Max-Sum Goes Private

Tamir Tassa
The Open University
Ra'anana, Israel
tamirta@openu.ac.il

Roie Zivan
Ben-Gurion University of the Negev
Beer-Sheva, Israel
zivanr@bgu.ac.il

Tal Grinshpoun
Ariel University
Ariel, Israel
talgr@ariel.ac.il

Abstract

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. Our experiments on structured and realistic problems show that the overhead of privacy preservation in terms of runtime is reasonable.

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 *et al.*, 2005; Petcu and Faltings, 2005b; Gershman *et al.*, 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 *et al.*, 2004; Zhang *et al.*, 2005; Zivan *et al.*, 2014; Teacy *et al.*, 2008].

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 *et al.*, 2010]. Agents in MAX-SUM propagate cost/utility information to all neighbors. This contrasts with other inference algorithms such as ADPOP [Petcu and Faltings, 2005a], in which agents only propagate costs up a pseudo-tree structure.

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 *et al.*, 2007; Grinshpoun, 2012; Léauté and Faltings, 2013]. In this paper we relate to the categorization of Léauté and Faltings [2013] that distinguish 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 *et al.* [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 *et al.*, 2006]. Doshi *et al.* [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. Several privacy-preserving versions of DPOP were proposed in the past [Greenstadt *et al.*, 2007; Silaghi *et al.*, 2006] including a recent study by Léauté and Faltings [2013] that proposed several versions of DPOP that provide strong privacy guarantees. Another recent paper [Grinshpoun and Tassa, 2014] devises a variation of SyncBB that preserves constraint 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 agents from revealing the identity of other agents with which their neighbors are constrained (topology privacy), the costs their neighbors assign to value assignments (constraint privacy), or the assignment selection of their neighbors (assignment/decision privacy). By allowing a single call to a trusted coordinator, P-MAX-SUM can also preserve agent privacy.

2 Preliminaries

A Distributed Constraint Optimization Problem (DCOP) [Hirayama and 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 $C : D_{i_1} \times \cdots \times D_{i_k} \rightarrow \mathbb{R}_+ := [0, \infty)$, for some $1 \leq i_1 < \cdots < i_k \leq m$.

A *value assignment* is a pair including a variable and a value from that variable's domain. A *complete assignment* consists of value assignments to all variables in $\mathcal{X}$. 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 $C_{i,j} : D_i \times D_j \rightarrow \mathbb{R}_+$. These assumptions are customary in DCOP literature (e.g., [Modi *et al.*, 2005; Petcu and Faltings, 2005b]).

Each DCOP induces a 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.

Knowledge Assumptions. We make the following commonly used assumptions (see, e.g. [Léauté and Faltings, 2013]): A variable and its domain are known only to its owner agent and to the agents owning neighboring variables. In addition, a constraint is fully known to all agents owning variables in its scope, and no other agent knows anything about the constraint (not even its existence).

We describe a variant of our protocol that maintains also agent privacy. For this variant we make also the additional assumption that each agent knows all agents that own a variable that is a neighbor of their own variable in the constraint graph, but does not know any of the other agents, not even their existence.

Communication Assumptions. Here too we make the standard communication assumptions (see, e.g. [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.

2.1 The Max-Sum algorithm

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 synchronous step. 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 step all messages are zero. After completing the kth step, the messages in the next step will be as follows. Fixing a variable node X_i and letting N_i be its set of neighbors, then for each $X_e \in N_i$, X_i will send to X_e the vector

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

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

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

while for each $y \in D_j$,

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

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

In order to prevent the entries in the messages from growing uncontrollably, it is customary to reduce from each entry in each message $Q^{k+1}_{i \rightarrow e}$, where $e = (X_i, X_j)$, the value $\alpha^{k+1}_{i,j} := (\sum_{x \in D_i} Q^{k+1}_{i \rightarrow e}(x))/|D_i|$ or $\alpha^{k+1}_{i,j} := \min_{x \in D_i} Q^{k+1}_{i \rightarrow e}(x)$. (The latter option ensures that all message entries always remain nonnegative.)

