Journal of Artificial Intelligence Research 59 (2017) 311-349 Submitted 3/17; published 7/17

Privacy Preserving Implementation of the Max-Sum Algorithm and its Variants

Tamir Tassa TAMIRTA@OPENU.AC.IL
The Open University,
Ra'anana, Israel

Tal Grinshpoun TALGR@ARIEL.AC.IL
Ariel University,
Ariel, Israel

Roie Zivan ZIVANR@BGU.AC.IL
Ben-Gurion University of the Negev,
Beer Sheva, Israel

Abstract

One of the basic motivations for solving DCOPs is maintaining agents' privacy. Thus, researchers have evaluated the privacy loss of DCOP algorithms and defined corresponding notions of privacy preservation for secured DCOP algorithms. However, no secured protocol was proposed for Max-Sum, which is among the most studied DCOP algorithms. As part of the ongoing effort of designing secure DCOP algorithms, we propose P-Max-Sum, the first private algorithm that is based on Max-Sum. The proposed algorithm has multiple agents preforming the role of each node in the factor graph, on which the Max-Sum algorithm operates. P-Max-Sum preserves three types of privacy: topology privacy, constraint privacy, and assignment/decision privacy. By allowing a single call to a trusted coordinator, P-Max-Sum also preserves agent privacy. The two main cryptographic means that enable this privacy preservation are secret sharing and homomorphic encryption. In addition, we design privacy-preserving implementations of four variants of Max-Sum. We conclude by analyzing the price of privacy in terns of runtime overhead, both theoretically and by extensive experimentation.

1. Introduction

The Distributed Constraint Optimization Problem (DCOP) is a general model for distributed problem solving that has a wide range of applications in multi-agent systems. Many algorithms for solving DCOPs have been proposed. Complete algorithms (Modi, Shen, Tambe, & Yokoo, 2005; Petcu & Faltings, 2005b; Gershman, Meisels, & Zivan, 2009) are guaranteed to find the optimal solution, but because DCOPs are NP-hard, these algorithms require exponential time in the worst case. Thus, there is growing interest in incomplete algorithms, which may find suboptimal solutions but run quickly enough to be applied to large problems or real-time applications (Maheswaran, Pearce, & Tambe, 2004a; Zhang, Wang, Xing, & Wittenburg, 2005; Zivan, Okamoto, & Peled, 2014; Teacy, Farinelli, Grabham, Padhy, Rogers, & Jennings, 2008).

Whether complete or incomplete, DCOP algorithms generally follow one of two broad approaches: distributed search (Modi et al., 2005; Gershman et al., 2009; Maheswaran et al., 2004a; Zhang et al., 2005) or inference (Petcu & Faltings, 2005b, 2005a; Farinelli, Rogers, Petcu, & Jennings, 2008; Stranders, Farinelli, Rogers, & Jennings, 2009). In search algorithms, agents directly traverse the solution space by choosing value assignments and communicating these assignments to each other. By contrast, agents in inference algorithms traverse the solution space indirectly; each agent maintains *beliefs* about the best costs (or utilities) that can be achieved for each value assignment to its own variables, and selects value assignments that are optimal according to its beliefs. Agents calculate and communicate costs/utilities for each possible value assignment to neighboring agents' variables, and update their beliefs based on messages received from their neighbors. These update methods are specific realizations of the Generalized Distributive Law (GDL) algorithm (Aji & McEliece, 2000), and hence inference-based algorithms are often referred to as GDL-based algorithms.

The Max-Sum algorithm (Farinelli et al., 2008) is an incomplete, GDL-based, algorithm that has drawn considerable attention in recent years, including being proposed for multi-agent applications such as sensor systems (Teacy et al., 2008; Stranders et al., 2009) and task allocation for rescue teams in disaster areas (Ramchurn, Farinelli, Macarthur, & Jennings, 2010). Agents in Max-Sum propagate cost/utility information to all neighbors. This contrasts with other inference algorithms such as ADPOP (Petcu & Faltings, 2005a), in which agents only propagate costs up a pseudo-tree structure. As is typical of inference algorithms, Max-Sum is purely exploitative both in the computation of its beliefs and in its selection of values based on those beliefs.

One of the main motivations for solving constraint problems in a distributed manner is that of *privacy*. The term privacy is quite broad, a fact that gave rise to several categorizations of the different types of privacy (Greenstadt, Grosz, & Smith, 2007; Grinshpoun, 2012; Léauté & Faltings, 2013). In this paper we relate to the categorization of Léauté and Faltings (2013) that distinguished between agent privacy, topology privacy, constraint privacy, and assignment/decision privacy.

Most studies that evaluated distributed constraint algorithms in terms of privacy considered either search algorithms or complete inference algorithms. Some examples are Maheswaran, Pearce, Bowring, Varakantham, and Tambe (2006) who proposed the VPS framework that was initially used to measure the constraint privacy loss in SyncBB and OptAPO. Later, VPS was also applied to DPOP and ADOPT (Greenstadt, Pearce, & Tambe, 2006). Doshi, Matsui, Silaghi, Yokoo, and Zanker (2008) proposed to inject privacy-loss as a criterion to the problem solving process. Some previous work was also directed towards reducing constraint privacy loss. Most effort in the development of privacy-preserving search algorithms focused on DisCSP, which is the *satisfaction* variant of DCOP. Examples include (Nissim & Zivan, 2005; Silaghi & Mitra, 2004; Yokoo, Suzuki, & Hirayama, 2005). The work of Silaghi and Mitra (2004) addressed both satisfaction and optimization problems. However, the proposed solution is strictly limited to small scale problems since it depends on an exhaustive search over all possible assignments. Several privacy-preserving versions of DPOP were proposed in the past (Greenstadt et al., 2007; Silaghi, Faltings, & Petcu, 2006) including a recent study by Léauté and Faltings (2013) that proposed several versions of DPOP that provide strong privacy guarantees. While these versions are aimed for DCSPs, some of them may be also applicable to DCOPs. Considering a different aspect of constraint privacy, researchers have addressed problems in which the nature of a constraint is distributed among the constrained agents. Solutions to such problems include the

PEAV formulation (Maheswaran, Tambe, Bowring, Pearce, & Varakantham, 2004b) and *asymmetric* DCOPs (Grinshpoun, Grubshtein, Zivan, Netzer, & Meisels, 2013). Here, we restrict ourselves to the traditional symmetric DCOPs. Another recent paper (Grinshpoun & Tassa, 2014) devised a variation of SyncBB (Hirayama & Yokoo, 1997) that preserves constraint and topology privacy. The subsequent study (Grinshpoun & Tassa, 2016) offered a privacy-preserving algorithm that implements SyncBB while preserving also decision privacy, in addition to constraint and topology privacy.

In this paper we propose the first private algorithm that is based on Max-Sum. The proposed algorithm, P-Max-Sum, has multiple agents preforming the role of each node in the factor graph, on which the Max-Sum algorithm operates. Using secret sharing and homomorphic encryption, the agents may execute the nodes' role without revealing the content of the messages they receive or the details of the computation they perform and the messages that they generate. Thus, the proposed algorithm prevents any given agent from discovering topological constructs in the constraint graph, such as the number of other nodes, or existence of edges not adjacent to its own node (topology privacy), the costs that other agents assign to value assignments (constraint privacy), or the assignment selection of other agents (assignment/decision privacy). By allowing a single call to a trusted coordinator, P-Max-Sum can also preserve agent privacy.

The paper is organized as follows. In Section 2 we provide the necessary background and definitions regarding distributed constraint optimization problems and the Max-Sum algorithm. Section 3 describes the main contribution of this study, the P-Max-Sum algorithm, which is a privacy-preserving implementation of Max-Sum. In Section 4 we prove the correctness of P-Max-Sum and its privacy properties. Then, in Section 5 we discuss privacy-preserving implementations of several variants of Max-Sum; some of them can be handled with the same cryptographic tools that were implemented in the design of P-Max-Sum, while others require different cryptographic weaponry. We evaluate the performance of our privacy-preserving algorithms, both analytically and experimentally, in Section 6, and conclude in Section 7.

This article has evolved from a paper that was published at IJCAI 2015 conference (Tassa, Zivan, & Grinshpoun, 2015). The extended version includes privacy-preserving implementations of various Max-Sum variants, including the needed adjustments for the anytime mechanism. This version also includes complete proofs, comprehensive analysis of privacy and efficiency, and a considerably more thorough experimental evaluation.

2. Preliminaries

This section contains the relevant background and formal definitions and notations that we use in this paper.

2.1 Distributed Constraint Optimization Problems

A Distributed Constraint Optimization Problem (DCOP) (Hirayama & Yokoo, 1997) 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 $X_1, X_2, \ldots, X_m$, $\mathcal{D}$ is a set of finite domains $D_1, D_2, \ldots, D_m$, and $\mathcal{R}$ is a set of relations (constraints). Each variable X_i takes values in the domain D_i, and it is held by a single agent. Each constraint $C \in \mathcal{R}$ defines a non-negative cost for every possible value combination of a set of variables, and is of the form $C : D_{i_1} \times \cdots \times D_{i_k} \to \mathbb{R}_+ := [0, \infty)$, for some $1 \le i_1 < \cdots < i_k \le m$.[1]

A *value assignment* is a pair including a variable and a value from that variable's domain. We denote by a_i the value assigned to the variable X_i. A *partial assignment* (PA) is a set of assignments in which each variable appears at most once. A constraint $C \in \mathcal{R}$ is applicable to a PA if all variables that are constrained by C are included in the PA. The cost of a PA is the sum of all applicable constraints to the PA. A *complete assignment* is a partial assignment that includes all of the variables. The objective is to find a complete assignment of minimal cost.

For simplicity, we make the common assumption that each agent holds exactly one variable, i.e., $n = m$. We let n denote hereinafter the number of agents and the number of variables. For the same reasons we also concentrate on binary DCOPs, in which all constraints are binary, i.e., they refer to exactly two variables. Such constraints take the form $C_{i,j} : D_i \times D_j \to \mathbb{R}_+$. These assumptions are customary in DCOP literature, e.g., the works of Modi et al. (2005) and Petcu and Faltings (2005b).

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

- V' has two types of nodes: (a) variable nodes – $X_1, \ldots, X_n$, and (b) function nodes – for each $e = (X_i, X_j) \in E$ there is a node X_e in V'.

- E' contains an edge that connects X_i with X_e if and only if e is an edge in G which is adjacent to X_i.

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

2.2 Assumptions

We make herein the following assumptions:

2.2.1 Knowledge Assumptions

The following are commonly used assumptions, see, e.g., the work of Léauté and Faltings (2013): A variable's domain is known only to the agent that owns it and agents owning neighboring variables. In addition, a constraint is fully known to all agents owning variables in its scope, while no other agent knows anything about that constraint (not even its existence).

As for agent knowledge, we make the following assumptions. For each $1 \le i \le n$, denote by $\mathcal{N}(A_i)$ the set of all agents that own a variable that is a neighbor of X_i in the constraint graph. Then, in our main privacy-preserving protocol, we assume that (1) A_i knows all agents in $\mathcal{N}(A_i)$, and (2) all agents in $\mathcal{N}(A_i)$ know each other. We also describe a variant

1. The non-negative assumption could be easily elevated in applications where the costs can be negative. If LB is any lower bound on the costs, then the agents may consider the shifted constraint function $C' = C + |LB|$ which is non-negative. The lower bound LB need not be tight; it can be any value that the semantics of the constraints implies that all constraint values are no smaller than LB.

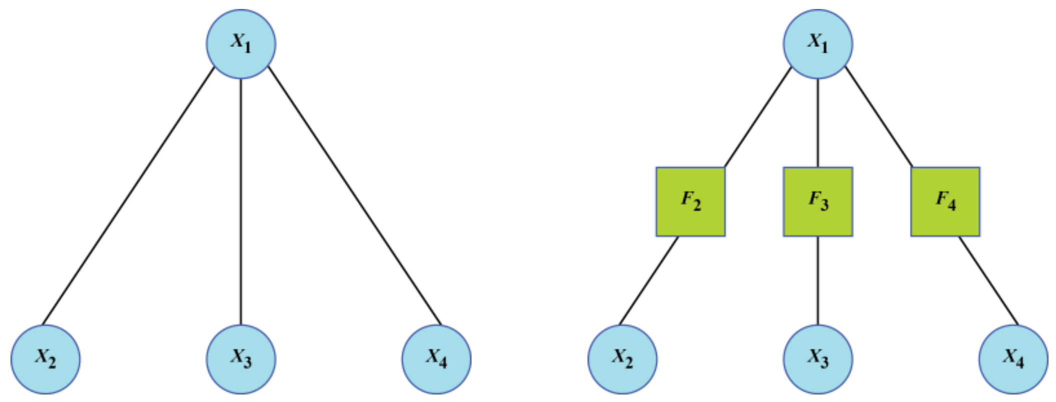

Figure 1: A constraint graph G of a DCOP with 4 variable nodes (left) and the corresponding factor graph G' that has 4 variable nodes and 3 function nodes (right).

of our protocol that maintains agent privacy, in addition to the privacy guarantees of the main protocol. For this variant we rely only on assumption (1) above. Apart from its direct neighbors, A_i does not need to know any of the other agents, not even their existence.

2.2.2 Communication Assumptions

Here too we make the standard communication assumptions, see, e.g. the work of Modi et al. (2005). Each agent can send messages to any of its neighboring agents. The communication system is resilient in the sense that messages do not get lost and they are received by their intended recipient in the same order that they were sent out.

2.3 The Max-Sum Algorithm

Our description of Max-Sum follows its description in recent published papers, e.g., the works of Zivan and Peled (2012) and Zivan, Parash, and Naveh (2015). The Max-Sum algorithm operates on the factor graph G'. Each agent A_i, $1 \le i \le n$, controls its corresponding variable node X_i. 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) that 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_i with X_e, where $e = (X_i, X_j)$. The messages, in both directions, will be vectors of dimension $|D_i|$ and they will be denoted by either $Q^k_{i\rightarrow e}$ or $R^k_{e\rightarrow i}$, depending on the direction, where k is the index of the iteration. If x is one of the elements in D_i then its corresponding entry in the message will be denoted by $Q^k_{i\rightarrow e}(x)$ or $R^k_{e\rightarrow i}(x)$.

In the first 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_i and letting N_i be the set of function nodes adjacent to X_i in G', then for each $X_e \in N_i$, X_i will send to

X_e the vector

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

Fixing a function node X_e, where $e = (X_i, X_j)$, then for each $x \in D_i$,

$$R^{k+1}_{e\to i}(x) := \min_{y\in D_j} \left[C_{i,j}(x,y) + Q^k_{j\to e}(y) \right] \,, \tag{2}$$

while for each $y \in D_j$,

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

Finally, after completing a preset number K of iterations, each variable node X_i computes $M_i := \sum_{X_e \in N_i} R^K_{e\to i}$ and then selects $x \in D_i$ for which $M_i(x)$ is minimal.

During the run of Max-Sum, the entries in the messages Q^k and R^k grow exponentially. In order to prevent the entries in the messages from growing uncontrollably, it is customary to reduce from each entry in each message $Q^{k+1}_{i\to e}$, where $e = (X_i, X_j)$, the value $\alpha^{k+1}_{i,j} := \frac{\sum_{x\in D_i} Q^{k+1}_{i\to e}(x)}{|D_i|}$ or $\alpha^{k+1}_{i,j} := \min_{x\in D_i} Q^{k+1}_{i\to e}(x)$ (Farinelli et al., 2008). (Choosing the minimum instead of the average ensures that all message entries always remain nonnegative.)

