Privacy Preserving Solution of DCOPs by Local Search

Shmuel Goldklang[1], **Tal Grinshpoun**[2], **Tamir Tassa**[1]

[1]Department of Mathematics and Computer Science, The Open University of Israel

[2]Department of Industrial Engineering and Management, Ariel University

shmuel.goldklang@gmail.com, talgr@ariel.ac.il, tamirta@openu.ac.il

Abstract

One of the main reasons for solving constraint optimization problems in a distributed manner is maintaining agents' privacy. Several studies in the past decade devised privacy-preserving versions of Distributed Constraint Optimization Problem (DCOP) algorithms. Some of those algorithms were complete, i.e., finding an optimal solution, while others were incomplete. The main advantage of the incomplete approach is in its scalability to large problems. One of the important incomplete paradigms for solving DCOPs is local search. Yet, so far no privacy-preserving algorithm for solving DCOPs by means of local search was devised. We present P-DSA, a privacy-preserving implementation of the classical local-search algorithm DSA that preserves topology, constraint, and assignment/decision privacy. Comparing its performance to that of P-Max-Sum, which is another privacy-preserving implementation of an incomplete DCOP algorithm, shows that P-DSA is significantly more scalable and issues much better solutions than P-Max-Sum. Therefore, P-DSA emerges as a suitable solution for practitioners addressing large-scale DCOPs with privacy considerations.

1 Introduction

The Distributed Constraint Optimization Problem (DCOP) is a general model for solving distributed combinatorial problems that has a wide range of applications in artificial intelligence. Complete algorithms for DCOP-solving [Modi *et al.*, 2005; Petcu and Faltings, 2005; Gershman *et al.*, 2009] are guaranteed to find the optimal solution, but because DCOPs are NP-hard, these algorithms' worst-case runtime is exponential. Thus, there is a growing interest in incomplete algorithms, which may find sub-optimal solutions but run quickly enough to be applied on large-scale problems or real-time applications [Maheswaran *et al.*, 2004; Zhang *et al.*, 2005; Teacy *et al.*, 2008; Zivan *et al.*, 2014].

Approaches of incomplete DCOP algorithms include inference (Max-Sum [Farinelli *et al.*, 2008]), sampling (DUCT [Ottens *et al.*, 2017], D-Gibbs [Nguyen *et al.*, 2019]), region optimal (KOPT [Katagishi and Pearce, 2007], DALO [Kiekintveld *et al.*, 2010]), and local search (DSA [Zhang *et al.*, 2005], MGM [Maheswaran *et al.*, 2004], and DBA [Hirayama and Yokoo, 2005]). The latter approach is extremely popular due to its simplicity and runtime efficiency.

Privacy is one of the main motivations for solving constraint problems in a distributed manner. Preserving privacy is most important in distributed scenarios in which agents represent people who would not like their personal preferences and actions to be revealed, e.g., meeting scheduling [Gershman *et al.*, 2008], and smart environments (such as smart homes) [Rust *et al.*, 2016; Fioretto *et al.*, 2017]. The term privacy is quite broad, a fact that gave rise to several categorizations of the different types of privacy [Léauté and Faltings, 2013; Greenstadt *et al.*, 2007; Grinshpoun, 2012]. In this paper, we relate to the categorization of Léauté and Faltings [2013] that distinguishes between agent privacy, topology privacy, constraint privacy, and decision privacy.

Most studies that evaluated distributed constraint algorithms in terms of privacy considered complete algorithms [Silaghi and Mitra, 2004; Maheswaran *et al.*, 2006; Greenstadt *et al.*, 2006; Doshi *et al.*, 2008; Léauté and Faltings, 2013; Grinshpoun and Tassa, 2016]. Some work has focused on measuring the extent of constraint privacy loss [Maheswaran *et al.*, 2006; 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. Most efforts in the development of privacy-preserving search algorithms focused on DCSP, which is the *satisfaction* variant of DCOP. Examples include [Nissim and Zivan, 2005; Silaghi and Mitra, 2004; Yokoo *et al.*, 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 of all possible assignments. Several privacy-preserving versions of the DPOP algorithm [Petcu and Faltings, 2005] were proposed in the past [Greenstadt *et al.*, 2007; Silaghi *et al.*, 2006], including a more recent study by Léauté and Faltings [2013] that proposed several versions of DPOP that provide strong privacy guarantees. While these versions have been designed for DCSPs, some of them may also be applicable to DCOPs. Explicitly for DCOPs, Grinshpoun and Tassa [2016] and Tassa et al. [2021] devised variants of

SyncBB [Hirayama and Yokoo, 1997], which preserve topology, constraint, and decision privacy.

While the problem sizes for which complete DCOP algorithms are applicable are limited, the problem worsens when privacy-preserving algorithms are considered, due to the substantial runtime overhead that privacy preservation incurs. Consequently, several studies focused on privacy-preserving incomplete algorithms. Tassa et al. [2017] and Kogan et al. [2023] proposed variations of an incomplete inference algorithm, Max-Sum [Farinelli *et al.*, 2008], which preserve topology, constraint, and decision privacy. Additionally, Grinshpoun et al. [2019] devised an incomplete region-optimal algorithm that preserves constraint privacy and partial decision privacy. However, though incomplete, the above algorithms are very elaborate and are inapplicable to large-scale problems. Specifically, the runtime of the Max-Sum-based algorithms is exponential in the *arity* of the constraints, which makes them unsuitable for problems with global constraints, e.g., satellite scheduling [Krigman *et al.*, 2024] and course allocation [Khakhiashvili *et al.*, 2021].

Recently, Vion et al. [2022] proposed a local search algorithm that *controls* the loss of domain privacy [Grinshpoun, 2012] by following the Utilitarian DCOP model [Doshi *et al.*, 2008; Savaux *et al.*, 2020], in which privacy loss is traded off with solution quality. However, their approach is only relevant in the Open Constraints Programming model [Faltings and Macho-Gonzalez, 2005], where the domains are not known in advance and grow as the solving process advances.

