Differentially Private Multi-Agent Constraint Optimization

Sankarshan Damle
sankarshan.damle@research.iiit.ac.in
Machine Learning Lab, IIIT, Hyderabad
Hyderabad, India

Aleksei Triastcyn
Boi Faltings
aleksey.tryastsyn@alumni.epfl.ch
boi.faltings@epfl.ch
Artificial Intelligence Laboratory, EPFL
Lausanne, Switzerland

Sujit Gujar
sujit.gujar@iiit.ac.in
Machine Learning Lab, IIIT, Hyderabad
Hyderabad, India

ABSTRACT

Several optimization scenarios involve multiple agents that desire to protect the privacy of their preferences. There are distributed algorithms for constraint optimization that provide improved privacy protection through secure multiparty computation. However, it comes at the expense of high computational complexity and does not constitute a rigorous privacy guarantee for optimization outcomes, as the result of the computation itself may compromise agents' preferences. In this work, we show how to achieve privacy, specifically differential privacy, through the randomization of the solving process. In particular, we present P-Gibbs, which adapts the SD-Gibbs algorithm to obtain differential privacy guarantees with much higher computational efficiency. Experiments on graph coloring and meeting scheduling show the algorithm's privacy-performance trade-off for varying privacy budgets, and the SD-Gibbs algorithm.

CCS CONCEPTS

• **Computing methodologies** → **Distributed algorithms**; • **Security and privacy**;

KEYWORDS

Distributed Constrained Optimization, Differential Privacy

ACM Reference Format:
Sankarshan Damle, Aleksei Triastcyn, Boi Faltings, and Sujit Gujar. 2021. Differentially Private Multi-Agent Constraint Optimization. In *IEEE/WIC/ACM International Conference on Web Intelligence (WI-IAT '21), December 14–17, 2021, ESSENDON, VIC, Australia.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3486622.3493929

1 INTRODUCTION

One of the most successful applications of distributed computing is *distributed constraint optimization problem* (DCOP), first introduced in [26]. DCOP is a problem where agents collectively compute their value assignments to maximize the sum of resulting *constraint* rewards. In DCOP, constraints quantify the *preference* that each agent places on each of its possible assignments. DCOPs help model various multi-agent coordination and resource allocation problems. E.g., distributed scheduling of meetings and graph-coloring related applications such as mobile radio frequency assignments.

1.1 DCOP Algorithms

Solving a DCOP instance is NP-Hard. Nevertheless, the field has grown steadily over the years, with several algorithms being introduced to solve DCOP instances, each providing some improvement over the previous. These algorithms are either: (1) search-based algorithms like SynchBB [12], and MGM [17], where the agents enumerate through sequences of assignments in a decentralized manner; and (2) inference-based algorithms like DPOP [20], and max-sum [6], where the agents use dynamic programming to propagate aggregated information to other agents.

Ottens et al. [19] propose Distributed Upper Confidence Tree (DUCT), an extension of UCB [7] and UCT [8]. While DUCT outperforms the algorithms above, its per-agent memory requirement is exponential in the number of agents. It prohibits it from scaling up to larger problems.

Nguyen et al. [18] improve upon DUCT through their sampling-based DCOP algorithms: *Sequential Distributed Gibbs* (SD-Gibbs) and *Parallel Distributed Gibbs* (PD-Gibbs). These are distributed extensions of the Gibbs algorithm [15]. Both SD-Gibbs and PD-Gibbs have a linear-space memory requirement, i.e., the memory requirement per agent is linear in the number of agents. The authors show empirically that SD-Gibbs and PD-Gibbs find better solutions than DUCT, run faster, and solve large problems that DUCT fails to solve due to memory limitations. Therefore, in this paper, we focus on SD-Gibbs[1].

1.2 Privacy in DCOPs

The need for preserving the privacy of an agent's sensitive information is vital. This need holds for DCOPs, too, as, in the process of 'solving' a DCOP instance, the transfer of information across agents may leak sensitive information, such as agent's preferences, to the other participating agents. Thus, privacy-preserving solutions to DCOPs are necessary.

1.2.1 Achieving Privacy through Crypto-systems. Privacy in DCOPs has focused on using cryptographic primitives, such as *partial homomorphic encryption*. Several privacy-preserving algorithms exist, which use cryptographic primitives atop existing DCOP algorithms to provide strong privacy guarantees. These include P-DPOP [5],

[1] Our results also follow for PD-Gibbs.

WI-IAT '21, December 14–17, 2021, ESSENDON, VIC, Australia

ACM ISBN 978-1-4503-9115-3/21/12…$15.00
https://doi.org/10.1145/3486622.3493929

$P^{3/2}$-DPOP, P^2-DPOP [14], which build on the DPOP algorithm; P-SyncBB [10] over SynchBB; and P-MaxSum [23] which presents the privacy variant of the max-sum algorithm. However, cryptographic primitives and the computationally expensive nature of DCOPs results in these algorithms not being scalable. For instance, P-MaxSum requires a computational overhead that can range from minutes to an hour. Also, the algorithm's run-time itself increases by a factor of 1000s over its non-private variant [23]. PC-SyncBB [22], which adds collusion resistance to SyncBB, also does not scale.

1.2.2 Other Privacy Notions. In a parallel line of work, the authors in [16] use *information entropy* to quantify the privacy loss incurred by an algorithm in the process of solving a distributed constraint problem. The work is later furthered by [2, 9]. Grinshpoun et al. [9] present private local-search algorithms based on the algorithms above. The authors use this quantification to show that their algorithms provide a high quality of solutions while preserving privacy. While the privacy loss metric defined in [16] is interesting, it does not provide a worst-case guarantee. Practically, even a minor leak may result in information being revealed completely.

In summary, we aim to provide rigorous and provable privacy guarantees for agent constraints. Another serious problem of all exact DCOP algorithms is that the final assignment leaks further information about agent preferences. We adopt *differential privacy* (DP) techniques to avoid such leaks. That is, unlike the existing literature, our privacy variant is immune to post-processing.

1.3 Differential Privacy (DP)

To reiterate, we aim to preserve privacy of agent preferences, i.e., ensuring *constraint privacy* in DCOPs. We employ DP [3] for the same. One may note that when the set of variables and agents involved is *globally* known, there are more efficient techniques for distributed optimization using a central coordinator and stochastic gradient descent. Researchers have developed DP techniques for this context as well [13]. While such algorithms are well-suited for contexts such as federated learning, where the model parameters are common knowledge, in meeting scheduling, they would leak the information of who is meeting with whom, which is usually the most sensitive information. Therefore, we focus on algorithms where each participant has *local* information, i.e., only knows information about agents it shares constraints with and nothing about the rest of the problem. In particular, we focus on achieving privacy in SD-Gibbs using DP techniques. Furthermore, we consider a *stronger* local model of privacy [4], which ensures the indistinguishability of any two agents.

