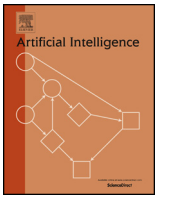
Artificial Intelligence 319 (2023) 103916

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



Privacy preserving solution of DCOPs by mediation

Pablo Kogan [a,c], Tamir Tassa [a], Tal Grinshpoun [b,c,*]

[a] Department of Mathematics and Computer Science, The Open University, Ra'anana, Israel
[b] Department of Industrial Engineering and Management, Ariel University, Ariel, Israel
[c] Ariel Cyber Innovation Center, Ariel University, Ariel, Israel

ARTICLE INFO

Article history:
Received 15 June 2022
Received in revised form 15 January 2023
Accepted 24 March 2023
Available online 29 March 2023

Keywords:
DCOP
Privacy
Mediation
Max-Sum
Multiparty computation
Collusion-secure

ABSTRACT

In this study, we propose a new paradigm for solving DCOPs, whereby the agents delegate the computational task to a set of external mediators who perform the computations for them in an oblivious manner. That is, the mediators perfectly simulate the operation of the chosen DCOP algorithm, but without getting access to the problem inputs or to its outputs. Specifically, we propose MD-Max-Sum, a mediated implementation of the Max-Sum algorithm. MD-Max-Sum offers topology, constraint, and decision privacy. Moreover, MD-Max-Sum is collusion-secure, as long as the set of mediators has an honest majority. We evaluate the performance of MD-Max-Sum on different benchmarks, problem sizes, and constraint densities. In particular, we compare its performance to PC-SyncBB, the only privacy-preserving DCOP algorithm to date that is collusion-secure, and show the significant advantages of MD-Max-Sum in terms of runtime. We conclude that MD-Max-Sum can be used in practice for solving DCOPs when strong privacy guarantees are required. The main takeaway from this study is a demonstration of the power of mediated computing. It allows either a single party or a set of parties, who may have limited computational or communication resources, to delegate an intricate computation to external dedicated servers who can perform the computation for them in an oblivious manner that protects the privacy of the initiating parties.

1. Introduction

A Distributed Constraint Optimization Problem (DCOP) [1,2] is a commonly accepted and practical mathematical framework for solving coordination challenges in multi-agent systems. DCOPs are well-suited to handle many real-world artificial intelligence problems, such as meeting scheduling [3], sensor networks [4], and management of energy resources [5].

A DCOP consists of a set of variables that are controlled by several independent agents. Each variable can take values in some finite domain. Some subsets of variables may be dependent in the sense that when they are assigned values from their respective domains, different combinations of those values may incur costs. The goal of the agents is to find assignments to their variables so that the sum of all costs that those assignments incur would be minimal.

Many algorithms were proposed over the years to solve DCOPs. Some of those algorithms are *complete*, in the sense that they always issue an optimal solution, namely, assignment to all variables with an overall minimal cost. Examples for such algorithms are SyncBB [1], ADOPT [6], OptAPO [7], and DPOP [8]. As DCOPs are NP-hard, complete algorithms are characterized by low scalability and they are usually limited to small-scale problems. In contrast, *incomplete* DCOP-solving

* Corresponding author.
E-mail address: talgr@ariel.ac.il (T. Grinshpoun).

https://doi.org/10.1016/j.artint.2023.103916
0004-3702/

algorithms, such as DSA [9], MGM [10], and Max-Sum [4], find solutions of typically low cost, but those solutions are not necessarily optimal. Such algorithms are usually much more efficient than complete algorithms and they can be applied to larger problems.

One of the main motivations for solving constraint optimization problems in a distributed manner is to preserve the privacy of the interacting agents. For example, when the agents represent parties that engage in coordinating event scheduling [3], each party might wish to keep its schedule details hidden from the other parties, since revealing such information might leak sensitive business data, e.g., meetings with competitors or with potential partners. Other examples of DCOP applications with privacy concerns include scheduling devices in smart homes [11,12], scheduling observations in satellite constellations [13,14], and assigning students to courses based on sensitive friendships information [15]. In Section 2 we survey some of the DCOP algorithms that preserve privacy.

All existing DCOP algorithms (privacy-preserving or not) are carried out by the agents themselves. However, some of those algorithms, and in particular the privacy-preserving ones, require significant computing resources. In addition, all DCOP algorithms assume that the agents are connected through a communication network. We propose here a new paradigm in DCOPs: solving them in what is known in cryptography as *the mediated model* [16,17]. Namely, We assume that there exist external servers, or *mediators*, that may serve as computing engines for the agents. The agents send the problem inputs to those mediators in some protected manner. The mediators simulate a DCOP algorithm on those inputs. At its completion, they send to the agents messages from which the agents extract the outputs, i.e., the variable assignments. Throughout this process, the mediators remain oblivious to the problem inputs, to the content of the protected messages that they exchange, and to the outputs.[1]

Performing the DCOP algorithm in such a mediated manner offers several significant advantages:

1. It protects the privacy of the agents since the agents no longer exchange among themselves messages that relate to their private data, while the mediators work on protected values (say, on secret sharing of the input values, or on homomorphic encryption of those values).
2. In some settings, the agents cannot efficiently communicate among themselves. This can happen in scenarios in which the agents are energy-constrained, such as static or mobile sensors [19,20], or in scenarios in which the agents do not even know with whom they are constrained as in the case with requesters of satellite observations [14]. In such cases, it is hard to run DCOP algorithms, because they require the agents to exchange messages. Some of those algorithms (like SyncBB) require all agents to communicate between themselves. The problem in Max-Sum is somewhat relaxed, as it requires only pairs of agents that are constrained to exchange messages. However, if two agents are constrained it does not imply that they can efficiently communicate. Delegating the computational task to a set of dedicated servers that are connected among themselves offers a remedy for this problem.
3. Agents who do not have the computational resources to run the DCOP algorithm may benefit from delegating the computation to external servers that will perform it for them in a secure manner. Examples include sensors [19,20], smart home devices [11,12], and students that want to register to courses [15].
4. While DCOP algorithms depend on having all agents active and cooperative, the mediated model is much more robust. Once all agents submit their inputs in a secure manner to the mediators, even if some of them experience a failure, the mediators can still complete the computation and provide the needed outputs to all agents that are still operational.

In this work we demonstrate the power of mediated computing by presenting MD-Max-Sum, a *Mediated* execution of the Max-Sum algorithm [4]. We use here *distributed* mediation, where the number of mediators, which we denote by L, is greater than 1. With such distributed mediation, the cryptographic protection shield (see point 1 above) is secret sharing (see Section 4.1).[2] MD-Max-Sum starts with the agents sending to the mediators secret shares in their private inputs. In the main part of the algorithm, the mediators execute the Max-Sum computations on the shares of the problem inputs, while remaining oblivious to the value of the underlying inputs. Finally, the mediators send the agents messages from which the agents may infer the assignments to their variables. We show that if the mediators have an honest majority (namely, the number of colluding mediators is smaller than the number of mediators outside the coalition), then MD-Max-Sum provides topology, constraint, and decision privacy. Those security guarantees hold against any collusion among the set of agents.

The outline of this work is as follows. In Section 2 we provide a summary of the previous work in the field of privacy-preserving DCOP algorithms. Then, Section 3 covers the relevant DCOP background – definitions and the Max-Sum algorithm. Later, in Section 4 we provide the necessary cryptographic background – threshold secret sharing and secure multiparty computation. In particular, we focus on efficient ways for performing secure comparisons, a significant building block of Md-Max-Sum. Section 5 holds the main part of this work – the description and analysis of MD-Max-Sum, our novel privacy-preserving collusion-secure DCOP algorithm. Section 6 provides an experimental evaluation of MD-Max-Sum and a comparison of its performance against Max-Sum and other privacy-preserving algorithms. We conclude in Section 7.

[1] Note that the external mediators herein should not be confused with the mediators in the OptAPO algorithm [7,18], which are regular (internal) agents that perform computations in their neighborhood as an inherent part of the algorithm.

[2] It is possible also to devise a mediated implementation of Max-Sum with a single mediator, $L = 1$. In such a case, the cryptographic protection shield would have to be homomorphic encryption, in order to allow the mediator to perform arithmetic computations on secret values. However, the costs of encryption are significantly higher than those of secret sharing.

A preliminary version of this paper was published at CSCML 2022 conference [21]. The present journal version extends the preliminary version with the inclusion of two algorithmic enhancements, formal proofs, extended experimental evaluation, as well as a much more elaborate presentation throughout the paper. The first algorithmic enhancement is in the computation of minima in Protocol 1, which provides perfect privacy (see the discussion in Sections 4.4 and 5.2.4). The second algorithmic enhancement is in the incorporation of a topology privacy index that allows us to balance between the level of topology privacy preservation and computational and communication costs (see Section 5.1.1). Other notable additions include elaborate introduction, related work, and background (Sections 1-4), formal proofs (Sections 5.1 and 5.3), computational and communication analysis (Section 5.4), and a much extended and comprehensive experimental evaluation (Section 6). In addition, we include in this version a demonstration of the privacy issues in Max-Sum (Appendix A) as well as the vulnerability of the privacy-preserving implementation of Max-Sum [22] to collusion attacks (Appendix B). We also include examples that illustrate the secret sharing of the private inputs (Appendix C) and the MPC computations over secret shared values (Appendix D).

2. Related work

The study of privacy in the context of Distributed Constraint Satisfaction Problems (DCSPs) started with the work of Silaghi and Faltings [23], who compared different solution techniques and arranged them in a hierarchy according to their corresponding level of privacy loss. DCSPs are a special case of DCOPs: while in DCOPs, combinations of assignments incur a cost in the range $[0, \infty]$, in DCSPs all costs are in $\{0, \infty\}$. Namely, a combination of assignments is either allowed or prohibited, and the goal is to find an assignment to all variables with no prohibited combinations.

In a later study, Silaghi and Mitra [24] proposed a privacy-minded solution to Distributed Weighted Constraint Satisfaction Problems, another framework that is closely related to DCOPs. Their solution was based on the BGW protocol for a multiparty secure evaluation of polynomial functions [25], and it was applicable only to small problems due to its dependency on an exhaustive search over all possible full assignments.

Doshi et al. [26] and Savaux et al. [27] considered the idea of injecting privacy loss as a criterion for the solving process. These studies relate to the trade-off between solution quality and privacy loss, whereas our focus is on preventing privacy loss via strong privacy guarantees.

In order to better describe the privacy guarantees of DCOP algorithms, Léauté and Faltings [28] suggested four notions of privacy for the DCOP framework: agent, topology, constraint, and decision privacy. We adhere to those notions and provide their definition in Section 3.1. Furthermore, they devised three secure versions of DPOP, each with different runtimes and privacy guarantees: P-DPOP$^{(+)}$, P$^{3/2}$-DPOP$^{(+)}$, and P^{2}-DPOP$^{(+)}$.

Grinshpoun and Tassa [29] presented a privacy-preserving version of another complete algorithm – SyncBB. The resulting method, P-SyncBB, preserves topology, constraint, and decision privacy.

Since complete algorithms are not scalable, research efforts were invested also in designing privacy-preserving implementations of incomplete algorithms. A notable example is the GDL-based [30] Max-Sum algorithm [4]. Unlike search-based algorithms, where the agents systematically search the entire solution space, GDL-based algorithms require the agents to maintain a set of beliefs. As the protocol progresses, the agents communicate and update those beliefs and ultimately choose their assignment according to them. Such schemes are also known as *inference-based* algorithms. In the context of DCOPs, Max-Sum was shown to produce high-quality solutions while keeping runtimes low in comparison with both incomplete or complete algorithms.

Tassa et al. [22] developed the P-Max-Sum algorithm, which runs Max-Sum with cryptographic enhancements in order to preserve privacy (see Appendix A where we demonstrate how Max-Sum fails to preserve privacy). In P-Max-Sum, every message between two nodes is *secret-shared* between the two agents that control those nodes. The main task is then to use the set of message shares in one iteration in order to compute from them proper shares in the messages of the next iteration. Homomorphic encryption is used in order to perform that computation in a privacy-preserving manner. P-Max-Sum provides topology, constraint, and decision privacy, and it may be extended to provide also agent privacy. Grinshpoun et al. [31] devised another incomplete privacy-preserving algorithm, P-RODA, which is based on region optimality [32,33]. P-RODA provides constraint privacy and partial decision privacy.

All of the above-mentioned privacy-preserving DCOP algorithms assume *solitary* conduct of the agents. However, if two or more agents collude and combine the information that they have, they may extract valuable information about other agents. (In Appendix B we demonstrate such possible attacks in the context of P-Max-Sum). To address the risk of collusion, Tassa et al. [34] introduced PC-SyncBB, the first privacy-preserving DCOP algorithm that is *collusion-secure*. It is secure under the assumption that the agents have an honest majority. PC-SyncBB does extensive usage of Secure Multiparty Computation (MPC) in order to obliviously compare between costs of partial assignments.

In this work we propose MD-Max-Sum, the first *incomplete* privacy-preserving DCOP algorithm that is *collusion-secure*. MD-Max-Sum is also the first DCOP algorithm that is implemented in the mediated model. We note that one of the variants of PC-SyncBB [34] suggests exporting some computation to an external committee of mediators. That is, PC-SyncBB is executed entirely by the agents but there is one specific computation (the comparison between the cost of the current partial assignment to the cost of the best full assignment that was found so far in the search) that could be exported to external mediators. In contrast, MD-Max-Sum is the first algorithm that is fully mediated, in the sense that it is executed

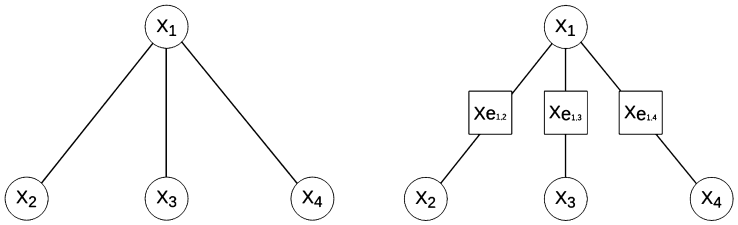


Fig. 1. A constraint graph G of a DCOP with 4 variable nodes (left) and the corresponding factor graph G' that has 4 variable nodes and 3 function nodes (right).

entirely by external mediators. The advantages of that model of computation are discussed in the Introduction and in Section 7.

3. Background: distributed constraint optimization problems

In this section, we present the necessary DCOP background: definitions of the DCOP framework (Section 3.1) and the Max-Sum algorithm (Section 3.2).

3.1. DCOP definitions

A Distributed Constraint Optimization Problem (DCOP) [1,2] is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ where $\mathcal{A}$ is a set of agents $A_1, A_2, \ldots, A_N$, $\mathcal{X}$ is a set of variables $X_1, X_2, \ldots, X_N$, $\mathcal{D}$ is a set of finite domains $D_1, D_2, \ldots, D_N$, and $\mathcal{R}$ is a set of relations (constraints). Each variable X_n, $n \in [N] := \{1, 2, \ldots, N\}$, takes values in the domain D_n, and it is held by the agent A_n.[3] Each constraint $C \in \mathcal{R}$ defines a non-negative cost for every possible value combination of some subset of variables and is of the form $C : D_{n_1} \times \cdots \times D_{n_k} \to [0, q]$, for some $1 \leq n_1 < \cdots < n_k \leq N$, and a publicly known maximal constraint cost q.

An *assignment* is a pair including a variable and a value from that variable's domain. The goal of the agents is to find assignments to their variables so that the sum of all costs that those assignments incur would be minimal.

We consider here a binary version of DCOPs, in which every $C \in \mathcal{R}$ constraints exactly two variables and takes the form $C_{n,m} : D_n \times D_m \to [0, q]$, where $1 \leq n < m \leq N$. Such an assumption is customary in DCOP literature, see e.g. [6,8]. As the domains are finite, they may be ordered. Hence, the binary constraint $C_{n,m}$ between X_n and X_m may be described by a matrix, which we also denote by $C_{n,m}$, of dimensions $|D_n| \times |D_m|$; in that matrix, $C_{n,m}(i, j)$ equals the cost that corresponds to the assignment of the ith value in D_n to X_n and the jth value in D_m to X_m, where $1 \leq i \leq |D_n|$ and $1 \leq j \leq |D_m|$.

