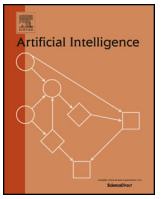
Artificial Intelligence 279 (2020) 103212

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint



Governing convergence of Max-sum on DCOPs through damping and splitting

Liel Cohen, Rotem Galiki, Roie Zivan *

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

A R T I C L E I N F O

Article history:
Received 24 February 2019
Received in revised form 4 October 2019
Accepted 13 November 2019
Available online 2 December 2019

Keywords:
Distributed Constraint Optimization
Incomplete inference algorithms
Max-sum

A B S T R A C T

Max-sum is a version of Belief Propagation, used for solving DCOPs. In tree-structured problems, Max-sum converges to the optimal solution in linear time. Unfortunately, when the constraint graph representing the problem includes multiple cycles (as in many standard DCOP benchmarks), Max-sum does not converge and explores low quality solutions. Recent attempts to address this limitation proposed versions of Max-sum that guarantee convergence, while ignoring some of the problem's constraints. Damping is a method that is often used for increasing the chances that Belief Propagation will converge. That being said, it has not been suggested for inclusion in the algorithms that propose Max-sum for solving DCOPs.

In this paper we advance the research on incomplete-inference DCOP algorithms by: 1) investigating the effect of damping on Max-sum. We prove that, while damping slows down the propagation of information among agents, on tree-structured graphs, Max-sum with damping is guaranteed to converge to the optimal solution in weakly polynomial time; and 2) proposing a novel method for adjusting the level of asymmetry in the factor graph, in order to achieve a balance between exploitation and exploration, when using Max-sum for solving DCOPs. By converting a standard factor graph to an equivalent split constraint factor graph (SCFG), in which each function-node is split into two function-nodes, we can control the level of asymmetry for each constraint.

Our empirical results demonstrate a drastic improvement in the performance of Max-sum when using damping (referred to herein as Damped Max-sum, DMS). However, in contrast to the common assumption that Max-sum performs best when converging, we demonstrate that non converging versions perform efficient exploration, and produce high quality results, when implemented within an anytime framework. On most standard benchmarks, the best results were achieved using versions with a high damping factor, which outperformed existing incomplete DCOP algorithms. In addition, our results imply that by applying DMS to SCFGs with a minor level of asymmetry, we can find high quality solutions within a small number of iterations, even without using an anytime framework. We prove that for a factor graph with a single constraint, if this constraint is split symmetrically, Max-sum applied to the resulting cycle is guaranteed to converge to the optimal solution. We further demonstrate that for an asymmetric split, convergence is not guaranteed.

* Corresponding author.
E-mail addresses: lielc@bgu.ac.il (L. Cohen), rosha@bgu.ac.il (R. Galiki), zivanr@bgu.ac.il (R. Zivan).

https://doi.org/10.1016/j.artint.2019.103212
0004-3702/

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. Many algorithms for solving DCOPs have been proposed. Complete algorithms [1–4] are guaranteed to find the optimal solution, but because DCOPs are NP-hard, solving optimally requires 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 [5–7].

Whether complete or incomplete, DCOP algorithms generally follow one of two broad approaches: distributed search [1, 3,5,6] or inference [2,8–11]. In search algorithms, agents directly traverse the solution space by choosing value assignments and communicating these assignments to each other. 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, for 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 [12], and hence inference-based algorithms are often referred to as GDL-based algorithms.

The Max-sum algorithm [9,13] is an incomplete, GDL-based algorithm that has drawn considerable attention in recent years, including being proposed for multi-agent applications (such as sensor systems) [14,10], task allocation for rescue teams in disaster areas [15], and scheduling the operation of devices in smart homes [16]. Max-sum is actually a version of the well known Belief Propagation algorithm [17], used for solving DCOPs. Agents in Max-sum propagate cost/utility information to all neighbors. This contrasts with other inference algorithms, such as ADPOP [8], 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 the selection of values based on those beliefs.

Belief Propagation in general (and Max-sum specifically) is known to converge to the optimal solution for problems in which the constraint graph is acyclic. Unfortunately, there is no such guarantee for problems with cycles [17,9]. 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. Unfortunately, many DCOPs that have been investigated in previous studies are dense and indeed include multiple cycles (e.g., [1,3]). However, in contrast to most DCOP algorithms, Max-sum was found to produce solutions of similar quality when applied to symmetric and asymmetric problems [18].

Damping is a method that has been used within Belief Propagation in order to decrease the effect of cyclic information propagation. By balancing the weight of the new calculation performed in each iteration with the weight of calculations performed in previous iterations, researchers have reported success in increasing the chances for the convergence of Belief Propagation when applied in different scenarios [19–22]. Nevertheless, damping has not been mentioned in papers that adapt Max-sum for solving DCOPs and propose extended versions of the algorithm [9,23,24]. To the best of our knowledge, besides our preliminary work, there are no published investigations of the effect of damping on the Max-sum algorithm when solving DCOPs.

In this paper we contribute to the development of incomplete inference algorithms for solving DCOPs by

1. Investigating the effect of using damping within the Max-sum algorithm. It is important to emphasize that the contribution and novelty of this part of our work lie not in proposing the use of damping, which is a well known method that has been studied by researchers in the graphical models community (see details in Section 2), but rather in investigating the unique properties of this method when applied to Max-sum in order to improve its performance in solving DCOPs. We demonstrate that, in contrast to the common assumption, the best performance is achieved when Max-sum with damping does not converge, but rather performs efficient exploration, which can be captured when used within an anytime framework [7]. The combination of Max-sum with damping using a high damping factor, and the anytime mechanism, results in an algorithm that outperforms all other versions of Max-sum (except for some of the versions that use splitting, as we will demonstrate) on a variety of benchmarks. It also outperforms local search DCOP algorithms.
2. Proving that on tree-structured graphs, Max-sum with damping converges in weakly polynomial time. This result is extremely significant in distributed scenarios, where agents are not aware of the global topology (only of their own neighborhood); thus, they cannot avoid the use of damping when the graph has a structure that guarantees convergence (this might be the reason that this property was not investigated in studies that consider the use of damping for improving the performance of Belief Propagation when solving centralized problems).
3. Proposing a novel degree of freedom for achieving a balance between exploration and exploitation in Max-sum. This degree of freedom is the level of asymmetry in function-nodes representing constraints in the factor graph. The ability to control the level of asymmetry for each constraint is achieved by converting a standard factor graph to an equivalent split constraint factor graph (SCFG), where each constraint is represented by two function-nodes instead of one.[1]

[1] A similar factor graph was used in [18] for representing asymmetric DCOPs.

The rest of this paper is organized as follows: We present related work in Section 2. The relevant background, including the DCOP formalism and the description of the Max-sum algorithm, is presented in Section 3. Section 4 describes how damping can be used within the Max-sum algorithm, and proves that on tree-structured graphs, Max-sum with damping is guaranteed to converge in weakly polynomial time. Section 5 presents the proposed splitting method and investigates theoretically the relationship between the symmetric split of a constraint and convergence. Section 6 includes an empiric investigation of the effect of the value of the damping factor on the performance of Max-sum when solving standard DCOP benchmarks, as well as an evaluation of the use of split constraint factor graphs (SCFGs). Our conclusions, which state that the best results are achieved by combining both methods (i.e., by using damped Max-sum (DMS) on SCFGs), are presented in Section 7.

2. Related work

The graphical models literature includes many examples of the use of damping to improve the performance of Belief Propagation (BP). While we obviously cannot mention all of them, we specify a number of studies that bear some resemblance to our work and from which one can learn the points of agreement and common assumptions in this research community regarding the effect of damping on BP.

The results of an attempt to apply damping to BP when solving both synthetic and realistic problems, represented by Bayesian networks, were presented in [25]. The damping factor used in these experiments was rather small (0.1). The results indicated that damping reduced oscillations and increased the chances that the algorithm would converge. However, in many cases the algorithm converged to inaccurate solutions (i.e., solutions that did not adequately approximate the optimal solution). An investigation of the effect of the selection of the level of damping on the convergence rate of BP when solving K-SAT problems was presented in [22]. The results indicated that the fastest convergence was achieved for a damping factor of approximately 0.5, while larger damping factors (0.9) were more effective at reducing oscillations. This study also shed light on the relationship between the success of damping at achieving convergence and the density of the problem: a higher damping factor was required in order to achieve convergence when solving dense (as opposed to sparse) problems.

Lazic et al. reported that damping increases the chances of convergence in maximum-a-posteriori (MAP) inference problems [19]. They found that a damping factor of 0.8 was sufficient to achieve convergence in most cases, although their results indicated that a high damping factor may increase the number of iterations required for convergence.

The most similar study to our own regarding damping seems to be that of Som and Chockalingam [26]. For bit error problems in communication channels, they investigated the effect of damping on both the convergence and the quality of the result of BP. The results obtained for these problems were quite different from those presented in the current study when applying damped Max-sum (DMS) to DCOPs. They reported that the method was successful in producing high quality solutions for damping factors between 0.3 and 0.7, and that the best solutions were found when using a damping factor of 0.45.

An investigation of the effect of damping on the convergence of BP when solving clustering data problems was presented in [27]. The results indicated that when the algorithm converged, it consistently produced high quality results (regardless of the damping factor). It was only for very small damping factors (up to 0.3) that the algorithm failed to converge. Again, these authors reported that high damping factors slowed the rate of convergence.

The conclusion from this short survey of studies on damping in the graphical models literature is that the effect of damping on BP is highly dependent on the problem being solved. Thus, there is merit in investigating the effect of damping when solving DCOP benchmarks. Furthermore, none of the papers mentioned above (nor any others of which we are aware) reports the theoretical bounds proven in the current paper. Nor do they investigate the possibility that damping can be used to achieve a balance between exploration and exploitation for the Max-sum algorithm, as is commonly done for local search algorithms.

Besides our own preliminary work, very few studies that use Max-sum for solving DCOPs report the use of damping. One such paper used a damping factor of 0.5 and found the algorithm to be inferior to standard local search algorithms [28]. There was also an attempt to use damping for local search algorithms [29], which is obviously less relevant to our work.

Max-sum was applied to asymmetric DCOPs in [18], by having each agent involved in a constraint hold a function-node representing its personal costs for that constraint. Thus, for each binary constraint, there were two representing function-nodes. The study showed that, in contrast to other DCOP algorithms, Max-sum versions maintain the quality of the solutions that they produce when applied to asymmetric problems. The main difference with respect to our work is that, while the previous study used more than one function-node for a single constraint in order to represent the given natural structure of an asymmetric problem, we use an algorithmic method to initiate a split of a standard function-node into two function-nodes representing the same constraint.

Belief Propagation has been successfully adapted to problems in which variables have continuous domains [30,31]. Recently, the splitting of nodes in factor graphs was shown to be effective for such problems. For the *Consensus* problem, splitting can guarantee the convergence of the algorithm at competitive rates [32]. For combinatorial domains (as in our settings) a continuous concave lower bound function exists and its convergence can be guaranteed and accelerated using a combination of splitting and damping [33]. While these results do not provide guarantees or indications regarding the convergence of Belief Propagation in combinatorial domains, they encourage and inspire our efforts to using splitting as a means of taming the convergence in such domains, so as to achieve a balance between exploration and exploitation.

