The Open University of Israel
Department of Mathematics and Computer Science

Privacy Preserving Solution of DCOPs by Mediation

A Thesis submitted in partial fulfillment
of the requirements for the degree of
M.Sc. in Computer Science
Department of Mathematics and Computer Science
The Open University of Israel

by

Pablo Kogan

Prepared under the supervision of
Prof. Tamir Tassa
Dr. Tal Grinshpoun (Ariel University)

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 neither to the problem inputs, nor 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, as well as partial agent 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. Such a model of computation may be beneficial well beyond the realm of DCOPs, and in particular in federated learning.

Contents

1 Introduction

A Distributed Constraint Optimization Problem (DCOP) [20, 13] 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 problems, such as meeting scheduling [27], sensor networks [12], and the Internet of Things [24].

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 [20], ADOPT [31], OptAPO [28], and DPOP [34]. 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 algorithms, such as DSA [15], MGM [26], and Max-Sum [12], 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 [27], 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 [14] and scheduling observations in satellite constellations [35]. 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* [2, 3]. Namely, there are external servers to whom the agents send the problem inputs in some protected manner. The external servers, whom we call *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]

[1]Note that the external mediators herein should not be confused with the mediators in the OptAPO algorithm [28, 17], which are regular (internal) agents that perform computations in their neighborhood as an inherent part of the algorithm.

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

1. It protects the privacy of the agents, since the agents do not exchange any messages between themselves, while the mediators work on protected values (say, on secret sharing of the input values, or on homomorphic encryption of those values).

2. In settings where the agents cannot efficiently communicate among themselves, it is hard to run DCOP algorithms, because such algorithms 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 or the expertise to run the DCOP algorithm may benefit from delegating that computation to external servers who will perform it for them in a secure manner.

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 [12]. In MD-Max-Sum, the agents send to the mediators secret shares [37] in their private inputs. The mediators proceed to execute the Max-Sum computations on the shares of the problem inputs, while remaining oblivious to the value of the underlying inputs. At the end, the mediators send to the agents messages from which the agents may infer the assignments to their variables.

We show that if the mediators have an honest majority, then MD-Max-Sum provides constraint, topology, and decision privacy, as well as partial agent 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. In section 5 we provide two collusion attacks on P-Max-Sum [42], a previously introduced privacy-preserving, yet not collusion-secure, DCOP algorithm that is based on Max-Sum; the attacks demonstrate the importance of collusion-secure algorithms. Section 6 holds the main part of this work – the description and analysis of MD-Max-Sum, our novel privacy-preserving collusion-secure DCOP algorithm. Section 7 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 8.

2 Related work

The study of privacy in the context of Distributed Constraint Satisfaction Problems (DCSPs) started with the work of Silaghi and Faltings [9], who compared different solution techniques and arranged them in a hierarchy according to their corresponding level of privacy loss. DCSPs are a private 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 [38] 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 [5], and it was applicable only to small problems due to its dependency on an exhaustive search over all possible full assignments.

In order to better describe the privacy guarantees of DCOP algorithms, Léauté and Faltings [23] suggested four notions of privacy for the DCOP framework: agent, constraint, topology, 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 [18] 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 [1] Max-Sum algorithm [12]. 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. [42] developed the P-Max-Sum algorithm, which runs Max-Sum with cryptographic enhancements in order 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 constraint, topology, and decision privacy, and it may be extended to provide also agent privacy. Grinshpoun et al. [19] devised another incomplete privacy-preserving algorithm, P-RODA, which is based on region optimality [21, 22]. 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 Section 5 we demonstrate such possible attacks in the context of P-Max-Sum). To address the risk of collusion, Tassa et al. [41] introduced PC-SyncBB, the first privacy-preserving DCOP algorithm that is *collusion-secure.* It is secure under the assumption that is known in cryptogarphy as *honest majority*, namely, that the number of colluding agents is smaller than the number of agents outside the coalition. 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 [41] 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 entirely by the external mediators. The advantages of that model of computation are discussed in the Introduction and in Section 8.

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) [20] 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.[2] 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.

[2]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. [31, 34].

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. [31, 34]. As the domains are finite, they me 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 Figure 1.

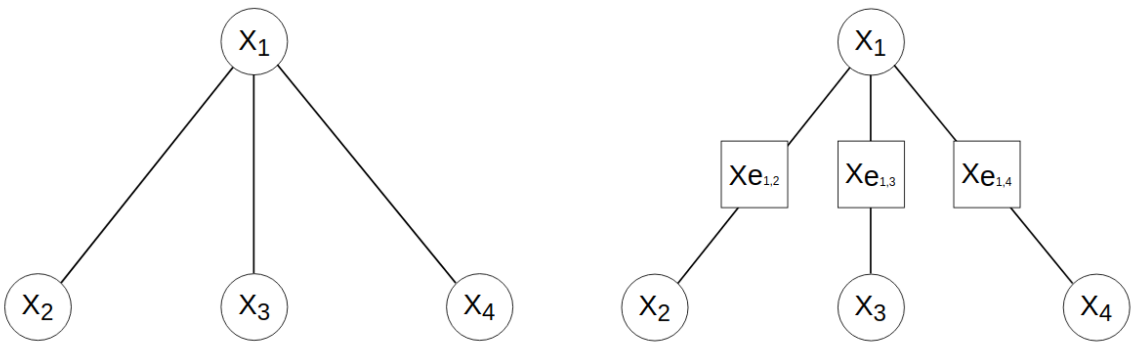

Figure 1: A constraint graph G of a DCOP with 4 variable nodes (left) and the corresponding factor graph G' that has 4 variable nodes and 3 function nodes (right).

Léauté and Faltings [23] 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.

3.2 The Max-Sum Algorithm

The Max-Sum algorithm [12] 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 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

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

Fixing a function node X_e, where $e = (X_n, X_m)$, then for each $x \in D_n$,

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

while for each $y \in D_m$,

$$R^{k+1}_{e\to m}(y) := \min_{x \in D_n} \left[C_{n,m}(x,y) + Q^k_{n\to e}(x) \right] \,. \tag{4}$$

Finally, after completing a preset number K of iterations, each variable node X_n computes $\overline{R}_n := \sum_{X_e \in V_n} R^K_{e\to n}$ 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 $\alpha^{k+1}_{n\to e} := \min_{x \in D_n} Q^{k+1}_{n\to e}(x)$ [12].

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. Secure comparison allows the parties to compare secret-shared values while preventing the participants from learning anything else (Section 4.3).

4.1 Shamir's Secret Sharing

Secret sharing schemes [37] 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 [37], 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 less 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 $\mathsf{Reconstruct}$:

- $\mathsf{Share}_{t,L}(x)$. 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, $f(x) = s + a_1x + a_2x^2 + \ldots + a_{t-1}x^{t-1}$, 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.

