Autonomous Agents and Multi-Agent Systems (2023) 37:40
https://doi.org/10.1007/s10458-023-09621-w

Effect of asynchronous execution and imperfect communication on max-sum belief propagation

Roie Zivan[1] · **Ben Rachmut**[1] · **Omer Perry**[1] · **William Yeoh**[2]

Accepted: 27 July 2023

Abstract
Max-sum is a version of belief propagation that was adapted for solving distributed constraint optimization problems. It has been studied theoretically and empirically, extended to versions that improve solution quality and converge rapidly, and is applicable to multiple distributed applications. The algorithm was presented both as synchronous and asynchronous algorithms. However, neither the differences in the performance of the two execution versions nor the implications of imperfect communication (i.e., massage delay and message loss) on the two versions have been investigated to the best of our knowledge. We contribute to the body of knowledge on Max-sum by: (1) Establishing the theoretical differences between the two execution versions of the algorithm, focusing on the construction of beliefs; (2) Empirically evaluating the differences between the solutions generated by the two versions of the algorithm, with and without message delay or loss; and (3) Establishing both theoretically and empirically the positive effect of damping on reducing the differences between the two versions. Our results indicate that, in contrast to recent published results indicating that message latency has a drastic (positive) effect on the performance of distributed local search algorithms, the effect of imperfect communication on Damped Max-sum (DMS) is minor. The version of Max-sum that includes both damping and splitting of function nodes converges to high quality solutions very fast, even when a large percentage of the messages sent by agents do not arrive at their destinations. Moreover, the quality of solutions in the different versions of DMS is dependent of the number of messages that were received by the agents, regardless of the amount of time they were delayed or if these messages are only a portion of the total number of messages that was sent by the agents.

Keywords Belief propagation · Distributed constraints · Distributed problem solving

1 Introduction

Recent advances in computation and communication have resulted in realistic distributed applications in which humans and technology interact and aim to optimize mutual goals (e.g., IoT applications). A promising multi-agent approach to solve these types of problems is to model them as *distributed constraint optimization problems* (DCOPs), where decision

Extended author information available on the last page of the article

Published online: 14 September 2023

makers are modeled as *agents* that assign *values* to their *variables*. The goal in a DCOP is to optimize a global objective in a decentralized manner. Unfortunately, the communication assumptions of the DCOP model are overly simplistic and often unrealistic: (1) Messages are never lost; (2) Messages have very small and bounded delays; and (3) Messages arrive in the order that they were sent. These assumptions do not reflect real-world characteristics, where messages may be disproportionately delayed, or dropped, due to congestion and bandwidth limitations.

Recently, a study that investigated the effect of message latency on common DCOP local search algorithms (e.g., MGM and DSA) has shown that message delays have a *dramatic positive effect* on the performance of the asynchronous versions of these algorithms [1]. Specifically, message latency generates an exploration effect, which significantly improves the quality of the solutions found. Nevertheless, this study did not investigate the effect on distributed incomplete inference algorithms (e.g., Max-sum), even though they have been shown to be very successful [2, 3].

Max-sum is a version of the belief propagation algorithm [4, 5] that is used to solve DCOPs. It has been used for solving multi-agent optimization problems in applications such as sensor networks [6, 7], task allocation for rescue teams in disaster areas [8], and smart homes [9]. As with most belief propagation algorithms, Max-sum is known to converge to an optimal solution when solving problems represented by acyclic graphs. On problems represented by cyclic graphs, the beliefs may fail to converge, and the resulting assignments that are considered optimal under those beliefs may be of low quality [10, 11]. This occurs because the cyclic structure results in the propagation of duplicated information, leading to computation of inaccurate and inconsistent information [4].

To decrease the effect of duplicated information propagation, *damping* can be used. It balances the weight of the new calculation performed in each iteration and the weight of calculations performed in previous iterations, resulting in an increased probability for convergence [3]. Recently, splitting nodes in the factor graph on which belief propagation operates has been shown to be an effective method for accelerating the convergence of the algorithm when combined with damping [3, 12].

Max-sum has been presented both as an asynchronous algorithm and as a synchronous algorithm [10, 11, 13]. In the synchronous version, agents perform in iterations. In each iteration, agents send messages to all their neighbors and wait for the messages sent to them from all their neighbors to arrive before moving to the next iteration. In the asynchronous version, agents react to messages as and when they arrive.

To best of our knowledge, the implications of this difference in the execution of the algorithm on its performance have not been studied to date. Moreover, when message loss is considered, the synchronous version is not applicable since an agent may remain idle while it waits for the arrival of a message that was lost. While message latency does not affect the actions that agents perform (only delays them) in the synchronous version, intuitively, it is expected to have a major effect on the performance of the asynchronous version. The reason is that the beliefs included in messages are used by agents in the construction of beliefs that they propagate to others and in their assignment selection. In asynchronous execution, belief construction and assignment selection might be performed while considering imbalanced and inconsistent information.

In this paper, we make the following contributions[1]:

[1] This work is an extension of our published paper in the International Conference on Principles and Practice of Constraint Programming (CP) 2021 [14].

1. We investigate the differences in the properties of the two execution versions of Max-sum, synchronous and asynchronous. More specifically, using backtrack cost trees (BCTs) [15], we investigate the possible differences between the propagated beliefs in the two versions of Max-sum. Since BCTs, as originally defined [15], are applicable for the synchronous version only, in order to perform the analysis, we define a general BCT (GBCT) structure that is applicable for both modes of execution.
2. We investigate the effect of damping on asynchronous Max-sum. While there are clear indications (both empirical and theoretical) that damping improves the performance of the synchronous version of Max-sum [3, 15], to best of our knowledge, the effect of damping on the asynchronous version of Max-sum has not been studied prior to our study. We analyze this effect both theoretically and empirically. Both indicate that damping reduces the differences between synchronous and asynchronous execution.
3. We investigate the performance of the different versions of the algorithm in the presence of message latency and message loss. While the beliefs propagated and the computation that agents perform are not affected by message latency in the synchronous version (only delayed), this is not true for the asynchronous version. Once again, our empirical results reveal that damping reduces the differences. Moreover, the version of Max-sum proposed by Cohen et al. [3] that includes both damping and splitting maintains its fast convergence properties and high quality of solutions, even in asynchronous execution with message delays and when many messages are lost.

Our results include experiments that reveal that the quality of solutions produced by the different versions of DMS is mainly determined by the number of messages received by the agents, regardless of the time they were delayed or the number of messages that were sent. This finding is consistent with our theoretical results.

The paper is constructed as follows: We start by presenting related work in Sect. 2. Section 3 details the relevant background. Our theoretical study on the effect of asynchronous execution of the Max-sum algorithm is presented in Sect. 4 followed by our empirical study in Sect. 5. Finally, our conclusions are presented in Sect. 6.

2 Related work

Belief propagation was first introduced by Pearl [4] and was intensively studied before it was adopted by the multi-agent optimization community [16–19]. The version of belief propagation that was adapted to solve DCOPs, the Max-sum algorithm, was proposed by Farinelli et al. [10]. In that paper, the algorithm was described in its asynchronous version. Early on, researchers noticed that when the algorithm fails to converge, it performs poorly and, thus, they suggested versions that guaranteed convergence [20, 21].

Rogers et al. [20] proposed a manipulation of the factor graph that the algorithm uses that will guarantee its convergence. The algorithm starts by eliminating edges from the factor graph until a spanning tree of the original factor graph remains. Then the Max-sum algorithm is used in order to produce the optimal solution to the spanning tree factor graph. By accumulating the maximal additional cost of every removed edge, it is possible to calculate a bound on the difference between the cost of the optimal solution for the spanning tree and the optimal solution for the original factor graph. Hence, this algorithm is known as *Bounded Max-sum*. Later studies proposed methods for selecting the spanning tree that improve the bound [22, 23]. Unfortunately, while the algorithm offered a bound from the

optimum, the cost of the solution it proposed was insignificantly better than the non-converging standard version of Max-sum [11].

Zivan and Peled [21] proposed a different manipulation on the factor graph to trigger convergence. They converted the factor graph to a directed acyclic graph by selecting an order on all nodes of the graph and allowing messages to be sent only in this order. This algorithm is guaranteed to converge in linear time. However, in order to consider all constraints, the order was reversed. The best outcome was achieved by performing a small number of phases in alternating directions, and then performing a number of phases that include value propagation. Chen et al. [2] later extended this algorithm to versions that balance between exploration and exploitation.

Later, Cohen et al. [3] introduced a version of the algorithm that included damping, a method for encouraging convergence that was used in other versions of belief propagation [16], and splitting of function nodes, a method that was theoretically investigated by Ruozzi and Tatikonda [12]. This version converged very fast to high quality solutions.

Recently, a number of papers addressed the main limitation of Max-sum, which is the exponential calculation required by function-nodes in order to produce the messages they send to neighboring variable nodes [13, 24]. While these proposed methods evidently reduce the computation effort required for producing messages by function-nodes, the process is still exponential in the arity of the constraints. Thus, as in prior work [3, 11], we focus on binary DCOPs where Max-sum performs efficiently in this work as well.

Max-sum has been used to solve asymmetric DCOPs [25] 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 the splitting method [3] is that, the use of more than one function-node for a single constraint was intended to represent the given natural structure of an asymmetric problem [25]. In contrast, in the work by Cohen et al. [3] (and in this study), it was used as an algorithmic method to accelerate convergence.

3 Background

In this section we provide background on *graphical models*, *distributed constraint optimization problems* (DCOPs), the DCOP versions of belief propagation—*Max-sum* and its variants—and *backtrack cost tree* (BCT)—the tool we use to analyze the algorithms' behavior. While the Max-sum variants that we discuss are actually solving a min-sum problem [12], we will still refer to them as "Max-sum" since this name is commonly used in the DCOP literature [10, 11, 26].

3.1 Graphical models

Graphical models such as Bayesian networks or constraint networks are a widely used representation framework for reasoning and solving optimization problems. The graph structure is used to capture dependencies between variables [27]. Our work extends the theory established by Weiss [17] that considered the Maximum a posteriori (MAP) assignment, which is solved using the Max-product version of belief propagation.