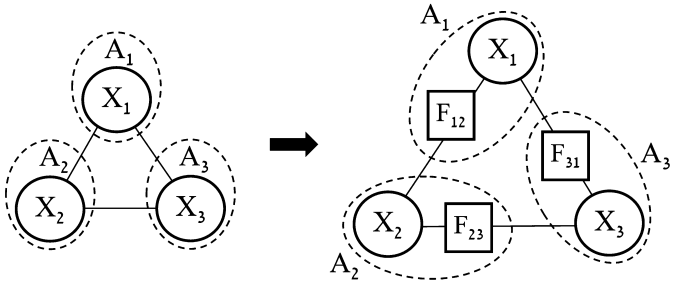


Fig. 1. Transformation of a DCOP to a factor graph.

3. Background

In this section we present background information on DCOPs and on the Max-sum algorithm.

3.1. Distributed constraint optimization

To avoid confusion, and without loss of generality, in the rest of this paper we will assume that all problems are minimization problems (as presented in many DCOP papers, e.g., [1,3,4]). Thus, we assume that all constraints define costs and not utilities. The inference algorithm for minimization problems is actually *Min-sum*. 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., [1–3].

A $DCOP$ is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$. $\mathcal{A}$ is a finite set of agents $\{A_1, A_2, ..., A_n\}$. $\mathcal{X}$ is a finite set of variables $\{X_1, X_2, ..., X_m\}$. Each variable is held by a single agent (an agent may hold more than one variable). $\mathcal{D}$ is a set of domains $\{D_1, D_2, ..., D_m\}$. 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 $R_j \in \mathcal{R}$ defines a non-negative *cost* for every possible value combination of a set of variables, and is of the form $R_j : D_{j_1} \times D_{j_2} \times \ldots \times D_{j_k} \rightarrow \mathbb{R}^+ \cup \{0\}$. A *binary constraint* refers to exactly two variables and is of the form $R_{ij} : D_i \times D_j \rightarrow \mathbb{R}^+ \cup \{0\}$.[2] A *binary DCOP* is a DCOP in which all constraints are binary. A *partial assignment* (PA) is a set of value assignments for a set of variables, in which each variable appears at most once. *vars(PA)* is the set of all variables that appear in PA, $vars(PA) = \{X_i \mid \exists d \in D_i \wedge \langle X_i, d \rangle \in PA\}$. A constraint $R_j \in \mathcal{R}$ of the form $R_j : D_{j_1} \times D_{j_2} \times \ldots \times D_{j_k} \rightarrow \mathbb{R}^+ \cup \{0\}$ is *applicable* to PA if each of the variables $X_{j_1}, X_{j_2}, \ldots, X_{j_k}$ is included in $vars(PA)$. The *cost of a partial assignment* PA is the sum of all constraints applicable to PA over all the value assignments in PA. A *complete assignment* (or a *solution*) is a partial assignment that includes all the DCOP's variables ($vars(PA) = \mathcal{X}$). An *optimal solution* is a complete assignment with minimal cost.

For simplicity, we make the common assumption that each agent holds exactly one variable, i.e., $n = m$. We let n denote hereinafter the number of agents and the number of variables. For the same reason, we also assume that all constraint costs are integers, and we concentrate on binary DCOPs, in which all constraints are binary. These assumptions are customary in the DCOP literature (e.g., [1,2]).

Each DCOP induces a graph $G = (V, E)$, where $V = \mathcal{X}$, and an edge connects the nodes $X_i, X_j \in V$ if there is a constraint $R_{ij} \in \mathcal{R}$ that is defined on $D_i \times D_j$. The corresponding factor-graph is a bipartite graph $G' = (V', E')$, which is defined as follows:

- V' has two types of node: (a) variable-nodes – $X_1, \ldots, X_n$, and (b) function-nodes – for each $e = (X_i, X_j) \in E$ there is a node F_e (or F_{ij}) in V'.
- E' contains an edge that connects X_i with F_e if and only if e is an edge in G that is adjacent to X_i.

3.2. The Max-sum algorithm

The Max-sum algorithm [9] operates on a *factor graph* [34] as defined above. 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 computations 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 of each function-node is performed by one of the agents whose variable is involved in the constraint it represents.

Fig. 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

[2] We say that a variable is *involved* in a constraint if it is one of the variables to which the constraint refers.

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: $\{R^{i-1}_{F \to v} : F \in N_v \setminus \{v'\}\}$ 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: $\{Q^{i-1}_{X \to v} : X \in N_v \setminus \{v'\}\}$ by Eq. (2)
12. $\quad\quad\quad$ send message $R^i_{v \to v'}$ to v'

Fig. 2. Standard Max-sum.

adopts the role of the node representing its own variable, as well as the role of one of the function-nodes representing a constraint in which it is involved (e.g., in this factor graph, agent A_1 adopts the role of function-node F_{12}, which represents the constraint between its own variable X_1 and variable X_2 held by agent A_2).

Fig. 2 presents an outline of the Max-sum algorithm.[3] The pseudo-code for variable-nodes and function-nodes is similar apart from the computation of the content of messages 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-node 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-node neighbors apart from F at iteration $i-1$. Formally, this is given by the function $Q^i_{X \to F} : D_X \to \mathbb{R}$ with

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

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 [9,23]. Note that provided 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

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

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

[3] In contrast to previous papers on Max-sum, we present our algorithm using pseudo-code. This follows standard DCOP literature, e.g., [1,2,6]. Nevertheless, only the presentation is different; the algorithm itself is identical to the algorithms presented in [9,23].

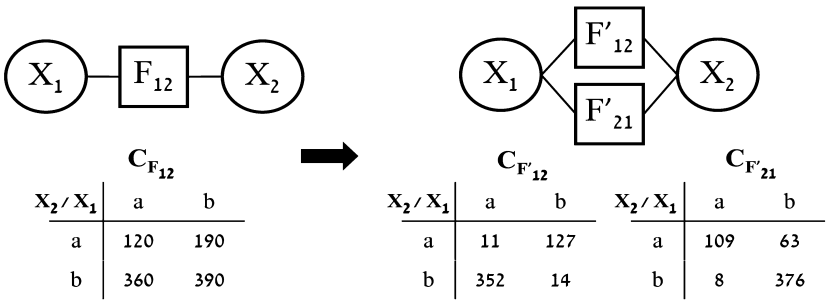


Fig. 3. An acyclic DCOP factor graph (on the left) and its equivalent SCFG (on the right).

Agents compute their beliefs about their variables according to the messages received by the corresponding variable-nodes. Formally, the beliefs regarding variable-node X at iteration i are represented by 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 Equations (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 at subsequent iterations only the most recently received messages are retained, overwriting those received in previous iterations. Most descriptions of Max-sum do not index the messages or beliefs by iteration; we do so only to emphasize that new messages are synchronously generated based on messages generated by neighbors in the previous iteration.

3.2.1. Applying Max-sum to asymmetric problems

When Max-sum is applied to an asymmetric problem, each (binary) constraint in the corresponding factor graph is represented by two function-nodes, one for each part of the constraint held by one of the involved agents. Each function-node is connected to both variable-nodes representing the variables involved in the constraint [18]. Fig. 3 presents two equivalent factor graphs that include two variable-nodes, each with two values in its domain, and a single binary constraint. On the left, the factor graph represents a (symmetric) DCOP including a single constraint between variables X_1 and X_2; hence, it includes a single function-node representing this constraint. On the right, the equivalent factor graph representing an equivalent asymmetric DCOP is depicted. It includes two function-nodes, representing the parts of the constraint held by the two agents involved in the asymmetric constraint. Thus, the cost table in each function-node includes the asymmetric costs that the agent holding this function-node incurs. The factor graphs are equivalent since the sum of the two cost tables pertaining to the function-nodes representing the constraints in the factor graph on the right is equal to the cost table of the single function-node representing this constraint in the factor graph on the left (see [18] for details).

3.2.2. Exploration and exploitation in Max-sum

In local search algorithms, an agent commonly selects an assignment that minimizes the cost according to the information available to it, i.e., it exploits the information. On the other hand, the agent can select an assignment that does not minimize (and may even increase) its cost, in the hope that this will allow it to find assignments with lower cost in subsequent iterations. Such actions, which do not result in immediate benefit, are referred to as exploration. In a distributed local search, even if each agent performs only exploitive actions, concurrent actions by a number of agents can generate exploration, e.g., in distributed stochastic algorithms when neighboring agents replace assignments concurrently.

In inference algorithms, such as Max-sum, agents do not propagate assignments. However, each variable-node can select an assignment at each iteration based on the costs it receives. Thus, as in the distributed stochastic algorithm (DSA) [6], a global view that examines the quality of the global assignment at each iteration can reveal whether the agents are either improving the global assignment or, involuntarily, selecting assignments that have higher costs (in which case, they are exploring). Recent attempts to generate versions of Max-sum that balance exploration and exploitation were presented in [35,13]. However, such an approach requires the use of an anytime mechanism in order to capture the best solution that the algorithm traverses, as in the case of local search algorithms [7]. In contrast, the approach we propose in this paper, which combines damping and splitting, produces high quality results even when an anytime mechanism cannot be used.

Formally we define:

Denote by the *state* of the algorithm at some iteration k the value assignments that the variable-nodes would select according to the messages sent to them at iteration $k-1$. Each state obviously has a global cost, which is the sum of the costs incurred by the violated constraints.

Definition 1. An exploitive action of the algorithm is an action that must result in a lower global cost than the global cost of the current state.

Definition 2. An explorative action of the algorithm is an action that may result in an equal or higher global cost than the global cost of the current state.

Thus, we say that an algorithm that includes exploitive and explorative actions attempts to achieve a balance between exploitation and exploration.

4. Damping

Two common assumptions regarding Belief Propagation are that it is successful when it converges, and that its main drawback is its failure to converge in problems in which the graph used to represent those problems includes multiple cycles. Thus, different methods have been proposed in order to guarantee the convergence of Belief Propagation, e.g., revising the optimization function or changing the graph structure [36,23,24].

Damping is a less radical method that has been proposed for increasing the chances that BP will converge [19–22]. However, in contrast to the methods mentioned above, although introducing damping into Belief Propagation is found empirically to increase the probability of convergence, to the best of our knowledge, there is no theoretical guarantee (or even an estimation or prediction method) that can identify when BP with damping will converge.

4.1. Introducing damping into Max-sum

While Max-sum is a version of Belief Propagation, papers describing Max-sum and proposing specific versions for solving DCOPs do not mention the use of damping [9,10,23,24,37]. This is surprising, considering the efforts made by researchers to trigger convergence of the Max-sum algorithm (e.g., [23,24]). Thus, in the following, we specify how damping can be used within Max-sum, and we compare its performance with that of standard Max-sum, guaranteed-convergence versions of Max-sum, and other incomplete DCOP algorithms.

In order to add damping to Max-sum we introduce a parameter $\lambda \in (0, 1]$. Before sending a message at iteration k, an agent performs calculations, as in standard Max-sum. Denote by $\widehat{m^k_{i\to j}}$ the result of the calculation made by agent A_i of the content of the message intended to be sent from A_i to agent A_j at iteration k. Denote by $m^{k-1}_{i\to j}$ the message sent by A_i to A_j at iteration $k-1$. The message sent from A_i to A_j at iteration k is calculated as follows:

$$m^k_{i\to j} = \lambda m^{k-1}_{i\to j} + (1-\lambda)\widehat{m^k_{i\to j}}. \tag{5}$$

Thus, λ expresses the weight attributed to previously performed calculations with respect to the most recent calculation performed. Moreover, when $\lambda = 0$, the resulting algorithm is standard Max-sum. We demonstrate further in this paper that a selection of a high value for λ (close to 1) increases the chances that the algorithm will converge.