Our contributions. We present here P-DSA, a privacy-preserving implementation of the classical local-search algorithm DSA. We show that it offers topology privacy, constraint privacy, and assignment/decision privacy. We compare its performance to that of P-Max-Sum [Tassa *et al.*, 2017], a privacy-preserving implementation of the incomplete DCOP algorithm Max-Sum [Farinelli *et al.*, 2008] which also protects topology, constraint and assignment/decision information. We show that P-DSA is significantly more scalable and issues much better solutions than P-Max-Sum. In fact, while P-DSA was able to solve in short time (3 minutes) problems involving as high as 100 agents, prior studies on privacy-preserving DCOP algorithms report experiments with at most 24 agents and runtimes that are significantly higher.[1] Therefore, P-DSA emerges as a suitable choice for solving large-scale DCOPs in a privacy-preserving manner.

2 DCOP background

A Distributed Constraint Optimization Problem (DCOP, [Hirayama and Yokoo, 1997]) is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ where $\mathcal{A} = \{A_1, \ldots, A_n\}$ is a set of agents, $\mathcal{X} = \{X_1, \ldots, X_n\}$ is a set of variables, $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of finite domains, 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 the agent A_i. Each constraint $C \in \mathcal{R}$ defines for a given pair of variables some non-negative cost; formally, a constraint takes the form $C_{i,j} : D_i \times D_j \to [0, q]$, for some $1 \le i \le j \le n$, where q is a publicly known maximal constraint cost q. (Note that if $i = j$ then the constraint is unary.) The goal in constraint optimization problems is to find an assignment of values to all n variables,

$$(X_1, \ldots, X_n) \leftarrow \mathbf{x} := (x_1, \ldots, x_n) \in \mathbf{D} := D_1 \times \cdots \times D_n \,,$$

such that the overall incurred cost $\sum_{C_{i,j} \in \mathcal{R}} C_{i,j}(x_i, x_j)$ is minimal.

Our framework can also include the case of *hard* constraints, i.e., combinations of assignments that are strictly forbidden, see [Kumar *et al.*, 2008]. Our framework is the one that is studied in most prior art. Some studies consider extensions to this framework by (a) assuming that each agent may hold more than one variable [Yokoo and Hirayama, 2000; Burke and Brown, 2006; Grinshpoun, 2015; Fioretto *et al.*, 2016], (b) including constraints of arity greater than two [Kim and Lesser, 2013], and (c) assuming asymmetric constraints that incur different costs to each of the involved agents [Grinshpoun *et al.*, 2013]. However, here we focus on the framework as defined above, which already introduces the main challenges of DCOPs.

Léauté and Faltings [2013] distinguished between four notions of privacy: agent privacy (who are the agents in the problem setting), topology privacy (hiding information on the constraint graph), constraint privacy (hiding information on the costs in the constraints), and assignment/decision privacy (protecting the intermediate/final assignments).

2.1 The Distributed Stochastic Algorithm

Here we describe the classic local search DCOP algorithm that was presented by Zhang et al. [2005] – the Distributed Stochastic Algorithm (DSA). We start by introducing a key notion in local search algorithms:

Definition 1 (Neighborhood)**.** *The neighborhood of agent A_i is the set of all agents that are constrained with A_i, i.e., $N(A_i) := \{A_j \in \mathcal{A} : \exists C_{i,j} \in \mathcal{R}\}$. The complete neighborhood of A_i is $N^+(A_i) := N(A_i) \cup \{A_i\}$.*

Algorithm 1: The DSA algorithm

```
forall i ∈ [n] do
    A_i selects at random x_i ∈ D_i
forall ℓ = 1, ..., L do
    forall i ∈ [n] do
        A_i sends x_i to all A_j ∈ N(A_i)
    forall i ∈ [n] do
        A_i samples uniformly at random a real
          x ∈ [0, 1]
        if x ≤ p then
            A_i chooses y_i ∈ D_i that minimizes
              Σ_{A_j ∈ N(A_i)} C_{i,j}(y_i, x_j)
            A_i updates x_i ← y_i
```

[1] To the best of our knowledge, the only exception is the work of Damle et al. [2024] that presented P-Gibbs, which is a differentially private implementation of SD-Gibbs [Nguyen *et al.*, 2019]. However, differential privacy is a paradigm that is based on injecting random noise; hence it is not directly comparable to the cryptographic paradigm that does not alter the outputs of the underlying algorithm.

Algorithm 1 describes DSA. The algorithm starts by generating an initial random assignment $\mathbf{a} \in \mathbf{D}$ (Lines 1-2).[2] It then keeps updating that assignment by performing a preset number of iterations L (Lines 3-10). The assignment in the final iteration is the algorithm's output. Each iteration starts with each agent updating its neighbors on its current assignment (Lines 4-5). Then, each agent is allowed, with probability p, to change its local assignment to the best possible value (Lines 6-10).

The utilization of the stochastic factor p enables DSA to escape local minima and avoid infinite loops. However, it renders DSA non-monotone in the sense that the cost of the solution in one iteration is not necessarily smaller than the cost of the solution in the previous iteration. It is possible to enhance DSA with a so-called *anytime mechanism* [Zivan *et al.*, 2014]. Such a mechanism finds the best solution visited throughout the run of the algorithm. In general, in order to report the best solution visited, the algorithm needs to compute the overall cost after each iteration, and if that overall cost is the minimum so far, record that cost and the corresponding assignment.

3 Cryptographic background

Here, we briefly describe the cryptographic machinery we use in our protocols. In Section 3.1 we discuss threshold secret sharing, and then, in Section 3.2, we describe secure computations over secret-shared values.

3.1 Shamir's secret sharing

Secret sharing schemes [Shamir, 1979] are protocols that enable distributing a secret scalar s among a set of agents, $A_1, \ldots, A_n$. Each agent, A_h, $h \in [n]$, gets a random value $[[s]]_h$, called *a share*, so that some subsets of those shares enable the reconstruction of s, while each of the other subsets of shares reveals no information on s. In its most basic form, called *Threshold Secret Sharing*, there is a threshold value $t \leq n$, and then a subset of shares enables the reconstruction of s iff its size is at least t.

Shamir's t-out-of-n threshold secret sharing scheme [Shamir, 1979] operates over a finite field $\mathbf{Z}_q$, where $q > n$ is a prime sufficiently large so that all possible secrets may be represented in $\mathbf{Z}_q$. It has two procedures: Share and Reconstruct:

• $\text{Share}_{t,n}(s)$. The procedure samples a uniformly random polynomial $f(\cdot)$ over $\mathbf{Z}_q$, of degree at most $t-1$, where the free coefficient is the secret s. That is, $f(x) = s + a_1 x + a_2 x^2 + \ldots + a_{t-1} x^{t-1}$, where a_j, $1 \leq j \leq t-1$, are selected independently and uniformly at random from $\mathbf{Z}_q$. The procedure outputs n values, $[[s]]_h = f(h)$, $h \in [n]$, where $[[s]]_h$ is the share given to A_h. The entire set of sheares, denoted $[[s]] := \{[[s]]_h : h \in [n]\}$, is called a (t,n)-sharing of s.