4.2 Secure Multiparty Computation

A Secure Multiparty Computation (MPC) protocol [16, 43] 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 circuits in which each gate g has exactly two predecessor gates. Let α_l denote the output value of g_l, the left predecessor of g, and α_r denote the output value of g_r, the right predecessor of g. Then the output of g is a simple function of those two values, $g(\alpha_l, \alpha_r)$.

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 operation gate g, once both g_l and g_r are assigned values, say α_l and α_r, respectively, the gate g 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 $C(x_1, \ldots, x_L)$.

Two main types of circuits are discussed in the MPC literature: an *arithmetic* circuit, meaning that 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); and a *Boolean* circuit, meaning that the values assigned to each gate are from $\{0, 1\}$, and the operation gates are either the logical **XOR** or **AND** functions. Both types of circuits can express any function. Since the MPC computation that we will need to perform in the course of MD-Max-Sum is secure comparison of secret-shared values, we will focus hereinafter on arithmetic circuits, which are more suitable for that purpose.

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 Figure 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.

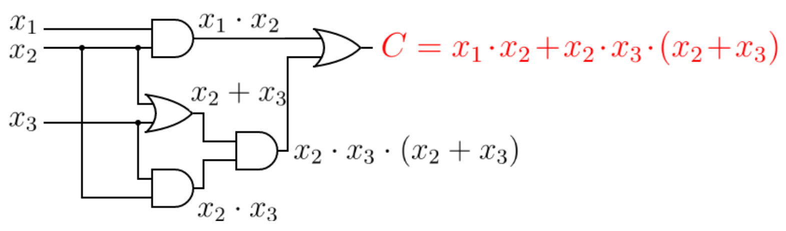
x_1
x_2
x_3
$x_1 \cdot x_2$
$x_2 + x_3$
$x_2 \cdot x_3 \cdot (x_2 + x_3)$
$x_2 \cdot x_3$
$C = x_1 \cdot x_2 + x_2 \cdot x_3 \cdot (x_2 + x_3)$

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

4.2.2 Secret-sharing-based MPC

Here we describe a general approach to perform 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 Figure 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 computing 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.

Below, we describe the generic secret-sharing-based protocol of Damgård and Nielsen [11]. 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. [7].

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 Comparison

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. (For example, since $\max(a,b) = a + 1_{a<b} \cdot (b-a)$, then it is possible to compute a t-sharing of $\max(a,b)$, using the t-sharings of a and b, a t-sharing of $1_{a<b}$, and the emulation of arithmetic circuits that we described in Section 4.2.2.)

Secure comparison is an area of active study, and improvements are made rapidly.

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

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

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. [10] by means of *bit-decomposition*. It required $O(\ell \cdot \log(\ell))$ multiplications, where $\ell = \log_2(p)$. A year later, Nishide and Ohta [33] 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 $(2q \bmod p)$ is zero.*

Proof. If $q < \frac{p}{2}$ then $2q < p$. Hence, $2q \bmod p = 2q$ (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, $2q \bmod p = 2q - p$. Since that number is odd, its LSB is 1. $\square$

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 use those shares in order to compute shares in the corresponding LSB. The reader is referred to [33] 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 [10] that was based on bit decomposition required 44 rounds and $205\ell + 188 + \log_2 \ell$ multiplications.

5 Privacy violating collusion attacks

The original Max-Sum algorithm is implemented by the agents themselves who perform all computations and communicate with each other. As was shown by Tassa et al. [42], such an implementation of Max-Sum may leak sensitive information on the private inputs of the agents. Motivated by the privacy issues that Max-Sum raises, they introduced P-Max-Sum, a privacy-preserving version of Max-Sum. That algorithm is still 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 demonstrate below two such attacks. In doing so we use the same notations as in [42].

5.1 Collusion attack 1: Sharing private keys

Consider the setting depicted in Figure 3 and assume that A_1 and A_2, the agents corresponding to the two neighboring variable nodes X_1 and X_2, decide to collude. Then A_1 can send to A_2 the private key in $\mathcal{E}_2(\cdot)$ (which is known to all agents except A_2). After doing so, A_2 will be able to recover all Q-messages that emerge from the variable node X_2 and also all R-messages that are sent to X_2. Now, assume that X_2 has in G the neighboring variable node X_3, and that X_3 has no other neighbors apart from X_2. We claim that A_2, who now can recover all messages that its variable node sends and receives, can learn that X_3 has degree 1 in G. Indeed, since X_3 has degree 1, all Q-messages that X_3 sends to $X_{e_{2,3}}$ are zero. Consequently, the R-messages from $X_{e_{2,3}}$ to X_2 will be the same in all iterations and will equal $R_{e_{2,3}\to 2}(x) := \min_{y\in D_3} [C_{2,3}(x,y)]$. Thus, by examining the R-messages, A_2 can easily deduce the degree of X_3.

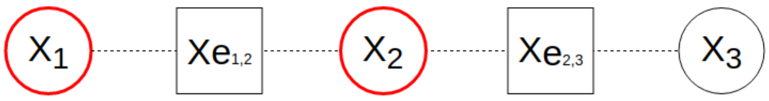

Figure 3: Sharing private keys between X_1 and X_2 to attack X_3.

5.2 Collusion attack 2: Message recombination

Consider the setting depicted in Figure 4 and 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 X_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 [42]). After the completion of that protocol, that 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. In our case, $W = \{4\}$. Next, A_1 sends to A_3 the value $S^{k+1,1}_{2\to e_{1,2}}$. If $W = \emptyset$ then A_3 would see that it equals $S^{k,3}_{e_{2,3}\to 2}$. Such equality implies, with very high probability, that indeed $W = \emptyset$. If, similarly, A_1 sees that $S^{k+1,3}_{2\to e_{2,3}}$ equals $S^{k,1}_{e_{1,2}\to 2}$, then A_1 and A_3 can safely infer that the degree in G of the variable node X_2 is two.

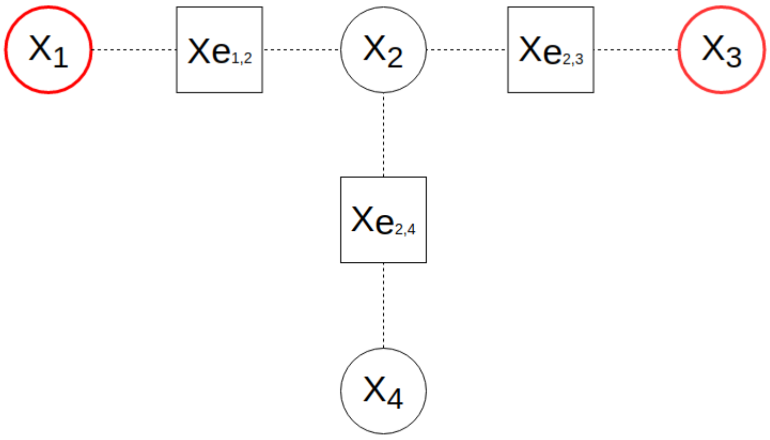

Figure 4: Combining messages from X_1 and X_3 to attack X_2

Thus, while P-Max-Sum does respect topology privacy, this privacy holds as long as their is no collusion among the agents. Even if only two agents collude, they may infer private topology information on other agents, as shown here.