However, since in our private version of Max-Sum that we present in the next section we utilize very large arithmetics (as common in public key cryptography), we are not intimidated by such an exponential growth. To that end, we state the following theorem (the proof of which is given in Appendix A.1).

Theorem 2.1 *Define the maximum value of a single constraint,*

$$M_C := \max_{1\le i<j\le n} \max_{x\in D_i, y\in D_j} C_{i,j}(x,y) \,,$$

and assume that the maximal degree in the constraint graph G is $D+1$. Denote

$$q^k := \max_{X_i} \max_{X_e\in N_i} \max_{x\in D_i} Q^k_{i\to e}(x) \tag{4}$$

where the first maximum in Eq. (4) is taken over all variable nodes X_i and the second maximum is over all neighboring function nodes X_e. In addition, we denote

$$r^k := \max_{e=(X_i,X_j)} \max_{\ell\in\{i,j\}} \max_{x\in D_\ell} R^k_{e\to \ell}(x) \tag{5}$$

where the first maximum in Eq. (5) is taken over all function nodes X_e and the second maximum is over the two neighboring variable nodes. Then

$$q^k \le M_C D \cdot \frac{D^{\lfloor k/2 \rfloor} - 1}{D-1} \,, \tag{6}$$

and

$$r^k \le M_C \cdot \frac{D^{\lceil k/2 \rceil} - 1}{D-1} \,. \tag{7}$$

A simple consequence of Theorem 2.1 is that a unified bound for both types of messages is given by

$$q^k, r^k \leq B_k := M_C D \cdot \frac{D^{\lfloor k/2 \rfloor} - 1}{D - 1} \,. \tag{8}$$

Hence, all entries in all messages in the first K iterations of Max-Sum are integers in the interval $[0, B_K]$.

In order to avoid ties in the calculated beliefs, random unary constraints (preferences), with costs that are orders of magnitude lower than the binary costs, can be added by each of the agents; by adding such small random noise constraints, the probability of ties becomes negligible (Farinelli et al., 2008).

We would like to note that on tree-structured (acyclic) factor graphs, Max-Sum is guaranteed to converge to the optimal solution (Rogers, Farinelli, Stranders, & Jennings, 2011). This happens because for every variable node in the graph, the algorithm simulates a DPOP running on a pseudo-tree with no back-edges, where this node is the root. Thus, after a linear number of steps at most, for each value, the variable node holds the cost of the best solution in which that value is involved, and therefore (when there are no ties) it can select the value assignment that is included in the optimal solution.

3. The P-Max-Sum Algorithm

It can be shown that a naïve execution of Max-Sum may reveal private information. For example, assume that agent A_i is the only neighbor of agent A_j. The function e between them is represented by the node X_e and the messages $R^k_{e \to i}$ include costs that are derived only from the constraint between them. Thus, A_i can learn that A_j has no other neighbors. In case A_j has additional neighbors, the messages $R^k_{e \to i}$ include costs of constraints that A_j has with its other neighbors. Thus, A_i may learn information that should not be revealed to it. To avoid such leakage of information, it is imperative to hide from each node X_i (which is controlled by agent A_i) the content of the messages it receives from its neighbors in N_i. To achieve that, any message $Q^k_{i \to e}$ or $R^k_{e \to i}$ (where $e = (X_i, X_j)$) will be split into two random additive shares that will be held by A_i and A_j. Each of these shares, on its own, does not reveal any information on the underlying message. Our secure protocols will take as input those shares, but they will make sure that none of the interacting agents reveal the value of the complementing share so that it remains oblivious of the value of the underlying message. Moreover, the classic Max-Sum dilemma of "which agent controls the function node X_e?" becomes irrelevant since the operation of X_e will be jointly performed by A_i and A_j.

Let μ be a large integer such that

$$\mu > 2B_K \,, \tag{9}$$

where B_K is a uniform bound on the size of all message entries in the first K iterations (see Theorem 2.1 and Eq. (8)). Then each of the entries in any of the messages $Q^k_{i \to e}$ or $R^k_{e \to i}$ will be treated by our secure protocols as an element in $\mathbb{Z}_\mu$, the additive group modulo μ.

Let X_i and X_j be two variable nodes that are connected through a constraint edge $e = (X_i, X_j)$, and let w denote one of the entries in one of the messages that are sent between the corresponding two agents in iteration k (namely, one of the four messages

$Q^k_{i\to e}$, $R^k_{e\to i}$, $Q^k_{j\to e}$, or $R^k_{e\to j}$). Then the two agents A_i and A_j will engage in a secure protocol that will provide each of them a random element in $\mathbb{Z}_\mu$, denoted s_i and s_j, such that $s_i + s_j = w$ (where all arithmetics hereinafter is modulo μ, unless stated otherwise). The sharing procedure will be carried out for all entries in all messages independently (namely, the selection of random shares for one entry will be independent of the selection of shares in other entries, or the selection of shares in the same entry in another iteration).

For any $0 \le k \le K$ let us denote by M^k the set of all messages that are transmitted along the edges of G' in the kth iteration. Eqs. (1)–(3) describe how each of the messages in M^{k+1} is computed from the messages in M^k. Let X_i, X_j be two nodes in V that are connected through an edge $e = (X_i, X_j)$. Then each message $Q^k_{i\to e}$ will be split to two random additive shares as follows:

$$Q^k_{i\to e} = S^{k,i}_{i\to e} + S^{k,j}_{i\to e}\,; \tag{10}$$

the first share will be known only to A_i and the second one only to A_j. Similar sharing will be applied to messages that emerge from function nodes; i.e.,

$$R^k_{e\to i} = S^{k,i}_{e\to i} + S^{k,j}_{e\to i}\,. \tag{11}$$

(Recall that the messages and the shares in Eqs. (10) and (11) are vectors in $\mathbb{Z}_\mu^{|D_i|}$.)

Now, let S^k denote the set of all shares of M^k. Then we should devise a secure computation protocol that will take us from S^k to S^{k+1}. In doing so we shall assume that all agents are curious-but-honest. Namely, their honesty implies that they will follow the protocol and will not collude, but they will try to use their legal view in the secure protocols in order to extract private information on constraints of other agents.

We proceed to describe our secure protocols, starting with the initial creation of shares in iteration $k = 0$ (Section 3.1). Then, in the main body of our work (Section 3.2), we describe the computation of shares in the progression between successive iterations, i.e., the computation of S^{k+1} from S^k. Finally, we discuss the secure computation of assignments from the shares that were obtained in the last iteration (Section 3.3).

3.1 Initialization

For $k = 0$ all messages are zero. To create random shares of all those vectors, every pair of neighboring agents, say A_i and A_j, generate the initial splitting to random shares. To create a splitting as in Eq. (10), A_i and A_j create jointly a random vector $S^{0,i}_{i\to e} \in \mathbb{Z}_\mu^{|D_i|}$, that will be A_i's share, and then they set $S^{0,j}_{i\to e} = -S^{0,i}_{i\to e}$ as A_j's share. Similar initial splitting will be used for the zero message $R^0_{e\to i}$ in Eq. (11).

In order to implement this iteration in a manner that saves communication messages, every pair of connected agents will decide upfront on a pseudo-random number generator (PRNG) (Goldreich, 2001) and an initial seed for that PRNG. Then, whenever required to generate a new pseudo-random number in $\mathbb{Z}_\mu$, each one of them will use that PRNG in order to get the same pseudo-random number, without the need in any further communications between them.

3.2 Progression

Here we describe the protocols that allow the agents to compute S^{k+1} from S^k. We begin with a preliminary discussion about homomorphic encryption and its usage in our protocols (Section 3.2.1). Then we split our discussion to the two types of messages – Q-messages, that are sent from variable nodes to function nodes (Section 3.2.2), and R-messages that are sent from function nodes to variable nodes (Section 3.2.3).

3.2.1 Using Homomorphic Encryption

An additive homomorphic encryption is a public-key encryption function $\mathcal{E} : \Omega_P \to \Omega_C$ where Ω_P and Ω_C are the domains of plaintexts and ciphertexts, respectively, Ω_P is an additive group, Ω_C is a multiplicative group, and $\mathcal{E}(x + y) = \mathcal{E}(x) \cdot \mathcal{E}(y)$ for all $x, y \in \Omega_P$. Examples for such ciphers are Benaloh (1994) and Paillier (1999) ciphers.

In our secure protocols we assume that every set of agents of the form $\mathcal{A}_{-i} := \{A_j : 1 \leq j \leq n, j \neq i\}$ has an additive homomorphic cryptosystem, in which the encryption function is denoted $\mathcal{E}_i$. The pair of public (encryption) and private (decryption) keys in $\mathcal{E}_i$ is known to all agents in $\mathcal{A}_{-i}$ (namely, to all agents except for A_i); A_i itself, on the other hand, will be notified only of the public encryption key in $\mathcal{E}_i$. That encryption will be used by the neighbors of A_i in G in order to convey messages between them. Since the topology of the graph is private, the neighbors of A_i do not know each other and hence they will send those messages through A_i; the encryption will guarantee that A_i cannot recover those messages.

Letting $m(\mathcal{E}_i)$ denote the modulus of $\mathcal{E}_i$, $1 \leq i \leq n$; i.e., $m(\mathcal{E}_i)$ is the size of the domain of plaintexts of $\mathcal{E}_i$. Then a good selection of μ (the size of the domain $\mathbb{Z}_\mu$ in which all shares take value) would be

$$\mu = \min\{m(\mathcal{E}_i) : 1 \leq i \leq n\}\,. \tag{12}$$

Such a setting of μ is also consistent with the assumption (9). Indeed, typical sizes of the modulus in public-key cryptosystems in general, and in homomorphic encryption in particular, are 512 or 1024 bits. As for B_K, it is defined in Eq. (8) and it equals roughly $M_C D^{K/2}$. In most settings in which Max-Sum is executed, the latter value is much smaller than 512 bits. However, even if K has to be set for larger values, we can either use 1024-bit moduli (thus increasing the security of the secure protocol at the expense of increasing also the computation and communication costs), or apply one of the two corrective mechanisms (for preventing explosion in the message entries) as described in Section 2.3. To keep our discussion simple and focused we stick hereinafter to our assumption (9).

3.2.2 Computing Shares in Messages that emerge from a Variable Node

Fix a variable node X_i. As discussed earlier, for each $X_e \in N_i$ (namely, a function node adjacent to the variable node X_i in G'), X_i needs to send to X_e the vector $Q^{k+1}_{i\to e} := \sum_{X_f \in N_i \setminus \{X_e\}} R^k_{f\to i}$ (see Eq. (1)). Let us denote $t := |N_i|$ and let $1 \leq j_1 < j_2 < \cdots < j_t \leq n$ be the indices of all variables that are constrained with X_i so that $N_i = \{X_{e_\ell} : 1 \leq \ell \leq t\}$, where $e_\ell = (X_i, X_{j_\ell})$. Let us concentrate on one of the function nodes adjacent to X_i, say X_{e_1}. Then

$$Q^{k+1}_{i\to e_1} = \sum_{\ell=2}^{t} R^k_{e_\ell \to i}\,. \tag{13}$$

We start by dealing with the case $t \geq 2$. In that case, the sum on the right-hand side of Eq. (13) is non-empty. Each of the vectors in the sum on the right-hand side of Eq. (13) is shared between A_i and A_{j_ℓ}, $2 \leq \ell \leq t$. Therefore, the two shares in $Q^{k+1}_{i \to e_1} := S^{k+1,i}_{i \to e_1} + S^{k+1,j_1}_{i \to e_1}$ (see Eq. (10)) can be computed as follows:

- The share that A_i will get, denoted $S^{k+1,i}_{i \to e_1}$, is the sum of the $t-1$ shares that A_i has for the $t-1$ messages $R^k_{e_\ell \to i}$, $2 \leq \ell \leq t$. A_i can compute it on its own.
- The share that A_{j_1} will get, $S^{k+1,j_1}_{i \to e_1}$, is the sum of the $t-1$ shares that $A_{j_2}, \ldots, A_{j_t}$ have in iteration k for the $t-1$ messages $R^k_{e_\ell \to i}$, $2 \leq \ell \leq t$. This is done in a secure manner as described in Protocol 1 below.

Protocol 1 describes the process of share generation in messages that emerge from a fixed variable node, X_i. Let $S^{k,i}_{e_\ell \to i}$ and $S^{k,j_\ell}_{e_\ell \to i}$ be the shares that A_i and A_{j_ℓ} hold, respectively, in $R^k_{e_\ell \to i}$, $1 \leq \ell \leq t$. Those shares will be the inputs that A_i and its neighbors A_{j_ℓ}, $1 \leq \ell \leq t$, bring to Protocol 1. The output to A_i will be the shares $S^{k+1,i}_{i \to e_\ell}$ for each $1 \leq \ell \leq t$. The output to the neighboring agent A_{j_ℓ}, $1 \leq \ell \leq t$, will be $S^{k+1,j_\ell}_{i \to e_\ell}$, which is the complement share in $Q^{k+1}_{i \to e_\ell}$.

In Steps 1–2 of Protocol 1, all neighbors send to A_i their shares, encrypted with $\mathcal{E}_i$, to prevent A_i from recovering them. (The encryption $\mathcal{E}_i(\cdot)$ is applied independently on each of the $|D_i|$ components of the share $S^{k,j_\ell}_{e_\ell \to i}$.) The subsequent loop in Steps 3–6 describes the interaction of A_i vis-a-vis each of the neighboring agents A_{j_ℓ}, $1 \leq \ell \leq t$. In Step 4, A_i computes its share in $Q^{k+1}_{i \to e_\ell}$. In Step 5, A_i sends to A_{j_ℓ} a message W_ℓ where, owing to the additive homomorphic property of $\mathcal{E}_i$, $W_\ell = \mathcal{E}_i(\sum_{1 \leq \ell' \neq \ell \leq t} S^{k,j_{\ell'}}_{e_{\ell'} \to i})$. Hence, A_{j_ℓ} recovers in Step 6 its share $S^{k+1,j_\ell}_{i \to e_\ell} = \sum_{1 \leq \ell' \neq \ell \leq t} S^{k,j_{\ell'}}_{e_{\ell'} \to i}$ as required.

Protocol 1 Computing shares in messages that emerge from a variable node

```
1: for $\ell = 1, \ldots, t$ do
2:     $A_{j_\ell}$ sends to $A_i$ the encryption of its share $\mathcal{E}_i(S^{k,j_\ell}_{e_\ell \to i})$.
3: for $\ell = 1, \ldots, t$ do
4:     $A_i$ computes $S^{k+1,i}_{i \to e_\ell} \leftarrow \sum_{1 \leq \ell' \neq \ell \leq t} S^{k,i}_{e_{\ell'} \to i}$.
5:     $A_i$ computes $W_\ell := \prod_{1 \leq \ell' \neq \ell \leq t} \mathcal{E}_i(S^{k,j_{\ell'}}_{e_{\ell'} \to i})$ and sends it to $A_{j_\ell}$.
6:     $A_{j_\ell}$ sets $S^{k+1,j_\ell}_{i \to e_\ell} \leftarrow \mathcal{E}_i^{-1}(W_\ell)$.
```

Note that if the graph topology was not private, then each of the neighbors A_{j_ℓ} of A_i could have obtained its share $S^{k+1,j_\ell}_{i \to e_\ell}$ if all other neighboring agents of A_i would have sent their shares directly to A_{j_ℓ}, in the clear, without involving A_i. However, such a course of action would reveal to each neighbor of A_i the entire neighborhood of A_i. The solution that we suggest here hides the topology by using A_i as a proxy for those messages. Using encryption hides the content of the sent shares from A_i. Using homomorphic encryption allows A_i to perform the computation in Step 5 and then send to A_{j_ℓ} just a single message. Without the homomorphic property A_i would have needed to send all $t-1$ messages to A_{j_ℓ}, thus revealing to A_{j_ℓ} the number of A_i's neighbors.