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 $R^k_{e \rightarrow i}$ 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 $R^k_{e \rightarrow i}$ 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 $Q^k_{i \rightarrow e}$ or $R^k_{e \rightarrow i}$ (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 p be a very large integer and let $\mathbb{Z}_p$ be the additive group modulo p. Then each of the entries in any of the messages $Q^k_{i \rightarrow e}$ or $R^k_{e \rightarrow i}$ will be treated by our secure protocols as an element in $\mathbb{Z}_p$. As will be clarified later, p will be of the size of typical moduli of public-key cryptosystems. (Usually, modern public-key cryptography uses moduli of size 1024

bits.) Let c denote an upper bound on the entries of all messages $Q^k_{i\to e}$ and $R^k_{e\to i}$. Applying the procedure described at the end of Section 2.1 usually prevents an overflow from a standard double-precision arithmetic, namely, it guarantees that all entries remain bounded by $c = 2^{64}$. However, as we work in very large arithmetic size (modulo p), all we need to assume is that all entries in all messages remain bounded by some constant $c \ll p$. (So, if $\log_2 p \approx 1024$, we may assume that, say, $\log_2 c \approx 950$.)

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 step 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 $\mathbb{Z}_p$, denoted s_i and s_j, such that $s_i + s_j = w$ (where all additions hereinafter are modulo p, 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 synchronous step).

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 synchronous step. 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:

$$Q^k_{i\to e} = S^{k,i}_{i\to e} + S^{k,j}_{i\to e}\,; \tag{4}$$

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.,

$$R^k_{e\to i} = S^{k,i}_{e\to i} + S^{k,j}_{e\to i}\,. \tag{5}$$

(Recall that the messages and the shares in Eqs. (4) and (5) are vectors in $\mathbb{Z}_p^{|D_i|}$.)

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 step $k = 0$. Then, in the main body of our work, we describe the computation of shares in the progression between successive steps, 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 step.

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. (4), A_i and A_j create jointly a random vector $S^{0,i}_{i\to e} \in \mathbb{Z}_p^{|D_i|}$, 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. (5).

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.

Using homomorphic encryption

An additive homomorphic encryption is an 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 agent has a public key additive homomorphic cryptosystem. The encryption of messages under the cryptosystem of agent A_i will be denoted $\mathcal{E}_i$, $1 \le i \le n$. It will be used by agents A_j that are constrained with A_i in order to send encrypted messages to A_i; the private decryption key of $\mathcal{E}_i$ will be known only to A_i. In addition, we assume that every set of agents of the form $\mathcal{A}_{-i} := \{A_j : 1 \le j \le n, j \ne i\}$ has an additive homomorphic cryptosystem, in which the encryption function is denoted $\mathcal{F}_i$. The private key in $\mathcal{F}_i$ is known to all agents in $\mathcal{A}_{-i}$ (namely, to all agents except for A_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 $m(\mathcal{E}_i)$ and $m(\mathcal{F}_i)$ denote the moduli of $\mathcal{E}_i$ and $\mathcal{F}_i$ respectively, $1 \le i \le n$, then a good selection of p (the size of the domain $\mathbb{Z}_p$ in which all shares take value) would be $p = \min\{m(\mathcal{E}_i), m(\mathcal{F}_i) : 1 \le i \le n\}$.

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 $1 \le j_1 < j_2 < \cdots < j_t \le n$ be the indices of all variables that are constrained with X_i so that $N_i = \{X_{e_\ell} : 1 \le \ell \le t\}$, 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

$$Q^{k+1}_{i\to e_1} = \sum_{\ell=2}^{t} R^k_{e_\ell\to i}\,. \tag{6}$$

We start by dealing with the case $t \ge 2$. In that case, the sum on the right-hand side of Eq. (6) is non-empty. Each of the vectors in the sum on the right-hand side of Eq. (6) is shared between A_i and A_{j_ℓ}, $2 \le \ell \le t$. 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. (4)) can be computed as follows:

- The share that A_i will get, denoted $S_{i\to e_1}^{k+1,i}$, is the sum of the $t-1$ shares that A_i has for the $t-1$ messages $R_{e_\ell\to i}^k$, $2 \le \ell \le t$. A_i can compute it on its own.
- The share that A_{j_1} will get, $S_{i\to e_1}^{k+1,j_1}$, is the sum of the $t-1$ shares that $A_{j_2}, \ldots, A_{j_t}$ have in step k for the $t-1$ messages $R_{e_\ell\to i}^k$, $2 \le \ell \le t$. 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 $S_{e_\ell\to i}^{k,i}$ and $S_{e_\ell\to i}^{k,j_\ell}$ be the shares that A_i and A_{j_ℓ} hold, respectively, in $R_{e_\ell\to i}^k$, $1 \le \ell \le t$. Those shares will be the inputs that A_i and its neighbors A_{j_ℓ}, $1 \le \ell \le t$, bring to Protocol 1. The output to A_i will be the shares $S_{i\to e_\ell}^{k+1,i}$ for each $1 \le \ell \le t$. The output to the neighboring agent A_{j_ℓ}, $1 \le \ell \le t$, will be $S_{i\to e_\ell}^{k+1,j_\ell}$, which is the complement share in $Q_{i\to e_\ell}^{k+1}$.

In Steps 1-3 of Protocol 1, all neighbors send to A_i their shares, encrypted with $\mathcal{F}_i$, to prevent A_i from recovering them. (The encryption $\mathcal{F}_i(\cdot)$ is applied independently on each of the $|D_i|$ components of the share $S_{e_\ell\to i}^{k,j_\ell}$.) The subsequent loop in Steps 4-8 describes the interaction of A_i vis-a-vis each of the neighboring agents A_{j_ℓ}, $1 \le \ell \le t$. In Step 5, A_i computes its share in $Q_{i\to e_\ell}^{k+1}$. In Step 6, A_i sends to A_{j_ℓ} a message W_ℓ where, owing to the additive homomorphic property of $\mathcal{F}_i$, $W_\ell = \mathcal{F}_i(\sum_{1\le\ell'\ne\ell\le t} S_{e_{\ell'}\to i}^{k,j_{\ell'}})$. Hence, A_{j_ℓ} recovers in Step 7 its share $S_{i\to e_\ell}^{k+1,j_\ell} = \sum_{1\le\ell'\ne\ell\le t} S_{e_{\ell'}\to i}^{k,j_{\ell'}}$ as required.

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

1: **for** $\ell = 1, \ldots, t$ **do**
2: A_{j_ℓ} sends to A_i the encryption of its share $\mathcal{F}_i(S_{e_\ell\to i}^{k,j_\ell})$.
3: **end for**
4: **for** $\ell = 1, \ldots, t$ **do**
5: A_i computes $S_{i\to e_\ell}^{k+1,i} \leftarrow \sum_{1\le\ell'\ne\ell\le t} S_{e_{\ell'}\to i}^{k,i}$.
6: A_i computes $W_\ell := \prod_{1\le\ell'\ne\ell\le t} \mathcal{F}_i(S_{e_{\ell'}\to i}^{k,j_{\ell'}})$ and sends it to A_{j_ℓ}.
7: A_{j_ℓ} sets $S_{i\to e_\ell}^{k+1,j_\ell} \leftarrow \mathcal{F}_i^{-1}(W_\ell)$.
8: **end for**

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 $S_{i\to e_\ell}^{k+1,j_\ell}$ 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 6 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 $t = 1$. In such a case $Q_{i\to e_1}^{k+1}$ is zero. Here, Steps 1-3 are redundant (but are still carried out in order to hide from A_{j_1} the fact that it is the single neighbor of A_i). Then, instead of Step 5 in Protocol 1, A_i will generate a new random share for itself, denoted $S_{i\to e_1}^{k+1,i}$. Then, in Step 6, A_i will set $W_1 := \mathcal{F}_i(-S_{i\to e_1}^{k+1,i})$. Hence, A_{j_1} will recover in Step 7 the share $S_{i\to e_1}^{k+1,j_1} = -S_{i\to e_1}^{k+1,i}$ as required, obliviously of the fact that $t = 1$.

