graph by ordering each function-node between the two variable-nodes it connects:

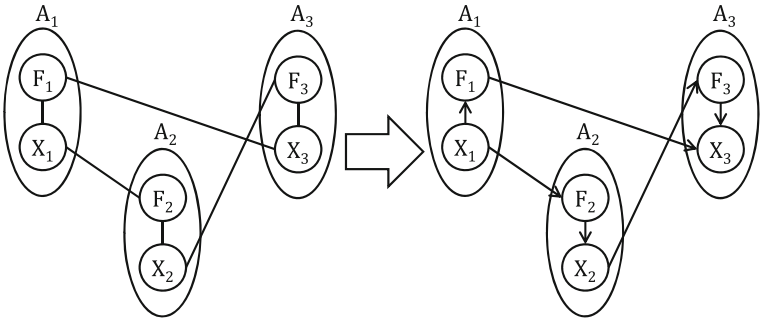

Fig. 3 Transformation of a factor graph to a directed acyclic graph

$$v_1 \prec v_2 \iff \begin{cases} v_1 \prec_{\mathcal{X}} v_2 & v_1, v_2 \in \mathcal{X} \\ v_1 \prec_{\mathcal{X}} X & v_1 \in \mathcal{X}, v_2 \in \mathcal{F}, N_{v_2} = \{v_1, X\} \\ X \prec_{\mathcal{X}} v_2 & v_1 \in \mathcal{F}, v_2 \in \mathcal{X}, N_{v_1} = \{X, v_2\} \end{cases} \tag{5}$$

This defines a directed acyclic graph (DAG) on the nodes, with directed edges from a node only to the neighbors after it in $\prec$. The construction method ensures that the starting and end points of the DAG must be variable-nodes. Reversing the order flips the direction of all edges so that nodes have edges only to neighbors before it according to $\prec$.

We note that, like any Max-sum algorithm, Max-sum_AD requires a generation of a function node, and like Bounded Max-sum [30], Max-sum_AD requires that agents generate a special structure on the factor graph, and replace directions during execution. Such structures are common in DCOP literature, e.g., algorithms that use pseudo-trees [22,26,40], or BFS trees [42]. On the other hand, local search algorithms such as DSA [41] do not require any pre-processing. This is definitely an issue that one needs to consider when deciding which DCOP algorithm to implement.

An example of the orientation of a factor graph to yield a DAG is depicted in Fig. 3. This example uses $\prec_{\mathcal{X}}$ that orders the variable-nodes by the indices of the agents performing their roles, with a variable-node performed by A_i ordered before a variable-node performed by A_j if $i < j$. The initial order on variable-nodes is $X_1 \prec_{\mathcal{X}} X_2 \prec_{\mathcal{X}} X_3$. This is extended to all nodes in the factor graph by setting $X_1 \prec X_2 \prec X_3$ along with $X_1 \prec F_1 \prec X_3$ and $X_1 \prec F_2 \prec X_2$ and $X_2 \prec F_3 \prec X_3$.

The pseudo-code for Max-sum_AD given an order $\prec$ and phase length l is presented in Fig. 4. Each node partitions its neighbors into two sets: those that come before it according to $\prec$, and those that come after it. Initially, the set *recipients* of nodes that will receive its messages are the neighbors that come after it according to $\prec$ (line 2). After a phase of length l, the *recipients* changes to the set of neighbors that come before the node according to $\prec$, and the direction of message flow will continue to alternate every l iterations until termination (lines 15–17).

The content of messages in Max-sum_AD is computed according to Eqs. (1) and (2) (lines 9 and 12), just as in standard Max-sum. It is important to note that these computations are based on the messages from *all* neighbors, not only those in $N_v \setminus$ *recipients*. This means that the messages from neighbors in *recipients* that are used as inputs to Eqs. (1) and (2) are those received in the previous phase, or the initialized values of 0 if it is the first phase. This is what facilitates cyclic information propagation across phases and allows agents to eventually accumulate information based on all constraints in the DCOP.

Max-sum_AD (node v, order $\prec$, phase_length l)
1. *recipients* $\leftarrow \{v' \in N_v : v \prec v'\}$
2. initialize all received messages to 0
3. *counter* $\leftarrow 0$
4. **while** (no termination condition is met)
5. *counter* $\leftarrow$ *counter* $+ 1$
6. **if** (not first iteration) collect messages from $N_v \setminus$ *recipients*
7. **for each** $v' \in$ *recipients*
8. **if** (v is a variable-node)
9. produce message $Q_{v \to v'}$ using messages $\{R_{F \to v} : F \in N_v \setminus \{v'\}\}$ by Eq. (1)
10. send message $Q_{v \to v'}$ to v'
11. **if** (v is a function-node)
12. produce message $R_{v \to v'}$ using constraint and messages $\{Q_{X \to v} : X \in N_v \setminus \{v'\}\}$ by Eq. (2)
13. send message $R_{v \to v'}$ to v'
14. **if** (*counter* $= l$)
15. *recipients* $\leftarrow N_v \setminus$ *recipients*
16. *counter* $\leftarrow 0$

Fig. 4 Max-sum_AD

For example, consider again the DAG in Fig. 3. In the first iteration, variable-node X_1 computes and sends messages to F_1 and F_2, function-node F_1 computes and sends a message to X_3, function-node F_2 computes and sends a message to X_2, variable-node X_2 computes and sends a message to F_3, and function-node F_3 computes and sends a message to X_3; because this is the first iteration, these messages are all based on the received messages initialized to 0. Variable-node X_3 has no outgoing edges in the DAG and hence it sends no messages. In the second iteration, all nodes but X_1 receive the messages sent by their in-neighbors, and all nodes but X_3 again compute and send messages to their out-neighbors. For X_1, these are the same messages (because it has received no messages and hence continues to use the initialized values), but for the other nodes their messages are updated to reflect the messages they received from their neighbors. This continues until the end of the first phase.

After l iterations, the first phase ends and the directions of all edges in the DAG in Fig. 3 are reversed. Thus, in iteration $l+1$ all nodes but X_1 receive the messages generated and sent in iteration l, and all nodes but X_1 then compute and send messages to their new recipients: F_1 and F_2 send to X_1, X_2 sends to F_2, F_3 sends to X_2, and X_3 sends to F_1 and F_3. These are the first messages sent by X_3, and they are based on the messages that were generated by F_1 and F_3 in iteration l. In contrast, X_1 sent messages in the first phase but will not send any messages in the second phase.

It is only in the second phase that cyclic information propagation begins, with nodes being able to incorporate cost information accumulated in the first phase. For example, in the first phase X_1 sent messages that were not based on any constraint information since that is generated by the function-nodes and X_1 received no messages. However, in the second phase, X_1 begins to receive this information from F_1 and F_2. Similarly, in the first phase X_2 received cost information from F_2 but not from F_3. In the second phase, it receives cost information from F_3 and its beliefs thereby consider all constraints it is involved in. The messages X_2 receives from F_3 are also based on information from F_1 by way of X_3. In this way X_2 and all other variable-nodes are able to consider all constraints in the problem.

Observe that in the first phase, X_1 never receives any messages and hence its outgoing messages $Q_{X_1 \to F_1}$ and $Q_{X_1 \to F_2}$ never change. Since F_1 and F_2 compute their messages based on the constraints and X_1's messages, neither of which change, it follows that $R_{F_1 \to X_3}$ and $R_{F_2 \to X_2}$ never change after the second iteration when F_1 and F_2 first receive X_1's

unchanging messages. This trend continues in subsequent iterations of the first phase with X_2, F_3, and finally X_3 arriving at stable, unchanging outgoing messages. Likewise, in the second phase, X_3 never receives any messages after iteration $l+1$, and hence its outgoing messages do not change within that phase. The other nodes then arrive at fixed outgoing messages in subsequent iterations in a reverse order to that they did in the first phase. In this way, the messages and hence the beliefs and the value selections of all nodes converge if the phases are sufficiently long. We next formally prove this property of *single-phase convergence* (SPC).

Lemma 1 *Given a DAG of the factor graph, the content of the messages received by any node v does not change after $\ell_v + 1$ iterations, where ℓ_v is the length of the longest path in the DAG that reaches v.*

Proof We prove the lemma by induction on ℓ_v. When $\ell_v = 0$, it follows that v has no incoming edges and hence the claim is vacuously true, establishing the base case. To prove the inductive step we assume that the lemma holds for any node v' with $\ell_{v'} < \ell_v$. Because this is a DAG, the longest path to any of the in-neighbors of v must be strictly less than ℓ_v. Thus, according to the inductive hypothesis the messages received by all of v's in-neighbors do not change after ℓ_v iterations. Since the content of messages produced by a node is dependent only on the most recently-received messages, it follows that the outgoing messages from the in-neighbors of v also do not change after ℓ_v iterations. Since the outgoing message generated by an in-neighbor of v at iteration ℓ_v is received by v on iteration $\ell_v + 1$, the claim also holds for v. □

Using Lemma 1, we can now establish the worst-case bound on the number of iterations required in each phase to guarantee convergence in Max-sum_AD.

Theorem 1 (Single-Phase Convergence) *In Max-sum_AD, if the phase length $l \geq 2n$ then the messages of all nodes and the beliefs of all variable-nodes do not change after iteration $2n$ in each phase.*

Proof By Lemma 1, the messages received by all nodes in a single phase do not change after $\max_v \ell_v + 1$ iterations, where ℓ_v is the length of the longest path that reaches v in the DAG for that phase. Since the variable-nodes' beliefs are based solely on the most recently received messages according to Eq. (3), they also do not change after iteration $\max_v \ell_v + 1$ of each phase.

Although the length of the longest path to each node depends on which of the two alternating DAGs is being considered, $\max_v \ell_v$ does not. This is because a sequence of nodes is a path in a DAG if and only if the reversed sequence of nodes is a path in the DAG in which all edges are reversed. Thus, the lengths of the longest paths are the same in both of the alternating DAGs used by Max-sum_AD, and we can speak of $\max_v \ell_v$ without reference to the specific phase.

Paths in the DAGs must alternate between variable-nodes and function-nodes because the DAGs are bipartite. Moreover, each path must begin and end with a variable-node and contain each variable-node at most once because of the definition of $\prec$ in Eq. (5). Thus, each path can contain at most n variable-nodes and $n-1$ function-nodes, so $\max_v \ell_v + 1 \leq (2n-1)+1 = 2n$. □

The guarantee of SPC is a virtue of Max-sum_AD that is not shared by standard Max-sum. Theorem 1 ensures that SPC can be achieved in a linear number of iterations even in the worst

case, and convergence may be guaranteed in even fewer iterations depending on the diameter of the DAG, as outlined in the proof.

If there is a unique value assignment that optimizes each variable-nodes' beliefs, or if value selection is deterministic in the case of ties, then Theorem 1 also implies that agents converge to a complete assignment in each phase. We refer to the messages, beliefs, and, when applicable, assignments that are converged to within a phase as the SPC messages, beliefs, and assignments, respectively.

Changing the direction of communication is crucial to allowing variable-nodes to consider all constraints in the problem. However, there is no guarantee that the new solution that the agents converge to in the next phase will be an improvement over the previous assignment. Guaranteeing such monotonic improvement across phases, or *cross-phase convergence* (CPC) requires a modification to the algorithm, as described in the next section.

We note that it is possible to design Max-sum_AD such that each node in the factor-graph sends messages only in a single iteration in each phase. Each node waits to receive messages from all neighbors that come before it according to the order of nodes in the phase. Nodes with no such neighbors send messages in the first iteration of the phase. Such a design avoids sending redundant messages and reduces the communication cost. However, in this design agents have more global awareness, e.g., they know the length of the paths they are involved in, in each direction. A similar trade-off was presented in [30]. While the alternative design reduces the number of messages sent, it does not affect the number of iterations until convergence.

6 Max-sum_AD with value propagation (Max-sum_ADVP)

In this section we introduce value propagation into the Max-sum_AD algorithm to yield Max-sum_AD with Value Propagation (Max-sum_ADVP). We start by presenting the motivation for value propagation and then go into the algorithmic details. We then prove that when combined with value propagation, the algorithm visits monotonically improving complete assignments starting in the fourth phase (second phase of value propagation), and that CPC is guaranteed.

6.1 Motivation for value propagation

In order to understand the need for value propagation in Max-sum in general and specifically in Max-sum_AD we identify two phenomena that deteriorate the ability of agents to identify the value assignments that will minimize the cost of the solution.

The first phenomenon was identified in previous studies on complete GDL algorithms [26, 37] and is exemplified by the graph coloring example in Fig. 5. In the first iteration, F_1 uses Eq. (2) to compute its message $R_{F_1 \to X_1}$ by finding, for each possible value assignment of X_1, the minimum constraint cost over all possible value assignments to X_2. Because there is a value assignment for X_2 that results in a cost of 0 for each value assignment of X_1 (namely, assigning X_2 a different value than X_1), $R_{F_1 \to X_1}(d) = 0$ for all $d \in D_1$. By symmetry, all messages from function-nodes to variable-nodes also map every value assignment to 0. Due to initialization, all messages from variable-nodes to function-nodes also map every value assignment to 0, and hence it is easy to verify that using Eqs. (1) and (2) will likewise map every value assignment to 0 for all future messages.