In all our implementations, damping was performed only by variable-nodes. This allowed us to analyze the level of damping with respect to n (the number of variables/agents in the problem).

4.2. Convergence runtime bounds

Standard Max-sum guarantees convergence to the optimal solution in linear time when the constraint graph (and as a result, the corresponding factor-graph as well) is tree-structured, i.e., contains no cycles. We first establish a weakly polynomial lower bound for this guarantee, i.e., that there exists a problem for which damping slows down the convergence by a factor of $log_{1/\lambda}(C)$, where C is the cost of the optimal solution plus 1.[4] Next, we prove a (loose) upper bound on the time for convergence, which is also weakly polynomial.

Let n be the number of variables in a problem, and C the cost of the optimal solution plus 1.

Lemma 1. *There exists a scenario in which Max-sum with damping will converge on a tree-structured graph in no less than* $2(n-2) + log_{1/\lambda}(C)$.

[4] Since we assume that all costs are integers, C is the smallest possible cost of a solution that is larger than the optimal cost.

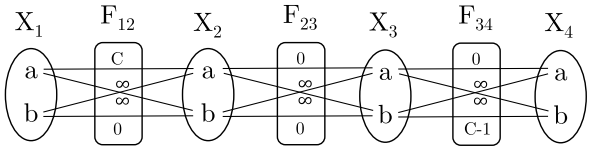


Fig. 4. Lower Bound Example.

Proof. Consider a factor graph with four variable-nodes, X_1, X_2, X_3 and X_4, and three function-nodes, F_{12}, F_{23} and F_{34}, as depicted in Fig. 4. Each variable has two values in its domain, a and b. All functions include infinite costs for any non equal combination of value assignments, i.e., when one variable assigns a and the other b. Function F_{23} entails the cost 0 for both equal combinations of the variables (either a or b). Function F_{12} entails a cost of $C > 1$ if both agents assign a and a cost of zero if they both assign b. Function F_{34} entails a cost of $C - 1$ if both agents assign b and a cost of zero if they both assign a. Obviously, the optimal solution is when all variables assign b. However, in order for variable X_4 to realize that it should assign b, X_2, which receives cost C for its value a from F_{12} at every iteration, must perform $log_{1/\lambda}(C)$ iterations before it sends a message to X_2 with a cost larger than $C - 1$ for the assignment of a. This information must path to X_4 before it can learn that it is better to assign b than a, which requires 4 sequential messages. Obviously, if we add more variables to the chain (each with two values a and b), such that the two functions adjacent to the first and last variable-nodes in the chain are identical (in terms of their costs) to F_{12} and F_{34}, while all other functions are identical in costs to F_{23}, the number of sequential messages will increase by 2 for each additional variable. Thus, a lower bound to this problem is $log_{1/\lambda}(C) + 2(n-2)$. □

Proposition 1. *The guaranteed runtime for convergence for Max-sum with damping is at least weakly polynomial.*

Proof. An immediate corollary from Lemma 1 □

The example presented in Lemma 1, which we use to prove Proposition 1, indicates that a strong polynomial upper bound does not exist for DMS when solving a tree-structured graph. However, it does not specify what the upper bound is. In order to produce an upper bound on the runtime (i.e., number of steps) that DMS will perform before converging to the optimal solution, we note that the convergence of standard Max-sum to the optimal solution on tree-structured graphs is achieved in linear time because each variable-node in the factor graph can be considered as a root of a tree at which all other agents accumulate costs (similar to a DPOP running on a pseudo-tree with no back-edges). Thus, each agent, after at most a linear number of steps, knows, for each of the values in its variable domain, the costs of the best solution in which it is involved (for simplicity and without loss of generality, we will assume no ties).

Let n be the number of variables in a problem represented by a tree-structured factor graph G' and $\widehat{C}$ be the maximal cost sent by an agent in standard Max-sum, when solving the same problem. We use ϵ to represent the largest *non ignorable* difference for a problem, i.e., the largest number such that for each cost c sent in standard Max-sum by some agent when solving the same problem, if the agent would send cost $c' = c - \epsilon$, the receiving agent would perform exactly the same actions as when receiving c in standard Max-sum, i.e., select the same value assignments to calculate function costs (if the receiver is a function-node) or make the same selection of a value assignment (if the receiver is a variable-node).

Lemma 2. *After at most $2(n-2) \cdot log_{1/\lambda}(\widehat{C}/\epsilon)$ steps of the algorithm, a variable-node X_i in G' can select its value assignment in the optimal solution.*

Proof. Allow each of the variable-nodes, from the farthest from X_i in G' to the closest, to perform $log_{1/\lambda}(\widehat{C}/\epsilon)$ steps of the algorithm, taking into consideration only the last message received from its neighbors. Further, allow each function-node receiving a message to perform a single step immediately. Obviously, once these steps are completed, X_i receives costs that allow it to select its assignment in the optimal solution. □

Proposition 2. *Max-sum with Damping is guaranteed to converge to the optimal solution on tree-structured graphs in weakly polynomial time.*

Proof. Immediate from Lemma 2. After $2(n-2) \cdot log_{1/\lambda}(\widehat{C}/\epsilon)$ steps, all variable-nodes can select their assignment in the optimal solution; thus, the convergence rate is polynomial in n and $\widehat{C}/\epsilon$, i.e., weakly polynomial. □

We note that similar proofs can establish that Max-sum with damping produces the optimal solution on graphs with a single cycle in weakly polynomial time (subject to some restrictions [38]), and that using damping in Max-sum_AD and Max-sum_ADVP [24] slows down the convergence in each phase to, at most, weakly polynomial time.

5. Split constraint factor graphs

Damping can be used as a degree of freedom, to balance exploration and exploitation in Max-sum (as described above in Section 4). However, when using damping, thousands of iterations are required for the algorithm to find high quality solutions (see Section 6). By combining damped Max-sum (DMS) with an anytime framework [7], the number of iterations required to find high quality solutions is an order of magnitude smaller.

We aim to shorten the process of producing high quality solutions by Max-sum and to eliminate the dependency on the anytime framework, which requires agents to share value assignments, in contrast to the requirements of the algorithm. Thus, we propose an additional degree of freedom: the level of asymmetry of constraints in the factor graph. To this end, we propose the use of *Split Constraint Factor Graphs* (SCFGs).

5.1. Max-sum SCFG

An SCFG is a factor graph generated as an algorithmic action, in which each constraint that was represented by a single function-node in the original factor graph is represented by two function-nodes. The SCFG is equivalent to the original factor graph if the sum of the cost tables of the two function-nodes representing each constraint in the SCFG is equal to the cost table of the single function-node representing the same constraint in the original factor graph. By adjusting the degree of similarity between the two function-nodes representing the same constraint, we can determine the level of asymmetry in the SCFG.

Formally, an SCFG G' is equivalent to a factor graph G if their sets of variable-nodes (and their domains) are equal, and if, for each function-node F in G with cost table C_F, there exist function-nodes F' and F'' in G' that are connected to the same variable-nodes as F, and that have constraint tables satisfying $C_F = C_{F'} + C_{F''}$. Thus, each SCFG has a single equivalent DCOP factor graph; however, a standard DCOP factor graph can have countless equivalent SCFGs.[5]

Returning to the example portrayed in Fig. 3, given a standard factor graph of a symmetric DCOP, as presented on the left, the cost table of the function-node F_{12} in this factor graph is split using a different random ratio for each entry, thus generating a new equivalent SCFG (on the right) containing 2 function-nodes F'_{12} and F'_{21}.

We differentiate between constant SCFGs, in which constraint costs are split according to a predetermined constant ratio, and random SCFGs, in which each cost in each constraint table is split according to a randomly generated ratio.

It is important to notice the difference between the use we make here of SCFGs and the use we made of factor graphs with two function-nodes representing a constraint in [18]. There, we assumed that this type of factor graph is required in order to represent the asymmetric state of the world, while here we assume that the input problem is symmetric, and the generation of the SCFG is an algorithmic action (represented by the small black arrow between the factor graphs in Fig. 3).

5.2. Splitting a graph with a single constraint

In Section 6 we present empirical evidence of the success of applying DMS to symmetric SCFGs and SCFGs with minor asymmetry. As part of an attempt to explain this success, we investigate the differential effects of a symmetric split and an asymmetric split in a single constraint factor graph.

Lemma 3. *On a factor graph with k variable-nodes $X_1, X_2, ..., X_k$ and s identical function-nodes, $F_1, F_2, ..., F_s$, each of which is connected to all k variable-nodes, Max-sum is guaranteed to converge to the optimal solution after the first iteration.*

Proof. Herein, we carry out a proof by induction on the number of iterations. If the smallest cost c in the cost table held by each of the s function-nodes[6] is for entry $i_1, i_2, ..., i_k$, representing the cost when the value assignments selected are the i_j'th values in the domain of X_j, $1 \leq j \leq k$, then, in the first iteration of the algorithm, each function-node will send to X_j a vector in which the i_j'th cost is c. All other costs in these vectors must be larger than c. Thus, following the first iteration, value i_j is selected by each variable-node X_j. The induction assumption is that at the t'th iteration,[7] $t > 1$, the i_j'th cost in the messages sent to X_j will be

$$c[((k-1)(s-1))^{\frac{t+1}{2}} + ((k-1)(s-1))^{\frac{t-1}{2}} + ... + (k-1)(s-1) + 1] = c\sum_{z=0}^{\frac{t+1}{2}}((k-1)(s-1))^z$$

while all other costs in these vectors will be larger. At the next iteration, $(t+1)$, each variable-node X_j will send to each of its function-node neighbors the sum of messages it received from each of its other function-node neighbors. Thus, in each

[5] We note that, while we focus on binary constraints, the same splitting method can be used for k-ary constraints for $k > 2$.

[6] Recall that we assumed in Section 3.1 that there are no ties; therefore, such a cost is unique.

[7] Without loss of generality, we assume that t is odd. If it were even, then the calculation would include an additional iteration (the first) in which variable-nodes send messages including zeros. Thus, the assumption would be that the cost is $c\sum_{z=0}^{\frac{t}{2}}((k-1)(s-1))^z$.

of these messages, the i_j'th cost will be

$$(s-1)c\sum_{z=0}^{\frac{t+1}{2}}((k-1)(s-1))^z$$

At iteration $t+2$, the i_j'th cost in the vector sent by each function-node F_q to each variable-node X_j will correspond to the sum of the smallest cost in the $k-1$ vectors received by F_q from its neighbors excluding X_j, and the smallest cost in the cost table, c. Thus, the resulting cost, which must still be smallest in the vector, is

$$(k-1)(s-1)c\sum_{z=0}^{\frac{t+1}{2}}((k-1)(s-1))^z + c = c\sum_{z=0}^{\frac{t+3}{2}}((k-1)(s-1))^z$$

All other costs in the vector will correspond to the sum of $\sum_{z=0}^{\frac{t+3}{2}}((k-1)(s-1))^z$ entries in the cost tables that are greater than or equal to c. □

Lemma 4. *Assume a factor graph with k variable-nodes $X_1, X_2, ..., X_k$ and s identical function-nodes, $F_1, F_2, ..., F_s$, each of which is connected to all k variable-nodes. If, for any vector of $s-1$ real numbers $m_2, m_3, ..., m_s$, the cost tables held by the s function-nodes maintain the relation $C_{F_1} = m_2 C_{F_2} = m_3 C_{F_2} = ... = m_s C_{F_s}$, then Max-sum is guaranteed to converge to the optimal solution after the first iteration.*

To avoid redundancy, we omit the full proof for this lemma, which is very similar to the proof of Lemma 3. The main difference is that at the first iteration, each variable-node X_j is sent one message with the i_j'th cost equal to c, and $s-1$ other messages such that, for each message from F_q, $2 \leq q \leq s$, the i_j'th cost is equal to $m_q c$. Nevertheless, as in the case of Lemma 3, at each iteration $t > 1$, a message sent to X_j will include a cost for the i_j'th value that is equal to some number $E > 1$ multiplied by c. Each of the other costs in the message will correspond to this same number E multiplied by a number $c' \geq c$.