The relation between MAP and constraint optimization is well established [10, 27, 28] and, thus, results that consider Max-product for MAP apply to Max/Min-sum for solving constraint optimization problems, as well as the other way around [12]. Therefore, without loss of generality, we will focus on constraint optimization in this paper. Moreover, we will consider the distributed version of the problem since it is a natural representation for message passing algorithms. Nevertheless, our results apply to any version of problem represented by a graphical model and solved by distributed belief propagation.

3.2 Distributed constraint optimization problems

A *distributed constraint optimization problem* (DCOP) is defined by a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$, where:

- $\mathcal{A}$ is a finite set of agents $\{A_1, A_2, \ldots, A_n\}$.
- $\mathcal{X}$ is a finite set of variables $\{X_1, X_2, \ldots, X_m\}$, where each variable is held by a single agent and an agent may hold more than one variable.
- $\mathcal{D}$ is a set of domains $\{D_1, D_2, \ldots, D_m\}$, where each domain D_i contains the finite set of values that can be assigned to variable X_i. We denote an assignment of value $x \in D_i$ to X_i by an ordered pair $\langle X_i, x \rangle$.
- $\mathcal{R}$ is a set of relations (constraints), where 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] For each binary constraint R_{ij}, there is a corresponding cost table T_{ij} with dimensions $|D_i| \times |D_j|$ in which the cost in every entry e_{xy} is the cost incurred when x is assigned to X_i and y is assigned to X_j. A *binary DCOP* is a DCOP in which all constraints are binary. A *partial assignment* is a set of value assignments to variables, in which each variable appears at most once. *vars*(*PA*) is the set of all variables that appear in partial assignment *PA* (i.e., $vars(PA) = \{X_i \mid \exists x \in D_i \wedge \langle X_i, x \rangle \in PA\}$). 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 applicable constraints to *PA* over the value assignments in *PA*. A *complete assignment* (or a *solution*) is a partial assignment that includes all the DCOP's variables (i.e., $vars(PA) = \mathcal{X}$). 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., $n = m$) and we concentrate on binary DCOPs. These assumptions are common in the DCOP literature [29, 30]. In addition to the standard motivation for focusing on binary DCOPs, in the case of Max-sum, it is essential since the runtime complexity of each iteration of Max-sum is exponential in the arity of the constraints.

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

3.3 The max-sum algorithm

Max-sum operates on a *factor graph*, which is a bipartite graph in which the nodes represent variables and constraints [31]. Each variable-node representing a variable of the original DCOP is connected to all function-nodes representing constraints that it is involved in. Similarly, a function-node is connected to all variable-nodes representing variables in the original DCOP that are involved in it. Variable-nodes and function-nodes are considered "agents" in Max-sum (i.e., they can send and receive messages and can perform computation).

A message sent to or from variable-node X (for simplicity, we use the same notation for a variable and the variable-node representing it) is a vector of size $|D_X|$, including a cost for each value in D_X. These costs are also called *beliefs*. Before the first iteration, all nodes assume that all messages they previously received (in iteration 0) are vectors of zeros. A message sent from a variable-node X to a function-node F in iteration $k \geq 1$ is formalized as follows:

$$Q^k_{X \to F} = \sum_{F' \in F_X, F' \neq F} R^{k-1}_{F' \to X} - \alpha \tag{1}$$

where $Q^k_{X \to F}$ is the message that variable-node X intends to send to function-node F in iteration k, F_X is the set of function-node neighbors of variable-node X, and $R^{k-1}_{F' \to X}$ is the message sent to variable-node X by function-node F' in iteration $k-1$. α is a constant that is reduced from all beliefs included in the message (i.e., for each $x \in D_X$) in order to prevent the costs carried by messages throughout the run of the algorithm from growing arbitrarily large.

A message $R^k_{F \to X}$ sent from a function-node F to a variable-node X in iteration k includes for each value $x \in D_X$:

$$\min_{PA_{-X}} cost(\langle X, x \rangle, PA_{-X}) \tag{2}$$

where PA_{-X} is a possible combination of value assignments to variables involved in F not including X. The term $cost(\langle X, x \rangle, PA_{-X})$ represents the cost of a partial assignment $a = \{\langle X, x \rangle, PA_{-X}\}$, which is:

$$f(a) + \sum_{X' \in X_F, X' \neq X, \langle X', x' \rangle \in a} (Q^{k-1}_{X' \to F})_{x'} \tag{3}$$

where $f(a)$ is the original cost in the constraint represented by F for the partial assignment a, X_F is the set of variable-node neighbors of F, and $(Q^{k-1}_{X' \to F})_{x'}$ is the cost that was received in the message sent from variable-node X' in iteration $k-1$, for the value x' that is assigned to X' in a. X selects its value assignment $\hat{x} \in D_X$ following iteration k as follows:

$$\hat{x} = \arg\min_{x \in D_X} \sum_{F \in F_X} (R^k_{F \to X})_x \tag{4}$$

In the synchronous version (*Syn_Max-sum*), in each iteration, an agent waits to receive all messages sent to it in the previous iteration before performing computation and generating the messages to be sent in the current iteration [11]. In the asynchronous version (*Asy_Max-sum*), agents react to messages they receive. Whenever a node receives a message, it performs computation and sends out messages to its neighbors, taking into consideration

the last message received from each of its neighbors [10]. In both versions, the logic for the actions of the agents are identical, only the trigger for performing those actions is different.

3.3.1 Damped max-sum

Damped Max-sum (DMS) has an additional feature, which is the damping of the propagated beliefs. In order to add damping to Max-sum, a parameter $\lambda \in [0, 1)$ is used. Before sending a message in iteration k, a node in the factor graph (whether it is a variable-node or a function-node) performs calculations as in standard Max-sum. We use $\widehat{m^k_{i\to j}}$ to denote the result of the calculation made by node N_i for the content of a message intended to be sent from N_i to node N_j in iteration k and $m^{k-1}_{i\to j}$ to denote the message sent by N_i to N_j at iteration $k-1$. Notice that $m^k_{i\to j}$ can be either a Q message or a R message. The message sent by N_i to N_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 given to previously performed calculations with respect to the most recent calculation performed. Moreover, when $\lambda = 0$ the resulting algorithm is standard Max-sum.

We use *Syn_DMS* and *Asy_DMS* to denote the synchronous and asynchronous versions of DMS, respectively, in this paper.

3.3.2 Asynchronous execution

All the definitions used for describing Max-sum (and DMS) above use the iteration number k. It was used to describe how a message is generated, using the information received by the factor graph node in the previous iteration $(k-1)$. In asynchronous execution, their are no iterations, and agents perform computation steps whenever they receive messages. Thus, in asynchronous execution, the information that a node N_i uses to generate a message at time t is the information included in the last message received from each of its neighbors prior to t, regardless of when it was sent by the neighbors. If no message has been received from a particular neighbor yet, N_i uses a vector of zeros in its computation for that neighbor.

Notice, that in the presence of message delays, a node N_i may receive messages from its neighbor not in the order they were sent. This is true for both the synchronous and the asynchronous versions of the algorithm. Nevertheless, the agents use the messages in the order in which they were received. In order to avoid this phenomenon, we implemented a time-stamp method that allows the agents receiving messages to consider the information they include in the order that they were sent. However, the results were not significantly different from the results obtained when this method was not used. Thus, we do not report these results in our empirical study.

3.3.3 Max-sum with split constraint factor graphs

When Max-sum is applied to an asymmetric problem, the representing factor graph has each (binary) constraint represented by two function-nodes, one for each part of the

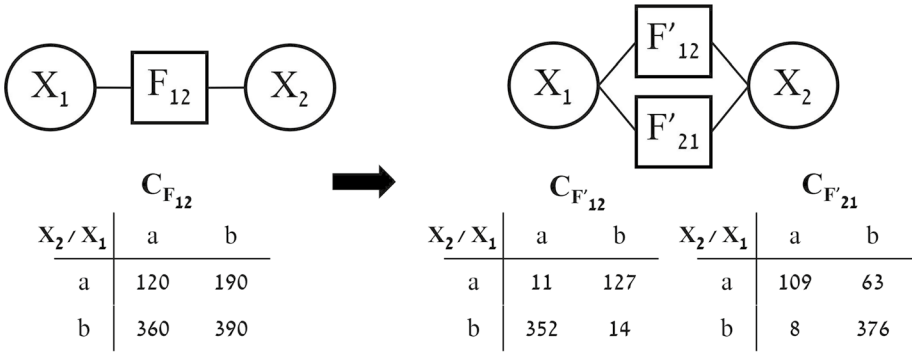

Fig. 1 An acyclic DCOP factor graph (on the left) and its equivalent SCFG (on the right)

constraint held by one of the involved agents. Each function-node is connected to both variable-nodes representing the variables involved in the constraint [32]. Figure 1 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 the 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. In this example, function-node F'_{12} is held by agent A_1, while F'_{21} is held by A_2. The factor graphs are equivalent since the sum of the two cost tables held by 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 [25] for details). Such *split constraint factor graphs* (SCFGs) can be used as an enhancement method for Max-sum [3, 12]. This is achieved by splitting each constraint that was represented by a single function-node in the original factor graph into 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 tuning the similarity between the two function-nodes representing the same constraint one can determine the level of asymmetry in the SCFG. The use of symmetric SCFGs was shown to trigger very fast convergence to high quality solutions. However, generating mild asymmetry, postpones convergence and generates some exploration, which results in improved solution quality [3].

3.3.4 Non-concurrent logic operations

In order to evaluate the runtime performance of distributed algorithms performing in a distributed environment, independent of the implementation details, there is a need to establish which of the operations performed by agents could not have been performed concurrently. Thus, the runtime performance of the algorithm is the longest non-concurrent sequence of operations that the algorithm performed. This method was first

proposed for the evaluation of asynchronous distributed algorithms for solving distributed constraint satisfaction problems (DisCSPs) [33]. As the basic logic operations of DisCSP algorithms are constraint checks, researchers have measured their runtimes in terms of *non-concurrent constraint checks* (NCCCs) [33]. To better compare different logic operations in other classes of algorithms, researchers generalized NCCCs to *non-concurrent logic operations* (NCLOs) [34]. We adopt NCLOs in this study.