The constraint graph $G = (V, E)$ is an undirected graph over the set of variables $V = \mathcal{X}$, where an edge in E connects two variables if and only if they are constrained. If we define for every pair of variables $(X_n, X_m) \notin E$ a constraint matrix $C_{n,m}$ which is the zero matrix of dimensions $|D_n| \times |D_m|$, then the set of all such matrices,

$$\mathcal{C} := \{C_{n,m} : 1 \leq n < m \leq N\}, \tag{1}$$

fully determines the problem; namely, it encompasses all topology and constraint information.

Every DCOP is also associated with a so-called *factor graph*. It is a bipartite graph $G' = (V', E')$ that is defined as follows.

- V' has two types of nodes: *variable nodes*, $X_1, \ldots, X_N$, and *function nodes*, X_e, for each $e = (X_n, X_m) \in E$.
- E' has an edge connecting X_n with X_e if and only if e is an edge in G that is adjacent to X_n.

An example of a DCOP constraint graph and its corresponding factor graph is given in Fig. 1.

Léauté and Faltings [28] have distinguished between four notions of privacy in the context of distributed constraints:

- *Agent privacy* – hiding from each agent the identity or even the existence of other agents with whom it is not constrained.
- *Topology privacy* – hiding from each agent the topological structures in the constraint graph beyond its own direct neighborhood in the graph.
- *Constraint privacy* – hiding from each agent the constraints in which it is not involved. Namely, agent A_k should not know anything about $C_{n,m}(\cdot, \cdot)$ if $k \notin \{n, m\}$.
- *Decision privacy* – hiding from each agent the final assignments to other variables (i.e., variables owned by other agents).

[3] We make herein the standard assumption that the number of variables equals the number of agents, and that each variable is held by a distinct single agent, see e.g. [6,8].

3.2. The Max-Sum algorithm

The Max-Sum algorithm [4] operates on the factor graph G'. Each agent A_n, $n \in [N]$, controls its corresponding variable node X_n. As for the function nodes, they are controlled by either of the two agents corresponding to the adjacent variable nodes; the decision of which agent controls each function node is made a-priori. The Max-Sum algorithm performs synchronous steps (iterations), where in each of them a couple of messages are sent along each edge of G' in both directions. Let us consider the edge that connects X_n with X_e, where $e = (X_n, X_m)$. The messages, in both directions, will be vectors of dimension $|D_n|$, and they will be denoted by $Q^k_{n\to e}$ and $R^k_{e\to n}$, depending on the direction, where $k \geq 0$ is the number of the iteration. If x is one of the elements in D_n then its corresponding entry in the message will be denoted by $Q^k_{n\to e}(x)$ or $R^k_{e\to n}(x)$.

In the $k = 0$ iteration, all messages are zero. After completing the kth iteration, the messages in the next iteration will be as follows. Fixing a variable node X_n and letting V_n be the set of function nodes adjacent to X_n in G', then for each $X_e \in V_n$, X_n will send to X_e the vector

$$Q^{k+1}_{n\to e} := \sum_{X_f \in V_n \setminus \{X_e\}} R^k_{f\to n}\,. \tag{2}$$

As for messages sent from function nodes, if X_e is a function node that connects the two variable nodes X_n and X_m then the message sent from X_e to X_n is

$$R^{k+1}_{e\to n}(x) := \min_{y\in D_m}\left[C_{n,m}(x,y) + Q^k_{m\to e}(y)\right], \quad \forall x \in D_n\,; \tag{3}$$

the message that X_e sends to X_m is constructed similarly. Finally, after completing a preset number K of iterations, each variable node X_n computes

$$\overline{R}_n := \sum_{X_e \in V_n} R^K_{e\to n}\,, \tag{4}$$

and then selects a value $x \in D_n$ for which $\overline{R}_n(x)$ is minimal.

During the run of Max-Sum, the entries in the messages Q^k and R^k may grow exponentially. In order to prevent the entries in the messages from growing uncontrollably, it is customary to "normalize" the messages. One manner in which messages are normalized in Max-Sum is to subtract from each entry in each message $Q^{k+1}_{n\to e}$, where $e = (X_n, X_m)$, the value $\alpha^{k+1}_{n\to e} := \min_{x\in D_n} Q^{k+1}_{n\to e}(x)$ [4].

4. Background: cryptographic tools

In this section, we describe the cryptographic tools and protocols that we will use in our privacy-preserving DCOP algorithm. We begin with two general-purpose tools: threshold secret sharing (Section 4.1) and secure multiparty computation (Section 4.2). We then describe efficient ways for performing a secure comparison of secret-shared values (Section 4.3) and how to securely compute their minimum (Section 4.4).

4.1. Shamir's secret sharing

Secret sharing schemes [35] are protocols that enable to distribute a secret among a group of parties, denoted $\mathbf{M} = \{M_1, \ldots, M_L\}$, such that each of them is allocated a random value, called *a share*, so that some subsets of those shares enable the reconstruction of the secret. In its most basic form, called *Threshold Secret Sharing*, the secret can be reconstructed only when a sufficient number of shares are combined together, while smaller sets of shares reveal no information at all on the secret.

Shamir's t-out-of-L threshold secret sharing scheme [35], for some $t \leq L$, is a scheme in which the secret shares enable the recovery of the secret from any subset of t shares, while any subset of $t-1$ or fewer shares reveals nothing on the secret. The scheme operates over a finite field $\mathbb{Z}_p$, where $p > L$ is a prime sufficiently large so that all possible secrets may be represented in $\mathbb{Z}_p$. It has two procedures: Share and Reconstruct:

- $\mathsf{Share}_{t,L}(x)$. The procedure samples a uniformly random polynomial $f(\cdot)$ over $\mathbb{Z}_p$, of degree at most $t-1$, where the free coefficient is the secret s. That is, $f(x) = s + a_1 x + a_2 x^2 + \ldots + a_{t-1}x^{t-1}$, where a_j, $1 \leq j \leq t-1$, are selected independently and uniformly at random from $\mathbb{Z}_p$. The procedure outputs L values, $s_\ell = f(\ell)$, $\ell \in [L] := \{1, \ldots, L\}$, where s_ℓ is the share given to M_ℓ, $\ell \in [L]$. The entire set $\{s_1, \ldots, s_L\}$ is called a t-sharing of s, and will be denoted henceforth by $[s]_t$.
- $\mathsf{Reconstruct}_t(s_1, \ldots, s_L)$. The procedure is given any selection of t shares out of $[s]_t$. Then it interpolates a polynomial $f(\cdot)$ of degree at most $t-1$ using the given points and outputs $s = f(0)$. Any selection of t shares out of $[s]_t$ will yield the secret s, as t points determine a unique polynomial of degree at most $t-1$. On the other hand, any selection of $t-1$ shares or less reveals nothing about the secret s.

The operations of generating random shares in a secret and reconstructing a secret from its shares are demonstrated in Appendix C and Appendix D.

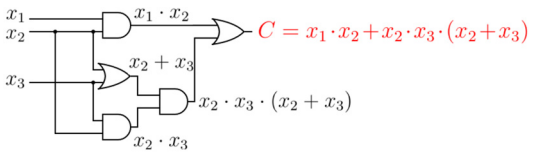


Fig. 2. An arithmetic circuit C that realizes the function $f(x_1, x_2, x_3) = x_1 \cdot x_2 + x_2 \cdot x_3 \cdot (x_2 + x_3)$.

4.2. Secure multiparty computation

A Secure Multiparty Computation (MPC) protocol [36,37] allows a group of parties, denoted $\mathbf{M} = \{M_1, \ldots, M_L\}$, to compute any function f over private inputs that they hold, $x_1, \ldots, x_L$, where x_ℓ is known only to M_ℓ, $\ell \in [L]$, so that at the end of the protocol, everyone learns the result of $f(x_1, \ldots, x_L)$, but nothing else beyond what every party may infer from the final output and its own input.

It is customary to distinguish between *semi-honest* and *malicious* parties. Semi-honest parties follow the prescribed MPC protocol but at the same time, they try to glean more information than allowed from what they receive during the execution of the protocol. In contrast, malicious parties may deviate from the prescribed protocol, or even defect during the protocol. Designing protocols that are secure against malicious parties is a significantly more intricate task and the resulting protocols are usually much less efficient. We concentrate here on the case of semi-honest parties.

It should be noted that while semi-honest parties are trusted to follow the protocol, some of them may collude in order to combine their inputs and messages received during the protocol's execution in order to extract private information on other parties. A common assumption in studies that consider honest parties is that of an *honest majority*. It means that if some of the parties collude, the number of colluding parties is strictly smaller than half the overall number of parties. We make that assumption herein.

4.2.1. Circuit representation

MPC protocols require the function f to be represented by a circuit C such that for every set of inputs, $x_1, \ldots, x_L$, the output of the circuit, $C(x_1, \ldots, x_L)$, equals $f(x_1, \ldots, x_L)$. A circuit representation of a function f is essentially a directed acyclic graph with the following properties. The graph has a leaf node (i.e., a node with in-degree zero) for every input of f, and a root node (i.e., a node with out-degree zero) for the output of f. The former nodes are called input gates, while the latter one is called an output gate. In addition, the graph may have multiple internal nodes (ones with positive in-degrees and out-degrees) that are called operation gates.

We restrict our attention to arithmetic circuits, in which the values assigned to gates are from an arbitrary finite field $\mathbb{F}$, and the operation gates are either the addition or the multiplication functions (over two operands). For illustration, consider the arithmetic function $f(x_1, x_2, x_3) = x_1 \cdot x_2 + x_2 \cdot x_3 \cdot (x_2 + x_3)$. The circuit C in Fig. 2 evaluates that function. It consists of three layers. The first layer has two multiplication gates that compute $x_1 \cdot x_2$ and $x_2 \cdot x_3$, and one addition gate that computes $x_2 + x_3$. The second layer has a single multiplication gate for computing $x_2 \cdot x_3 \cdot (x_2 + x_3)$. Finally, the third and last layer has a single addition gate that issues the desired output.

The private values $x_1, \ldots, x_L$ determine the input values to all of the circuit's input gates. Then, the following process is performed repeatedly: for each gate g, once both its input wires are assigned values, say α_1 and α_2, the output wire of the gate is assigned the value $g(\alpha_l, \alpha_r)$. This process is repeated until the output gate is assigned a value. That output value is denoted $C(x_1, \ldots, x_L)$.

4.2.2. Secret-sharing-based MPC

Here we describe a general approach to performing an MPC evaluation of arithmetic circuits based on Shamir's secret sharing scheme (Section 4.1). Assume that $C(x_1, \ldots, x_h)$ is an arithmetic circuit that realizes some polynomial function $f(x_1, \ldots, x_h)$. Assume next that each of the inputs x_i, $i \in [h]$, is secret-shared using t-out-of-L secret sharing among a set of L parties, $M_1, \ldots, M_L$. The goal is to design an MPC protocol that will allow the L parties to compute t-out-of-L sharing of the output value $f(x_1, \ldots, x_h)$.

Since the L parties already have t-out-of-L shares in each of the input values, then it is necessary to devise sub-protocols that will allow them to emulate arithmetic gates. Namely, assuming that the parties hold t-out-of-L shares in the two inputs of a gate, it is necessary to describe a manner in which they will be able to compute t-out-of-L shares in the gate's output, without learning in the process any information on the underlying inputs or output. Clearly, if we can emulate addition gates and multiplication gates, then it would be possible to emulate the entire circuit, gate by gate, until the parties get t-out-of-L shares in the final output.

In the circuit shown in Fig. 2, the parties have t-out-of-L shares in each of the input values x_1, x_2, x_3. With those shares, they proceed to emulate the circuit layer by layer, by computing proper t-out-of-L shares in the output wires of the three gates in the first layer, then proceeding to compute t-out-of-L shares in the output of the multiplication gate in the second layer, using the already computed shares in the output wires of two of the gates in the first layer, and finally computing t-out-of-L shares in the output wire using the computed shares in the output wires of the multiplication gate in the second layer and the first multiplication gate in the first layer.

Table 1
Truth table for an indirect comparison of a and b using Eq. (6).

$w = 1_{a<\frac{p}{2}}$	$x = 1_{b<\frac{p}{2}}$	$y = 1_{[(a-b) \mod p]<\frac{p}{2}}$	$z = 1_{a<b}$
T	F	*	T
F	T	*	F
F	F	F	T
F	F	T	F
T	T	F	T
T	T	T	F

Below, we describe the generic secret-sharing-based protocol of Damgård and Nielsen [38]. Specifically, we explain the manner in which that protocol emulates addition and multiplication gates. As the multiplication procedure requires the parties to generate shares in an unknown random value, we also explain how they perform that. We would like to note that in our experiments we used that protocol, with the performance improvements that were proposed by Chida et al. [39].

Hereinafter, we set the secret sharing threshold to be

$$t := \lfloor (L+1)/2 \rfloor . \tag{5}$$

Namely, in order to reconstruct the secret, at least half of the parties must combine their shares. We will explain the importance of that setting later on when we discuss the privacy of MD-Max-Sum.

• **Addition**. Let $\{a_1, \ldots, a_L\}$ be a t-sharing of the field element a and $\{b_1, \ldots, b_L\}$ be a t-sharing of the field element b. Then it is easy to see that $\{a_1 + b_1, \ldots, a_L + b_L\}$ is a t-sharing of the sum $a + b$. Hence, addition gates can be emulated easily with no interaction between the parties.

• **Random number generation**. In emulating multiplication gates, it is necessary for the parties to generate secret shares in a random field number that will remain unknown to them. To do that, each party M_ℓ, $\ell \in [L]$, generates a uniformly random field value ρ_ℓ and performs t-sharing of it among $M_1, \ldots, M_L$. At the completion of this stage, each M_ℓ adds up all the L shares that it had received and gets a value that we denote by r_ℓ. It is easy to see that $\{r_1, \ldots, r_L\}$ is a t-sharing of the random value $\rho = \sum_{\ell \in [L]} \rho_\ell$. Clearly, ρ is a uniformly random field element, as it is a sum of uniformly random independent field elements.

• **Multiplication**. Let $[a]_t$ be a t-sharing of a, which was generated by a polynomial $f(\cdot)$ of degree $t-1$; and let $[b]_t$ be a t-sharing of b, which was generated by a polynomial $g(\cdot)$ (also of degree $t-1$). The goal is to obtain t-sharing of $c = a \cdot b$.

First, M_ℓ computes $c_\ell = a_\ell \cdot b_\ell$, $\ell \in [L]$. Those values are point values of the polynomial fg, which is a polynomial of degree $2t-2$. Hence, $\{c_1, \ldots, c_L\}$ is a $(2t-1)$-sharing of c. Note that as $t = \lfloor (L+1)/2 \rfloor$, Eq. (5), then $2t - 1 \leq L$; therefore, the L parties have a sufficient number of shares in order to recover c. However, our goal is to obtain a t-sharing of c, namely a set of shares in c, of which any selection of only t shares can be used to reconstruct c. Hence, we proceed to describe a manner in which the parties can translate this $(2t-1)$-sharing of c into a t-sharing of c.

To do that, the parties generate two sharings of the same uniformly random (and unknown) field element R: a t-sharing, denoted $\{r_1, \ldots, r_L\}$, and a $(2t-1)$-sharing, denoted $\{R_1, \ldots, R_L\}$. Next, each M_ℓ computes $\tilde{c}_\ell = c_\ell + R_\ell$ and sends the result to M_1. Since $\{\tilde{c}_1, \ldots, \tilde{c}_L\}$ is a $(2t-1)$-sharing of $c + R$, M_1 can use any $2t-1$ of those shares in order to reconstruct $\tilde{c} := c + R$. M_1 broadcasts that value to all parties. Consequently, each M_ℓ computes $\hat{c}_\ell = \tilde{c} - r_\ell$, $\ell \in [L]$. Since $\tilde{c}$ is a constant and r_ℓ is a t-out-of-L share in R, then $\hat{c}_\ell$ is a t-out-of-L share in $\tilde{c} - R = c + R - R = c$, as needed. This procedure is perfectly secure since $\tilde{c} = c + R$ reveals no information on c because R is a uniformly random field element that is unknown to the parties.

4.3. Secure comparisons

Let a and b be two integers smaller than p, which is the size of the underlying field $\mathbb{Z}_p$. Assume that the parties $M_1, \ldots, M_L$ hold t-out-of-L shares in both a and b. They wish to compute a t-sharing of the bit $1_{a<b}$ that indicates whether $a < b$ or not, without learning any information on a and b. A protocol that does that is called *secure comparison*. Such a protocol is instrumental in MD-Max-Sum, and it is a fundamental building block in other computations as well (see Section 4.4 below).