The result is that agents do not propagate any useful information. Because Max-sum and Max-sum_AD both generate messages according to Eqs. (1) and (2), they are both subject to

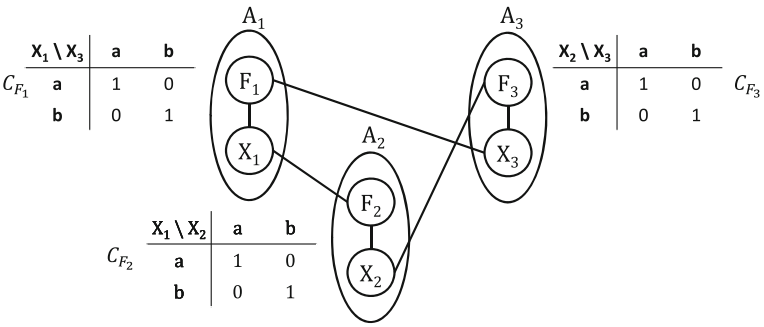

Fig. 5 Example for the need for value propagation to resolve indeterminacy in value selection

this phenomenon. Upon termination, the variable-nodes are indifferent between their possible value assignments: all values in their domains satisfy Eq. (4). A simple, deterministic value selection implementation results in all agents selecting the same value, resulting in a global cost of 3, the worst possible. A uniformly random value selection implementation results in lower expected global cost, although it is still possible that the worst case solution is found. Note, however, that this phenomenon of uninformative messages being passed is not dependent on the graph coloring "not-equals" constraints; it also arises with coordination "equals" constraints in which neighboring agents that select the same value incur no cost, while neighboring agents that select different values incur a cost of 1. In such problems, deterministic value selection would be optimal, while stochastic value selection would be suboptimal. More generally, the constraints may impose costs on permutations of assignment pairs (as in unique label cover problems [15]), for which both deterministic and stochastic value selection would be suboptimal. It is unreasonable to defer addressing this shortcoming to the value selection implementation, as the lack of information from Max-sum means it is essentially equivalent to solving the original DCOP.

Value propagation resolves the indifference by having agents communicate their selected values. An agent receiving such a message can then compute the costs that would be incurred given the received value selections. For example, in DPOP, agents condition their beliefs on the value selections of their direct ancestors in a pseudo-tree. After these beliefs are computed, the root of the pseudo-tree initiates the value selection stage of the algorithm by choosing its value and propagating this to its children. Subsequent agents in turn select their own values and propagate the necessary value selections further down the tree. This guarantees the optimality of DPOP even in the presence of ties in agents' beliefs.

A different method to break ties was suggested by Farinelli et al. [7]. Each agent randomly selects preferences over its variables' values. These preferences are costs that are orders of magnitude smaller than the costs in the actual constraints. The use of such preferences reduces the probability of ties to be insignificant. However, if we use this method in graph coloring problems similar to the problem above, it is easy to see that only personal preferences will be propagated by the algorithm. Thus, the information propagated by the agents will be arbitrary and the selection of values will be as well. We demonstrate in our experiments that the use of this preference-based method is not as beneficial as value propagation when solving problems with multiple cycles.

Although breaking ties is the only motivation for value propagation in complete algorithms, incomplete GDL algorithms such as Max-sum_AD have a second motivation: inconsistent cost calculations. We will illustrate this phenomenon using the factor graph depicted

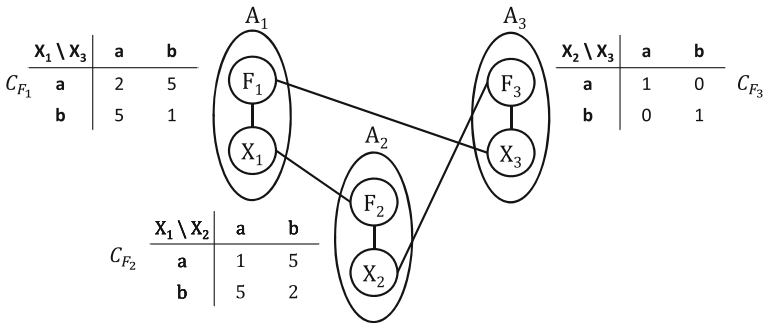

Fig. 6 Example for the need for value propagation to avoid inconsistent cost calculation

in Fig. 6. There are two optimal assignments with total cost 4: $\{\langle X_1, a\rangle, \langle X_2, a\rangle, \langle X_3, a\rangle\}$ and $\{\langle X_1, b\rangle, \langle X_2, b\rangle, \langle X_3, b\rangle\}$. Consider now Max-sum_AD with $\prec_{\mathcal{X}}$ according to the agents' indices.[3] By Theorem 1, the messages and beliefs converge by the sixth iteration of each phase. Adopting the vector notations $b_i = (b_i(a), b_i(b))$ for beliefs (and similarly for messages $Q_{X\to F}$ and $R_{F\to X}$), the SPC messages and beliefs in the first two phases are as follows:

Phase 1 (Max-sum_AD):

$$b_1 = (0, 0) \qquad Q_{X_1\to F_1} = (0, 0) \qquad Q_{X_1\to F_2} = (0, 0)$$
$$R_{F_1\to X_3} = (2, 1) \qquad R_{F_2\to X_2} = (1, 2)$$
$$b_2 = (1, 2) \qquad Q_{X_2\to F_3} = (-0.5, 0.5)$$
$$R_{F_3\to X_3} = (0.5, -0.5)$$
$$b_3 = (2.5, 0.5)$$

Phase 2 (Max-sum_AD):

$$b_3 = (2.5, 0.5) \qquad Q_{X_3\to F_1} = (0.5, -0.5) \qquad Q_{X_3\to F_3} = (0.5, -0.5)$$
$$R_{F_1\to X_1} = (2.5, 0.5) \qquad R_{F_3\to X_2} = (-0.5, 0.5)$$
$$b_2 = (0.5, 2.5) \qquad Q_{X_2\to F_2} = (-0.5, 0.5)$$
$$R_{F_2\to X_1} = (0.5, 2.5)$$
$$b_1 = (3, 3)$$

In both the first and second phases, X_2 will choose a and X_3 will choose b, thereby precluding selection of an optimal joint assignment of cost 4. Instead, the agents will incur a total cost of 6: a cost of 1 from F_1 or F_3 and a cost of 5 from F_3 or F_1, depending on whether X_1 chooses a or b, respectively.

This shortcoming arises because in the first phase, F_1 and F_2 each independently consider possible assignments to X_1 and optimistically assume X_1 chooses the value assignment that minimizes costs for each value of the recipient, according to Eq. (2). This optimism results in messages to X_2 and X_3 that assume X_1 will choose the same value assignment as the recipients, while the independence leads to inconsistent assumptions by X_2 and X_3 about the assignment X_1 will actually take. In this case, X_2's choice of a is based on the assumption that X_1 will choose a, while X_3's choice of b is based on an assumption that X_1 will choose b.

[3] We demonstrate the phenomenon for Max-sum_AD since it is easier to follow. In standard Max-sum, such inconsistent information concerning the conflicting assignment of some node is propagated in all directions and fed back to the node itself through cycles.

Even when the cost information is propagated back through the factor graph in the second phase, the messages are based on the inconsistent assumptions from the first phase, resulting in suboptimal assignments. Continuing the algorithm for additional phases does not remedy this problem:

Phase 3 (Max-sum_AD):

$b_1 = (3, 3)$ $Q_{X_1 \to F_1} = (-1, 1)$ $Q_{X_1 \to F_2} = (1, -1)$
$R_{F_1 \to X_3} = (1, 2)$ $R_{F_2 \to X_2} = (2, 1)$
$b_2 = (1.5, 1.5)$ $Q_{X_2 \to F_3} = (0.5, -0.5)$
$R_{F_3 \to X_3} = (-0.5, 0.5)$
$b_3 = (0.5, 2.5)$

Phase 4 (Max-sum_AD):

$b_3 = (0.5, 2.5)$ $Q_{X_3 \to F_1} = (-0.5, 0.5)$ $Q_{X_3 \to F_3} = (-0.5, 0.5)$
$R_{F_1 \to X_1} = (1.5, 1.5)$ $R_{F_3 \to X_2} = (0.5, -0.5)$
$b_2 = (2.5, 0.5)$ $Q_{X_2 \to F_2} = (0.5, -0.5)$
$R_{F_2 \to X_1} = (1.5, 1.5)$
$b_1 = (3, 3)$

Phase 5 (Max-sum_AD):

$b_1 = (0, 0)$ $Q_{X_1 \to F_1} = (0, 0)$ $Q_{X_1 \to F_2} = (0, 0)$
$R_{F_1 \to X_3} = (2, 1)$ $R_{F_2 \to X_2} = (1, 2)$
$b_2 = (1, 1.5)$ $Q_{X_2 \to F_3} = (-0.5, 0.5)$
$R_{F_3 \to X_3} = (0.5, -0.5)$
$b_3 = (2.5, 0.5)$

In each of these phases, X_2 and X_3 will either choose different values (phases 4 and 5) or there is indeterminacy that may cause them to choose different values (phase 3). (Note that this example also features the first phenomenon of indeterminacy, so that even if X_2 and X_3 choose the same value in phase 3, X_1 may choose a different value resulting in a suboptimal solution.)

At the end of the fifth phase, the contents of all communicated messages are exactly the same as those at the end of the first phase. Thus, the SPC messages and beliefs in the sixth phase will be the same as those in the second phase, and the algorithm will cycle. Note that X_2 has different SPC beliefs in the first and fifth phases. This is because those beliefs were based on $R_{F_3 \to X_2} = (0, 0)$ for the first phase (as initialized), and $R_{F_3 \to X_2} = (0.5, -0.5)$ for the fifth phase (as received from F_3 in the fourth phase). This difference has no impact on the execution of the algorithm since beliefs do not affect the outgoing messages.

Although the difference in cost between the optimal solution and the solution found by Max-sum_AD may seem small in this example, it is possible for it to be arbitrarily large in general. Because F_1 and F_2 optimistically take the minimum over possible value assignments of X_1, the magnitude of the dispreferred costs do not matter. Thus, the behavior of the agents and hence the solution that is found would be the same even if, for example, the cost of $C_{F_1}(a, b) = C_{F_1}(b, a) = C_{F_2}(a, b) = C_{F_2}(b, a) = 1000$ instead of 5. On such a problem, Max-sum_AD would find solutions of cost 1001.

Value propagation avoids inconsistency in the assumptions for calculating beliefs at variable-nodes by computing consistent costs at function-nodes. In the example, if X_1 had selected a value and communicated it to F_1 and F_2, the function-nodes would have been able to compute consistent costs based on this value for X_1 and send these costs to X_2 and X_3.

Max-sum_ADVP (variable-node X, order $\prec$, phase_length l)
1. *recipients* $\leftarrow \{F \in N_X : X \prec F\}$
2. initialize all $\{R_{F\to X} : F \in N_X\}$ to 0
3. *counter* $\leftarrow 0$
4. **while** (no termination condition is met)
5. *counter* $\leftarrow$ *counter* $+ 1$
6. **if** (not first iteration) collect message $R_{F\to X}$ from each $F \in N_X \setminus$ *recipients*
7. Update beliefs b_X according to Eq. (3) and value assignment $\widehat{d_X}$ according to Eq. (4)
8. **for each** $F \in$ *recipients*
9. produce message $Q_{X\to F}$ using messages $\{R_{F'\to X} : F' \in N_X \setminus \{F\}\}$ by Eq. (1)
10. **if** (first or second phase) send message $\langle Q_{X\to F}, \text{null}\rangle$ to F
11. **else** send message $\langle Q_{X\to F}, \widehat{d_X}\rangle$ to F
12. **if** (*counter* $= l$)
13. *recipients* $\leftarrow N_F \setminus$ *recipients*
14. *counter* $\leftarrow 0$

Fig. 7 Max-sum_ADVP for variable-nodes

Max-sum_ADVP (function-node F, order $\prec$, phase_length l)
1. *recipients* $\leftarrow \{X \in N_F : F \prec X\}$
2. initialize all received messages $\{Q_{X\to F} : X \in N_F\}$ to 0 and PA_F to the empty set
3. *counter* $\leftarrow 0$
4. **while** (no termination condition is met)
5. *counter* $\leftarrow$ *counter* $+ 1$
6. **if** (not first iteration)
7 **for each** $X \in N_F \setminus$ *recipients*
8 collect message $\langle Q_{X\to F}, \widehat{d_X}\rangle$
9. **if** ($\widehat{d_X}$ is not null) set $PA_F \leftarrow PA_F^{-X} \cup \{\langle X, \widehat{d_X}\rangle\}$
10. **for each** $X \in$ *recipients*
11. produce message $R_{F\to X}$ using C_F, $\{Q_{X'\to F} : X' \in N_F \setminus \{X\}\}$ and PA_F by Eq. (6)
12. send message $R_{F\to X}$ to X
13. **if** (*counter* $= l$)
14. *recipients* $\leftarrow N_F \setminus$ *recipients*
15. *counter* $\leftarrow 0$

Fig. 8 Max-sum_ADVP for function-nodes

This in turn would have affected the beliefs of X_2 and X_3, allowing them to improve their value selection and improve the quality of information being propagated through the factor graph. This is especially important in larger problems, where more distant nodes will not have direct knowledge of the value assignments that are selected but will still benefit from value propagation in each neighborhood.

6.2 Introducing value propagation into Max-sum_AD

We overcome the two pathologies we identified above by augmenting Max-sum_AD with a value propagation method similar to that used in complete GDL algorithms for avoiding ties [26,30,37]. The resulting algorithm is Max-sum_AD with Value Propagation (Max-sum_ADVP), whose pseudo-code is presented in Figs. 7 and 8 for variable-nodes and function-nodes, respectively.

The structure of Max-sum_ADVP is very similar to that of Max-sum_AD, except that the contents of the messages are altered to reflect the value propagation. Importantly, value propagation is not used from the beginning of the algorithm, but starts in the third phase. This is because effective value selection requires informed beliefs, and agents have not received information to allow them to consider all constraints in the problem until the end of the second phase. Indeed, our experimental study indicates that the best assignment found by Max-sum_AD (without value propagation) is at the end of the second phase. Were value

propagation to begin instead on the first iteration of the first phase, agents would have no information to base their choices on, and the premature selection and propagation of value assignments would likely prevent them from considering low cost solutions.

When using value propagation, each variable-node X explicitly selects a value $\widehat{d_X}$ in each iteration (Fig. 7, line 7) and sends it with the cost messages to recipient function-nodes (Fig. 7, line 11); the cost message itself is computed using Eq. (1), just as in standard Max-sum and Max-sum_AD. Each function-node F in turn maintains a partial assignment PA_F that stores the value selections of its neighbors. PA_F is initially empty (Fig. 8, line 2) and is updated with $\langle X, \widehat{d_X}\rangle$ whenever value selection $\widehat{d_X}$ is received from a neighbor $X \in N_F$ (Fig. 8, line 9).