1.4 Contributions

We show that SD-Gibbs may leak information about agent constraints during its execution. Its iterative nature may further lead to a high privacy loss over the iterations. As such, we must construct a scalable DCOP algorithm that preserves the privacy of the constraints without requiring a centralized authority.

Towards this, we develop a new differentially private variant of SD-Gibbs. We present a novel algorithm *P-Gibbs*: which uses *soft-max with temperature* to smooth sampling distributions in SD-Gibbs. Additionally, during computation, we add *Gaussian* noise to the relative utility in our algorithm. We then provide a refined privacy analysis within the framework of (ϵ, δ)-DP. Our experiments demonstrate our algorithm's practicality and robust performance for a reasonable *privacy budget*, i.e., ϵ, with SD-Gibbs as the baseline.

2 PRELIMINARIES

Distributed Constraint Optimization Problem (DCOP) is a class of problems comprising a set of variables, a set of agents owning them, and a set of constraints defined over the set of variables. These constraints reflect each agent's *preferences*.

Definition 1 (DCOP). *A Distributed Constraint Optimization Problem (DCOP) is a tuple $\langle \mathcal{X}, \mathcal{A}, \mathcal{D}, \mathcal{F}, \alpha \rangle$ wherein,*

- $\mathcal{X} = \{x_1, \ldots, x_p\}$ *is a set of variables;*
- $\mathcal{A} = \{1, \ldots, m\}$ *is a set of agents;*
- $\mathcal{D} = D_1 \times \ldots \times D_p$ *is a set of finite domains such that D_i is the domain of x_i;*
- $\mathcal{F}$ *is a set of utility functions $F_{ij} : D_i \times D_j \to \mathbb{R}$. F_{ij} gives the utility of each combination of values of variables in its scope. Let $var(F_{ij})$ denote the variables in the scope of F_{ij}.*
- $\alpha : \mathcal{X} \to \mathcal{A}$ *maps each variable to one agent.*

In this work, w.l.o.g [25], we assume that $p = m$, i.e., the number of agents and the number of variables are equal. Also, $D = D_i = D_j,\ \forall i, j$, i.e., all variables have the same domain. Total utility in DCOP, for a complete assignment $\mathbf{X} = (x_1, \ldots, x_p)$ is:

$$F(\mathbf{X}) \triangleq \sum_{i=1}^{m} \left(\sum_{j} F_{ij}(\mathbf{X}_{||D}) \right), \tag{1}$$

where $\mathbf{X}_{||D}$ is the projection of $\mathbf{X}$ to the subspace on which F_{ij} is defined. The *objective* of a DCOP is to find an assignment $\mathbf{X}^*$ that maximizes the total utility, i.e., $F(\mathbf{X}^*) = \max_{\mathbf{X} \in \mathcal{D}} F(\mathbf{X})$.

In DCOP, each combination of variables/agents is referred to as a *constraint*. The utility functions over these constraints quantify how much each agent *prefers* a particular constraint. This constraint structure is captured through a *constraint graph*.

Definition 2 (Constraint Graph (CG)). *Given a DCOP defined by $\langle \mathcal{X}, \mathcal{A}, \mathcal{D}, \mathcal{F}, \alpha \rangle$, its constraint graph $\mathcal{G} = \langle \mathcal{X}, \mathcal{E} \rangle$ is such that $(x_i, x_j) \in \mathcal{E},\ \forall j \in var(F_{ij})$.*

A *pseudo-tree* arrangement has the same nodes and edges as the constraint graph. The tree satisfies (i) there is a subset of edges, called *tree edges*, that form a rooted tree; and (ii) two variables in a utility function appear in the same branch of that tree. Such an arrangement can be constructed using a distributed-DFS [11].

For the algorithms presented in this paper, let N_i refer to the set of neighbors of x_i in CG. Also, let C_i denote the set of children x_i in the pseudo-tree, P_i as the parent of variable x_i, and PP_i as the set of pseudo-parents of x_i.

2.1 Sequential Distributed Gibbs (SD-Gibbs)

We now describe Sequential Distributed Gibbs (SD-Gibbs) as first introduced in [18]. In this, the authors map DCOP to a *maximum a posteriori* (MAP) estimation problem. Consider MAP on a *Markov Random Field* (MRF). MRF consists of a set of random variables represented by *nodes*, and a set of *potential functions*. Each potential

Variables	Definition
d_i and $\hat{d}_i$	Values in current and previous iteration
d_i^*	Value in the best complete solution so far
$\bar{d}_i$	Best response value
C_i and $\bar{C}_i$	Context and best-response context
t_i, t_i^*, $\bar{t}_i^*$	Time index, best-response and non-best response index
Δ_i	Difference in current and previous local solution of agent i
$\bar{\Delta}_i$	Difference in current best-response solution with previous
Ω	Shifted utility of the current complete solution
$\bar{\Omega}$	Shifted utility of the best-response solution
Ω^*	Shifted utility of the best complete solution

Table 1: Variables maintained by each agent x_i in SD-Gibbs

Algorithm 1: Sequential Distributed Gibbs [18]

1 Create pseudo-tree
2 Each agent x_i calls INITIALIZE()

Procedure 1: INITIALIZE() [18]

1 $d_i \leftarrow \hat{d}_i \leftarrow d_i^* \leftarrow \bar{d}_i \leftarrow \text{ValInit}(x_i)$
2 $C_i \leftarrow \bar{C}_i \leftarrow \{(x_j, \text{ValInit}(x_j)) | x_j \in N_i\}$
3 $t_i \leftarrow t_i^* \leftarrow \bar{t}_i^* \leftarrow 0$
4 $\Delta_i \leftarrow \bar{\Delta}_i \leftarrow 0$
5 **if** x_i *is root* **then**
6 $\quad t_i \leftarrow t_i^* \leftarrow \bar{t}_i^* \leftarrow 0$
7 $\quad$ SAMPLE()
8 **end**

function, represented by $\theta_{ij}(x_i; x_j)$, is associated with an edge. We denote the nodes and edges of the graph constituting MRF by $\langle V, E \rangle$.