6 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 6.1). As demonstrated in Corollary 4, running Max-Sum over a constraint graph that is augemented 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.

6.1 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 6.2, 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. Max-Sum typically operates over the reals, $\mathbb{R}$. However, one may assume that all constraints are rational numbers and, consequently, it is possible to translate all of them to integers so that Max-Sum can operate over the integers $\mathbb{Z}$. Hence, 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.

Theorem 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 $Q^k_{n\to e}$, $R^k_{n\to e}$, $Q^k_{+,n\to e}$ and $R^k_{+,n\to e}$ denote the Q- and R-messages between those two nodes in the kth iteration of Max-Sum *when operating on G' and G'_+, respectively. Then $Q^k_{+,n\to e} \sim Q^k_{n\to e}$ and $R^k_{+,n\to e} \sim R^k_{n\to e}$.*

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

$$Q^{k+1}_{+,n\to e} = \sum_{X_f\in V_n\setminus\{X_e\}} R^k_{+,f\to n}\,, \tag{8}$$

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

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

By induction, the vector sum on the right-hand side of Eq. (9) 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. (9) is a constant vector. That vector is given by

$$R^k_{+,e_+\to n}(x) = \min_{y\in D_m} \left[C_{n,m}(x,y) + Q^k_{+,m\to e_+}(y)\right]. \tag{10}$$

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. $\square$

Corollary 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 Theorem 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. $\square$

6.1.1 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.

6.2 The MD-Max-Sum algorithm

We assume that all agents 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.

6.2.1 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, $1 \leq n \leq N-1$, shares with the mediators $\mathbf{M}$ the constraint matrices $C_{n,m} \in \mathcal{C}$ for all $n < m \leq N$, 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 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

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

After each mediator M_ℓ, $\ell \in [L]$, got its share matrix $C^\ell_{n,m}$ for all $1 \leq n < m \leq N$, they have all problem inputs and they may now begin an MPC emulation of Max-Sum over those inputs.

We assume that the set of mediators have an *honest majority.* That means that some of the mediators may collude in order to combine the pieces of information that they got in attempt to extract private information; however, the number of colluding agents is smaller than $L/2$ under the assumption of 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 $t = \lfloor (L+1)/2 \rfloor \geq L/2$, 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, 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). Figure 5 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

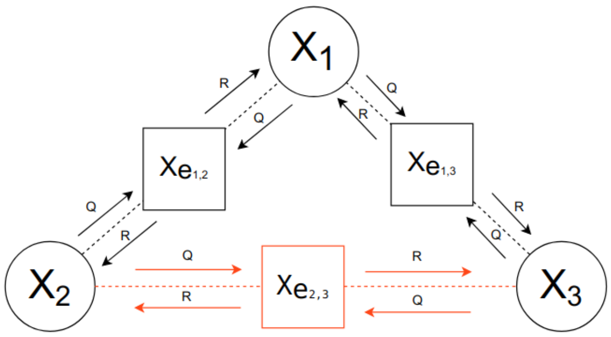

Figure 5: An Augmented Factor Graph: $X_{e_{1,2}}$ and $X_{e_{1,3}}$ are actual function nodes; $X_{e_{2,3}}$ is a phantom function node.

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 $m \in [N] \setminus \{n\}$.

6.2.2 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,

$$Q^0_{n\to e} = (0, \ldots, 0) \in \mathbb{Z}_p^{|D_n|}\,, \quad R^0_{e\to n} = (0, \ldots, 0) \in \mathbb{Z}_p^{|D_n|}\,. \tag{11}$$

(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:

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

Note that no interaction between the mediators is needed at this stage, and that the L vector shares of the Q-messages in Eq. (12) are t-out-of-L vector shares in the zero Q-messages in Eq. (11), and likewise for the R-vectors.

6.2.3 Producing shares in Q-messages

In iteration $k+1$, the mediators have to emulate the message $Q^{k+1}_{n\to e}$ 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:

$$Q^{k+1,\ell}_{n\to e} := \sum_{X_f \neq X_e} R^{k,\ell}_{f\to n}\,. \tag{13}$$

Note that in Eq. (2), the sum on the right-hand side was over all function nodes in G' that are neighbors of the variable node X_n, except for X_e. In Eq. (13), on the other hand, the sum goes over all $N-2$ function nodes, X_f, between X_n and X_i for any $i \in [N] \setminus \{n, m\}$. This extended summation is a result of the fact that

MD-Max-Sum operates on the augmented factor graph (where there is a function node, genuine or phantom, between every pair of variable nodes) and not on the actual factor graph, as Max-Sum does.

6.2.4 Producing shares in R-messages

Here, we concentrate on the more involved task of computing t-out-of-L shares in the R-messages, $R^{k+1}_{e\to n}$, 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,

$$R^{k+1}_{e\to n}(x) := \min_{y\in D_m} B^k_{n,m}(x,y)\,, \quad x \in D_n\,, \tag{14}$$

