The Effect of Asynchronous Execution and Imperfect Communication on Max-sum Belief Propagation

Roie Zivan (zivanr@bgu.ac.il)
Ben-Gurion University of the Negev
Ben Rachmut
Ben-Gurion University of the Negev
Omer Perry
Ben-Gurion University of the Negev
William Yeoh
Washington University in St. Louis

Research Article

Keywords: Belief Propagation, Distributed constraints, Distributed problem solving

Posted Date: February 24th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-2611342/v1

Additional Declarations: No competing interests reported.

Version of Record: A version of this preprint was published at Autonomous Agents and Multi-Agent Systems on September 14th, 2023. See the published version at https://doi.org/10.1007/s10458-023-09621-w.

The Effect of Asynchronous Execution and Imperfect Communication on Max-sum Belief Propagation

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

[1]Industrial Engineering and Management, Ben-Gurion University of the Negev, David Ben Gurion Blvd, Beer-Sheva, 8410501, Israel.
[2]Computer Science and Engineering Department, Washington University in St. Louis, Brookings Drive, St. Louis, 63130, Missouri, United States.

Contributing authors: zivanr@bgu.ac.il; rachmut@post.bgu.ac.il; omerpe@post.bgu.ac.il; wyeoh@wustl.edu;

Abstract

Max-sum is a version of belief propagation that was adapted for solving distributed constraint optimization problems (DCOPs). 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 a synchronous and an asynchronous algorithm, however, neither the differences in the performance of these 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 the drastic effect that message latency has on distributed local search, damped Max-sum is robust to imperfect communication.

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 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 standard distributed 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 systems [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 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. We analyze the properties of the two execution versions of Max-sum, synchronous and asynchronous. More specifically, using backtrack cost trees [15], we investigate the possible differences between the propagated beliefs in synchronous and asynchronous executions of Max-sum.
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 the quality of solutions, even in asynchronous execution with message delays and when many messages are lost.

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

2 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, 16].

2.1 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 [17]. Our work extends the theory established by Weiss [18], which 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, 17, 19] 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 round [12]. Without loss of generality, we will focus on constraint optimization since it is more common in AI literature. 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 belief propagation, as do the results by Weiss [18].

2.2 Distributed Constraint Optimization Problems

Without loss of generality, in the rest of this paper, we will assume that all problems are minimization problems, as it is common in the DCOP literature [20]. Thus, we assume that all constraints define costs and not utilities.

A DCOP is defined by a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$. $\mathcal{A}$ is a finite set of agents $\{A_1, A_2, \ldots, A_n\}$. $\mathcal{X}$ is a finite set of variables $\{X_1, X_2, \ldots, X_m\}$. Each variable is held by a single agent, and an agent may hold more than one variable. $\mathcal{D}$ is a set of domains $\{D_1, D_2, \ldots, D_m\}$. Each domain D_i contains the finite set of values that can be assigned to variable X_i. We denote an assignment of value $x \in D_i$ to X_i by an ordered pair $\langle X_i, x \rangle$. $\mathcal{R}$ 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] For each binary constraint R_{ij}, there is a

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

corresponding cost table T_{ij} with dimensions $\mathsf{D\_i} \times \mathsf{D\_j}$ 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 [21, 22]. 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.

2.3 The Max-Sum Algorithm

Max-sum operates on a *factor graph*, which is a bipartite graph in which the nodes represent variables and constraints [23]. 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 $\mathsf{D\_X}$ 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) include vectors of zeros. A message sent from a variable-node X to a function-node F in iteration $i \geq 1$ is formalized as follows:

$$Q^i_{X \rightarrow F} = \sum_{F' \in F_X, F' \neq F} R^{i-1}_{F' \rightarrow X} - \alpha \tag{1}$$

where $Q^i_{X \rightarrow F}$ is the message variable-node X intends to send to function-node F in iteration i, F_X is the set of function-node neighbors of variable-node X, and $R^{i-1}_{F' \rightarrow X}$ is the message sent to variable-node X by function-node F' in iteration $i-1$. 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 $R^i_{F\to X}$ sent from a function-node F to a variable-node X in iteration i 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:

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

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 $(Q^{i-1}_{X'\to F})_{x'}$ is the cost that was received in the message sent from variable-node X' in iteration $i-1$, 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*), at each iteration t, an agent waits to receive all messages sent to it in iteration $t-1$ before performing computation and generating the messages to be sent in that 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.