In view of the discussion at the end of Section 2.1, Protocol 1 must be augmented by a post-processing procedure in which every pair of neighbors A_i and A_j compute $\alpha_{i,j}^{k+1}$ and then A_j decreases $\alpha_{i,j}^{k+1}$ from its share $S_{i\to e}^{k+1,j}$. We omit the details of that procedure due to lack of space.

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 $R_{e\to i}^{k+1}(x)$ 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 $Q_{j\to e}^k(y)$, 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

$$s^{k+1,i}(x)+s^{k+1,j}(x) = \min_{y\in D_j} \left[C_{i,j}(x,y) + s^{k,i}(y) + s^{k,j}(y)\right] \tag{7}$$

where all additions are modulo p, but when computing the minimum in Eq. (7), the numbers in the brackets are viewed as nonnegative integers.

Such computations can be carried out with perfect privacy using Yao's garbled circuit protocol [Yao, 1982]. However, implementing Yao's protocol for each entry in each transmitted message emerging from a function node in each of the algorithm's steps 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.

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

1: A_i sends to A_j the value $\mathcal{E}_i(s^{k,i}(y))$ for all $y \in D_j$.
2: A_j selects uniformly at random a scalar integer $r \in [0, p-c-1]$.
3: A_j computes $W(y) := \mathcal{E}_i(s^{k,i}(y))\cdot\mathcal{E}_i(C_{i,j}(x,y)+s^{k,j}(y)+r)$ for all $y \in D_j$.
4: A_j sends a permutation of $\{W(y) : y \in D_j\}$ to A_i.
5: A_i computes $\mathcal{E}_i^{-1}(W(y)) = C_{i,j}(x,y)+s^{k,i}(y)+s^{k,j}(y)+r$ for all $y \in D_j$.
6: A_i computes $w := \min_{y\in D_j} \mathcal{E}_i^{-1}(W(y))$.
7: A_i generates a random share for itself $s^{k+1,i}(x) \in \mathbb{Z}_p$.
8: A_i sends to A_j the value $w' = w - s^{k+1,i}(x)$.
9: A_j computes $s^{k+1,j}(x) = w' - r$.

In Step 1 of Protocol 2, A_i sends to A_j an encryption of its share $s^{k,i}(y)$ for all $y \in D_j$. It uses its own encryption function $\mathcal{E}_i$ so that A_j will not be able to recover A_i's private shares. However, as $\mathcal{E}_i$ is homomorphic, A_j can perform the needed arithmetics on the received shares.

In Step 2, A_j selects at random a masking scalar $r \in [0, p-c-1]$ which will be used to protect private information from A_i, as we shall see shortly. (Recall that c is a publicly known upper bound on all entries in all messages.) 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

$$w(y) := C_{i,j}(x,y) + s^{k,i}(y) + s^{k,j}(y) + r\,, \qquad (8)$$

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. (8) equals the corresponding argument in the minimum on the right-hand side of Eq. (2),

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

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

Let us now examine $w(y)$. As discussed above, it equals $m(y) + r$. Since $m(y)$ is an integer in the range $[0, c]$ (by our assumption) and as $r \in [0, p-c-1]$, then $w(y)$ equals $m(y) + r$, where the sum is in the usual sense of a sum of integers. Namely, even though all sums in our discussion are sums modulo p, in this particular case, the equality holds in the stronger sense of sum of integers. Hence, $\min_{y\in D_j}[m(y) + r]$, which is the value w that A_i computes in Step 6, equals $r + \min_{y\in D_j} m(y)$ (where, again, the latter sum is a regular sum of integers, and not just a modular sum). Next, A_i generates for itself a random share $s^{k+1,i}(x) \in \mathbb{Z}_p$ (Step 7) and sends to A_j the value $w' = w - s^{k+1,i}(x)$ (Step 8), where the subtraction is modulo p. Finally, A_j computes its share in Step 9, $s^{k+1,j}(x) = w' - r$, where once again the subtraction is modulo p. As a result, A_i and A_j end up with random shares, $s^{k+1,i}(x)$ and $s^{k+1,j}(x)$ respectively, whose sum modulo p equals $\min_{y\in D_j} m(y)$, as required.