We now attend to the case where $t = 1$. This case calls for a special attention since the sum in Step 4 and the product in Step 5 are empty. We note that in this case the message to be sent, $Q^{k+1}_{i\to e_1}$, is zero (since the sum in the equation that defines it, Eq. (1), is empty). However, A_i does not wish to reveal the fact that this message is zero to A_{j_1} since then A_{j_1} will learn that it is the only neighbor of A_i. Therefore, we still carry out Steps 1–2, even though they are redundant, in order to hide from A_{j_1} the fact that it is the single neighbor of A_i. Then, instead of Step 4 in Protocol 1, A_i will generate a new random share for itself, denoted $S^{k+1,i}_{i\to e_1}$, and in Step 5, A_i will set $W_1 := \mathcal{E}_i(-S^{k+1,i}_{i\to e_1})$. Hence, A_{j_1} will recover in Step 6 the share $S^{k+1,j_1}_{i\to e_1} = -S^{k+1,i}_{i\to e_1}$; the sum of those two shares is zero, as required, and we achieve it without revealing to A_{j_1} the fact that it is the only neighbor of A_i.

In view of the discussion in Section 3.2.1, if needed then Protocol 1 may be augmented by a post-processing procedure in which every pair of neighbors A_i and A_j compute $\alpha^{k+1}_{i,j}$ and then A_j decreases $\alpha^{k+1}_{i,j}$ from its share $S^{k+1,j}_{i\to e}$. We omit further details.

3.2.3 Computing Shares in Messages that emerge from a Function Node

Fix a function node X_e, where $e = (X_i, X_j)$. Eqs. (2)–(3) describe the messages emerging from X_e. Those messages consist of $|D_i|+|D_j|$ scalars, that A_i and A_j have to share between themselves. Let us concentrate on one of those scalars, say $R^{k+1}_{e\to i}(x)$ for some $x \in D_i$. The secure multiparty problem that A_i and A_j are facing is as follows: if $s^{k,i}(y)$ and $s^{k,j}(y)$ are the two scalar additive shares that A_i and A_j hold in $Q^k_{j\to e}(y)$, for some $y \in D_j$, then they need to get two random additive shares $s^{k+1,i}(x)$ and $s^{k+1,j}(x)$ so that

$$s^{k+1,i}(x) + s^{k+1,j}(x) = R^{k+1}_{e\to i}(x) = \min_{y\in D_j}\left[C_{i,j}(x,y) + s^{k,i}(y) + s^{k,j}(y)\right] \tag{14}$$

where all additions are modulo μ, but when computing the minimum in Eq. (14), the numbers in the brackets are viewed as nonnegative integers.

The above computational problem may be viewed as a problem of secure two-party computation. Assume that two mutually non-trusting players, Alice and Bob, hold private inputs x and y respectively, and they wish to compute $z := f(x,y)$ without revealing the value of their private inputs to each other. Yao's garbled circuit protocol (Yao, 1982) performs such computations with perfect privacy, in the sense that at its completion, either Alice or Bob or both of them learn the output z; however, Alice does not learn anything on Bob's input y beyond what is implied by z and her input x, and similarly for Bob. The protocol assumes that f can be computed by a Boolean circuit and it accomplishes the privacy goal by "garbling" the inputs and outputs of each gate in the circuit. Yao's protocol can be used in order to compute the shares in the left hand side of Eq. (14) with perfect privacy. Indeed, in order to compute the shares $s^{k+1,i}(x)$ and $s^{k+1,j}(x)$ for a given $x \in D_i$, the input that A_i will provide to the arithmetic circuit will be $\{s^{k,i}(y) : y \in D_j\}$ as well as an already selected random share $s^{k+1,i}(x) \in \mathbb{Z}_\mu$. A_j's input will be $\{C_{i,j}(x,y) + s^{k,j}(y) : y \in D_j\}$. The output, which will go only to A_j, will be the value $s^{k+1,j}(x)$ for which $s^{k+1,i}(x) + s^{k+1,j}(x) = R^{k+1}_{e\to i}(x)$.

However, implementing Yao's protocol for each entry in each transmitted message emerging from a function node in each of the algorithm's iterations is impractical due to the hefty computational toll of that protocol. Therefore, we proceed to describe a much simpler and more efficient protocol that A_i and A_j can execute for carrying out the same secure computation. Protocol 2, which we describe below, is not perfectly secure as it leaks some excessive information. But, as we argue later on, that excessive information is benign and of no practical use; on the other hand, the gain in efficiency (in comparison to a solution that relies on Yao's garbled circuit protocol) is enormous since choosing Protocol 2 makes the difference between a theoretical solution and a practical one. (We remind the reader that the privacy discussion is given in Section 4, while the computational cost analysis is given in Section 6.)

Protocol 2 Computing shares in messages that emerge from a function node

```
1: $A_j$ sends to $A_i$ the value $\mathcal{E}_i(s^{k,j}(y))$ for all $y \in D_j$.
2: $A_i$ selects uniformly at random $r \in \mathbb{Z}_\mu$.
3: $A_i$ computes $W(y) := \mathcal{E}_i(s^{k,j}(y)) \cdot \mathcal{E}_i(C_{i,j}(x,y) + s^{k,i}(y) + r)$ for all $y \in D_j$.
4: $A_i$ sends a permutation of $\{W(y) : y \in D_j\}$ to $A_j$.
5: $A_j$ computes $w(y) := \mathcal{E}_i^{-1}(W(y)) = C_{i,j}(x,y) + s^{k,i}(y) + s^{k,j}(y) + r$ for all $y \in D_j$.
6: $A_j$ computes
       $w := \min^*\{w(y)|y \in D_j\} :=$
       $= \begin{cases} \min\{w(y) \mid y \in D_j\} & \max\{w(y)|y \in D_j\} - \min\{w(y)|y \in D_j\} \le (\mu-1)/2 \\ \min\{w(y) \mid w(y) > (\mu-1)/2\} & \text{otherwise} \end{cases}$
7: $A_j$ generates a random share for itself $s^{k+1,j}(x) \in \mathbb{Z}_\mu$.
8: $A_j$ sends to $A_i$ the value $w' = w - s^{k+1,j}(x)$.
9: $A_i$ computes $s^{k+1,i}(x) = w' - r$.
```

All arithmetic operations in Protocol 2, that we proceed to describe and discuss, are operations in $\mathbb{Z}_\mu$ (namely, operations modulo μ), unless otherwise stated.

In Step 1 of Protocol 2, A_j sends to A_i an encryption by $\mathcal{E}_i$ of its share $s^{k,j}(y)$ for all $y \in D_j$. Since A_i does not have the decryption key of $\mathcal{E}_i$, it cannot decrypt $\mathcal{E}_i(s^{k,j}(y))$ in order to recover A_j's private shares. However, as $\mathcal{E}_i$ is homomorphic, A_i can perform the needed arithmetics on the received shares.

In Step 2, A_i selects at random a masking scalar $r \in \mathbb{Z}_\mu$ that will be used to protect private information from A_j, as we shall see shortly. Then, the computation in Step 3 results, for each $y \in D_j$, in a value $W(y)$ that equals $\mathcal{E}_i(w(y))$ where

$$w(y) := C_{i,j}(x,y) + s^{k,i}(y) + s^{k,j}(y) + r\,, \tag{15}$$

owing to the additive homomorphism of $\mathcal{E}_i$. Since $s^{k,i}(y) + s^{k,j}(y) = Q^k_{j\to e}(y)$, the value of $w(y)$ in Eq. (15) equals the corresponding argument in the minimum on the right-hand side of Eq. (2),

$$m(y) := C_{i,j}(x,y) + Q^k_{j\to e}(y)\,, \tag{16}$$

shifted by the random mask r. Specifically,

$$w(y) = m(y) + r \mod \mu\,. \tag{17}$$

In Step 4, A_i sends all $W(y)$, $y \in D_j$, to A_j but it randomly permutes them so that A_j will not be able to associate any value of $W(y)$ to any assignment $y \in D_j$. After A_j decrypts the received values it recovers in Step 5 all values $w(y)$, $y \in D_j$.

Before moving on to discussing Step 6 of Protocol 2, we state the following general lemma.

Lemma 3.1 *Let M and μ be two integers such that $M \leq (\mu-1)/2$. Assume that $m_1 \leq m_2 \leq \cdots \leq m_k$ are integers from the range $[0, M]$ and let $r \in \mathbb{Z}_\mu$. Set $n_i = m_i + r \mod \mu$, $1 \leq i \leq k$. Let $\min n_i$ and $\max n_i$ be the minimum and maximum of the set of integers $\{n_1, \ldots, n_k\}$, and define*

$$\min{}^* n_i = \begin{cases} \min n_i & \max n_i - \min n_i \leq (\mu-1)/2 \\ \min\{n_i \mid n_i > (\mu-1)/2\} & \textit{otherwise} \end{cases} \tag{18}$$

Then

$$\min m_i = m_1 = \begin{cases} \min^* n_i - r & \min^* n_i - r \geq 0 \\ \min^* n_i - r + \mu & \textit{otherwise} \end{cases} \tag{19}$$

The proof of the lemma is given in Appendix A.2. What Lemma 3.1 says is very simple: If we take a sequence of integers $m_1 \leq \cdots \leq m_k$ and apply on it a simple form of encryption, by shifting each of the integers by a fixed shift r modulo some modulus μ, we will be able to identify in the resulting set of integers (denoted $\{n_1, \ldots, n_k\}$) the term that corresponds to the minimal term in the original sequence m_1 if the known bound M on that sequence is smaller than $(\mu-1)/2$. Eq. (18) tells us what is the term in the shifted set that is the image of m_1, while Eq. (19) tells us how to recover m_1 from its shifted value, if we know the shift r.

Example. Assume that $M = 10$ and that the original sequence is $m_1 = 1$, $m_2 = 5$, and $m_3 = 8$. Assume that we try to "encrypt" this sequence in $\mathbb{Z}_\mu$ for a too small μ, say $\mu = 12$ (such a μ is too small in this context since it does not satisfy $\mu \geq 2M + 1$). Suppose that the chosen value of r was $r = 9$. Then the resulting set of shifted values would be $\{2, 5, 10\}$. It is impossible to infer from this set which of its terms is the image of the original minimal term m_1, since that set could have been originated from:

1. The sequence $(0, 3, 8)$ with $r = 2$ or $(1, 4, 9)$ with $r = 1$ or $(2, 5, 10)$ with $r = 0$; in all of those cases, it is the shifted value 2 that is the image of the minimal term m_1 in the original sequence.

2. The sequence $(0, 5, 9)$ with $r = 5$ or $(1, 6, 10)$ with $r = 4$; in all of those cases, 5 is the image of m_1.

3. The sequence $(0, 4, 7)$ with $r = 10$ or $(1, 5, 8)$ with $r = 9$ or $(2, 6, 9)$ with $r = 8$ or $(3, 7, 10)$ with $r = 7$; in all of those cases, 10 is the image of m_1.

Now let us assume that we use $\mu = 30 \geq 2M + 1$, and we picked $r = 27$. In that case the sequence $(1, 5, 8)$ will be mapped to the set $\{2, 5, 28\}$. If we know that $M = 10$ then it is clear that 2 cannot be the image of m_1 since then 28 would have to be the image of m_3 and then $m_3 - m_1 = 26$; that is impossible since we know that $m_1, m_2, m_3 \in [0, M = 10]$. Similarly, 5 cannot be the image of m_1. Then here, owing to the fact that $\mu \geq 2M + 1$, we can identify the image of m_1 as 28. The possible original sequences are $(0, 4, 7)$, $(1, 5, 8)$, $(2, 6, 9)$ or $(3, 7, 10)$ with $r = 28$, $r = 27$, $r = 26$ or $r = 25$, respectively. Note that since r was chosen uniformly at random, all of those sequences are equally probable and there is no way to distinguish between them. $\square$

We now return to discuss Step 6 of Protocol 2. This is the only step in the protocol in which the arithmetics is the standard one of integers, and not modular arithmetics; namely, here we view all values $w(y)$, $y \in D_j$, which are given in Eq. (17), as integers. In view of our discussion at the end of Section 2.3, all values $m(y)$, Eq. (16), are bounded to the interval $[0, B_K]$, where B_K is defined in Eq. (8) and K is the overall number of iterations. Recall also our assumption in Eq. (9), which implies that $B_K \leq (\mu - 1)/2$. In Step 6, A_j computes $w := \min^*\{w(y) | y \in D_j\}$, where the operator $\min^*$ is as the one that we defined in Eq. (18). In view of Lemma 3.1, we infer that $\min\{m(y)|y \in D_j\}$ equals either $w - r$ or $w - r + \mu$. In either case, $\min\{m(y)|y \in D_j\} = w - r$ in $\mathbb{Z}_\mu$. Therefore, the computations that A_j and A_i carry out in Steps 7–9 end up with them holding random shares, $s^{k+1,i}(x)$ and $s^{k+1,j}(x)$ respectively, whose sum modulo μ equals $\min\{m(y)|y \in D_j\}$, as required.

3.3 Termination

After completing K iterations, the sum of the last incoming messages from all function nodes that are adjacent to X_i is $M_i := \sum_{\ell=1}^{t} R^K_{e_\ell \to i}$ (recall that $N_i = \{X_{e_\ell} : 1 \leq \ell \leq t\}$, where $e_\ell = (X_i, X_{j_\ell})$). Then, A_i needs to assign X_i the value x for which the corresponding entry in M_i is minimal. $R^K_{e_\ell \to i} = S^{K,i}_{e_\ell \to i} + S^{K,j_\ell}_{e_\ell \to i}$ where $S^{K,i}_{e_\ell \to i}$ is held by A_i and $S^{K,j_\ell}_{e_\ell \to i}$ is held by A_{j_ℓ}. We proceed to describe Protocol 3 that performs that computation securely.