where $B^k_{n,m}(x,y)$ denotes the sum

$$B^k_{n,m}(x,y) := C_{n,m}(x,y) + Q^k_{m\to e}(y)\,. \tag{15}$$

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 $C^\ell_{n,m}(x,y)$, $\ell \in [L]$), 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 $Q^k_{m\to e}(y)$ for all $y \in D_m$, where M_ℓ's shares are denoted $Q^{k,\ell}_{m\to e}(y)$. Hence, $C^\ell_{n,m}(x,y) + Q^{k,\ell}_{m\to e}(y)$, which we denote by $B^{k,\ell}_{n,m}(x,y)$, are t-out-of-L shares in $B^k_{n,m}(x,y)$, as implied by Eq. (15) 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. (14) from the shares that the mediators hold in each of the terms on the right-hand side of Eq. (14). 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 $R^{k+1,\ell}_{e\to n}$ in $R^{k+1}_{e\to n}$. 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 $M_\ell$, $\ell \in [L]$, holds a $t$-out-of-$L$ share, $B^{k,\ell}_{n,m}(x,y)$, in
       $B^k_{n,m}(x,y)$, for every $x \in D_n$ and $y \in D_m = \{y_1,\dots,y_{|D_m|}\}$.
1 forall $x \in D_n$ do
2     $M_\ell$ sets $\beta^\ell_{n,m}(x) \leftarrow B^{k,\ell}_{n,m}(x,y_1)$
3     forall $j = 2,\dots,|D_m|$ do
4         if COMPARE($\{B^{k,\ell}_{n,m}(x,y_j)\}_{\ell\in[L]}, \{\beta^\ell_{n,m}(x)\}_{\ell\in[L]}$) = true then
5             $M_\ell$ sets $\beta^\ell_{n,m}(x) \leftarrow B^{k,\ell}_{n,m}(x,y_j)$
6     $M_\ell$ sets $R^{k+1,\ell}_{e\to n}(x) \leftarrow \beta^\ell_{n,m}(x)$
Output: Mediator $M_\ell$, $\ell \in [L]$, gets a $t$-out-of-$L$ share $R^{k+1,\ell}_{e\to n}(x)$ in $R^{k+1}_{e\to n}(x)$.
```

The external loop in the protocol (lines 1-6) is over all values x in the domain D_n, i.e., over all entries in the vector message $R^{k+1}_{e\to n}$. For each such $x \in D_n$, the mediators have to find the minimum among $\{B^k_{n,m}(x,y) : y \in D_m\}$, 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 $\beta^\ell_{n,m}(x)$, $\ell \in [L]$. First (line 2), each mediator initiates its β-shares with the shares corresponding to $B^k_{n,m}(x,y_1)$. Then (lines 3-5), for each y_j, $j = 2,\ldots,|D_m|$, the mediators compare $B^k_{n,m}(x,y_j)$ to the current minimum, in which they have t-out-of-L shares in $\beta^\ell_{n,m}(x)$, $\ell \in [L]$. The comparisons are performed in a secure manner by invoking a distributed sub-protocol that all mediators jointly execute, as will be explained below. If $B^k_{n,m}(x,y_j)$ is smaller than the current minimum, then each mediator updates its share of the minimum (line 5).

In order to perform comparisons between values that are known to the mediators only through t-out-of-L shares, without recovering those values and perform the comparison over those recovered values, Protocol 1 calls upon an MPC sub-protocol called **COMPARE** (line 4), which all the mediators run together in a distributed manner. That sub-protocol assumes that the mediators have t-out-of-L shares in two values $x, y \in \mathbb{Z}_p$; it returns **true** if $x < y$ (when x and y are interpreted as integers) and **false** otherwise. Such a sub-protocol was described earlier in Section 4.3. It is perfectly secure in the sense that it reveals to the mediators nothing about the two compared values beyond the final output bit which indicates which of the two is smaller.

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

6.2.5 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 $Q^{k+1}_{n\to e}$ the value $\alpha^{k+1}_{n,m} := \min_{x\in D_n} Q^{k+1}_{n\to e}(x)$ (see [12]). To perform such normalization, we need to find for each message $Q^{k+1}_{n\to e}$ the minimum entry $\alpha^{k+1}_{n,m}$, and then to subtract it from each of the entries in $Q^{k+1}_{n\to e}$.

To find the minimal entry in $Q^{k+1}_{n\to e}$, a vector of dimension $|D_n|$, it is necessary to perform $|D_n| - 1$ comparisons. As the mediators hold shares in each of those entries, they may apply the **COMPARE** sub-protocol $|D_n| - 1$ 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 $Q^{k+1}_{n\to e}$.

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

$$q = \max_{1\le n<m\le N} \max_{x\in D_n, y\in D_m} C_{n,m}(x,y) \tag{16}$$

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 [42, 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

$$d^h < \Gamma := \frac{(d-1)p}{q} + 1\,, \tag{17}$$

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.

6.2.6 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 $\overline{R}_n = \sum_{X_e \in V_n} R^K_{e \to n}$. 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 $\overline{R}_n$. Afterwards, A_n finds the minimal entry in $\overline{R}_n$ and then assigns the corresponding value to X_n.

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