Secure comparison is an area of active study, and improvements are being developed rapidly. An efficient constant-round protocol was considered an open problem for a significant amount of time. The first efficient solution was introduced by Damgård et al. [40] by means of *bit-decomposition*. It required $O(\ell \cdot \log(\ell))$ multiplications, where $\ell = \log_2(p)$. A year later, Nishide and Ohta [41] presented an improved method for secure comparison. It is based on the following simple observation. If we denote the bits $1_{a<\frac{p}{2}}$, $1_{b<\frac{p}{2}}$, $1_{[(a-b) \mod p]<\frac{p}{2}}$, and $1_{a<b}$ by w, x, y, z, respectively, then

$$z = w\bar{x} \vee \bar{w}\bar{x}\bar{y} \vee wx\bar{y} . \tag{6}$$

The equality in Eq. (6) can be confirmed by the truth table in Table 1. Next, we translate the Boolean expression in Eq. (6) to an equivalent arithmetic expression:

$$\begin{aligned} z &= w(1-x) + (1-w)(1-x)(1-y) + wx(1-y) \\ &= 1 - x - y + xy + w(x + y - 2xy)\,. \end{aligned} \tag{7}$$

Hence, we reduced the problem of comparing two secret shared values, a and b, to computing three other comparison bits, w, x, y, and then evaluating an arithmetic function of them, Eq. (7). What makes this alternative expression efficiently computable is the fact that in the three comparison bits, $w = 1_{a<\frac{p}{2}}$, $x = 1_{b<\frac{p}{2}}$, and $y = 1_{[(a-b) \mod p]<\frac{p}{2}}$, the right-hand side is $\frac{p}{2}$, as we proceed to explain.

Lemma 1. *Given a finite field $\mathbb{Z}_p$ and a field element $q \in \mathbb{Z}_p$, then $q < \frac{p}{2}$ if and only if the least significant bit (LSB) of $(2q \mod p)$ is zero.*

Proof. If $q < \frac{p}{2}$ then $2q < p$. Hence, $2q \mod p = 2q$ (there is no modular reduction), and therefore, as $2q$ is even, its LSB is 0. On the other hand, if $q > \frac{p}{2}$ then $2q > p$. Hence, $2q \mod p = 2q - p$. Since that number is odd, its LSB is 1. □

In view of Lemma 1, the parties may compute t-out-of-L shares in $1_{q<\frac{p}{2}}$ by computing shares in the field element $2q$ and then using those shares in order to compute shares in the corresponding LSB. The reader is referred to [41] for the details of that last step in the computation.

We conclude this section by commenting on the complexity of the above-described secure comparison protocol. Computing shares in the LSB of a shared value requires 13 rounds of communication and $93\ell + 1$ multiplications. Since we have to compute three such bits (i.e., w, x, and y) then we can compute shares of those three bits in 13 rounds and a total of $279\ell + 3$ multiplications. Finally, we should evaluate the expression in Eq. (7), which entails two additional rounds and two additional multiplications. Hence, the total complexity is 15 rounds and $279\ell + 5$ multiplications. In comparison, the cost of the protocol of [40] that was based on bit decomposition required 44 rounds and $205\ell + 188 + \log_2 \ell$ multiplications.

4.4. Secure computation of minima

As in Section 4.3, let us assume that a and b are two integers smaller than p, and that the parties $M_1, \ldots, M_L$ hold t-out-of-L shares in both a and b. They wish to compute a t-sharing of $\min(a, b)$. Since

$$\min(a, b) = b + 1_{a<b} \cdot (a - b)\,, \tag{8}$$

the parties may perform that computation securely (i.e., without revealing a, b, or $1_{a<b}$) as follows. First, they compute a t-sharing of the bit $1_{a<b}$, as described above. Then, using the emulation of multiplications (see Section 4.2.2), they proceed to compute a t-sharing of the value $c := 1_{a<b} \cdot (a - b)$. Finally, each party adds its share in b to its share in c. In view of Eq. (8), the resulting shares are t-sharing of $\min(a, b)$.

In what follows we let **MIN**(a, b) denote the MPC protocol that takes as inputs a t-sharing of a and a t-sharing of b and outputs a t-sharing of $\min(a, b)$. As that protocol amounts to performing the secure comparison protocol (Section 4.3) with an additional round that involves a single multiplication (as can be seen in Eq. (8)), its total complexity is 16 rounds and $279\ell + 6$ multiplications.

As a concluding remark, we refer the reader to Appendix D in which we exemplify the inputs and outputs of the MPC protocols discussed in this section.

5. Mediated Max-Sum

In order to implement Max-Sum in a manner that preserves the privacy of the agents even when some of them collude, we propose herein an implementation of the algorithm in the mediated model. Let $\mathbf{M} = \{M_1, \ldots, M_L\}$ be an external committee of so-called *mediators*. The agents in $\mathcal{A}$ will share their DCOP private inputs, namely, the topology and constraint information, with the mediators using a t-out-of-L secret sharing scheme, where $t = \lfloor (L+1)/2 \rfloor$ (see Eq. (5)). The agents trust the mediators to have an honest majority, in the sense that if some of the mediators decide to collude in order to reconstruct the shared private data, the number of colluding mediators would be smaller than the number of mediators outside the coalition. Under that assumption, the mediators cannot recover the private inputs that were shared with them, since at least t mediators have to collude in order to be able to reconstruct the shared secrets, and $t = \lfloor (L+1)/2 \rfloor \geq L - t$.

After the agents had completed sharing all their private inputs with the mediators, they go to rest and the mediators start emulating the performance of the entire Max-Sum algorithm by implementing MPC techniques on the shared data. The main challenge in this regard is to design an implementation of Max-Sum that operates on *shared* data, namely, in a manner that is oblivious to the underlying topology and constraint values. When the mediators complete their emulation of Max-Sum, say by running an agreed preset number of iterations, K, they send to each of the agents a message from which that agent can infer the assignment of its variable in the solution that the algorithm had found. We call this algorithm MD-Max-Sum.

In order to hide the constraint graph topology from the mediators, MD-Max-Sum operates on an augmented version $G_+ = (V, E_+)$ of the constraint graph $G = (V, E)$, which includes, in addition to the actual edges in G, also some phantom

edges (see Section 5.1). As demonstrated in Theorem 4, running Max-Sum over a constraint graph that is augmented by phantom edges does not change the output of the algorithm. With such added phantom edges, the mediators cannot tell which of the edges in the augmented graph are actual ones and which are phantom ones, so the graph topology is preserved.

The remainder of this section is organized as follows. In Section 5.1 we discuss phantom edges, prove that adding such edges has no effect on the computed solution, and explain how we use such edges in order to hide the constraint topology from the mediators. In Section 5.2 we describe the main body of MD-Max-Sum: namely, the protocol that the mediators run on the shares that they had received from the agents in order to emulate in a privacy-preserving manner the operation of Max-Sum. In Section 5.3 we prove the correctness of MD-Max-Sum and its privacy guarantees. The costs of MD-Max-Sum are analyzed in Section 5.4.

5.1. Phantom edges and their effect on Max-Sum

Assume that X_n and X_m are two variables that are not constrained. As discussed in Section 3.1, the fact that those two variables are not constrained may be represented by setting a zero constraint matrix $C_{n,m}$ between them. Namely, one may add to the constraint graph G an edge $e = (X_n, X_m)$ with a corresponding constraint matrix $C_{n,m}$ which is the zero matrix. We refer to such an edge as a *phantom edge*.

Phantom edges are not needed when executing Max-Sum. However, we will add phantom edges in the mediated version of Max-Sum, which we present in Section 5.2, in order to hide the topology of the constraint graph. The question which we address here is the following: does the addition of phantom edges affect the operation of Max-Sum? We will show that while the content of the messages will be affected by adding phantom edges, the output of the algorithm will remain unchanged.

Before we start our discussion, we make the following observation. While in principle the constraints in a given DCOP are real values, when represented in finite precision arithmetic they are approximated by rational numbers of the form $c \cdot 2^{-d}$, where c and d are non-negative integers. Hence, by multiplying all constraints in a given DCOP with a suitable power of 2, we arrive at an equivalent DCOP where all constraints are non-negative integers. Therefore, all Q- and R-messages that are generated in the course of Max-Sum are vectors in $\mathbb{Z}^D$ for some $D \in \{|D_1|, \ldots, |D_N|\}$.

Definition 2. Two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{Z}^D$ are called equivalent if there exists an integer $w \in \mathbb{Z}$ such that $\mathbf{u} = \mathbf{v} + w$, where $\mathbf{v} + w$ is the vector in which $(\mathbf{v} + w)(i) = \mathbf{v}(i) + w$, for all $1 \leq i \leq D$. In that case, we denote such a relation by $\mathbf{u} \sim \mathbf{v}$.

It is easy to see that $\sim$ is an equivalence relation. Our main claim is the following.

Lemma 3. *Let:*

- $G = (V, E)$ *be the constraint graph of a given DCOP;*
- $G_+ = (V, E_+)$ *be an augmented graph over the same set of nodes* V*, where* E_+ *is a superset of* E *and all edges* $e \in E_+ \setminus E$ *are phantom edges;*
- G' *and* G'_+ *be the corresponding factor graphs.*

Fix a variable node X_n *and an adjacent function node* X_e *in* G' *and* G'_+*, where* e *is not a phantom edge. For* $k \geq 0$*, let* $Q^k_{n\to e}$*,* $R^k_{e\to n}$*,* $Q^k_{+,n\to e}$ *and* $R^k_{+,e\to n}$ *denote the* Q*- and* R*-messages between those two nodes in the* k*th iteration of* Max-Sum *when operating on* G' *and* G'_+*, respectively. Then* $Q^k_{+,n\to e} \sim Q^k_{n\to e}$ *and* $R^k_{+,e\to n} \sim R^k_{e\to n}$*.*

Namely, while the addition of phantom edges may change the content of the messages that Max-Sum generates, the change will be in the form of a constant shift of the components of each such message.

Proof. At first, we assume that there is only one phantom edge e_+. Later, we will consider the general case of any number of phantom edges.

The proof goes by induction on the iteration number k. Clearly, the claim holds when $k = 0$, since then all messages are zero and, consequently, $Q^0_{+,n\to e} \sim Q^0_{n\to e}$ and $R^0_{+,e\to n} \sim R^0_{e\to n}$. Assume next that the claim holds for the kth iteration. We proceed to prove that it also holds for the subsequent $(k+1)$-th iteration.

We start with the R-messages. Let $e = (X_n, X_m)$ be any function node. By Eq. (3), $R^{k+1}_{+,e\to n}(x) = \min_{y\in D_m}[C_{n,m}(x, y) + Q^k_{+,m\to e}(y)]$. By the induction hypothesis,

$$\begin{aligned} R^{k+1}_{+,e\to n}(x) &= \min_{y\in D_m}[C_{n,m}(x, y) + Q^k_{m\to e}(y) + w] \\ &= \min_{y\in D_m}[C_{n,m}(x, y) + Q^k_{m\to e}(y)] + w \\ &= R^{k+1}_{e\to n}(x) + w\,, \end{aligned}$$

for some integer w. We infer that $R^{k+1}_{+,e\to n} \sim R^{k+1}_{e\to n}$, as required.

Next, we prove the claim for the Q-messages. Here we distinguish between two cases. In messages emerging from a variable node, X_n, that is not adjacent to the phantom function node, X_{e_+}, we get by Eq. (2) that

$$Q^{k+1}_{+,n\to e} = \sum_{X_f \in V_n \setminus \{X_e\}} R^k_{+,f\to n}\,, \tag{9}$$

while, in the original graph, the messages are as in Eq. (2). As, by induction, $R^k_{+,f\to n} \sim R^k_{f\to n}$, we infer that $Q^{k+1}_{+,n\to e} \sim Q^{k+1}_{n\to e}$, as required.

As for messages emerging from a variable node X_n that is adjacent to X_{e_+}, we observe that the set of adjacent function nodes to X_n in the augmented factor graph G'_+ is $V_{+,n} = V_n \cup \{X_{e_+}\}$. Hence, while $Q^{k+1}_{n\to e}$ is given by Eq. (2), the entries of the corresponding Q-message in the augmented graph are given by

$$Q^{k+1}_{+,n\to e} = R^k_{+,e_+\to n} + \sum_{X_f \in V_n \setminus \{X_e\}} R^k_{+,f\to n}\,. \tag{10}$$

By induction, the vector sum on the right-hand side of Eq. (10) is equivalent to $Q^{k+1}_{n\to e}$, as given by Eq. (2). It remains to prove that the additional vector $R^k_{+,e_+\to n}$ on the right-hand side of Eq. (10) is a constant vector. That vector is given by

$$R^k_{+,e_+\to n}(x) = \min_{y\in D_m}[C_{n,m}(x,y) + Q^k_{+,m\to e_+}(y)]\,. \tag{11}$$

As e_+ is a phantom edge, we have $C_{n,m}(x,y) = 0$ for all $x \in D_n$ and $y \in D_m$. Hence, $R^k_{+,e_+\to n}(x) = \min_{y\in D_m}[Q^k_{+,m\to e_+}(y)]$. Since the arguments within the minimum function do not depend on x, we infer that all entries in the $R^k_{+,e_+\to n}$ vector are equal. That completes the proof, in the case of a single phantom edge.

The generalization of the proof to any number of phantom edges follows by induction on the number of phantom edges, because the equivalence relation is transitive. □

Theorem 4. *Running* Max-Sum *twice, once on G' and once on G'_+, for the same number K of iterations, results in the same set of final assignments.*

Proof. By Lemma 3, if X_n is any variable node, then $\overline{R}_n \sim \overline{R}_{+,n}$, where $\overline{R}_n$ is the final vector that X_n computes in the execution of Max-Sum on G', while $\overline{R}_{+,n}$ is the final vector that X_n computes in the execution of Max-Sum on G'_+. Hence, the value $x \in D_n$ that minimizes $\overline{R}_n$ is also the one that minimizes $\overline{R}_{+,n}$. Therefore, the two executions yield the same assignment to X_n. □

5.1.1. Topology privacy index

Definition 5. Let $\gamma \in [0,1]$ denote the density of phantom edges. Namely, it is the fraction of the number of phantom edges in $G_+ = (V, E_+)$ (i.e., $|E_+ \setminus E|$) divided by the number of non-edges in $G = (V, E)$ (i.e., $\binom{N}{2} - |E|$). Then γ is called the topology privacy index.

When $\gamma = 0$ we get $G_+ = G$; in that case, the topology is fully revealed to the mediators. When $\gamma = 1$, G_+ is the complete graph; in that case, the topology of graph G is fully hidden from the mediators.

When $0 < \gamma < 1$ we get an intermediate level of topology privacy. With such values of γ we still achieve topology privacy, but somewhat weaker, since the mediators could infer that all edges in $V_2 \setminus E_+$ are not in the actual constraint graph topology E. For convenience, in our description below of MD-Max-Sum we will assume that $\gamma = 1$ so that the algorithm operates on the complete graph.

5.2. The MD-Max-Sum algorithm

We assume that all parties (agents and mediators) know the total number of agents N, and the identifying index $n \in [N]$ of each agent. In addition, the sizes of all domains, $|D_n|$, $n \in [N]$, are also publicly known.