We now turn to discuss the privacy provided by Protocol 2. The only information that A_j receives in the course of the protocol is the encryption of A_i's shares by $\mathcal{E}_i$. Assuming that $\mathcal{E}_i$ is based on a strong key, then Protocol 2 provides computational privacy against A_j. As for A_i, it gets all values $m(y) + r$, $y \in D_j$. The usage of the random shift r ensures that in very high probability, $1 - \frac{2c}{p-c}$, A_i can learn absolutely no information on $m(y)$, while in probability $\frac{2c}{p-c}$ it may learn either a lower or an upper bound on $m(y)$ for some $y \in D_j$. (We omit further discussion here; the interested reader is referred to [Tassa and Bonchi, 2014, Theorem 4.1].) In addition, the usage of the secret permutation, which A_j selects, prevents A_i from associating any value of $m(y)+r$ to any particular $y \in D_j$. However, A_i may sort the values $m(y)+r$ in a non-decreasing order and find the correct differences between consecutive entries in the corresponding ordering of $m(y)$. In addition, it may compute any function of those differences, such as the variance of the sequence $m(y)$. We consider this leakage of information completely benign, and certainly a modest price to pay for a fully practical secure implementation of this part of the P-MAX-SUM algorithm.

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 $N_i = \{X_{e_\ell} : 1 \le \ell \le t\}$, where $e_\ell = (X_i, X_{j_\ell})$). Then, A_i needs to assign X_i the value x_i 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 1-3, all agents that are connected to A_i send to it an $\mathcal{F}_i$-encryption of their share in the last message sent from their respective function node to X_i. Then, in Step 4, A_i selects a random masking scalar $r \in [0, p-c-1]$, defines $S_r = (r, \ldots, r) \in \mathbb{Z}_p^{|D_i|}$, and computes

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

Owing to the homomorphic property of $\mathcal{F}_i$, the vector $\hat{M}$ equals $\mathcal{F}_i(S_r + \sum_{\ell=1}^t S^{K,j_\ell}_{e_\ell\to i} + \sum_{\ell=1}^t S^{K,i}_{e_\ell\to i}) = \mathcal{F}_i(S_r + M_i)$, where M_i is as defined above. In Steps 5-6, A_i sends a secret and random permutation of the entries of $\hat{M}$ to one of its neighbors who proceeds to decrypt it and notify A_i of the index h in which a minimum was obtained. Finally (Step 7), A_i assigns to X_i the value $x \in D_i$ in which $S_r + M_i$ (and consequently also M_i) was minimal.

We omit a discussion of the privacy of Protocol 3 due to space limitations.

Protocol 3 Computing the best assignment for X_i

1: **for** $\ell = 1, \ldots, t$ **do**
2: $\quad A_{j_\ell}$ sends to A_i the encryption of its share $\mathcal{F}_i(S^{K,j_\ell}_{e_\ell\to i})$.
3: **end for**
4: A_i selects $S_r = (r, \ldots, r) \in \mathbb{Z}_p^{|D_i|}$, where $r \in [0, p-c-1]$, and computes $\hat{M} = \mathcal{F}_i(S_r) \cdot \prod_{\ell=1}^t \mathcal{F}_i(S^{K,j_\ell}_{e_\ell\to i}) \cdot \mathcal{F}_i(\sum_{\ell=1}^t S^{K,i}_{e_\ell\to i})$.
5: A_i selects a secret random permutation π on D_i and sends $\pi(\hat{M})$ to A_{j_1}.
6: A_{j_1} decrypts the entries of the received vector and informs A_i of the index h of the minimal entry.
7: A_i assigns to X_i the value that was mapped by π to h.

4 Privacy analysis (overview)

Our proposed algorithm preserves three types of privacy: topology privacy (by using the homomorphic encryption functions $\mathcal{F}_i$), constraint privacy, and assignment/decision privacy (by using the homomorphic encryption functions $\mathcal{E}_i$, secret sharing, random maskings and random permutations). In the version presented in the paper, the algorithm does not

preserve agent privacy due to the need for generating the private keys in the encryption functions $\mathcal{F}_i$. 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 key pairs for $\mathcal{F}_i$ and send them to all of A_i's neighbors, for all $1 \leq i \leq n$.