Function-node F utilizes PA_F in generating $R_{F\to X}$ messages by minimizing only over assignments that are consistent with PA_F for the other variables $Y \in N_F \setminus \{X\}$. Formally, this space of possible assignments is the consistent domain of C_F given by

$$\mathrm{CD}(C_F, PA_F, X, d) = \{\mathbf{d} \in \mathrm{Dom}(C_F) : d_X = d \text{ and } \forall Y \in vars(PA_F) \setminus \{X\}(d_Y = \widehat{d_Y})\}$$

and the message generation for function-nodes in Max-sum_ADVP is

$$R_{F\to X}(d) = \min_{\mathbf{d}\in \mathrm{CD}(C_F, PA_F, X, d)} C_F(\mathbf{d}) + \sum_{Y\in N_F\setminus\{X\}} Q_{Y\to F}(d_Y). \tag{6}$$

The function-node uses Eq. (6) to generate the messages to recipient variable-nodes in Fig. 8, line 11.

To demonstrate the functioning of Max-sum_ADVP, consider again the example in Fig. 6. The first two phases proceed identically as described above for Max-sum_AD, as there is no value propagation in Max-sum_ADVP until the third phase. The SPC beliefs and messages for the third phase are as follows, assuming that X_1 selects a given its indifference between a and b.

Phase 3 (Max-sum_ADVP):

$b_1 = (3, 3), \widehat{d_{X_1}} = a$

$b_2 = (1.5, 6.5), \widehat{d_{X_2}} = a$

$b_3 = (0, 2)$

$Q_{X_1\to F_1} = (-1, 1)$ $R_{F_1\to X_3} = (1, 4)$ $Q_{X_2\to F_3} = (-2, 2)$ $R_{F_3\to X_3} = (-1, -2)$

$Q_{X_1\to F_2} = (1, -1)$ $R_{F_2\to X_2} = (2, 6)$

Note that value propagation has already caused the agents to converge within the phase to one of the two optimal solutions. This doesn't change in subsequent phases, as we show next.

Phase 4 (Max-sum_ADVP):

$b_3 = (0, 2), \widehat{d_{X_3}} = a$

$b_2 = (1.5, 4.5), \widehat{d_{X_2}} = a$

$b_1 = (4, 11)$

$Q_{X_3\to F_1} = (0.5, -0.5)$ $R_{F_1\to X_1} = (2.5, 5.5)$ $Q_{X_2\to F_2} = (0.5, -0.5)$ $R_{F_2\to X_1} = (1.5, 5.5)$

$Q_{X_3\to F_3} = (-1.5, 1.5)$ $R_{F_3\to X_2} = (-0.5, -1.5)$

Phase 5 (Max-sum_ADVP):

$b_1 = (4, 11), \widehat{d_{X_1}} = a$

$b_2 = (-1, 2), \widehat{d_{X_2}} = a$

$b_3 = (-1, 1)$

$Q_{X_1\to F_1} = (-2, 2)$ $R_{F_1\to X_3} = (0, 3)$ $Q_{X_2\to F_3} = (-2, 2)$ $R_{F_3\to X_3} = (-1, -2)$

$Q_{X_1\to F_2} = (-1.5, 1.5)$ $R_{F_2\to X_2} = (-0.5, 3.5)$

Phase 6 (Max-sum_ADVP):

$b_3 = (-1, 1), \widehat{d_{X_3}} = a$ $\quad Q_{X_3 \to F_1} = (0.5, -0.5)$ $\quad Q_{X_3 \to F_3} = (-1.5, 1.5)$

$R_{F_1 \to X_1} = (2.5, 5.5)$ $\quad R_{F_3 \to X_2} = (-0.5, -1.5)$

$b_2 = (-1, 2), \widehat{d_{X_2}} = a$ $\quad Q_{X_2 \to F_2} = (0.5, -0.5)$

$R_{F_2 \to X_1} = (1.5, 5.5)$

$b_1 = (4, 11)$

The messages and beliefs at the end of the sixth phase are identical to those at the end of the fourth phase and hence behavior in subsequent iterations will be cyclic. Moreover, observe that the SPC beliefs in phase 4 and phase 5 are equal up to an additive constant for each agent. That is to say, the relative valuations that each agent attributes to each of its domain values remains constant from phase 4 onward: $b_1(a) - b_1(b) = -7$, $b_2(a) - b_2(b) = -3$, and $b_3(a) - b_3(b) = -2$. Because it is the relative beliefs that matter in value selection, the agents' beliefs from phase 4 onward are functionally the same. In a sense, the agents have converged to a stable solution across phases. In the next subsection, we prove that this cross-phase convergence is not just an accident peculiar to this specific example but rather a guaranteed property of Max-sum_ADVP.

6.3 Establishing monotonicity and convergence

In order to prove the monotonicity of Max-sum_ADVP we first prove the following Lemmas:

Lemma 2 *Let F be a function-node with neighbor $X \in N_F$. In the fourth or later phase of Max-sum_ADVP, there exists a constant c such that the SPC message $R_{F \to X}$ satisfies*

$$R_{F \to X}(d) = C_F(PA_F^{-X} \cup \{\langle X, d\rangle\}) + c \qquad \forall d \in D_X.$$

Proof The fourth phase of Max-sum_ADVP is the second phase with VP, so by the time SPC is achieved, F has received value assignments from all neighbors. Thus, C_F is applicable to PA_F and Eq. (6) can be rewritten as

$$R_{F \to X}(d) = C_F(PA_F^{-X} \cup \{\langle X, d\rangle\}) + \sum_{\langle Y, \widehat{d_Y}\rangle \in PA_F^{-X}} Q_{Y \to F}(\widehat{d_Y}) \qquad \forall d \in D_X.$$

It is clear that the second term, $\sum Q_{Y \to F}(\widehat{d_Y})$, does not depend on d and hence is a constant with respect to the SPC message $R_{F \to X}$. □

An immediate corollary of Lemma 2 is that for each pair of values $d, d' \in D_x$,

$$R_{F \to X}(d) - R_{F \to X}(d') = C_F(PA_F^{-X} \cup \{\langle X, d\rangle\}) - C_F(PA_F^{-X} \cup \{\langle X, d'\rangle\}).$$

That is, the difference between the costs included in the message for different values are exactly the differences between the original costs in the constraint C_F represented by F for these values and the latest value assignment of F's variable-node neighbors that were sent to F.

Lemma 3 *The complete assignment in each VP phase of Max-sum_ADVP is equal to a complete assignment that would have been selected if agents had selected values sequentially.*

Proof We prove the claim for a specific order of the variable-nodes, namely $\prec_{\mathcal{X}}$ if the directionality of edges in the factor graph is according to $\prec$ (i.e., in odd-numbered phases

of Max-sum_ADVP), and the reverse order on variable-nodes if the directionality of edges is the reverse of $\prec$ (i.e., in even-numbered phases of Max-sum_ADVP). Since the two cases are symmetric, we prove the claim for the first case.

We prove the claim by complete induction on the ordinal rank of variable-nodes according to $\prec_{\mathcal{X}}$. The ordinal rank o_X of $X \in \mathcal{X}$ is $o_X = |\{X' \in \mathcal{X} : X' \prec_{\mathcal{X}} X\}|$. For the base case, suppose that $o_X = 0$. By the definition of the DAG, X must have no incoming edges and hence it receives no messages in the current phase. Thus, its beliefs do not change and so its selection of value assignment can be performed before that of any other variable-node, and so the base case is proved.

Suppose instead that $o_X > 0$ and that the claim is proved for all X' with $o_{X'} < o_X$. Let $X' \in \mathcal{X}$ be a variable-node that can reach X in the DAG. By construction of the DAG according to the order $\prec$ defined by Eq. (5), it follows that $o_{X'} < o_X$. By Lemma 1, the messages received by X' will not change after iteration $\ell_{X'} + 1$ of the current phase, so $\widehat{d_{X'}}$ also will not change after iteration $\ell_{X'} + 1$. Moreover, by inductive hypothesis, this value selection is equal to one performed sequentially according to $\prec_{\mathcal{X}}$.

Thus, on iteration $\ell_X + 1$ of the current phase, X has received messages from all in-neighbors based on all the value selections of nodes that can reach X. Furthermore, by Lemma 1, these messages will not change in subsequent iterations of the same phase.

Let Y be a variable-node constrained with X. Then by the definition of $\prec$, either $Y \prec X$ or $X \prec Y$. Thus, Y will either make its SPC value selection before X (if $Y \prec X$) or after X (if $X \prec Y$). Since Y and X will not make their value selection in the same iteration, the SPC value selection made by X in iteration $\ell_X + 1$ is equivalent to one made by a sequential process according to $\prec_{\mathcal{X}}$. Therefore, the inductive step and the lemma are proved. □

The crucial step in the proof of Lemma 3—that two constrained variable-nodes never make their SPC value selection in the same iteration—is dependent on our assumption of binary DCOPs in this paper. This facilitated the definition of $\prec$ resulting in each function-node lying between its neighboring variable-nodes in the DAG. The neighbors of constraints of higher arity cannot be ordered in such a manner since at least two of them must come after the function-node in one of the two DAG directions. Resolving this to establish convergence requires the addition of an auxiliary method to ensure that only one variable-node neighbor of each function-node replaces its assignment at a time. One method to do this distributedly would be for each function-node to create a single token that is passed between the variable-node neighbors, allowing them to change their value selections.

Lemma 4 *During the VP phases of Max-sum_ADVP, starting from the second VP phase, if at some iteration the value assignment of a single variable-node is replaced, then the global cost is reduced by an integral amount.*

Proof According to the corollary of Lemma 2, from the second VP phase, the differences between costs sent in messages from function-nodes to variable-nodes represent the differences between the original costs in the constraints, thus, an assignment replacement must decrease the local state of the variable (the sum of costs of the constraints it is involved in). Denote by C_i^- the set of constraints whose costs are reduced as a result of the assignment replacement of variable-node X_i and denote by $\Delta(C_i^-)$ the aggregated decrease in cost for these constraints caused by the replacement. Similarly denote by C_i^+ the set of constraints whose costs are increased following X_i's assignment replacement and denote by $\Delta(C_i^+)$ the aggregated increase in costs on these constraints. Obviously, $\Delta(C_i^-) > \Delta(C_i^+)$, otherwise X_i would not have replaced its value assignment. Since we assumed that no other variable-node replaces its assignment at this iteration, the cost of the global state is reduced as well. As all costs are integers, this reduction must also be integral. □

The properties stated in Lemmas 3 and 4 allow us to prove the following theorem:

Theorem 2 *After the second VP phase, Max-sum_ADVP is weakly monotonic improving.*

Proof According to Lemma 3, the value assignment selection made by the variable-nodes is equivalent to a sequential selection. According to Lemma 4, from the fourth phase (second VP phase) any change of a value assignment to a single variable causes an integral reduction in the cost of the global state. Thus, according to both lemmas we have a process that is equivalent to a sequential assignment selection process, in which at each iteration at most a single value assignment is replaced, and this replacement results in an integral reduction in the cost of the global state.[4] □

Corollary 1 (Cross-Phase Convergence) *There exists a phase* Φ *such that in all subsequent phases* $\Phi' > \Phi$*, the SPC assignment found by Max-sum_ADVP in phase* Φ' *is the same as the SPC assignment found in phase* Φ.

Proof Immediate from Theorem 2. The number of constraints in a DCOP is finite and the costs are finite as well. Thus, if in every phase there is a monotonic improvement in cost, this process must converge. □

Corollary 2 *Max-sum_ADVP achieves cross-phase convergence in a number of iterations that is pseudo-polynomial in the number of agents and maximum cost of a single constraint.*

Proof The number of constraints is polynomial in the number of agents because it is a binary DCOP so the magnitude of the maximum possible cost of an assignment is pseudo-polynomial in the number of agents and the maximum cost of a single constraint. Thus, if in every phase there is an integral improvement in cost and there are a polynomial number of iterations per phase, cross-phase convergence must occur in pseudo-polynomial time. □

7 Exploration methods

Max-sum_ADVP is purely exploitive both in selecting values given current beliefs and in updating those beliefs. While this guarantees monotonicity, the lack of exploration also restricts the solutions that are ultimately found. We therefore propose two classes of exploration methods that expand the set of candidate solutions and may thereby allow better solutions to be found. The first class introduces exploration into the selection of values based on current beliefs. The second class alters the message-passing protocol, to introduce more widespread exploration that may alter the agents' beliefs.

When analyzing the results per iteration of Max-sum_ADVP, one may notice that the first four phases incur dramatic reductions in the global cost. Thus, the exploration methods we proposed next were performed from the fifth phase and on. Attempts to begin sooner or later were not as effective.

7.1 Value selection exploration methods

The first class introduces exploration to the selection of values given current beliefs. These methods behave identically to Max-sum_ADVP for the first four phases: one phase in each

[4] We note that in standard Max-sum, the use of VP does not guarantee monotonicity since neighboring agents can replace assignments concurrently (as in DSA).

direction without value propagation followed by one phase in each direction with value propagation where a best value is selected as each agent's current value assignment, according to Eq. (4). Beginning in the fifth phase, and inspired by exploration in local search algorithms such as distributed simulated annealing [2,32], agents stochastically select value assignments from subsets of their domains. Formally, for every iteration i and variable node X with domain D_X, each method defines a subset $\widehat{D_X^i} \subseteq D_X$ of possible value assignments, and a sampling function $\Psi_X^i : \widehat{D_X} \to [0, 1]$ that specifies the probability of choosing each value in $\widehat{D_X^i}$, with $\sum_{d \in \widehat{D_X}} \Psi_X^i(d) = 1$.