```
1 Initial sharing of all topology and constraint information (Section 6.2.1)
2 Perform the initial $k = 0$ iteration (Section 6.2.2)
3 forall $k = 1, \ldots, K$ do
4     forall $(X_n, X_e) \in E'_+$ do
5         Compute $t$-out-of-$L$ shares in the $Q^k_{n \to e}$-messages (Section 6.2.3)
6         Compute $t$-out-of-$L$ shares in the $R^k_{e \to n}$-messages (Section 6.2.4)
7     if $(k \mod K_1) = 0$ then
8         Normalize messages (Section 6.2.5)
9 Terminate (Section 6.2.6)
```

6.3 Correctness and privacy

In this section we provide proofs for the correctness and privacy guarantees of MD-Max-Sum. Namely, Theorem 6 and Corollary 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.

Theorem 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 kth 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 kth iteration in* MD-Max-Sum *between those two nodes.*

Then $\left\{Q^{\ell,k}_{+,n\to e} : \ell \in [L]\right\}$ *are t-out-of-L shares in* $Q^k_{+,n\to e}$ *and, similarly,* $\left\{R^{\ell,k}_{+,e\to n} : \ell \in [L]\right\}$ *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 6.2.3, 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 6.2.4, and the linearity of secret sharing and the correctness of the **COMPARE** sub-protocol, imply that $\left\{R^{\ell,k}_{+,e\to n} : \ell \in [L]\right\}$ are t-out-of-L shares in $R^k_{+,e\to n}$. □

Corollary 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 Theorem 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 Theorem 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. □

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 constraint and topology information, as encoded in the matrices $\mathcal{C} := \{C_{n,m} : 1 \le n < m \le N\}$, 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 6.2.6) 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 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 6.1.1, but will provide better runtimes, as demonstrated in our experimental evaluation, see Section 7.

A note on agent privacy. As discussed in Section 6.2.1, the agents send to the mediators shares in all of the matrices in $\mathcal{C} := \{C_{n,m} : 1 \leq n < m \leq N\}$. Hence, the agents must know the overall number of agents N, as well as the domain sizes of all variables. Beyond that, each agent must know its identifying index $n \in [N]$. The mediators could order the agents and then notify each agent of its identifying index, so that the agents do not need to communicate with each other. Hence, MD-Max-Sum can provide partial agent privacy: the agents do learn N and $|D_n|$ for all $n \in [N]$, but no agent will learn anything else about agents with whom it is not constrained.

6.4 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 6.2.1), then the algorithm iterates for a preset number of iterations (Sections 6.2.2–6.2.5), and finally the mediators send to each of the agents a vector from which the agents can infer the assignments to their variables (Section 6.2.6). 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. (12) and (13)). The only parts in which the mediators engage in a costly MPC sub-protocol, **COPMARE**, is in computing shares in the R-messages (Section 6.2.4) and in normalizing messages (Section 6.2.5).

In Protocol 1, when executed on the pair of nodes X_n and X_e, where $e = (X_n, X_m)$, the **COMPARE** 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 **COMPARE** 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 **COMPARE** sub-protocol is needed to be executed $|D_n| - 1$ times. Hence, the overall number of **COMPARE** 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 **COMPARE** sub-protocol is bounded by

$$2|E_+|(\Delta - 1) \cdot (\Delta K + \lfloor K/K_1 \rfloor) \ . \tag{18}$$

Since in the most secure version of MD-Max-Sum we have $|E_+| = \binom{N}{2}$, the bound in Eq. (18) is at most

$$N(N-1)(\Delta-1) \cdot (\Delta K + \lfloor K/K_1 \rfloor) \ . \tag{19}$$

As for the computational and communication costs of the **COMPARE** sub-protocol, see the theoretical discussion in Section 4.3 and the experimental evaluation in Section 7.1.

7 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 [25], running on AWS C5a instances comprised of a 2nd generation AMD EPYC™ 7R32 processor and 64 GB memory, except for the call to the **COMPARE** sub-protocol, which is described and evaluated separately in Section 7.1.

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) [30], 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 [39] instead.