Let $Pr(x_i = d_i; x_j = d_j)$ be defined as $\exp(\theta_{ij}(x_i = d_i; x_j = d_j))$. Then, the most probable assignment is:

$$\Pr(\mathbf{X}) = \frac{1}{Z} \prod_{i,j \in E} e^{\theta_{ij}(x_i, x_j)} = \frac{1}{Z} \exp\left[\sum_{i,j \in E} \theta_{ij}(x_i, x_j)\right].$$

Here, Z is the normalization factor. This corresponds to the maximum solution of DCOP if,

$$F(\mathbf{X}) = \sum_{i,j \in E} \theta_{ij}(x_i, x_j).$$

2.1.1 Sampling. We now describe *sampling* in SD-Gibbs. Let C_i denote agent i's *context*, defined as the set consisting of its neighbors and the value assigned to them. In each iteration, each agent i samples a value d_i with the following equation,

$$\Pr(x_i | x_j \in N_i) = \frac{1}{Z} \exp\left[\sum_{\langle x_j, d_j \rangle \in C_i} F_{ij}(d_i, d_j)\right] \quad (2)$$

Let, $\mathbb{P}_i(\mathbf{x}_i) = \{\Pr(x_i | x_j \in \mathcal{X} \setminus \{x_i\}) | x_i = d_i \ \forall d_i \in D_i\}$. That is, $\mathbb{P}_i$ represents the SD-Gibbs's probability distribution of each agent i. The relevant notations required for the SD-Gibbs algorithm are presented in Table 1.

Procedure 2: SAMPLE() [18]

1 $t_i \leftarrow t_i + 1$; $\hat{d}_i \leftarrow d_i$
2 $d_i \leftarrow$ Sample based on (2)
3 $\bar{d}_i \leftarrow \text{argmax}_{d_i' \in D_i} \sum_{\langle x_j, \bar{d}_j \rangle \in \bar{C}_i} F_{ij}(d_i', \bar{d}_j)$
4 $\Delta_i \leftarrow \sum_{\langle x_j, d_j \rangle \in C_i} \left[F_{ij}(d_i, d_j) - F_{ij}(\hat{d}_i, d_j)\right]$
5 $\bar{\Delta}_i \leftarrow \sum_{\langle x_j, \bar{d}_j \rangle \in C_i} \left[F_{ij}(\bar{d}_i, \bar{d}_j) - F_{ij}(\hat{d}_i, \bar{d}_j)\right]$
6 Send VALUE(x_i, d_i, $\bar{d}_i$, t_i^*, $\bar{t}_i^*$) to each $x_j \in N_i$

Procedure 3: VALUE($x_s, d_s, \bar{d}_s, t_s^*, \bar{t}_s^*$) [18]

1 Update $\langle x_s, d_s' \in C_i \rangle$ with (x_s, d_s)
2 **if** $x_s \in PP_i \cup \{P_i\}$ **then**
3 $\quad$ Update $\langle x_s, d_s' \in \bar{C}_i \rangle$ with $(x_s, \bar{d}_s)$
4 **else**
5 $\quad$ Update $\langle x_s, d_s' \in C_i \rangle$ with $(x_s, \bar{d}_s)$
6 **end**
7 **if** $x_s = P_i$ **then**
8 $\quad$ **if** $\bar{t}_s^* \geq t_s^*$ **and** $\bar{t}_s^* > max\{t_i^*, \bar{t}_i^*\}$ **then**
9 $\quad\quad d_i^* \leftarrow \bar{d}_i; \bar{t}_i^* \leftarrow \bar{t}_s^*$
10 $\quad$ **else if** $t_s^* \geq \bar{t}_s^*$ **and** $\bar{t}_s^* > max\{t_i^*, \bar{t}_i^*\}$ **then**
11 $\quad\quad d_i^* \leftarrow \bar{d}_i; t_i^* \leftarrow t_s^*$
12 $\quad$ **end**
13 $\quad$ SAMPLE()
14 $\quad$ **if** x_i *is a leaf* **then**
15 $\quad\quad$ Send BACKTRACK(x_i, Δ_i, $\bar{\Delta}_i$) to P_i
16 $\quad$ **end**
17 **end**

Procedure 4: BACKTRACK($x_s, \Delta_s, \bar{\Delta}_s$) [18]

1 $\Delta_i \leftarrow \Delta_i + \Delta_s$; $\bar{\Delta}_i \leftarrow \bar{\Delta}_i + \bar{\Delta}_s$
2 **if** Received BACKTRACK from all children in this iteration **then**
3 $\quad$ Send BACKTRACK(x_i, Δ_i, $\bar{\Delta}_i$) to P_i
4 $\quad$ **if** x_i *is root* **then**
5 $\quad\quad \bar{\Omega} \leftarrow \Omega + \bar{\Delta}_i; \Omega \leftarrow \Omega + \Delta_i$
6 $\quad\quad$ **if** $\Omega \geq \bar{\Omega}$ ***and*** $\Omega > \Omega^*$ **then**
7 $\quad\quad\quad \Omega^* \leftarrow \Omega; d_i^* \leftarrow d_i; t_i^* \leftarrow t_i$
8 $\quad\quad$ **else if** $\bar{\Omega} \geq \Omega$ ***and*** $\bar{\Omega} > \Omega^*$ **then**
9 $\quad\quad\quad \Omega^* \leftarrow \bar{\Omega}; d_i^* \leftarrow \bar{d}_i; \bar{t}_i^* \leftarrow t_i$
10 $\quad\quad$ **end**
11 $\quad\quad$ SAMPLE()
12 $\quad$ **end**
13 **end**

2.1.2 Algorithm. Table 1 presents the values each agent i maintains in SD-Gibbs. Procedure 2 describes the complete sampling function. For completeness, we present the SD-Gibbs algorithm in Algorithm 1. The algorithm can be summarized as follows:

(1) The algorithm starts with the construction of the pseudo-tree. Each agent then initializes its variables, from Table 1 to their default values. The root then starts the sampling, as described in Procedure 2 and sends the VALUE message (line 6) to each of its neighbors.
(2) Upon receiving a VALUE message, each agent invokes Procedure 3. In it, an agent i first updates its current contexts, C_i

and $\bar{C}_i$ with the sender's values. If the message is from agent i's parents, then the agent itself samples, i.e., executes Procedure 2. This *sampling stage* continues until all the leaf agents have sampled.

(3) Each leaf agent j then sends a BACKTRACK message to its parent comprising $x_j, \Delta_j,$ and $\bar{\Delta}_j$. As described in Procedure 4, when a parent receives such a message, it too sends a BACKTRACK message to its parent. The process continues until the root receives the message – concluding one iteration.

(4) To reach a solution, each agent i uses its current (Δ_i) and current best-response ($\bar{\Delta}_i$) local utility differences. We refer to these differences as *relative* utilities. Upon receiving a BACKTRACK message, agent i adds the delta variables of its children to its own. Consequently, these variables for the root agent quantify the relative global utility. Based on this, at the end of an iteration, the root decides to keep or throw away the current solution (Procedure 4, line 4).

As aforementioned, in this work, we focus on *constraint privacy* to ensure the privacy of agent preferences. From Faltings et al. [5], constraint privacy states that no agent must be able to discover the nature of constraint that does not involve a variable it owns. Since absolute privacy is not an achievable goal [3], we formalise constraint privacy in terms of (ϵ, δ)-DP [4].