Individual methods differ in the way that $\widehat{D_X^i}$ and Ψ_X^i are computed. While we considered many possible implementations, we found the following four to be the most successful:

– *K neighbors.* Variable-nodes select a value assignment uniformly at random from the set of the K best values according to their beliefs. Because making such choices may cause highly-constrained variable-nodes (i.e., those with many neighbors) to increase the costs of a large number of constraints, this exploration is only performed on variable-nodes that are constrained with at most a fraction $\phi \in (0, 1]$ of the total number of variables. Formally, the subset of possible value assignments is

$$\widehat{D_X^i} = \begin{cases} \underset{D' \subseteq D_X : |D'| = K}{\arg\min} \sum_{d \in D'} b_X^i(d) & \text{if } K < |D_X| \text{ and } |N_X| \leq \phi n \\ \left\{ \underset{d \in D_X}{\arg\min}\, b_X^i(d) \right\} & \text{otherwise} \end{cases}$$

and the sampling function is the uniform sampling function

$$\Psi_X^i(d) = \frac{1}{|\widehat{D_X^i}|} \qquad \text{for all } d \in \widehat{D_X^i}. \tag{7}$$

– *K depth.* The size of $\widehat{D_X^i}$ is based on ℓ_X, the length of the longest path to X in the DAG, decreasing with increasing ℓ_X from K down to 1. The actual elements in $\widehat{D_X^i}$ are selected greedily according to the current beliefs. Formally,

$$\widehat{D_X^i} = \begin{cases} \underset{D' \subseteq D_X : |D'| = K - \ell_X}{\arg\min} \sum_{d \in D'} b_X^i(d) & \text{if } K > \ell_X \\ \left\{ \underset{d \in D_X}{\arg\min}\, b_X^i(d) \right\} & \text{otherwise.} \end{cases}$$

If $K \leq \ell_X$, and there are more than one values $d \in D_X$ that minimize $b_X^i(d)$, one of them is selected so that $\widehat{D_X^i}$ is a singleton.
The sampling function is the uniform sampling function in Eq. (7).

– *Decreasing bias.* Instead of fixing the size of $\widehat{D_X^i}$ as the two previous methods did, the decreasing bias method includes all value assignments with beliefs within a predefined bias factor of the minimum belief. This bias factor is initially β and is reduced every two phases by an amount δ. The reduction of the bias every two phases reflects the fact that cost information fully cycles through the system every two phases, and the principle that exploitation should be increased at the expense of exploration as the algorithm progresses. Formally,

$$\widehat{D_X^i} = \begin{cases} \left\{ \arg\min_{d \in D_X} b_X^i(d) \right\} & \text{if } \beta - \delta \cdot (i \operatorname{div} 2l) < 0 \\ \left\{ d \in D_X : b_X^i(d) \leq \left(1 + \beta - \delta \cdot (i \operatorname{div} 2l)\right) \min_{d' \in D_X} b_X^i(d') \right\} & \text{otherwise} \end{cases}$$

where div is integer division and l is the phase length.
The sampling function is the uniform sampling function in Eq. (7).

– *SA*. This is an adaptation of the simulated annealing heuristic. Each variable-node X finds a best alternative d to its current value assignment $\widehat{d_X^i}$. If d is an improvement over $\widehat{d_X^i}$, then X replaces its value assignment. If not, X replaces its assignment with a probability calculated by the formula $e^{-\Delta/(i \bmod l)}$ where $\Delta = b_X^i(d) - b_X^i(\widehat{d_X^i})$ is the cost increment when replacing the value assignment, mod is the modulo operator, and l is the phase length, so that $(i \bmod l)$ is the number of iterations performed in the current phase.[5] The use of $i \bmod l$ resets the temperature of simulated annealing at the beginning of each phase, facilitating renewed exploration with each change of direction in communication. Formally,

$$\widehat{D_X^i} = \left\{ \widehat{d_X^i}, \arg\min_{d \in D_X \setminus \{\widehat{d_X^i}\}} b_X^i(d) \right\}$$

and

$$\Psi_X^i(d) = \begin{cases} 1 & \text{if } b_X^i(d) < b_X^i(\widehat{d_X^i}) \\ e^{-\Delta/(i \bmod l)} & \text{if } d \neq \widehat{d_X^i} \text{ and } b_X^i(d) \geq b_X^i(\widehat{d_X^i}) \\ 1 - e^{-\Delta/(i \bmod l)} & \text{otherwise.} \end{cases}$$

7.2 Message passing exploration methods

The second class of exploration methods explores the Max-sum family of algorithms by allowing the message-passing protocol to change in each phase. Although all members of this family are designed to be purely exploitive, in practice they achieve different degrees of exploration due to the presence or absence of different convergence and consistency guarantees. By executing different versions of the Max-sum algorithm, agents are able to take advantage of the differing balance of exploration and exploitation in each variant. We consider the following four variants from most explorative to least explorative:

1. *STD*–standard Max-sum. This has the most exploration, with no guarantee of convergence or consistency.
2. *VP*–standard Max-sum with the addition of value propagation. The use of value propagation reduces the inconsistent information being propagated through the factor graph, but there is still no guarantees on convergence due to cyclic information propagation.
3. *AD*–Max-sum_AD. Communicating on DAGs guarantees SPC but CPC is not guaranteed.
4. *ADVP*–Max-sum_ADVP. The most exploitive version of Max-sum considered, with both SPC and CPC guaranteed. Unlike the Max-sum_ADVP algorithm presented in Sect. 6.2, this variant will use value propagation in the first two phases if specified.

The variant to use in each phase is specified through a sequence $\langle \langle k_1, alg_1 \rangle, \langle k_2, alg_2 \rangle, \dots \rangle$ of pairs $\langle k_j, alg_j \rangle$ specifying that algorithm $alg_j \in \{STD, VP, AD, ADVP\}$ should be run for k_j phases.

[5] We are aware that in the literature there exist different versions of simulated annealing. We have implemented a variety of them and present the most successful.

For example, one instance of this class is $\langle\langle 2, AD\rangle, \langle 20, ADVP\rangle\rangle$, where Max-sum_AD is run for two phases, then Max-sum_ADVP is run for twenty phases. This combination actually replicates the standard Max-sum_ADVP algorithm as described in Sect. 6.2: agents perform Max-sum_AD for the first two phases and only commence value propagation beginning in the third phase.

Interleaving execution of the different algorithms can make use of their different convergence properties to improve solution quality. For example, interleaving execution according to $\{\langle 2, AD\rangle, \langle 10, ADVP\rangle, \langle 5, STD\rangle, \langle 5, VP\rangle \langle 10, ADVP\rangle\}$ is expected to converge to a local optimum (using $\langle 2, AD\rangle$ and $\langle 10, ADVP\rangle$), escape it (using $\langle 5, STD\rangle$ and $\langle 5, VP\rangle$) and converge again (using $\langle 10, ADVP\rangle$). If successful, the agents will cross-phase converge to a better solution using *ADVP* after the *STD* and *VP* phases, than they did when using Max-sum_ADVP initially.

The transition from one version to another needs to be performed with care since some versions are very different from the others and messages have different forms. For example, when transitioning from *ADVP* to *STD*, in the first iteration of the new *STD* phase, the agents will receive the *ADVP* messages sent in the last iteration of the previous, *ADVP* phase. These messages may contain value assignment selections that would not normally be received under the *STD* protocol. The myopic nature of Max-sum helps simplifying these transfers from one version to the other. Agents need to follow the following three guidelines in every iteration they perform, regardless of the version of the algorithm in the previous iteration:

1. When producing messages or selecting value assignments consider the most recently received message received from each neighbor.
2. Use only the information that is relevant to the current version of the algorithm.
3. If a message does not include information expected in the current version (e.g., a value assignment in *VP* or *ADVP*), interpret the message using the version of the algorithm that does not require this information, but generate new messages so that the current protocol can resume in the next iteration.

The above guidelines allow agents to execute the algorithm they are supposed to, regardless of the algorithm performed in the previous iteration. For example, when shifting from *ADVP* to *STD*, a function-node considers the messages received most recently from each of its variable-node neighbors, no matter how long ago that may have been, and ignores the value assignments the messages include. When shifting from *STD* to *ADVP* on the other hand, a function-node will expect value assignment selections to be included in messages sent from variable-nodes. However, in the first iteration these will not be included in the messages it receives. Thus, in the current iteration it performs *AD* (i.e., generating messages according to Eq. (2) instead of Eq. (6)) and starts *ADVP* (by generating messages according to Eq. (6)) in the next iteration. Variable-nodes starting *ADVP*, however, are required to send their value assignments in the first iteration of the phase, so that the function-nodes can begin running *ADVP* properly in the second iteration.

8 Experimental evaluation

In this section we present experiments comparing the performance of Max-sum_AD and Max-sum_ADVP to leading incomplete DCOP algorithms, including various versions of the Max-sum algorithm. After establishing the quality of the purely exploitive versions of Max-sum_AD and Max-sum_ADVP in comparison with existing algorithms, we investigate

the effects of adding exploration to these algorithms, and the effect of using them within the anytime framework proposed in [42].
The experiments were performed on four types of minimization problems commonly used in the DCOP literature. Each type of problem exhibits a different level of structure in the constraint graph topology and in the constraint functions. All problems were formulated as minimization problems.

1. *Uniformly random DCOPs*. Each agent holds a single variable with ten values in its domain. The constraint graph for each problem instance was generated randomly by adding a constraint between each pair of agents/variables independently with probability p_1. The cost of each pair of assignments of values to a constrained pair of variables was selected uniformly at random between 1 and 10. Such uniform random DCOPs with constraint graphs of n variables, k values in each domain, constraint density of p_1 and bounded range of costs/utilities are commonly used in experimental evaluations of centralized and distributed algorithms for solving constraint optimization problems [9, 18]. Both the constraint graph and the constraint functions are unstructured.
2. *Graph coloring problems*. These problems use random constraint graph topologies as in the random minimization DCOPs, but all constraints $C_{ij} \in \mathcal{R}$ are "not-equal" cost functions of the form

$$C_{ij}(d_i, d_j) = \begin{cases} 1 & \text{if } d_i = d_j \\ 0 & \text{otherwise} \end{cases}$$

   for all $d_i \in D_i, d_j \in D_j$. Such random graph coloring problems are commonly used in DCOP formulations of resource allocation problems [7,41]. Following the literature, we used $p_1 = 0.05$ and three values (i.e., colors) in each domain. While the constraint graph is unstructured, the constraint functions are highly structured.
3. *Scale-free network problems*. The constraint graph topology was generated using the Barabási–Albert (BA) model. An initial set of 10 agents was randomly selected and connected. Additional agents were added sequentially and connected to 3 other agents with a probability proportional to the number of links that the existing agents already had. The cost of each joint assignment between constrained variables was independently drawn from the discrete uniform distribution from 0 to 99. Each variable had 10 values in its domain and the total number of agents in each problem instance was $n = 50$. Similar problems were previously used to evaluate DCOP algorithms by Kiekintveld et al. [16]. The constraint graph is somewhat structured but the constraint functions are unstructured.
4. *Meeting scheduling problems*. Ninety agents scheduled 20 meetings into 20 time slots. Each agent was a participant in two randomly chosen meetings. For each pair of meetings, a travel time was chosen uniformly at random between 6 and 10, inclusive. When the difference between the time slots of two meetings is less than the travel time between those meetings, any participants in both meetings are overbooked, and a cost equal to the number of overbooked agents is incurred. These realistic problems are identical to those used by Zivan et al. [42]. Both the constraint graph and the constraint functions are highly structured.

In all experiments presented we depict the cost of the solution that the different algorithms would have selected as a function of the number of iterations performed. In all the versions of Max-sum in our experiments we used the random personal preferences method for breaking ties that was suggested by Farinelli et al. [7]. In algorithms of the Max-sum family, agents performed the roles of the variable-nodes representing their own variables, and each function-

node was performed by the agent with the smaller index who held a variable involved in its constraint.

8.1 Comparison of exploitive algorithms

We first empirically establish the effectiveness of the purely exploitive versions of Max-sum_AD and Max-sum_ADVP. We compared them to leading incomplete algorithms: standard Max-sum [7], standard Max-sum with damping factor 0.8 [19], Bounded Max-sum [30], DSA-C [41] and ADPOP [27]. Unless otherwise noted, ADPOP was limited to $maxDims = 3$ due to practical considerations because the maximum table size (and hence the running time) is exponential in twice the value of $maxDims$, as described in Sect. 2.

We also compared our algorithms with the solutions found using the primal and dual formulations of a standard linear programming relaxation (LPR) [10]. The combinatorial solutions were obtained from the LP solutions by choosing the assignments that maximized the agents' beliefs (directly for the primal formulation and using the same method as EMPLP for the dual formulation [10]). Many recent inference algorithms such as MPLP [10] and TRBP [39] attempt to converge to the solutions to the dual formulation and hence the dual LPR is expected to provide higher quality assignments on average than any of these algorithms. However, we found that the assignments obtained using the primal formulation had costs no worse and in many cases much better than those obtained using the dual formulation, and hence we report only the results obtained using the primal LPR.

In each experiment we report the solution cost averaged over 200 randomly-generated problem instances for the algorithms when run for 2000 iterations. For Max-sum_AD and Max-sum_ADVP, we used phase length of $2n$ to guarantee SPC, although in most cases they converged much faster. Note that because LPR, Bounded Max-Sum and ADPOP are not iterative algorithms, we report the solutions found by those algorithms at the end of their execution.