Lemma 5. *Assume a factor graph with k variable-nodes $X_1, X_2, ..., X_k$ and s identical function-nodes, $F_1, F_2, ..., F_s$, each of which is connected to all k variable-nodes. If, for any vector of $s-1$ real numbers $m_2, m_3, ..., m_s$, the cost tables held by the s function-nodes maintain the relation $C_{F_1} = m_2 C_{F_2} = m_3 C_{F_2} = ... = m_s C_{F_s}$, then DMS is guaranteed to converge to the optimal solution after the first iteration, regardless of the damping factor being used.*

Again, we omit the full proof and only mention the similarities to the proofs for Lemmas 3 and 4. At each iteration, the costs calculated are multiplied by $1-\lambda$ and added to the previous message sent, multiplied by λ. Thus, in the first iteration, the costs mentioned in the proofs for the above lemmas will be multiplied by $1-\lambda$, but the smallest entries for each variable-node X_j will remain i_j. The same will be true for iteration $t+2$ in the induction. The smallest entry will not change as a result of multiplying all messages by the same factor.

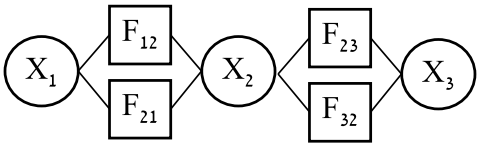
Proposition 3. *If DMS is applied to a constant SCFG generated from a factor graph consisting of k variable-nodes and a single function-node representing the constraint among them, it will converge to the optimal solution after the first iteration, regardless of the damping factor used.*

Proof. Immediate from Lemmas 3, 4 and 5.

When Max-sum is applied to an SCFG generated by a random split of a single constraint, function-nodes might choose in their calculations minimal costs, where these costs do not correspond with identical value assignments, and further calculations may be based on inconsistent value assignment choices for the same variable, producing impossible belief costs for the variable-nodes. Hence, Max-sum does not necessarily converge. We empirically observed this behavior in experiments on 20,000 randomly generated, single cycle factor graphs, in which Max-sum did not converge in many problem instances.

Nevertheless, as demonstrated in Section 6, in SCFGs based on DCOPs containing multiple cycles, this pathology can induce exploration, which can be adjusted by selecting the level of asymmetry of split constraints, and can be exploited by using a high damping factor.[8]

[8] Further insights into the relationship between the success of our empirical results and the properties presented in this section are detailed in Section 6.4.



$C_{F_{12/21}}$

X_2 / X_1	a	b
a	50	20
b	60	300

$C_{F_{23/32}}$

X_3 / X_2	a	b
a	45	300
b	100	6

Fig. 5. Split of two adjacent constraints.

5.3. Splitting adjacent constraints

When reviewing the results presented in Section 5.2, one may wonder whether these results can be generalized to larger graphs. It turns out that even in the simple case of a graph with three variable-nodes and two constraints, the guaranteed convergence property does not apply.

Fig. 5 presents an example of two constraints that are constant and symmetrically split. On this factor graph Max-sum does not converge, but rather oscillates between suboptimal solutions. However, DMS with $\lambda = 0.5$ converges to the optimal solution after 17 iterations.

6. Experimental evaluation

Our experimental study is divided into two parts. In the first, we evaluate the advantage of introducing damping into the Max-sum algorithm. In the second we evaluate the effect of using Max-sum after generating split constraint factor graphs.

We evaluated the algorithms on random uniform minimization DCOPs and on structured and realistic problems, i.e., graph coloring, meeting scheduling and scale-free nets (see details below). For each experiment we randomly generated 50 different problem instances. The results presented in the graphs are an average of those 50 runs. For each iteration we present the sum of the costs of the constraints involved in the assignment that would have been selected by each algorithm at that iteration. In order to maximize the benefit of the algorithms' exploration property, we implemented all algorithms within the anytime framework proposed in [7]. While in a centralized system, it is trivial to preserve the best result achieved by the algorithm, in a distributed system, agents are not aware of the global state; thus, there is a need for a framework that aggregates agents' costs at each iteration and propagates the finding regarding the new best state (for more details see [7]). This approach allowed us to report, for each of the algorithms, the best result traversed by the algorithm across all iterations [7]. Also, in all versions of Max-sum, we used value preferences selected randomly for the purpose of tie breaking, as was suggested in [39].

As mentioned above, the experiments were performed on four types of distributed constraint optimization problem, commonly used for evaluating DCOP algorithms. 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. Naturally, we focused on problems containing a large number of agents (50 or more), as we sought to improve the solutions produced by the Max-sum algorithm for those problems that cannot be solved by complete algorithms within reasonable time.

The uniform random problems were generated by adding, for each problem, a constraint for each pair of agents/variables with probability p_1. For each constrained pair, we set a cost for each combination of value assignments, selected uniformly between 0 and 10. Each problem included either 50 or 100 variables, with 10 values in each domain. Both the constraint graph and the constraint functions were unstructured.

The graph coloring problems consisted of random constraint graph topologies in which all constraints $R_{ij} \in \mathcal{R}$ were "non-equal" cost functions, where an equal assignment of neighbors in the graph incurred a cost of 10 and non-equal value assignments incurred a cost of 0. Such random graph coloring problems are commonly used in DCOP formulations of resource allocation problems. Following the literature, we used $p_1 = 0.05$ and three values (i.e., colors) in each domain to generate these problems, which included 50 agents [6,9,7]. While the constraint graph was unstructured, the constraint functions were highly structured.

Scale-free network problems were generated using the Barabási–Albert (BA) model. An initial set of 7 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 a discrete uniform distribution between 0 and 99. Each variable had 10 values in its domain and the total number of agents in each problem instance was $n = 50$. Similar problems have been previously used to evaluate DCOP algorithms by Kiekintveld et al. [40]. The constraint graph was somewhat structured but the constraint functions were unstructured.

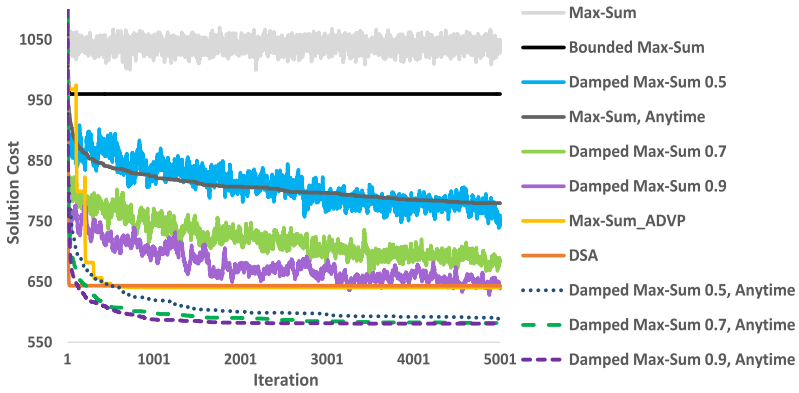


Fig. 6. Costs of solutions for random uniform problems containing 50 agents with relatively low density ($p_1 = 0.1$). (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

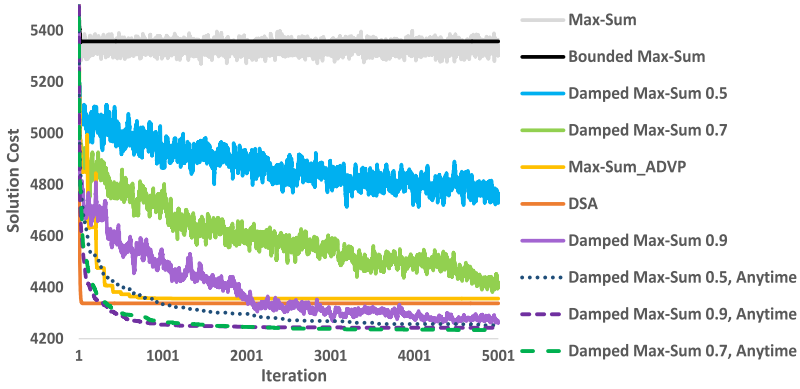


Fig. 7. Costs of solutions for random uniform problems containing 50 agents with relatively high density ($p_1 = 0.7$).

The meeting scheduling problems included 90 agents that 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 was less than the travel time between those meetings, any participants in both meetings were overbooked, and a cost equal to the number of overbooked agents was incurred. These realistic problems are identical to those used by Zivan et al. [7]. Both the constraint graph and the constraint functions were highly structured.

6.1. Evaluation of the effect of damping

In order to investigate the advantages of using damping in Max-sum, we present a set of experiments comparing versions of the algorithm that use different λ values. We also compare these different versions with standard Max-sum and with two versions of Max-sum that guarantee convergence: Bounded_Max-sum, in which the factor graph is converted into a tree structure by eliminating edges from the graph; and Max-sum_ADVP, which converts the graph to an alternating directed acyclic graph and also uses value propagation (for detailed descriptions of these algorithms, see [23,24]). We also include in our experiments the results of the well known distributed stochastic algorithm, DSA (we use type C with $p = 0.7$ [6]), in order to provide insight into the quality of the results in comparison with local search DCOP algorithms.

Our experiments investigated various λ values; however, to avoid redundancy, we only present results with $\lambda \in 0.5, 0.7, 0.9$. This selection allows us to avoid overcrowding the graphs while still enabling the trend of the improvement of the algorithm as λ approaches one to be appreciated. Each of the algorithms in each of the experiments in this section performed 5,000 iterations.