2.3.1 Damped Max-Sum (DMS)

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, an agent performs calculations as in standard Max-sum. We use $\widehat{m^k_{i\to j}}$ to denote the result of the calculation made by agent A_i for the content of a message intended to be sent from A_i to agent A_j in iteration k and $m^{k-1}_{i\to j}$ to denote the message sent by A_i to A_j at iteration $k-1$. The message sent by A_i to A_j at iteration k is calculated as follows:

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

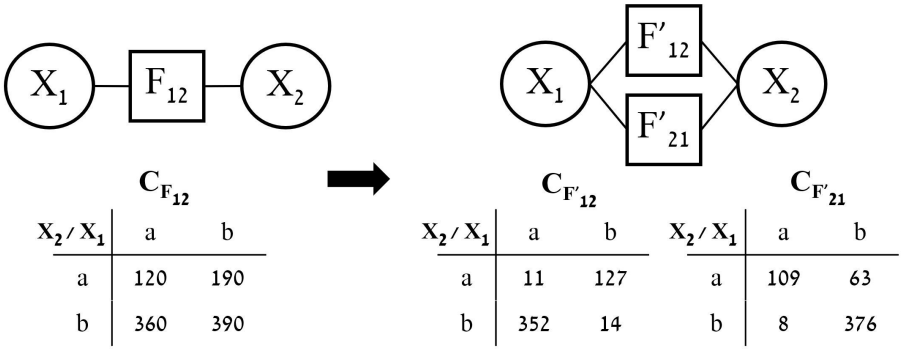


Fig. 1 An acyclic DCOP factor graph (on the left) and its equivalent SCFG (on the right).

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.

2.3.2 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, when it generates a message in some time t, is, for each neighbor N_j, the information included in the last message received from N_j (prior to t), regardless of when it was sent by N_j. If no message has been received from N_j yet, N_i uses a vector of zeros in its computation. Notice, that in the presence of message delays, a node N_i may receive messages from its neighbor N_j, 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 allow 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 without this method. Thus, we do not report these results in our empirical study.

2.3.3 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 constraint held by one of the involved agents. Each function-node is connected to both variable-nodes representing the variables involved in

the constraint [24]. 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). Researchers have used such *Split Constraint Factor Graphs* (SCFGs) 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].

2.3.4 Non-Concurrent Logic Operations

In order to evaluate the performance of distributed algorithms performing in a distributed environment, there is a need to establish which of the operations performed by agents could not have been performed concurrently and, thus, the runtime performance of the algorithm is the longest non-concurrent sequence of operations that the algorithm performed. As the basic logic operations of DisCSP algorithms are constraint checks, researchers have measured their runtimes in terms of *non-concurrent constraint checks* (NCCCs) [26]. To better compare different logic operations in other classes of algorithms, researchers generalized NCCCs to *non-concurrent logic operations* (NCLOs) [27]. We adopt NCLOs in this study.

2.4 Backtrack Cost Trees

For analyzing the behavior of Max-sum on factor graphs with an arbitrary (finite) number of cycles, Zivan *et al.* proposed the use of a *backtrack cost tree* (BCT) [15]. 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,