In Steps 1–2, all agents that are connected to A_i send to it an $\mathcal{E}_i$-encryption of their share in the last message sent from their respective function node to X_i. (Recall that the encryption is applied on each entry of the vector independently.) Then, in Step 3, A_i selects a random masking scalar $r \in \mathbb{Z}_\mu$, defines $S_r = (r, \ldots, r) \in \mathbb{Z}_\mu^{|D_i|}$, and computes

$$\hat{M} := \mathcal{E}_i(S_r) \cdot \prod_{\ell=1}^{t} \mathcal{E}_i(S^{K,j_\ell}_{e_\ell \to i}) \cdot \mathcal{E}_i(\sum_{\ell=1}^{t} S^{K,i}_{e_\ell \to i}),$$

where the multiplication of vectors is done component-wise. Owing to the homomorphic property of $\mathcal{E}_i$, the vector $\hat{M}$ equals $\mathcal{E}_i(S_r + \sum_{\ell=1}^{t} S^{K,j_\ell}_{e_\ell \to i} + \sum_{\ell=1}^{t} S^{K,i}_{e_\ell \to i}) = \mathcal{E}_i(S_r + M_i)$, where M_i is as defined above. In Step 4, A_i sends a secret and random permutation of the entries of $\hat{M}$ to one of its neighbors, say A_{j_1}. Then, A_{j_1} decrypts it (Step 5) and finds the index h of the entry in the decrypted vector in which the $\min^*$ of the vector entries is obtained; the $\min^*$ of a set of values in $\mathbb{Z}_\mu$ is as defined in Eq. (18). A_{j_1} notifies A_i of the found index h (Step 6). Finally (Step 7), A_i assigns to X_i the value $x \in D_i$ which was mapped by π to h; in view of Eq. (9) and Lemma 3.1 this is the x for which $M_i(x)$ is minimal.

We conclude this section with two notes:

3.3.1 Controlling Function Nodes

Assume that agents A_i and A_j are connected through a function node X_e in the factor graph G'. When implementing Max-Sum, there is a need to decide which of those two agents controls X_e. In P-Max-Sum the question of who controls the function node becomes irrelevant since, by the privacy-preserving design of the algorithm, the operation of X_e is performed jointly by A_i and A_j.

Protocol 3 Computing the best assignment for X_i

```
1: for $\ell = 1, \dots, t$ do
2:     $A_{j_\ell}$ sends to $A_i$ the encryption of its share $\mathcal{E}_i(S^{K,j_\ell}_{e_\ell \to i})$.
3: $A_i$ selects uniformly at random $r \in \mathbb{Z}_\mu$ and computes $\hat{M} = \mathcal{E}_i(S_r) \cdot \prod_{\ell=1}^{t} \mathcal{E}_i(S^{K,j_\ell}_{e_\ell \to i}) \cdot \mathcal{E}_i(\sum_{\ell=1}^{t} S^{K,i}_{e_\ell \to i})$, where $S_r = (r, \dots, r) \in \mathbb{Z}_\mu^{|D_i|}$.
4: $A_i$ selects a secret random permutation $\pi$ on $D_i$ and sends $\pi(\hat{M})$ to $A_{j_1}$.
5: $A_{j_1}$ decrypts the entries of the received vector.
6: $A_{j_1}$ computes the index $h$ of the entry in which the min* of the decrypted vector is obtained and notifies $A_i$ of $h$.
7: $A_i$ assigns to $X_i$ the value that was mapped by $\pi$ to $h$.
```

3.3.2 Handling Unary Costs

If, in addition to the binary constraints there are also unary costs, then they too may be easily handled by our solution. Assume that agent A_i has a unary cost of $C_i(x)$ whenever its variable X_i is assigned the value $x \in D_i$, $1 \le i \le n$. Then in Eq. (14), agent A_i replaces its share $s^{k,i}(y)$ with $s^{k,i}(y) + C_i(x)$ while A_j replaces $s^{k,j}(y)$ with $s^{k,j}(y) + C_j(y)$. That modification will replace the cost $C_{i,j}(x,y)$, for the possible assignment of x to X_i and y to X_j, with $C_{i,j}(x,y) + C_i(x) + C_j(y)$. As such modification does not entail any new exchange of data between the agents, it offers the same privacy guarantees as in the case of binary costs only.

4. Correctness and Privacy

Here we prove that algorithm P-Max-Sum perfectly and privately simulates Max-Sum.

Theorem 4.1 *Algorithm* P-Max-Sum *perfectly simulates* Max-Sum *in the following sense. Let X_i and X_e be neighboring nodes in the factor graph G', where $e = (X_i, X_j)$. Let $S^{k,i}_{i \to e}$ and $S^{k,i}_{e \to i}$ be the corresponding two shares that X_i holds after the completion of the kth iteration in* P-Max-Sum*, and let $S^{k,j}_{i \to e}$ and $S^{k,j}_{e \to i}$ be the two shares held by X_j. Denote also by $Q^k_{i \to e}$ and $R^k_{e \to i}$ the messages sent between X_i and X_e in the kth iteration of* Max-Sum*. Then they satisfy Eqs. (10) and (11). Moreover, after completing K iterations, both algorithms will make the same assignment choices for all variables.*

Proof. The correctness claims of the theorem follow directly from the description and analysis of Protocols 1–3. □

Theorem 4.2 *Assuming that the ciphers $\mathcal{E}_i$, $1 \le i \le n$, are secure then Algorithm* P-Max-Sum *maintains topology, constraint, and decision privacy. If in addition we assume a one-time intervention of a trusted coordinator then* P-Max-Sum *also maintains agent privacy.*

Proof. First, we observe that P-Max-Sum does not preserve agent privacy due to the need for generating the private keys in the encryption functions $\mathcal{E}_i$. This problem can be resolved by the use of a trusted coordinator that intervenes only in the initialization stage. Each agent A_i will tell the coordinator who are its neighbors. The coordinator can then create, for each $1 \leq i \leq n$, key pairs for $\mathcal{E}_i$ and send them to all of A_i's neighbors, while A_i itself will get only the public key.[2]

We proceed to show that each of the protocols of which P-Max-Sum consists preserves topology, constraint, and decision privacy, as well as agent privacy under the above assumption of a trusted coordinator. In addition, we shall prove that all shares are uniformly random in $\mathbb{Z}_\mu$, namely, that each entry in every share can be any element of $\mathbb{Z}_\mu$ in equal probabilities.

The first protocol is the initiation protocol that was described in Section 3.1. Here there is no leakage of information since every agent communicates only with its neighbors, and the only values that are being communicated between them are random numbers. In addition, it is clear that all shares at this point are uniformly random.

As for Protocol 1, it too does not make any agent any wiser than it was before. Indeed, the only information that is communicated between agents in that protocol is in Steps 1–2 and then in Step 5. The communication in Steps 1–2 is encrypted by $\mathcal{E}_i$, which A_i cannot decipher. As for the communication in Step 5, it ends with A_{j_ℓ} learning the sum of random shares. Hence, none of those communications reveal any information about constraints or about the corresponding R or Q messages, and the resulting shares are still uniformly and randomly distributed.

In addition, both communications are between neighbors (hence, there is no topology or agent information leakage). Note that we devised a special treatment for the case $t = 1$ in order to prevent any agent from learning that it is the only neighbor of another agent. Since A_i performs the multiplication in Step 5 and it sends to A_{j_ℓ} only a single message W_ℓ, no neighbor of A_i learns how many neighbors A_i has.

Next, we discuss Protocol 2. Since it only involves two neighboring agents, there is no topology or agent information leakage. The only information that A_i receives in the course of the protocol is the encryption of A_j's shares by $\mathcal{E}_i$. Hence, under our assumption on the security of $\mathcal{E}_i$, A_i can extract no information from the data that it receives. As for A_j, it gets all values $w(y) = m(y) + r$, $y \in D_j$ (see Eq. (17)). However, as A_i used a random permutation in Step 4, A_j cannot associate any of the values $\{w(y)|y \in D_j\}$ with any $y \in D_j$. In addition, the usage of the random shift r prevents A_j from learning the actual values of $m(y)$, $y \in D_j$ (see Eq. (16)). However, A_j may learn some information on $m(y)$, which renders Protocol 2 not perfectly secure. Later on we characterize this leakage information.

Finally, we discuss the termination protocol, Protocol 3. Here too, there is no topology or agent information leakage since all communication is between neighbors. As in the previous protocols, A_i cannot use the information that it receives to extract sensitive information on its neighbors since it is encrypted by $\mathcal{E}_i$. As for the information that is passed to A_{j_1} it is protected by the same mechanisms as before – random shifts and random permutations. Finally, no agent apart from A_i may learn any information on A_i's assignment decision for X_i since the only agent that receives data that relates to that selection is A_{j_1}, but it too remains oblivious of the choice made thanks to the random and secret permutation π. $\square$

2. A setting of secure multi-party computation in which there is a coordinator to whom some computations are exported is referred to as "the mediated model"; see e.g., the works of Alwen, Shelat, and Visconti (2008) and Alwen, Katz, Lindell, Persiano, Shelat, and Visconti (2009).

We now turn to characterize the information that A_j may extract on the values $m(y)$, $y \in D_j$, that were defined in Eq. (16). We recall that $m(y)$ are integers in the range $[0, B_k]$, where B_k is defined in Eq. (8). Denote $k := |D_j|$ and assume that $m_1 \leq m_2 \leq \cdots \leq m_k$ is an ordering of the set $\{m(y)|y \in D_j\}$. Then, as A_j gets a random and secret permutation of $\{w(y) = m(y) + r|y \in D_j\}$, and it knows that all of $m(y)$ are bounded to $[0, B_k]$ where $B_k \leq (\mu - 1)/2$, it can find the min* of the latter set of values (see Step 6 of Protocol 2) and recover from it all differences $m_{\ell+1} - m_\ell$, $1 \leq \ell < k$. In addition, it can deduce that m_1 can be any of the values in the set $\{0, 1, \ldots, B_k - (m_k - m_1)\}$, with equal probabilities. Going back to the example after Lemma 3.1, the usage of the random shift r modulo $\mu = 30$ enabled us there to deduce that the set of shifted values $\{2, 5, 28\}$ originated from one of the original sequences $(0, 4, 7)$, $(1, 5, 8)$, $(2, 6, 9)$ or $(3, 7, 10)$, but they were all equally probable (in that example $B_k = 10$ and $m_k - m_1 = 7$).

Such information leakage does not enable A_j to infer any information on constraints of A_i or any other agent, because the usage of the random permutation prevents A_j from associating any of the values m_ℓ with any $y \in D_j$. However, by comparing messages that are sent from A_i to A_j in Step 4 of Protocol 2, in subsequent iterations, A_j may infer that two such messages are identical. Hence, if the algorithm converges, A_j may detect that. A detection of early convergence may imply that the factor graph has a small number of cycles or that it is even a tree. Such inferences, even though they do not allow A_j to draw any conclusions on existence of specific edges, are inconsistent with perfect topology privacy. If such inferences are to be avoided, then finding the minimum in Protocol 2 should be carried out differently. For example, instead of sending the whole set of values $\{W(y) : y \in D_j\}$, A_i may retain those values and perform on them oblivious sorting by sending to A_j in each time a pair of encrypted values to compare, where in each such query it uses a new random shift r. Such a procedure completely prevents the topology information leakage described above, but it requires more communication rounds.

Concluding remarks. There exist several complete privacy-preserving DCOP algorithms. While there is no point to compare their efficiency to that of P-Max-Sum (for obvious scaling problems of complete algorithms), it is interesting to observe their privacy features. The algorithm P-SyncBB (Grinshpoun & Tassa, 2014) preserves constraint and topology privacy, but not agent or decision privacy. The enhanced P-SyncBB (Grinshpoun & Tassa, 2016) preserves constraint, topology and decision privacy, but not agent privacy. As for the study of Léauté and Faltings (2013), they presented a sequence of three privacy-preserving versions of DPOP: P-DPOP$^{(+)}$, P$^{3/2}$-DPOP$^{(+)}$, and P^2-DPOP$^{(+)}$. All three versions preserve agent privacy and partial topology privacy. The least private and most efficient version, P-DPOP$^{(+)}$, preserves constraint and decision privacy only partially, as it may leak related information. P$^{3/2}$-DPOP$^{(+)}$ preserves decision privacy fully but it still respects constraint privacy only partially. The last version, P^2-DPOP$^{(+)}$ (most private, least efficient), preserves constraint and decision privacy fully.

5. Private Versions of Variants of Max-Sum

In this section we discuss privacy-preserving versions of variants of Max-Sum.

5.1 Finding Anytime Solutions for Max-Sum

The basic Max-Sum algorithm performs a preset number K of iterations and then infers the solution, which it outputs from the results of the last iteration only. However, like other non-monotonic incomplete DCOP algorithms, it may be possible that a better solution could have been found by Max-Sum in an earlier iteration of its run. The best solution visited throughout the run of the algorithm is the *anytime* solution. It is commonly preserved using an anytime mechanism as the one proposed by Zivan et al. (2014). In general, in order to report the best solution visited, the Max-Sum algorithm needs to perform the termination step after each iteration, compute the overall cost, which results from the selected assignments at that point, and if that overall cost is the minimal so far, record that cost and the corresponding assignments.

A privacy-preserving version of the Max-Sum algorithm that reports the anytime solution can be achieved by introducing modifications to P-Max-Sum as we describe below. In what follows we assume that all agents know each other and that $X_1, \ldots, X_n$ is a publicly known ordering of all variables. We shall refer hereinafter to this privacy-preserving version of the Max-Sum algorithm that reports the anytime solution as P-Max-Sum_AT.

1. The agents perform the termination stage, as described in Section 3.3, after each iteration. In particular, Protocol 3 is executed for each agent after each iteration.
2. As a consequence, each of the agents discovers the currently best assignment for its variable. We denote the assignment that was found for X_i by a_i, $1 \leq i \leq n$.
3. Assume that X_i and X_j are two neighboring nodes in the constraint graph and that $i < j$. Then agent A_i sends to A_j its currently selected assignment a_i.
4. Each agent A_j, $j \geq 2$, computes $b_j := \sum_i C_{i,j}(a_i, a_j)$ where the sum is over all $i < j$ such that X_i and X_j are neighbors in the constraint graph.
5. Hence, $b := \sum_{j=2}^{n} b_j$ is the overall cost that the current assignment $(X_1 = a_1, \ldots, X_n = a_n)$ causes. The agents engage in a secure summation protocol that ends with A_1 recovering b, but no agent learns anything about the value b_j of other agents (see (Grinshpoun & Tassa, 2014, Protocol 2)).
6. Agent A_1 checks whether the current overall cost b is the minimal so far. If it is, A_1 informs all its peers to store their current assignments as the currently best solution.
7. After K iterations the algorithm stops. At this stage, each agent A_i has an assignment a_i, $1 \leq i \leq n$, such that $(a_1, \ldots, a_n)$ is the best solution that was visited throughout the execution of the algorithm.

5.1.1 Privacy

P-Max-Sum_AT respects constraint and topology privacy in similarity to P-Max-Sum, since the additional protocol steps do not reveal any information that relates either to topological properties of the constraint graph nor to constraint values. We proceed to discuss the remaining two notions of privacy.

P-Max-Sum_AT does not respect agent privacy (in similarity to P-Max-Sum in the absence of a trusted coordinator), because of the need to create an ordering of all agents. By using a trusted coordinator, that intervenes in each iteration in order to perform the summation and recovery of the current overall cost, we may respect agent privacy as well.

In contrast to P-Max-Sum that respected assignment/decision privacy, P-Max-Sum_AT does not respect that type of privacy. It fails to do so, at least partially, since every agent A_j gets to know the assignments of its neighbors that precede it in the order. (Note, however, that no agent learns assignment values of variables with which it is not constrained.)

The above implementation of P-Max-Sum_AT may be enhanced so that it respects also assignment/decision privacy. However, such an enhancement imposes a dear toll in communication costs. We therefore proceed to describe it only briefly.