• $\text{Reconstruct}_t([[s]])$. The procedure is given any selection of t shares out of the (t,n)-sharing of s. It then interpolates a polynomial $f(\cdot)$ of degree at most $t-1$ using the given points and outputs $s = f(0)$. Any selection of t shares will yield the secret s, as t points determine a unique polynomial of degree at most $t-1$. On the other hand, any selection of $t-1$ shares or less reveals nothing about the secret s.

Hereinafter, we set the secret sharing threshold to be

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

Namely, to reconstruct the secret, at least half of the agents must collaborate. Hence, if the set of n agents has an honest majority (in the sense that more than $n/2$ agents would not try to combine their shares in order to recover secret values), all shared values will remain fully protected.

In what follows, we shall use the following terminology and notations. Let s be a secret known to some agent A_i, $i \in [n]$. Then if A_i performs the procedure $\text{Share}_{t,n}(s)$, we will simply say that A_i distributes a (t,n)-sharing of s.

If the agents have a (t,n)-sharing $[[s]]$ in some secret s and they wish to let one of them, say A_i, reconstruct the secret s, then at least $t-1$ agents would send their shares to A_i who will proceed to apply the Reconstruct procedure on the t shares it has. We will describe this procedure shortly by writing $s \leftarrow \text{Reconstruct}([[s]]; A_i)$.

If $\mathbf{s} = (s_1, \ldots, s_m) \in \mathbf{Z}_q^m$ is a vector of secrets held by A_i, then by saying that A_i distributes a (t,n)-sharing of $\mathbf{s}$ we mean that A_i distributes a (t,n)-sharing of each of the entries of $\mathbf{s}$, independently.

Let $a \in \mathbf{Z}_q$ be any value that is publicly known to all agents. Then by $[[a]]$ we mean the set of (t,n)-shares $\{[[a]]_h = a : h \in [n]\}$. It is easy to see that this set of shares indeed defines a unique polynomial of degree at most $t-1$, which is the constant polynomial $f(\cdot) \equiv a$, and therefore it is a proper (t,n)-sharing of the value a. Such a sharing does not require any communication between the agents nor any polynomial computations, since a is publicly known.

Let $a, b, c \in \mathbf{Z}_q$ be three values publicly known to all, and let s and s' be two secrets in which the agents already hold (t,n)-sharings, denoted $[[s]]$ and $[[s']]$. Then

$$a + b[[s]] + c[[s']] := \{a + b[[s]]_h + c[[s']]_h : h \in [n]\} \tag{2}$$

is a proper (t,n)-sharing of $\hat{s} := a + bs + cs'$, and its computation needs no interaction between the agents, thanks to the affinity of secret sharing. By writing

$$[[\hat{s}]] \leftarrow a + b[[s]] + c[[s']]$$

we mean that each agent A_h, $h \in [n]$, sets $[[\hat{s}]]_h \leftarrow a + b[[s]]_h + c[[s']]_h$, so that now the agents hold a (t,n)-sharing of $\hat{s} = a + bs + cs'$ without needing to interact or perform any further polynomial computations.

3.2 Secure computations over secret sharings

Let a and b be two secret values in the field $\mathbf{Z}_q$, and assume that $A_1, \ldots, A_n$ hold (t,n)-sharings in them, denoted $[[a]] = \{[[a]]_h : h \in [n]\}$ and $[[b]] = \{[[b]]_h : h \in [n]\}$. A secure multiplication protocol is a protocol of the form

$$[[c]] \leftarrow \text{SecureMult}([[a]], [[b]]) , \tag{3}$$

that takes the (t,n)-sharings of a and b and computes from them a (t,n)-sharing of $c = a \cdot b$ in a secure manner, namely, without revealing to the agents any information on a, b, or $c = ab$. Damgård and Nielsen [2007] designed such a secure

[2]Throughout this paper, for any integer n, $[n] := \{1, \ldots, n\}$.

multiplication protocol. In our experiments, we used that protocol with the performance improvements that were proposed by Chida et al. [2018].

Another computation on secret shares that we will need is secure comparison. Under the same assumptions as above, a secure comparison protocol is a protocol of the form

$$[[c]] \leftarrow \text{SecureCompare}([[a]],[[b]])\,, \tag{4}$$

that takes the (t,n)-sharings of a and b and computes from them a (t,n)-sharing of $c = 1_{a<b}$, where hereinafter if $\mathcal{P}$ is a predicate then $1_{\mathcal{P}}$ is a bit that equals 1 if the predicate $\mathcal{P}$ holds and equals 0 otherwise. As before, such a protocol is secure in the sense that it does not reveal to the agents any information on a, b, or $c = 1_{a<b}$. Nishide and Ohta [2007] proposed such a secure comparison protocol.

4 Private DSA

In this section, we describe Private DSA (P-DSA), an implementation of DSA that preserves topology, constraint, and decision privacy. In order to achieve those privacy goals, P-DSA employs the following principles:

(1) To achieve topology privacy, every pair of agents that are not constrained creates a zero constraint matrix between themselves, and, subsequently, the algorithm acts on a complete constraint graph. None of the other agents is able to distinguish between fake constraint matrices (i.e., zero matrices) and genuine ones due to the next principle in P-DSA's design, which distinguishes its operation from that of the basic DSA.

(2) To achieve constraint privacy, all constraint matrices are secret-shared among all agents, and all computations that rely on those matrices use the shares rather than the actual constraint matrices.

(3) To achieve decision privacy, in each iteration of the algorithm whenever an agent selects an assignment to its variable, it does not send that assignment to its neighbors; instead, it secret shares information on the costs that such an assignment incurs vis-a-vis each of the other agents.

The latter principle raises a considerable computational challenge: how can each of the agents perform the computations that DSA mandates when it does not know the current assignments of its neighboring agents? We tackle that challenge by designing multi-party sub-protocols to be run jointly by all agents. In those collaborative sub-protocols, all agents use the secret shares they hold in order to enable each agent to compute the next assignment from its domain. In doing so, none of the agents get any wiser about that assignment or any other private information.