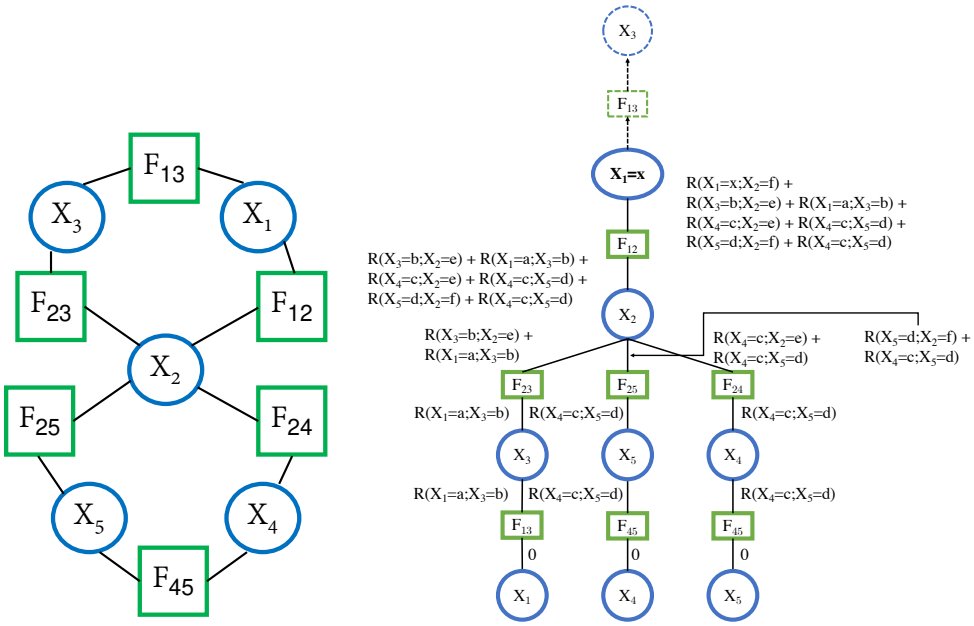


Fig. 2 (a) A lemniscate factor-graph. (b) An example of a BCT for a belief in the message sent from X_1 to the function-node F_{13} at time $t = 6$ in the lemniscate depicted on the left hand side.

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 A Backtrack Cost Tree (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 $BCT^t_{i=x\to j}$.

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 Figure 2(b)).

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 Figure 2(b)), which is the *belief* (cost) of assigning to X_i some value x from its domain D_i, as sent to a neighboring node. 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 Figure 2(a) is depicted in Figure 2(b).

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

2.5 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 [18, 28]. Moreover, when the algorithm does not converge, it periodically changes its set of assignments. In order to explain this behavior, Forney *et al.* 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 [28].

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.* 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 [15]. When a large enough damping factor is used, this minimal path is indeed a minimal path in the factor graph and, thus, if it is consistent, then the algorithm converges to an optimal solution.

3 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. 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 BCT 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 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 BCTs of the beliefs that are exchanged by agents depends on the arrival times of messages from which they are composed. Each BCT (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 BCT. 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. □

The proposition above seems to put an end to the natural wish that the convergence property 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 BCTs 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 BCT with depth k (distance from the root) and BCT_k denote the layers of the BCT with depth k or less. We say that a $layer_k$ is *effective* if and only if there exists a belief calculated using BCT_k that is different than the belief calculated when taking into consideration the complete BCT. For each BCT B, we say that its effective BCT B' is $BCT_{k'}$ 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',[3] 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,[4] 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 BCTs is effective, then the number of nodes in $layer_k$ of both BCTs are equal.*

Proof: A node in the BCT represents a node in the factor-graph, and its children are the nodes that sent messages to it that arrived. Assume that in two BCTs of beliefs sent in the same message, there exists an effective layer k in which one BCT has a smaller number of nodes that 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 BCTs will be equal until the end of the run of the algorithm. □

This will be true for all non-concurrent steps $ns > ns_0$ and, thus, layers of BCTs of beliefs that are sent in the same message with depth k following $ns \geq ns_0 + e\delta k$ (where δ is the maximal message delay, in terms of non-concurrent steps) must have the same number of nodes. The reason is the following: Damping with a large enough damping factor causes the bottom layers of BCTs to have less influence on the calculation made by the nodes in the algorithm following each computation step (see the work by Zivan *et al.* [15] for details). Let ϵ denote the smallest cost that can affect the nodes' actions in the algorithm. If we wait for a sufficiently large number of steps, the maximal sum of costs in the BCTs of steps performed before ns_0 will be smaller than ϵ. We use ns_1 to denote this sufficiently large enough number of steps. □

An immediate corollary from Lemma 1 is that in Asy_DMS (using a large enough damping factor), following ns_1, the effective BCTs 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*

[3] 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).

[4] 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].

in which the value 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 BCTs 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 BCTs 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 BCT_i^{ns} will be identical). Similarly, the addition of nodes to BCTs of beliefs corresponding to assignments that are not included in the minimal route represent costs that belong to routes with larger overall costs. □