Figs. 6 and 7 present the solution costs found by all algorithms when solving uniform random problems containing 50 agents, with a relatively low density ($p_1 = 0.1$) and a higher density ($p_1 = 0.7$), respectively. The graphs show that, on average, standard Max-sum and Bounded Max-sum produce results with relatively high costs. Also, the solutions achieved by the Damped Max-sum (DMS) algorithms of all parameter sizes are of considerably better quality than the solutions found by standard Max-sum. The DSA and Max-sum_ADVP algorithms managed to achieve better solutions than those obtained by DMS with λ values of 0.5 and 0.7. However, although DMS with a λ value of 0.9 does not seem to converge within 5,000 iterations, for problems with density $p_1 = 0.1$, it produces solutions similar in quality to those of DSA and Max-sum_ADVP,

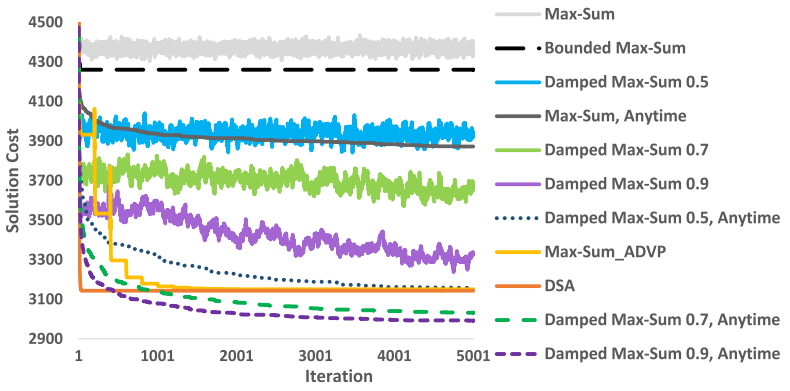


Fig. 8. Costs of solutions for random uniform problems containing 100 agents with relatively low density ($p_1 = 0.1$).

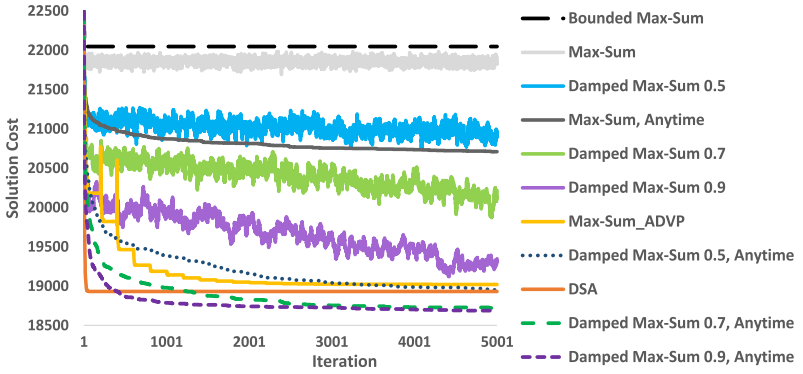


Fig. 9. Costs of solutions for random uniform problems containing 100 agents with relatively high density ($p_1 = 0.7$).

and it outperforms these algorithms on dense problems (with $p_1 = 0.7$). As for the anytime solutions, all DMS versions produce significantly better anytime solutions than the solutions produced by all other algorithms.

Similar results were obtained for random uniform problems consisting of 100 agents. The solutions for the low and high density problems ($p_1 = 0.1$ and $p_1 = 0.7$) are presented in Figs. 8 and 9 respectively. For these larger problems, the results per iteration show that DMS is inferior to both DSA and the guaranteed convergence version Max-sum_ADVP. That being said, the anytime results of DMS using high λ values (0.7 and 0.9) significantly outperform DSA and Max-sum_ADVP. This suggests that damping triggers efficient exploration by the Max-sum algorithm, i.e., that in contrast to the assumption made in the Belief Propagation literature, the best results of Max-sum are achieved not when it converges, but rather (like in the case of local search) when there is a balance between exploration and exploitation. The success of damping in dense problems that include smaller cycles is surprising, and contradicts previous results for Belief Propagation in which success was mostly reported in problems with larger cycles.

Figs. 10, 11 and 12 present results for scale-free nets, meeting scheduling and graph coloring problems, respectively. For scale-free nets, the trends are similar to those obtained for uniform random problems. As more iterations are performed, DMS explores solutions of higher quality. Towards the end of the run, the results per iteration for the version with $\lambda = 0.9$ are superior to the solutions produced by DSA in some iterations, and they are similar to the results of Max-sum_ADVP. The anytime results outperform the converging algorithm significantly. For meeting scheduling and graph coloring problems, the results of the DMS versions do not exhibit such an improvement, and seem to explore solutions of similar quality throughout the run. Interestingly, for graph coloring problems, the $\lambda = 0.9$ version seems to perform limited exploration and traverses solutions of similar quality throughout its runtime, while the 0.5 and 0.7 versions perform a higher level of exploration.

Fig. 13 presents a closer look at the graph coloring results, thus allowing analysis of this phenomenon in greater detail.[9] It seems that the $\lambda = 0.9$ version, after a small number of iterations, starts to perform limited oscillations that follow a strict repeating pattern. In contrast, the $\lambda = 0.5$ and $\lambda = 0.7$ versions perform rapid oscillations, which do not follow a specific pattern. Throughout the run, the average results per iteration of the $\lambda = 0.9$ version outperform those of both the $\lambda = 0.5$ and $\lambda = 0.7$ versions. On the other hand, the corresponding anytime results of the $\lambda = 0.9$ version converge quickly to a

[9] We omitted the results of standard Max-sum without damping, which performs very poorly on these problems (as demonstrated in [24]).

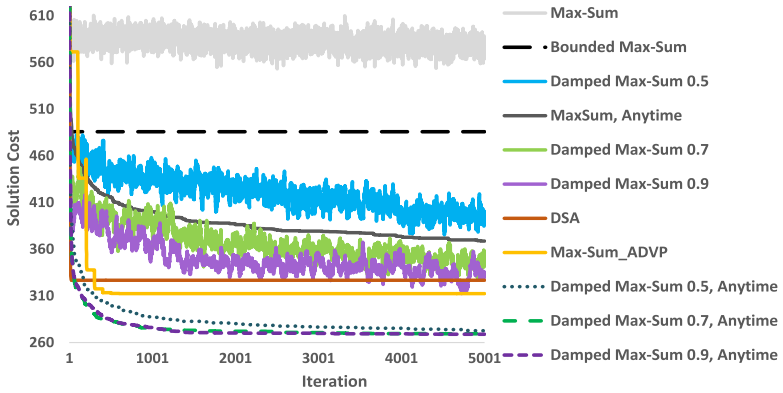


Fig. 10. Costs of solutions for scale-free net problems.

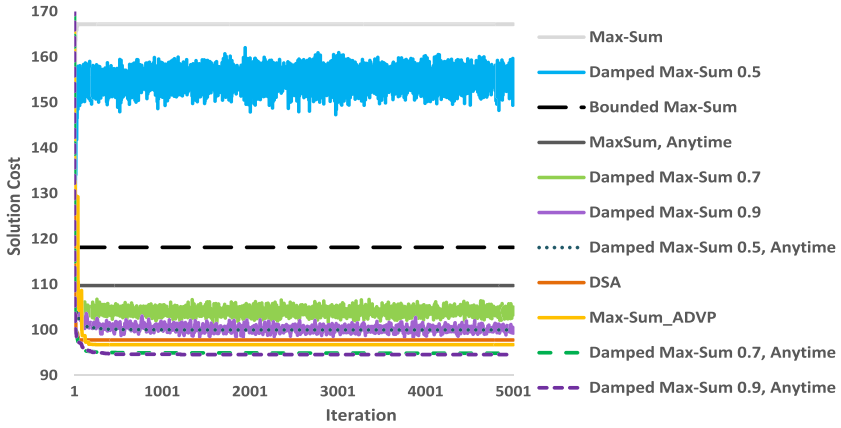


Fig. 11. Costs of solutions for meeting scheduling problems.

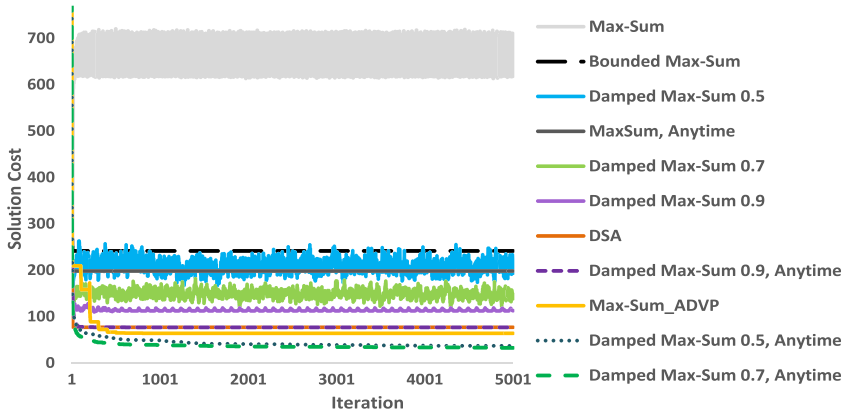


Fig. 12. Costs of solutions for graph coloring problems.

solution of higher cost than the anytime solutions reported for the $\lambda = 0.5$ and $\lambda = 0.7$ versions. This sheds further light on the nature of the relationship between the level of exploration performed by Max-sum with damping and the quality of the results captured and reported by the anytime mechanism.

The results of our experiments indicate that, in contrast to the common assumption regarding the role of damping in improving Belief Propagation (i.e., that it should increase the convergence rate), the success of damping lies in generating useful exploration of high quality solutions that can be captured by an anytime framework and that can outperform versions of Max-sum that guarantee convergence, such as Max-sum_ADVP. In order to strengthen this statement, we present the convergence success rate and anytime performance of the Max-sum versions for the uniform random problem settings and for the graph coloring problems (the convergence results for the meeting scheduling problems and the scale-free nets

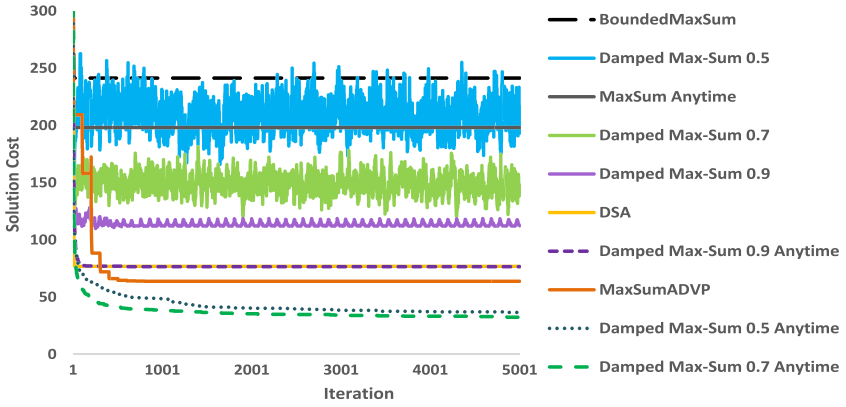


Fig. 13. Costs of solutions for graph coloring problems (a closer look).

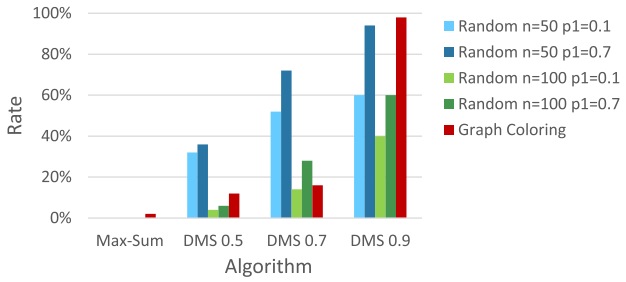


Fig. 14. Convergence success rate for Max-sum and Damped Max-sum with various λ values.