3.4 Backtrack cost trees

For analyzing the behavior of Max-sum on factor graphs with an arbitrary (finite) number of cycles, Zivan et al. [15] proposed the use of a *backtrack cost tree* (BCT). It allows one to trace, for each belief, the entries in the cost tables held by function-nodes that were used to compose this belief. In other words, the components of the assignment's cost. Their analysis included insights regarding the construction of beliefs from costs incurred by constraints. Thus, for every pair of constrained variables X_i and X_j, for each $x \in D_i$, $x' \in D_j$, the cost incurred by the constraint for assigning x to X_i and x' to X_j was denoted as $R(X_i = x, X_j = x')$. Formally, a BCT is defined as follows:

Definition 1 (*Backtrack Cost Tree (BCT)*) A BCT is defined for a belief that appears either in a message sent from variable X_i at time t to a function node connecting it to a variable X_j or in a message sent from that function node to variable X_i. The belief is on the cost of assigning some value $x \in D_i$ to variable X_i. Without loss of generality, we will elaborate on the first among these two and denote it as $BCT^t_{i=x\rightarrow j}$.

The belief, as constructed by the Max-sum algorithm, is a sum of various components and the tree is composed from them. At the root is the decision to assign some value to a variable (e.g., assigning some $x \in D_i$ to X_i) and the directed edges from its children in the tree include the beliefs that were summed in order to generate the cost (the belief) for this assignment. These edges lead to nodes representing the neighboring nodes from which X_i received messages in time $t-1$. Each of those nodes is connected to the nodes from which they received messages at time $t-2$, with the edges containing the beliefs that passed to it and their sum ended up in its message. The tree leaves are all at time 0 (see Fig. 2b).

For a single-cycle factor graph, the BCT for every belief is a chain. Factor graphs with multiple cycles include variable-nodes with more than two neighbors and, thus, the BCTs of their beliefs include nodes with multiple children.

A BCT starts from an end point (e.g., the root of the BCT as presented in Fig. 2b), which is the *belief* (cost) of assigning to X_i some value x from its domain D_i, as sent to a neighboring node (in our example it is the assignment of $x \in D_1$ to X_1. The values from which that belief was calculated can then be backtracked to the messages and costs due to all the individual constraints that were summed up to create that belief. An example of such a tree for a belief generated when Max-sum solves the factor-graph depicted in Fig. 2a is depicted in Fig. 2b.

For each BCT, there is an implied assignment tree that consists of the value assignments that the variables at each time-point of the tree would need to be assigned in order to incur the costs included in the BCT. The value assignment selected by a variable at time t is the one with the minimal sum of beliefs sent to the corresponding variable-node at iteration $t-1$. The tree for this minimal sum of beliefs will be denoted by

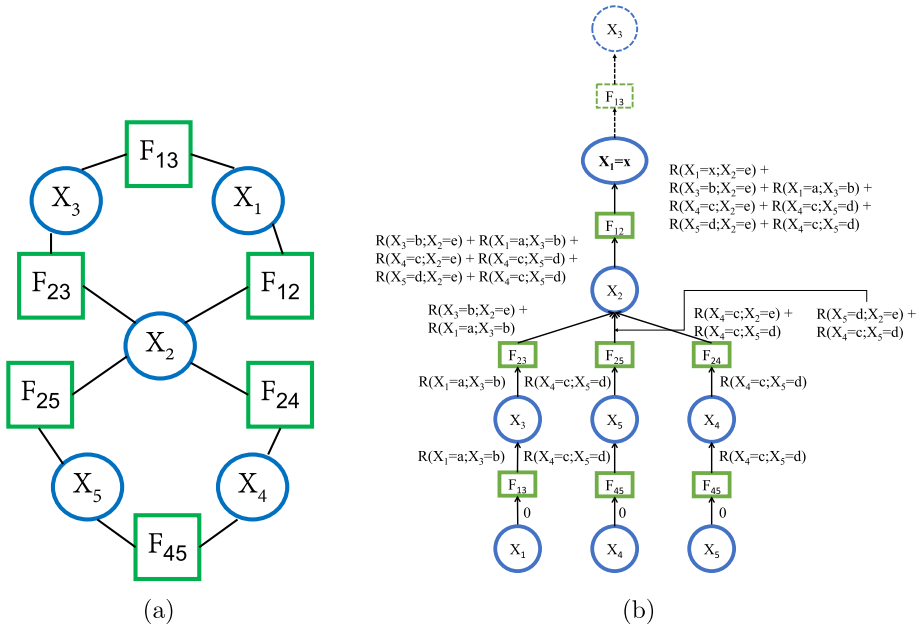

Fig. 2 **a** A lemniscate factor-graph. **b** An example of a BCT for a belief in the message sent from X_1 to the function-node F_{13} at time $t = 6$ in the lemniscate depicted on the left hand side

BCT_i^t, as it does not depend on any specific belief that appears in a message to another variable.

3.5 Convergence properties

Belief propagation converges in linear time to an optimal solution when the problem's corresponding factor graph is acyclic [4]. For a single-cycle factor graph, we know that if belief propagation converges, then it is to an optimal solution [17, 18]. Moreover, when the algorithm does not converge, it periodically changes its set of assignments. In order to explain this behavior, Forney et al. [18] show the similarity in the performance of the algorithm on a cycle to its performance on a chain, whose nodes are similar to the nodes in the cycle, but whose length is equal to the number of iterations performed by the algorithm. One can consider a sequence of messages starting at the first node of the chain and heading towards its other end. Each message carries beliefs accumulated from costs added by function-nodes. Each function-node adds a cost to each belief, which is the constraint value of a pair of value assignments to its neighboring variable-nodes. Each such sequence of cost accumulation (route) must at some point become periodic, and the minimal belief would be generated by the minimal periodic route. If this periodic route is consistent (i.e., the set of assignments implied by the costs contain a single value assignment for each variable), then the algorithm converges. Otherwise, it does not.

Recently, these insights were generalized such that similar statements can be made when the algorithm is solving factor graphs with multiple cycles. Specifically (using

BCTs), Zivan et al. [15] proved that, as in the single cycle case, on every finite factor graph, Max-sum at some point in time starts to repeatedly follow a path that minimizes its beliefs.

4 Effect of asynchronous execution

In order to analyze the differences in performance of the synchronous version of Max-sum (Syn_Max-sum) and the asynchronous version of Max-sum (Asy_Max-sum), one must investigate the differences in the structure of the BCTs of beliefs sent by the algorithms' nodes. However, in Sect. 3.4, BCTs were defined with respect to synchronous execution, referring to messages sent in a specific time. Thus, there is a need for a more general definition that will apply to both synchronous and asynchronous execution, as well as environments that include message latency and message loss.

Definition 2 (*General BCT (GBCT)*) A GBCT is defined for a belief that appears either in a message sent from variable X_i to a function node connecting it to a variable X_j or in a message sent from that function node to variable X_i. The belief is on the cost of assigning some value $x \in D_i$ to variable X_i. Without loss of generality (as we did above), we will elaborate on the first among these two and denote it as $GBCT^t_{i=x \rightarrow j}$.

As in a standard BCT, at the root of a GBCT is the decision to assign some value to a variable (e.g., assigning some $x \in D_i$ to X_i) and the directed edges from its children in the tree include the beliefs that were sent to X_i, which were summed in order to generate the cost (the belief) for this assignment. Similar to a BCT, the definition is recursive and applies to every cost sent by a node in the tree that was summed in order to generate the belief at the root of the tree. For every node X_j, that sent a message with a belief $x' \in D_j$, the cost on an edge connecting it to a child is the belief carried by the last message received by X_j from that child, before X_j sent the message with the belief for x'.

In contrast to the definition of the standard BCT, in GBCT, we do not know when the messages were sent or received. All we know is the content of the messages that were received. Specifically, the last message that was received by X_i from each of its neighbors (except for the neighbor to whom b is sent), before generating the message with the belief b for the assignment $\langle x, X_i \rangle$, is the one that is considered in the GBCT. Each of the nodes sending these messages is the parent in the tree of the nodes sending messages to it. For example, assume that F_{ij} sent a message m to X_i with a cost for value x and that this cost corresponds to the assignment of $x' \in D_j$. Further assume that m was the last message that X_i received from F_{ij} before producing b. Thus, F_{ij} is the child of the node X_i in the GBCT and X_j is the child of F_{ij}. The cost on the edge between F_{ij} and X_i is the belief corresponding to x in m. F_{ij} is the parent of X_j and the cost on the edge between them is the cost included in the last message received by F_{ij} before it produced the belief that was sent in m.

In Syn_Max-sum, the height of a BCT for a belief included in a message sent at iteration t is t and, for each node in the tree, the heights of the sub-trees rooted by each of its children nodes are equal. On the other hand, in Asy_Max-sum, messages can have different delays and, thus, each sub-tree in a GBCT can have a different height.

Our first theoretical property addresses the results proved by Zivan et al. [15] regarding the convergence of Syn_Max-sum. More specifically, we prove that the property that was proved in Lemma 1 in [15], and was used to prove the main theorem of that study (i.e., the

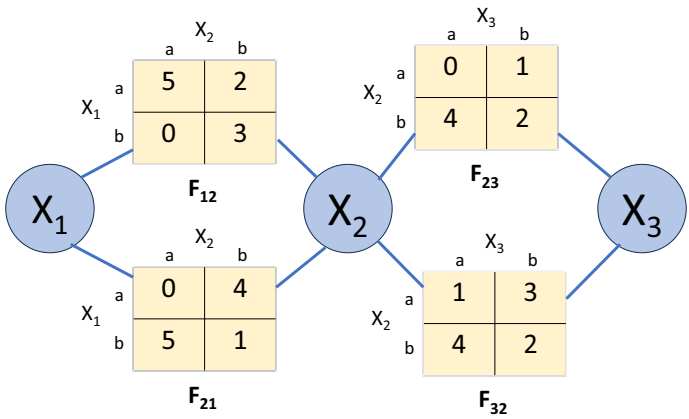

Fig. 3 Example of how message delays affect convergence

main theorem in [15]), is not guaranteed when Max-sum operates asynchronously in an environment that includes message delays.