Figure 9 presents the costs of the solutions when solving uniformly random DCOPs with $n = 50$ and a relatively low constraint density of $p_1 = 0.2$. It is most apparent that the standard Max-sum algorithm does not converge and instead traverses complete assignments with high costs. LPR and Bounded Max-sum also do poorly, finding solutions only slightly better than those of Max-sum. Max-sum_AD converges to solutions of lower cost than any of these three

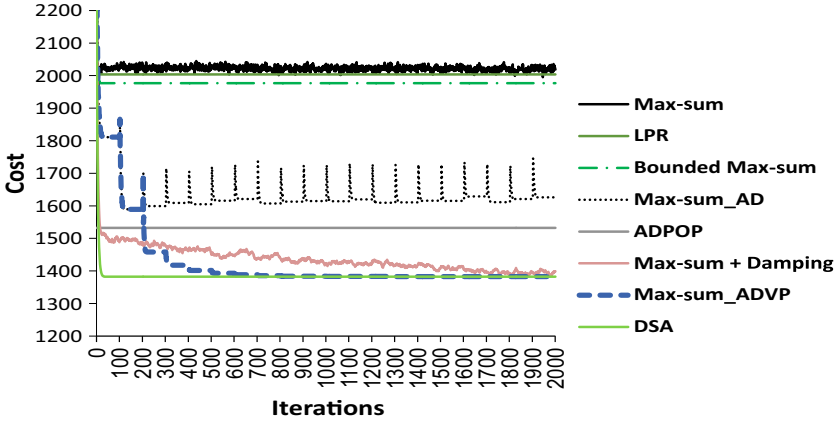

Fig. 9 Solution cost when solving uniformly random DCOPs with low density ($p_1 = 0.2$)

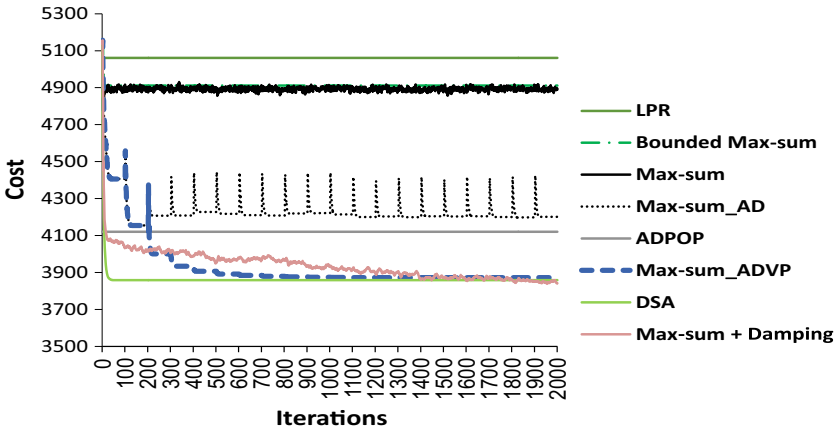

Fig. 10 Solution cost when solving uniformly random DCOPs with high density ($p_1 = 0.6$)

algorithms in the first phase, and continues to find much improved solutions in the second phase. However, in the third and subsequent phases, the SPC assignments have costs similar to or even higher than those of the second phase. These solutions also have higher costs than those found by ADPOP. Max-sum with damping also does not converge, however, it explores solutions of much lower cost than standard Max-sum, and produces solutions with lower cost than ADPOP. DSA and Max-sum_ADVP find the solutions with the lowest costs. Since we begin value propagation only after the second phase, Max-sum_ADVP performs identically to Max-sum_AD in the first 200 iterations. However, in the third phase, Max-sum_ADVP converges to solutions with much lower cost than Max-sum_AD does. In the following phases it is observable that Max-sum_ADVP monotonically improves, as we proved in Sect. 6.3, until it finally converges to the solution with the lowest cost after the seventh phase.

Figure 10 presents the results for uniformly random problems with $n = 50$ and a higher constraint density of $p_1 = 0.6$. LPR does very poorly, suggesting that the inference algorithms that approximate it will be of little use for such dense, unstructured problems. Bounded Max-sum and standard Max-sum find solutions with similar costs, in contrast to the results on sparse problems showing Bounded Max-sum narrowly outperforming standard Max-sum. This is reasonable because Bounded Max-sum must remove more edges in denser problems and thus, more constraints are ignored when producing the solution. The relative performance of the other algorithms is similar to that for sparse problems, with Max-sum_ADVP again finding solutions of low cost, as do DSA and Max-sum with damping. However, Max-sum with damping does even better than on low density problems, and in final iterations of the run, outperforms Max-sum_ADVP.

The next set of experiments tested the incomplete inference algorithms on graph coloring problems with $n = 50$ agents, density $p_1 = 0.05$, and three colors. Because these problems are much sparser with much smaller domains than the uniformly random problems, it was possible to run ADPOP with $maxDims = 8$. The results presented in Fig. 11 are similar to the results presented for random problems, but standard Max-sum and LPR perform exceptionally poorly. As we mentioned in Sect. 6.1, when solving this type of problems, the only information communicated by Max-sum agents are their random tie-breaking preferences and thus the solutions selected are arbitrary. LPR fails for similar reasons: the many ties result in poor solutions when rounding the fractional LP solutions to obtain discrete value

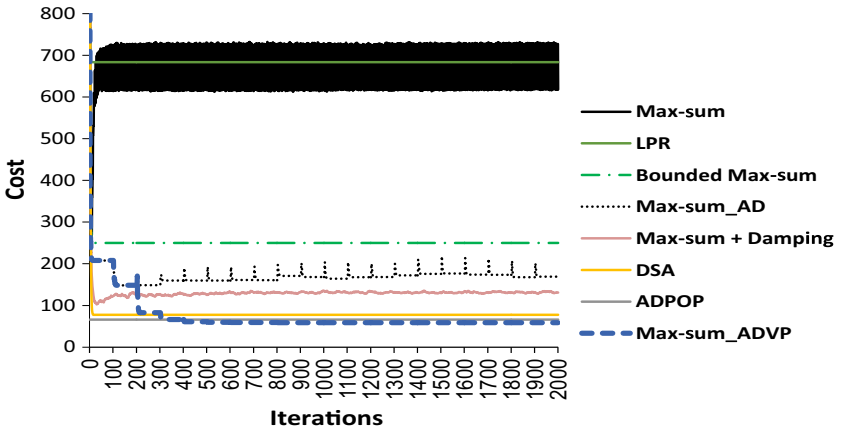

Fig. 11 Solution cost when solving graph coloring problems

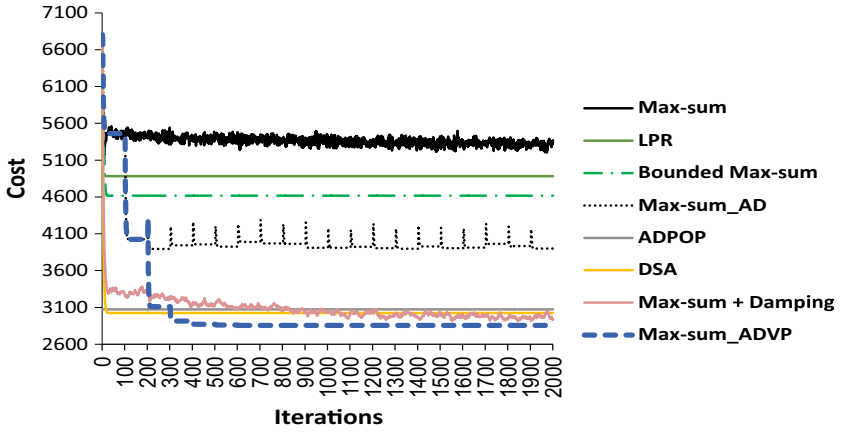

Fig. 12 Solution cost when solving scale-free networks

assignments. Max-sum with damping does much better than standard Max-sum, but here it fails to improve while exploring after the first few iterations. The other algorithms do not suffer this shortcoming and hence find much better solutions than Max-sum and LPR. Their performance relative to each other is similar to that for the uniformly random problems, although ADPOP and Max-sum_ADVP find solutions of very similar cost. ADPOP's strong performance is due to the high value of *maxDims* that we were able to use for these problems, as it allows closer approximation of the true cost tables passed by the complete DPOP algorithm. Max-sum_ADVP still finds the best solutions on average, and its advantage over ADPOP, while slight, is statistically significant at the $p = 0.01$ level.

The next set of experiments were on scale-free networks with the results presented in Fig. 12. As expected, standard Max-sum does not exhibit the pathology it exhibited when solving graph coloring problems. However, all other algorithms outperform it and the differences between their results are similar to the differences between the results produced in the experiments presented above, with Max-sum_ADVP again dominating. It is notable that Bounded Max-sum has an advantage over standard Max-sum on structured problems.

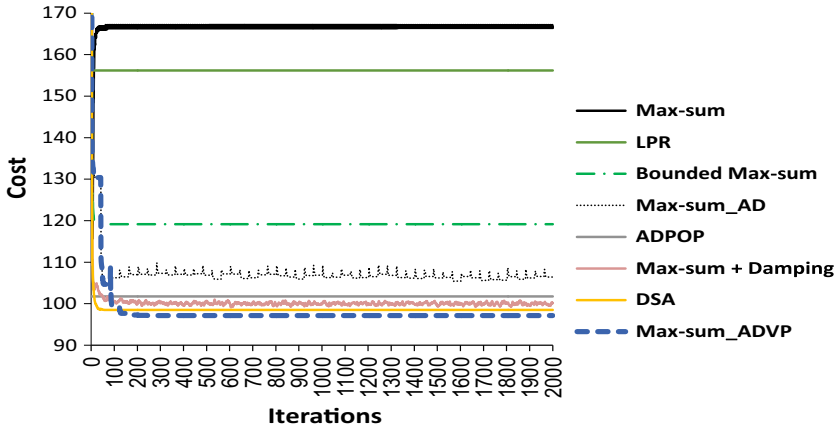

Fig. 13 Solution cost when solving meeting scheduling problems

Max-sum_ADVP outperforms both DSA and ADPOP in these settings and Max-sum with damping gets close in the latest iterations of its run.

The final type of problems we considered were the meeting scheduling problems, with results presented in Fig. 13. The results are generally similar to those for graph coloring, which is not surprising because the cost structure contains soft mutual exclusion constraints over subsets of the domains to penalize overlapping meetings. Standard Max-sum and LPR both find very bad solutions, although Max-sum is worse by a considerable margin. Max-sum_ADVP again finds better solutions than all competing incomplete algorithms.

8.2 Comparison with optimal and scalability

We next compare the solutions found by the incomplete algorithms to the optimal solutions. The optimal solutions were found by solving a mixed integer program formulation of the problems using a modern, centralized, commercial solver, Gurobi 5.6. Although this solver is highly optimized, the inherent complexity of the DCOPs restricted these experiments to smaller uniformly random problems with 10 agents ($n = 10$). The problems generated were either sparse with $p_1 = 0.2$ or dense with $p_1 = 0.6$. In both settings each agent held a single variable with domain size of 10.

Figures 14 and 15 present the ratios of the costs achieved by the incomplete algorithms to the optimal costs. On sparse problems ADPOP does best, finding solutions with an average approximation factor of 1.2 of the optimal cost. Max-sum_ADVP also does very well on average, finding solutions with aproximation factor of 1.3 of the optimum. These two algorithms have a significant advantage over all other algorithms including DSA and Max-sum with damping. On dense problems Max-sum_ADVP outperforms ADPOP, with an approximation factor of roughly 1.2. However, DSA and Max-sum with damping also do well. LPR and Bounded Max-sum produce similar results and outperform standard Max-sum on both settings. Interestingly, standard Max-sum, Max-sum with damping and Max-sum_AD seem to be slowly improving with the increasing number of iterations; this effect is more pronounced with dense problems than with sparse problems. Overall, standard Max-sum still does very poorly (even worse relatively speaking than in the larger problems), and Max-sum_AD at best matches the performance of ADPOP in dense problems. However, in dense problems, Max-sum with damping outperforms Max-sum_ADVP towards the end of the run.

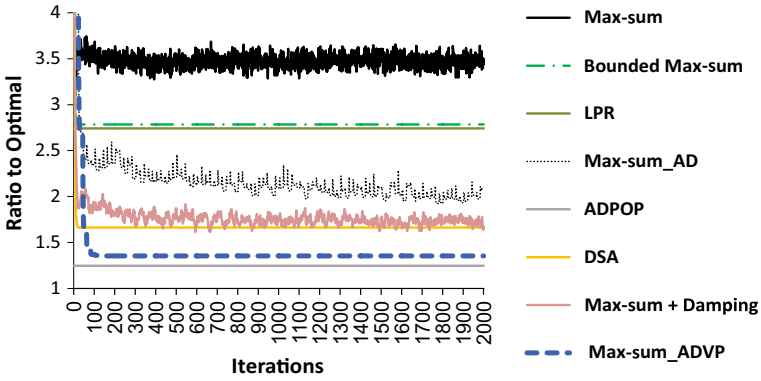

Fig. 14 Ratio of costs to optimal when solving small uniformly random DCOPs with low density ($p_1 = 0.2$)

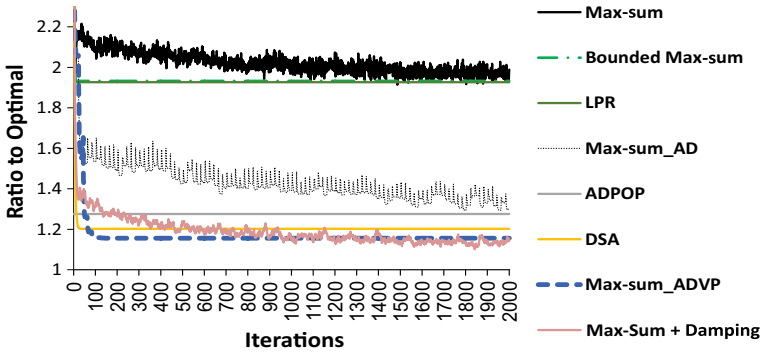

Fig. 15 Ratio of costs to optimal when solving small uniformly random DCOPs with high density ($p_1 = 0.6$)

Overall, the trends for the incomplete algorithms appear to be roughly consistent with the results for larger problems in Figs. 9 and 10, which suggests that their performance, relative to the optimal in those problems (where it is prohibitively expensive to compute the optimal solution), may also be similar.

The graph coloring problems are easier to solve than the uniformly random problems, due to their structure, constraint sparseness, and smaller domains. This allowed us to compute the optimal solutions to the full $n = 50$ problems. Because 19% of the problems were perfectly colorable (i.e., had optimal solutions of cost 0), it is not meaningful to consider approximation ratios. Instead we note that Max-sum_ADVP found solutions that on average violated only 4.225 more constraints than optimal.