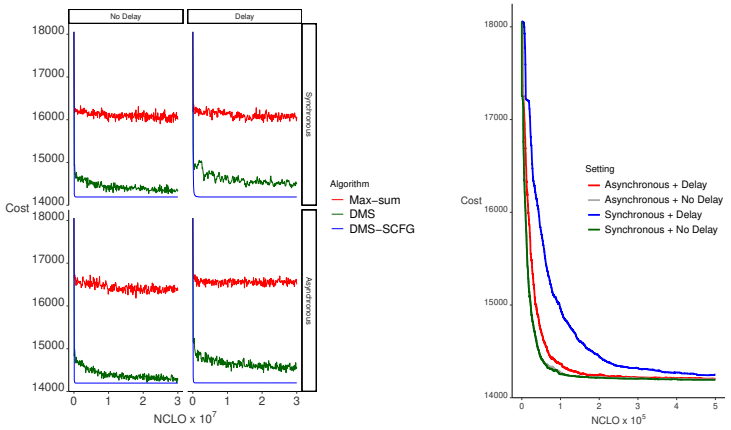
Proposition 2 has a major importance to our discussion. Both the asynchronous and the synchronous versions of DMS will converge when they reach a consistent minimal path. In other words, the differences between them can exist only when the minimal path is inconsistent. In such a case, the synchronous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

4 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 *et al.* [26]. Using this type of simulator allows us to implement any type of message delay pattern. Other simulators, such as ns-3 [29, 30], offer a number of communication patterns from which one can select. However, we prefer the use of the simulator proposed by Zivan *et al.* [26], 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 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. Both the constraint graph and the constraint functions are unstructured.
- **Graph Coloring Problems:** These problems are random constraint graph topologies in which each variable has three values (i.e., colors), 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 had three values (i.e., colors) in each domain [3, 10, 31].
- **Scale-free Network Problems:** Problems generated using the model by Barabási and Albert [32]. An initial set of 10 agents was randomly selected and connected. Additional agents were added sequentially and connected to 3 other agents with a probability proportional to the number of links that the existing agents already had. The cost of each joint assignment between constrained variables was independently drawn from the discrete uniform distribution from 100 to 199. 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.* [33].
- **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 [29, 30]. 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



(a) (b)

Fig. 3 (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.

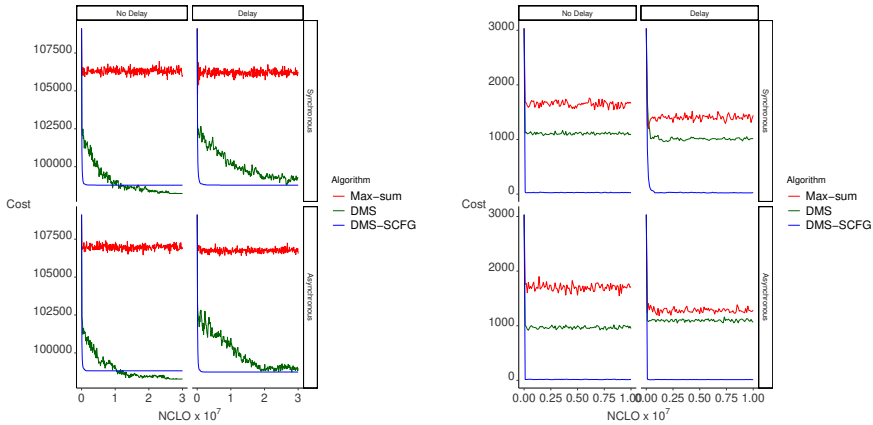
between agent A_i and the location of the farthest agent, and β expresses the dependency of the probability that both agents will be neighbors on their distance one from the other (in our experiments we used $\beta = 3$). For each pair of agents, a random probability $p_r \in [0, 1]$ was generated, and 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 with respect to physical distance as specified below in Section 4.1.

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 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.

4.1 Communication Scenarios

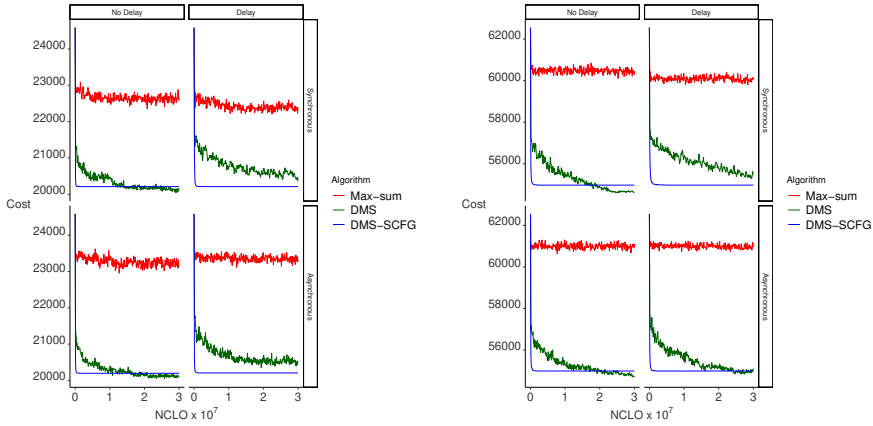
For random graph problems, graph coloring problems, and scale-free network problems, we used three types of communication scenarios: (1) Perfect communication; (2) Message latency selected from a uniform distribution $td_e \sim U(0, 10k)$ NCLOs; and (3) Message loss determined by $p \sim U(0, 1)$ such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