Proposition 1 *In the presence of message delays, Asy_Max-sum is not guaranteed to converge to a minimal repeated route.*

Proof The structure of the GBCTs of the beliefs that are exchanged by agents depends on the arrival times of messages from which they are composed. Each GBCT (and, as a result, its corresponding belief) is an outcome of a specific combination of message arrivals, depending on whether messages were lost or delayed and by how much. These consequences result in different orders of message arrivals and the number of such combinations is exponential in the maximal number of messages that the beliefs they carry can be included in the GBCT. Moreover, due to message losses and delays, a specific minimal route of beliefs is not guaranteed to repeat itself. Thus, even if the algorithm reaches a minimal route, it may not repeat it. □

In order to provide an intuitive explanation to Proposition 1, we present an example in Fig. 3, which includes a leminiscate factor graph with three variable nodes and two function nodes. When performing Max-sum where messages arrive instantaneously, the algorithm oscillates between solutions: $\langle X_1 = a, X_2 = a, X_3 = a\rangle, \langle X_1 = b, X_2 = a, X_3 = a\rangle, \langle X_1 = b, X_2 = a, X_3 = b\rangle, \langle X_1 = b, X_2 = b, X_3 = b\rangle,$ $\langle X_1 = a, X_2 = b, X_3 = b\rangle, \langle X_1 = a, X_2 = a, X_3 = b\rangle, \langle X_1 = a, X_2 = a, X_3 = a\rangle$..., If messages from function nodes F_{12} and F_{21} to variable node X_2 are delayed while messages from other nodes arrive instantaneously, then the algorithm performing on the cycle including X_2, F_{23}, X_3 and F_{32} will converge to the solution $\langle X_2 = a, X_3 = a\rangle$[3] and X_1 will maintain its current assignment. When the messages from function nodes F_{12} and F_{21} will finally arrive and the communication limitation is resolved, the algorithm will oscillate once again.

[3] This is because this assignment results in a normalized cost that is lower than any alternative oscillating path. See [18, 35] for details.

Therefore, Proposition 1 seems to put an end to the natural wish that the convergence properties of Syn_Max-sum can be established for Asy_Max-sum as well. However, the differences between the executions of the two versions of the algorithm can be minimized. More specifically, the effect caused by sub-trees of the GBCTs having different heights in Asy_Max-sum can be significantly reduced through the use of damping.

Let $layer_k$ denote the set of nodes of a GBCT with depth k (distance from the root) and $GBCT_k$ denote the layers of the GBCT with depth k or less. We say that a $layer_k$ is *effective* if and only if there exists a number $\hat{k} \geq k$, such that the belief calculated by $GBCT_{\hat{k}}$ is different than the belief calculated by $GBCT_{k-1}$. For each GBCT G, we say that its effective GBCT G' is $GBCT_{k'}$ such that $layer_{k'}$ is effective and, for any $layer_k$ that is effective in B, $k' \geq k$.

In the proofs of the following properties, we assume that the messages have bounded delays and a probability of message loss that is small enough to prevent starvation (i.e., there is no agent A_i and number of non-concurrent steps ns',[4] such that following ns', A_i does not receive messages anymore), and there is a limit e for the number of consecutive messages that can be lost on a communication link (i.e., that are sent from an agent A_i to another agent A_j).

Lemma 1 *When asynchronous DMS (Asy_DMS) is performed with a large enough damping factor,*[5] *there exists a finite number of non-concurrent steps of the algorithm ns_1, such that in the steps following it, for every two beliefs included in the same message, if $layer_k$ in each of the corresponding GBCTs is effective, then the number of nodes in $layer_k$ of both GBCTs are equal.*

Proof A node in the GBCT represents a node in the factor-graph, and its children are the nodes from which it received messages. Assume that in two GBCTs of beliefs sent in the same message, there exists an effective layer k in which one GBCT has a smaller number of nodes than the other. That means that the factor-graph nodes represented by nodes in $layer_{k-1}$ did not receive messages from all their neighbors yet. However, since the delays are bounded and so is the number of messages that are lost, their must exist a time when messages from all neighbors will arrive. Following that time the size of the $layer_k$ in both GBCTs will be equal until the end of the run of the algorithm. □

An immediate corollary from Lemma 1 is that in Asy_DMS (using a large enough damping factor), following ns_1, the effective GBCTs of all beliefs included in each message have the same number of nodes. This reduces the possible differences between beliefs that can be generated by each node. Moreover, for the case that the algorithm does converge, the effect of the asynchronous performance vanishes, as we prove below.

Proposition 2 *When Asy_DMS is using a large enough damping factor, if after performing $ns_2 > ns_1$ (ns_1 as described in Lemma 1) non-concurrent steps, it reaches a minimal consistent route (i.e., all nodes perform k sequential asynchronous steps in which the value*

[4] We consider a step to be an action that starts when a node in the graph received some messages (at least one), performed computation, and ends when it sent some messages (at least one).

[5] For an analysis on the size of the damping factor required, with respect to the largest number of neighbors (degree) that a node in the factor graph has, see the work by Zivan et al. [15].

assignments corresponding to the minimal route are selected), *then it will repeatedly follow this route* (*i.e.*, *it has converged*).

Proof As established above, following ns_1, the effective GBCTs for beliefs included in the same message have the same number of nodes (in each layer and altogether) regardless of message delays. When the algorithm reaches a minimal consistent route, the beliefs corresponding to this minimal route involve only one value in each domain, and the belief corresponding to it is minimal in each message. Additional nodes added to the GBCTs of the beliefs corresponding to the assignments in the minimal route represent costs in the entries of the cost tables of function-nodes that are part of the minimal route. Hence, they will not change its minimal property or the choice of the minimal route assignments (i.e., for every $ns > ns_2$, the effective $GBCT_i^{ns}$ will be identical). Similarly, the addition of nodes to GBCTs of beliefs corresponding to assignments that are not included in the minimal route represent costs that belong to routes with larger overall costs. □

Proposition 2 has a major importance to our discussion. Both the asynchronous and the synchronous versions of DMS will converge when they reach a consistent minimal path. In other words, the differences between them can exist only when the minimal path is inconsistent. In such a case, the synchronous version will repeat the minimal inconsistent route while the asynchronous version may leave it and explore other routes.

5 Experimental evaluation

In order to evaluate the implications of asynchronous execution (compared to synchronous execution) and imperfect communication on the different versions of Max-sum, we used an asynchronous simulator, in which agents are implemented by Java threads. It includes a *mailing agent* that simulates the delays of messages as suggested by Zivan and Meisels [33]. Using this type of simulator allows us to implement any type of message delay pattern. Other simulators, such as ns-3 [36, 37], offer a number of communication patterns from which one can select. However, we prefer the use of the simpler simulator proposed by Zivan and Meisels [33], which allows complete flexibility in the design of the message delay patterns and it allows us to measure runtimes in implementation-agnostic units. Thus, the results are presented as a function of the number of *non-concurrent logic operations* (NCLOs). The atomic logic operations in these algorithms are the evaluation of the cost of a combination of two assignments (i.e., an access to the cost table of a function-node). Each agent performed the computation for the function-nodes that were assigned to it. We used a greedy heuristic to evenly assign function-nodes to agents and, thus, increase concurrency. In order to simulate message delays, for each message sent between nodes managed by different agents, a delay in terms of NCLOs was selected, and the message was delivered to the receiving agent after that agent had the opportunity to perform this number of logic operations.

We evaluated the algorithms on problems with 50 agents, which are often too large for complete DCOP algorithms to solve, and across four different types of DCOPs, described below. 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.

- *Random graph problems* These problems are random constraint graph topologies with density $p_1 = \{0.1, 0.6\}$. They include variables with 10 values in each domain. The cost tables held by function-nodes include costs that were selected uniformly between 100 and 200. This range was chosen because when the range is closer to zero, beliefs may be very small and the effect of damping is less effective. Both the constraint graph and the constraint functions in these problems are unstructured.
- *Graph coloring problems* These problems are random constraint graph topologies in which each variable has a number of values (i.e., colors) that it can take, and all constraints are "not-equal" cost functions, where an equal assignment of neighbors in the graph incurs a random cost between 100 and 200 and non-equal value assignments incur zero cost. Such random graph coloring problems are commonly used in DCOP formulations of resource allocation problems. We set the density to $p_1 = 0.05$ and set the number of values in each domain to 3 [3, 10, 38].
- *Scale-free network problems* These problems are generated using the model by Barabási and Albert [39]. An initial set of 10 agents was randomly selected and connected. Additional agents were added sequentially and connected to 3 other agents with a probability proportional to the number of links that the existing agents already had. The cost of each joint assignment between constrained variables was selected uniformly between 100 and 200. Each variable had 10 values in its domain. The constraint graph is somewhat structured but the constraint functions are unstructured. Similar problems were previously used to evaluate DCOP algorithms by Kiekintveld et al. [40].
- *Overlapped solar system problems* The overlapped solar system is a realistic problem, inspired by the Constant Speed Propagation Delay Model implemented in the ns-3 simulator [36, 37]. The graph topology is inspired by scale-free networks. An initial set of 5 agents are randomly selected to be the centers of the solar systems, and they are connected. Each of these agents A_i^c is assigned two coordinates that are drawn from a continuous uniform distribution: $x_i^c \sim U(0, 1)$ and $y_i^c \sim U(0, 1)$. All other agents (i.e., stars in the solar systems) are randomly assigned to one of the solar systems. The index c represents the solar system to which the agent is assigned, and it is equal to the index of the center agent of the solar system (i.e., if A_i^c is the center of a solar system, then $i = c$). The coordinates for an assigned agent (A_j^c where $j \neq c$) are drawn from a Normal distribution as follows: $x_j^c \sim N(\mu = x_i^c, \sigma = 0.05)$ and $y_j^c \sim N(\mu = y_i^c, \sigma = 0.05)$ based on the location of the center of the solar system that it was added to. The probability that two arbitrary agents A_i and A_j will be neighbors is defined by $p_{ij} = (1 - \frac{distance_{ij}}{maxDistance})^{\beta}$ where $distance_{ij}$ is the Euclidean distance between agents A_i and A_j, $maxDistance$ is the Euclidean distance between agent A_i and the location of the farthest agent, and β expresses the dependency of the probability that both agents will be neighbors on their distance one from the other (in our experiments we used $\beta = 3$). For each pair of agents, a random probability $p_r \in [0, 1]$ was generated, and two agents were considered as neighbors if $p_r < p_{ij}$. Costs between connected agents were selected uniformly between 100 and 200. While the structure of these problems is similar to scale-free networks, the addition of the geographic locations of nodes allows one to set the size of message delays and the probability of a message loss with respect to physical distance as specified below in Sect. 5.1.