We were also able to compute the optimal solutions to the meeting scheduling problems due to their considerable structure. Because these problems do not have any solutions with 0 cost, it is meaningful to consider approximation ratios. Max-sum_ADVP achieved an average approximation ratio of 1.027 on these problems, getting very close to optimal. The plot of the approximation ratios is very similar to Fig. 13 and therefore is omitted.

Figure 16 presents the performance of the incomplete inference algorithms solving large, sparse, uniformly random problems with $n = 70$. It is clear that the broad trends, with Max-sum_AD outperforming standard Max-sum and Bounded Max-sum, Max-sum_ADVP performing similar to DSA and Max-sum with damping slowly improving until achieving

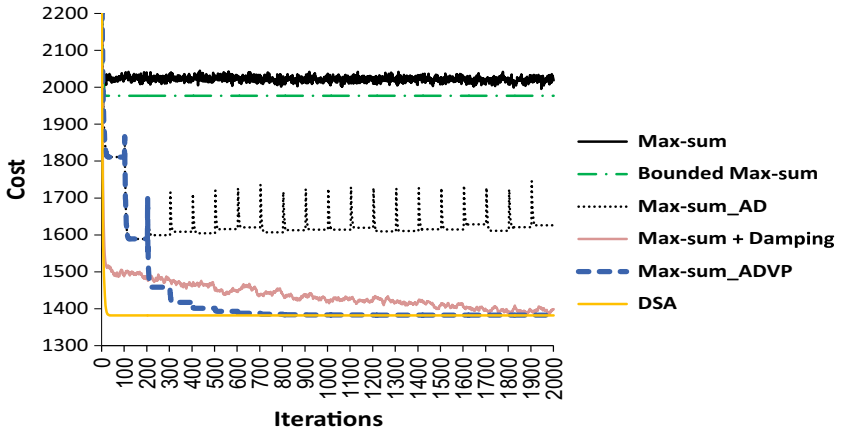

Fig. 16 Solution quality when solving large uniformly random DCOPs with low density ($p_1 = 0.2$)

similar results to DSA and Max-sum_ADVP, continue at this larger problem size. Combined with the results in Figs. 9 and 14, this strongly suggests that our empirical findings hold over a wide range of problem sizes and will continue to apply as the problems scale. We obtained similar results when scaling the other problem types and omit them to avoid redundancy.

8.3 Comparison of exploration methods

After establishing the dominance of the exploitive Max-sum_ADVP algorithm over all other converging incomplete inference algorithms, we move on to check whether the exploration methods proposed in Sect. 7 can improve its performance further. We first compare to the value selection exploration methods, then the message passing exploration methods.

The value selection exploration methods were evaluated for different parameter values, and we present the results using the most successful settings. We found these to be $K = 2$ and $\phi = 0.5$ for K neighbors; $K = 3$ for K depth; and $\beta = 0.5$ and $\delta = 0.1$ for decreasing bias. There were no tunable parameters for SA, although we also considered several alternative variants (e.g., randomly choosing a candidate alternative selection); the form described in Sect. 7 performed best.

Figure 17 presents results for sparse random problems with $n = 50$ and $p_1 = 0.2$. The results demonstrate that K neighbors and SA were not beneficial. The curve for SA is similar in appearance to that of Max-sum_AD in previous experiments. This is because SA also demonstrates SPC due to decreasing temperature, but no CPC due to the temperature resetting at the beginning of each phase. Decreasing bias, which transitions from exploration to exploitation on a cross-phase level, initially performs extremely poorly, but eventually improves to be very close to Max-sum_ADVP. K depth performs very similarly to Max-sum_ADVP for most of its execution and ends with just slightly better solutions on average. The differences at the end of execution, although very narrow, are all statistically significant at the $p = 0.01$ level.

Similar results were achieved on the different dense random problems, with the exception of K neighbors, which performed little exploration because most nodes have high degree and thus do not satisfy the condition that $|N_X| \leq \phi n$. K neighbors thus performed very similarly to Max-sum_ADVP. We omit the graph to avoid redundancy.

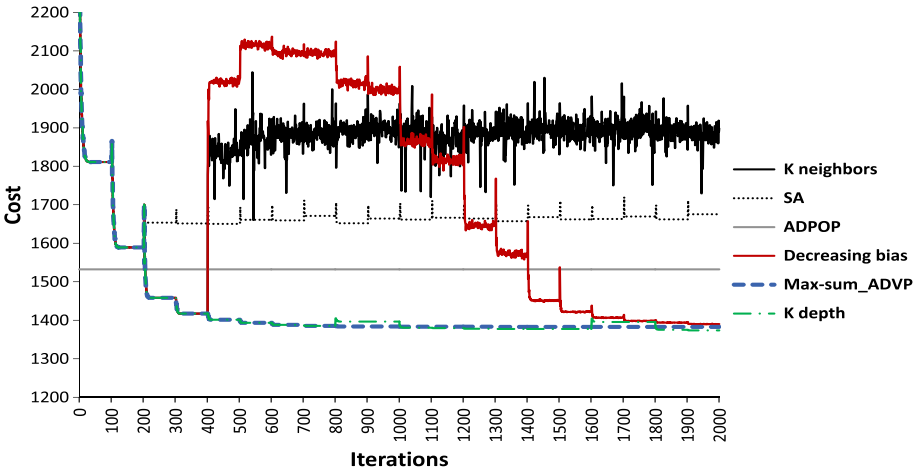

Fig. 17 Solution cost of Max-sum_ADVP compared to value selection exploration methods when solving random DCOPs with low density ($p_1 = 0.2$)

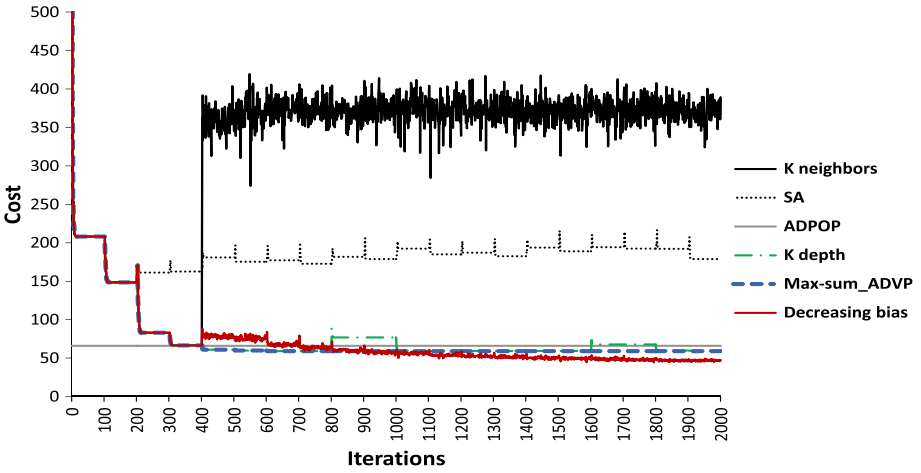

Fig. 18 Solution cost of Max-sum_ADVP compared to value selection exploration methods when solving graph coloring problems

K neighbors and SA were not beneficial when solving graph coloring problems, as shown in Fig. 18. K depth again performed similarly to Max-sum_ADVP but ultimately found slightly worse solutions. Decreasing bias did not explore extremely bad solutions as it did with the random problems, and its improvement across phases eventually led it to find even better solutions than Max-sum_ADVP. Again, all apparent differences at the end of execution were statistically significant at the $p = 0.01$ level. Similar results were also found for scale-free networks (Fig. 19) and meeting scheduling problems (Fig. 20).

The results obtained with the value selection exploration heuristics support two broad conclusions. First, techniques designed to maintain CPC are preferable to those which demonstrate only SPC or no convergence at all. This is most clearly seen in the comparison between Decreasing bias, SA, and K neighbors. Second, these methods, while inspired by approaches

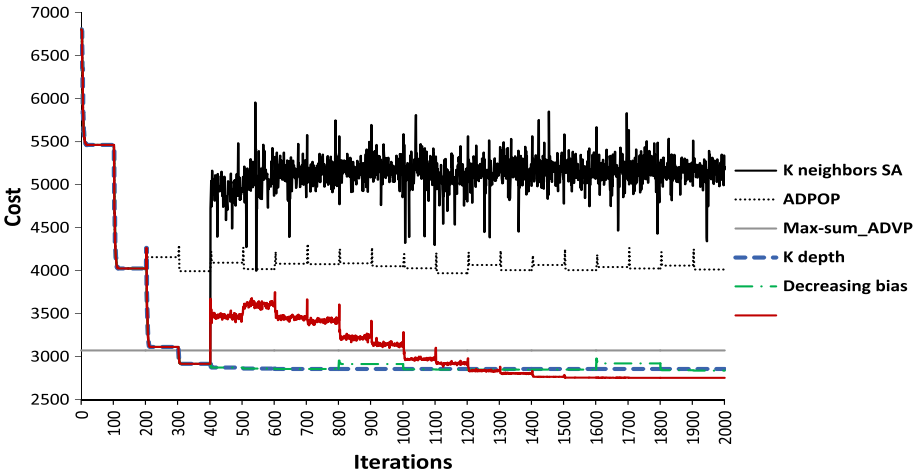

Fig. 19 Solution cost of Max-sum_ADVP compared to value selection exploration methods when solving scale-free network problems

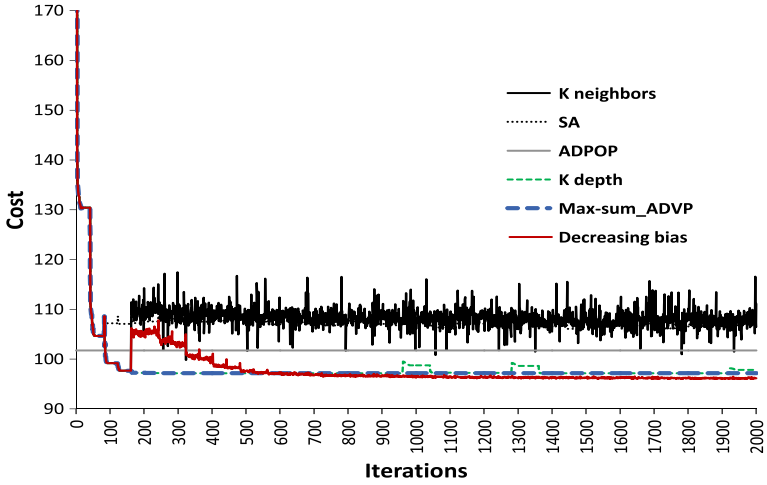

Fig. 20 Solution cost of Max-sum_ADVP compared to value selection exploration methods when solving meeting scheduling problems

successfully employed to introduce exploration to local search algorithms, are able to provide only limited amounts of useful exploration to Max-sum_ADVP. This can be seen both in the modest improvements achieved by K depth in uniformly random problems and Decreasing bias in graph coloring problems, and also in the relatively low values of K that were most successful with K neighbors and K depth. Because solution quality degrades as K is increased, the most successful forms of K neighbors and K depth tend to behave quite similarly to Max-sum_ADVP.

Next we present a comparison between Max-sum_ADVP and the message passing exploration methods, which include different combinations of converging and non-converging algorithms from the Max-sum family. We consider three specific combinations:

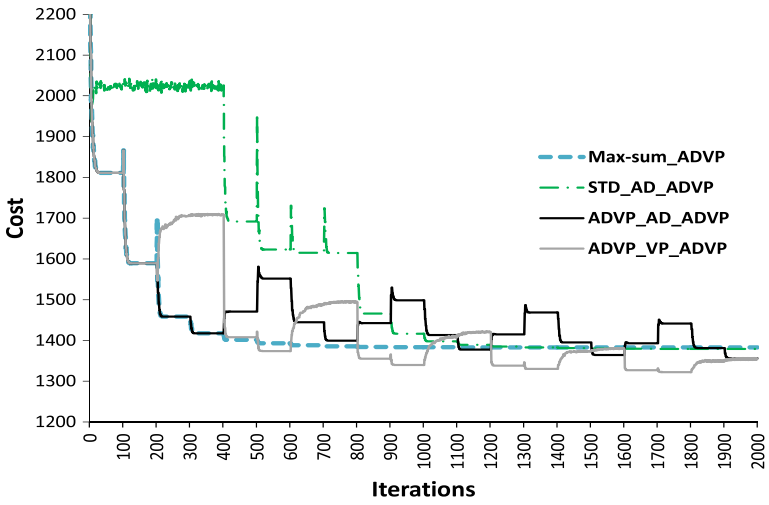

Fig. 21 Solution cost of Max-sum_ADVP and message passing exploration methods when solving random DCOPs with low density ($p_1 = 0.2$)

- ADVP_VP_ADVP $= \langle\langle 2, AD\rangle, \langle 2, ADVP\rangle, \langle 2, VP\rangle, \langle 2, ADVP\rangle, \langle 2, VP\rangle, \ldots\rangle$, alternating two phases of standard Max-sum with value propagation and two phases of Max-sum_ADVP until termination.
- STD_AD_ADVP $= \langle 4, STD\rangle, \langle 4, AD\rangle, \langle 12, ADVP\rangle\rangle$.
- ADVP_AD_ADVP $= \langle\langle 2, AD\rangle, \langle 2, ADVP\rangle, \langle 2, AD\rangle, \langle 2, ADVP\rangle \ldots\rangle$, alternating two phases of Max-sum_AD and two phases of Max-sum_ADVP until termination.

Figure 21 presents a comparison of Max-sum_ADVP with these three message passing explorative methods on sparse, random DCOPs. The graph curves illustrate the convergence properties of the algorithms. While *AD* and *ADVP* converge in each phase, the *STD* and *VP* versions do not. Moreover, the only version that exhibits CPC is *ADVP*. ADVP_VP_ADVP and ADVP_AD_ADVP both find statistically better solutions than Max-sum_ADVP upon termination. There is no significant difference between the quality of solutions found by STD_AD_ADVP and Max-sum_ADVP at the end of execution.