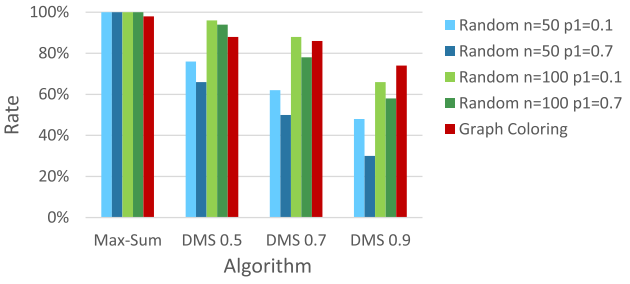


Fig. 15. The percentage of problems for which the anytime solution was better than the final solution.

showed similar trends to the uniform settings and have been omitted to avoid redundancy). Fig. 14 presents, for standard Max-sum and for DMS, the number of problems (out of the 50 problems solved) for which each of the versions of the algorithm converged. In addition, Fig. 15 presents the number of problems in which the anytime solution was better than the solution produced in the final iteration of the algorithm's run.

For the random problems, the results indicate that the closer the damping parameter λ is to one, the higher the chances of convergence of the DMS algorithm. Interestingly, the convergence success rate was higher for problems with higher density (higher density results in more cycles and smaller cycles; hence, this result is counter intuitive to the standard claims regarding Belief Propagation [41]). As for the anytime solutions, in problems for which the algorithm converged, it did not always converge to the best solution visited during the algorithm's run (Fig. 15). The number of problems for which the algorithm converged to the best solution traversed during the algorithm's run increased when a higher value of λ was selected. Nevertheless, for all versions, the anytime results were superior to the results in the final iteration of the algorithm for a significant proportion of the problem instances. It is apparent that when the problems are larger, the convergence rate is smaller, and the number of problems for which the anytime solution is better than the solution in the final iteration of the algorithm is higher.

The results in Figs. 14 and 15 further strengthen the analysis presented in Figs. 12 and 13. For graph coloring problems, the $\lambda = 0.9$ version has a much higher convergence success rate than the $\lambda = 0.5$ and $\lambda = 0.7$ versions. However, for the $\lambda = 0.9$ version, the anytime results are better than the results of the last iteration in a smaller number of runs of the algorithm. Thus, for these problems, a lower damping factor resulted in more effective exploration.

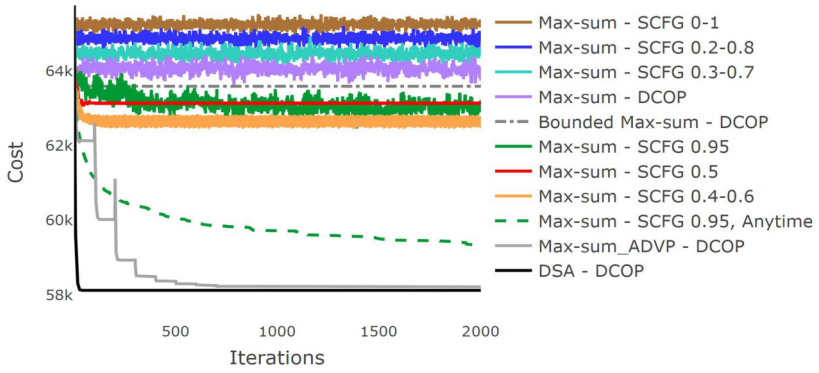


Fig. 16. Solution costs for Max-sum when solving SCFGs of random uniform problems ($p_1 = 0.2$).

In order to evaluate the statistical significance of the results, we performed paired t-tests on the results produced by the algorithms at the final iteration of each of the runs and on the anytime (best) result for each of the algorithms in each run. We established that, for all values of the parameter λ, the DMS anytime solutions were better, on average, than the anytime solutions reported for standard Max-sum, with statistical significance of $p = 0.01$. We also found that the DMS anytime solutions with λ values of 0.7 and 0.9 were better, on average, than the DSA solutions (except in the case of the 0.9 version of DMS in graph coloring problems), with the same significance level of $p = 0.01$.

6.2. The effect of SCFGs

In order to investigate the advantages of using SCFGs when applying Max-sum to DCOPs, we present a set of experiments comparing standard Max-sum and Damped Max-sum (DMS), when both are applied to different types of SCFG. For comparison, we also include the results of Max-sum, Bounded Max-sum, Max-sum_ADVP and DSA when applied to the equivalent DCOPs. We evaluated the algorithms on uniform random DCOPs, for structured and realistic problems (i.e., graph coloring, meeting scheduling and scale-free) with similar settings as for the problems described at the beginning of this section. In each experiment, we randomly generated 50 different problem instances and ran the algorithms for 2,000 iterations in each of them. In uniform random problems (with density $p_1 = 0.2$ and $p_1 = 0.6$), and in scale-free networks, we set a cost for each combination of value assignments, selected uniformly between 100 and 200. This range was selected so that the costs do not become too small (in which case, due to limited precision, distorted SCFGs might be generated). Obviously, if the input costs are between 0 and 100, adding 100 to each cost can be the first step of the splitting method. For similar reasons, in the graph coloring problems, an equal assignment of neighbors in the graph incurred a cost of 50.

In order to avoid redundancy we focus on the most significant results. We present results when applying Max-sum and DMS to two constant SCFGs, one in which the costs are split evenly among the two function-nodes (0.5 version) and one in which the split results in 95% of the cost at one function-node and 5% at the other (0.95 version). For random SCFGs we specify the range of costs from which the cost for the first function-node was selected; thus, the version referred to as 0.4–0.6 includes SCFGs where, for each entry containing cost c in the original constraint cost table, the cost for the corresponding entry of the first function-node was selected randomly between $0.4c$ and $0.6c$. We present random SCFG versions of ranges 0–1, 0.2–0.8, 0.3–0.7 and 0.4–0.6, in order to establish the effect of reducing the range size. Following the results in Section 6.1 we present results of Damped Max-sum with $\lambda = 0.9$. Our experiments with other λ values (0.5 and 0.7) validated that this version indeed performs best on most benchmarks.

Fig. 16 presents the costs per iteration, and the costs of the anytime solution per iteration, when applying Max-sum, DSA, Bounded Max-sum and Max-sum_ADVP to SCFGs. For constant SCFGs and random SCFGs, where the costs were selected from a small range, the solutions were found to have lower cost than standard Max-sum. However, the results were of much lower quality (higher cost) than DSA. To simplify the graph, we only depict the best anytime results among all versions of the algorithm, which were obtained when applying Max-sum to constant SCFGs 0.95. These results are significantly better than the costs per iteration when applying Max-sum to any of the SCFGs, but are considerably worse than the results produced by DSA and Max-sum_ADVP. The results produced by Max-sum for random SCFGs 0–1 are consistent with the results in [18], where only random splits were considered and damping was not used.

Fig. 17 presents results for DMS applied to the same SCFGs. While the trends are the same, and the best results are achieved using constant SCFGs and random SCFGs with a small range for selecting the first cost, here the best results significantly outperform those of DSA. When using SCFG 0.5 and SCFG 0.4–0.6, the algorithm finds high quality solutions in a small number of iterations. When using SCFG 0.95, the algorithm performs more exploration, and after approximately 500 iterations, it finds, on average, solutions of higher quality than when the 0.5 and the 0.4–0.6 SCFGs are used. In order to emphasize the fast convergence of DMS on SCFGs, we added the results of two algorithms that aim to find a balance between the exploration achieved in Max-sum_AD and the exploitation achieved by Max-sum_ADVP [13]. These algorithms, Max-sum_ADSSVP with $t = 2$ and Max-sum_HBVP, perform in phases, similar to Max-sum_AD and Max-sum_ADVP, and

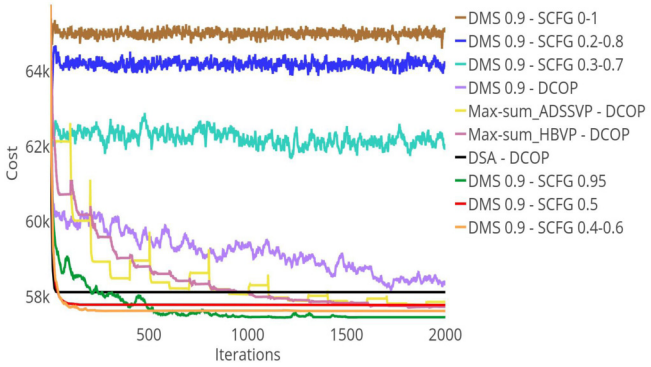


Fig. 17. Solution costs for DMS when solving SCFGs of random uniform problems ($p_1 = 0.2$).

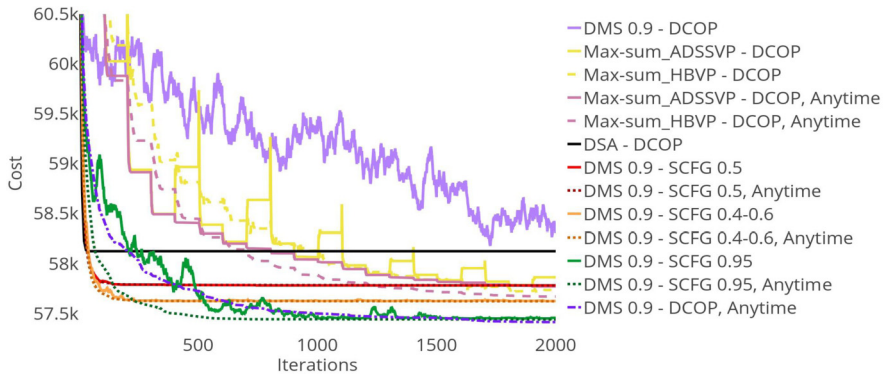


Fig. 18. Solution and anytime costs for DMS when solving SCFGs of random uniform problems ($p_1 = 0.2$).

apparently, they eventually reach higher quality results. However, it is clear that DMS on SCFG 0.5 and SCFG 0.4–0.6 reaches high quality solutions much faster.

Notice that all the results presented in Fig. 17 are the cost per iteration and not the anytime costs. Fig. 18 provides a closer look at the differences between the most successful versions of the algorithm and their anytime results. The anytime results of DMS on standard factor graphs are similar to the results per iteration of DMS on the SCFG 0.95 version. However, the anytime results of the 0.95 version converge must faster. Both of these anytime results are superior to the anytime results of the Max-sum versions presented by [13].[10] When applied to SCFG 0.5, the algorithm converges very fast, yet the costs of the solutions are higher. When applied to random SCFGs 0.4–0.6, solutions with lower costs are also reached very fast. The small level of additional exploration provides an advantage in this case.

The general trends were similar for all other problem types; therefore, only selective graphs that give a closer view of the results of the most successful versions of DMS for these problems are presented. For scale-free nets (Fig. 19), both the costs per iteration and the anytime costs of solutions found by DMS when applied to constant SCFGs and random SCFGs 0.4–0.6 converge much faster than the anytime result of DMS solving standard factor graphs. In addition, the costs per iteration and the anytime costs when applying DMS to random SCFGs 0.3–0.7 (for scale-free nets) were lower than for constant SCFGs 0.5 and random SCFGs 0.4–0.6; thus, we depict these solutions as well. For constant SCFGs 0.95, DMS found high quality solutions very fast.