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



(a) (b)

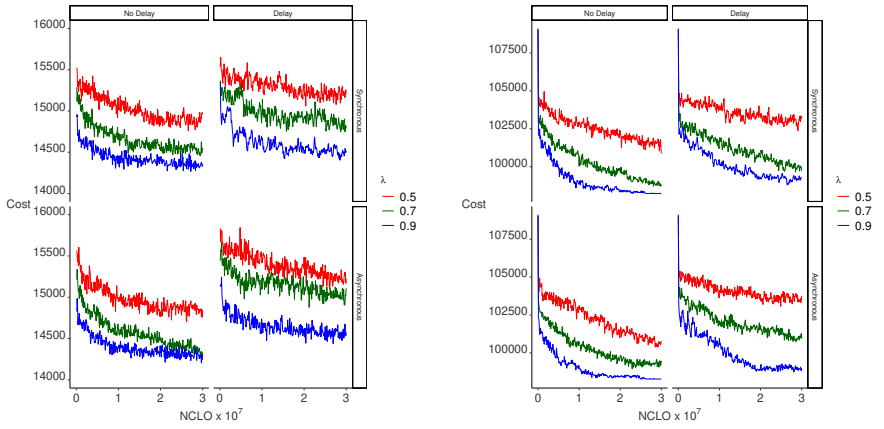
Fig. 4 Solution quality as a function of NCLOs of Max-sum versions, with and without message delays, solving (a) dense random graph problems with $p_1 = 0.6$ and (b) graph coloring problems.



(a) (b)

Fig. 5 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.

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 [29, 30]. 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.



(a) (b)

Fig. 6 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) $p_1 = 0.6$.

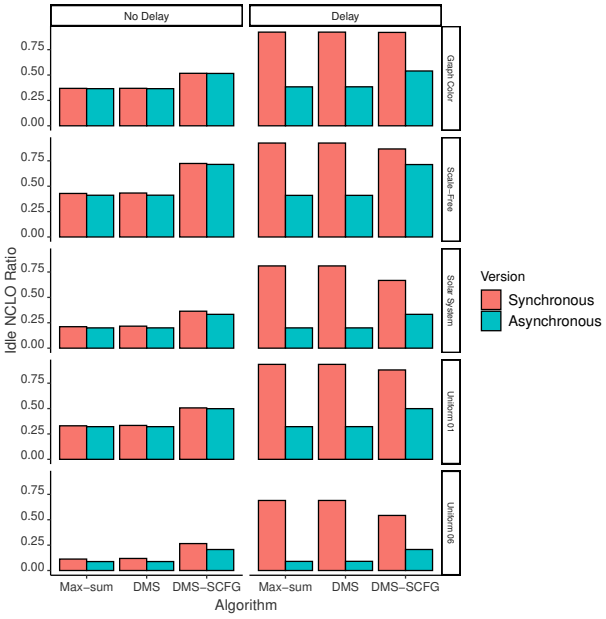


Fig. 7 Ratio between the number of NCLOs in which agents were idle and the total number of NCLOs for all algorithms and all execution modes.

4.2 Experimental Results on Impact of Message Delays

Figure 3(a) 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, Figure 3(a)

includes four graphs presenting results of the algorithms when performing synchronously, asynchronously, with message delays, and without. The versions of the algorithms include Max-sum, DMS with $\lambda = 0.9$, DMS-SCFG.[5]

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 Proposition 1).

Another observation is that all versions of DMS-SCFG converged very fast compared to the other versions of the algorithm. Figure 3(b) 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.

Figure 4(a) 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 Figure 3(a), 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 4(b) 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 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 Section 3.