In Section 7.1 we evaluate the runtime of the **COMPARE** sub-protocol, which is a fundamental and computationally expensive component of MD-Max-Sum. Then, in Section 7.2, 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 7.3, we compare the performance of MD-Max-Sum with other algorithms: the basic (not-private) Max-Sum, the P-Max-Sum algorithm [42] that preserves privacy, but is not collusion-secure, and PC-SyncBB [40, 41], which is the only other privacy-preserving DCOP algorithm that is collusion-secure.

7.1 The COMPARE sub-protocol

The **COMPARE** sub-protocol was executed using the secure comparison algorithm of Nishide and Ohta [33] (which we described in Section 4.3), as implemented by the generic secret-sharing-based protocol of Damgård and Nielsen [11] with the performance improvements of Chida et al. [7]. 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 **COMPARE** sub-protocol.

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

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

7.2 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 6.1, 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 Figure 6. 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 Figure 7, when L increases, the runtime increases too, in consistency with the runtimes of the **COMPARE** sub-protocol, as reported in Section 7.1.

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

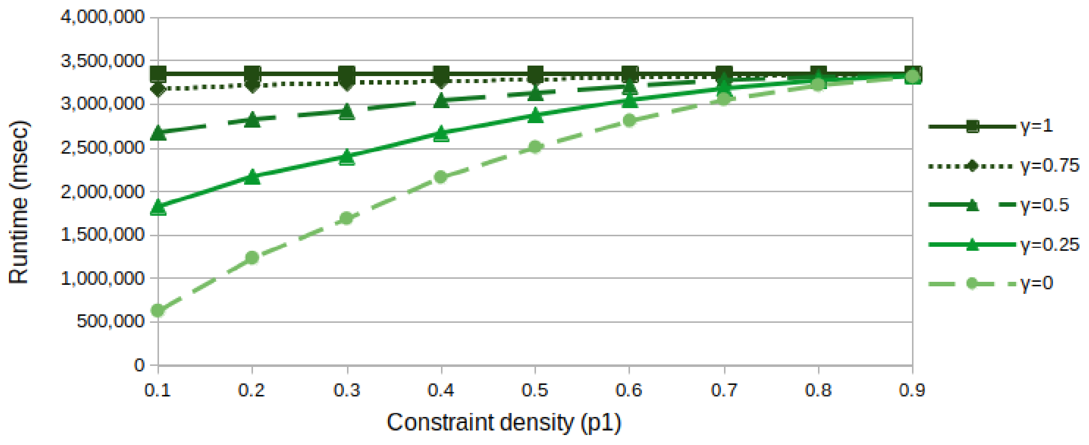

Figure 6: The effect of the topology privacy index γ on MD-Max-Sum's runtime: Unstructured random graphs, $N = 24$ agents, varying constraint density.

$L = 5$, since a protocol with that number of mediators is secure as long as the number of colluding mediators is smaller than 3.

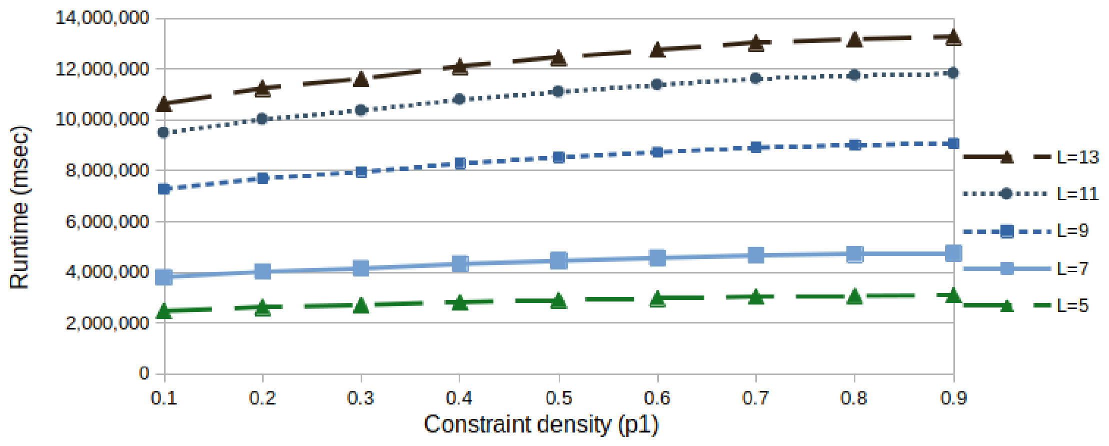

Figure 7: The effect of the number of mediators L on MD-Max-Sum's runtime: Unstructured random graphs, $N = 24$ agents, varying constraint density.

7.3 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. 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 [42] (provides privacy, but not against coalitions). As shown in [42], and herein in Corollary 4, 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 [42].

In addition, we included in our experiment the PC-SyncBB algorithm [40, 41], 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 Figure 8. 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 is the price of completeness.

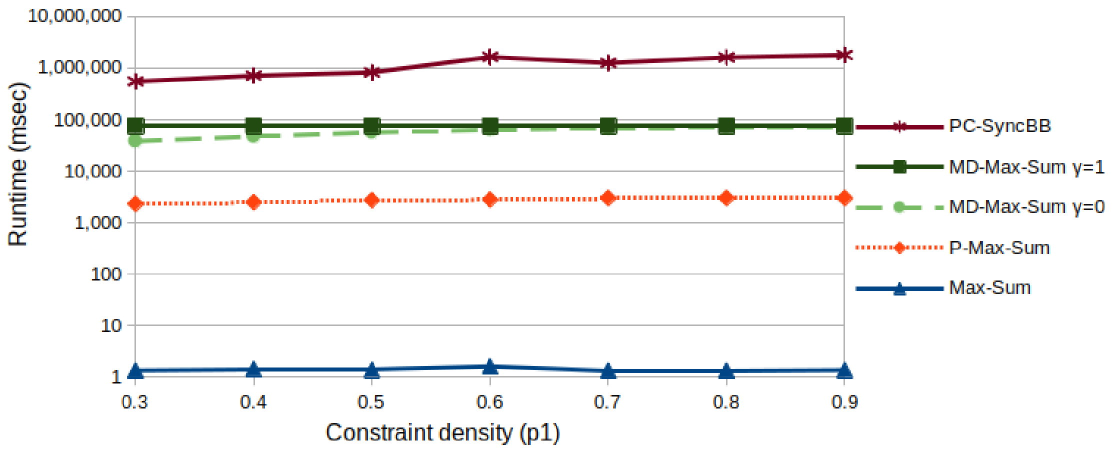

Figure 8: Unstructured random graphs, $N = 9$ agents, varying constraint density.

In the next experiment, shown in Figure 9, 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 P-Max-Sum 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 [41]. 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 **COMPARE**, since N affects the size of the graph.)

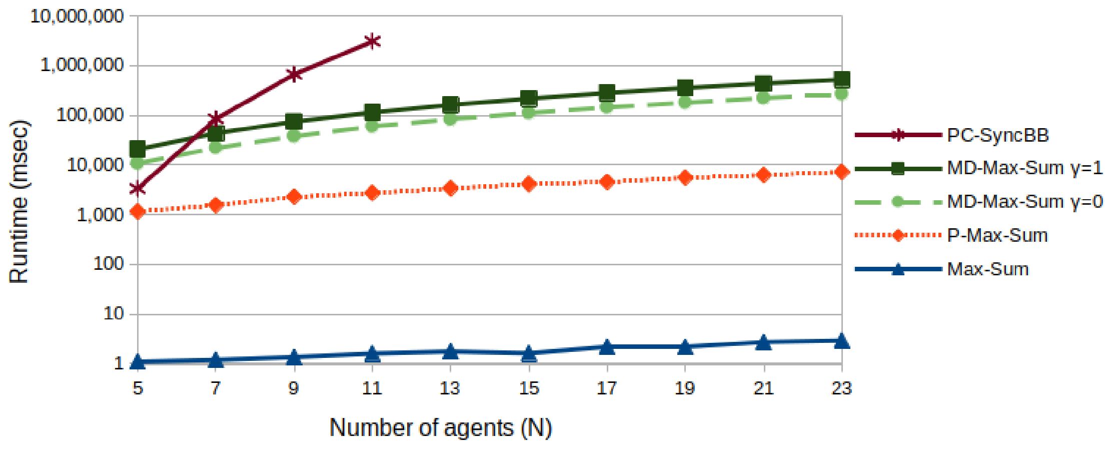

Figure 9: Unstructured sparse random graphs, $p_1 = 0.3$, varying N

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 Figure 10, 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 runtime, which is consistent with our experimentation with the impact of the topology privacy index (Figure 6). In view of these experiments for dense problems it is advised to apply the MD-Max-Sum variant that offers full topology privacy ($\gamma = 1$).

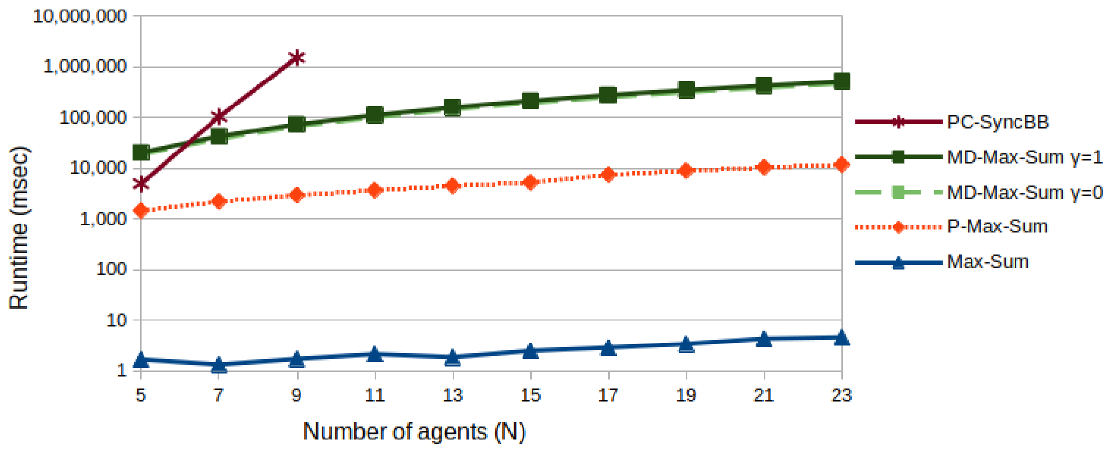

Figure 10: Unstructured dense random graphs, $p_1 = 0.7$, varying N

We proceed to compare the performance of the various algorithms on problems of other types. Using the model of Barabási and Albert [4], 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 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 number of nodes N that varies from 5 to 23.

Figure 11 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 MD-Max-Sum with $\gamma = 0$ variant, it remains similar to what we saw in previous experiments. As for PC-SyncBB, it remains competitive for problems of up to 9 agents, and it reaches the cut-off limit after handling 11 agents.

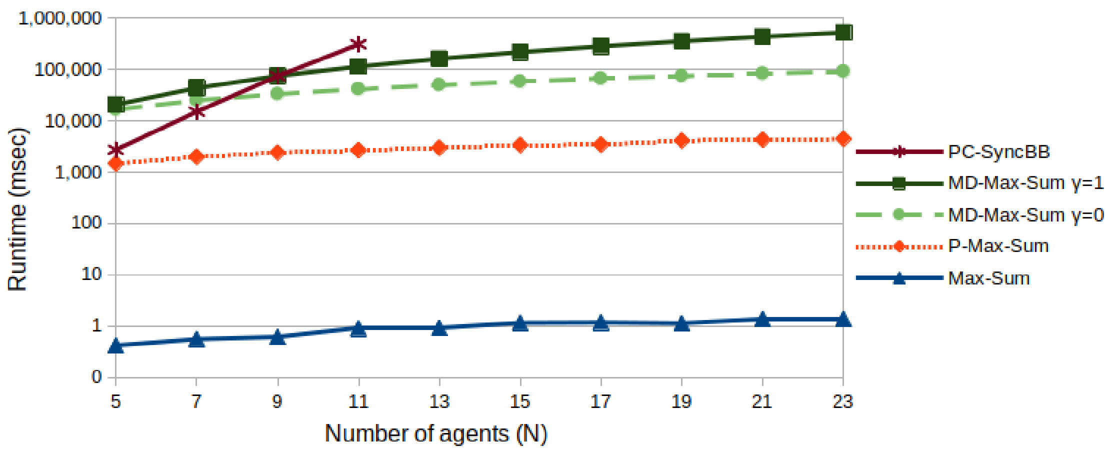

Figure 11: Scale-free graphs ($m_0 = 4$, $m = 2$), varying N

Next, we evaluated the algorithms on distributed meeting scheduling problems, in a setting similar to the one described in [41]. In that setting, the problems are constructed similarly to the PEAV formulation [27], but instead of multiple-variable agents, we follow the *decomposition method* that is used in the P-SyncBB experimentation [42] to separate each variable into a *virtual agent* [44].

Inspired by the setting of Léauté and Faltings [23], 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. Afterwards, 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. Figure 12 presents the performance of the algorithms in this 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.

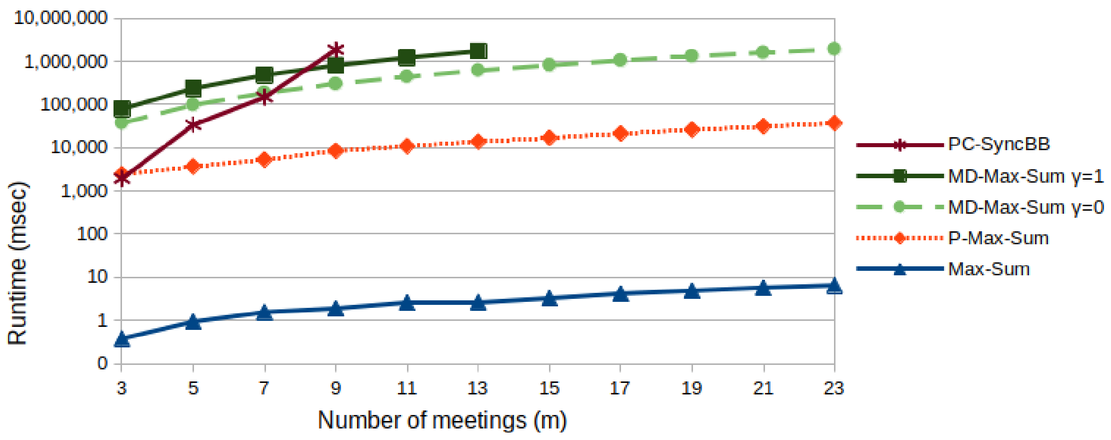