Let $C_{i,j}$ be the constraint matrix between two agents A_i and A_j. It is a matrix of dimensions $|D_i| \times |D_j|$ where the (k, ℓ)th entry in it equals the constraint value if X_i is assigned the kth value in D_i and X_j is assigned the ℓth value in D_j. If X_i and X_j are not constrained then $C_{i,j}$ is the zero matrix. Agents A_i and A_j can then compute two matrices of the same dimensions, $C^1_{i,j}$ and $C^n_{i,j}$ with entries in $\mathbb{Z}_\mu$, such that all entries in $C^1_{i,j}$ are selected uniformly at random from $\mathbb{Z}_\mu$ and then $C^n_{i,j} := C_{i,j} - C^1_{i,j} \bmod \mu$.

After each iteration, every pair of agents (even pairs of agents that are not constrained) will reconstruct their constraint matrix $C_{i,j}$ by selecting a new random ordering of their domains D_i and D_j and then will compute a new random splitting of $C_{i,j}$ as described above into $C^1_{i,j}$ and $C^n_{i,j}$. They will send to A_1 the share matrix $C^1_{i,j}$ and to A_n the other share matrix $C^n_{i,j}$. In addition, A_i will inform both A_1 and A_n of the index $u_{i,j}$ of its assignment selection at that stage, a_i, in the new random ordering, and A_j will inform both A_1 and A_n of the index $v_{i,j}$ of its assignment selection at that stage, a_j.

After A_1 and A_n receive the above inputs from all $\binom{n}{2}$ pairs of agents, A_1 may compute the following sum

$$b_1 := \sum_{1 \le i < j \le n} C^1_{i,j}(u_{i,j}, v_{i,j}) \,,$$

while A_n can compute

$$b_n := \sum_{1 \le i < j \le n} C^n_{i,j}(u_{i,j}, v_{i,j}) \,,$$

where all additions are modulo μ. Finally, A_n sends to A_1 the value b_n and A_1 computes $b = b_1 + b_n \bmod \mu$. It is easy to see that b equals the overall cost that results from the current assignment selection $(X_1 = a_1, \ldots, X_n = a_n)$. It is also clear that by using a new random secret sharing for the constraint matrix in each iteration as well as new random orderings of the domain values, neither A_1 nor A_n can extract any constraint information or assignment information. Our demand of performing the above procedure for all $\binom{n}{2}$ pairs of agents also prevents A_1 and A_n from learning topology information.

5.2 Bounded Max-Sum

Bounded Max-Sum (Rogers et al., 2011) is a version of Max-Sum that starts with a preliminary phase in which the factor graph is reduced to a tree subgraph, and then the Max-Sum algorithm is executed on that tree subgraph until it converges to the optimal solution for this tree structured subgraph. The rationale behind this approach is that by implementing such a preprocessing phase, the protocol is guaranteed to converge in a linear number of iterations to the optimal solution of the problem represented by the subgraph. By accounting for the worse case (highest cost) for every edge removed from the original graph, we can calculate a bound on the distance of the cost of the solution found from the cost of the optimal solution for the original problem.

In more detail, every pair of neighboring agents selects a weight for the edge between them, which is a function of the cost values for the binary constraint between them, and then all of the agents find a Maximum Spanning Tree (MST) for the factor graph with those weights. Different studies considered different selections for those edge weights (Rogers et al., 2011; Rollon & Larrosa, 2012). However, the manner in which the edge weights are selected is immaterial for our present discussion.

When privacy becomes a concern, the agents need to solve the following secure multi-party computation problem. Each of the agents controls its own variable node in the constraint graph, and it knows the weights only of the edges that are adjacent to its node; they then need to find an MST for that distributed graph in a way that preserves their private information (being their local topology and edge weight information). A solution to that problem should provide to each agent the list of edges adjacent to its node that it should remove from the constraint graph. From that point on, the agents may proceed with the protocol P-Max-Sum, applied to the new reduced tree graph.

Problems of secure multi-party computations on distributed graphs are of much interest and importance. Nonetheless, due to their apparent difficulty, very few studies were published so far on such problems. The first such study was by Brickell and Shmatikov (2005) who presented new algorithms for privacy-preserving computation of the all-pairs-shortest-distance and single-source-shortest-distance problems. A more recent study is that of Aly, Cuvelier, Mawet, Pereira, and Vyve (2013) who designed secure multi-party computation techniques for the shortest path and the maximum flow problems. Another example is the work of Keller and Scholl (2014) who presented oblivious implementations of several data structures for secure multi-party computation and then offered a secure computation of Dijkstra's shortest path algorithm on general graphs, where the graph structure is secret.

However, the problem of privacy-preserving computation of the MST was never addressed until the recent study by Laud (2015). In that study, Laud shows how the MST-finding algorithm by Awerbuch and Shiloach (1987) can be executed without revealing any details about the underlying graph, beside its size.[3] The size in our case equals the number of agents. As our basic version of the P-Max-Sum protocol assumes that that number is known to all, such leakage of information does not pose any problem.

Concluding remarks. Since Bounded Max-Sum differs from Max-Sum only in a preliminary preprocessing stage, it is possible to solve the MST computation problem by a one-time intervention of a trusted coordinator. However, while previously the coordinator was trusted only with information on the graph topology, here it must be trusted also with the weight information. Therefore, in cases where the coordinator may be trusted also with that additional information, the MST computation problem can be solved without resorting to a secure multi-party protocol (such as Laud's). However, if it cannot be trusted with

3. Note that we are interested in finding the Maximum Spanning Tree, while Laud considered the problem of computing the Minimum Spanning Tree. However, each of those problems can be easily reduced to the other by negating the edge weights.

that additional information, or in the absence of such a trusted coordinator, then Laud's protocol (or any other algorithm for that purpose that might be developed in the future) is the way to go.

5.3 Max-Sum_AD

The Max-Sum_AD algorithm (Zivan & Peled, 2012) is an implementation of Max-Sum with alternating direction of messages. It operates as follows:

- Each agent has a unique index, so that the set of indices induces a total order on the set of agents.
- The protocol's run is separated into phases, where each phase consists of k_0 consecutive iterations, k_0 being a publicly known fixed integer. A customary setting of k_0 is $k_0 = n$ (number of agents).
- The odd phases are *upstream* phases, while the even ones are *downstream*. In an upstream phase, messages are computed as described in Section 2.3 only in the upstream direction, while in the opposite direction the messages are "frozen", as we explain below. In downstream phases, the opposite occurs.
- Assume that X_i and X_j are neighboring agents, which are connected through the function node X_e. Assume further that $i < j$ (i and j are the order-inducing indices). Then in an upstream phase, messages will be computed and sent from X_i to X_e and from X_e to X_j, as described in Eqs. (1)–(3) in Section 2.3. However, during such a phase, the messages from X_j to X_e and from X_e to X_i will be either zero (in the first phase) or a replication of the last messages along those edges that were sent during the preceding downstream phase.

Max-Sum_AD guarantees convergence in each phase of the algorithm, though not necessarily to the optimal solution. Unlike Bounded Max-Sum, it does so without eliminating edges of the factor graph.

In view of the above, modifying P-Max-Sum so that it implements Max-Sum_AD only requires creating an ordering of the agents. In the basic version of P-Max-Sum that does not respect agent privacy, such an ordering can be jointly computed by the agents in a public manner. In the version of P-Max-Sum that assumes a one-time intervention of a trusted coordinator in order to achieve also agent privacy, the coordinator can generate such an order and then inform each agent of its index in the order.

5.4 Max-Sum_ADVP

The Max-Sum_ADVP algorithm (Zivan & Peled, 2012) is a variant of Max-Sum_AD that adds value propagation. While Max-Sum_AD guarantees convergence in each phase of the algorithm, Max-Sum_ADVP guarantees cross-phase convergence and evidently produces better results than all other versions of Max-Sum (Zivan & Peled, 2012; Zivan, Okamoto, Parash, Cohen, & Peled, 2017).

The modifications introduced in Max-Sum_ADVP, with respect to Max-Sum_AD, are as follows: At the beginning of each iteration, each variable node X_i selects a currently optimal assignment from D_i, exactly as done in Max-Sum at the termination stage. Specifically, at the beginning of iteration $k+1$, the variable node X_i computes $M_i^{k+1} := \sum_{X_e \in N_i} R_{e\to i}^k$ and then sets $x^{k+1} = \text{arg-min}_{x\in D_i} M_i^{k+1}(x)$. Assume next that X_j is a neighbor of X_i and that X_e is the function node between them. Then, when X_i sends to X_e the message $Q_{i\to e}^{k+1}$, it propagates alongside with it also the value x^{k+1}. In the subsequent iteration when X_e computes the message $R_{e\to j}^{k+2}$, then instead of Eq. (3), by which

$$R_{e\to j}^{k+2}(y) = \min_{x\in D_i} \left[C_{i,j}(x,y) + Q_{i\to e}^{k+1}(x) \right] \quad \forall y \in D_j \,,$$

it computes and sends the following message,

$$R_{e\to j}^{k+2}(y) = C_{i,j}(x^{k+1}, y) + Q_{i\to e}^{k+1}(x^{k+1}) \quad \forall y \in D_j \,. \tag{20}$$

Best performance is obtained if that practice is executed only starting from the third phase (Zivan & Peled, 2012).

We now turn to describe a privacy-preserving implementation of this version of Max-Sum. Let X_i and X_j be variable nodes that are connected through the function node X_e. At the beginning of the $(k+1)$th iteration, X_i computes $x^{k+1} = \text{arg-min}_{x\in D_i} M_i^{k+1}(x)$ by invoking Protocol 3. However, as opposed to the non-private implementation of Max-Sum_ADVP, it does not send that value to the function node, for the sake of privacy preservation vis-a-vis X_j. Instead, X_i uses x^{k+1} as an input when X_i and X_j compute shares in the message $R_{e\to j}^{k+2}$, Eq. (20), that X_e sends to X_i in the subsequent iteration. They do this by executing Protocol 4 below (instead of Protocol 2 which is based on Eqs. (2) and (3)).

In that protocol, we denote by $s^{k+1,i}(x)$ and $s^{k+1,j}(x)$ the shares in $Q_{i\to e}^{k+1}(x)$ which A_i and A_j hold respectively, for all $x \in D_i$. Those shares were already computed in the $(k+1)$th iteration using Protocol 1. Recall that such a sharing means that each of those shares is a random number in $\mathbb{Z}_\mu$, that by its own carries no information, and that

$$Q_{i\to e}^{k+1}(x) = s^{k+1,i}(x) + s^{k+1,j}(x) \tag{21}$$

(where all additions hereinafter are modulo μ). Let us denote the sought-after shares in $R_{e\to j}^{k+2}(y)$, for all $y \in D_j$, by $s^{k+2,i}(y)$ and $s^{k+2,j}(y)$. Namely, the former share is to be held by X_i, the latter is to be held by X_j, both should be uniformly and randomly distributed over $\mathbb{Z}_\mu$, and

$$R_{e\to j}^{k+2}(y) = s^{k+2,i}(y) + s^{k+2,j}(y) \,. \tag{22}$$

The protocol starts with X_j selecting for itself a random share $s^{k+2,j}(y)$ (Step 1). The rest of the protocol is designed so that X_i gets the corresponding complement share

$$s^{k+2,i}(y) = R_{e\to j}^{k+2}(y) - s^{k+2,j}(y) \,.$$

In view of Eqs. (20) and (21),

$$s^{k+2,i}(y) = C_{i,j}(x^{k+1}, y) + s^{k+1,i}(x^{k+1}) + \sigma(x^{k+1}) \,, \tag{23}$$

where, for each $x \in D_i$,

$$\sigma(x) := s^{k+1,j}(x) - s^{k+2,j}(y) \,. \tag{24}$$

Since X_i knows the value selection x^{k+1} then it knows the first two addends on the right-hand side of Eq. (23). In order to compute the last addend, $\sigma(x^{k+1})$, it needs to cooperate with X_j, since X_j knows $\sigma(x)$ for all $x \in D_i$, but only X_i knows x^{k+1}. This is an instance of the most basic problem of secure multi-party computation – Oblivious Transfer (Rabin, 1981), and it can be solved by one of the many protocols that were suggested for it, e.g., the works of Aiello, Ishai, and Reingold (2001), Naor and Pinkas (2005) and Laur and Lipmaa (2007). Specifically, after A_j computes $\sigma(x)$ for all $x \in D_i$ (Step 2), A_i and A_j engage in a 1-out-of-$|D_i|$ oblivious transfer protocol, where A_i's input is x_k and A_j's input is $\{\sigma(x) : x \in D_i\}$. At the end, A_i learns the value $\sigma(x^{k+1})$ only, without learning the value of $\sigma(x)$ for other values of x, while A_j remains oblivious of x^{k+1} that was used by A_i for selecting the value $\sigma(x^{k+1})$ to be obliviously transferred to it (Step 3). Finally, A_i can proceed to compute $s^{k+2,i}(y)$ (Step 4).

Protocol 4 Computing shares in messages that emerge from a function node in Max-Sum_ADVP

```
1: $A_j$ selects a random share $s^{k+2,j}(y) \in \mathbb{Z}_\mu$ in $R^{k+2}_{e \to j}(y)$ for all $y \in D_j$.
2: $A_j$ computes $\sigma(x)$ for all $x \in D_i$, using Eq. (24).
3: $A_i$ and $A_j$ engage in a 1-out-of-$|D_i|$ oblivious transfer protocol, where $A_i$'s input is $x_k$ and $A_j$'s
   input is $\{\sigma(x) : x \in D_i\}$.
4: $A_i$ computes $s^{k+2,i}(y)$, using Eq. (23).
```

6. Efficiency Analysis

In this section we analyze the price of privacy, by comparing the computational and communication overhead of P-Max-Sum with respect to the basic Max-Sum, as well as that of the private implementations of the variants of Max-Sum, that were presented in Section 5, with respect to their non-private counterpart. In Sections 6.1–6.4 we analyze the computational and communication overhead that Protocols 1–4 inflict on any given node. Specifically, we compare the computations that each node has to do and the total number and size of messages that each node has to send out in each of those protocols, on one hand, and in the equivalent stage of the non-private algorithm on the other hand. In doing so we always fix a variable node X_i and let t be its number of neighbors; we also denote by $d := \max_{1 \le j \le n} |D_j|$ the size of the largest variable domain. In Section 6.5 we summarize the overall costs for both P-Max-Sum and P-Max-Sum_AT. Then, in Section 6.6, we present experimental results regarding the computational overhead in various problem settings. Finally, we present in Section 6.7 simulator experiments that compare the overall runtime of P-Max-Sum to that of the basic Max-Sum. In all of our experiments, the modulus of the encryption functions and the size of additive group $\mathbb{Z}_\mu$ in which all shares take values are of size 512 bits.

6.1 Analysis of Protocol 1

Protocol 1 is used for computing shares in messages that emerge from variable nodes. The communication and computational costs that a single invocation of Protocol 1 (for some agent A_i) incurs are (at most) as follows:

1. td encryptions (Steps 1–2).
2. t messages of overall size $td\lceil\log\mu\rceil$, since each entry in the shares is an element in $\mathbb{Z}_\mu$ (Steps 1–2).
3. $(2t-1)d$ additions/subtractions of $\lceil\log\mu\rceil$-bit integers (Step 4).
4. $(2t-1)d$ multiplications/divisions of $\lceil\log\mu\rceil$-bit integers (Step 5).
5. t messages of overall size $td\lceil\log\mu\rceil$ (Step 5).
6. td decryptions (Step 6).