2.2 Differential Privacy (DP)

Differential Privacy (DP) is normally defined for *adjacent* databases, i.e., databases differing in a single entry. However, in this instance, we not only want to protect privacy against external adversaries but also against curious fellow agents, i.e., agents looking to decipher sensitive information. To do so, we consider the *local* model of privacy [4]. It is defined on individual entries rather than databases, or in our setting, on individual agents. As a result, local-DP does not require defining adjacency. Formally, we want our algorithm for any two utility functions (vectors in $\mathbb{R}^p$) to satisfy the following definition, from [4],

Definition 3 (Local Differential Privacy). *A randomized mechanism $\mathcal{M} : \mathcal{F} \to \mathcal{R}$ with domain $\mathcal{F}$ and range $\mathcal{R}$ satisfies (ϵ, δ)-DP if for any two inputs $F, F' \in \mathcal{F}$ and for any subset of outputs $O \subseteq \mathcal{R}$ we have,*

$$\Pr[\mathcal{M}(F) \in O] \leq e^{\epsilon} \Pr[\mathcal{M}(F') \in O] + \delta \tag{3}$$

Privacy loss, useful for our analysis of DP, is defined as

$$L^{o}_{\mathcal{M}(F)||\mathcal{M}(F')} = \ln\left(\frac{\Pr[\mathcal{M}(F) = o]}{\Pr[\mathcal{M}(F') = o]}\right) \tag{4}$$

2.2.1 Privacy Leakage in SD-Gibbs. In SD-Gibbs, constraint privacy is compromised in the following two ways:

(1) *By sampling.* Each variable value in SD-Gibbs is sampled according to agent i's utility F_{ij}. As values with *more utility are more likely to be drawn*, SD-Gibbs leaks sensitive information about these utility functions. Fortunately, this stage can be secured by simply making distributions more similar across agents (Section 4.2).

(2) *By relative utility* Δ. Every leaf agent j in the pseudo-tree sends its Δ_j and $\bar{\Delta}_j$ to its parent i. The parent agent adds the values to its Δ_i and $\bar{\Delta}_i$, respectively, and passes them on up the tree. The process continues until the values reach the root. Thus, any intermediate agents, or an adversary observing Δ, can learn something about j's utility even if sampling is private. E.g., suppose a particular assignment has a high utility for agent j but low for others (and it is known). In that case, an intermediate agent will learn about agent j even from the aggregated utility.

These privacy leaks follow by observing what critical information gets transferred by each agent i in Algorithm 1. We ignore t^* and $\bar{t}^*$ because these are simply functions of utility, i.e., will be private by post-processing property once the utility is private.

Sensitivity. In order to achieve DP, particularly for Δ's, we need to bound its *sensitivity*. Sensitivity is defined as the maximum possible change in the output of a function we seek to make privacy-preserving. Formally,

Definition 4 (Sensitivity (τ)). *It is the maximum absolute difference between any two relative utility values Δ and Δ', i.e.,*

$$\tau = \max_{\Delta, \Delta'} \left|\Delta - \Delta'\right| \tag{5}$$

3 OUR APPROACH AND PAPER OVERVIEW

In a nutshell, we aim to ensure constraint privacy in DCOP using DP techniques. Firstly, observe that SD-Gibbs in its current form is non-private. This is because the probability distributions defined in (2) may not be bounded. As a result, ϵ in (3) tends to ∞.

To provide meaningful privacy guarantees for constraint privacy in DCOPs, we present P-Gibbs (Section 4). We first use soft-max with temperature to bound the SD-Gibbs distributions (Section 4.2). The resulting bound only depends on the temperature parameter and does not leak any agent's sensitive information. Then, we "clip" the relative utilities to further bound the sensitivity (Section 4.3). Lastly, to reduce the growth of ϵ, we randomly select a subset of agents to sample new values at each iteration. We then provide a refined privacy analysis for the resulting (ϵ, δ)-DP (Theorem 1).

We validate P-Gibbs empirically over several problem instances of benchmark problems in DCOP literature (Section 5). Our experiments highlight our privacy variant's efficiency. Specifically, we show P-Gibbs provides only a tiny drop in solution qualities than SD-Gibbs for a desirable privacy budget, i.e., ϵ.

With these as a backdrop, we now build upon SD-Gibbs to formally present our novel, scalable algorithm for DCOPs that preserve constraint privacy, namely P-Gibbs.

4 P-GIBBS: PRESERVING CONSTRAINT PRIVACY IN DCOP WITH SD-GIBBS

First, typically for DP, we need to ensure full support of the outcome distribution. Indeed, if $\Pr[\mathcal{M}(D') = o] = 0$ for some o, the privacy loss incurred is infinite and one cannot bound ϵ. It implies that all agents must have the same domain for their variables and non-zero utility for each value within the domain.[2] In other words, $D_1 = D_2 = \ldots = D_p$ and $\left|F_{ij}(\cdot, \cdot)\right| > 0, \forall i$.

[2] If an agent has a zero utility for some value, then all agents must have zero utility, and w.l.o.g., we can exclude such values from all domains.

4.1 P-Gibbs

The novelty in P-Gibbs, when compared to SD-Gibbs, is in the sampling procedure. We formally provide the sampling in P-Gibbs with Procedure 5. The differences, compared to SD-Gibbs, are summarized as follows:

(1) To preserve constraint privacy loss due to sampling:
   - P-Gibbs uses soft-max function over SD-Gibbs distributions for sampling d_i's, $\forall i$. As shown later in Claim 1, this bounds any two agent distributions in SD-Gibbs, resulting in finite privacy loss.
   - P-Gibbs randomly chooses subsets of agents to sample new values in each iteration. More specifically, in every iteration, each agent i samples a new value d_i with probability q or uses previous values with probability $1-q$.

(2) To preserve constraint privacy loss due to relative utilities:
   - In P-Gibbs, we sanitize the relative utilities with calibrated *Gaussian Noise*.
   - To further bound the sensitivity, we "clip" the relative utilities by $\pm c$, where c is the *clipping constant* (Procedure 5, Lines 13 and 14). This trivially implies, from (5), that $\tau = 2 \cdot c$.

In the following subsection, we formally show that soft-max bounds the SD-Gibbs probability distributions. We then provide a formal analysis for privacy loss due to sampling.