Figure 12: Meeting scheduling problems, varying number of meetings

Finally, we evaluated the algorithms on structured problems starting with 3-color graph coloring problems, similar to the setting described by Zivan et al. [45]. 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. Figure 13 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 Figure 13 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 topology, but the number of mediators remains the same ($L = 5$) and 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.

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

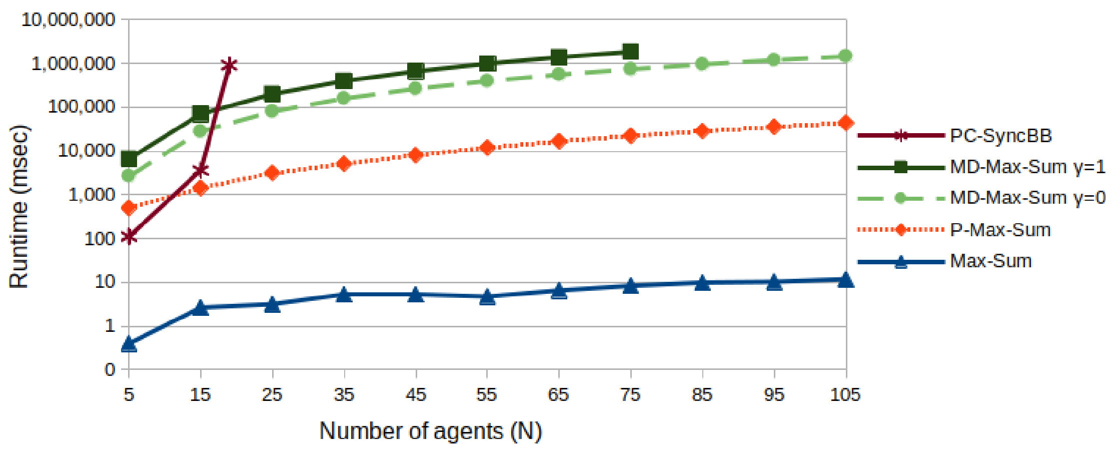

Figure 13: 3-color graph coloring problems ($p_1 = 0.4$), varying N

8 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 constraint, topology, decision, and partial agent 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 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.

A major bottleneck of MD-Max-Sum is the cryptographic MPC protocol behind the **COMPARE** sub-protocol. In our experiments we used the implementation of Nishide and Ohta [33] for the Damgård and Nielsen [11] protocol. The computational problem in **COMPARE** 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 [41] 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 cryptogaphic community, e.g. [6, 29, 32, 36]. 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 [29] reports promising improvements in terms of throughput from which MD-Max-Sum might benefit.

While MD-Max-Sum preserves partial agent privacy, we believe it is possible to enhance it to support full agent privacy. The main idea is to assign random identifiers to the agents, instead of ordinal indices, and to provide them with only an upper bound on the maximum domain size Δ and number of agents N. We leave the concrete implementation of this enhancement for future work.

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 synchronize the intermediate results between iterations.

MD-Max-Sum simulates the original version of Max-Sum [12]. 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 [8] demonstrate improved solution quality. The downside is that these versions require a considerably higher number of iterations (hundreds or even thousands of iterations) until reaching their solution-quality potential. We leave for future work the interesting challenge of efficiently implementing such versions of Max-Sum in the mediated model.

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 of 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.

References

[1] Srinivas M. Aji and Robert J. McEliece. The generalized distributive law. *IEEE transactions on Information Theory*, 46(2):325–343, 2000.

[2] Joël Alwen, Jonathan Katz, Yehuda Lindell, Giuseppe Persiano, Abhi Shelat, and Ivan Visconti. Collusion-free multiparty computation in the mediated model. In *CRYPTO*, pages 524–540, 2009.

[3] Joël Alwen, Abhi Shelat, and Ivan Visconti. Collusion-free protocols in the mediated model. In *CRYPTO*, pages 497–514, 2008.

[4] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[5] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC*, pages 1–10, 1988.

[6] Octavian Catrina and Sebastiaan De Hoogh. Improved primitives for secure multiparty integer computation. In *SCN*, pages 182–199. Springer, 2010.

[7] Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. Fast large-scale honest-majority MPC for malicious adversaries. In *CRYPTO*, pages 34–64, 2018.

[8] Liel Cohen, Rotem Galiki, and Roie Zivan. Governing convergence of Max-sum on DCOPs through damping and splitting. *Artificial Intelligence*, 279:103212, 2020.

[9] Marius călin Silaghi. A comparison of distributed constraint satisfaction approaches with respect to privacy. In *DCR*, 2002.

[10] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *TCC*, pages 285–304. Springer, 2006.

[11] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *CRYPTO*, pages 572–590, 2007.

[12] Alessandro Farinelli, Alex Rogers, and Nick R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *AAMAS*, pages 639–646, 2008.

[13] Ferdinando Fioretto, Enrico Pontelli, and William Yeoh. Distributed constraint optimization problems and applications: A survey. *Journal of Artificial Intelligence Research*, 61:623–698, 2018.

[14] Ferdinando Fioretto, William Yeoh, and Enrico Pontelli. A multiagent system approach to scheduling devices in smart homes. In *AAMAS*, pages 981–989, 2017.

[15] Stephen Fitzpatrick and Lambert Meertens. Distributed coordination through anarchic optimization. In *Distributed Sensor Networks*, pages 257–295. Springer, 2003.