The results for dense, random problems are very similar to those for sparse problems and are omitted. Max-sum_ADVP and STD_AD_ADVP again perform statistically equally, while the other two exploration methods outperform Max-sum_ADVP at the $p = 0.01$ significance level.

Figures 22, 23 and 24 present the results of the message passing exploration methods when solving graph coloring, scale-free networks and meeting scheduling problems, respectively. Overall, the results are broadly similar to those on the uniformly random problems. On graph coloring and meeting scheduling problems, ADVP_VP_ADVP explores very low quality solutions during its *VP* phases. This strongly suggests that value propagation alone is not sufficient for standard Max-sum to escape its pathologies on these types of problems as described in Sect. 6.1. Its exploration is much more successful on the scale-free networks, as it was on the uniformly random problems. ADVP_AD_ADVP improves Max-sum_ADVP on both scale-free networks and graph coloring problems. However, ADVP_AD_ADVP's advantage over Max-sum_ADVP on graph coloring problems is very slight although statistically significant. Coupled with the success of exploration methods on the uniformly

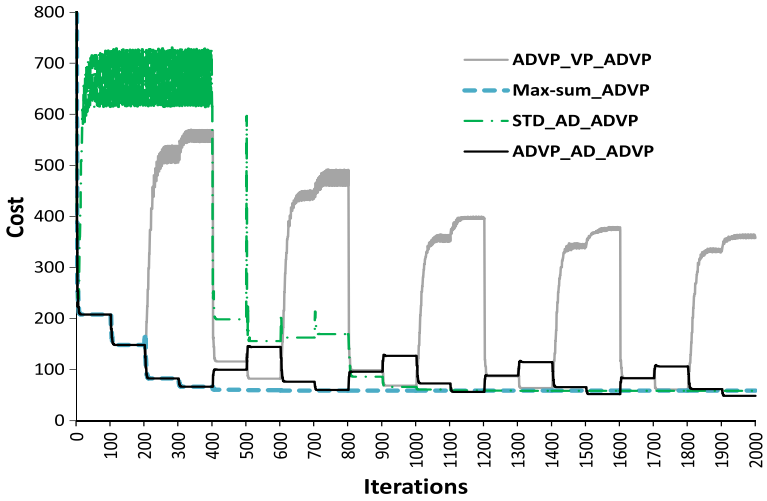

Fig. 22 Solution cost of Max-sum_ADVP and message passing exploration methods when solving graph coloring problems

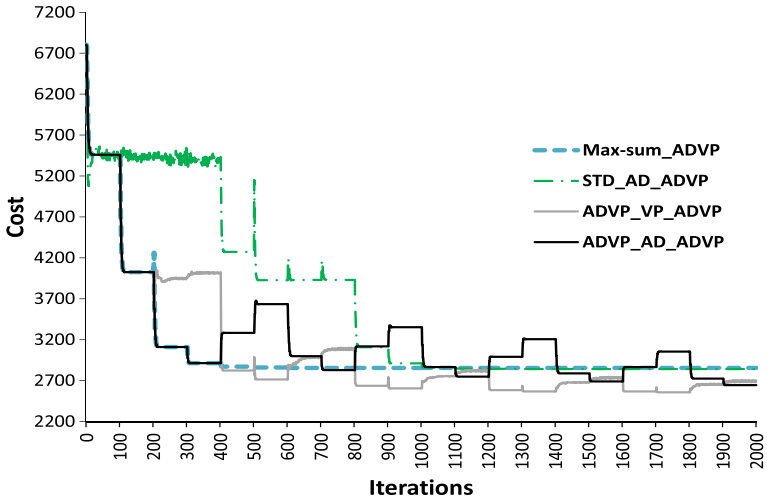

Fig. 23 Solution cost of Max-sum_ADVP and message passing exploration methods when solving scale-free networks

random problems, this set of experiments suggests that the absence of constraint structure is a more significant factor in the effectiveness of message passing exploration methods than the constraint topology.

8.4 Anytime results

While the advantages of the proposed Max-sum_ADVP algorithm over standard Max-sum and Bounded Max-sum, are obviously considerable, the comparison with Max-sum with damping and with Max-sum_ADVP with exploration heuristics is not conclusive. Not only are the results often close, but the exploration also makes it hard to determine when (after

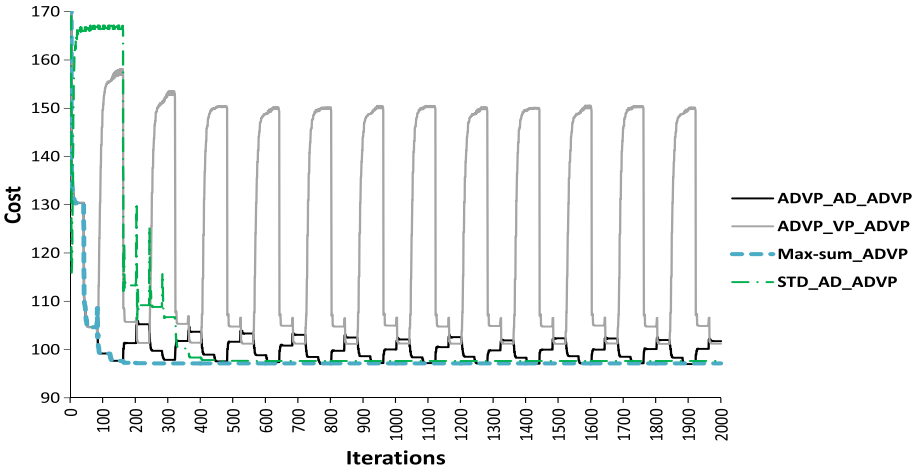

Fig. 24 Solution cost of Max-sum_ADVP and message passing exploration methods when solving meeting scheduling problems

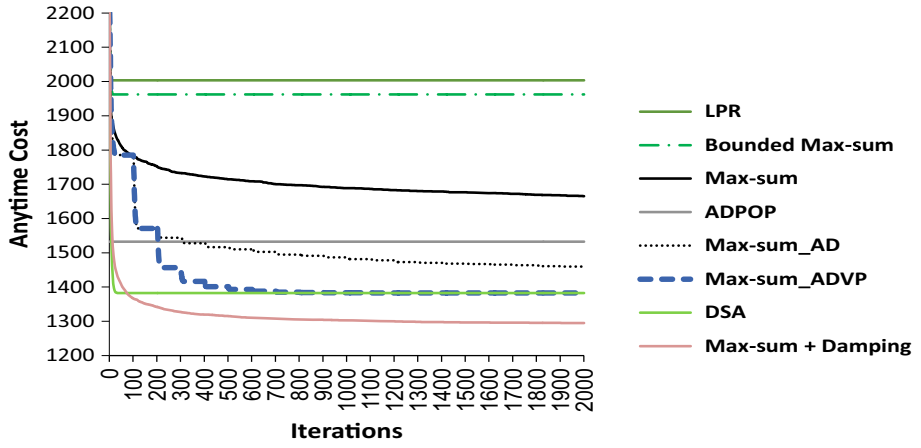

Fig. 25 Anytime cost of the Max-sum versions solving random DCOPs with low density ($p_1 = 0.2$)

how many iterations) to evaluate the proposed heuristics, since at any point in time some of the algorithms may be in an explorative phase. To this end we implemented the anytime framework proposed by Zivan et al. [42] and implemented all algorithms we experimented with above within the framework. We could then perform paired difference tests to statistically determine which algorithms had found significantly better solutions at any point during execution.

Figure 25 presents the anytime results of the incomplete algorithms on sparse random problems. It is clear that the exploration by Max-sum and Max-sum_AD that is evident in Fig. 9 is actually productive, resulting in decreasing anytime costs as the algorithms proceed. This exploration allows Max-sum_AD to surpass ADPOP. While versions of Max-sum that are guaranteed to converge, i.e., LPR, Bounded Max-sum and Max-sum_ADVP do not benefit from the anytime implementation, Max-sum with damping (like standard Max-sum and Max-sum_AD) performs exploration that is captured by the anytime mechanism and produces a

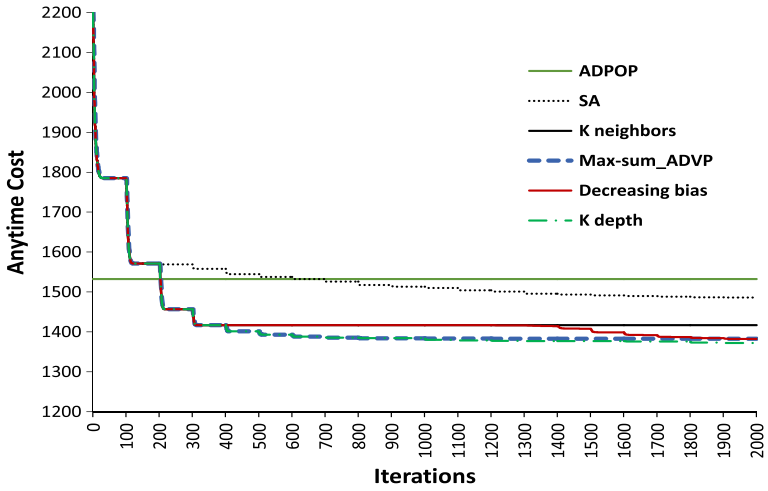

Fig. 26 Anytime cost of Max-sum_ADVP and value selection exploration methods when solving random DCOPs with low density ($p_1 = 0.2$)

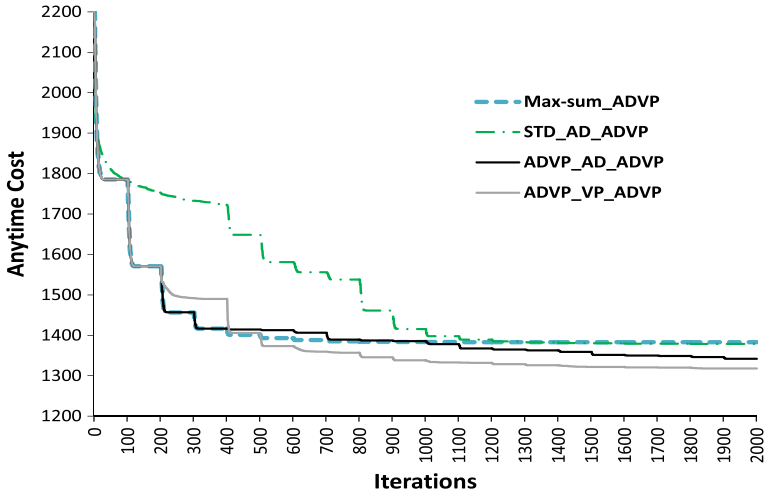

Fig. 27 Anytime cost of Max-sum_ADVP and message passing exploration methods when solving random DCOPs with low density ($p_1 = 0.2$)

significant improvement over Max-sum_ADVP. This suggests (as we discuss at the end of this section), that in contrast to the assumption that belief propagation algorithms perform best when they converge, effective exploration may produce better solutions.

Figure 26 presents the anytime results for the value selection exploration methods. These results show that SA performs useful exploration, finding better solutions than ADPOP over the course of execution. While the anytime framework does help the value selection exploration methods, allowing K depth to find better solutions than Max-sum_ADVP at a significance level of $p = 0.01$, the actual difference is still very small for these problems.

Figure 27 presents the anytime results for the message passing exploration methods. These results confirm that ADVP_AD_ADVP and ADVP_VP_ADVP continue to improve

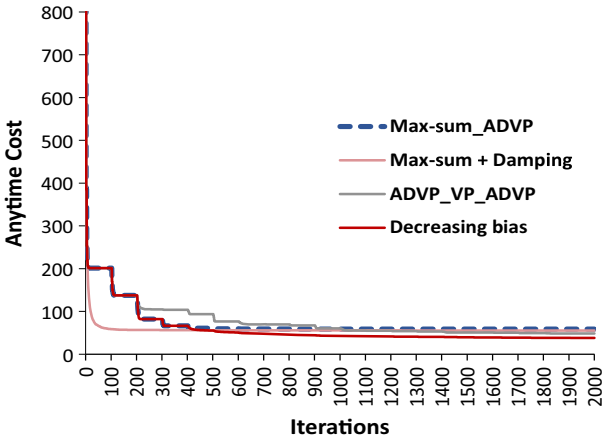

Fig. 28 Anytime cost of Max-sum_ADVP, Max-sum with damping and the best exploration methods when solving graph coloring problems

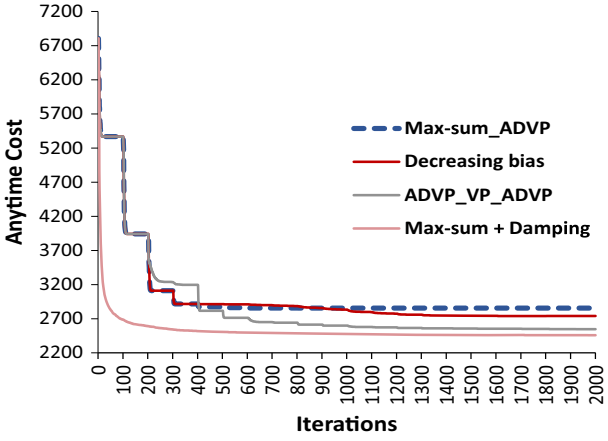

Fig. 29 Anytime cost of Max-sum_ADVP, Max-sum with damping and the best exploration methods when solving scale free net problems

over time through exploration. In contrast, STD_AD_ADVP appears to have converged to solutions of the same quality as Max-sum_ADVP.

To avoid redundancy, we do not present the anytime results for dense random uniform problems. The trends seen in Figs. 25, 26, and 27 concerning which algorithms continue to improve through exploration hold in the dense problems as well.

In Figs. 28, 29 and 30 we present the results of the best exploration methods of both types, Max-sum_ADVP and Max-sum with damping, on the structured and realistic benchmarks. The results on these setups were much less conclusive than on the random uniform problems. On scale free nets, Max-sum with damping still has an advantage over Max-sum_ADVP with the exploration methods, but the results of ADVP_VP_ADVP are quite close. On meeting scheduling problems, Decreasing bias produces almost similar anytime results to the results produced by Max-sum with damping. Finally, on graph coloring problem, Decreasing bias has a small advantage, although once again, the results are very close.