5.2.1. Distributing to the mediators shares in the problem inputs

In this preliminary stage, the agents share with the mediators the problem inputs, which, as explained in Section 3.1, are encoded through the set of matrices $\mathcal{C}$, see Eq. (1). To do so, each agent A_n, $1 \le n \le N-1$, shares with the mediators $\mathbf{M}$ the constraint matrices $C_{n,m} \in \mathcal{C}$ for all $n < m \le N$, where, as explained in Section 3.1, the matrix $C_{n,m}$ is of dimensions $|D_n| \times |D_m|$ and it either spells out the constraint values between those two agents or, if they are not constrained, it is the zero matrix. The matrices are shared by performing an independent t-out-of-L secret sharing for each entry in each of those

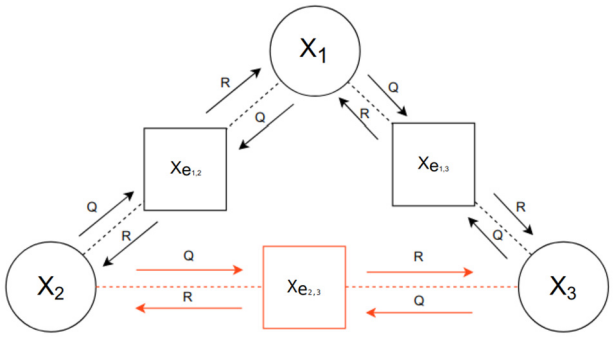


Fig. 3. An Augmented Factor Graph: $X_{e_{1,2}}$ and $X_{e_{1,3}}$ are actual function nodes; $X_{e_{2,3}}$ is a phantom function node. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

matrices, where $t = \lfloor (L+1)/2 \rfloor$ (see Section 4.2.2). Letting $n < m \in [N]$ be indices of two agents, and $\ell \in [L]$ be an index of a mediator, we denote the share of the cost $C_{n,m}(i,j)$ that the mediator M_ℓ receives by $C^\ell_{n,m}(i,j)$. The entire matrix of shares that M_ℓ receives is denoted

$$C^\ell_{n,m} = \left(C^\ell_{n,m}(i,j) : 1 \le i \le |D_n|, 1 \le j \le |D_m| \right) .$$

After each mediator M_ℓ, $\ell \in [L]$, got its share matrix $C^\ell_{n,m}$ for all $1 \le n < m \le N$, they have all problem inputs and they may now begin an MPC emulation of Max-Sum over those inputs. We exemplify the above-described procedure of secret sharing the private inputs in Appendix C.

The addition of phantom edges is carried out in the following manner. Let γ be the topology index that was selected upfront by the agents (see Definition 5). Then for each pair of variables that are not constrained, the agents can choose to add it as a phantom edge in probability γ. Specifically, if X_n and X_m are not constrained, i.e. $(X_n, X_m) \notin E$, and $n < m$, then agent A_n may select a number r uniformly at random from the interval $[0,1]$, and if $r \le \gamma$ then it will add (X_n, X_m) as a phantom edge and will distribute to the mediators shares in a zero matrix $C_{n,m}$ of dimensions $|D_n| \times |D_m|$. (Recall that the overall number of agents N, the index of each agent, and the domain sizes are known to all.)

We assume that the set of mediators has an *honest majority*. That means that some of the mediators may collude in order to combine the pieces of information that they got in an attempt to extract private information; however, the number of colluding agents is smaller than $L/2$ under the assumption of an honest majority. As we use t-out-of-L secret sharing with $t = \lfloor (L+1)/2 \rfloor$, then at least t mediators have to collude in order to recover the problem inputs. Since $t = \lfloor (L+1)/2 \rfloor \ge L/2$, such a scenario is impossible under our working assumption of an honest majority. Hence, the mediators cannot learn any information on the content of the constraint matrices. Therefore, not only the constraints themselves are kept secret, but also the topology is kept secret since the mediators cannot tell from their shares whether $C_{n,m}$ is the zero matrix or not.

While Max-Sum, as well as P-Max-Sum, operate on the exact factor graph G', the mediated algorithm MD-Max-Sum operates on an *augmented factor graph*, denoted $G'_+ = (V'_+, E'_+)$, in which every two variable nodes are connected through a function node, even if some of those function nodes stand for a zero/phantom constraint (which was introduced only for the purpose of hiding the real topology of G from the mediators). Fig. 3 is an example of an augmented factor graph: the edges $e_{1,2}$ and $e_{1,3}$ are actual constraints between the agents, while $e_{2,3}$ is a zero/phantom constraint (and is marked by red). The mediators operating on the augmented factor graph cannot distinguish between the actual constraints and the phantom constraints.

After the agents finish distributing to the mediators shares in the problem inputs, they go to rest and let the mediators do the work. The mediators start an emulation of each of the iterations in Max-Sum. They do so by producing proper shares in the true messages that would have been sent along each edge of the factor graph, if the agents had run the Max-Sum algorithm by themselves.

We proceed to explain in the next sections the details of the MD-Max-Sum implementation. Specifically, we need to explain how in each iteration of the algorithm, the mediators can create proper shares in the Q- and R-messages that the corresponding Max-Sum algorithm would have generated. In doing so, we focus on an arbitrary pair of neighboring nodes in the augmented factor graph: a variable node X_n, $n \in [N]$, and a function node, X_e, where $e = (X_n, X_m)$ and $m \in [N] \setminus \{n\}$.

5.2.2. Producing shares in the messages of the initial iteration

In iteration 0, all of the L mediators have to emulate zero messages between X_n and X_e,

$$Q^0_{n \to e} = (0, \ldots, 0) \in \mathbb{Z}_p^{|D_n|}, \quad R^0_{e \to n} = (0, \ldots, 0) \in \mathbb{Z}_p^{|D_n|} . \tag{12}$$

(Recall that $\mathbb{Z}_p$ is the underlying field in which all computations take place, see Section 4.1.) To do so, each mediator M_ℓ, $\ell \in [L]$, creates for himself corresponding zero share vectors as follows:

$$Q_{n\to e}^{0,\ell} = (0,\ldots,0) \in \mathbb{Z}_p^{|D_n|}, \quad R_{e\to n}^{0,\ell} = (0,\ldots,0) \in \mathbb{Z}_p^{|D_n|}. \tag{13}$$

Note that no interaction between the mediators is needed at this stage and that the L vector shares of the Q-messages in Eq. (13) are t-out-of-L vector shares in the zero Q-messages in Eq. (12), and likewise for the R-messages.

5.2.3. Producing shares in Q-messages

In iteration $k+1$, the mediators have to emulate the message $Q_{n\to e}^{k+1}$ from the variable node X_n to the adjacent function node, X_e, where $e = (X_n, X_m)$. In view of Eq. (2), and the fact that the mediators already have t-out-of-L shares in R-messages of the kth iteration, such a computation can be done locally, without interaction between the mediators, as follows:

$$Q_{n\to e}^{k+1,\ell} := \sum_{X_f \neq X_e} R_{f\to n}^{k,\ell}, \tag{14}$$

where the sum is over all $N-2$ function nodes, X_f, between X_n and X_i for any $i \in [N] \setminus \{n, m\}$.

5.2.4. Producing shares in R-messages

Here, we concentrate on the more involved task of computing t-out-of-L shares in the R-messages, $R_{e\to n}^{k+1}$, from the function node, X_e, where $e = (X_n, X_m)$, to the variable node X_n. We rewrite Eq. (3) in the following manner,

$$R_{e\to n}^{k+1}(x) := \min_{y\in D_m} B_{n,m}^{k}(x,y), \quad x \in D_n, \tag{15}$$

where $B_{n,m}^{k}(x,y)$ denotes the sum

$$B_{n,m}^{k}(x,y) := C_{n,m}(x,y) + Q_{m\to e}^{k}(y). \tag{16}$$

The L mediators hold t-out-of-L shares in $C_{n,m}(x,y)$ for all $(x,y) \in D_n \times D_m$ (denoted $C_{n,m}^{\ell}(x,y)$, $\ell \in [L]$), since such shares were generated and distributed to them by the agents in the preliminary stage. Moreover, the mediators had computed in the kth iteration t-out-of-L shares in $Q_{m\to e}^{k}(y)$ for all $y \in D_m$, where M_ℓ's shares are denoted $Q_{m\to e}^{k,\ell}(y)$. Hence, $C_{n,m}^{\ell}(x,y) + Q_{m\to e}^{k,\ell}(y)$, which we denote by $B_{n,m}^{k,\ell}(x,y)$, are t-out-of-L shares in $B_{n,m}^{k}(x,y)$, as implied by Eq. (16) and the linearity of secret sharing. Hence, the main computational challenge is to compute t-out-of-L shares in the left-hand side of Eq. (15) from the shares that the mediators hold in each of the terms on the right-hand side of Eq. (15). This task is non-trivial because the minimum function is non-linear.

Protocol 1, which we describe below, is simultaneously executed by each of the L mediators. It is executed for each pair of a function node in the augmented factor graph, X_e, where $e = (X_n, X_m)$, and one of its two adjacent variable nodes, X_n. At the completion of that protocol, each mediator M_ℓ holds a share $R_{e\to n}^{k+1,\ell}$ in $R_{e\to n}^{k+1}$. That protocol will be executed in every iteration $N(N-1)$ times, as there are $\binom{N}{2} = N(N-1)/2$ function nodes in the augmented factor graph G'_+, and each one of them has two adjacent variable nodes.

Protocol 1: Computing shares in an R-message from the function node X_e, where $e = (X_n, X_m)$, to the variable node X_n.

Input: Mediator M_ℓ, $\ell \in [L]$, holds a t-out-of-L share, $B_{n,m}^{k,\ell}(x,y)$, in $B_{n,m}^{k}(x,y)$, for every $x \in D_n$ and $y \in D_m = \{y_1,\ldots,y_{|D_m|}\}$.

1 **forall** $x \in D_n$ **do**
2 $\quad M_\ell$ sets $\beta_{n,m}^{\ell}(x) \leftarrow B_{n,m}^{k,\ell}(x, y_1)$
3 $\quad$ **forall** $j = 2,\ldots,|D_m|$ **do**
4 $\quad\quad \{\beta_{n,m}^{\ell}(x)\}_{\ell\in[L]} \leftarrow$ **MIN**$(\{B_{n,m}^{k,\ell}(x,y_j)\}_{\ell\in[L]}, \{\beta_{n,m}^{\ell}(x)\}_{\ell\in[L]})$
5 $\quad M_\ell$ sets $R_{e\to n}^{k+1,\ell}(x) \leftarrow \beta_{n,m}^{\ell}(x)$

Output: Mediator M_ℓ, $\ell \in [L]$, gets a t-out-of-L share $R_{e\to n}^{k+1,\ell}(x)$ in $R_{e\to n}^{k+1}(x)$.

The external loop in the protocol (Lines 1-5) is over all values x in the domain D_n, i.e., over all entries in the vector message $R_{e\to n}^{k+1}$. For each such $x \in D_n$, the mediators have to find the minimum among $\{B_{n,m}^{k}(x,y) : y \in D_m\}$, where each of the values in that set is shared by a t-out-of-L scheme among them. The t-out-of-L shares of the minimum will be stored in $\beta_{n,m}^{\ell}(x)$, $\ell \in [L]$. First (Line 2), each mediator initiates its β-shares with the shares corresponding to $B_{n,m}^{k}(x,y_1)$. Then (Lines 3-4), for each y_j, $j = 2,\ldots,|D_m|$, the mediators compare $B_{n,m}^{k}(x,y_j)$ to the current minimum, in which they have t-out-of-L shares in $\beta_{n,m}^{\ell}(x)$, $\ell \in [L]$, and update their shares in the minimum accordingly. Specifically, in order to compute the minimum of two values that are known to the mediators only through t-out-of-L shares, without recovering those values or realizing which of the two is the minimum, Protocol 1 calls upon the MPC sub-protocol **MIN** (Line 4), which we described

in Section 4.4. That sub-protocol takes as input the t-sharings in each of the two secret-shared values and outputs to the mediators a t-sharing of their minimum.[4]

Finally (Line 6), each mediator stores in $R_{e\to n}^{k+1,\ell}(x)$ its share in the minimum that was found above, $\beta_{n,m}^{\ell}(x)$.

5.2.5. Normalizing messages

In order to prevent the entries in the messages from growing uncontrollably, it is customary to subtract from each entry in each message $Q_{n\to e}^{k+1}$ the value $\alpha_{n,m}^{k+1} := \min_{x\in D_n} Q_{n\to e}^{k+1}(x)$ (see [4]). To perform such normalization, we need to find for each message $Q_{n\to e}^{k+1}$ the minimum entry $\alpha_{n,m}^{k+1}$, and then subtract it from each of the entries in $Q_{n\to e}^{k+1}$.

To find the minimal entry in $Q_{n\to e}^{k+1}$, a vector of dimension $|D_n|$, it is necessary to perform $|D_n|-1$ secure computations of minima. As the mediators hold shares in each of those entries, they may apply the **MIN** sub-protocol (Section 4.4) $|D_n|-1$ times in a similar manner to the minimum computations in Protocol 1. After finding the minimum, each mediator will subtract the share that it holds in that minimum from the share that it holds in each entry in $Q_{n\to e}^{k+1}$.

To reduce computation time, the normalization procedure could be performed every K_1 iterations, instead of every iteration, for a suitable selection of K_1 that depends on the size p of the underlying field $\mathbb{Z}_p$.

Let us find first the maximal number of iterations for which we can be ascertained that all entries in all messages do not exceed $p-1$. Let

$$q = \max_{1\le n<m\le N} \max_{x\in D_n, y\in D_m} C_{n,m}(x,y) \tag{17}$$

denote the maximum value of any single constraint, and assume that the maximal degree in the constraint graph G is $d+1$. It was shown in [22, Theorem 2.1] that all entries in all messages in Max-Sum that are sent during the first k iterations are bounded by $q\cdot\frac{d^h-1}{d-1}$, where $h=\lfloor k/2\rfloor+1$. Hence, it is easy to verify that as long as

$$d^h < \Gamma := \frac{(d-1)p}{q}+1\,, \tag{18}$$

the content of all entries in all messages will be strictly smaller than p. From the latter inequality, we infer that as long as $h<\frac{\log\Gamma}{\log d}$, or $k<\frac{2\log\Gamma}{\log d}$, no normalization is needed. In our experiments, we set K_1 to be half that upper bound, namely $K_1 := \lfloor\frac{\log\Gamma}{\log d}\rfloor$, and we then applied normalization every K_1 iterations.

5.2.6. Termination

After completing a preset number of K iterations, the final assignment to X_n, $n\in[N]$, is determined by the minimal entry in $\overline{R}_n = \sum_{X_e\in V_n} R_{e\to n}^{K}$. To perform that computation, each agent A_n, $n\in[N]$, selects a subset of t mediators and asks them for their shares in the vector $\overline{R}_n$. Using those vector shares, A_n can recover the entire vector $\overline{R}_n$ by executing $\mathsf{Reconstruct}_t(s_1,\ldots,s_L)$ (see Section 4.1) on the shares of each of the $|D_n|$ components in $\overline{R}_n$. Afterward, A_n finds the minimal entry in $\overline{R}_n$ and then assigns the corresponding value to X_n.

5.2.7. A bird's-eye view of MD-Max-Sum

As a summary of our discussion so far, we provide here a bird's-eye view of the MD-Max-Sum algorithm (Algorithm 2). In addition, Fig. 4 illustrates the whole system.

Algorithm 2: Bird's-eye view of MD-Max-Sum.

```
1 Initial sharing of all topology and constraint information (Section 5.2.1)
2 Perform the initial k = 0 iteration (Section 5.2.2)
3 forall k = 1, ..., K do
4     forall (X_n, X_e) ∈ E'_+ do
5         Compute t-out-of-L shares in the Q^k_{n→e}-messages (Section 5.2.3)
6         Compute t-out-of-L shares in the R^k_{e→n}-messages (Section 5.2.4)
7     if (k mod K_1) = 0 then
8         Normalize messages (Section 5.2.5)
9 Terminate (Section 5.2.6)
```

[4] In the preliminary version of this paper [21] the mediators computed the minimum in a manner that disclosed ordering information between the compared entries, as well as the index of the minimal entry. Here, we prevent any such unwanted leakage of information as we rely on the MPC sub-protocol **MIN**, which is designed for computing the minimum and nothing beyond that.

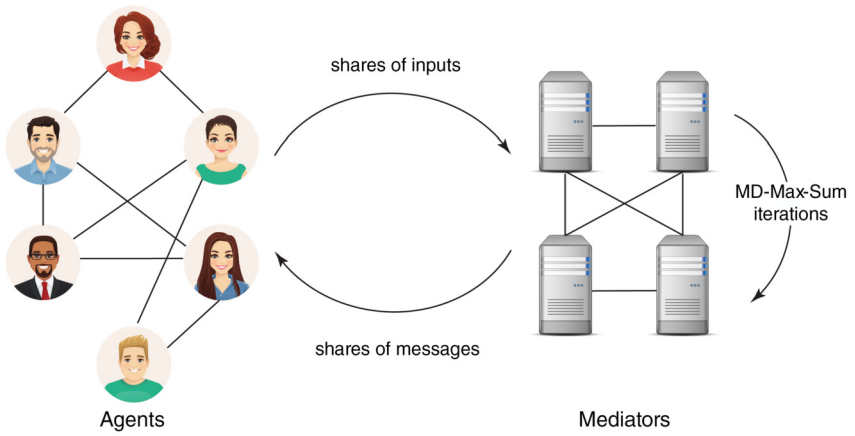


Fig. 4. An overview of MD-Max-Sum. The agents and the constraint graph between them are shown on the left, while the mediators are shown on the right. The top arrow represents the operation in Line 1 of Algorithm 2. The right arrow stands for all computations that the mediators perform among themselves (Lines 2-8). The bottom arrow represents the termination stage (Line 9) in which the mediators send back to the agents shares in the final R-messages from which the agents infer the assignments to their variables.