Note that the costs due to Steps 1–2 and 6 (items 1 and 6 in the list above) are inflicted on A_i's neighbors; the other costs are inflected on A_i.

In the non-private Max-Sum algorithm, the messages from a variable node, see Eq. (1), are sent in the clear, without any protective mechanism. The communication and computational costs that they entail are as follows:

1. The nodes adjacent to X_i send to it the messages $R^k_{f\to i}$. Cost: those are t messages where each one is a vector of integers of at most d entries.
2. X_i adds all incoming messages from the previous step. Cost: $t-1$ additions of d-dimensional integer vectors.
3. For each of its neighbors, X_i subtracts from the sum in the previous step the message that it got from that neighbor. Cost: t additions of d-dimensional integer vectors.
4. X_i sends the proper sum to each neighbor: Cost: t messages of overall size td integers.

In order to assess the price of privacy, we focus only on encryption and decryption operations, since the other computations that Protocols 1–3 perform (modular additions/subtractions/multiplications/divisions, random numbers' generation, and computing minima) have computational costs which are few orders of magnitude smaller than those of the cryptographic operations. Therefore, in view of the above detailed analysis of Protocol 1, its computational overhead in each iteration for each node is (at most)

$$\Gamma_1 := td(C_{enc} + C_{dec})\,, \tag{25}$$

where C_{enc} and C_{dec} are the costs of encryption and decryption, respectively. As for the communication overhead, the total number of messages is $2t$ in both Protocol 1 and its non-private counterpart, but their total length is $2td\lceil\log\mu\rceil$ bits in Protocol 1, as opposed to $2td$ integers in the non-private protocol.

6.2 Analysis of Protocol 2

As explained in Section 6.1, we focus on the cryptographic operations which are the most costly. The communication and computational costs that Protocol 2 (for some agent A_i) incurs are (at most) as follows:

1. td encryptions (Step 1). Indeed, let X_j be one of X_i's neighbors. Then Step 1 in Protocol 2 has to be carried out by X_i vis-a-vis X_j just once, and it includes $|D_j| \leq d$ encryptions. It should be noted that while Protocol 2 has to be executed for each $x \in D_i$, Step 1 in it (as opposed to all subsequent steps) can be carried out by X_i vis-a-vis X_j just once.
2. t messages of total length $td\lceil \log \mu \rceil$ bits (Step 1).
3. td^2 encryptions (Step 3). The factor d^2 stems from the fact that if X_j is one of X_i's neighbors then X_i has to perform the encryption in this step for every selection of $x \in D_i$ and $y \in D_j$.
4. t messages of total length of $td^2\lceil \log \mu \rceil$ bits (Step 4).
5. td^2 decryptions (Step 5).
6. t messages of total size of $td\lceil \log \mu \rceil$ bits (Step 8).

Therefore, Protocol 2's computational overhead in each iteration for each node is (at most)

$$\Gamma_2 := td((d+1) \cdot C_{enc} + d \cdot C_{dec})\,. \tag{26}$$

It also requires each node to send out $3t$ messages of total length $td(d+2)\lceil \log \mu \rceil$ bits.

In comparison, the non-private Max-Sum performs no cryptographic computations, and the communication cost for X_i is only t outgoing messages of overall length of td integers.

6.3 Analysis of Protocol 3

The communication and computational costs that a single invocation of Protocol 3 (for some agent A_i) incurs are (at most) as follows:

1. td encryptions (Steps 1–2).
2. t messages of total length $td\lceil \log \mu \rceil$ bits (Steps 1–2).
3. 1 encryption (Step 3).
4. 1 message of length of $d\lceil \log \mu \rceil$ bits (Step 4).
5. d decryptions (Step 5).
6. 1 message that consists of one integer only (Step 6).

Therefore, Protocol 3's computational overhead in each iteration for each node is (at most)

$$\Gamma_3 := (td+1) \cdot C_{enc} + d \cdot C_{dec}\,. \tag{27}$$

It also requires each node to send out $t+2$ messages of total length $(t+1)d\lceil \log \mu \rceil + 16$ bits (we assume that the message in Step 6 can be encoded as a short integer, since it represents an index).

In comparison, the non-private Max-Sum performs no cryptographic computations, and the communication cost for X_i is only t outgoing messages of overall length of td integers.

6.4 Analysis of Protocol 4

The communication and computational costs that Protocol 4 inflicts on A_i and A_j are due to the oblivious transfer that takes place in Step 3. A_i and A_j need to perform a single 1-out-of-$|D_i|$ oblivious transfer protocol in order for A_i to learn the relevant $\sigma(x^{k+1})$ for each of the assignments $y \in D_j$ (namely, at the end of the oblivious transfer A_i learns $|D_j|$ values). Utilizing the efficient oblivious transfer protocols of Naor and Pinkas (2001) the corresponding overhead is $|D_j|$ exponentiations that A_j has to do, one for each $y \in D_j$. We note that A_j has also to perform $|D_i| - 1$ multiplications and $|D_i|$ hash function evaluations but, as stated before, we ignore those due to their significantly lower cost; in addition, A_j has to perform once in the outset of the protocol $|D_i|$ exponentiations. Since an exponentiation is comparable to a single encryption, the bound on the total overhead for each node is comparable to td encryptions.

The communication overhead consists of two rounds: in the first one A_i sends to A_j a single message of length $|D_j|\lceil \log \mu \rceil$ bits; in the second, A_j sends back a message of length $|D_i||D_j|\lceil \log \mu \rceil$ bits.[4]

6.5 Overall Computational Overhead

Let K be the number of iterations that P-Max-Sum performs. Then the overall cost of privacy, in terms of runtime, for each node is bounded by

$$(\Gamma_1 + \Gamma_2) \cdot K + \Gamma_3 \,,$$

where Γ_1, Γ_2, and Γ_3 are the costs of Protocols 1, 2, 3, respectively, and are given in Eqs. (25)–(27). As for P-Max-Sum_AT, its privacy overhead is bounded by

$$(\Gamma_1 + \Gamma_2 + \Gamma_3) \cdot K \,.$$

We note that the runtime of P-Max-Sum and P-Max-Sum_AT is independent of n, the number of agents. However, it does depend on the maximal degree of a node in the constraint graph.

We now comment on the privacy overhead of the remaining variants. The operation of the privacy-preserving version of Bounded Max-Sum is similar to that of P-Max-Sum only that the value of t (that denoted the degree of a node) could be reduced in Bounded Max-Sum since the algorithm starts by trimming the factor graph into a tree, and hence the corresponding costs are reduced. However, the algorithm starts by implementing a privacy-preserving MST algorithm. The cost of that operation depends on the selection of algorithm. As noted earlier, the only algorithm that exists currently for that purpose is the one which was recently proposed by Laud (2015); the reader is referred to Section 7 there for a description of that algorithm and an analysis of its efficiency.

The operation of the privacy-preserving version of Max-Sum_AD is similar to that of P-Max-Sum, but it will have reduced computational and communication costs (for a given number of iterations) since in each iteration only "half" of the messages need to be computed and sent. The same comment applies also to Max-Sum_ADVP with the additional observation that in this variant, Protocol 4 is executed instead of Protocol 2;

4. For more details, the reader is referred to the work of Naor and Pinkas (2001).

comparing the computational overhead of the two we see that Protocol 4 is cheaper (since its overhead bound is comparable to td encryptions while that of Protocol 2 is dominated by td^2 encryptions, see Sections 6.2 and 6.4).

6.6 Computational Overhead Experiments

According to Theorem 4.1, P-Max-Sum perfectly simulates Max-Sum, i.e., it reaches exactly the same solutions as Max-Sum. A thorough experimental evaluation of the solution costs of Max-Sum (with and without applying the anytime mechanism), including comparisons to other local search algorithms, was conducted by Zivan et al. (2017). What remains to be investigated is the overhead of privacy preservation. The communication overhead was analyzed in Sections 6.1–6.4 and was shown to be quite small. Contrary to that, the analysis in Section 6.5 reveals that the computational overhead is much more substantial due to the cryptographic operations. Therefore, our experimental evaluation focuses on the computational overhead of P-Max-Sum (and P-Max-Sum_AT).

To realize the actual time it will take P-Max-Sum to run we followed the *simulated time* approach (Sultanik, Lass, & Regli, 2008) by measuring the time of atomic operations performed in the algorithm and then counting the non-concurrent times these operations are performed. This is a common practice in synchronous algorithms that perform the same operations in each round. The significant advantage of this approach is that its reported results are implementation-independent and (almost) environment-independent (except for the runtimes of the atomic operations in the given environment). We measured the runtimes of the encryption and decryption operations by averaging multiple runs of the common Java implementation of the Paillier cryptosystem[5] on a hardware comprised of an Intel i7-4600U processor and 16GB memory. Our tests show that C_{enc} takes at most 2 milliseconds, while C_{dec} takes at most 3 milliseconds. The runtimes of other operations on multiple precision variables are considerably smaller (e.g., a generation of a random number takes about 350 nanoseconds, addition about 170 nanoseconds, modulo multiplication about 1 microsecond) and are thus omitted.

There are three factors that directly affect the computational overhead of P-Max-Sum (see Section 6.5): the maximal degree of a node in the constraints graph (t), the size of the largest variable domain (d), and the number of performed iterations (K). The influence of these factors is investigated by considering four sets of problems: unstructured random problems, scale-free problems, structured graph coloring problems, and realistic meeting scheduling problems. In all the presented results each data point represents the average over 100 independently generated problems, and the computational overhead is measured in minutes.

We start by evaluating the direct effect of t and d on the computational overhead in unstructured random problems. Our basic setting consists of $n = 100$ agents, constraint density of $p = 0.1$, and domain sizes of $d = 5$. We also need to select a value for K, being the number of iterations. On unstructured problems Max-Sum is known to perform well in the first few iterations and then it traverses low quality solutions (Zivan & Peled, 2012; Zivan et al., 2014). However, occasionally the extreme exploration is beneficial since the algorithm in some iteration explores a high quality assignment. Therefore, it is beneficial to apply the

5. `http://www.csee.umbc.edu/~kunliu1/research/Paillier.html`

anytime mechanism in this setting (Zivan et al., 2014). We run the algorithm for $K = 50$ iterations, in which Max-Sum with anytime usually displays most of the improvement in terms of solution quality.

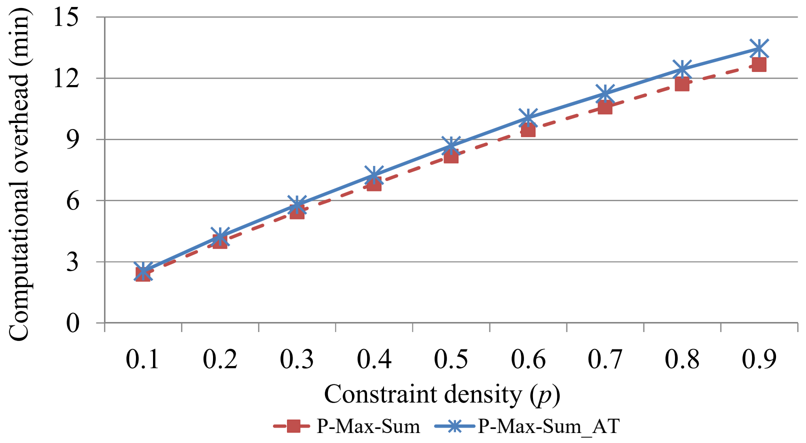

Figure 2: Computational overhead as a function of p.

In order to examine the effect of the parameter t on the computational overhead, it is customary in DCOP literature to vary the constraint density p. Figure 2 depicts the effect of the parameter t by varying the constraint density $p = 0.1, \ldots, 0.9$. The computational overhead displays linear growth with respect to the constraint density, which supports the analysis in Section 6.5. As expected, the overhead of P-Max-Sum_AT is slightly higher than that of P-Max-Sum.

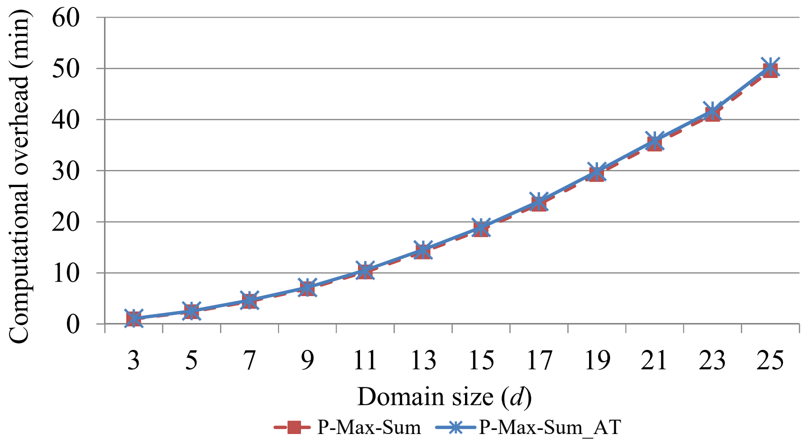

Figure 3: Computational overhead as a function of d.

Figure 3 depicts the effect of the parameter d by varying the domain sizes $d = 3, \ldots, 25$. Here, the computational overhead displays quadratic growth with respect to the constraint density, which again supports the analysis in Section 6.5. The same trends of Figures 2 and 3 are repeated when varying the constraint density and domain sizes in the other problem settings, so we do not present these experiments.

Next, we focus on scalability by evaluating the computational overhead when varying the number of agents n. We note that the runtime of P-Max-Sum and P-Max-Sum_AT is independent of the number of agents. Yet, depending on the type of the underlying constraint graph, the number of nodes (agents) in the graph may indirectly influence the maximal degree of a node in that graph. For this purpose we consider two types of underlying graphs – random graphs and scale-free networks.

A random graph is obtained by starting with a set of n isolated nodes and adding successive edges between them at random. In our experiments we relied on the model of Erdős and Rényi (1959) in order to generate random graphs. Each node is connected to m randomly chosen nodes, resulting in an average node degree of $2m$. Contrary to that, in a scale-free graph, the distribution of node degrees follows a power law. Many real-world networks, such as the Internet and various social networks, are considered to have a scale-free structure. We used the model of Barabási and Albert (1999) in order to generate random scale-free networks. The network begins with an initial connected network of 6 nodes. New nodes are added to the network one at a time. Each new node is connected to m existing nodes with a probability that is proportional to the number of links that the existing nodes already have.

While the construction processes of these graph types are quite different from each other, they both share two important parameters – the number of nodes in the network (n) and the number of new links (neighbors) for each added node (m). Consequently, we can plot our results for the two graph types together, by fixing $m = 5$ (which results in an average node degree of 10), and varying the number of agents $n = 100, \ldots, 1000$.