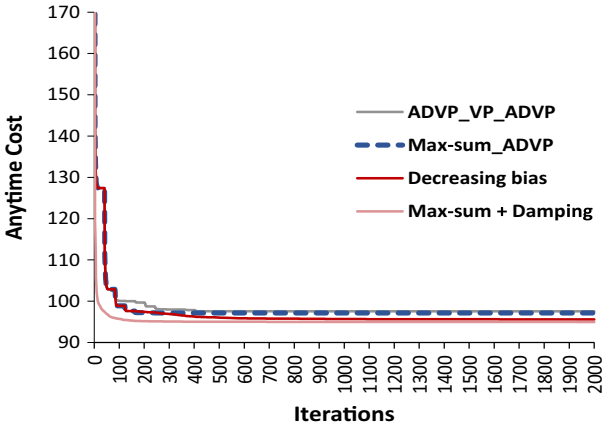

Fig. 30 Anytime cost of Max-sum_ADVP, Max-sum with damping and the best exploration methods when solving meeting scheduling problems

8.5 Discussion

Belief propagation literature, including the papers that apply Max-sum to DCOP, follow a consistent thesis, that these algorithms perform best when they converge. Thus, this study follows others within the DCOP community and in the larger graph-model community, to design a guaranteed convergence version of Max-sum that would produce high quality results. Our empirical study indicates that we have succeeded in this task. The Max-sum_ADVP algorithm converges to solutions with much higher quality than other guaranteed convergence versions such as Bounded Max-sum and LPR. It also performed better than ADPOP and DSA on most benchmarks.

Nevertheless, our previous experience with incomplete DCOP algorithms encouraged us to investigate further, and reveal whether a balanced combination between exploration and exploitation can produce higher quality results, and be captured using the anytime mechanism we proposed in a previous research. When considering the actual average results per iteration, Max-sum_ADVP outperformed standard Max-sum and Max-sum with damping, on most benchmarks. The most successful exploration methods we proposed improved it (slightly) further. However, when we used Max-sum with damping in combination with the anytime mechanism, the results on random uniform problems and on scale free nets significantly outperformed the guaranteed convergence versions of Max-sum, and on graph coloring problems produced similar results to Max-sum_ADVP. The comparison with the exploration methods we proposed revealed that on random uniform problems Max-sum with damping performs more efficient exploration, while on realistic structure problems, this advantage is not conclusive.

The significant advantage of Max-sum with damping, when combined with an anytime mechanism, over guaranteed convergence incomplete inference algorithms on some benchmarks, indicates that the basic common assumption that belief propagation performs best when it converges may not always hold, and that although we were able to generate the most successful guaranteed converging version of Max-sum, the most powerful property of Max-sum is its ability (in some versions) to perform efficient exploration.

9 Conclusion

The Max-sum algorithm offers an innovative approach for solving DCOPs. Unfortunately, when problems include multiple cycles in the constraint graph, the algorithm does not converge and the solutions it visits are of low quality. This is in spite of its exploitive structure.

In this paper we proposed new algorithms of the Max-sum family that guarantee convergence. The Max-sum_AD algorithm uses an alternating DAG to avoid cycles. We proved that when the algorithm is performed in a single direction, it converges after a linear number of iterations. After performing a linear number of iterations in each direction the algorithm converges to a high quality solution after taking all of the problem's constraints into consideration. As execution continues in the following phases, the algorithm will converge to solutions that are not necessarily monotonically improving. In fact, our empirical results reveal that after the second direction change, the algorithm may explore complete assignments of the same or even lower quality.

We further solve a weakness of the algorithm that cost calculations, which are propagated and used for selecting value assignments by agents, are often considering different value assignments for the same variable and hence, are inconsistent. In order to overcome this shortcoming of the algorithm we propose the use of value propagation. To validate that we propagate values with high quality we begin value propagation after the algorithm has converged for the second time so that all of the problem's constraints are taken into consideration.

The resulting algorithm Max-sum_ADVP guarantees to monotonically improve after a constant (small) number of direction changes and to converge in pseudo-polynomial time. Our empirical study reveals that convergence is achieved after a small number of direction changes.

After designing an exploitive version of the algorithm that guarantees convergence we made an attempt to improve the performance further by introducing two classes of exploration methods, one inspired by simulated annealing has agents select a value assignment from a subset of high quality assignments and the second balanced between exploration and exploitation by combining exploitive and explorative versions of the algorithm. Thus, after the algorithm converged to a solution using a monotonic, convergence-guaranteeing version, we performed an explorative version for a limited number of iterations and then used the exploitive version to converge to a possibly different solution.

Our empirical study reveals the advantage of Max-sum_ADVP over all guaranteed convergence incomplete inference algorithms. It also outperforms by a large factor standard Max-sum, which did not converge on any problem setup we have experimented with. On the other hand, Max-sum with damping, although it also did not converge in most cases, explored solutions of much higher quality than standard Max-sum, and on some benchmarks, even in similar quality to the quality of solutions that Max-sum_ADVP converged to. Among the exploration methods proposed, a member of the second class that combines exploitive and explorative versions of the algorithm, was found to improve the performance of Max-sum_ADVP further on benchmarks with non-uniform constraint graphs. That been said, on most benchmarks the anytime results of this method were inferior to Max-sum with damping. This indicates that, in contrast to the common assumption, Max-sum has a, yet unrevealed, potential as an explorative algorithm, which can be captured via an anytime mechanism. We intend to investigate this potential in future work.

Compliance with ethical standards

Conflicts of interest The authors declare that they have no conflict of interest.

References

1. Aji, S. M., & McEliece, R. J. (2000). The generalized distributive law. *IEEE Transactions on Information Theory*, *46*(2), 325–343.
2. Arshad, M., & Silaghi, M. C. (2004). Distributed simulated annealing. *Distributed constraint problem solving and reasoning in multi-agent systems, frontiers in artificial intelligence and applications series*, *112* November 2004.
3. Bejar, R., Domshlak, C., Fernandez, C., Gomes, K., Krishnamachari, B., Selman, B., et al. (2005). Sensor networks and distributed CSP: Communication, computation and complexity. *Artificial Intelligence*, *161*(1–2), 117–148.
4. Brito, I., & Meseguer, P. (2010). Improving dpop with function filtering. In *AAMAS* (pp. 141–148).
5. Brito, I., Meisels, A., Meseguer, P., & Zivan, R. (2009). Distributed constraint satisfaction with partially known constraints. *Constraints*, *14*(2), 199–234.
6. Dechter, R. (1999). Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, *113*(1–2), 41–85.
7. Farinelli, A., Rogers, A., Petcu, A., & Jennings, N. R. (2008). Decentralized coordination of low-power embedded devices using the max-sum algorithm. In *AAMAS* (pp. 639–646).
8. Gershman, A., Grubshtein, A., Rokach, L., Meisels, A., & Zivan, R. (2008). Scheduling meetings by agents. In *DCR workshop at AAMAS 2008*, Estoril, Portugal, May.
9. Gershman, A., Meisels, A., & Zivan, R. (2009). Asynchronous forward bounding. *Journal of Artificial Intelligence Research*, *34*, 25–46.
10. Globerson, A., & Jaakkola, T. (2007). Fixing max-product: Convergent message passing algorithms for map lp-relaxations. In *NIPS*.
11. Hatano, D., & Hirayama, K. (2013). Deqed: An efficient divide-and-coordinate algorithm for dcop. In *IJCAI*.
12. Hazan, T., & Shashua, A. (2010). Norm-product belief propagation: Primal-dual message-passing for approximate inference. *IEEE Transactions on Information Theory*, *56*(12), 6294–6316.
13. Heras, F., & Larrosa, J. (2006). Intelligent variable orderings and re-orderings in dac-based solvers for WCSP. *Journal of Heuristics*, *12*(4–5), 287–306.
14. Hirayama, K., & Yokoo, M. (2000). An approach to over-constrained distributed constraint satisfaction problems: Distributed hierarchical constraint satisfaction. In *Proceedings of the third international joint conference on autonomous agents and multiagent systems* (pp. 135–142).
15. Khot, S. (2002). On the power of unique 2-prover 1-round games. In *Proceedings of the thirty-fourth annual ACM symposium on theory of computing* (pp. 767–775).
16. Kiekintveld, C., Yin, Z., Kumar, A., & Tambe, M. (2010). Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In *AAMAS* (pp. 133–140).
17. Kschischang, F. R., Frey, B. J., & Loeliger, H. A. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, *47*(2), 181–208.
18. Larrosa, J., & Schiex, T. (2004). Solving weighted csp by maintaining arc consistency. *Artificial Intelligence*, *159*, 1–26.
19. Lazic, N., Frey, B., & Aarabi, P. (2010). Solving the uncapacitated facility location problem using message passing algorithms. In *International conference on artificial intelligence and statistics* (pp. 429–436).
20. Maheswaran, R. T., Pearce, J. P., & Tambe, M. (2004). Distributed algorithms for dcop: A graphical-game-based approach. In *PDCS)* (pp. 432–439), September 2004.
21. Maheswaran, R. T., Tambe, M., Bowring, E., Pearce, J. P., & Varakantham, P. (2004). Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *3rd International joint conference on autonomous agents and multiagent systems (AAMAS 2004), 19–23 August 2004, New York* (pp. 310–317).
22. Modi, P. J., Shen, W., Tambe, M., & Yokoo, M. (2005). Adopt: asynchronous distributed constraints optimizationwith quality guarantees. *Artificial Intelligence*, *161*(1–2), 149–180.
23. Netzer, A., Grubshtein, A., & Meisels, A. (2012). Concurrent forward bounding for distributed constraint optimization problems. *Artificial Intelligence*, *193*, 186–216.
24. Okimoto, T., Joe, Y., Iwasaki, A., Yokoo, M., & Faltings, B. (2011). Pseudo-tree-based incomplete algorithm for distributed constraint optimization with quality bounds. In J. Lee, (Ed.), *CP 2011, LNCS 6876* (pp. 660–674).
25. Pearce, J. P., & Tambe, M. (2007). Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In *IJCAI* (pp. 1446–1451), Hyderabad, India, January 2007.
26. Petcu, A., & Faltings, B. (2005). A scalable method for multiagent constraint optimization. In *IJCAI* (pp. 266–271).

27. Petcu, A., & Faltings, B. (2005). Approximations in distributed optimization. In P. van Beek (Ed.), *CP 2005, LNCS 3709* (pp. 802–806).
28. Ramchurn, S. D., Farinelli, A., Macarthur, K. S., & Jennings, N. R. (2010). Decentralized coordination in robocup rescue. *The Computer Journal*, *53*(9), 1447–1461.
29. Reeves, C. R. (Ed.). (1993). *Modern heuristic techniques for combinatorial problems*. New York, NY: Wiley.
30. Rogers, A., Farinelli, A., Stranders, R., & Jennings, N. R. (2011). Bounded approximate decentralized coordination via the max-sum algorithm. *Artificial Intelligence*, *175*(2), 730–759.
31. Rollon, E., & Larrosa, J. (2012). Improved bounded max-sum for distributed constraint optimization. In *CP* (pp. 624–632).
32. Smith, M., & Mailler, R. (2010). Getting what you pay for: Is exploration in distributed hill climbing really worth it? In *IAT* (pp. 319–326).
33. Sontag, D., Meltzer, T., Globerson, A., Jaakkola, T., & Weiss, Y. (2008). Tightening lp relaxations for map using message passing. In *UAI* (pp. 503–510).
34. Stranders, R., Farinelli, A., Rogers, A., & Jennings, N. R. (2009). Decentralized coordination of continuously valued control parameters using the max-sum algorithm. In *AAMAS* (pp. 601–608).
35. Teacy, W. T. L., Farinelli, A., Grabham, N. J., Padhy, P., Rogers, A., & Jennings, N. R. (2008). Max-sum decentralized coordination for sensor systems. In *AAMAS* (pp. 1697–1698).
36. Vinyals, M., Pujol, M., Rodríguez-Aguilar, J. A., & Cerquides, J. (2010). Divide-and-coordinate: Dcops by agreement. In *AAMAS* (pp. 149–156).
37. Vinyals, M., Rodríguez-Aguilar, J. A., & Cerquides, J. (2011). Constructing a unifying theory of dynamic programming dcop algorithms via the generalized distributive law. *Autonomous Agents and Multi-Agent Systems*, *22*(3), 439–464.
38. Vinyals, M., Shieh, E., Cerquides, J., Rodriguez-Aguilar, J. A., Yin, Z., Tambe, M., & Bowring, E. (2011). Quality guarantees for region optimal dcop algorithms. In *AAMAS* (pp. 133–140). Tapei.
39. Yanover, C., Meltzer, T., & Weiss, Y. (2006). Linear programming relaxations and belief propagation: An empirical study. *Journal of Machine Learning Research*, *7*, 1887–1907.
40. Yeoh, W., Felner, A., & Koenig, S. (2010). Bnb-adopt: An asynchronous branch-and-bound dcop algorithm. *Artificial Intelligence Research (JAIR)*, *38*, 85–133.
41. Zhang, W., Xing, Z., Wang, G., & Wittenburg, L. (2005). Distributed stochastic search and distributed breakout: Properties, comparishon and applications to constraints optimization problems in sensor networks. *Artificial Intelligence*, *161*(1–2), 55–88.
42. Zivan, R., Okamoto, S., & Peled, H. (2014). Explorative anytime local search for distributed constraint optimization. *Artificial Intelligence*, *212*, 1–26.
43. Zivan, R., & Peled, H. (2012). Max/min-sum distributed constraint optimization through value propagation on an alternating DAG. In *AAMAS* (pp. 265–272).
44. Zivan, R., Yedidsion, H., Okamoto, S., Glinton, R., & Sycara, K. P. (2015). Distributed constraint optimization for teams of mobile sensing agents. *Autonomous Agents and Multi-Agent Systems*, *29*(3), 495–536.