4.2 Bounding Sampling Divergence with Soft-max

Towards achieving bounded sampling divergence without compromising on constraint privacy itself, we propose to apply *soft-max* to sampling distributions. Let p_i be the soft-max distribution with *temperature* parameter as γ, i.e.,

$$p_i(\mathbf{x}_i, \gamma) = \left\{ \frac{\exp(\mathbb{P}_i(x_i = d_k)/\gamma)}{\sum_{d_l \in D} \exp(\mathbb{P}_i(x_i = d_l)/\gamma)}; \forall d_k \in D \right\} \tag{6}$$

Firstly, observe that $p_i(\cdot, \gamma)$, for a finite γ, has full support of the outcome determination. That is, $p_i(x_i, \gamma) > 0$ s.t. $x_i = d_k, \forall d_k \in D$. Secondly, to also ensure that ϵ is finite, we require that the bound $\frac{p_i(\cdot)}{p_j(\cdot)}$ for any distinct pair i and j is bounded. To this end, the following claim shows that the ratio of the resulting soft-max probabilities, $p_i(\cdot)$ and $p_j(\cdot)$ for any two agents i and j, is *bounded* by $2/\gamma$. The proof uses the fact that $D = D_i = D_j$ and $1/e \leq \exp(p_i(x) - p_j(x)) \leq e$.

Claim 1. *For two probability distributions using soft-max, p_i and p_j defined by* (6)*, we have, $\forall i, j, \forall d \in D$ and $\forall D$, s.t. $|D| > 1, \gamma \geq 1$*

$$\ln \left[\frac{p_i(x_i = d, \gamma)}{p_j(x_j = d, \gamma)} \right] \leq \frac{2}{\gamma}$$

Proof. (SKETCH) We have,

$$\max \left(\ln \left[\frac{p_i}{p_j} \right] \right) = \max \left(\ln \left[\frac{\frac{\exp(\mathbb{P}_i(x_k)/\gamma)}{\sum_{x_l \in D} \exp(\mathbb{P}_i(x_l)/\gamma)}}{\frac{\exp(\mathbb{P}_j(x_k)/\gamma)}{\sum_{x_l \in D} \exp(\mathbb{P}_j(x_l)/\gamma)}} \right] \right)$$

$$= \max \left(\ln \left[\frac{\exp(1/\gamma(\mathbb{P}_i - \mathbb{P}_j)))}{N_1/N_2} \right] \right).$$

Procedure 5: P-Gibbs SAMPLE()

1 $t_i \leftarrow t_i + 1$; $\hat{d}_i \leftarrow d_i$
2 $\beta \sim$ Uniform$(0, 1)$
  // Sub-sampling
3 **if** $\beta \in (0, q]$ **then**
4 $\mathbb{P}_i(\mathbf{x}_i) \leftarrow$ from (2)
  // Bounding SD-Gibbs distribution with Soft-max
5 $p_i(\mathbf{x}_i, \gamma) \leftarrow$ from (6)
6 $d_i \leftarrow$ Sample based on $p_i(\mathbf{x}_i, \gamma)$
7 **else**
8 $d_i \leftarrow d_i$
9 **end**
10 $\bar{d}_i \leftarrow \text{argmax}_{d'_i \in D_i} \sum_{\langle x_j, \bar{d}_j \rangle \in \bar{C}_i} F_{ij}(d'_i, \bar{d}_j)$
11 $\Delta_i \leftarrow \sum_{\langle x_j, d_j \rangle \in C_i} \left[F_{ij}(d_i, d_j) - F_{ij}(\hat{d}_i, d_j) \right]$
12 $\bar{\Delta}_i \leftarrow \sum_{\langle x_j, \bar{d}_j \rangle \in \bar{C}_i} \left[F_{ij}(\bar{d}_i, \bar{d}_j) - F_{ij}(\hat{d}_i, \bar{d}_j) \right]$
  // Clipping
13 **if** $|\Delta_i| > c$ **then** $\Delta_i = (\Delta_i \geq 0)\ ?\ c : -c$
14 **if** $|\bar{\Delta}_i| > c$ **then** $\bar{\Delta}_i = (\bar{\Delta}_i \geq 0)\ ?\ c : -c$
  // Perturbing utilities with Gaussian noise
15 $\Delta_i \leftarrow \Delta_i + \mathcal{N}(0, \tau^2\sigma^2)$
16 $\bar{\Delta}_i \leftarrow \bar{\Delta}_i + \mathcal{N}(0, \tau^2\sigma^2)$
17 Send VALUE$(x_i, d_i, \bar{d}_i, t_i^*, \bar{t}_i^*)$ to each $x_j \in N_i$

Here, $N_1 = \sum_{x_l \in D} \exp(\mathbb{P}_i(x_l)/\gamma)$ and $N_2 = \sum_{x_l \in D} \exp(\mathbb{P}_j(x_l)/\gamma)$. The claim follows by observing that $N_1/N_2 \leq 1/e^{1/\gamma}$ and the maximum value of the numerator is $e^{1/\gamma}$. □

Discussion.

- *Effect of Soft-max.* We illustrate the effect of soft-max on the SD-Gibbs sampling distribution with the following example. Let $D_j = \{d_1, d_2, d_3\}, \forall j$ such that $\mathbb{P}_i = [0.8, 0.15, 0.05]$. Observe that the distribution is such that the probability of sampling d_1 is significantly more than others. Now, the corresponding soft-max distributions, from (6), will be: $p(\cdot, \gamma = 1) = [0.50, 0.26, 0.24]$, $p(\cdot, \gamma = 2) = [0.41, 0.30, 0.29]$, and $p(\cdot, \gamma = 10) = [0.35, 0.33, 0.32]$. That is, the soft-max distribution is more *uniform* than the original distribution. This implies that the maximum ratio of the probabilities will be smaller. That is, an adversary will be more indifferent towards the domain values while sampling. For e.g., d_1 and d_2 in $p(\cdot, \gamma = 10)$ compared to in $p(\cdot, \gamma = 1)$.
- Observe that the bound provided in Claim 1 *does not* depend on an agent's sensitive information. This implies that the bound does not encode (and reveal) any sensitive information. Thus, we conclude that the bound provided in Claim 1 is desirable; and hence use it to construct the sampling distribution in P-Gibbs.