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 and Tassa, 2014] preserves constraint and topology privacy, but not agent or decision 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 Efficiency analysis

We analyze here the computational cost of P-MAX-SUM. We focus on analyzing the induced cost in the progression steps, since both initialization and termination steps are performed only once. In addition, we count only encryption and decryption operations, since the other performed operations (random numbers' generation, additions, multiplications, and computing minima) have computational costs which are few orders of magnitude smaller than those of the cryptographic operations. Let us fix an agent A_i and assume that it has t neighboring agents. Let us also denote the costs of encryption and decryption by C_e and C_d, respectively. Finally, let $\hat{d} := \max_{1 \leq j \leq n} |D_j|$ denote the size of the largest variable domain. Then, Protocol 1 involves t concurrent encryptions (Steps 1-3) followed by t non-concurrent decryptions (Step 7). Hence, the non-concurrent runtime of Protocol 1 for A_i in any given synchronous step is $C_e + tC_d$. As for Protocol 2, A_i has to run it vis-a-vis each of its neighbors. In Step 1 it performs up to $\hat{d}$ encryptions vis-a-vis each of its t neighbors. Hence, the non-concurrent runtime of that step is bounded by $t\hat{d}C_e$. Then, in Step 3 all neighboring agents have to perform up to $\hat{d}$ encryptions, concurrently. Finally, Step 5 entails up to $t\hat{d}$ non-concurrent decryptions. Hence, the overall non-concurrent runtime for A_i due to Protocol 2 in any given step is bounded by $(t+1)\hat{d}C_e + t\hat{d}C_d$. Letting $\hat{t}$ denote the maximal degree of a node in the graph G, we conclude that the overall runtime of the P-MAX-SUM algorithm is (roughly) bounded by $\Big(C_e \cdot [(\hat{t}+1)\hat{d}+1] + C_d \cdot \hat{t}(\hat{d}+1)\Big) \cdot K$, where K is the number of synchronous steps.

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

To realize the actual time it will take P-MAX-SUM to run we followed the *simulated time* approach [Sultanik *et al.*, 2008] by measuring the time of atomic operations performed in the algorithm and then counting the non-concurrent times these operations are performed. We measured the runtimes of the encryption and decryption operations by averaging multiple runs of the common Java implementation of the Paillier cryptosystem[1] on a hardware comprised of an Intel i7-4600U processor and 16GB memory. Our tests show that C_e takes at most 2 msec, while C_d takes at most 3 msec.

There are a number of indications that MAX-SUM performs best in the first few iterations of its run [Zivan and Peled, 2012; Zivan *et al.*, 2014]. The experiments presented by Zivan *et al.* [2014] indicate this phenomenon on various benchmarks. We reproduced these experiments to investigate the number of iterations required for MAX-SUM to find a solution of high quality on realistic and structured benchmarks, where privacy is expected to be important, as well as on uniform random problems.

The results were conclusive. The best solutions for relatively large graph coloring problems (including 100 agents, three colors in each domain and constraint density of 0.05) and meeting scheduling problems (including 90 agents, 20 meetings among them)[2] were found within the first 5 and 10 iterations, respectively. Thus, all iterations following the tenth iteration in each run were redundant. For uniform random problems, the average result was also the best after a small number of iterations. However, occasionally, the exploration in further iterations revealed higher quality solutions. That been said, the fact that MAX-SUM is not guaranteed to converge indicates the need for the use of the *anytime framework* as proposed by Zivan *et al.* [2014]. The privacy preserving methods presented in this paper can be used to generate a private version of the anytime framework. A comprehensive discussion of those issues is delayed to the full version of this paper due to space limitations.

Considering the limited number of needed iterations along with the overhead of its cryptographic operations, P-MAX-SUM needs at most a few seconds to complete the run of a large-scale structured or realistic problem. For instance, a graph coloring problem with 100 agents takes about 0.5 seconds, while a common meeting scheduling problem takes about 5 seconds. Such running times are clearly acceptable considering the privacy preservation requirement.

A detailed efficiency analysis (computational and communication costs), comparing P-MAX-SUM to the original non-private MAX-SUM algorithm, is omitted due to space limitations.