[16] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[17] Tal Grinshpoun and Amnon Meisels. Completeness and performance of the APO algorithm. *Journal of Artificial Intelligence Research*, 33:223–258, 2008.

[18] Tal Grinshpoun and Tamir Tassa. P-SyncBB: A privacy preserving branch and bound DCOP algorithm. *Journal of Artificial Intelligence Research*, 57:621–660, 2016.

[19] Tal Grinshpoun, Tamir Tassa, Vadim Levit, and Roie Zivan. Privacy preserving region optimal algorithms for symmetric and asymmetric DCOPs. *Artificial Intelligence*, 266:27–50, 2019.

[20] Katsutoshi Hirayama and Makoto Yokoo. Distributed partial constraint satisfaction problem. In *CP*, pages 222–236, 1997.

[21] Hideaki Katagishi and Jonathan P. Pearce. Kopt: Distributed DCOP algorithm for arbitrary k-optima with monotonically increasing utility. In *DCR*, 2007.

[22] Christopher Kiekintveld, Zhengyu Yin, Atul Kumar, and Milind Tambe. Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In *AAMAS*, pages 133–140, 2010.

[23] Thomas Léauté and Boi Faltings. Protecting privacy through distributed computation in multi-agent decision making. *Journal of Artificial Intelligence Research*, 47:649–695, 2013.

[24] Fernando Lezama, Jorge Palominos, Ansel Y Rodríguez-González, Alessandro Farinelli, and Enrique Munoz de Cote. Agent-based microgrid scheduling: An ICT perspective. *Mobile Networks and Applications*, 24(5):1682–1698, 2019.

[25] Benny Lutati, Inna Gontmakher, Michael Lando, Arnon Netzer, Amnon Meisels, and Alon Grubshtein. Agentzero: A framework for simulating and evaluating multi-agent algorithms. In *Agent-Oriented Software Engineering*, pages 309–327, 2014.

[26] Rajiv T. Maheswaran, Jonathan P. Pearce, and Milind Tambe. A family of graphical-game-based algorithms for distributed constraint optimization problems. In *Coordination of Large-Scale Multiagent Systems*, pages 127–146. Springer-Verlag, 2006.

[27] Rajiv T. Maheswaran, Milind Tambe, Emma Bowring, Jonathan P. Pearce, and Pradeep Varakantham. Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *AAMAS*, pages 310–317, 2004.

[28] Roger Mailler and Victor R. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS*, pages 438–445, 2004.

[29] Eleftheria Makri, Dragos Rotaru, Frederik Vercauteren, and Sameer Wagh. Rabbit: Efficient comparison for secure multi-party computation. In *FC*, volume 12674, pages 249–270, 2021.

[30] Amnon Meisels, Eliezer Kaplansky, Igor Razgon, and Roie Zivan. Comparing performance of distributed constraints processing algorithms. In *AAMAS*, pages 86–93. Citeseer, 2002.

[31] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. ADOPT: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161:149–180, 2005.

[32] Hiraku Morita, Nuttapong Attrapadung, Satsuya Ohata, Shota Yamada, Koji Nuida, and Goichiro Hanaoka. Tree-based secure comparison of secret shared data. In *ISITA*, pages 525–529, 2018.

[33] Takashi Nishide and Kazuo Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In *PKC*, pages 343–360, 2007.

[34] Adrian Petcu and Boi Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, pages 266–271, 2005.

[35] Gauthier Picard. Auction-based and distributed optimization approaches for scheduling observations in satellite constellations with exclusive orbit portions. *arXiv preprint arXiv:2106.03548*, 2021.

[36] Tord Ingolf Reistad. Multiparty comparison-an improved multiparty protocol for comparison of secret-shared values. In *SECRYPT*, volume 1, pages 325–330. SCITEPRESS, 2009.

[37] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[38] Marius C. Silaghi. and Debasis Mitra. Distributed constraint satisfaction and optimization with privacy enforcement. In *IAT*, pages 531–535, 2004.

[39] Evan A. Sultanik, Robert N. Lass, and William C. Regli. DCOPolis: a framework for simulating and deploying distributed constraint reasoning algorithms. In *AAMAS (demos)*, pages 1667–1668, 2008.

[40] Tamir Tassa, Tal Grinshpoun, and Avishay Yanai. A privacy preserving collusion secure DCOP algorithm. In *IJCAI*, pages 4774–4780, 2019.

[41] Tamir Tassa, Tal Grinshpoun, and Avishay Yanai. PC-SyncBB: A privacy preserving collusion secure DCOP algorithm. *Artificial Intelligence*, 297:103501, 2021.

[42] Tamir Tassa, Tal Grinshpoun, and Roie Zivan. Privacy preserving implementation of the Max-Sum algorithm and its variants. *Journal of Artificial Intelligence Research*, 59:311–349, 2017.

[43] Andrew C. Yao. Protocols for secure computation. In *FOCS*, pages 160–164, 1982.

[44] Makoto Yokoo and Katsutoshi Hirayama. Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems*, 3(2):185–207, 2000.

[45] Roie Zivan, Steven Okamoto, and Hilla Peled. Explorative anytime local search for distributed constraint optimization. *Artificial Intelligence*, 212:1–26, 2014.