5.3. Correctness and privacy

In this section, we provide proofs for the correctness and privacy guarantees of MD-Max-Sum. Namely, Lemma 6 and Theorem 7 show that the mediated algorithm MD-Max-Sum perfectly simulates Max-Sum, while Theorem 8 presents the types of privacy that MD-Max-Sum offers.

Lemma 6. *Let:*

- X_n *and* X_e *be neighboring nodes in the factor graph* G'.
- $Q^k_{+,n\to e}$ *and* $R^k_{+,e\to n}$ *be the two messages that are sent between them in the* k*th iteration in* Max-Sum *when it is executed on the augmented factor graph* G'_+.
- $\left\{Q^{\ell,k}_{+,n\to e} : \ell \in [L]\right\}$ *and* $\left\{R^{\ell,k}_{+,e\to n} : \ell \in [L]\right\}$ *be the sets of shares generated in the* k*th iteration in* MD-Max-Sum *between those two nodes.*

Then $\{Q^{\ell,k}_{+,n\to e} : \ell \in [L]\}$ *are* t*-out-of-*L *shares in* $Q^k_{+,n\to e}$ *and, similarly,* $\{R^{\ell,k}_{+,e\to n} : \ell \in [L]\}$ *are* t*-out-of-*L *shares in* $R^k_{+,e\to n}$.

Proof. The claim is obviously correct for iteration $k = 0$. We may now proceed by induction. The computation of the shares in the Q-messages in the kth iteration, as described in Section 5.2.3, and the linearity of secret sharing, imply that $\left\{Q^{\ell,k}_{+,n\to e} : \ell \in [L]\right\}$ are t-out-of-L shares in $Q^k_{+,n\to e}$. The computation of the shares in the R-messages in the kth iteration, as described in Section 5.2.4, and the linearity of secret sharing and the correctness of the **MIN** sub-protocol, imply that $\left\{R^{\ell,k}_{+,e\to n} : \ell \in [L]\right\}$ are t-out-of-L shares in $R^k_{+,e\to n}$. □

Theorem 7. *When* MD-Max-Sum *and* Max-Sum *are executed the same number of iterations* K *on the same input problem, they will issue the same assignments to all variables.*

Proof. In view of Lemma 6, the set of shares that the mediators will hold in the R-messages at the completion of the Kth iteration, $\left\{R^{\ell,K}_{+,e\to n} : \ell \in [L]\right\}$, are t-out-of-L shares in $R^K_{+,e\to n}$. By Lemma 3, $R^K_{+,e\to n} \sim R^K_{e\to n}$, where $R^K_{e\to n}$ is the message sent in Max-Sum from X_e to X_n, when it is executed on the original (non-augmented) factor graph G'. Hence, the vector that each agent A_n computes at termination, $\overline{R}_n$, is equivalent to the corresponding vector that would have been computed by A_n at the termination of Max-Sum when executed on G'. In particular, the final computed assignments would be the same in MD-Max-Sum and in Max-Sum. □

Hence, in view of the above, MD-Max-Sum perfectly simulates Max-Sum. However, if an agent "dies" at some stage of the algorithm's execution, the computed results may become wrong or irrelevant. For example, an agent may cease to operate before it completes sending shares in its constraint matrices; in such a case, the mediators would start emulating Max-Sum based on incomplete data. As another example, an agent may permanently cease to operate after it completed sending shares in all of its constraint matrices; in such a case the mediators will solve a DCOP that is no longer relevant, as the set of agents had changed. Therefore, we suggest increasing the robustness of the algorithm as follows. First, each

agent will send to all mediators a special message that indicates that it had finished distributing shares in its constraint matrices, together with the overall number of scalars (constraint matrices' entries) that it had shared (Line 1 in Algorithm 2). The mediators will not start their emulation of Max-Sum (Lines 2-8) before getting such messages from all N agents, and verifying that they had received all shares that there were supposed to receive. Then, at termination (Line 9), they will wait to receive an acknowledgment of receipt from all agents. If one or some of the agents do not respond at that point, the mediators will notify all agents about it, so that the remaining agents will decide on their next steps (namely, whether to use the final assignments that were obtained based on the existence of the agents who ceased to operate, or to start a new computation with the subset of remaining agents).

Next, we turn to discuss the privacy that is offered by the mediated algorithm.

Theorem 8. MD-Max-Sum *provides topology, constraint, and decision privacy, as long as the mediators have an honest majority.*

Proof. The agents share all of their information with the mediators using t-out-of-L secret sharing, where $t = \lfloor (L+1)/2 \rfloor$. Under the assumption of honest majority, the number of mediators that might attempt to reconstruct the shared secrets is smaller than t and, therefore, all topology and constraint information, as encoded in the matrices $\mathcal{C} := \{C_{n,m} : 1 \leq n < m \leq N\}$, Eq. (1), remains fully protected from the mediators, as well as from the agents. Also, the vectors $\overline{R}_n$, $n \in [N]$, that determine the final decisions (see Section 5.2.6) remain out of reach for the mediators, as well as for the other agents. Hence, each agent's final decision remains unknown to all mediators and all other agents. □

A note on the connection between L and the algorithm's security. MD-Max-Sum relies on the honest majority assumption. If the number of mediators is L then at least $t = \lfloor (L+1)/2 \rfloor$ of them would need to betray the trust vested in them and collude in order to reveal all private information. Hence, larger values of L would imply higher levels of security, because a larger number of mediators would need to be corrupted. The trade-off is clear: if the agents wish to increase the algorithm's security they would need to find more mediators for the task, and the implied communication and computational costs would increase, as demonstrated in Section 6.

A note on topology privacy. We recall that MD-Max-Sum provides full topology privacy, as stated in Theorem 8, when the topology privacy index γ is set to 1 (see Definition 5). Lower settings of γ would result in weaker levels of topology privacy, as explained in Section 5.1.1, but will provide better runtimes, as demonstrated in our experimental evaluation, see Section 6.

A note on agent privacy. As stated at the beginning of Section 5.2, all parties (agents and mediators) know the total number of agents N, the identifying index $n \in [N]$ of each agent, and the sizes of all domains, $|D_n|$, $n \in [N]$. Hence, MD-Max-Sum does not provide agent privacy. Another reason why the mediated approach cannot achieve agent privacy is that the agents need to agree upfront on the set of mediators, which requires them to communicate among themselves.

5.4. Computational and communication costs

Here we analyze the computational and communication costs of MD-Max-Sum. The algorithm begins with the agents sending shares in the constraint matrices to the mediators (Section 5.2.1), then the algorithm iterates for a preset number of iterations (Sections 5.2.2–5.2.5), and finally the mediators send to each of the agents a vector from which the agents can infer the assignments to their variables (Section 5.2.6). The opening and concluding phases are executed only once and they include simple secret sharing computations so we ignore them in this analysis. The main computational and communication costs are in the main body of the algorithm.

Also there, the work in the initial iteration and in producing shares in the Q-messages consists of very efficient local computations (see Eqs. (13) and (14)). The only parts in which the mediators engage in a costly MPC sub-protocol, **MIN**, is in computing shares in the R-messages (Section 5.2.4) and in normalizing messages (Section 5.2.5).

In Protocol 1, when executed on the pair of nodes X_n and X_e, where $e = (X_n, X_m)$, the **MIN** sub-protocol is executed $|D_n| \cdot (|D_m| - 1)$ times (see Line 4 there). Letting Δ denote the maximal domain size, then when the topology privacy index γ equals 1 (see Definition 5), the overall number of invocations of **MIN** from Protocol 1 in one iteration of MD-Max-Sum is bounded by $\Delta \cdot (\Delta - 1) N(N-1)$. For general settings of $\gamma < 1$, the bound is $2\Delta \cdot (\Delta - 1)|E_+|$ (since there are $|E_+|$ function nodes and each one of them has two adjacent variable nodes).

In order to normalize a single $Q^k_{n \to e}$-message, the **MIN** sub-protocol is needed to be executed $|D_n| - 1$ times. Hence, the overall number of **MIN** invocations in iterations when we normalize all Q-messages is bounded by $2|E_+|(\Delta - 1)$.

In summary, if MD-Max-Sum is executed for K iterations, and every K_1 iterations we normalize the Q-messages, the overall number of invocations of the **MIN** sub-protocol is bounded by

$$2|E_+|(\Delta - 1) \cdot (\Delta K + \lfloor K/K_1 \rfloor) \,. \tag{19}$$

Since in the most secure version of MD-Max-Sum we have $|E_+| = \binom{N}{2}$, the bound in Eq. (19) is at most

$$N(N-1)(\Delta - 1) \cdot (\Delta K + \lfloor K/K_1 \rfloor) \,. \tag{20}$$

Table 2
Runtime in milliseconds, for different values of L, of the Damgård-Nielsen **MIN** sub-protocol.

L	5	7	9	11	13
runtime	4.5	6.9	13.2	17.2	19.3

As for the computational and communication costs of the **MIN** sub-protocol, see the theoretical discussion in Section 4.4 and the experimental evaluation in Section 6.1.

6. Experimental evaluation

In this section, we describe the experiments that we conducted in order to evaluate the performance of MD-Max-Sum.

We implemented and executed the algorithm on the AgentZero simulator [42], running on AWS C5a instances comprised of a 2nd generation AMD EPYC™ 7R32 processor and 64 GB memory, except for the call to the **MIN** sub-protocol, which is described and evaluated separately in Section 6.1.

To achieve maximal parallelism, the number of CPU threads that we used is greater or equal to the number of agents in all experiments (with one exception that we describe in the last experiment).

Runtime performance in DCOP is commonly evaluated in a logical manner that is independent of implementation and hardware issues. This is usually done by counting the number of non-concurrent constraint checks (NCCCs) [43] since a constraint check is the cardinal operation in most standard DCOP algorithms. However, in privacy-preserving DCOPs the burden of cryptographic operations considerably outweighs that of constraint checks. Therefore, we follow all previous studies on privacy-preserving DCOPs and use the *simulated time* approach [44] instead.

In Section 6.1 we evaluate the runtime of the **MIN** sub-protocol, which is a fundamental and computationally expensive component of MD-Max-Sum. Then, in Section 6.2, we evaluate the performance of the whole MD-Max-Sum algorithm, as a function of the two parameters that affect the level of privacy that it offers: the topology privacy index γ (Definition 5), and the number of mediators L. Finally, in Section 6.3, we compare the performance of MD-Max-Sum with other algorithms: the basic (not-private) Max-Sum, the P-Max-Sum algorithm [22] that preserves privacy, but is not collusion-secure, and PC-SyncBB [45,34], which is the only other privacy-preserving DCOP algorithm that is collusion-secure.

6.1. The *MIN* sub-protocol

The **MIN** sub-protocol was executed using the secure comparison algorithm of Nishide and Ohta [41] (which we described in Section 4.3), as implemented by the generic secret-sharing-based protocol of Damgård and Nielsen [38] with the performance improvements of Chida et al. [39]. The implementation is open source and available online. We executed the sub-protocol over LAN with EC2 machines of type c5.large in Amazon's North Virginia data center, with every agent running on a separate machine. We measured the performance of the protocol for various values of L (the number of mediators).

We selected the order of the underlying finite field $\mathbb{Z}_p$, over which MD-Max-Sum operates, to be $p = 2^{31} - 1$, which is a Mersenne prime (a prime of the form $p = 2^t - 1$ for some integer $t > 1$). Using Mersenne primes is advantageous since multiplying two elements in such fields can be done without performing an expensive division (in case the multiplication result exceeds the modulus).

Table 2 shows, for each L, the runtime of the Damgård-Nielsen **MIN** sub-protocol. Recall that the upper bound on the number of such calls to the **MIN** sub-protocol was analyzed in Section 5.4 (see Eqs. (19) and (20) there).

6.2. The effect of parameters on runtimes

The MD-Max-Sum algorithm depends on two parameters that affect the level of privacy that it offers. Those are the topology privacy index, γ, and the number of mediators, L.

As discussed in Section 5.1, we protect the topology of the constraint graph G from the mediators by adding phantom edges with zero constraints on them. Although such a mechanism does not alter the final output of the algorithm, it does add redundant computations along the phantom edges and subsequently entails a toll on runtime.

In our first experiment, we evaluated the price of topology privacy. We used unstructured random (connected) constraint graphs with $N = 24$ agents, domains of size $|D_n| = 5$, varying constraint densities, $p_1 = 0.1, \ldots, 0.9$, and $L = 5$ mediators. We ran all problem variants for $K = 50$ iterations, where the variants differ in the value of the topology privacy index: $\gamma \in \{0, 0.25, 0.5, 0.75, 1\}$. The resulting runtimes are shown in Fig. 5. As expected, runtime increases when γ increases. Also, for higher values of γ, the effect of the constraint density decreases. Note that when $\gamma = 1$, the constraint density p_1 does not affect runtime, since the algorithm operates on the complete graph regardless of the value of p_1.

In the second experiment, we evaluated the effect of the number of mediators on the runtime of MD-Max-Sum. As in the previous experiment, we used unstructured random (connected) constraint graphs with $N = 24$ agents, domains of size $|D_n| = 5$, and varying constraint densities, $p_1 = 0.1, \ldots, 0.9$. We ran all problem variants for $K = 50$ iterations, where the variants differ in the number of mediators: $L \in \{5, 7, 9, 11, 13\}$. The topology privacy index was set to $\gamma = 0.5$. As can be seen in Fig. 6, when L increases, the runtime increases too, in consistency with the runtimes of the **MIN** sub-protocol, as reported in Section 6.1.

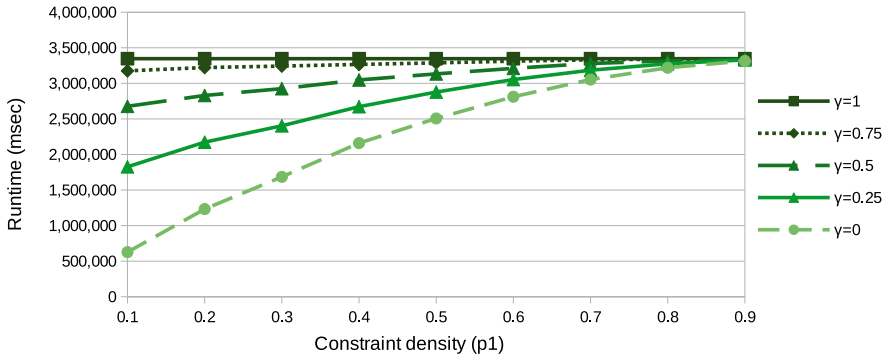


Fig. 5. The effect of the topology privacy index γ on MD-Max-Sum's runtime: Unstructured random graphs, $N = 24$ agents, varying constraint density.

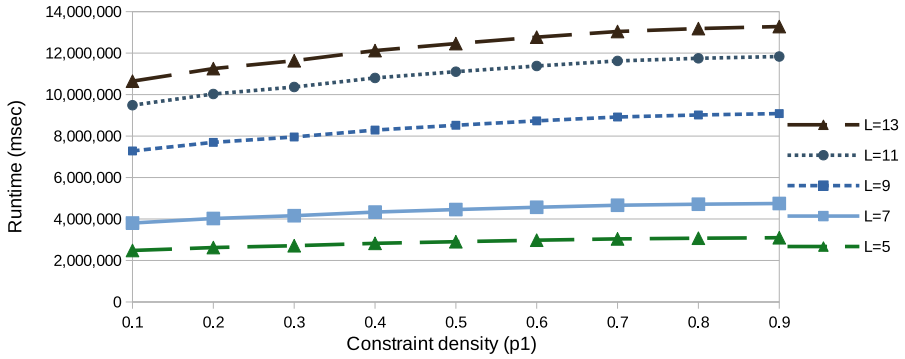


Fig. 6. The effect of the number of mediators L on MD-Max-Sum's runtime: Unstructured random graphs, $N = 24$ agents, varying constraint density.