The results of the algorithms when solving scale-free networks and overlapping solar system problems are presented in Figure 5. They are similar to the results presented in Figure 4(a). The differences between the performance of Asy_Max-sum and Syn_Max-sum were found to be significant when solving scale-free networks, 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

[5]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].

the performance of the different modes of execution (synchronous and asynchronous) in scenarios with different latency patterns. Figure 6 presents the results of the algorithm with three different values of the 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 Propositions 1 and 2, 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 Figure 7. 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, it did not affect the quality of the solutions; the quality of solutions was the most similar between the asynchronous and the synchronous versions among all algorithms, as well as their convergence times.

4.2.1 Experimental Results on 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 of message loss.

Figure 8 presents the results for sparse random graph problems with density $p_1 = 0.1$. The results demonstrate that the largest difference 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 BCT. DMS, on the other hand, suffers from message loss since relevant information is lost. The agents use in their calculation information that was supposed to be damped and, thus, the convergence to high quality solutions is delayed. 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.

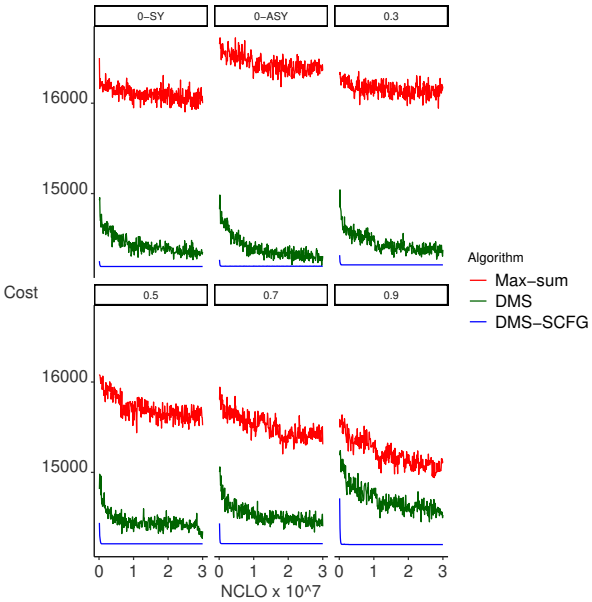


Fig. 8 Solution quality as a function of NCLOs of Max-sum versions, with and without message loss, solving sparse random graph problems with $p_1 = 0.1$.

The results of the algorithms on the other types of problems were similar and we omit them in order to avoid redundancy.

4.3 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 BCTs with different heights is eliminated in DMS and, thus, after enough NCLOs are performed, the effective BCTs 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 in not large enough, then the effect of the lower layers of the BCTs, 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.

5 Conclusions

In this paper, we filled the gap in the Max-sum literature on the difference of synchronous and asynchronous executions of the algorithm in distributed environments. 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 minimal route, it will converge, as does the synchronous version (Proposition 2). Finally, 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. Taken together, these results extend significantly our understanding of Max-sum in distributed environments with more realistic communication assumptions, propose algorithmic tools that are theoretically grounded to alleviate the issues raised, and enable a more effective use of Max-sum by real-world practitioners.

Declarations

- **Ethical approval:** Not applicable.
- **Competing interests:** Not applicable.
- **Authors' 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.

References

[1] Rachmut, B., Zivan, R., Yeoh, W.: Latency-aware local search for distributed constraint optimization. In: Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems, pp. 1019–1027 (2021)

[2] Chen, Z., Deng, Y., Wu, T., He, Z.: A class of iterative refined max-sum algorithms via non-consecutive value propagation strategies. Auton. Agents Multi Agent Syst. **32**(6), 822–860 (2018)

[3] Cohen, L., Galiki, R., Zivan, R.: Governing convergence of max-sum on dcops through damping and splitting. Artificial Intelligence Journal (AIJ) **279** (2020)

[4] Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Francisco, California (1988)

[5] Yanover, C., Meltzer, T., Weiss, Y.: Linear programming relaxations and belief propagation - an empirical study. Journal of Machine Learning Research **7**, 1887–1907 (2006)

[6] Teacy, W.T.L., Farinelli, A., Grabham, N.J., Padhy, P., Rogers, A., Jennings, N.R.: 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 (2008)

[7] Stranders, R., Farinelli, A., Rogers, A., Jennings, N.R.: 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 (2009)