In each experiment, we randomly generated 50 different problem instances. The results presented in the graphs are an average of those 50 runs. In order to demonstrate the convergence of the algorithms, we present the sum of costs of the constraints involved in the assignment

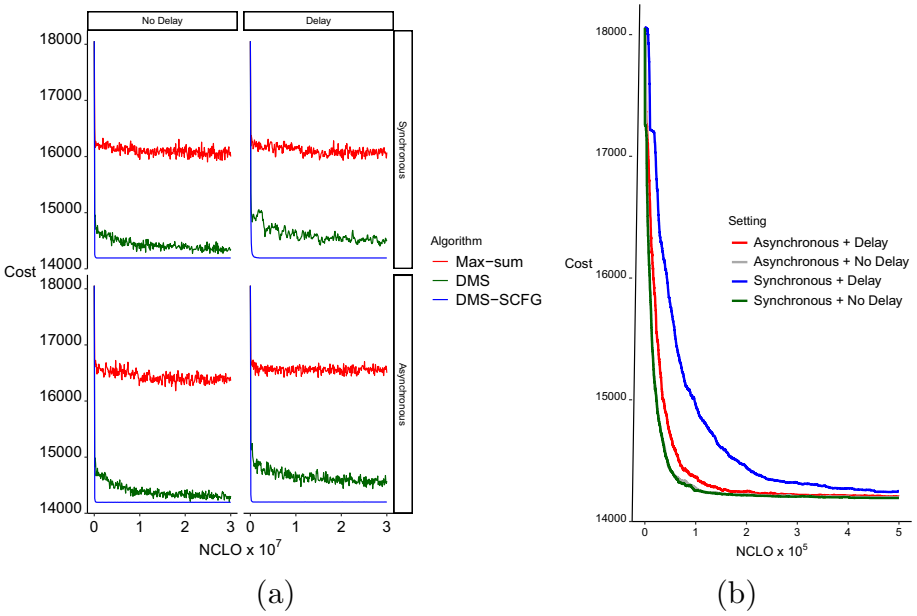

Fig. 4 **a** Solution quality as a function of NCLOs of Max-sum versions, with and without message delays, solving sparse random graph problems with $p_1 = 0.1$. **b** A closer look at the solution quality of DMS_SCFG versions on these problems

that would have been selected by each algorithm every 100k NCLOs. We also performed *t-tests* to evaluate the significance of differences between all presented results.

5.1 Communication scenarios

For random graph problems, graph coloring problems, and scale-free network problems, we used four types of communication scenarios: (1) Perfect communication; (2) Message latency selected from a uniform distribution $td_e \sim U(0, 10k)$ NCLOs; (3) Message loss determined by $p \sim U(0, 1)$ such that a message is not delivered if $p < pl_e$, where $pl_e = [0.3, 0.5, 0.7, 0.9]$ is a parameter denoting the probability for message loss; and, (4) Scenarios including both message latency and message loss.

For overlapped solar system problems, we set td_e and pl_e as follows: td_e was drawn from a Poisson distribution $d \sim \text{Pois}(\Gamma \cdot distance_{ij})$, where Γ is a constant and $distance_{ij}$ is the distance between the locations of the agents A_i and A_j. This is also in contrast to the constant speed propagation delay model implemented in ns-3, where the delays that were calculated as a function of the distance between the geographic locations of the nodes were fixed and never changed [36, 37]. Regarding message loss, we define the probability pl_e that a message sent on edge e between agents A_i and A_j is delivered as follows: $pl_e = \frac{distance_{ij}}{maxDistance_{ij}}$, where $maxDistance_{ij}$ is the distance of the furthest agent from A_i or A_j.

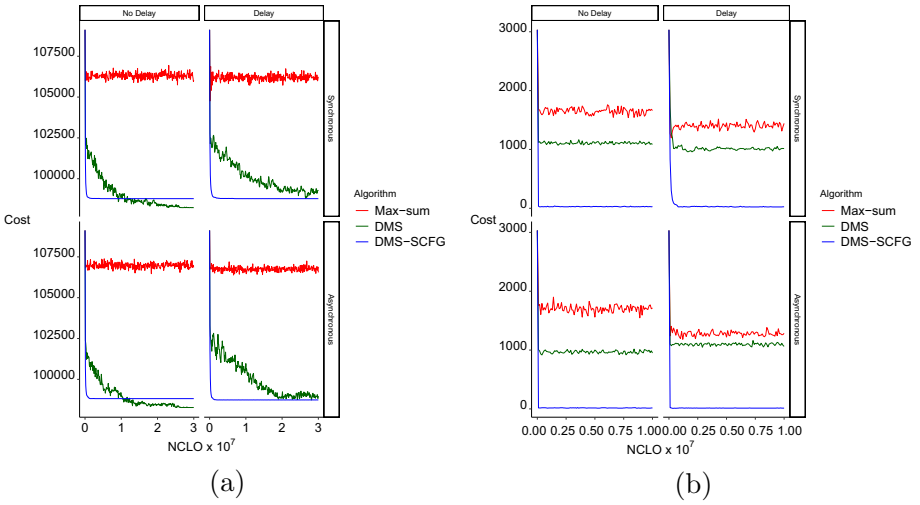

Fig. 5 Solution quality as a function of NCLOs of Max-sum versions, with and without message delays, solving **a** dense random graph problems with $p_1 = 0.6$ and **b** graph coloring problems

5.2 Impact of message delays

Figure 4a presents the quality of solutions produced by the different versions of Max-sum when solving sparse random graph problems with density $p_1 = 0.1$. Similar to most of the figures presented in this section, Fig. 4a includes four graphs presenting results of the algorithms when performing synchronously, asynchronously, with message delays, and without. The versions of the algorithm presented are Max-sum, DMS with $\lambda = 0.9$, and DMS_SCFG. DMS_SCFG is the damped Max-sum (DMS) algorithm with split constraint factor graphs (SCFGs). We used the 0.4−0.6 version of DMS_SCFG, which was found to perform best by Cohen et al. [3].

Asy_Max-sum (with and without message delays) traversed solutions with higher costs on average compared to Syn_Max-sum. The results of the different runs of the algorithms were scattered and, thus, the differences from the synchronous versions were not found to be statistically significant. Asy_DMS, on the other hand, performed similarly to Syn_DMS, with and without message delays (as expected following Lemma 1, its corollary, and Proposition 2).

Another observation is that all versions of DMS_SCFG converged very fast compared to the other versions of the algorithm. Figure 4b provides a closer look that allows one to better compare their convergence rates. Both the synchronous and asynchronous versions converge at the same rate in environments that do not include message delays. Clearly, message delays affect the synchronous version more than the asynchronous version of the algorithm. Nevertheless, in all execution modes, the algorithm converges very fast to solutions with the same quality. The algorithm's fast convergence has been reported for the synchronous version [3]. The fact that the asynchronous version maintains the properties of the algorithm can be explained by Lemma 1, its corollary, and Proposition 2, that is, the damping of messages results in an effective GBCT of the asynchronous version that is similar to the effective BCT of the synchronous version.

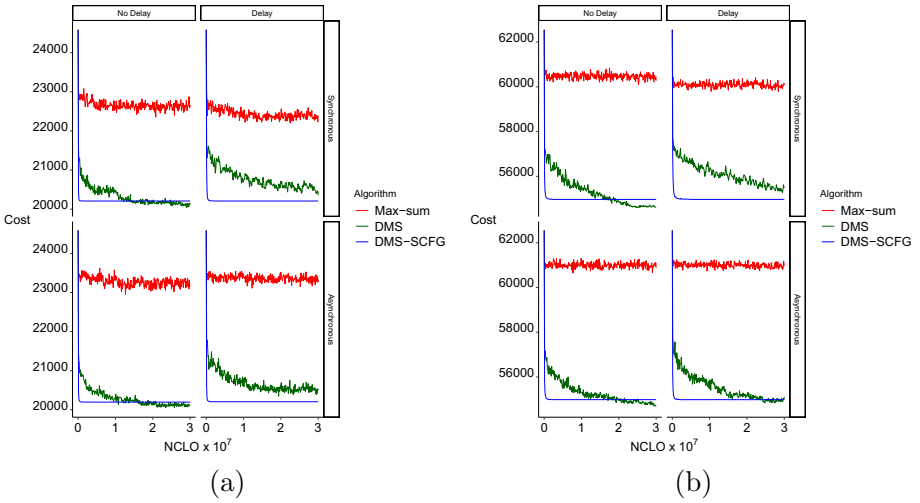

Fig. 6 Solution quality as a function of NCLOs of Max-sum versions, with and without message delays, solving **a** scale-free network problems and **b** overlapped solar system problems

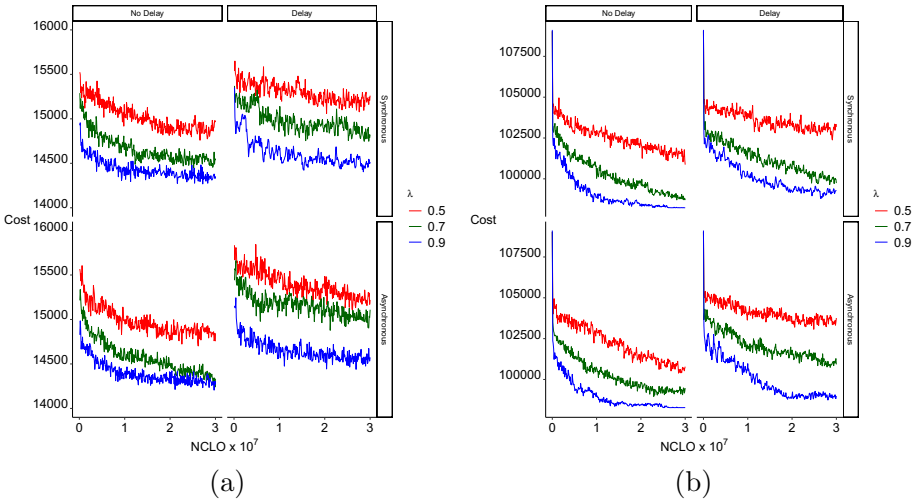

Fig. 7 Solution quality as a function of NCLOs of DMS with different λ values, with and without message delays, solving random graph problems with **a** $p_1 = 0.1$ and **b** $p_1 = 0.6$

Figure 5a presents results for the same algorithms solving dense random graph problems with density $p_1 = 0.6$. While the results seem similar to the results presented in Fig. 4a, there were smaller differences between the Max-sum versions. On these problems, the DMS versions in scenarios that did not include message delays found high quality (lower cost) solutions faster and converged.