In practical applications of MD-Max-Sum, it would be necessary to assess the probability of the mediators becoming corrupted and forming coalitions, and according to that to select the number of mediators. For example, if the probability of corrupting 3 mediators is deemed sufficiently small, then it would suffice to take $L = 5$, since a protocol with that number of mediators is secure as long as the number of colluding mediators is smaller than 3. We used MD-Max-Sum with $L = 5$ in all subsequent experiments.

6.3. Comparison of MD-Max-Sum with other algorithms

In the set of experiments that we report next we compared MD-Max-Sum against other similar-purpose algorithms. For the runtime comparisons, we used a logarithmic scale in all these experiments. Unless otherwise stated, we used domains of size $|D_n| = 5$, $n \in [N]$, in these experiments.

We used two variants of MD-Max-Sum: one with $\gamma = 0$ (no topology privacy) and one with $\gamma = 1$ (full topology privacy). The runtimes of those two extreme variants provide lower and upper bounds on the runtimes of all other variants, with $0 < \gamma < 1$. We compared those two variants with the baseline algorithm Max-Sum (no privacy) and P-Max-Sum [22] (provides privacy, but not against coalitions). As shown in [22], and herein in Theorem 7, both of those privacy-preserving implementations of Max-Sum simulate perfectly the basic Max-Sum. We used in all experiments $K = 10$ iterations in all of those algorithms, similarly to [22].

In addition, we included in our experiment the PC-SyncBB algorithm [45,34], which is the only other DCOP-solving algorithm that is privacy-preserving and collusion-secure. Recall that unlike MD-Max-Sum, PC-SyncBB is a complete algorithm; hence, it outputs the optimal solution but it is expected to be more time-consuming.

In the first experiment in this set, we used unstructured random graphs with $N = 9$ agents, see Fig. 7. As can be clearly seen (recall the logarithmic scale), the constraint density highly affects the runtime of PC-SyncBB; this is a known phenomenon of Branch & Bound algorithms. In contrast, the effect of constraint density on the Max-Sum-based algorithms is much milder. Moreover, the constraint density does not affect at all MD-Max-Sum with $\gamma = 1$, see Fig. 5.

Fig. 7 also provides a clear view of the different trade-offs of the algorithms. The gap between the runtimes of Max-Sum and P-Max-Sum demonstrates the price of privacy. The gap between the runtimes of P-Max-Sum and MD-Max-Sum demonstrates the price of collusion security. The gap between the runtimes of MD-Max-Sum and PC-SyncBB demonstrates the price of completeness.

The results of this experiment are not surprising. In each application, it is essential to select a DCOP-solving method by considering the privacy risks and the potential damage in case the privacy is breached and weighing them against the

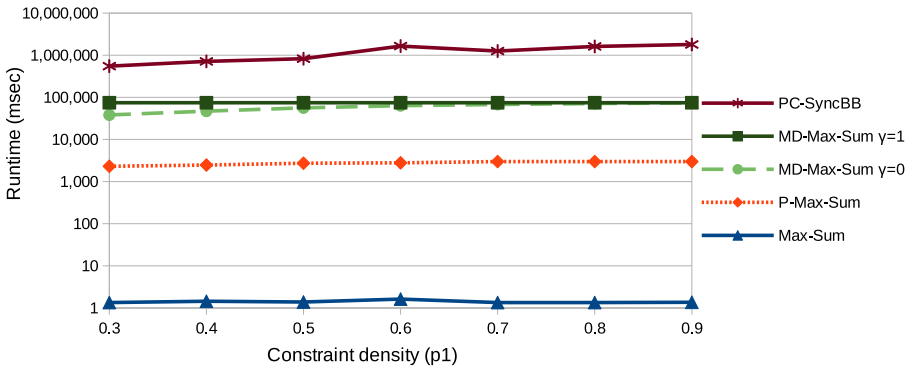


Fig. 7. Runtime on unstructured random graphs, $N = 9$ agents, varying constraint density.

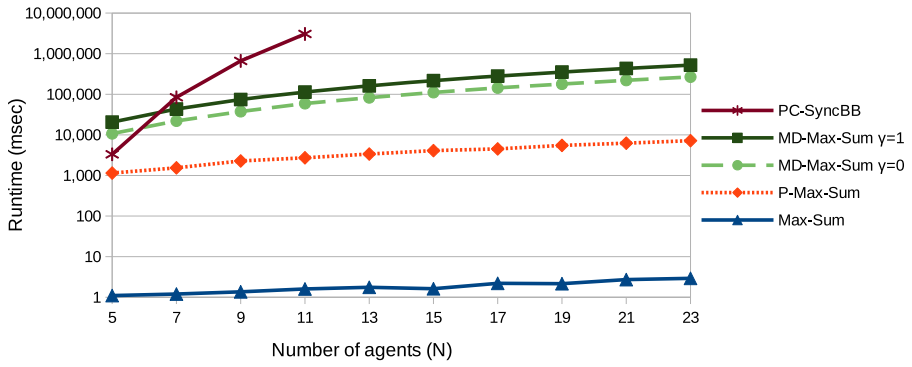


Fig. 8. Runtime on unstructured sparse random graphs, $p_1 = 0.3$, varying N.

needed response time (namely, how much time can the agents wait until they learn how to set their variable assignments), and utility requirements (i.e., how essential it is to find an optimal solution and not just a local optimum).

In the next experiment, shown in Fig. 8, we fixed the constraint density to be $p_1 = 0.3$ and varied the number of agents N in order to observe the scalability of the evaluated algorithms. The cut-off time for a single execution was set to 30 minutes.

The performance gap between P-Max-Sum and MD-Max-Sum is similar to what we witnessed in the previous experiment. For small problems with $N \leq 7$, PC-SyncBB is competitive with MD-Max-Sum and even with P-Max-Sum. However, as the number of agents increases, we can see that the performance of PC-SyncBB becomes much more time-consuming than MD-Max-Sum. This advantage of MD-Max-Sum over PC-SyncBB is explained as follows: a significant portion of the runtime of both algorithms is in performing secure comparisons between secret values. In PC-SyncBB, that MPC sub-protocol is carried out by all agents; in MD-Max-Sum, on the other hand, it is carried out by the mediators. The runtime of this computation depends on the number of interacting parties, as is evident from Table 2 herein and Table 1 in [34]. Hence, while the time spent in PC-SyncBB on secure comparisons increases with N, in MD-Max-Sum it is independent of N. This mitigation of the dependency of the runtime on N demonstrates the strength of the *mediated model*. (Of course, the runtime of MD-Max-Sum does depend on N through other computations, outside the secure comparisons in **MIN**, since N affects the size of the graph.)

In the next experiment, we increased the density of the constraints to $p_1 = 0.7$; all other settings remained the same as in the previous experiment. As can be seen in Fig. 9, the cut-off time of 30 minutes allowed PC-SyncBB to handle only up to 9 agents (as opposed to 11 agents that it could handle within this time limit when the graph was sparse, $p_1 = 0.3$). Furthermore, in comparison with the previous experiment, we can see that higher constraint density leads to an increased runtime performance gap between MD-Max-Sum and PC-SyncBB. Additionally, we observe that for dense problems, both MD-Max-Sum variants converge to roughly the same runtimes (notice that the respective curves are overlapping), which is consistent with our experimentation with the impact of the topology privacy index (Fig. 5). In view of these experiments for dense problems, it is advised to apply the MD-Max-Sum variant that offers full topology privacy ($\gamma = 1$).

We proceed to compare the performance of the various algorithms on problems of other types. Using the model of Barabási and Albert [46], we generated random scale-free networks. Unlike unstructured random graphs, in which the distribution of node degrees is the same for all nodes, the node degrees in a scale-free network follow a power law. In our experiments, we began with a complete graph over $m_0 = 4$ nodes. Then, new nodes were added, one at a time, where each

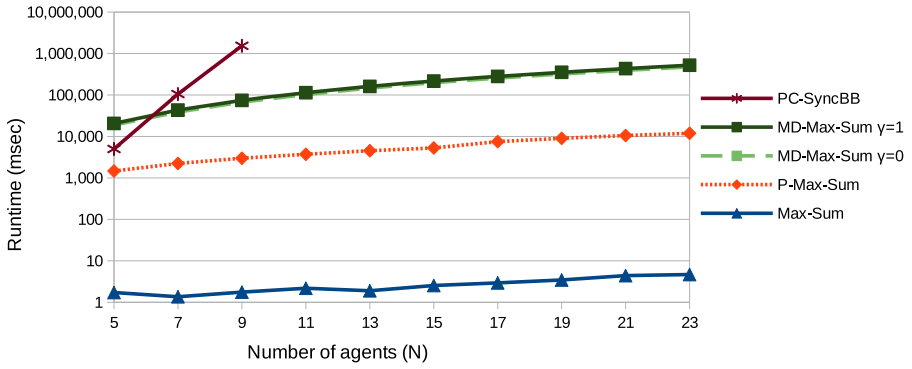


Fig. 9. Runtime on unstructured dense random graphs, $p_1 = 0.7$, varying N.

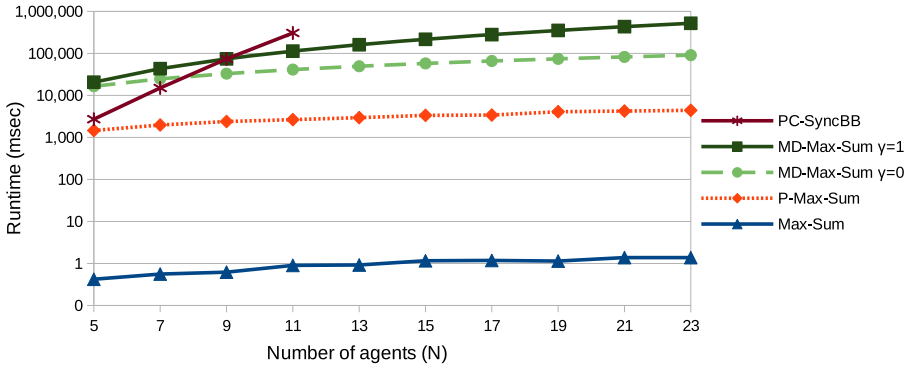


Fig. 10. Runtime on scale-free graphs ($m_0 = 4$, $m = 2$), varying N.

new node was randomly connected to $m = 2$ existing nodes with a probability that is proportional to the degrees of those nodes. We used such scale-free graphs with a number of nodes N that varies from 5 to 23.

Fig. 10 displays the result of this analysis. We can immediately observe that, unlike the previous experiments on unstructured random graphs, the performance gap between the two MD-Max-Sum variants increases significantly with N. The reason is that in unstructured random graphs the expected number of edges $|E|$, equals $p_1 \cdot \binom{N}{2}$, which is $\Theta(N^2)$, while in scale-free graphs $|E| = \binom{m_0}{2} + m \cdot (N - m_0) = \Theta(N)$. Consequently, the number of potential phantom edges in scale-free graphs, which is $\binom{N}{2} - |E|$, is greater, for large values of N, than that in unstructured random graphs. As a result, the gap between the two extreme variants of MD-Max-Sum is more evident in scale-free graphs. In such settings, one should tune γ in accordance with the characteristics of the application scenario. As for the performance gap between P-Max-Sum and the MD-Max-Sum variant with $\gamma = 0$, it remains similar to what we saw in previous experiments.

As noted above, the number of edges in scale-free graphs increases linearly with the number of nodes ($|E| = \Theta(N)$) compared to the quadratic increase in unstructured random graphs ($|E| = \Theta(N^2)$). Such a moderate increase of the constraint density in scale-free graphs is favorable by PC-SyncBB, the runtime of which is highly dependent on the constraint density, see Fig. 7. This explains the competitive runtimes of PC-SyncBB in scale-free graphs of up to 9 nodes. Nonetheless, PC-SyncBB reaches the cut-off limit after handling 11 agents (nodes) due to its strong dependence on the number of agents N.

Next, we evaluated the algorithms on distributed meeting scheduling problems, in a setting similar to the one described in [34]. In that setting, the problems are constructed similarly to the PEAV formulation [3], but instead of multiple-variable agents, we follow the *decomposition method* that is used in the P-SyncBB experimentation [22] to separate each variable into a *virtual agent* [47].

Inspired by the setting of Léauté and Faltings [28], we vary the number of meetings, while the number of participants per meeting is fixed to 2. Then, for each meeting, the participants are randomly drawn from a pool of 3 agents. Afterward, the objective is to select a time slot out of $|D_n| = 8$ options for each meeting. Both *time preference* and *meeting importance* are considered during the scheduling.

Fig. 11 presents the performance of the algorithms in the meeting scheduling setting. The trend is similar to previous experiments. In addition, we can see that MD-Max-Sum with topology privacy index $\gamma = 1$ reaches the cut-off time of 30 minutes when solving scheduling problems with 15 meetings or more, while the variant with $\gamma = 0$ can handle problems with 23 meetings without reaching this limit. This emphasizes the importance of the ability to control the trade-off between privacy and performance. Regarding PC-SyncBB, as in the case of scale-free graphs, it remains competitive for small

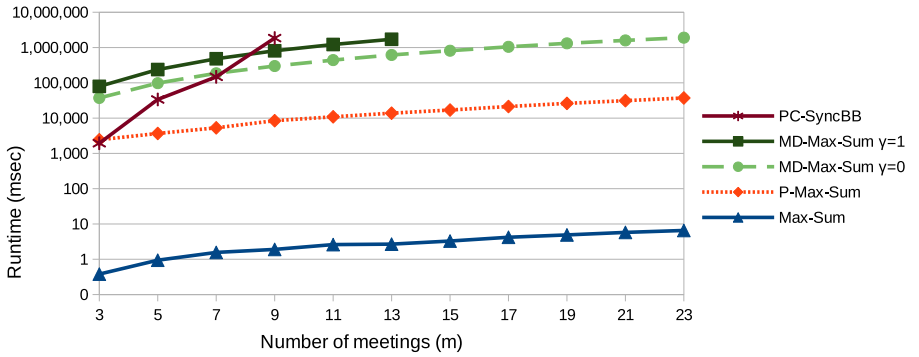


Fig. 11. Runtime of meeting scheduling problems, varying number of meetings.

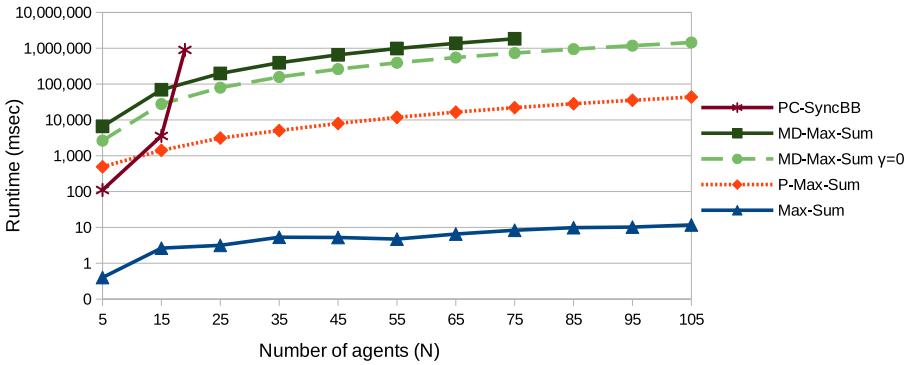


Fig. 12. Runtime of 3-color graph coloring problems ($p_1 = 0.4$), varying N.

problems; yet, for a different reason. The meeting scheduling problems herein include many *hard constraints* which are dictated by the PEAV formulation to enforce equality between variables of different agents that attend the same meeting. Hard constraints are an effective driver of pruning and thus are beneficial for Branch & Bound algorithms. Yet, PC-SyncBB reaches the cut-off limit after handling $m = 9$ meetings, again due to its strong dependence on the number of variables.