We assume hereinafter that all agents know the sizes of all domains, namely, $m_i := |D_i|$ for all $i \in [n]$. Moreover, each agent A_i, $i \in [n]$, generates an ordering of the values in its domain, $D_i = \{a^i_1, \ldots, a^i_{m_i}\}$, and publishes that ordering to each of its neighbors, $A_j \in N(A_i)$. Therefore, each constraint $C_{i,j}$ can be described as a matrix of m_i rows and m_j columns, where $C_{i,j}(r,s)$ equals the value of the constraint when $X_i = a^i_r$ and $X_j = a^j_s$. In what follows, we will think of $C_{i,j}$ as a matrix rather than a function over $D_i \times D_j$.

Protocol 2 describes P-DSA — a private implementation of DSA. First, each agent A_i selects a random assignment to its variable. A_i does that by selecting a random index $r_i \in [m_i]$, and then the corresponding assignment to X_i is $a^i_{r_i}$, $i \in [n]$ (Lines 1-2).

The main loop takes place in Lines 3-15. First, each agent A_i, $i \in [n]$, secretly shares its current assignment, $a^i_{r_i}$, with *all* agents. To do that, A_i distributes to all agents (t,n)-shares in the r_i-th row in each of the constraint matrices that it has vis-a-vis each of the other $n-1$ agents (namely, also with agents outside its neighborhood). Let $\mathbf{w}_{i,j}$ denote the r_i-th row in the constraint matrix $C_{i,j}$, for some $j \in [n] \setminus \{i\}$, i.e.,

$$\mathbf{w}_{i,j} = (\mathbf{w}_{i,j}(u) : u \in [m_j])\,, \\ \text{where} \quad \mathbf{w}_{i,j}(u) = C_{i,j}(r_i,u)\,, u \in [m_j]\,. \tag{5}$$

A_i distributes (t,n)-shares in each of the m_j entries of that vector, where the sharing of $\mathbf{w}_{i,j}(u)$ is denoted $[[\mathbf{w}_{i,j}(u)]] = \{[[\mathbf{w}_{i,j}(u)]]_h : h \in [n]\}$, while the sharing of the entire vector is denoted $[[\mathbf{w}_{i,j}]]$. The overall number of scalars that A_i shares at this stage (Lines 4-6) is $\sum_{j\in[n]\setminus\{i\}} m_j$.

We would like to clarify that the secret sharing done in Lines 4-6 is excessive. Indeed, if $a \neq b \in [n]$ then the scalar $C_{a,b}(r_a, r_b)$ is shared when $i = a$ and $j = b$, as it is in the r_a-th row of the matrix $C_{a,b}$, but also when $i = b$ and $j = a$, as it is in the r_b-th row of the matrix $C_{b,a}$ which is the transpose of $C_{a,b}$. However, this excessive secret sharing will pay off later on in the computation.

Before moving on, let us fix $i \in [n]$ and $j \in [n] \setminus \{i\}$. Then for any $u \in [m_i]$, $\mathbf{w}_{j,i}(u)$ is the cost that A_i would pay if it sets $X_i = a^i_u$, given the current assignment of A_j to its variable, $X_j = a^j_{r_j}$. Therefore, if we define

$$\mathbf{w}_i(u) := \sum_{j\in[n]\setminus\{i\}} \mathbf{w}_{j,i}(u)\,, \quad u \in [m_i]\,, \tag{6}$$

we have by Eq. (5) and the symmetry of the constraints (in the sense that $C_{i,j} = C^T_{j,i}$),

$$\mathbf{w}_i(u) = \sum_{j\in[n]\setminus\{i\}} C_{j,i}(r_j,u) = \sum_{j\in[n]\setminus\{i\}} C_{i,j}(u,r_j)\,, \quad u \in [m_i]\,. \tag{7}$$

Hence, $\mathbf{w}_i(u)$ is the overall cost for A_i if it sets $X_i = a^i_u$, given the current assignments that all other agents have for their variables. In Lines 7-9 all agents compute (t,n)-shares in $\mathbf{w}_i(u)$ for all $i \in [n]$ and for all $u \in [m_i]$. Note that it is a local computation that does not require the agents to communicate.

Next, the main task of each agent A_i is to find the best assignment to its variable given the current assignments of all neighboring variables (as encoded in the secret shares that all agents have distributed in Lines 4-6) and storing the index of that assignment in r_i. However, we recall that such a computation takes place only in probability p, while otherwise, in probability $1-p$, A_i retains its current assignment. Hence, A_i starts by generating a uniformly random real number $x \in [0,1]$ (Line 11), and only if $x \le p$ it proceeds to the computational task of finding the best assignment for its variable, given the current assignments of its neighboring agents. That computation is carried out in the sub-protocol FindBestAssignment (Line 13). In that sub-protocol, the agents jointly and securely compute a (t,n)-sharing of the index $k_i \in [m_i]$ of the currently best assignment to X_i from D_i. After its completion, all agents send to A_i their shares in k_i, and A_i proceeds to recover k_i (Line 14) and store it in r_i (Line 15).

After performing L such iterations (Lines 3-15), each of the agents stores the last assignment to its variable (Lines 16-17).

Protocol 2: P-DSA – Private DSA

1 **forall** $i \in [n]$ **do**
2 | A_i selects at random $r_i \in [m_i]$
3 **forall** $\ell = 1, \ldots, L$ **do**
4 | **forall** $i \in [n]$ **do**
5 | | **forall** $j \in [n] \setminus \{i\}$ **do**
6 | | | A_i distributes a (t, n)-sharing of $[[\mathbf{w}_{i,j}]]$
7 | **forall** $i \in [n]$ **do**
8 | | **forall** $u \in [m_i]$ **do**
9 | | | $[[\mathbf{w}_i(u)]] \leftarrow \sum_{j \in [n] \setminus \{i\}} [[\mathbf{w}_{j,i}(u)]]$
10 | **forall** $i \in [n]$ **do**
11 | | A_i samples uniformly at random $x \in [0, 1]$
12 | | **if** $x \leq p$ **then**
13 | | | FindBestAssignment$(i; [[k_i]])$
14 | | | $k_i \leftarrow$ Reconstruct$([[k_i]]; A_i)$
15 | | | A_i sets $r_i \leftarrow k_i$
16 **forall** $i \in [n]$ **do**
17 | A_i sets $X_i \leftarrow a^i_{r_i}$

4.1 The sub-protocol FindBestAssignment

Here, we describe Sub-protocol 3, called FindBestAssignment. The sub-protocol, which is executed by all agents, scans the values in X_i's domain, $D_i = \{a^i_u : u \in [m_i]\}$, and computes a (t, n)-sharing $[[k_i]]$ in the index $k_i \in [m_i]$ that issues the currently minimal aggregated cost for A_i.