4.2.1 Privacy Guarantees for Sampling in P-Gibbs. We first calculate the privacy parameters of the sampling stage, denoted by ϵ_s and δ, in P-Gibbs. We use an extension of the moments accountant method [1] for non-Gaussian mechanisms. Following derivations

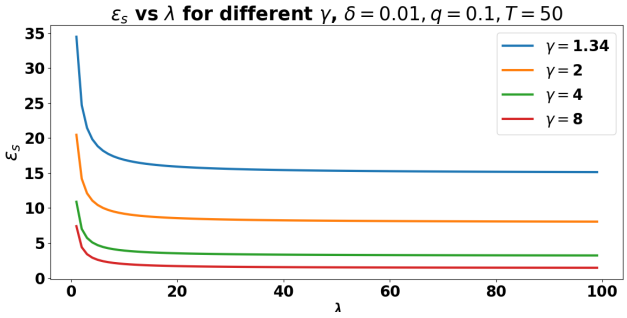

Figure 1: Variation of ϵ_s with λ

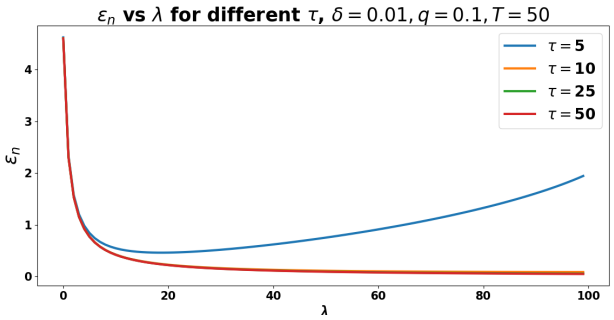

Figure 2: Variation of ϵ_n with λ

by [24],

$$\Pr[L \geq \epsilon_s] \leq \max_{F, F'} e^{\lambda \mathcal{D}_{\lambda+1}[\mathcal{M}(F)||\mathcal{M}(F')] - \lambda\epsilon_s}. \tag{7}$$

Here, L is the privacy loss between any two agents and $\mathcal{D}_\lambda(\cdot||\cdot)$ is Renyi divergence of order $\lambda \in \mathbb{N}$ with a slight abuse of notation (using $\mathcal{M}(\cdot)$ instead of a distribution imposed by it). Unlike [24], we consider the classical DP. Using their notion of Bayesian DP could improve the bounds, but we leave it for future work.

Also from [24], we borrow the notion of *privacy cost* $c_t(\lambda)$. By trivial manipulation, for each iteration t,

$$c_t(\lambda) = \max_{i,j} \lambda \mathcal{D}_{\lambda+1}\left[p_i(d)||p_j(d)\right] \leq \lambda 2/\gamma, \tag{8}$$

where (8) is due to monotonicity $\mathcal{D}_\lambda(P||Q) \leq \mathcal{D}_{\lambda+1}(P||Q) \leq \mathcal{D}_\infty(P||Q)$, $\forall \lambda \geq 0$. Importantly, this cost can be further reduced by subsampling agents with probability $q << 1$, as we outline next.

Reproducing the steps of the sampled Gaussian mechanism analysis by [24] for our mechanism and classical DP, we formulate the following result.

Theorem 1. *Privacy cost $c_t(\lambda)$ at iteration t of a sampling stage of P-Gibbs, with agent subsampling probability q, is*

$$c_t^{(s)}(\lambda) = \ln \mathbb{E}_{k \sim B(\lambda+1, q)}\left[e^{k2/\gamma}\right], \tag{9}$$

where $B(\lambda, q)$ is the binomial distribution with λ experiments and probability of success as q, $\lambda \in \mathbb{N}$.

Proof. The result follows by substituting $2/\gamma$ in place of the ratio of normality distributions in [24, Theorem 3]. □

Unlike the analysis in [24, Theorem 3], we do not have $c_t^L(\lambda)$ and $c_t^R(\lambda)$, as well as expectation over the data. This is because we compute the conventional differential privacy bounds, instead of Bayesian DP, and thus, directly use the worst-case ratio, i.e., $2/\gamma$. Finally, merging the results, we can compute ϵ_s, δ across multiple iterations as

$$\left.\begin{aligned} \ln \delta &\leq \textstyle\sum_{t=1}^T c_t^{(s)}(\lambda) - \lambda\epsilon_s \\ \epsilon_s &\leq \tfrac{1}{\lambda}\left(\textstyle\sum_{t=1}^T c_t^{(s)}(\lambda) - \ln \delta\right) \end{aligned}\right\} \tag{10}$$

Figure 1 shows the variation of ϵ_s for different values of λ and γ, with the sampling probability $q = 0.1$. We observe that λ has a clear effect on the final ϵ_s value, and one should ideally minimize the bound over λ.

4.2.2 P-Gibbs$_\infty$: An Extreme Case. We presented P-Gibbs, which uses a soft-max with temperature function to bound the sampling divergence, thereby bounding the privacy loss incurred by sampling. We smooth the distribution using soft-max's temperature parameter to reduce further the information encoded in SD-Gibbs sampling. We then use Theorem 1 to quantify privacy parameters ϵ_s and δ.

From Claim 1, observe that the temperature parameter in P-Gibbs may be tuned to decrease the overall privacy budget for sampling, i.e., ϵ_s. An "extreme" case occurs when $\gamma \to \infty$. For this, we have $p_i = p_j$, which implies that $\epsilon_s \to 0$. Thus, increasing γ leads to P-Gibbs sampling distribution mimicking an *uniform* distribution, as more information of SD-Gibbs sampling distribution is lost. To distinguish this extreme case, we refer to P-Gibbs with $\gamma \to \infty$ as P-Gibbs$_\infty$.

4.3 Privacy of Relative Utilities (Δ) in P-Gibbs

In the previous subsection, we deal with the privacy loss occurring due to sampling in P-Gibbs. As aforementioned, the values Δ and $\bar{\Delta}$ also leak information about agents' constraints. We must sanitize these values so as to fully preserve privacy. We achieve this through the *Gaussian noise mechanism* [4] defined as

$$\mathcal{M}_G(\Delta) \triangleq \Delta + Y_i,$$

where $Y_i \sim \mathcal{N}(0, \tau^2\sigma^2)$, τ is the sensitivity and σ is the noise parameter.

Privacy parameters for the relative utility Δ, denoted by ϵ_n and δ, can be computed either using the basic composition along with [4, Theorem A.1] or the moments accountant [1]. The latter can be unified with the accounting for the sampling stage by using:

$$c_t^{(n)}(\lambda) = \ln \mathbb{E}_{k \sim B(\lambda+1, q)}\left[e^{k\mathcal{D}_{\lambda+1}[\mathcal{N}(0,\tau^2\sigma^2)||\mathcal{N}(\tau,\tau^2\sigma^2)]}\right]. \tag{11}$$

Figure 2 shows the variation of ϵ_n for different values of λ and τ, with the sampling probability $q = 0.1$ and $\sigma = 1$. We observe that λ has a clear effect on the final ϵ_n value as well, although the change is virtually the same for $\tau = 10, 25$ and 50. The trend is similar to the one observed in Figure 1, i.e., ϵ_n decreases as λ increases. However, the decrease is not smooth when $\tau = 5$, which sees a sharp change in ϵ_n as λ increases. This change is similar to what is observed in [24, Figure 5], suggesting that one should be careful while deciding on the value of λ.