Figure 5b presents the results of the algorithms solving graph coloring problems. It is apparent that the exploration performed by Max-sum and DMS is less effective on

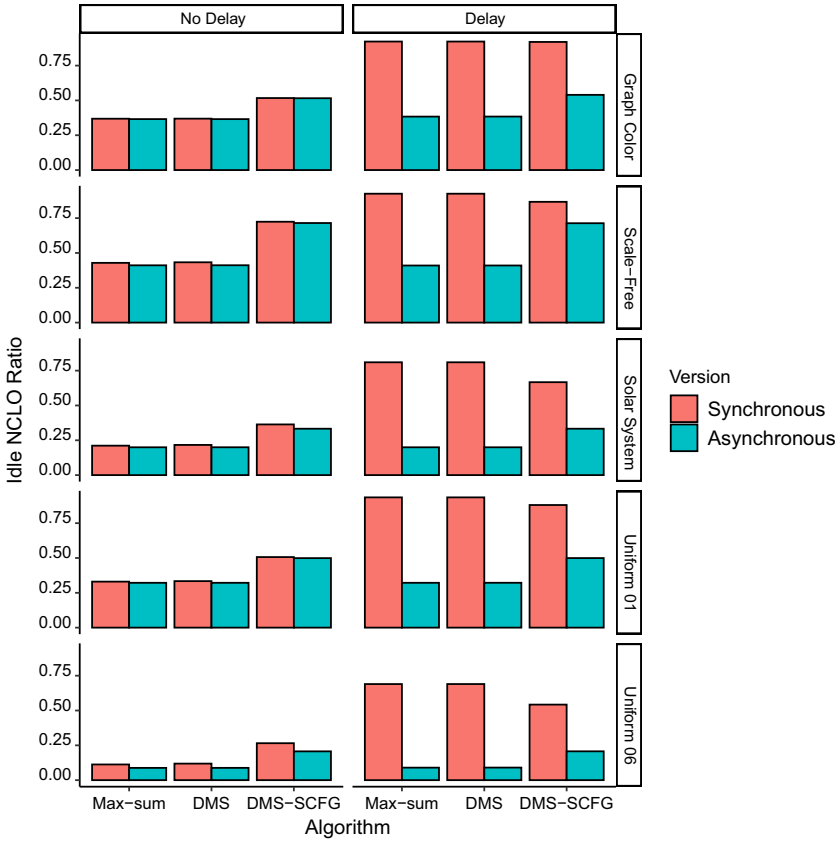

Fig. 8 Ratio between the number of NCLOs in which agents were idle and the total number of NCLOs for all algorithms and all execution modes

these problems and, thus, the advantage of DMS_SCFG is prominent. Moreover, in the presence of message delays, standard Max-sum improves its performance. We assume that delays break the very structured execution on this type of problems and has a positive exploration effect. This effect is diminished when damping is used, for reasons and properties similar to the ones established in Sect. 4.

The results of the algorithms when solving scale-free network problems and overlapping solar system problems are presented in Fig. 6. They are similar to the results presented in Fig. 5a. The differences between the performance of Asy_Max-sum and Syn_Max-sum were found to be significant when solving scale-free network problems, regardless of whether the scenarios solved included message delays. No significant differences were found between the synchronous and asynchronous versions when solving overlapped solar system problems. It seems that, for these problems, the structure of the problem affects the algorithms behavior more than the pattern of the message latency.

In our second set of experiments, we evaluated the importance of the selection of the damping factor in DMS, with respect to the differences in the performance of the different modes of execution (synchronous and asynchronous) in scenarios with different latency patterns. Figure 7 presents the results of the algorithm with three different values of the

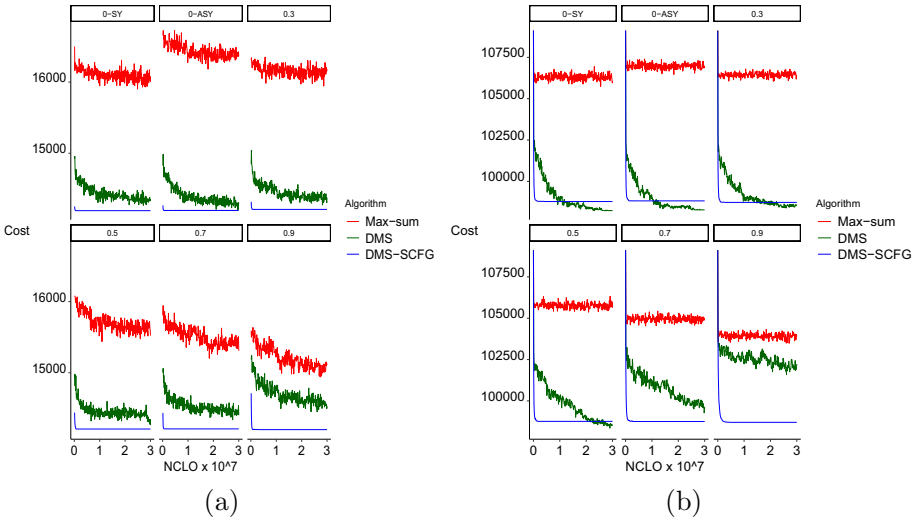

Fig. 9 Solution quality as a function of NCLOs of Max-sum versions, with and without message loss, solving **a** sparse random graph problems with $p_1 = 0.1$ and **b** dense problems $p_1 = 0.6$

damping parameter (i.e., $\lambda = 0.5$, $\lambda = 0.7$ and $\lambda = 0.9$) solving random uniform problems that are (a) sparse and (b) dense. As expected, following the properties established in Lemma 1 and its corollary, asynchronous execution affects the performance of all versions of DMS when it does not converge. However, it is apparent that the version with $\lambda = 0.9$ is less affected by message delays in the asynchronous execution (as expected). Similar results were obtained for all types of problems and were omitted to avoid redundancy.

In order to compare the effect that message delays have on the agents performing synchronously and asynchronously, we measured the average number of NCLOs in which agents were idle in each mode of execution of the algorithm. The results are presented in Fig. 8. It includes, for each algorithm, in each mode of execution, the average ratio of the number of NCLOs in which the agents were idle (i.e., waiting for messages to arrive) and the total number of NCLOs the algorithm executed. For all problem types, it is apparent that the agents spent less time idle when operating asynchronously compared to when they operate synchronously. This difference between the performance of the two versions was most apparent in DMS_SCFG. Nevertheless, for this version of the algorithm, while there is a difference in the time the agents spent idle, the quality of solutions was the most similar between the asynchronous and the synchronous versions among all algorithms, as well as their convergence times.

It is interesting to note that when the synchronous version of the algorithm is performing and messages are not delayed, there is still a significant portion of time that the agents spend idle. This seems to be the effect of having nodes of the factor graph with different number of neighbors. The amount of computation that agents perform in each iteration corresponds to this number, which also affects the number of function-nodes assigned to them. It is most apparent in problems where there is a large difference between the number of neighbors of different nodes in the graph (e.g., in scale-free network problems). In such problems, more idle time is reported. Specifically in the case of SCFGs, the number of neighbors is increased by the algorithm (following the initial split) and, thus, the difference

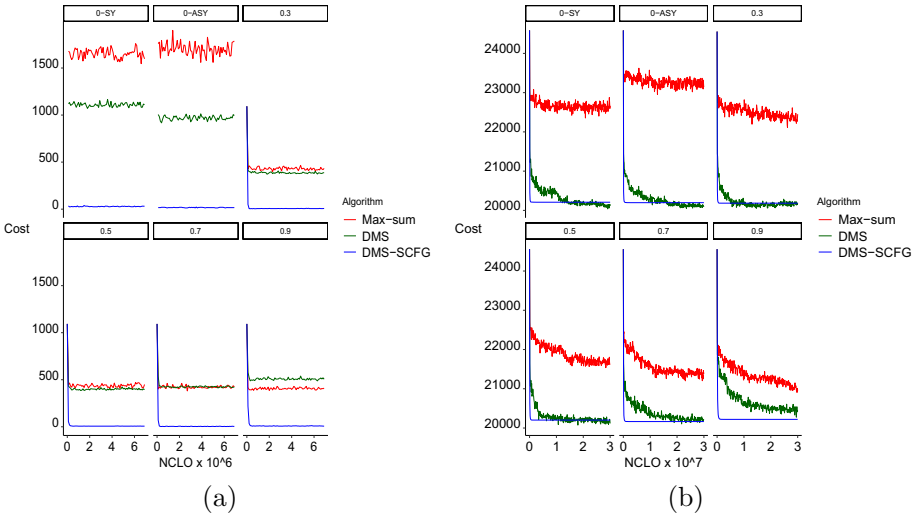

Fig. 10 Solution quality as a function of NCLOs of Max-sum versions, with and without message loss, solving **a** graph coloring problems and **b** scale-free network problems

between the computation performed by different agents grows and with it the time they spend idle.

5.3 Impact of message loss

In this subsection, we present results that demonstrate the resilience of the versions of Max-sum to message loss. Each experiment included the three versions of the algorithm (i.e., Max-sum, DMS, and DMS_SCFG (parameters set as in the previous section)) solving the same problems in synchronous execution, asynchronous execution, and asynchronous execution with different probability for message loss.

Figure 9a and b present the results for sparse random graph problems with density $p_1 = 0.1$ and dense random graph problems with density $p_1 = 0.6$, respectively. The results demonstrate that the largest differences between the performance of Max-sum and DMS are for the asynchronous version with no message loss. When the probability for message loss increases, the performance of Max-sum improves, while the performance of DMS deteriorates. For standard Max-sum, message loss slows the effect of the exponential explosion of the information sent in the bottom layers of the GBCT. DMS, on the other hand, suffers from message loss since as long as new messages from neighbors are not received, agents use in their calculation the last messages that were not received, while new messages that were received arrive instantly. Thus, there is a large chance for GBCTs with different heights (i.e., agents process information with different levels of damping). Finally, the performance of DMS_SCFG is consistent for all levels of message loss. This algorithm does not only produce the best results but it also shows high robustness to imperfect communication. On the dense problems, it is clear that the DMS version converges to better results than DMS_SCFG when the probability for message loss is low. For larger probabilities, as in the case of the sparse problems, DMS deteriorates.