For graph coloring problems (Fig. 20), DMS on standard factor graphs produces solutions with relatively high costs, and its anytime results are similar to the results of DSA. On the other hand, the solutions found by DMS when applied to constant SCFGs 0.5 and to random SCFGs 0.4–0.6 exhibit significantly lower costs per iteration and anytime costs. The 0.95 version performs a greater level of exploration and the resulting anytime costs are lowest. The results for the meeting scheduling problems (Fig. 21) show a similar trend. When DMS is applied to constant SCFGs 0.95, it performs a greater level of exploration than when applied to SCFGs 0.5 and 0.4–0.6. However, for this type of problem, the anytime results of the random 0.4–0.6 outperform the anytime results of the constant SCFG 0.95 version.

Fig. 22 presents the time for convergence for each of the 50 runs of each of the algorithms and the percentage of problems for which the algorithm converged (among the 50 runs conducted for each problem type). The random SCFGs

[10] In order to validate that this finding is consistent across all benchmarks, we added the best anytime result for these algorithms to the following graphs as well.

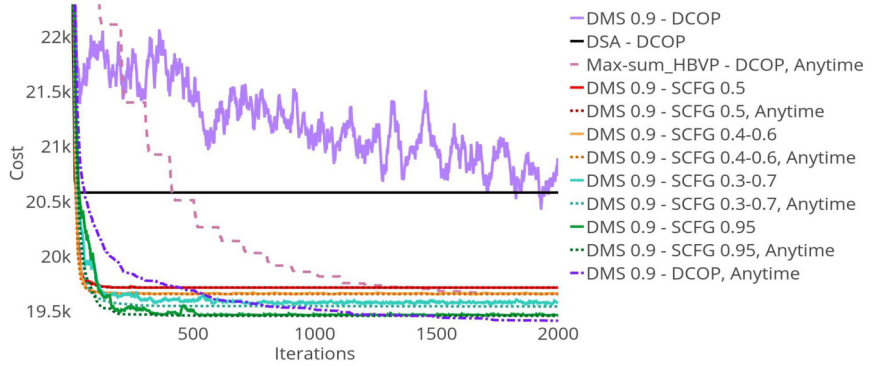


Fig. 19. Costs per iteration and anytime costs for DMS when solving SCFGs of scale-free nets.

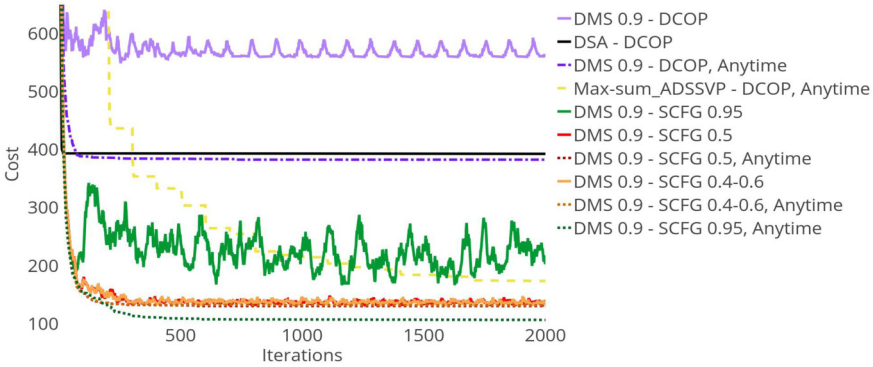


Fig. 20. Costs per iteration and anytime costs for DMS when solving SCFGs of graph coloring problems.

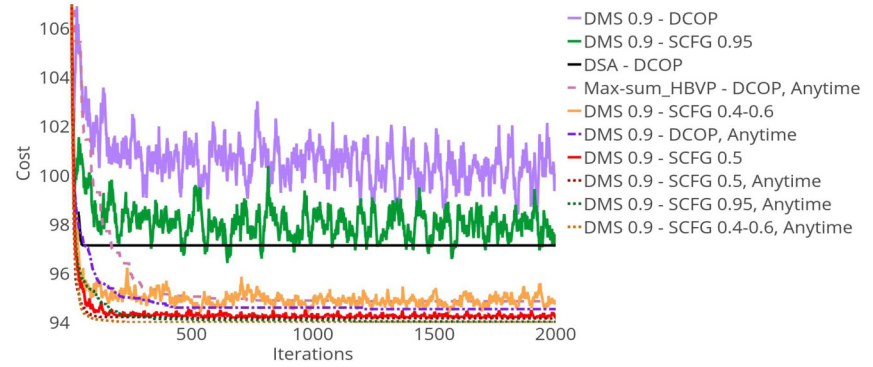


Fig. 21. Costs per iteration and anytime costs for DMS when solving SCFGs of meeting scheduling problems.

with ranges larger than 0.3–0.7 do not appear because they never converge. It is clear that constant SCFGs and the 0.4–0.6 random version have higher convergence success rates than when DMS is applied to standard factor graphs. Moreover, it is apparent that the constant 0.5 version converges very fast. Both the constant SCFG 0.95 and the random SCFG 0.4–0.6 versions converge more slowly and with slightly lower success rates than the SCFG 0.5 version. Thus, they have more opportunities to perform exploration, which can explain the advantage they have in terms of solution quality, as can be seen in Fig. 18.

Fig. 23 presents the time for convergence and the convergence success rate for graph coloring problems. Here, it is clear that fast convergence prevents the algorithm from finding solutions with low costs, when applied to standard factor graphs.

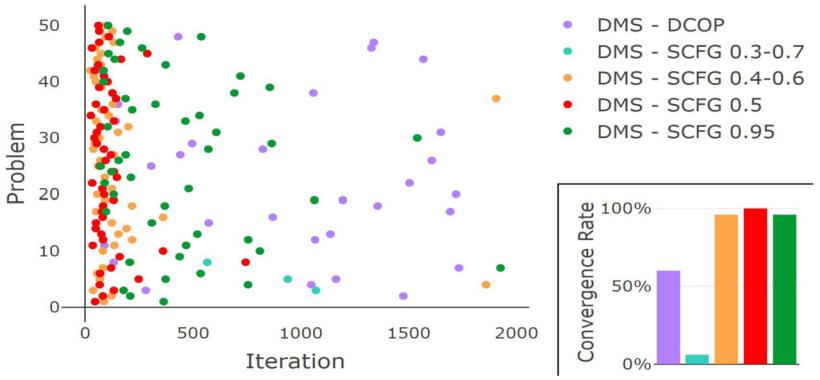


Fig. 22. Number of iterations for convergence and convergence success rate for relatively sparse random uniform problems ($p_1 = 0.2$).

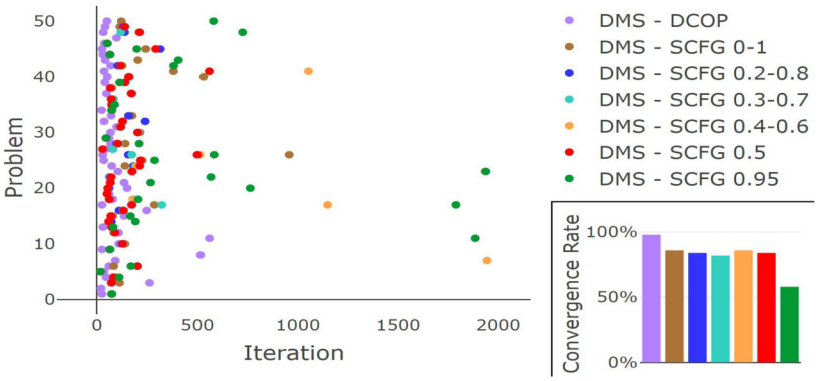


Fig. 23. Number of iterations for convergence and convergence success rate for graph coloring problems.

On the other hand, when applied to SCFGs, DMS performs balanced exploration, which results in solutions with low costs. The constant SCFG 0.95 version triggers a greater level of exploration, which results in lower anytime costs.[11]

6.3. Runtime overhead

If we consider each node in the factor graph as a separate agent, the overhead in runtime caused by splitting function-nodes is only for the variable-nodes, since they need to produce and send double the amount of messages. On the other hand, splitting function-nodes does not create a delay in the computation of function-nodes, since the latter is performed concurrently. However, it is commonly assumed that the roles of the nodes in the factor graphs are performed by the original ("real") DCOP agents. Thus, when adding function-nodes to the graph, we increment the runtime of each iteration. We note that a factor of 2 can be easily achieved by having each agent that performed the role of a function-node in the original factor graph perform the role of the two function-nodes resulting from its split. Our results indicate that for symmetric SCFGs and SCFGs with limited randomness, the convergence of DMS is orders of magnitude faster then when using standard factor graphs, and that this is still true when each iteration takes double the amount of time.

6.4. Discussion

Our results demonstrate success in combining damping and asymmetry for balancing exploration and exploitation. Previous Belief Propagation studies that evaluated the effect of damping focused on its success in triggering convergence. Thus, they ignored the process that leads to convergence, assuming that the best result will be achieved after convergence. Our experience with balancing exploration and exploitation in distributed local search algorithms, coupled with the development of the anytime mechanism [7], motivated our investigation into the behavior of the algorithm before convergence. Understanding that the essence of damping lies in its ability to refine an existing phenomenon encouraged us to investigate the effect of damping on the rapid oscillations generated by standard Max-sum (i.e., when damping is not used). As expected,

[11] In order to avoid redundancy, we do not present convergence graphs for the other problems. As expected, the meeting-scheduling convergence results were similar to the results for graph coloring, while the results for the other problem types were similar to the convergence results of the sparse uniform random problems.

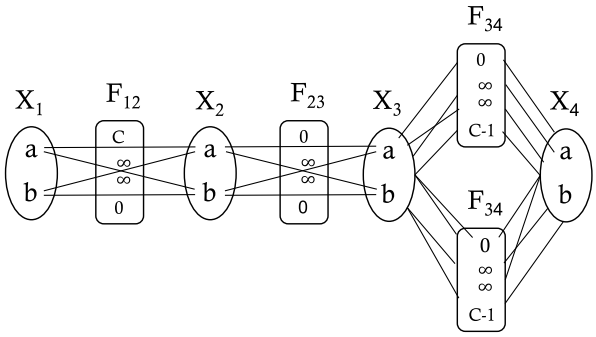


Fig. 24. Split effect example.

damping resulted in more refined oscillations, which offered a more balanced exploration. By coupling with an anytime mechanism, the effect was maximized and high quality results were produced.

Nevertheless, in cases where the anytime mechanism cannot be used, damping alone is not enough, since it does not converge to high quality solutions within two thousand iterations. However, when using constant SCFGs that are split evenly (0.5), convergence is achieved within a few tens of iterations. The use of an uneven split of a constant SCFG (0.95 version), or of random SCFGs with small ranges (0.4–0.6), allows limited exploration. The solutions produced by these versions have lower costs per iteration as well as lower anytime costs than the symmetric constant split version.

In order to explain this success one needs to look back at the results presented in Section 5.2. For symmetric splits, both Max-sum and DMS converge after a single iteration. Thus, in the general case, the difference between a single function-node representing a constraint and its symmetric split into two function-nodes lies in the ratio between the costs sent from the variable-nodes to the function-nodes and the costs in the function-nodes' tables. Going back to the example in Fig. 4, we proved that the convergence time is at least $2(n-2)+log_{1/\lambda}(C)$. For simplicity, assume that only variable node X_2 performs damping, and thus, the convergence time is exactly $2(n-2)+log_{1/\lambda}(C) = 2(n-2)+log_{\lambda}(1-\frac{1}{C})$. When the function-node F_{34} is split, as depicted in Fig. 24, the convergence time is reduced to $2(n-2)+log_{\lambda}(\frac{1-\frac{1}{C}}{2})$. For example, if $C = 100$ and $\lambda = \frac{1}{2}$, then the lower bound is trimmed from 7 to 1. If $\lambda = 0.9$, then it is trimmed from 44 to 7.