In the next experiment we evaluated the algorithms on 3-color graph coloring problems, similar to the setting described by Zivan et al. [48]. In this setting, for every $1 \leq n < m \leq N$, $C_{n,m}(x, y) = q$ if $x = y$ and $C_{n,m}(x, y) = 0$ if $x \neq y$, for some positive constant q. Fig. 12 presents the runtime of the algorithms on 3-color graph problems with $p_1 = 0.4$ and shows similar scalability properties to the previous experiments. The small domain size, $|D_n| = 3$, enables us to experiment with problems of larger sizes. For this experiment, we started with $N = 5$ and moved all the way to 105 agents in steps of 10. While all other algorithms remain within the cut-off limit of 30 minutes per single execution, the runtime of PC-SyncBB exceeded the cut-off limit already for $N = 20$. Hence, we include in Fig. 12 the runtime of PC-SyncBB for $N = 19$, which was the highest number of agents that could be processed within 30 minutes. While in all other experiments, we allocated a CPU thread per agent, in this experiment we used a 32-thread CPU even for higher values of N. Therefore, we can expect even better results for P-Max-Sum when the number of agents is greater than 32. This does not apply to MD-Max-Sum since the number of agents affects the constraint-graph's topology, but the number of mediators remains the same ($L = 5$); thus, the number of CPU threads is not a bottleneck. Like in the previous experiment, also here the MD-Max-Sum variant with topology privacy index $\gamma = 1$ reaches the cut-off time when solving problems with 85 agents or more, while the variant with $\gamma = 0$ solves up to 105 agents within this time frame.

Finally, we report the solution quality (cost) obtained by MD-Max-Sum and compare it to the optimal solution provided by PC-SyncBB. Since these two algorithms perfectly simulate Max-Sum and SyncBB, respectively, such comparisons have been studied in previous studies on Max-Sum and are, therefore, not directly required herein. Nonetheless, it is important to understand the price in solution quality when needing to decide between MD-Max-Sum and PC-SyncBB, especially given the substantial differences in runtimes, as shown in the above experiments.

Fig. 13 presents the average solution cost obtained after each of the first 50 iterations in unstructured random graphs with $N = 11$ agents (which are the largest problems that PC-SyncBB managed to compute in that setting, see Fig. 8). Fig. 14 presents the results of a similar experiment performed on scale-free graphs, again with $N = 11$ agents (see Fig. 10). In both experiments, the solution quality obtained by MD-Max-Sum is drastically improved in the first 10 iterations. However, the solution quality in subsequent iterations oscillates, which is a known phenomenon of Max-Sum for problems with cycles [49]. These results reinforce our choice to apply $K = 10$ iterations of MD-Max-Sum in most of the experiments. Similar trends were observed in other problem types.

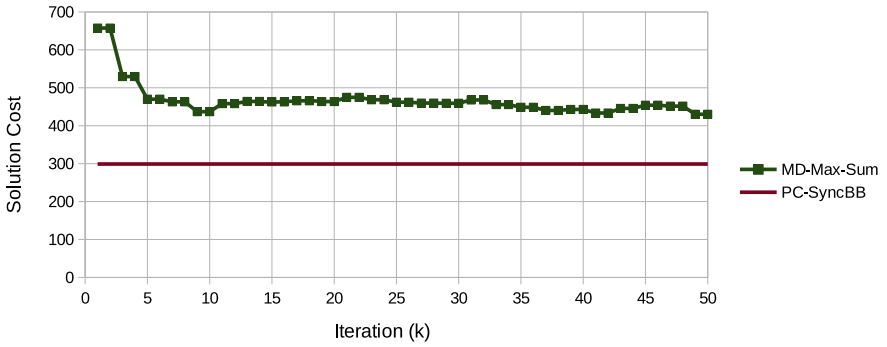


Fig. 13. Solution cost for unstructured random spare graphs ($p_1 = 0.3$, $N = 11$), varying number of iterations.

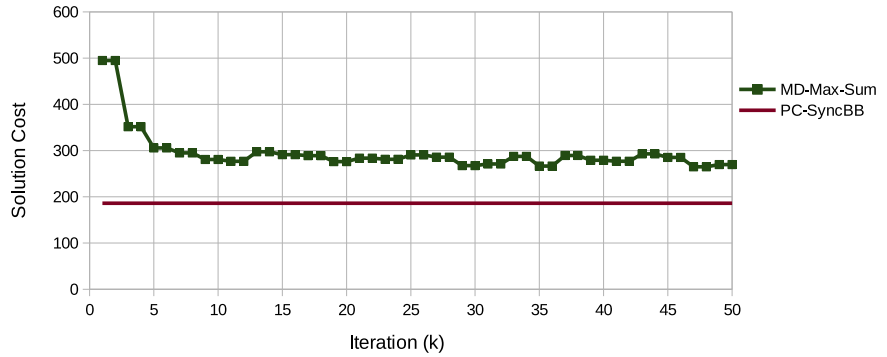


Fig. 14. Solution cost for scale-free random graphs ($m_0 = 4$, $m = 2$, $N = 11$), varying number of iterations.

In summary, we demonstrated that MD-Max-Sum can be applied to real-world problems in which privacy and security against coalitions are essential.

7. Conclusion

In this work, we introduced MD-Max-Sum, the first incomplete privacy-preserving DCOP algorithm that is also collusion-secure. It is an implementation of Max-Sum in the mediated model of computation. It preserves topology, constraint, and decision privacy. We analyzed the security and correctness of the algorithm and, using extensive experimentation, demonstrated its characteristics, its advantages over the only other collusion-secure DCOP algorithm, PC-SyncBB, and its viability.

Aside from the performance gains achieved by utilizing an incomplete algorithm (as opposed to PC-SyncBB that is based on a complete algorithm), the transition to the mediated model offers other significant benefits:

- The agents do not need to constantly communicate with each other. Such a feature may be most advantageous in settings where the agents do not have an efficient way to communicate among themselves.
- It allows the agents, that may run on computationally-bounded devices, to outsource costly and cryptographically-complex computations to dedicated servers. As a result, the agents' machines can be based on much cheaper hardware and their energy consumption could be reduced significantly.
- MD-Max-Sum is more robust than all previous DCOP algorithms in the following sense: if an agent goes offline (e.g., due to a technical failure) after secret sharing its private data to the mediators, the algorithm can still be executed and issue the correct outputs to all agents. However, the version of MD-Max-Sum that we presented here is not resilient to a failure of a mediator, because we took the maximal possible setting of the threshold, $t = \lfloor (L+1)/2 \rfloor$, see Eq. (5). However, if one reduces the threshold t to values smaller than $\lfloor (L+1)/2 \rfloor$, then our algorithm would be able to sustain a failure of up to $L - (2t-1)$ mediators. It should be noted that while reducing the value of the threshold t provides enhanced resilience, it also yields reduced security, since the algorithm is secure against coalitions of mediators of size up to $t-1$. Hence, the setting of the number of mediators L and of the threshold $t \leq \lfloor (L+1)/2 \rfloor$ should take into account the perceived chances of a mediator having a technical failure or a "moral" failure.

Before the mediated model can be practically assimilated in real settings, the involved agents need to first choose the set of mediators. That initial stage is out of the scope of this study. Nevertheless, there exist various trust and reputation models that can help the agents choose the most suitable and trusted mediators (cloud providers) for the task, e.g. [50–52].

According to the trustworthiness of the available cloud providers, the agents can decide on the number L of mediators and then choose the most appropriate cloud providers that will serve as the mediators.

A major bottleneck of MD-Max-Sum is the cryptographic MPC protocol behind the **MIN** sub-protocol. In our experiments, we used the implementation of Nishide and Ohta [41] for the Damgård and Nielsen [38] protocol. The computational problem that underlies **MIN** is as follows: given two integers a and b, which are secret-shared among a set of parties, those parties need to determine whether $a < b$ without recovering those values. That is a very basic problem in MPC, and it pops up in PC-SyncBB [34] as well as in many other application settings where privacy is of concern. Being a fundamental building block in MPC, secure comparison is a subject of extensive research in the cryptographic community, e.g. [53–56]. We intend to investigate the potential advantages of such alternative techniques of secure comparison (either existing ones or future ones) in the context of MD-Max-Sum. We note that the recent study of Makri et al. [54] reports promising improvements in terms of throughput from which MD-Max-Sum might benefit.

Furthermore, additional improvements can be made on the performance side by tasking multiple groups of mediators to compute in parallel different areas of the augmented constraint graph and synchronizing the intermediate results between iterations.

MD-Max-Sum simulates the original version of Max-Sum [4]. Since the introduction of that original version, there have been some advances in the research of Max-Sum. In particular, recent versions that include damping and splitting [57] demonstrate improved solution quality. We leave for future work the interesting challenge of efficiently implementing such versions of Max-Sum in the mediated model. Another direction for future work is to design a mediated version of the improved Max-Sum for DCOPs with constraints of general arity [58].

We believe that the mediated model of computation could be successfully implemented for other DCOP algorithms, in order to achieve enhanced privacy guarantees and to reap the advantages of the mediated model of computation as we have identified herein. But we believe that the advantages of mediated computing stretch well beyond the realm of DCOPs. In settings where the agents are running on devices with limited computational resources, or cannot efficiently communicate among themselves, they could benefit greatly from delegating the solution of their computational problems (that could be a DCOP or any other distributed computational task) to an external set of mediators that run on stronger platforms and could perform the computational task for the agents, in an oblivious manner, without sacrificing privacy.

Declaration of competing interest

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

Data availability

Data will be made available on request.

Acknowledgements

Pablo Kogan and Tal Grinshpoun were partially supported by the Ariel Cyber Innovation Center in conjunction with the Israel National Cyber Directorate in the Prime Minister's Office. The authors hereby declare that this support did not influence the work reported in this paper and that the funding source had no involvement in the project.

Appendix A. Privacy violations in Max-Sum

It is easy to see that all Q-messages that emerge from a variable node of degree 1 are always zero, as implied by their definition in Eq. (2), while Q-messages that emerge from a variable node of degree greater than 1 are typically non-zero in iteration k with $k \geq 2$. Hence, any agent A_i who controls the function node $X_{e_{i,j}}$ between its variable node X_i and another agent A_j's variable node X_j, will be able to tell whether the degree of X_j is 1 or greater than 1. Such leakage of information violates agent privacy (no agent should learn of the existence of agents with which it is not constrained) as well as topology privacy.

Moreover, the content of the Q-messages sent from X_j to $X_{e_{i,j}}$ in iteration $k = 2$ reveals to the function node, which A_i controls, information on the value of constraints of X_j vis-à-vis other variable nodes. If A_i learns, somehow, that X_j has degree 2 (namely, that X_j has, apart from X_i, only one additional neighbor, X_k), then the Q-message that X_j sends to $X_{e_{i,j}}$ in iteration 2 reveals the minimal entries in each row in the constraint matrix between X_j and the other variable node, X_k. In any case, such a leakage of information violates constraint privacy and it may expose to A_i, for example, that the magnitude of constraints between its variable and X_j is much smaller than the magnitude of constraints between X_j and other variable nodes.

Finally, if A_i learns that X_j has degree 1, as described above, it will know that $\overline{R}_j$ equals the last R-message sent from $X_{e_{i,j}}$ to X_j (see Eq. (4)) and, thus, it will know the final decision of X_j. Hence, also decision privacy is infringed.

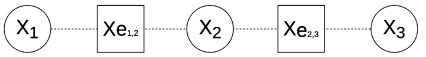


Fig. B.15. A sub-graph of the factor graph.

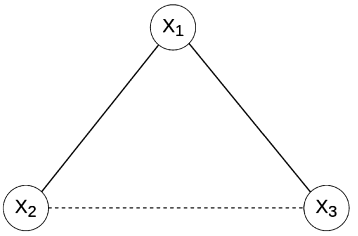


Fig. C.16. Regular and phantom edges.

Table C.3
The constraint matrices corresponding to all edges in the augmented constraint graph in Fig. C.16: the actual ones – $C_{1,2}$ and $C_{1,3}$, and the phantom one – $C_{2,3}$.

$C_{1,2}$	$X_2=0$	$X_2=1$
$X_1=0$	3	1
$X_1=1$	6	5

$C_{1,3}$	$X_3=0$	$X_3=1$
$X_1=0$	4	2
$X_1=1$	0	8

$C_{2,3}$	$X_3=0$	$X_3=1$
$X_2=0$	0	0
$X_2=1$	0	0

Appendix B. Privacy violating collusion attacks in P-Max-Sum

Motivated by the privacy issues that Max-Sum raises, as illustrated in Appendix A, Tassa et al. [22] introduced P-Max-Sum, a privacy-preserving version of Max-Sum. P-Max-Sum, like Max-Sum, is also executed as a distributed protocol that the agents execute on their own, but owing to the cryptographic machinery that it implements, it preserves topology, constraint, and decision privacy. However, P-Max-Sum is not secure against collusion: even if only two agents collude, they may learn sensitive private information on other agents. We proceed to demonstrate two such attacks. In doing so we use the same notations as in [22].

The P-Max-Sum algorithm assumes that for every agent A_n, all other agents, $\mathcal{A}_{-n} := \{A_1, \ldots, A_N\} \setminus \{A_n\}$, generate jointly a key pair in a homomorphic cipher $\mathcal{E}_n$, and that they keep the decryption key private from A_n (see [22, Section 3.2.1]). That encryption is then used in P-Max-Sum for securely generating shares in the Q- and R-messages in each iteration. If one of the agents in $\mathcal{A}_{-n}$ colludes with A_n and reveals to A_n the private decryption key in $\mathcal{E}_n$, then A_n will be able to recover all Q- and R-messages that are sent from and to its variable node X_n. As shown in Appendix A, the content of those messages can be used for violating privacy.

To illustrate another collusion attack on P-Max-Sum, let us assume that the factor graph has a sub-graph as shown in Fig. B.15. Assume that agents A_1 and A_3, who control X_1 and X_3, respectively, collude. If they compare the list of neighbors that they have, they will detect A_2 as a common neighbor. Then, A_1 and A_3 may combine messages that they have in order to infer whether X_2 has another neighboring variable node apart from them. To this end, A_1 and A_3 wait for A_2 to complete the execution of Protocol 1 in P-Max-Sum, which computes shares in the Q-messages that emerge from X_2 (see [22]). After the completion of that protocol, which involves A_2 as well as all of its neighboring agents, A_1 holds $S^{k+1,1}_{2\to e_{1,2}} = S^{k,3}_{e_{2,3}\to 2} + \sum_{j\in W} S^{k,j}_{e_{2,j}\to 2}$, and A_3 holds $S^{k+1,3}_{2\to e_{2,3}} = S^{k,1}_{e_{1,2}\to 2} + \sum_{j\in W} S^{k,j}_{e_{2,j}\to 2}$, where W is the set of indices of all neighbors of X_2 apart from X_1 and X_3. Next, A_1 sends to A_3 the value $S^{k+1,1}_{2\to e_{1,2}}$. If A_3 sees that it equals $S^{k,3}_{e_{2,3}\to 2}$, A_3 can infer, with very high probability, that $W = \emptyset$, namely, that A_2 is constrained only with A_1 and A_3. Otherwise, $W \neq \emptyset$, namely, X_2 has other neighbors in the factor graph. Thus, such collusion may infringe on topology privacy.

Appendix C. Exemplifying the secret sharing of the problem inputs among the mediators

Here we exemplify the first stage in MD-Max-Sum, in which the agents secret-share their private data among the mediators (as described in Section 5.2.1). To do that, we consider the following simple setting.

There are $N = 3$ agents and the domains of each of their three variables, X_1, X_2, X_3, are $D_1 = D_2 = D_3 = \{0, 1\}$. The constraint graph between them is shown in Fig. C.16 by the solid lines. Namely, X_1 is constrained with each of X_2 and X_3, but X_2 and X_3 are indifferent to each other. In order to hide the graph topology, agents A_2 and A_3 add a phantom edge between their variables, as shown by the dashed line in Fig. C.16. The constraint matrices are shown in Table C.3.

Table D.4
Shares in two secret values, their sum, product, comparison bit, and minimum.

	$a=4$	$b=9$	$a+b=2$	$ab=3$	$1_{a<b}=1$	$\min(a,b)=4$
s_1	1	3	4	6	7	1
s_2	1	1	2	6	3	5
s_3	4	3	7	3	0	5
s_4	10	9	8	8	9	1
s_5	8	8	5	10	8	4

Now, assume that there are $L=5$ mediators. Then the agents send shares in each of the constraint matrices to all L mediators, using t-out-of-L threshold secret sharing with $t=\lfloor (L+1)/2 \rfloor = 3$. Agent A_1 will distribute shares in each of the entries in the constraint matrices $C_{1,2}$ and $C_{1,3}$, while A_2 will distribute shares in each of the entries in the constraint matrix $C_{2,3}$.