Before describing the computations in the sub-protocol, we make the following observations. Let c_i and c_j be two indexed scalars, where $i < j$. Then

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

and

$$\arg\min(c_i, c_j) = i + 1_{c_j < c_i} \cdot (j - i) \tag{9}$$

(by $\arg\min$ we mean the smallest index in which the minimum is attained). Hence, if the agents hold (t, n)-shares in c_i and in c_j, they can jointly compute (t, n)-shares in $\min(c_i, c_j)$ and in $\arg\min(c_i, c_j)$, without learning any information on c_i and c_j, by invoking the secure comparison and multiplication protocols from Section 3.2. Specifically, they will first run

$$[[\beta]] \leftarrow \text{SecureCompare}([[c_j]], [[c_i]])$$

(see Eq. (4)) so that they will hold (t, n)-shares in the bit $\beta := 1_{c_j < c_i}$. Then they will run the secure multiplication protocol (see Eq. (3)),

$$[[\gamma]] \leftarrow \text{SecureMult}([[\beta]], [[c_j]] - [[c_i]])$$

to get (t, n)-shares in $\gamma := 1_{c_j < c_i} \cdot (c_j - c_i)$. Finally, each agent A_h, $h \in [n]$, will compute

$$[[w]]_h \leftarrow [[c_i]]_h + [[\gamma]]_h \,.$$

In view of Eq. (8), the set $[[w]] = \{[[w]]_h : h \in [n]\}$ is a (t, n)-sharing of $w := \min(c_i, c_j)$. In the process of computing those shares, the agents remain completely oblivious to the values of c_i, c_j, and w. A similar course of action can issue to the agents a (t, n)-sharing of $\arg\min(c_i, c_j)$, using Eq. (9).

We now turn to Sub-protocol 3. Its input is the index i of the agent who looks for the currently best assignment to its variable. Recall that FindBestAssignment is invoked from Protocol 2 in Line 13. At that stage in Protocol 2, all agents hold (t, n)-shares in $\mathbf{w}_i(u)$ for all $i \in [n]$ and all $u \in [m_i]$, being the aggregated cost for A_i if it sets $X_i \leftarrow a^i_u$, given the current assignments to the variables held by its neighbors.

The sub-protocol scans A_i's domain, D_i, and updates two values: k_i that will hold the index of the currently best assignment and w_i that will hold the corresponding cost. Those two values will not be computed explicitly; instead, the agents will hold secret shares in them.

Initially (Lines 1-2), the agents set $k_i = 1$ and $w_i = \mathbf{w}_i(1)$. Since the agents already hold a secret sharing of the latter value, they simply set $[[w_i]]_h = [[\mathbf{w}_i(1)]]_h$, $h \in [n]$. As for $k_i = 1$, since it is a publicly known value, then, in view of our discussion in Section 3.1, each agent sets $[[k_i]]_h = 1$, $h \in [n]$.

Next, the agents scan the remaining values in D_i (Lines 3-8). First, they compute shares in $\beta := 1_{\mathbf{w}_i(u) < w_i}$, using SecureCompare (see Eq. (4)), in order to compare w_i, the minimum found so far, to the cost of the next assignment, $\mathbf{w}_i(u)$ (Line 4). Then, they use SecureMult (see Eq. (3)) to compute shares in $\gamma := \beta \cdot (\mathbf{w}_i(u) - w_i)$ and in $\delta := \beta \cdot (u - k_i)$ (Lines 5-6). (Recall that since u is a publicly known value, each agent A_h, $h \in [n]$, sets locally $[[u]]_h = u$.) Finally, they update the shares in w_i and k_i using Eqs. (8) and (9), respectively (Lines 7-8). At the end of the loop, k_i equals the index of the best assignment, and w_i equals the associated cost. Since P-DSA needs only $[[k_i]]$, the sub-protocol issues that sharing as its output.

Comment. The computation of w_i (Lines 5+7) is needed for the computation of β (Line 4) in the subsequent iteration, a value that is used in updating k_i (Lines 6+8). Hence, since w_i is not a desired output of the sub-protocol, it is possible to skip Lines 5+7 in the last iteration ($u = m_i$).

Sub-protocol 3: FindBestAssignment – Computing a (t, n)-sharing of the index k_i of the currently best assignment for X_i.

Input: i – the index of agent A_i
1 **forall** $h \in [n]$ **do**
2 | A_h sets $[[k_i]]_h \leftarrow 1$ and $[[w_i]]_h \leftarrow [[\mathbf{w}_i(1)]]_h$
3 **forall** $u = 2, \ldots, m_i$ **do**
4 | $[[\beta]] \leftarrow$ SecureCompare$([[\mathbf{w}_i(u)]], [[w_i]])$
5 | $[[\gamma]] \leftarrow$ SecureMult$([[\beta]], [[\mathbf{w}_i(u)]] - [[w_i]])$
6 | $[[\delta]] \leftarrow$ SecureMult$([[\beta]], [[u]] - [[k_i]])$
7 | $[[w_i]] \leftarrow [[w_i]] + [[\gamma]]$
8 | $[[k_i]] \leftarrow [[k_i]] + [[\delta]]$
Output: A (t, n)-sharing of $[[k_i]]$

4.2 Privacy

Protocol 2 preserves topology, constraint, and assignment/decision privacy, owing to the cryptographic machinery that we use – see Theorem 1. It does not respect agent privacy since it requires all n agents to have a full communication network between them.