6 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

[1] `http://www.csee.umbc.edu/~kunliu1/research/Paillier.html`

[2] For more details on the setup see [Zivan *et al.*, 2014].

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 future work we shall extend P-MAX-SUM so that it handles non-binary constraints. We also plan to devise similar privacy-preserving versions to extensions of MAX-SUM, e.g., BOUNDED-MAX-SUM [Rogers *et al.*, 2011] and MAX-SUM_ADVP [Zivan and Peled, 2012].

References

[Benaloh, 1994] J. C. Benaloh. Dense probabilistic encryption. In *Workshop on Selected Areas of Cryptography*, pages 120–128, 1994.

[Doshi *et al.*, 2008] P. Doshi, T. Matsui, M. C. Silaghi, M. Yokoo, and M. Zanker. Distributed private constraint optimization. In *WI-IAT*, pages 277–281, 2008.

[Farinelli *et al.*, 2008] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the Max-Sum algorithm. In *AAMAS*, pages 639–646, 2008.

[Gershman *et al.*, 2009] A. Gershman, A. Meisels, and R. Zivan. Asynchronous forward bounding. *Journal of Artificial Intelligence Research*, 34:25–46, 2009.

[Greenstadt *et al.*, 2006] R. Greenstadt, J. Pearce, and M. Tambe. Analysis of privacy loss in distributed constraint optimization. In *AAAI*, pages 647–653, 2006.

[Greenstadt *et al.*, 2007] R. Greenstadt, B. Grosz, and M. D. Smith. SSDPOP: improving the privacy of DCOP with secret sharing. In *AAMAS*, pages 171:1–171:3, 2007.

[Grinshpoun and Tassa, 2014] T. Grinshpoun and T. Tassa. A privacy-preserving algorithm for distributed constraint optimization. In *AAMAS*, pages 909–916, 2014.

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

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

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

[Maheswaran *et al.*, 2004] R. T. Maheswaran, J. P. Pearce, and M. Tambe. Distributed algorithms for DCOP: A graphical-game-based approach. In *PDCS*, pages 432–439, 2004.

[Maheswaran *et al.*, 2006] R. T. Maheswaran, J. P. Pearce, E. Bowring, P. Varakantham, and M. Tambe. Privacy loss in distributed constraint reasoning: A quantitative framework for analysis and its applications. *JAAMAS*, 13:27–60, 2006.

[Modi *et al.*, 2005] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. ADOPT: asynchronous distributed constraints optimizationwith quality guarantees. *Artificial Intelligence*, 161:149–180, 2005.

[Paillier, 1999] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt*, pages 223–238, 1999.

[Petcu and Faltings, 2005a] A. Petcu and B. Faltings. Approximations in distributed optimization. In *CP*, pages 802–806, 2005.

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

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

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

[Silaghi *et al.*, 2006] M. C. Silaghi, B. Faltings, and A. Petcu. Secure combinatorial optimization simulating DFS tree-based variable elimination. In *ISAIM*, 2006.

[Stranders *et al.*, 2009] R. Stranders, A. Farinelli, A. Rogers, and N. R. Jennings. Decentralised coordination of continuously valued control parametersusing the Max-Sum algorithm. In *AAMAS*, pages 601–608, 2009.

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

[Tassa and Bonchi, 2014] T. Tassa and F. Bonchi. Privacy preserving estimation of social influence. In *EDBT*, pages 559–570, 2014.

[Teacy *et al.*, 2008] W. T. L. Teacy, A. Farinelli, N. J. Grabham, P. Padhy, A. Rogers, and N. R. Jennings. Max-Sum decentralised coordination for sensor systems. In *AAMAS*, pages 1697–1698, 2008.

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

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

[Zivan and Peled, 2012] R. Zivan and H. Peled. Max/min-sum distributed constraint optimization through value propagation on an alternating DAG. In *AAMAS*, pages 265–272, 2012.

[Zivan *et al.*, 2014] R. Zivan, S. Okamoto, and H. Peled. Explorative anytime local search for distributed constraint optimization. *Artificial Intelligence*, 212:1–26, 2014.