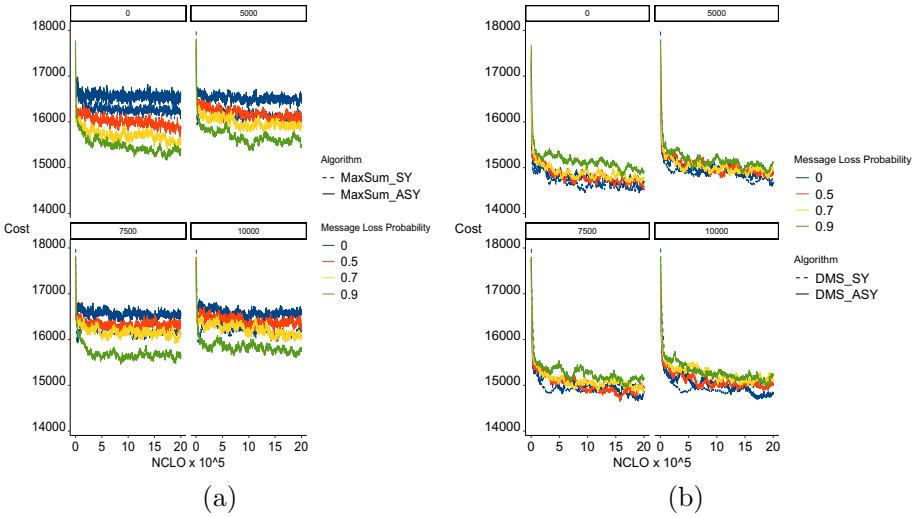

Fig. 11 Solution quality as a function of NCLOs of **a** Max-sum and **b** DMS, solving random sparse problems in environments with different communication patterns

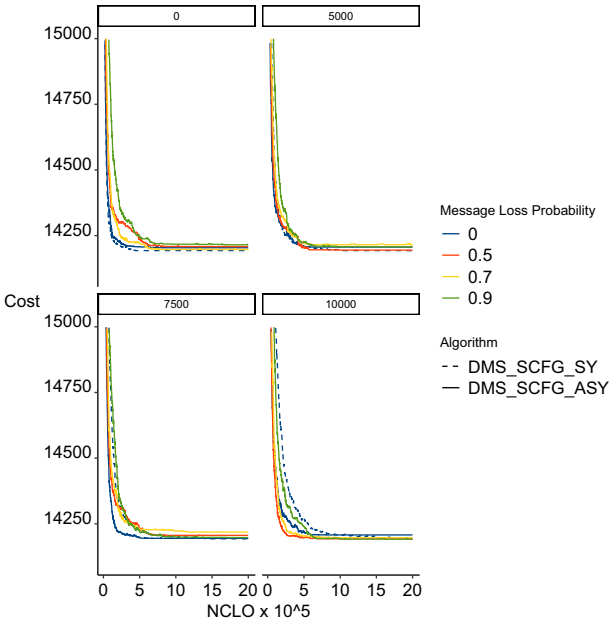

Fig. 12 Solution quality as a function of NCLOs of DMS_SCFG, for different solving sparse random graph problems with $p_1 = 0.1$ in environments with different communication patterns

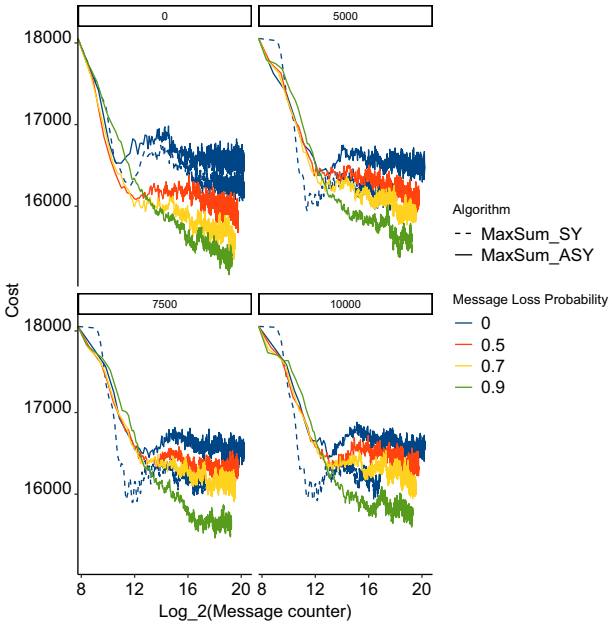

Fig. 13 Solution quality as function of the number of messages received by agents (logarithmic scale), of Max-sum solving sparse random graph problems with $p_1 = 0.1$, in environments with different communication patterns

Figure 10a and b present the results of the three versions of the algorithms when solving graph coloring problems and scale-free network problems, respectively, in environments in which there are different probabilities for message loss. It is clear that, in the case of graph coloring problems, the effect of message loss on both Max-sum and DMS is positive (in general). Except for the highest probability of message loss, on which DMS suffers some deterioration, both algorithms perform similarly when messages are lost. It is also apparent that they reach their best performance very fast and unlike the results on the other benchmarks, do not show further improvement or deterioration throughout the algorithm's run. On scale-free network problems on the other hand, the algorithms perform more similar to their performance on random uniform problems. However, the effect of message loss on DMS when solving these problems is less apparent.

The results of the algorithms on the solar system problems were similar to the results on scale-free network problems, and we omit them in order to avoid redundancy.

5.4 Impact of both message delay and loss

This section includes results of the three versions of the algorithm, solving problems in environments that include both message delay and possible message loss. Figure 11a and b present results for sparse uniform random problems solved by Max-sum and DMS, respectively. The different colored lines represent different probabilities for message loss, while each sub-graph represents a different upper bound for delays. Clearly, the magnitude of delays did not affect both algorithms, while the loss of messages had a reverse effect (as observed in the Fig. 9), improving the performance of Max-sum and deteriorating DMS's performance.

Figure 12 presents the results of DMS_SCFG solving sparse random problems in these mixed communication scenarios. Again, the robustness of this algorithm to imperfect

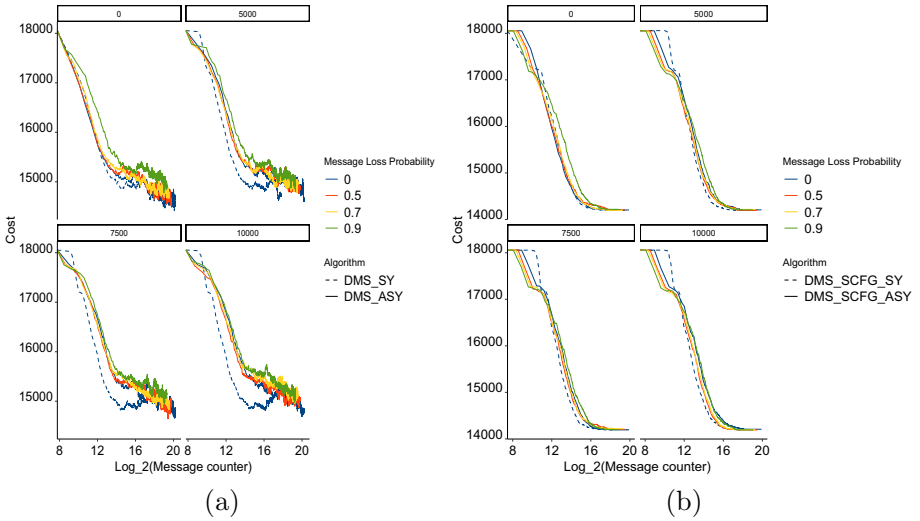

Fig. 14 Solution quality as function of the number of messages received by agents (logarithmic scale), of **a** DMS and **b** DMS_SCFG solving sparse random graph problems with $p_1 = 0.1$, in environments with different communication patterns

communication is apparent. The results for the algorithms solving the other problem types in mixed communication scenarios were similar, and we omit them to avoid redundancy.

In our last set of experiments, we evaluated the solution quality, as a function of the number of messages received by agents, regardless of the time the messages were delayed or the portion of messages that were lost. Figure 13 presents the results of Max-sum when solving sparse random uniform problems in scenarios with different communication patterns. It is clear from the presented graphs in the figure that message delays have a very minor effect on the performance of Max-sum. On the other hand, message loss has a major effect, and as we observed in the results presented above, a smaller probability for a message to arrive triggers higher quality.

Figure 14a presents the results of DMS in the same scenarios. In contrast to Max-sum, all versions of DMS produce solutions with similar quality when enough messages arrive. This is consistent with Lemma 1 and its corollary, in which we established the relationship between the quality of the solution of Asy_DMS and the structure of its effective GBCT. Figure 14b presents another indication for this property for DMS_SCFG. Once again we omit the similar results for the other benchmarks to avoid redundancy.

5.5 Discussion

The advantage of DMS over standard Max-sum when solving graphs with multiple cycles has been reported empirically [3] and explained theoretically [15]. In Max-sum, costs that are aggregated in the beginning of the run are duplicated in every node of the graph that has more than two neighbors and, thus, they are taken into consideration an exponential number of times in the calculation of beliefs and in the assignment selection. Damping reduces the weight of these costs in the belief calculation until it becomes negligible. A

similar phenomenon reduces the differences between the performance of Syn_DMS and Asy_DMS. As we established in the corollary of Lemma 1, when using a large enough damping factor, the effect of GBCTs with different heights is eliminated in DMS and, thus, after enough NCLOs are performed, the effective GBCTs of the beliefs in each message have the same number of nodes. The results comparing DMS with different damping factors demonstrate the need to use a large damping factor in order to achieve robustness to message delays. This empirical evidence strengthens the property established by Lemma 1 and its corollary, that if the damping factor used is not large enough, then the effect of the lower layers of the GBCTs, which may have different structure and a different number of nodes, on the generation of beliefs by the nodes is not eliminated. Thus, message delays have a greater effect on the algorithm's performance when the damping factor used is small.

When examining the algorithms in scenarios where there is a positive probability for message loss, there is an opposite effect on Asy_Max-sum and Asy_DMS. Message loss improves the performance of the former algorithm, but delays the convergence to a high quality solution of the latter algorithm, as we described above. Finally, Asy_DMS_SCFG maintains its fast convergence properties and high quality of solutions from its synchronous version. It is also robust to message latency and to message loss.