Theorem 1. *Under the assumption of honest majority, Protocol 2 preserves topology, constraint, and assignment/decision privacy.*

Proof. The honest majority assumption means that if there exist agents that will try combining their shares in attempt to recover some of the secret-shared values, their number will be smaller than the threshold $t = \lfloor (n+1)/2 \rfloor$, see Eq. (1). Shamir's secret sharing scheme is perfect, in the sense that any number of shares smaller than the threshold exposes zero information on the shared secret [Shamir, 1979]. Therefore, the secret shares in each of the private values that are secret-shared during P-DSA reveal no information on the underlying private value. Apart from secret sharing, the agents engage also in multi-party protocols for performing secure multiplication and secure comparison, see Eqs. (3) and (4). The protocols that we use are information-theoretic secure, see [Damgård and Nielsen, 2007; Chida *et al.*, 2018; Nishide and Ohta, 2007]. Given all of the above, it follows the P-DSA fully preserves all constraint information under the honest majority assumption; hence, it offers constraint privacy.

P-DSA operates over a complete constraint graph, in which every pair of agents has a constraint matrix between them. Since all matrices are secret-shared using the threshold t in Eq. (1), which guarantees perfect privacy under the assumption of honest majority, zero matrices are indistinguishable from matrices that represent actual constraints. Therefore, P-DSA offers also topology privacy.

As also all indices of all assignments are encoded through secret shares, we infer that all assignment information, as well as the final decisions, remain fully protected. Hence, P-DSA offers also assignment/decision privacy.

Note that while Protocol 2 hides from each agent the sequence of assignments of other agents, it does reveal to each agent its own sequence of assignments. Protocol 2 can be further enhanced to also hide from each agent the sequence of value assignments to its own variable, including the initial random value assignment. Due to space limitations, we omit the details of this enhancement.

5 Experiments

We implemented P-DSA and compared its performance to P-Max-Sum [Tassa *et al.*, 2017], which is a privacy-preserving implementation of an incomplete DCOP algorithm (Max-Sum [Farinelli *et al.*, 2008]).

Experiments were conducted on a machine equipped with an Intel i5-10400 CPU @ 2.90GHz, 2904 Mhz, 6 Core(s), 12 Logical Processor(s), 16GB DDR4 RAM. The system ran Microsoft Windows 10 Pro, and the code was written in Java 1.8.0 using the SinAlgo simulation framework. The source code is available on https://github.com/dcop2025/dcop-sim/tree/main.

P-DSA was implemented over $\mathbf{Z}_q$ with $q = 2^{31} - 1$. P-Max-Sum was implemented with 512-bit homomorphic encryption.

In our experiments, we compared the quality of the solutions issued by each of those two algorithms within a given time frame. We used the following settings of the main parameters that affect the algorithms' runtimes:

- Number of agents $n \in \{10, 20, \underline{\mathbf{30}}, 40, \dots, 100\}$.
- Domains' size $m \in \{5, \underline{\mathbf{10}}, 15, 20, 25\}$. For simplicity, we assumed that all domains have the same size m.
- Constraint density, $d \in \{0.2, \underline{\mathbf{0.4}}, 0.6, 0.8, 1.0\}$ — the fraction of constrained pairs of variables out of all $\binom{n}{2}$ pairs.

To test the effect of each of those three parameters, we set the other two to the value that is underlined in their respective set of tested values and varied the value of the tested parameter. For example, in testing the effect of the number of agents, we set all domain sizes to be $m = 10$ and used constraint density of $d = 0.4$ and then ran experiments with $n \in \{10, \dots, 100\}$.

We refer to each triple $\langle n, m, d \rangle$ as a *configuration*. In each tested configuration, we evaluated both algorithms in the following manner: We selected a new random problem (where a problem consists of the constraint graph as well as the constraint matrices), ran both algorithms on the same problem, and evaluated the cost of their output after $T = 1, 2, 3$ minutes of execution. We repeated that experiment 20 times, and we report the average of the costs obtained by each of the two algorithms within each of the prescribed time frames.

In one set of experiments we used random constraint graphs, where each graph is a random graph of n nodes in which each pair of nodes is connected by an edge in probability d. In another set of experiments we generated scale-free random graphs [Barabási and Albert, 1999] with an initial clique of size 5, and 4 backward edges for each additional node. In all experiments, each constraint matrix was a random $m \times m$ matrix with entries that distribute uniformly on the interval $[0, 10]$.

Number of agents in random graphs. We compared the average cost of solutions issued by each of the two algorithms within each of the three prescribed time frames for a varying number of agents n (where in all problems, the domain size was $m = 10$ and the network density was $d = 0.4$). Table 1 shows the average costs issued by the two algorithms (rounded to the nearest integer). The symbol $\perp$ indicates that the algorithm did not manage to complete even one iteration within the time frame.

We see the overwhelming advantages of P-DSA over P-Max-Sum in terms of scalability and quality of solutions. Indeed, while P-Max-Sum could not produce a solution within 1 minute already for $n = 40$ and could not produce a solution within 3 minutes for $n \geq 60$, P-DSA was able to produce solutions within 1 minute for all $n \leq 60$ and managed to produce a solution within 3 minutes for all tested values of n. Furthermore, the solutions produced by P-DSA were better than those issued by P-Max-Sum by more than 50%

T	$n = 10$	20	30	40	50	60	70	80	90	100
1	$27 \mid 66$	$207 \mid 330$	$591 \mid 783$	$1280 \mid \perp$	$2197 \mid \perp$	$3127 \mid \perp$	$\perp \mid \perp$	$\perp \mid \perp$	$\perp \mid \perp$	$\perp \mid \perp$
2	$27 \mid 63$	$186 \mid 333$	$526 \mid 733$	$1121 \mid 1433$	$1944 \mid \perp$	$3023 \mid \perp$	$4361 \mid \perp$	$5796 \mid \perp$	$\perp \mid \perp$	$\perp \mid \perp$
3	$27 \mid 59$	$178 \mid 318$	$494 \mid 763$	$1058 \mid 1362$	$1822 \mid 2299$	$2902 \mid \perp$	$4112 \mid \perp$	$5752 \mid \perp$	$7354 \mid \perp$	$9087 \mid \perp$

Table 1: Average costs obtained by P-DSA (left in each table cell) and P-Max-Sum (right) for problems in random graphs over a varying number n of agents, within time frames of $T = 1, 2, 3$ minutes. The symbol $\perp$ indicates that the algorithm did not manage to complete even a single iteration within the time frame.