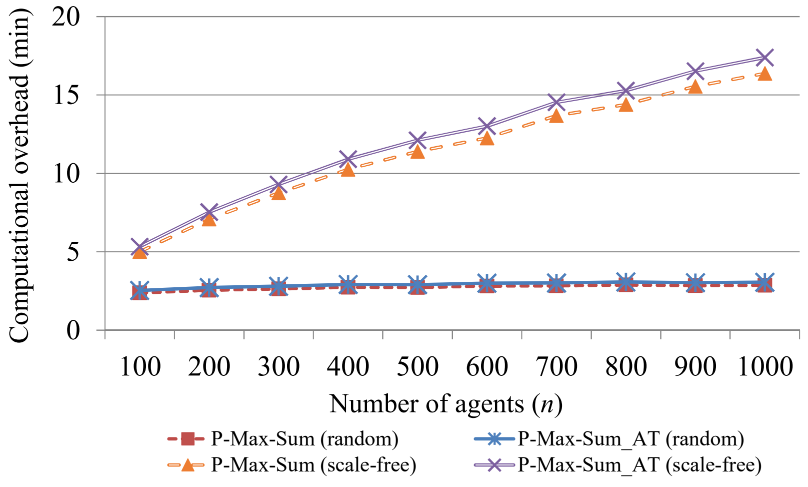

Figure 4: Computational overhead as a function of n on two graph types.

Figure 4 shows the scalability of P-Max-Sum and P-Max-Sum_AT on random graphs and scale-free networks. Increasing the number of nodes in random graphs has almost no effect on the maximal degree of a node in the graph, hence the respective computational overhead is marginal. However, scale-free networks are shown to have a higher computational overhead. This is not surprising since scale-free networks consist of several highly-connected hubs, and that affects the parameter t. Nevertheless, as depicted in Figure 4, this dependence is linear with a moderate growth factor.

Finally, we turn to structured and realistic problems, following the settings of Zivan et al. (2014). The structured problems are 3-color graph coloring problems, in which for all i, j, $C_{i,j}(x, y)$ equals 1 if $x = y$ and 0 if $x \neq y$. Such problems are commonly used in DCOP formulations of resource allocation problems (Zhang et al., 2005; Farinelli et al., 2008). In this setting, the "structure" is not in the constraint graph (the underlying graph is random), but rather in the highly-structured constraint functions. The structure of the constraint functions has no effect on the parameters t and d. However, it has an extreme effect on the needed number of iterations K, since Max-Sum was shown to converge in this setting in as few as $K = 5$ iterations (Zivan et al., 2014).

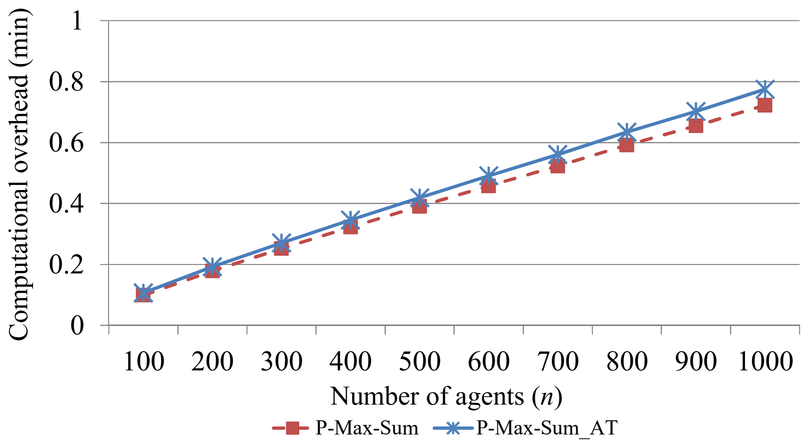

Figure 5: Computational overhead on structured graph coloring problems.

Figure 5 displays the computational overhead of P-Max-Sum and P-Max-Sum_AT on 3-color graph coloring with fixed constraint density $p = 0.05$ and varying number of agents $n = 100, \ldots, 1000$. Here, the computational overhead grows linearly with the number of agents because the constraint density is fixed, which results in the proportional growth of parameter t. This goes in contrast to the experiments in Figure 4, where the number of connected nodes per agent (m) was fixed, rather than the constraint density (p). Regardless of this growth, the computational overhead in 3-color graph coloring problems is clearly smaller than that in unstructured problems (see Figure 4), due to the small size of domains ($d = 3$) and the fast convergence ($K = 5$).

The realistic problems that we consider are meeting scheduling problems, in which 20 meetings are to be scheduled into 20 time slots. Each participant takes part in two randomly chosen meetings. For each pair of meetings, a travel time was chosen uniformly at random between 6 and 10, inclusive. When the difference between the time slots of two meetings is less than the travel time between those meetings, all participants in those meetings were marked as overbooked, and a cost equal to the number of overbooked participants is incurred. For representing the meeting scheduling problems we use the EAV (Events As Variables) formulation of Maheswaran et al. (2004b). In this formulation there is an agent per meeting that is aware of all constraints of participants in that meeting. We assume that each meeting is represented by a different agent. These realistic problems are identical to those used by Zivan et al. (2014). Both the constraint graph and the constraint functions in these problems are highly structured.

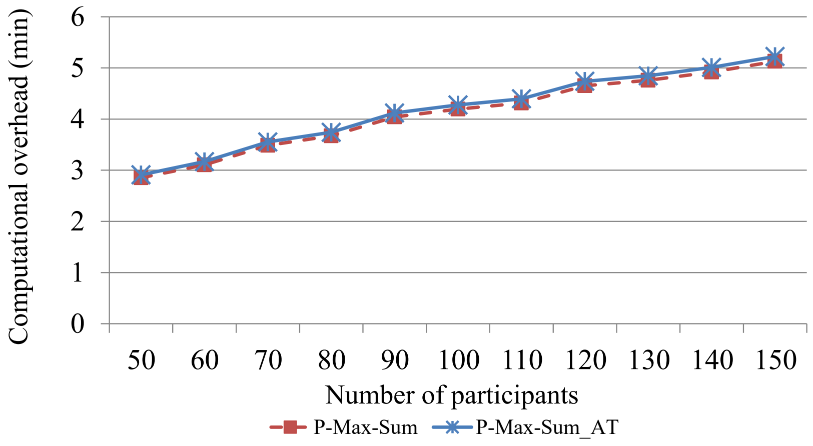

Figure 6: Computational overhead on realistic meeting scheduling problems.

Figure 6 displays the computational overhead of P-Max-Sum and P-Max-Sum_AT on meeting scheduling problems with a varying number of participants between 50 and 150. Here, the computational overhead is higher than that in the previous experiment due to the large size of domains ($d = 20$), but this is balanced by the relatively fast convergence of such problems ($K = 10$), as was previously shown by Zivan et al. (2014).

6.7 Overall Runtime Experiments on a Simulator

We implemented a full-scale version of the P-Max-Sum algorithm on the AgentZero simulator (Lutati, Gontmakher, Lando, Netzer, Meisels, & Grubshtein, 2014). Running experiments on a simulator brings with it several disadvantages. The first disadvantage is that a simulator runs on a single computer, as opposed to a real distributed system composed of several computers. Thus, the parallelism of different agents (threads in the simulator) is restricted to the number of cores in the computer's hardware. This is usually not a fundamental restriction for the evaluation of complete DCOP algorithms, since such algorithms are usually inherently restricted to small problems. This is also not a major restriction for the evaluation of incomplete DCOP algorithms, since usually the focus of evaluation in such algorithms is on solution quality rather than on runtime efficiency. Nevertheless, as already mentioned, even though P-Max-Sum is an incomplete algorithm, we are not interested in evaluating its solution quality, since it perfectly simulates the standard Max-Sum algorithm.

The second disadvantage of experiments on a simulator is that the results depend highly on overheads of the specific simulator (e.g., message passing, message handling, threading, idle detection), as well as on implementation issues of the specific algorithm, and the hardware on which the simulator is executed.

Having said that, experiments on a full-scale implementation can help one to confirm the applicability of an approach and to corroborate trends that were shown theoretically.

Consequently, we ran experiments on the AgentZero simulator that compare the runtime of P-Max-Sum to that of Max-Sum in varying constraint densities (p) and domain sizes (d). Given the aforementioned disadvantages of simulated experimentation, we ran the simulator on a dedicated Xeon 2.4GHz server with 24GB of memory and 24 cores, and focused on problems with the same number of agents ($n = 24$), to achieve maximal parallelism. In order to plot the results of both P-Max-Sum and Max-Sum on the same graphs, we display the runtime performance in a logarithmic scale. Each data point represents the average over 50 independently generated problems, and the overall runtime is measured in milliseconds.

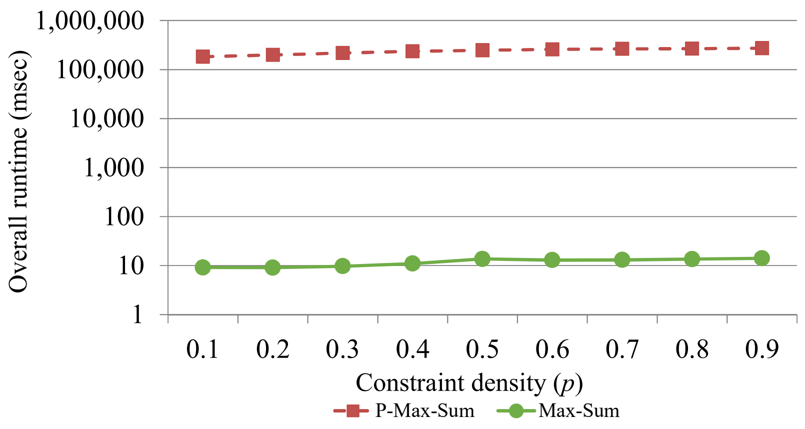

Figure 7: Overall runtime as a function of p.

In the first experiment we used a setting of unstructured random problems with $n = 24$ agents, domain size $d = 5$, and varying constraint densities, $p = 0.1, \ldots, 0.9$. We ran both algorithms for $K = 50$ iterations. Figure 7 shows that the runtime of the standard, non-privacy-preserving, Max-Sum algorithm is significantly smaller than that of P-Max-Sum. Indeed, as was shown in Section 6.6, the computational overhead of privacy preservation is substantial.

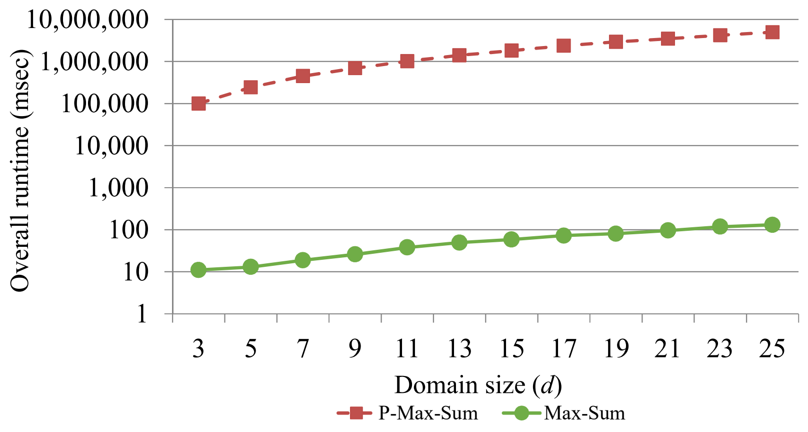

Figure 8: Overall runtime as a function of d.

In the second experiment we used a setting of unstructured random problems with $n = 24$ agents, constraint density $p = 0.2$, and varying domain sizes $d = 3, \dots, 25$. We ran both algorithms for $K = 50$ iterations. Figure 8 shows that the gap between the performances of the two algorithms grows when the domain sizes become larger. This corroborates the analysis of Protocol 2 (Section 6.2) that asserts a quadratic relation between the computational overhead of P-Max-Sum and the parameter d. The use of a logarithmic scale makes it hard to see that the relation in Figure 8 is indeed quadratic, so we relate herein to a few exemplary values taken from Figure 8; the runtime of P-Max-Sum with $d = 3$ is 100 seconds, with $d = 5$ it is 242 seconds, while with $d = 7$ it grows to 450 seconds – such a growth is clearly quadratic.

The simulator experiments of Figures 7 and 8 corroborate the trends shown in the respective Figures 2 and 3. As expected, the price of privacy preservation is quite high. Nevertheless, as is the case with non-privacy-preserving DCOP algorithms, the transition from complete methods to incomplete ones enables solving considerably larger problems. Indeed, unstructured random problems with $n = 24$ agents were solved with the incomplete P-Max-Sum algorithm, while state-of-the-art complete algorithms, such as P-DPOP$^{(+)}$ (Léauté & Faltings, 2013) and P-SyncBB (Grinshpoun & Tassa, 2016), are restricted to much smaller problems (Grinshpoun & Tassa, 2016, Figures 7 and 8).

7. Conclusion

One of the most important motivations for solving a problem distributively is preserving the privacy of agents. Therefore, a number of recent studies proposed private versions of existing DCOP-solving algorithms. Yet, no such study was based on the Max-Sum algorithm, which has recently been in the focus of both algorithmic and applicative DCOP research.

In this paper we proposed P-Max-Sum, a privacy-preserving version of Max-Sum. The proposed algorithm preserves topology, constraint and assignment/decision privacy. It may be enhanced to preserve also agent privacy by issuing a single call to a trusted coordinator. In addition, we designed a privacy-preserving implementation of Max-Sum with the anytime mechanism, as well as privacy-preserving implementations of three variants of the Max-Sum algorithm – Bounded Max-Sum (Rogers et al., 2011), Max-Sum_AD and Max-Sum_ADVP (Zivan & Peled, 2012). The computational and communication overhead of P-Max-Sum and its variants were theoretically analyzed.

We conducted experiments on a range of problems, from unstructured problems to structured and realistic ones. Our experiments show that the overhead of privacy preservation in terms of runtime is reasonable and highly scalable. For graph coloring problems with hundreds of agents the computational overhead was less than a minute, while in the other settings the overhead was between a couple of minutes and one hour, which is a reasonable price to pay in offline applications that require a high level of privacy.

The present work suggests several future research directions in order to enhance P-Max-Sum by removing some of the assumptions that underly our current protocols' design. One such assumption is that all constraints are binary. The extension to non-binary constraints is expected to be computationally challenging since already the complexity of the basic Max-Sum is exponential in the arity of the constraints. To mitigate that problem it would be needed to rely on methods such as Tractable HOP (Tarlow, Givoni, & Zemel, 2010; Pujol-Gonzalez, Cerquides, Meseguer, Rodríguez-Aguilar, & Tambe, 2013) or to represent constraints with high arity by multiple constraints with smaller arity (Smith, Stergiou, & Walsh, 2000). To the best of our knowledge, such scenarios have not been considered before from the privacy-preserving point of view.

Another assumption that we made here was that each agent controls a single variable. This simplifying assumption is widely accepted following two generic methods for dealing with multiple variables per agent that were proposed by Yokoo and Hirayama (2000). In the first method, all the local variables of an agent are compiled into a single *complex variable*; the domain of this complex variable is the Cartesian product of the domains of all local variables it was constructed from. Such an artificial coupling of distinct variables may dramatically reduce efficiency. In the second method, which is more commonly used, each variable is turned into a *virtual agent* that remains under the control of the original agent holding it. However, this method requires revisiting the protocols' design in order to accommodate for a setting in which an agent is a neighbor of itself in the constraint graph. A modification of our protocols for such settings is left as future work.

Lastly, we made herein the common assumption that agents do not collude. In order to deal with situations in which coalitions, up to some assumed size, are allowed, our protocols should be modified by further splitting of information. For example, we assumed that the decryption key in $\mathcal{E}_i$ is known to all agents in $\mathcal{A}_{-i}$ (namely, to all agents except for A_i); hence, if A_i colludes with another agent, it will be able to decrypt $\mathcal{E}_i$-encrypted messages. To prevent that, our usage of the Paillier cipher should be replaced with a corresponding threshold variant, such as the one proposed by Damgård and Jurik (2001).