Note. We provide the formal sampling procedure comprising the privacy techniques discussed above with Procedure 5. The rest of

Algorithm	(ϵ_s, δ)	(ϵ_n, δ)	$(\epsilon = \epsilon_s + \epsilon_n, \delta)$ **for** T **iterations**
P-Gibbs	$(2/\gamma, 0)$	$(\frac{\tau}{\sigma}\sqrt{2\ln\frac{1.25}{\delta}}, \delta)$	$\left(\frac{T}{\lambda}c_t^{(s)}(\lambda) + \frac{T}{\lambda}c_t^{(n)}(\lambda) - \frac{1}{\lambda}\ln\delta,\ \delta\right)$
P-Gibbs$_\infty$	$(0, 0)$	$(\frac{\tau}{\sigma}\sqrt{2\ln\frac{1.25}{\delta}}, \delta)$	$\left(\frac{T}{\lambda}c_t^{(n)}(\lambda) - \frac{1}{\lambda}\ln\delta,\ \delta\right)$

Table 2: Per-iteration and final (ϵ, δ) bounds.

the procedures are the same as provided with Algorithm 1. Table 2 summarises expressions for per-iteration and total ϵ values for P-Gibbs and P-Gibbs$_\infty$.

5 EXPERIMENTS

We now empirically evaluate the performance of our novel algorithms, P-Gibbs w.r.t. to SD-Gibbs.

Setup. *pyDCOP* [21] is a *Python* module that provides implementations of many DCOP algorithms (DSA, MGM, MaxSum, DPOP, etc.). It also allows easy implementation of one's DCOP algorithm by providing all the required infrastructure: agents, messaging system, metrics collection, etc. We use pyDCOP's public implementation of the SD-Gibbs algorithm to run our experiments. In addition, we also implement P-Gibbs.

Generating Test-cases. pyDCOP allows for generating random test-cases for various problems through its command line's *generate* option. We generate graph-coloring and meeting scheduling problem instances. These are benchmark problems in DCOP literature. We test the performance of our algorithms across 20 such randomly generated problems.

Method. We consider the utility given by SD-Gibbs' solution as our baseline. Further, these algorithms, i.e., SD-Gibbs and P-Gibbs, are random algorithms. Hence, we run each benchmark problem instance 25 times for a fair comparison and use the subsequent average utility for our results.

(ϵ, δ)**-bounds**. Throughout our experiments, we choose $\delta = 10^{-2}$, $T = 50$ and λs as 100. As standard, our choice of δ is such that $\delta < 1/m$. We calculate ϵ using different permutations of $\gamma \in \{8, 20, \infty\}$, $q \in \{0.1, 0.2, 0.3\}$, and $\sigma \in \{1, 5, 25, 1000\}$. We sample τ from $\{5, 10, 25, 50\}$ to get $\epsilon \in \{0.045, 0.655, 1.312, 2.03, 4.09, 7.18\}$. Note that, the case with $\epsilon = 0.045$ corresponds to P-Gibbs$_\infty$.

Solution Quality (SQ).

Definition 5 (Solution Quality (SQ)). *Solution quality* $\text{SQ}_{\mathcal{A}}$ *of an algorithm* $\mathcal{A}$ *is defined as*

$$\text{SQ}_{\mathcal{A}} = \begin{cases} \frac{U_S}{U_{\mathcal{A}}} & \textit{for minimization} \\ \frac{U_{\mathcal{A}}}{U_S} & \textit{for maximization} \end{cases}$$

for utility of $\mathcal{A}$ *as* $U_{\mathcal{A}}$ *and SD-Gibbs as* U_S.

With SQ, we normalize P-Gibbs' utility in the context of SD-Gibbs. SQ ≈ 1 indicates that utility does not deteriorate than SD-Gibbs. On the other hand, SQ ≈ 0 means little utility as compared to the SD-Gibbs solution. It is possible that SQ > 1 due to randomness and privacy noise acting as simulated annealing.

ϵ	SQ (mean ± std)	
	GC Benchmark	MS Benchmark
0.045	0.854 ± 0.0314	0.875 ± 0.0866
0.65	0.873 ± 0.0224	0.933 ± 0.0490
1.312	0.879 ± 0.0231	0.942 ± 0.0418
2.03	0.887 ± 0.0220	0.971 ± 0.0234
4.09	0.901 ± 0.0165	0.982 ± 0.0142
7.18	0.907 ± 0.0192	0.986 ± 0.0137

Table 3: P-Gibbs: SQs for GC and MS Benchmarks

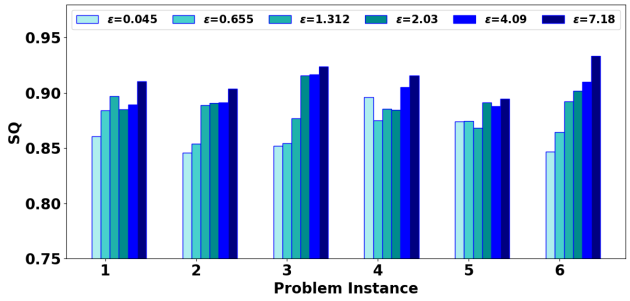

Figure 3: Problem-wise SQs for Graph-coloring

5.1 Benchmark Problems

Graph-Coloring (GC). We generate 20 sample graph-coloring problems. The problems are such that the number of agents/variables lies between $[30, 75)$ and agents' domain size between $[10, 20)$. Each constraint is a random integer taken from $(0, 10)$. Graph-coloring is a *minimization* problem.

Meeting-Scheduling (MS). We generate 20 sample meeting scheduling problems. The problems are that the number of agents and variables lie between $[1, 75)$ with the number of slots, i.e., domain for each agent randomly chosen from $[50, 100)$. Each constraint is a random integer taken from $(0, 100)$, while each meeting may randomly occupy $[1, 5]$ slots. Meeting-scheduling is a *maximization* problem.

Importantly, we perform our experiments on much larger problems than earlier complete algorithms (e.g., [5]) can handle.

5.2 Results

Table 3 and Figures 3 and 4 present our experimental results. They provide (i) SQ scores averaged across all problems and (ii) problem-wise SQ for P-Gibbs. We only plot 6 problem instances for each

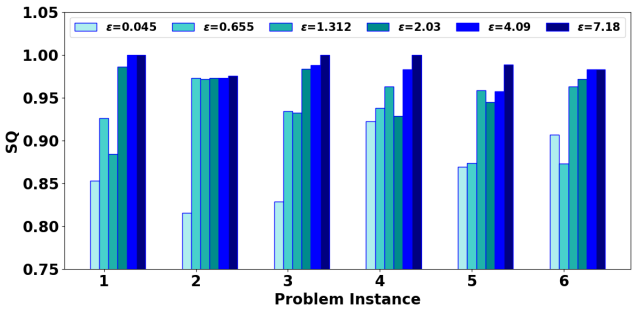

Figure 4: Problem-wise SQs for Meeting-scheduling

benchmark for readability out of the 20 generated. For both benchmarks, the average SQ improves between $\epsilon \in [0.045, 7.18]$. This behavior is expected as greater ϵs imply an increase in the sub-sampling probability and decrease in the noise added (σ). The increase in the probability of sub-sampling allows an agent to explore more values in its domain. That is an increase in the chance of encountering better assignments for itself. Other comments:

- We observe that the average solution quality for GC is slightly lower than that of MS. For both the benchmarks, the quality increases considerably with increasing privacy budget, i.e., ϵ. P-Gibbs' performance for meeting-schedule is strong, especially for higher ϵs.
- Note that $\epsilon < 1$ is desirable. We consider $\epsilon \geq 1$ for illustrative purposes. We observe that P-Gibbs also provides good solution qualities for $\epsilon < 1$. Specifically, for GC, the average quality remains above 0.87 and 0.933 for MS. The quality consistently increases as ϵ increases.
- The sudden increase in the qualities as ϵ varies from $0.045 \longrightarrow 0.65$ can be attributed to the large variation in σ, from $1000 \longrightarrow 5$.
- Since ours is the first method of its kind, to the best of our knowledge, we believe these are strong results, and future work will further improve the performance. One may study the fine-tuning of the hyperparameters $\gamma, \sigma, c,$ and q to arrive at the optimal (empirical) trade-off between the solution quality and ϵ.
- Concerning the infeasibility of a DCOP solution, we remark that incomplete (or random) algorithms like MGM, DUCT, SD-Gibbs, and PD-Gibbs do not aim to solve problems with hard constraints. A hard constraint will leak vital information about the constraints, and a differentially private solution will not work in such a setting. Like [18], we focus on soft constraints; thus, infeasible solutions will not occur.

6 CONCLUSION

In this paper, we addressed the problem of privacy-preserving distributed constraint optimization. With our novel algorithm– P-Gibbs, we are the first to show a DP guarantee for the same. As we use the local DP model, our algorithm preserves the privacy of unrelated agents' preferences. This guarantee also extends to the solution. We also achieve high-quality solutions with reasonably strong privacy guarantees and efficient computation, especially in meeting scheduling problems.

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on CCS*. 308–318.

[2] Ismel Brito, Amnon Meisels, Pedro Meseguer, and Roie Zivan. 2009. Distributed constraint satisfaction with partially known constraints. *Constraints* 14, 2 (2009), 199–234.

[3] Cynthia Dwork. 2006. Differential Privacy. In *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)* (33rd international colloquium on automata, languages and programming, part ii (icalp 2006) ed.) *(Lecture Notes in Computer Science, Vol. 4052)*. Springer Verlag, 1–12.

[4] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Theoretical Computer Science* 9, 3-4 (2014), 211–407.

[5] Boi Faltings, Thomas Léauté, and Adrian Petcu. 2008. Privacy guarantees through distributed constraint satisfaction. In *2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, Vol. 2. IEEE, 350–358.

[6] Alessandro Farinelli, Alex Rogers, Adrian Petcu, and Nicholas R Jennings. 2008. Decentralised coordination of low-power embedded devices using the max-sum algorithm. (2008).

[7] Wilfried Fischer and Bernd W Muller. 1991. Method and apparatus for the manufacture of a product having a substance embedded in a carrier. US Patent 5,043,280.

[8] Sylvain Gelly and David Silver. 2007. Combining online and offline knowledge in UCT. In *Proceedings of the 24th international conference on Machine learning*. 273–280.

[9] Tal Grinshpoun, Alon Grubshtein, Roie Zivan, Arnon Netzer, and Amnon Meisels. 2013. Asymmetric distributed constraint optimization problems. *Journal of Artificial Intelligence Research* 47 (2013), 613–647.

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

[11] Youssef Hamadi, Christian Bessiere, and Joël Quinqueton. 1998. Distributed Intelligent Backtracking.. In *ECAI*. 219–223.

[12] Katsutoshi Hirayama and Makoto Yokoo. 1997. Distributed partial constraint satisfaction problem. In *International Conference on Principles and Practice of Constraint Programming*. Springer, 222–236.

[13] Zhenqi Huang, Sayan Mitra, and Nitin Vaidya. 2015. Differentially private distributed optimization. In *Proceedings of the 2015 International Conference on Distributed Computing and Networking*. 1–10.

[14] Thomas Léauté. 2011. *Distributed Constraint Optimization: Privacy Guarantees and Stochastic Uncertainty*. PhD Thesis. Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland. http://thomas.leaute.name/main/DCOP_privacy_uncertainty_thesis.html

[15] JG Liao. 1998. Variance reduction in Gibbs sampler using quasi random numbers. *Journal of Computational and Graphical Statistics* 7, 3 (1998), 253–266.

[16] Rajiv T Maheswaran, Jonathan P Pearce, Emma Bowring, Pradeep Varakantham, and Milind Tambe. 2006. Privacy loss in distributed constraint reasoning: A quantitative framework for analysis and its applications. *Autonomous Agents and Multi-Agent Systems* 13, 1 (2006), 27–60.

[17] Rajiv T Maheswaran, Jonathan P Pearce, and Milind Tambe. 2006. A family of graphical-game-based algorithms for distributed constraint optimization problems. In *Coordination of large-scale multiagent systems*. Springer, 127–146.

[18] Duc Thien Nguyen, William Yeoh, Hoong Chuin Lau, and Roie Zivan. 2019. Distributed gibbs: A linear-space sampling-based dcop algorithm. *Journal of Artificial Intelligence Research* 64 (2019), 705–748.

[19] Brammert Ottens, Christos Dimitrakakis, and Boi Faltings. 2012. DUCT: An upper confidence bound approach to distributed constraint optimization problems. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

[20] Adrian Petcu and Boi Faltings. 2005. DPOP: A scalable method for multiagent constraint optimization. In *IJCAI 05*. 266–271.

[21] Pierre Rust, Gauthier Picard, and Fano Ramparany. 2019. pyDCOP: a DCOP library for Dynamic IoT Systems. In *International Workshop on Optimisation in Multi-Agent Systems*.

[22] Tamir Tassa, Tal Grinshpoun, and Avishay Yanai. 2019. A Privacy Preserving Collusion Secure DCOP Algorithm. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI*. 4774–4780.

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

[24] Aleksei Triastcyn and Boi Faltings. 2020. Bayesian Differential Privacy for Machine Learning. In *Proceedings of the 37th International Conference on Machine Learning*.

[25] Makoto Yokoo. 2012. *Distributed constraint satisfaction: foundations of cooperation in multi-agent systems*. Springer Science & Business Media.

[26] Makoto Yokoo, Edmund H Durfee, Toru Ishida, and Kazuhiro Kuwabara. 1998. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on knowledge and data engineering* 10, 5 (1998), 673–685.