6 Conclusions

In this paper, we filled the gap in the Max-sum literature on the differences between the synchronous and asynchronous executions of the algorithm in distributed environments and their impact. Our theoretical analyses revealed that, unlike its synchronous counterpart, the asynchronous version of Max-sum in the presence of message latency can cause the propagation of inconsistent beliefs, resulting in the loss of guaranteed properties (Proposition 1). However, not all is lost as one can use damping to minimize this effect and, subsequently, ensure that when asynchronous DMS finds a consistent minimal route, it will converge, as does the synchronous version (Proposition 2). Our experimental results show that when the algorithm is further optimized through split constraint factor graphs, it converges very fast to high-quality solutions even in the presence of message delays and when most of the messages are lost. Moreover, our experimental results indicate that the quality of the solutions produced by the different versions of DMS depend on the amount of information (number of messages) received by the agents. These results are consistent with our theoretical results that indicate that enough information needs to be received in order for the effective GBCTs of the beliefs to be complete and, thus, similar.

Taken together, these results extend significantly our understanding of Max-sum in distributed environments with more realistic communication assumptions and enable a more effective use of Max-sum by real-world practitioners.

Author Contributions This paper is a result of a number of years of investigation of both the Max-sum algorithm and the performance of distributed algorithms in scenarios with imperfect communication. The idea to investigate the performance of distributed algorithms in such environments was suggested by William Yeoh and Roie Zivan, and this research is part of a BSF granted project that they are the two PIs of. Most of the writing of the paper was done by Roie Zivan. The experimental work was done by Ben Rachmut and Omer Perri. Ben Rachmut wrote most of the experimental section. William Yeoh reviewed the results and the writing, and suggested improvements.

Funding This research is partially supported by US-Israel Binational Science Foundation (BSF) grant #2018081 and US National Science Foundation (NSF) grant #1838364.

Availability of data and materials: The simulation's code is available at https://github.com/benrachmut/CADCOP_2022_new.

Declarations

Conflict of interest The authors declare no competing interests.

Ethical approval Not applicable.

References

1. Rachmut, B., Zivan, R., & Yeoh, W. (2021). Latency-aware local search for distributed constraint optimization. In: *Proceedings of the 20th international conference on autonomous agents and multiagent Systems*, pp. 1019–1027.
2. Chen, Z., Deng, Y., Wu, T., & He, Z. (2018). A class of iterative refined max-sum algorithms via non-consecutive value propagation strategies. *Autonomous Agents and Multi-Agent Systems, 32*(6), 822–860.
3. Cohen, L., Galiki, R., & Zivan, R. (2020). Governing convergence of max-sum on dcops through damping and splitting. *Artificial Intelligence Journal (AIJ)*, 279.
4. Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. San Francisco, California: Morgan Kaufmann.
5. Yanover, C., Meltzer, T., & Weiss, Y. (2006). Linear programming relaxations and belief propagation—An empirical study. *Journal of Machine Learning Research, 7*, 1887–1907.
6. Teacy, W.T.L., Farinelli, A., Grabham, N.J., Padhy, P., Rogers, A., & Jennings, N.R. (2008). Max-sum decentralized coordination for sensor systems. In: *Proceeding of the 7th international conference on autonomous agents and multi-agent systems (AAMAS)*, pp. 1697–1698.
7. Stranders, R., Farinelli, A., Rogers, A., Jennings, N. R. (2009). Decentralised coordination of mobile sensors using the max-sum algorithm. In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence, (IJCAI)*, pp. 299–304.
8. Ramchurn, S. D., Farinelli, A., Macarthur, K. S., & Jennings, N. R. (2010). Decentralized coordination in robocup rescue. *Computer Journal, 53*(9), 1447–1461.
9. Rust, P., Picard, G., & Ramparany, F. (2016). Using message-passing DCOP algorithms to solve energy-efficient smart environment configuration problems. In: *Proceedings of the 25th international joint conference on artificial intelligence, (IJCAI)*, pp. 468–474.
10. Farinelli, A., Rogers, A., Petcu, A., Jennings, N.R. (2008). Decentralised coordination of low-power embedded devices using the max-sum algorithm. In: *Proceeding of the 7th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pp. 639–646.
11. Zivan, R., Parash, T., Cohen, L., Peled, H., & Okamoto, S. (2017). Balancing exploration and exploitation in incomplete min/max-sum inference for distributed constraint optimization. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS), 31*(5), 1165–1207.
12. Ruozzi, N., & Tatikonda, S. (2013). Message-passing algorithms: Reparameterizations and splittings. *IEEE Transactions on Information Theory, 59*(9), 5860–5881.
13. Deng, Y., & An, B. (2020). Speeding up incomplete gdl-based algorithms for multi-agent optimization with dense local utilities. In: *Proceedings of the 29th international joint conference on artificial intelligence, (IJCAI)*, pp. 31–38.
14. Zivan, R., Perry, O., Rachmut, B., & Yeoh, W. (2021). The effect of asynchronous execution and message latency on max-sum. In: *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.
15. Zivan, R., Lev, O., & Galiki, R. (2020). Beyond trees: Analysis and convergence of belief propagation in graphs with multiple cycles. In: *Proceedings of the 34th international conference of the association for the advancement of artificial intelligence (AAAI)*, pp. 7333–7340.
16. Murphy, K.P., Weiss, Y., & Jordan, M.I. (1999). 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, pp. 467–475.

17. Weiss, Y. (2000). Correctness of local probability propagation in graphical models with loops. *Neural Computation, 12*(1), 1–41.
18. Forney, G.D., Kschischang, F.R., Marcus, B., & Tuncel, S. (2001). Iterative decoding of tail-biting trellises and connections with symbolic dynamics. In: Marcus, B., Rosenthal, J. (eds.) Codes, systems, and graphical models, pp. 239–264.
19. Pretti, M. (2005). A message-passing algorithm with damping. *Journal of Statistical Mechanics: Theory and Experiment, 11*, 11008.
20. Rogers, A., Farinelli, A., Stranders, R., & Jennings, N. R. (2011). Bounded approximate decentralized coordination via the max-sum algorithm. *Artificial Intelligence, 175*(2), 730–759.
21. Zivan, R., & Peled, H. (2012). Max/min-sum distributed constraint optimization through value propagation on an alternating DAG. In: AAMAS, pp. 265–272.
22. Rollon, E., & Larrosa, J. (2012). Improved bounded max-sum for distributed constraint optimization. In: CP, pp. 624–632.
23. Rollon, E., & Larrosa, J (2014) Decomposing utility functions in bounded max-sum for distributed constraint optimization. In: *Principles and practice of constraint programming—20th international conference, CP 2014, Lyon, France*, September 8–12, 2014. Proceedings, pp. 646–654.
24. Khan, M. M., Tran-Thanh, L., Ramchurn, S. D., & Jennings, N. R. (2018). Speeding up gdl-based message passing algorithms for large-scale dcops. *The Computer Journal, 61*(11), 1639–1666.
25. Zivan, R., Parash, T., & Naveh, Y. (2015). 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, pp. 432–439.
26. Farinelli, A., Rogers, A., & Jennings, N. R. (2014). Agent-based decentralised coordination for sensor networks using the max-sum algorithm. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS), 28*(3), 337–380.
27. Marinescu, R., & Dechter, R. (2009). AND/OR branch-and-bound search for combinatorial optimization in graphical models. *Artificial Intelligence, 173*(16–17), 1457–1491.
28. Nguyen, D. T., Yeoh, W., Lau, H. C., & Zivan, R. (2019). Distributed Gibbs: A linear-space sampling-based DCOP algorithm. *Journal of Artificial Intelligence Research, 64*, 705–748.
29. Petcu, A., & Faltings, B. (2005). A scalable method for multiagent constraint optimization. In: *Proceedings of the 21st international joint conference on artificial intelligence, (IJCAI)*, pp. 266–271.
30. Yeoh, W., Felner, A., & Koenig, S. (2010). BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. *Journal of Artificial Intelligence Research, 38*, 85–133.
31. Kschischang, F. R., Frey, B. J., & Loeliger, H. A. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory, 47*(2), 181–208.
32. Zivan, R., Parash, T., Cohen-Lavi, L., & Naveh, Y. (2020). Applying max-sum to asymmetric distributed constraint optimization problems. *Journal of Autonomous Agents and Multi Agent Systems (JAAMAS), 34*(1), 13.
33. Zivan, R., & Meisels, A. (2006). Message delay and discsp search algorithms. *Annals of Mathematics and Artificial Intelligence (AMAI), 46*, 415–439.
34. Netzer, A., Grubshtein, A., & Meisels, A. (2012). Concurrent forward bounding for distributed constraint optimization problems. *Artificial Intelligence Journal (AIJ), 193*, 186–216.
35. Cohen, E., Zivan, R., & Lev, O. (2023). Separate but equal: Equality in belief propagation for single cycle graphs. In: *Proceedings of the 36th international conference of the association for the advancement of artificial intelligence (AAAI)*.
36. Mayuga-Marcillo, L., Urquiza-Aguiar, L., & Paredes-Paredes, M. (2018). Wireless Channel 802.11 in NS-3
37. Amewuda, A.B., Katsriku, F.A., & Abdulai, J.-D. (2018). Implementation and evaluation of wlan 802.11ac for residential networks in ns-3. *Journal of Computer Networks and Communications*, 2018.
38. Zhang, W., Xing, Z., Wang, G., & Wittenburg, L. (2005). Distributed stochastic search and distributed breakout: properties, comparishon and applications to constraints optimization problems in sensor networks. *Artificial Intelligence, 161*(1–2), 55–88.
39. Barabási, A.-L., & Albert, R. (1999). Emergence of scaling in random networks. *Science, 286*(5439), 509–512.
40. Kiekintveld, C., Yin, Z., Kumar, A., & Tambe, M. (2010). Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In: *AAMAS*, pp. 133–140.

Authors and Affiliations

Roie Zivan[1] · Ben Rachmut[1] · Omer Perry[1] · William Yeoh[2]

✉ Roie Zivan
zivanr@bgu.ac.il

Ben Rachmut
rachmut@post.bgu.ac.il

Omer Perry
omerpe@post.bgu.ac.il

William Yeoh
wyeoh@wustl.edu

[1] Industrial Engineering and Management, Ben-Gurion University of the Negev, David Ben Gurion Blvd, 8410501 Beer-Sheva, Israel

[2] Computer Science and Engineering Department, Washington University in St. Louis, Brookings Drive, Saint Louis, MO 63130, USA