T	$n = 10$	20	30	40	50	60	70	80	90	100
1	$55 \mid 101$	$503 \mid 661$	$1496 \mid \perp$	$3083 \mid \perp$	$5211 \mid \perp$	$7732 \mid \perp$	$\perp \mid \perp$	$\perp \mid \perp$	$\perp \mid \perp$	$\perp \mid \perp$
2	$53 \mid 118$	$467 \mid 683$	$1386 \mid 1671$	$2836 \mid \perp$	$4897 \mid \perp$	$7482 \mid \perp$	$10717 \mid \perp$	$14100 \mid \perp$	$\perp \mid \perp$	$\perp \mid \perp$
3	$53 \mid 106$	$463 \mid 677$	$1325 \mid 1671$	$2740 \mid \perp$	$4652 \mid \perp$	$7260 \mid \perp$	$10321 \mid \perp$	$14069 \mid \perp$	$18095 \mid \perp$	$\perp \mid \perp$

Table 2: Similar to Table 1 but with scale-free graphs.

for $n = 10$ and by 20% for the largest problem in which P-Max-Sum issued a solution. In addition, we see that P-DSA always improves the quality of its output when allowed to run for more time, while P-Max-Sum sometimes fluctuates (see, e.g., its outputs when $n = 30$). That is why it is sometimes executed with the anytime mechanism [Zivan *et al.*, 2014] that outputs the best solution visited throughout the run of the algorithm. Such a mechanism has its overhead, and in P-DSA, it appears that there is less need to apply it. (It is important to stress that P-DSA and P-Max-Sum issue the very same intermediate and final assignments as DSA and Max-Sum, respectively. Namely, the cryptographic layer protects the underlying private information but does not alter it.)

T	$m = 5$	10	15	20	25
1	$560 \mid 714$	$591 \mid 783$	$669 \mid \perp$	$662 \mid \perp$	$715 \mid \perp$
2	$550 \mid 722$	$526 \mid 733$	$570 \mid 830$	$587 \mid \perp$	$619 \mid \perp$
3	$550 \mid 689$	$494 \mid 763$	$516 \mid 830$	$527 \mid 834$	$550 \mid \perp$

Table 3: Average costs obtained by P-DSA (left in each table cell) and P-Max-Sum (right) for problems in random graphs over a varying domain size m, within time frames of $T = 1, 2, 3$ minutes.

T	$d = 0.2$	0.4	0.6	0.8	1.0
1	$254 \mid 316$	$591 \mid 783$	$980 \mid \perp$	$1353 \mid \perp$	$1784 \mid \perp$
2	$198 \mid 317$	$526 \mid 733$	$889 \mid 1212$	$1244 \mid 1641$	$1645 \mid 2053$
3	$174 \mid 327$	$494 \mid 763$	$843 \mid 1179$	$1197 \mid 1588$	$1575 \mid 2023$

Table 4: Average costs obtained by P-DSA (left in each table cell) and P-Max-Sum (right) for problems in random graphs over a varying constraint density d, within time frames of $T = 1, 2, 3$ minutes.

Number of agents in scale-free graphs. We repeated the previous experiment, but this time with scale-free graphs. The results are given in Table 2. Here, too, we see that P-DSA is more scalable and issues better solutions.

Domain size in random graphs. Here we fixed $n = 30$ and $d = 0.4$ and varied the domain size m. The results are given in Table 3. As already demonstrated, P-DSA is more scalable than P-Max-Sum and managed to issue outputs to problems in which P-Max-Sum failed to complete even one iteration within the same time frame. Moreover, when both algorithms issued outputs, those of P-DSA had costs lower than those of P-Max-Sum, with improvements ranging from 22% to 38%.

Constraint density in random graphs. Here we fixed $n = 30$ and $m = 10$ and varied the constraint density d. The results are given in Table 4. As before, P-DSA issues solutions with costs that are significantly lower than P-Max-Sum's (where in one configuration, the improvement was as high as 47%). As for scalability, P-DSA's runtime does not depend on the network density since it operates on the complete graph, where non-constrained pairs of agents are connected by an edge with a zero constraint matrix. P-Max-Sum, on the other hand, works on the original constraint graph, and therefore, its runtime does depend on the network density. Hence, it failed to issue an output on dense networks for which P-DSA did issue an output.

Due to lack of space we omit description of experiments that compare the runtimes of P-DSA and the basic DSA, in order to illustrate the price of privacy. We intend to include such experiments in the full version of this study.

6 Conclusion

We presented here P-DSA – the first privacy-preserving implementation of a DCOP algorithm that is based on local search. It offers topology, constraint, and assignment/decision privacy. The algorithm is much more scalable than P-Max-Sum, a privacy-preserving implementation of another incomplete DCOP algorithm. It also offers solutions with much better costs than those issued by P-Max-Sum. Since P-DSA was able to solve in short time (3 minutes) problems involving as high as 100 agents, while all prior studies on privacy-preserving DCOP algorithms report experiments of much smaller scale and runtimes that are significantly higher, P-DSA emerges as a suitable choice for solving large-scale DCOPs in a privacy-preserving manner.

Our approach can also be extended to develop a privacy-preserving version of MGM [Maheswaran *et al.*, 2004], another local search algorithm for DCOPs. Even though the "basic plot" in MGM is similar to DSA's, it is more involved as the decision to update local assignments is taken based on a competition among agents and not by a coin-toss. Implementing the more intricate logic of MGM in a privacy-preserving manner is a challenge that we intend to undertake in a future research.

Acknowledgments

This work was partially supported by the Ministry of Innovation, Science and Technology, Israel.

References

[Barabási and Albert, 1999] A.L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[Burke and Brown, 2006] D. Burke and K. Brown. A comparison of approaches to handling complex local problems in DCOP. In *Distributed Constraint Satisfaction Workshop*, pages 27–33, 2006.

[Chida *et al.*, 2018] K. Chida, D. Genkin, K. Hamada, D. Ikarashi, R. Kikuchi, Y. Lindell, and A. Nof. Fast large-scale honest-majority MPC for malicious adversaries. In *CRYPTO*, pages 34–64, 2018.

[Damgård and Nielsen, 2007] I. Damgård and J.B. Nielsen. Scalable and unconditionally secure multiparty computation. In *CRYPTO*, pages 572–590, 2007.

[Damle *et al.*, 2024] S. Damle, A. Triastcyn, B. Faltings, and S. Gujar. Differentially private multi-agent constraint optimization. *Autonomous Agents and Multi-Agent Systems*, 38, 2024.

[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.