[8] Ramchurn, S.D., Farinelli, A., Macarthur, K.S., Jennings, N.R.: Decentralized coordination in robocup rescue. Computer J. **53**(9), 1447–1461 (2010)

[9] Rust, P., Picard, G., Ramparany, F.: 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 (2016)

[10] Farinelli, A., Rogers, A., Petcu, A., Jennings, N.R.: 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 (2008)

[11] Zivan, R., Parash, T., Cohen, L., Peled, H., Okamoto, S.: 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 (2017)

[12] Ruozzi, N., Tatikonda, S.: Message-passing algorithms: Reparameterizations and splittings. IEEE Trans. Information Theory **59**(9), 5860–5881 (2013)

[13] Deng, Y., An, B.: 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 (2020)

[14] Zivan, R., Perry, O., Rachmut, B., Yeoh, W.: The effect of asynchronous execution and message latency on max-sum. In: 27th International Conference on Principles and Practice of Constraint Programming (CP 2021) (2021). Schloss Dagstuhl-Leibniz-Zentrum für Informatik

[15] Zivan, R., Lev, O., Galiki, R.: 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 (2020)

[16] Farinelli, A., Rogers, A., Jennings, N.R.: 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 (2014)

[17] Marinescu, R., Dechter, R.: AND/OR branch-and-bound search for combinatorial optimization in graphical models. Artif. Intell. **173**(16-17), 1457–1491 (2009)

[18] Weiss, Y.: Correctness of local probability propagation in graphical models with loops. Neural Computation **12**(1), 1–41 (2000)

[19] Nguyen, D.T., Yeoh, W., Lau, H.C., Zivan, R.: Distributed Gibbs: A linear-space sampling-based DCOP algorithm. Journal of Artificial Intelligence Resesrch **64**, 705–748 (2019)

[20] Modi, P.J., Shen, W., Tambe, M., Yokoo, M.: Adopt: asynchronous distributed constraints optimizationwith quality guarantees. Artificial Intelligence Journal (AIJ) **161:1-2**, 149–180 (2005)

[21] Petcu, A., Faltings, B.: A scalable method for multiagent constraint optimization. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence, (IJCAI), pp. 266–271 (2005)

[22] Yeoh, W., Felner, A., Koenig, S.: BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. Journal of Artificial Intelligence Research **38**, 85–133 (2010)

[23] Kschischang, F.R., Frey, B.J., Loeliger, H.A.: Factor graphs and the sum-product algorithm. IEEE Transactions on Information Theory **47:2**, 181–208 (2001)

[24] Zivan, R., Parash, T., Cohen-Lavi, L., Naveh, Y.: Applying max-sum to asymmetric distributed constraint optimization problems. Journal of Autonomous Agents and Multi Agent Systems (JAAMAS) **34**(1), 13 (2020)

[25] Zivan, R., Parash, T., Naveh, Y.: 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 (2015)

[26] Zivan, R., Meisels, A.: Message delay and discsp search algorithms. Annals of Mathematics and Artificial Intelligence (AMAI) **46**, 415–439 (2006)

[27] Netzer, A., Grubshtein, A., Meisels, A.: Concurrent forward bounding for distributed constraint optimization problems. Artificial Intelligence Journal (AIJ) **193**, 186–216 (2012)

[28] Forney, G.D., Kschischang, F.R., Marcus, B., Tuncel, S.: 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. Springer, ??? (2001)

[29] Mayuga-Marcillo, L., Urquiza-Aguiar, L., Paredes-Paredes, M.: Wireless Channel 802.11 in NS-3 (2018)

[30] Amewuda, A.B., Katsriku, F.A., Abdulai, J.-D.: Implementation and evaluation of wlan 802.11ac for residential networks in ns-3. Journal of Computer Networks and Communications **2018** (2018)

[31] Zhang, W., Xing, Z., Wang, G., Wittenburg, L.: Distributed stochastic search and distributed breakout: properties, comparishon and applications to constraints optimization problems in sensor networks. Artificial Intelligence **161:1-2**, 55–88 (2005)

[32] Barabási, A.-L., Albert, R.: Emergence of scaling in random networks.

Science **286**(5439), 509–512 (1999)

[33] Kiekintveld, C., Yin, Z., Kumar, A., Tambe, M.: Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In: AAMAS, pp. 133–140 (2010)