We exemplify the secret sharing for only one entry – $C_{1,2}(0,0)=3$. Let us take the underlying field to be $\mathbb{Z}_p$ with $p=11$. Hence, A_1 generates a random polynomial of degree $t-1=2$, for which $f(0)=C_{1,2}(0,0)=3$. Let us assume that the chosen polynomial is $f(x)=3+4x+7x^2$. Then, A_1 will distribute to the mediator M_ℓ, $\ell \in \{1,2,3,4,5\}$, the share $f(\ell)$. Hence, the shares that the mediators will receive in $C_{1,2}(0,0)$ will be the following:

$$
\begin{aligned}
C^1_{1,2}(0,0) &= f(1) = 3+4\cdot 1+7\cdot 1^2 = 3 \mod 11\,,\\
C^2_{1,2}(0,0) &= f(2) = 3+4\cdot 2+7\cdot 2^2 = 6 \mod 11\,,\\
C^3_{1,2}(0,0) &= f(3) = 3+4\cdot 3+7\cdot 3^2 = 1 \mod 11\,,\\
C^4_{1,2}(0,0) &= f(4) = 3+4\cdot 4+7\cdot 4^2 = 10 \mod 11\,,\\
C^5_{1,2}(0,0) &= f(5) = 3+4\cdot 5+7\cdot 5^2 = 0 \mod 11\,.
\end{aligned}
$$

A_1 will perform similar secret sharing for all entries in $C_{1,2}$ until M_ℓ, $\ell \in [5]$, receives a full matrix of shares, $C^\ell_{1,2}$. The same procedure is applied for each of the constraint matrices.

Appendix D. Exemplifying MPC over secret shared values

The MPC computations that the mediators need to perform over secret shared values are the following: given secret shares of $a \in \mathbb{Z}_p$ and $b \in \mathbb{Z}_p$, compute secret shares in their sum $a+b$, in their product ab, in their comparison bit $1_{a<b}$, and in their minimum $\min(a,b)$. The details of those computations were discussed in Section 4. Here we exemplify the inputs and outputs of such computations.

We take the underlying field to be $\mathbb{Z}_p$ with $p=11$, and set the number of mediators to be $L=5$. Then $t:=\lfloor (L+1)/2 \rfloor = 3$ and, consequently, all secret-generating polynomials will be of degree $t-1=2$. Assume that $a=4$ and $b=9$ and that the secret-generating polynomials for them are $f_a(x)=4+x+7x^2$ and $f_b(x)=9+3x+2x^2$, respectively. Then the shares in a that are held by each of the five mediators are $s_\ell = f_a(\ell)$, $\ell \in [5]$, while the shares in b are $s_\ell = f_b(\ell)$; those shares are shown in the first two columns of Table D.4.

The shares in $a+b$ are computed locally by each of the mediators as the sum of shares in a and b. They are shown in the third column of Table D.4.

As for the shares in $ab=3$, they are computed by the MPC protocol that we described in Section 4.2. That protocol outputs shares in ab to each of the mediators. Those shares correspond to some random secret-generating polynomial for the value ab. That polynomial remains unknown to all parties. For the sake of illustration, assume that it is $f_{ab}(x) = 3+10x+4x^2$. Then the share that the protocol outputs to M_ℓ is $f_{ab}(\ell)$, $\ell \in [5]$. Those shares are shown in the fourth column of Table D.4.

We now turn to shares in the comparison bit $1_{a<b}=1$. The MPC protocol for that computation was described in Section 4.3. It yields a secret-generating polynomial for the value $1_{a<b}$, and that polynomial remains hidden from all parties. Let us assume that it is $f_{a<b}(x)=1+6x^2$. Then the share that the protocol outputs to M_ℓ is $f_{a<b}(\ell)$, $\ell \in [5]$. Those shares are shown in the fifth column of Table D.4.

Finally, to compute shares in $\min(a,b)=4$, the mediators execute the MPC protocol that was described in Section 4.4. That protocol generates a secret-generating polynomial for the value $\min(a,b)$. Assuming, for the sake of illustration, that it is $f_{\min}(x)=4+10x+9x^2$, the share that the protocol outputs to M_ℓ is $f_{\min}(\ell)$, $\ell \in [5]$, as shown in the sixth column of Table D.4.

Recall that under the assumption of an honest majority, all shared values remain unknown to all mediators. However, if a majority ($t=3$) of the mediators collude, they can use the shares that they hold in order to find the secret-generating polynomial and then the secret value, which is the free term in that polynomial. For example, if M_1, M_2, and M_3 collude and wish to recover ab, they will find f_{ab} by means of Lagrange interpolation:

$$
f_{ab}(x) = s_1 \cdot \frac{(x-2)(x-3)}{(1-2)(1-3)} + s_2 \cdot \frac{(x-1)(x-3)}{(2-1)(2-3)} + s_3 \cdot \frac{(x-1)(x-2)}{(3-1)(3-2)}
$$

$$= 6 \cdot 2^{-1} \cdot (x-2)(x-3) - 6 \cdot (x-1)(x-3) + 3 \cdot 2^{-1} \cdot (x-1)(x-2)$$
$$= 3 \cdot (x^2 - 5x + 6) - 6 \cdot (x^2 - 4x + 3) + 7 \cdot (x^2 - 3x + 2)$$
$$= 4x^2 + 10x + 3\,. \tag{D.1}$$

Then, they will infer that $ab = f_{ab}(0) = 3$.

Note that the computation that each of the agents performs at the termination stage of MD-Max-Sum (see Section 5.2.6) goes along the same lines as above. An agent may select any t out of the L shares that were received from the mediators and it will recover the same share-generating polynomial (and thus also the same secret), regardless of the selection made.

References

[1] K. Hirayama, M. Yokoo, Distributed partial constraint satisfaction problem, in: CP, 1997, pp. 222–236.
[2] F. Fioretto, E. Pontelli, W. Yeoh, Distributed constraint optimization problems and applications: a survey, J. Artif. Intell. Res. 61 (2018) 623–698.
[3] R.T. Maheswaran, M. Tambe, E. Bowring, J.P. Pearce, P. Varakantham, Taking DCOP to the real world: efficient complete solutions for distributed multi-event scheduling, in: AAMAS, 2004, pp. 310–317.
[4] A. Farinelli, A. Rogers, N.R. Jennings, Decentralised coordination of low-power embedded devices using the max-sum algorithm, in: AAMAS, 2008, pp. 639–646.
[5] F. Lezama, J. Palominos, A.Y. Rodríguez-González, A. Farinelli, E.M. de Cote, Agent-based microgrid scheduling: an ICT perspective, Mob. Netw. Appl. 24 (5) (2019) 1682–1698.
[6] P.J. Modi, W.-M. Shen, M. Tambe, M. Yokoo, ADOPT: asynchronous distributed constraint optimization with quality guarantees, Artif. Intell. 161 (2005) 149–180.
[7] R. Mailler, V.R. Lesser, Solving distributed constraint optimization problems using cooperative mediation, in: AAMAS, 2004, pp. 438–445.
[8] A. Petcu, B. Faltings, A scalable method for multiagent constraint optimization, in: IJCAI, 2005, pp. 266–271.
[9] S. Fitzpatrick, L. Meertens, Distributed coordination through anarchic optimization, in: Distributed Sensor Networks, Springer, 2003, pp. 257–295.
[10] R.T. Maheswaran, J.P. Pearce, M. Tambe, A family of graphical-game-based algorithms for distributed constraint optimization problems, in: Coordination of Large-Scale Multiagent Systems, Springer-Verlag, 2006, pp. 127–146.
[11] P. Rust, G. Picard, F. Ramparany, Using message-passing DCOP algorithms to solve energy-efficient smart environment configuration problems, in: IJCAI, 2016, pp. 468–474.
[12] F. Fioretto, W. Yeoh, E. Pontelli, A multiagent system approach to scheduling devices in smart homes, in: AAMAS, 2017, pp. 981–989.
[13] G. Picard, Auction-based and distributed optimization approaches for scheduling observations in satellite constellations with exclusive orbit portions, in: AAMAS, 2022, pp. 1056–1064.
[14] S. Krigman, T. Grinshpoun, L. Dery, Scheduling satellite timetables using DCOP, in: PATAT, vol. 3, 2022, pp. 121–137.
[15] I. Khakhiashvili, T. Grinshpoun, L. Dery, Course allocation with friendships as an asymmetric distributed constraint optimization problem, in: WI-IAT, 2021, pp. 688–693.
[16] J. Alwen, J. Katz, Y. Lindell, G. Persiano, A. Shelat, I. Visconti, Collusion-free multiparty computation in the mediated model, in: CRYPTO, 2009, pp. 524–540.
[17] J. Alwen, A. Shelat, I. Visconti, Collusion-free protocols in the mediated model, in: CRYPTO, 2008, pp. 497–514.
[18] T. Grinshpoun, A. Meisels, Completeness and performance of the APO algorithm, J. Artif. Intell. Res. 33 (2008) 223–258.
[19] A. Farinelli, A. Rogers, N.R. Jennings, Agent-based decentralised coordination for sensor networks using the max-sum algorithm, Auton. Agents Multi-Agent Syst. 28 (3) (2014) 337–380.
[20] R. Zivan, H. Yedidsion, S. Okamoto, R. Glinton, K. Sycara, Distributed constraint optimization for teams of mobile sensing agents, Auton. Agents Multi-Agent Syst. 29 (3) (2015) 495–536.
[21] P. Kogan, T. Tassa, T. Grinshpoun, Privacy preserving DCOP solving by mediation, in: CSCML, 2022, pp. 487–498.
[22] T. Tassa, T. Grinshpoun, R. Zivan, Privacy preserving implementation of the Max-Sum algorithm and its variants, J. Artif. Intell. Res. 59 (2017) 311–349.
[23] M. călin Silaghi, A comparison of distributed constraint satisfaction approaches with respect to privacy, in: DCR, 2002.
[24] M.C. Silaghi, D. Mitra, Distributed constraint satisfaction and optimization with privacy enforcement, in: IAT, 2004, pp. 531–535.
[25] M. Ben-Or, S. Goldwasser, A. Wigderson, Completeness theorems for non-cryptographic fault-tolerant distributed computation, in: STOC, 1988, pp. 1–10.
[26] P. Doshi, T. Matsui, M. Silaghi, M. Yokoo, M. Zanker, Distributed private constraint optimization, in: WI-IAT, 2008, pp. 277–281.
[27] J. Savaux, J. Vion, S. Piechowiak, R. Mandiau, T. Matsui, K. Hirayama, M. Yokoo, S. Elmane, M. Silaghi, Privacy stochastic games in distributed constraint reasoning, Ann. Math. Artif. Intell. 88 (7) (2020) 691–715.
[28] T. Léauté, B. Faltings, Protecting privacy through distributed computation in multi-agent decision making, J. Artif. Intell. Res. 47 (2013) 649–695.
[29] T. Grinshpoun, T. Tassa, P-SyncBB: a privacy preserving branch and bound DCOP algorithm, J. Artif. Intell. Res. 57 (2016) 621–660.
[30] S.M. Aji, R.J. McEliece, The generalized distributive law, IEEE Trans. Inf. Theory 46 (2) (2000) 325–343.
[31] T. Grinshpoun, T. Tassa, V. Levit, R. Zivan, Privacy preserving region optimal algorithms for symmetric and asymmetric DCOPs, Artif. Intell. 266 (2019) 27–50.
[32] H. Katagishi, J.P. Pearce, KOPT: distributed DCOP algorithm for arbitrary k-optima with monotonically increasing utility, in: DCR, 2007.
[33] C. Kiekintveld, Z. Yin, A. Kumar, M. Tambe, Asynchronous algorithms for approximate distributed constraint optimization with quality bounds, in: AAMAS, 2010, pp. 133–140.
[34] T. Tassa, T. Grinshpoun, A. Yanai, PC-SyncBB: a privacy preserving collusion secure DCOP algorithm, Artif. Intell. 297 (2021) 103501.
[35] A. Shamir, How to share a secret, Commun. ACM 22 (11) (1979) 612–613.
[36] O. Goldreich, S. Micali, A. Wigderson, How to play any mental game or A completeness theorem for protocols with honest majority, in: STOC, 1987, pp. 218–229.
[37] A.C. Yao, Protocols for secure computation, in: FOCS, 1982, pp. 160–164.
[38] I. Damgård, J.B. Nielsen, Scalable and unconditionally secure multiparty computation, in: CRYPTO, 2007, pp. 572–590.
[39] K. Chida, D. Genkin, K. Hamada, D. Ikarashi, R. Kikuchi, Y. Lindell, A. Nof, Fast large-scale honest-majority MPC for malicious adversaries, in: CRYPTO, 2018, pp. 34–64.
[40] I. Damgård, M. Fitzi, E. Kiltz, J.B. Nielsen, T. Toft, Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation, in: TCC, Springer, 2006, pp. 285–304.
[41] T. Nishide, K. Ohta, Multiparty computation for interval, equality, and comparison without bit-decomposition protocol, in: PKC, 2007, pp. 343–360.
[42] B. Lutati, I. Gontmakher, M. Lando, A. Netzer, A. Meisels, A. Grubshtein, AgentZero: a framework for simulating and evaluating multi-agent algorithms, in: Agent-Oriented Software Engineering, 2014, pp. 309–327.

[43] A. Meisels, E. Kaplansky, I. Razgon, R. Zivan, Comparing performance of distributed constraints processing algorithms, in: AAMAS, Citeseer, 2002, pp. 86–93.
[44] E.A. Sultanik, R.N. Lass, W.C. Regli, DCOPolis: a framework for simulating and deploying distributed constraint reasoning algorithms, in: AAMAS (demos), 2008, pp. 1667–1668.
[45] T. Tassa, T. Grinshpoun, A. Yanai, A privacy preserving collusion secure DCOP algorithm, in: IJCAI, 2019, pp. 4774–4780.
[46] A.-L. Barabási, R. Albert, Emergence of scaling in random networks, Science 286 (5439) (1999) 509–512.
[47] M. Yokoo, K. Hirayama, Algorithms for distributed constraint satisfaction: a review, Auton. Agents Multi-Agent Syst. 3 (2) (2000) 185–207.
[48] R. Zivan, S. Okamoto, H. Peled, Explorative anytime local search for distributed constraint optimization, Artif. Intell. 212 (2014) 1–26.
[49] R. Zivan, O. Lev, R. Galiki, Beyond trees: analysis and convergence of belief propagation in graphs with multiple cycles, in: AAAI, vol. 34, 2020, pp. 7333–7340.
[50] P.S. Pawar, M. Rajarajan, S.K. Nair, A. Zisman, Trust model for optimized cloud services, in: IFIP International Conference on Trust Management, Springer, 2012, pp. 97–112.
[51] F.Z. Filali, B. Yagoubi, Global trust: a trust model for cloud service selection, Int. J. Comput. Network Inform. Secur. 7 (5) (2015) 41.
[52] M. Chiregi, N.J. Navimipour, Cloud computing and trust evaluation: a systematic literature review of the state-of-the-art mechanisms, J. Electr. Syst. Inform. Technol. 5 (3) (2018) 608–622.
[53] O. Catrina, S. De Hoogh, Improved primitives for secure multiparty integer computation, in: SCN, Springer, 2010, pp. 182–199.
[54] E. Makri, D. Rotaru, F. Vercauteren, S. Wagh, Rabbit: efficient comparison for secure multi-party computation, in: FC, vol. 12674, 2021, pp. 249–270.
[55] H. Morita, N. Attrapadung, S. Ohata, S. Yamada, K. Nuida, G. Hanaoka, Tree-based secure comparison of secret shared data, in: ISITA, 2018, pp. 525–529.
[56] T.I. Reistad, Multiparty Comparison-an Improved Multiparty Protocol for Comparison of Secret-Shared Values, in: SECRYPT, vol. 1, SCITEPRESS, 2009, pp. 325–330.
[57] L. Cohen, R. Galiki, R. Zivan, Governing convergence of Max-sum on DCOPs through damping and splitting, Artif. Intell. 279 (2020) 103212.
[58] Y. Kim, V.R. Lesser, Improved max-sum algorithm for DCOP with n-ary constraints, in: AAMAS, 2013, pp. 191–198.