[Faltings and Macho-Gonzalez, 2005] B. Faltings and S. Macho-Gonzalez. Open constraint programming. *Artificial Intelligence*, 161:1-2:181–208, 2005.

[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.

[Fioretto *et al.*, 2016] F. Fioretto, W. Yeoh, and E. Pontelli. Multi-variable agents decomposition for dcops. In *AAAI*, volume 30, 2016.

[Fioretto *et al.*, 2017] F. Fioretto, W. Yeoh, and E. Pontelli. A multiagent system approach to scheduling devices in smart homes. In *AAAI workshops*, 2017.

[Gershman *et al.*, 2008] A. Gershman, A. Grubshtein, A. Meisels, L. Rokach, and Roie Zivan. Scheduling meetings by agents. In *PATAT*, 2008.

[Gershman *et al.*, 2009] A. Gershman, A. Meisels, and R. Zivan. Asynchronous forward bounding for distributed COPs. *Journal of Artificial Intelligence Research*, 34:61–88, 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, 2016] T. Grinshpoun and T. Tassa. P-SyncBB: A privacy preserving branch and bound DCOP algorithm. *Journal of Artificial Intelligence Research*, 57:621–660, 2016.

[Grinshpoun *et al.*, 2013] T. Grinshpoun, A. Grubshtein, R. Zivan, A. Netzer, and A. Meisels. Asymmetric distributed constraint optimization problems. *Journal of Artificial Intelligence Research*, 47:613–647, 2013.

[Grinshpoun *et al.*, 2019] T. Grinshpoun, T. Tassa, V. Levit, and R. Zivan. Privacy preserving region optimal algorithms for symmetric and asymmetric DCOPs. *Artificial Intelligence*, 266:27–50, 2019.

[Grinshpoun, 2012] T. Grinshpoun. When you say (DCOP) privacy, what do you mean? - categorization of DCOP privacy and insights on internal constraint privacy. In *ICAART*, pages 380–386, 2012.

[Grinshpoun, 2015] T. Grinshpoun. Clustering variables by their agents. In *WI-IAT*, pages 250–256, 2015.

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

[Hirayama and Yokoo, 2005] K. Hirayama and M. Yokoo. The distributed breakout algorithms. *Artificial Intelligence*, 161:89–115, 2005.

[Katagishi and Pearce, 2007] H. Katagishi and J.P. Pearce. KOPT: Distributed DCOP algorithm for arbitrary k-optima with monotonically increasing utility. In *DCR*, 2007.

[Khakhiashvili *et al.*, 2021] I. Khakhiashvili, T. Grinshpoun, and L. Dery. Course allocation with friendships as an asymmetric distributed constraint optimization problem. In *WI-IAT*, pages 688–693, 2021.

[Kiekintveld *et al.*, 2010] C. Kiekintveld, Z. Yin, A. Kumar, and M. Tambe. Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In *AAMAS*, pages 133–140, 2010.

[Kim and Lesser, 2013] Y. Kim and V. Lesser. Improved Max-Sum algorithm for DCOP with n-ary constraints. In *AAMAS*, pages 191–198, 2013.

[Kogan *et al.*, 2023] P. Kogan, T. Tassa, and T. Grinshpoun. Privacy preserving solution of DCOPs by mediation. *Artificial Intelligence*, 319:103916, 2023.

[Krigman *et al.*, 2024] S. Krigman, T. Grinshpoun, and L. Dery. Scheduling of earth observing satellites using distributed constraint optimization. *Journal of Scheduling*, 27:507–524, 2024.

[Kumar *et al.*, 2008] A. Kumar, A. Petcu, and B. Faltings. H-DPOP: Using hard constraints for search space pruning in DCOP. In *AAAI*, pages 325–330, 2008.

[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. *Autonomous Agents and Multi-Agent Systems*, 13:27–60, 2006.

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

[Nguyen *et al.*, 2019] D.T. Nguyen, W. Yeoh, H.C. Lau, and R. Zivan. Distributed gibbs: A linear-space sampling-based DCOP algorithm. *Journal of Artificial Intelligence Research*, 64:705–748, 2019.

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

[Nissim and Zivan, 2005] K. Nissim and R. Zivan. Secure DisCSP protocols - from centralized towards distributed solutions. In *DCR Workshops*, 2005.

[Ottens *et al.*, 2017] B. Ottens, C. Dimitrakakis, and B. Faltings. DUCT: An upper confidence bound approach to distributed constraint optimization problems. *ACM Transactions on Intelligent Systems and Technology*, 8:69, 2017.

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

[Rust *et al.*, 2016] P. Rust, G. Picard, and F. Ramparany. Using message-passing DCOP algorithms to solve energy-efficient smart environment configuration problems. In *IJCAI*, pages 468–474, 2016.

[Savaux *et al.*, 2020] J. Savaux, J. Vion, S. Piechowiak, R. Mandiau, T. Matsui, K. Hirayama, M. Yokoo, S. Elmane, and M. Silaghi. Privacy stochastic games in distributed constraint reasoning. *Annals of Mathematics and Artificial Intelligence*, 88:691–715, 2020.

[Shamir, 1979] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

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

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

[Tassa *et al.*, 2017] T. Tassa, T. Grinshpoun, and R. Zivan. Privacy preserving implementation of the Max-Sum algorithm and its variants. *Journal of Artificial Intelligence Research*, 59:311–349, 2017.

[Tassa *et al.*, 2021] T. Tassa, T. Grinshpoun, and A. Yanai. PC-SyncBB: A privacy preserving collusion secure DCOP algorithm. *Artificial Intelligence*, 297:103501, 2021.

[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.

[Vion *et al.*, 2022] J. Vion, R. Mandiau, S. Piechowiak, and M. Silaghi. Integrating domain and constraint privacy reasoning in the distributed stochastic algorithm with breakouts. *Annals of Mathematics and Artificial Intelligence*, 90:31–73, 2022.

[Yokoo and Hirayama, 2000] M. Yokoo and K. Hirayama. Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems*, 3:185–207, 2000.

[Yokoo *et al.*, 2005] M. Yokoo, K. Suzuki, and K. Hirayama. Secure distributed constraints satisfaction: Reaching agreement without revealing private information. *Artificial Intelligence*, 161:229–246, 2005.

[Zhang *et al.*, 2005] W. Zhang, G. Wang, Z. Xing, and L. Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, 161:55–87, 2005.

[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.