Acknowledgments

The authors would like to thank Vadim Levit for his help with the implementation on the AgentZero simulator, and the anonymous referees for their thorough reviews that helped improve the paper.

Appendix A. Proofs

A.1 Proof of Theorem 2.1

Eq. (1) implies that

$$q^{k+1} \leq Dr^k \,, \tag{28}$$

while Eqs. (2)+(3) imply that

$$r^{k+1} \leq M_C + q^k \,. \tag{29}$$

We prove the claims of the theorem by induction on k. When $k = 0$ all messages are zero so that $q^0 = r^0 = 0$. As can be easily seen, those bounds do satisfy the inequalities in Eqs. (6)+(7).

We proceed by induction. Namely, we assume that q^k and r^k satisfy the inequalities in Eqs. (6)+(7) and prove that so do q^{k+1} and r^{k+1}. We distinguish between even and odd k.

If $k = 2t$ then, by the induction hypothesis,

$$q^k \leq M_C D \cdot \frac{D^t - 1}{D - 1}\,, \tag{30}$$

and

$$r^k \leq M_C \cdot \frac{D^t - 1}{D - 1}\,. \tag{31}$$

Hence, by Eqs. (28)+(31),

$$q^{k+1} \leq M_C D \cdot \frac{D^t - 1}{D - 1} = M_C D \cdot \frac{D^{\lfloor (k+1)/2 \rfloor} - 1}{D - 1},,$$

in accord with Eq. (6). In addition, by Eqs. (29)+(30),

$$r^{k+1} \leq M_C + M_C D \cdot \frac{D^t - 1}{D - 1} = M_C \cdot \left(1 + D \cdot \frac{D^t - 1}{D - 1}\right) = M_C \cdot \frac{D^{t+1} - 1}{D - 1} = M_C \cdot \frac{D^{\lceil (k+1)/2 \rceil} - 1}{D - 1}\,,$$

in accord with Eq. (7). The proof for odd values of k goes along the same lines. $\square$

A.2 Proof of Lemma 3.1

Since $0 \leq m_i \leq M \leq (\mu - 1)/2$ and $r \in \mathbb{Z}_\mu$, it follows that each n_i equals either $m_i + r$ or $m_i + r - \mu$. In particular, there exists an index $0 \leq k' \leq k$ such that

$$n_i = \begin{cases} m_i + r & 1 \leq i \leq k' \\ m_i + r - \mu & k' < i \leq k \end{cases}\,. \tag{32}$$

Let us distinguish between three cases:

- Case 1: $k' = k$. In that case $n_i = m_i + r$ for all $1 \leq i \leq k$.
- Case 2: $k' = 0$. In that case $n_i = m_i + r - \mu$ for all $1 \leq i \leq k$.
- Case 3: $0 < k' < k$. In that case $n_i = m_i + r$ for $1 \leq i \leq k'$ but $n_i = m_i + r - \mu$ for all $k' < i \leq k$.

In Cases 1 and 2, as the difference $n_i - m_i$ is the same for all i (where here we speak of the difference in the usual sense of integers, not in the modular sense), then it is clear that $\min n_i = n_1$ and $\max n_i = n_k$. Hence,

$$\max n_i - \min n_i = n_k - n_1 = m_k - m_r \leq M \leq (\mu - 1)/2$$

in both of those cases. Hence, the definition of $\min^* n_i$ in both of those cases yields, by the first case in Eq. (18), the value $\min^* n_i = n_1$. Now, $m_1 = n_1 - r$ in Case 1 and $m_1 = n_1 - r + \mu$ in Case 2. Therefore, in Case 1 we have $n_1 - r = m_1 \geq 0$ and then Eq. (19) is met through its first case. In Case 2 we have $n_1 - r = m_1 - \mu < 0$ and then q. (19) is met through its second case.

We now turn our attention to Case 3. In that case, $\min n_i = n_{k'+1}$ and $\max n_i = n_{k'}$, due to the occurrence of a wrap around after the k'th term in the sequence. Hence,

$$\max n_i - \min n_i = n_{k'} - n_{k'+1} = (m_{k'} + r) - (m_{k'+1} + r - \mu) = \mu - (m_{k'+1} - m_{k'})\,.$$

Since $m_{k'+1} - m_{k'} \leq M$ we infer that in Case 3

$$\max n_i - \min n_i \geq \mu - M \geq \mu - (\mu - 1)/2 > (\mu - 1)/2\,.$$

Moreover, it is clear that in Case 3 $n_i > (\mu - 1)/2$ for all $1 \leq i \leq k'$ while $n_i < (\mu - 1)/2$ for all $k' < i \leq k$. To prove the first inequality it suffices to show that $n_1 > (\mu - 1)/2$ while for the second inequality it suffices to show that $n_k < (\mu - 1)/2$. Assume, towards contradiction, that $n_1 = m_1 + r \leq (\mu - 1)/2$. Then $m_k + r = n_1 + (m_k - m_1) \leq (\mu - 1)/2 + M \leq \mu - 1$. But then we should have had $n_k = m_k + r$ and not $n_k = m_k + r - \mu$. That contradiction establishes our claim that $n_i > (\mu - 1)/2$ for all $1 \leq i \leq k'$. The proof of the second inequality is similar.

In view of the above, it follows that $\min^* n_i$, as defined in the second case of Eq. (18), equals $n_1 = m_1 + r$. Hence, $m_1 = n_1 - r$ in this case. Therefore, Eq. (19) is met through its first case. That concludes the proof. □

References

Aiello, W., Ishai, Y., & Reingold, O. (2001). Priced oblivious transfer: How to sell digital goods. In *EUROCRYPT*, pp. 119–135.

Aji, S. M., & McEliece, R. J. (2000). The generalized distributive law. *IEEE Transactions on Information Theory*, *46*(2), 325–343.

Alwen, J., Katz, J., Lindell, Y., Persiano, G., Shelat, A., & Visconti, I. (2009). Collusion-free multiparty computation in the mediated model. In *CRYPTO*, pp. 524–540.

Alwen, J., Shelat, A., & Visconti, I. (2008). Collusion-free protocols in the mediated model. In *CRYPTO*, pp. 497–514.

Aly, A., Cuvelier, E., Mawet, S., Pereira, O., & Vyve, M. V. (2013). Securely solving simple combinatorial graph problems. In *Financial Cryptography*, pp. 239–257.

Awerbuch, B., & Shiloach, Y. (1987). New connectivity and msf algorithms for shuffle-exchange network and pram. *IEEE Transactions on Computers*, *36*(10), 1258–1263.

Barabási, A.-L., & Albert, R. (1999). Emergence of scaling in random networks. *Science*, *286*(5439), 509–512.

Benaloh, J. C. (1994). Dense probabilistic encryption. In *Workshop on Selected Areas of Cryptography*, pp. 120–128.

Brickell, J., & Shmatikov, V. (2005). Privacy-preserving graph algorithms in the semi-honest model. In *ASIACRYPT*, pp. 236–252.

Damgård, I., & Jurik, M. (2001). A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *Public Key Cryptography*, pp. 119–136.

Doshi, P., Matsui, T., Silaghi, M. C., Yokoo, M., & Zanker, M. (2008). Distributed private constraint optimization. In *WI-IAT*, pp. 277–281.

Erdős, P., & Rényi, A. (1959). On random graphs. *Publicationes Mathematicae Debrecen*, *6*, 290–297.

Farinelli, A., Rogers, A., Petcu, A., & Jennings, N. R. (2008). Decentralised coordination of low-power embedded devices using the Max-Sum algorithm. In *AAMAS*, pp. 639–646.

Gershman, A., Meisels, A., & Zivan, R. (2009). Asynchronous forward bounding. *Journal of Artificial Intelligence Research*, *34*, 25–46.

Goldreich, O. (2001). *Foundations of Cryptography: Basic Tools.* Cambridge University Press.

Greenstadt, R., Grosz, B., & Smith, M. D. (2007). SSDPOP: improving the privacy of DCOP with secret sharing. In *AAMAS*, pp. 1098–1100.

Greenstadt, R., Pearce, J., & Tambe, M. (2006). Analysis of privacy loss in distributed constraint optimization. In *AAAI*, pp. 647–653.

Grinshpoun, T. (2012). When you say (DCOP) privacy, what do you mean?. In *ICAART*, pp. 380–386.

Grinshpoun, T., & Tassa, T. (2014). A privacy-preserving algorithm for distributed constraint optimization. In *AAMAS*, pp. 909–916.

Grinshpoun, T., Grubshtein, A., Zivan, R., Netzer, A., & Meisels, A. (2013). Asymmetric distributed constraint optimization problems. *Journal of Artificial Intelligence Research*, *47*, 613–647.

Grinshpoun, T., & Tassa, T. (2016). P-SyncBB: A privacy preserving branch and bound DCOP algorithm. *Journal of Artificial Intelligence Research*, *57*, 621–660.

Hirayama, K., & Yokoo, M. (1997). Distributed partial constraint satisfaction problem. In *CP*, pp. 222–236.

Keller, M., & Scholl, P. (2014). Efficient, oblivious data structures for MPC. In *ASIACRYPT*, pp. 506–525.

Laud, P. (2015). Parallel oblivious array access for secure multiparty computation and privacy-preserving minimum spanning trees. *Proceedings on Privacy Enhancing Technologies*, *2015*(2), 188–205.

Laur, S., & Lipmaa, H. (2007). A new protocol for conditional disclosure of secrets and its applications. In *ACNS*, pp. 207–225.

Léauté, T., & Faltings, B. (2013). Protecting privacy through distributed computation in multi-agent decision making. *Journal of Artificial Intelligence Research*, *47*, 649–695.

Lutati, B., Gontmakher, I., Lando, M., Netzer, A., Meisels, A., & Grubshtein, A. (2014). Agentzero: A framework for simulating and evaluating multi-agent algorithms. In *Agent-Oriented Software Engineering - Reflections on Architectures, Methodologies, Languages, and Frameworks*, pp. 309–327.

Maheswaran, R. T., Pearce, J. P., Bowring, E., Varakantham, P., & Tambe, M. (2006). Privacy loss in distributed constraint reasoning: A quantitative framework for analysis and its applications. *Autonomous Agents and Multi-Agent Systems*, *13*, 27–60.

Maheswaran, R. T., Pearce, J. P., & Tambe, M. (2004a). Distributed algorithms for DCOP: A graphical-game-based approach. In *ISCA PDCS*, pp. 432–439.

Maheswaran, R. T., Tambe, M., Bowring, E., Pearce, J. P., & Varakantham, P. (2004b). Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *AAMAS*, pp. 310–317.

Modi, P. J., Shen, W., Tambe, M., & Yokoo, M. (2005). Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, *161*(1-2), 149–180.

Naor, M., & Pinkas, B. (2001). Efficient oblivious transfer protocols. In *SODA*, pp. 448–457.

Naor, M., & Pinkas, B. (2005). Computationally secure oblivious transfer. *Journal of Cryptology*, *18*(1), 1–35.

Nissim, K., & Zivan, R. (2005). Secure DisCSP protocols - from centralized towards distributed solutions. In *IJCAI, DCR Workshops*.

Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pp. 223–238.

Petcu, A., & Faltings, B. (2005a). Approximations in distributed optimization. In *CP*, pp. 802–806.

Petcu, A., & Faltings, B. (2005b). A scalable method for multiagent constraint optimization. In *IJCAI*, pp. 266–271.

Pujol-Gonzalez, M., Cerquides, J., Meseguer, P., Rodríguez-Aguilar, J. A., & Tambe, M. (2013). Engineering the decentralized coordination of uavs with limited communication range. In *Advances in Artificial Intelligence (CAEPIA)*, pp. 199–208.

Rabin, M. (1981). How to exchange secrets by oblivious transfer. Tech. rep. TR-81, Aiken Computation Laboratory.

Ramchurn, S. D., Farinelli, A., Macarthur, K. S., & Jennings, N. R. (2010). Decentralized coordination in robocup rescue. *The Computer Journal*, *53*(9), 1447–1461.

Rogers, A., Farinelli, A., Stranders, R., & Jennings, N. R. (2011). Bounded approximate decentralised coordination via the Max-Sum algorithm. *Artificial Intelligence*, *175*(2), 730–759.

Rollon, E., & Larrosa, J. (2012). Improved bounded max-sum for distributed constraint optimization. In *CP*, pp. 624–632.

Silaghi, M. C., Faltings, B., & Petcu, A. (2006). Secure combinatorial optimization simulating DFS tree-based variable elimination. In *ISAIM*.

Silaghi, M. C., & Mitra, D. (2004). Distributed constraint satisfaction and optimization with privacy enforcement. In *IAT*, pp. 531–535.

Smith, B. M., Stergiou, K., & Walsh, T. (2000). Using auxiliary variables and implied constraints to model non-binary problems. In *AAAI/IAAI*, pp. 182–187.

Stranders, R., Farinelli, A., Rogers, A., & Jennings, N. R. (2009). Decentralised coordination of continuously valued control parametersusing the Max-Sum algorithm. In *AAMAS*, pp. 601–608.

Sultanik, E., Lass, R. N., & Regli, W. C. (2008). DCOPolis: a framework for simulating and deploying distributed constraint reasoning algorithms. In *AAMAS (demos)*, pp. 1667–1668.

Tarlow, D., Givoni, I. E., & Zemel, R. S. (2010). HOP-MAP: efficient message passing with high order potentials. In *AISTATS*, pp. 812–819.

Tassa, T., Zivan, R., & Grinshpoun, T. (2015). Max-sum goes private. In *IJCAI*, pp. 425–431.

Teacy, W. T. L., Farinelli, A., Grabham, N. J., Padhy, P., Rogers, A., & Jennings, N. R. (2008). Max-Sum decentralised coordination for sensor systems. In *AAMAS*, pp. 1697–1698.

Yao, A. C. (1982). Protocols for secure computation. In *FOCS*, pp. 160–164.

Yokoo, M., & Hirayama, K. (2000). Algorithms for distributed constraint satisfaction problems: A review. *Autonomous Agents and Multi-Agent Systems*, *3*(2), 185–207.

Yokoo, M., Suzuki, K., & Hirayama, K. (2005). Secure distributed constraint satisfaction: reaching agreement without revealing private information. *Artificial Intelligence*, *161*(1-2), 229–245.

Zhang, W., Wang, G., Xing, Z., & Wittenburg, L. (2005). Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, *161*(1-2), 55–87.

Zivan, R., Okamoto, S., Parash, T., Cohen, L., & Peled, H. (2017). Balancing exploration and exploitation in incomplete min/max-sum inference for distributed constraint optimization. *Autonomous Agents and Multi-Agent Systems*, *accepted for publication*.

Zivan, R., Okamoto, S., & Peled, H. (2014). Explorative anytime local search for distributed constraint optimization. *Artificial Intelligence*, *212*, 1–26.

Zivan, R., & Peled, H. (2012). Max/min-sum distributed constraint optimization through value propagation on an alternating DAG. In *AAMAS*, pp. 265–272.

Zivan, R., Parash, T., & Naveh, Y. (2015). Applying max-sum to asymmetric distributed constraint optimization. In *IJCAI*, pp. 432–439.