Consider a variable-node v and its neighboring function-node f in factor graph G, which is symmetrically split into f' and f'' in factor graph G' (for simplicity, assume that other function-nodes are not split). Let vc be the vector that holds the sum of the costs that v has received from its function-node neighbors, excluding f, in iteration i. In G, at iteration $i+1$, vc will be sent to f. In G', vc will be sent to both function-nodes f' and f''. However, each cost in the table of f is reduced by a factor of one-half in the tables of f' and f''. Thus, f' and f'' will make different calculations from those made by f, giving greater consideration to the differences between costs in vc.

This phenomenon allows DMS to converge faster. Damping allows Max-sum to explore high quality solutions because it reduces the effect of multiple counting of information, as observed by Pearl [42,43]. On the other hand, it slows the aggregation of costs in the propagated cost vectors, and thus the differences between costs in the vectors are less pronounced at the beginning of the run. In contrast, the function-node cost tables are fixed throughout the run. Thus, damping delays assignment replacements, which are required for convergence. Symmetric SCFGs reduce the differences in the cost tables by half, but do not reduce the costs in the propagated vectors; thus, they allow the required changes to take place faster.

7. Conclusions

We investigated the effect of using damping and split constraint factor graphs within the Max-sum algorithm, the distributed version of Belief Propagation, which was adopted for solving DCOPs.

In terms of computational bounds for convergence, we proved that in acyclic problems, where Max-sum is guaranteed to converge to the optimal solution, in the worst case scenario, damping slows the convergence to weakly polynomial time. This bound is significant in distributed scenarios where agents are not aware of the global topology of the system. Thus, they cannot avoid the use of damping when the structure of the graph guarantees convergence.

We empirically investigated the performance of Max-sum when using damping (DMS), and we compared the results for different values of the damping parameter λ. Our results indicate that the most successful versions of the algorithm are for relatively high values of λ. Moreover, while damping improves the results of the algorithm drastically, in most cases it does not converge within $5,000$ iterations. However, when combined with an anytime framework that reports the best solution explored by the algorithm, DMS significantly outperforms the best versions of Max-sum, as well as outperforming a standard local search algorithm. This finding is in contrast to the common assumption in the graphical models literature that BP performs best when it converges [25,22].

We introduced a novel degree of freedom for balancing exploration and exploitation when using Max-sum for solving DCOPs: the level of asymmetry in the factor graph. To this end, we proposed converting standard factor graphs representing

a DCOP to equivalent split constraint factor graphs (SCFGs), in which each constraint is represented by two function-nodes. The level of asymmetry in SCFGs is determined by the similarity between the table costs of function-nodes representing the same constraint. Our empirical results indicate that by tuning the two degrees of freedom, the damping factor and the level of asymmetry, Max-sum can produce solutions of high quality within a small number of iterations, even when an anytime framework cannot be used.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This research was partially supported by the Israeli Ministry of Science and Technology.

References

[1] P.J. Modi, W. Shen, M. Tambe, M. Yokoo, Adopt: asynchronous distributed constraints optimization with quality guarantees, Artif. Intell. 161 (1–2) (2005) 149–180.
[2] A. Petcu, B. Faltings, A scalable method for multiagent constraint optimization, in: IJCAI, 2005, pp. 266–271.
[3] A. Gershman, A. Meisels, R. Zivan, Asynchronous forward bounding, J. Artif. Intell. Res. 34 (2009) 25–46.
[4] W. Yeoh, A. Felner, S. Koenig, BnB-adopt: an asynchronous branch-and-bound DCOP algorithm, J. Artif. Intell. Res. (JAIR) 38 (2010) 85–133.
[5] R.T. Maheswaran, J.P. Pearce, M. Tambe, Distributed algorithms for DCOP: a graphical-game-based approach, in: PDCS, 2004, pp. 432–439.
[6] W. Zhang, Z. Xing, G. Wang, L. Wittenburg, Distributed stochastic search and distributed breakout: properties, comparison and applications to constraints optimization problems in sensor networks, Artif. Intell. 161 (1–2) (2005) 55–88.
[7] R. Zivan, S. Okamoto, H. Peled, Explorative anytime local search for distributed constraint optimization, Artif. Intell. 212 (2014) 1–26.
[8] A. Petcu, B. Faltings, Approximations in distributed optimization, in: P. van Beek (Ed.), CP 2005, in: LNCS, vol. 3709, 2005, pp. 802–806.
[9] A. Farinelli, A. Rogers, A. Petcu, N.R. Jennings, Decentralized coordination of low-power embedded devices using the max-sum algorithm, in: AAMAS, 2008, pp. 639–646.
[10] R. Stranders, A. Farinelli, A. Rogers, N.R. Jennings, Decentralised coordination of mobile sensors using the max-sum algorithm, in: Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009, Pasadena, California, USA, July 11–17, 2009, 2009, pp. 299–304.
[11] M. Vinyals, J.A. Rodríguez-Aguilar, J. Cerquides, Constructing a unifying theory of dynamic programming DCOP algorithms via the generalized distributive law, Auton. Agents Multi-Agent Syst. 22 (3) (2011) 439–464.
[12] S.M. Aji, R.J. McEliece, The generalized distributive law, IEEE Trans. Inf. Theory 46 (2) (2000) 325–343.
[13] Z. Chen, Y. Deng, T. Wu, Z. He, A class of iterative refined max-sum algorithms via non-consecutive value propagation strategies, Auton. Agents Multi-Agent Syst. 32 (6) (2018) 822–860.
[14] W.T.L. Teacy, A. Farinelli, N.J. Grabham, P. Padhy, A. Rogers, N.R. Jennings, Max-sum decentralized coordination for sensor systems, in: AAMAS, 2008, pp. 1697–1698.
[15] S.D. Ramchurn, A. Farinelli, K.S. Macarthur, N.R. Jennings, Decentralized coordination in robocup rescue, Comput. J. 53 (9) (2010) 1447–1461.
[16] P. Rust, G. Picard, F. Ramparany, Using message-passing DCOP algorithms to solve energy-efficient smart environment configuration problems, in: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9–15 July, 2016, pp. 468–474.
[17] C. Yanover, T. Meltzer, Y. Weiss, Linear programming relaxations and belief propagation – an empirical study, J. Mach. Learn. Res. 7 (2006) 1887–1907.
[18] R. Zivan, T. Parash, Y. Naveh, Applying max-sum to asymmetric distributed constraint optimization, in: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, 2015, pp. 432–439.
[19] N. Lazic, B. Frey, P. Aarabi, Solving the uncapacitated facility location problem using message passing algorithms, in: International Conference on Artificial Intelligence and Statistics, 2010, pp. 429–436.
[20] P. Som, A. Chockalingam, Damped belief propagation based near-optimal equalization of severely delay-spread UWB MIMO-ISI channels, in: 2010 IEEE International Conference on Communications (ICC), IEEE, 2010, pp. 1–5.
[21] D. Tarlow, I. Givoni, R. Zemel, B. Frey, Graph cuts is a max-product algorithm, in: Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence, 2011.
[22] M. Pretti, A message-passing algorithm with damping, J. Stat. Mech. Theory Exp. 11 (2005) P11008.
[23] A. Rogers, A. Farinelli, R. Stranders, N.R. Jennings, Bounded approximate decentralized coordination via the max-sum algorithm, Artif. Intell. 175 (2) (2011) 730–759.
[24] R. Zivan, H. Peled, Max/min-sum distributed constraint optimization through value propagation on an alternating DAG, in: AAMAS, 2012, pp. 265–272.
[25] K.P. Murphy, Y. Weiss, M.I. Jordan, Loopy belief propagation for approximate inference: an empirical study, in: UAI '99: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, Stockholm, Sweden, July 30–August 1, 1999, 1999, pp. 467–475.
[26] A.C.P. Som, Damped belief propagation based near-optimal equalization of severely delay-spread UWB MIMO-ISI channels, in: 2010 IEEE International Conference on Communications (ICC), 2010, pp. 1–5.
[27] D. Dueck, Affinity propagation: clustering data by passing messages, Ph.D. thesis, University of Toronto, 2009.
[28] S. Okamoto, R. Zivan, A. Nahon, Distributed breakout: beyond satisfaction, in: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9–15 July 2016, 2016, pp. 447–453.
[29] A.B.M. Verman, P. Stutz, Solving distributed constraint optimization problems using ranks, in: AAAI Workshop: Statistical Relational Artificial Intelligence, 2014.
[30] A.T. Ihler, D.A. McAllester, Particle belief propagation, in: Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, Florida, USA, April 16–18, 2009, 2009, pp. 256–263.
[31] J. Pacheco, E.B. Sudderth, Proteins, particles, and pseudo-max-marginals: a submodular approach, in: Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6–11 July 2015, 2015, pp. 2200–2208.
[32] P. Rebeschini, S.C. Tatikonda, Accelerated consensus via min-sum splitting, in: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4–9 December 2017, Long Beach, CA, USA, 2017, pp. 1374–1384.
[33] N. Ruozzi, S. Tatikonda, Message-passing algorithms: reparameterizations and splittings, IEEE Trans. Inf. Theory 59 (9) (2013) 5860–5881.
[34] F.R. Kschischang, B.J. Frey, H.A. Loeliger, Factor graphs and the sum-product algorithm, IEEE Trans. Inf. Theory 47 (2) (2001) 181–208.

[35] R. Zivan, T. Parash, L. Cohen, H. Peled, S. Okamoto, Balancing exploration and exploitation in incomplete min/max-sum inference for distributed constraint optimization, Auton. Agents Multi-Agent Syst. 31 (5) (2017) 1165–1207.
[36] D. Sontag, T. Meltzer, A. Globerson, T. Jaakkola, Y. Weiss, Tightening LP relaxations for MAP using message passing, in: UAI, 2008, pp. 503–510.
[37] H. Yedidsion, R. Zivan, A. Farinelli, Explorative max-sum for teams of mobile sensing agents, in: International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '14, Paris, France, May 5–9, 2014, 2014, pp. 549–556.
[38] Y. Weiss, Correctness of local probability propagation in graphical models with loops, Neural Comput. 12 (1) (2000) 1–41.
[39] A. Farinelli, A. Rogers, A. Petcu, N. Jennings, Decentralised coordination of low-power embedded devices using the max-sum algorithm, in: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2, International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 639–646.
[40] C. Kiekintveld, Z. Yin, A. Kumar, M. Tambe, Asynchronous algorithms for approximate distributed constraint optimization with quality bounds, in: AAMAS, 2010, pp. 133–140.
[41] F. DiMaio, J.W. Shavlik, Belief propagation in large, highly connected graphs for 3d part-based object recognition, in: Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), Hong Kong, China, 18–22 December 2006, 2006, pp. 845–850.
[42] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann, San Francisco, California, 1988.
[43] L. Cohen, R. Zivan, Max-sum revisited: the real power of damping, in: Workshop on Multi Agent Optimization (OptMAS) at AAMAS 2017, São Paulo, Brazil, May, 2017, 2017, pp. 1505–1507.