Differentially Private Multi-Agent Constraint Optimization

Sankarshan Damle,[1] **Aleksei Triastcyn,**[2] **Boi Faltings,**[2] **Sujit Gujar**[1]
[1] Machine Learning Lab, International Institute of Information Technology, Hyderabad
[2] Artificial Intelligence Laboratory, Ecole Polytechnique Fédérale de Lausanne
sankarshan.damle@research.iiit.ac.in, {aleksei.triastcyn, boi.faltings}@epfl.ch, sujit.gujar@iiit.ac.in

Abstract

Several optimization scenarios involve multiple agents that desire to protect the privacy of their preferences. There exist 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, as the result of the computation itself may compromise agents' preferences. In this work, we show how to achieve differential privacy through 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 compared to variants with uniform sampling and the SD-Gibbs algorithm.

1 Introduction

The idea of *distributed computation* has been a trending topic among computer scientists for decades. Distributing the computation has several well-known advantages over centralized computing. Some of these include no single point failure, incremental growth, reliability, open system, parallel computing, and easier management of resources. One of the successful applications of distributed computing is *distributed constraint optimization problem* (DCOP), first introduced in (Yokoo et al. 1998).

Introducing DCOPs. DCOP is a problem where agents collectively compute their value assignments in order 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. DCOPs find use in problems such as distributed scheduling of meetings, the distributed allocation of targets to sensors in a network, the distributed allocation of resources and coordination of mobile agents in disaster evacuation scenarios, and the distributed management of power distribution networks.

DCOP Algorithms. Many a times, 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 (Fioretto, Pontelli, and Yeoh 2018). These algorithms are either: (1) search-based algorithms like SynchBB (Hirayama and Yokoo 1997), ADOPT (Modi et al. 2005) and its variants, AFB (Gershman, Meisels, and Zivan 2009), and MGM (Maheswaran, Pearce, and Tambe 2006), where the agents enumerate through sequences of assignments in a decentralized manner; and (2) inference-based algorithms like DPOP (Petcu and Faltings 2005), max-sum (Farinelli et al. 2008), and Action GDL (Vinyals, Rodriguez-Aguilar, and Cerquides 2009), where the agents use dynamic programming to propagate aggregated information to other agents.

Ottens, Dimitrakakis, and Faltings (2012) propose Distributed Upper Confidence Tree (DUCT), an extension of UCB (Fischer and Muller 1991) and UCT (Gelly and Silver 2007). 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. (2019) improve upon DUCT through their sampling-based DCOP algorithms called *Sequential Distributed Gibbs* (SD-Gibbs) and *Parallel Distributed Gibbs* (PD-Gibbs). These are distributed extensions of the Gibbs algorithm (Liao 1998). 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 showed empirically that SD-Gibbs and PD-Gibbs could find better solutions than DUCT, run faster, and solve some large problems that DUCT fails to solve due to memory limitations. Therefore, in this work, we focus on SD-Gibbs.[1]

Privacy in DCOPs. The need for preserving the privacy of an agent's private information is vital and is an active research area (Damle, Faltings, and Gujar 2019; Sharma, Xing, and Liu 2019; Li et al. 2020; Lu, Tang, and Wang 2018). 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 and form the basis of this work.

Privacy in DCOPs has focused on the use of cryptographic primitives, such as *partial homomorphic encryption*. Sev-

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

eral privacy-preserving algorithms are present in literature, which use cryptographic primitives atop existing DCOP algorithms to provide strong privacy guarantees. These include P-DPOP, $P^{3/2}$-DPOP, P^2-DPOP (Léauté 2011), which build on the DPOP algorithm; P-SyncBB (Grinshpoun and Tassa 2016) over SynchBB; and most recently P-MaxSum (Tassa, Grinshpoun, and Zivan 2017) which presents the privacy variant of the max-sum algorithm. However, the use of cryptographic primitives and the computationally expensive nature of DCOPs results in these algorithms not being scalable, even for moderate problems. Moreover, these techniques cannot be generalized to sampling-based algorithms such as DUCT, SD-Gibbs, which are computationally efficient.

Brito et al. (Brito et al. 2009) use *information entropy* to quantify the privacy loss incurred by an algorithm in the process of solving a DCOP. Grinshpoun et al. (Grinshpoun et al. 2013) present private local-search algorithms, based on the aforementioned algorithms. The authors use the quantification to show that their algorithms provide high quality of solutions while simultaneously preserving privacy. While the privacy loss metric defined in (Brito et al. 2009) is interesting, we believe it is not rigorous. Moreover, the improved performance of SD-Gibbs, both in terms of solution quality and memory, motivates us to design new private algorithms with rigorous privacy guarantees.

Differential Privacy (DP). We use DP (Dwork 2006; Dwork et al. 2006) to preserve privacy of agent preferences, i.e., ensuring *constraint privacy* in DCOPs. For cases when the set of variables and agents involved is common knowledge, 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 (Huang, Mitra, and Vaidya 2015). 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 only knows about agents that 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 (Dwork and Roth 2014), which ensures the indistinguishability of any two agents.

Contributions. Achieving a scalable, privacy-preserving DCOP, without a centralized authority is a challenge. We show that SD-Gibbs may leak information about agent constraints during its execution. Further, the iterative nature of efficient algorithms like SD-Gibbs may lead to a high privacy loss over the iterations.

Addressing this problem, we develop new differentially private variants of the SD-Gibbs algorithm. Specifically, we present (i) P-Gibbs: which uses *softmax with temperature* to smooth sampling distributions in SD-Gibbs; and (ii) P-Uniform: which samples values uniformly. Additionally, during computation, we add *Gaussian* noise to the relative utility in both algorithms. We then provide a refined analysis of privacy within the framework of (ϵ, δ)-DP (Sections 4.1, 4.2 and 4.3). Our experiments demonstrate our algorithms' practicality and strong performance for a reasonable *privacy budget*, i.e, ϵ, with SD-Gibbs as the baseline.

Throughout the paper, we omit proofs of some of the results presented. These proofs are provided in the appendix.

2 Preliminaries

2.1 DCOP

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*. Formally,

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;*
- $\mathcal{D} = \{D_1, \ldots, D_p\}$ *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 (Yokoo 2012), we assume that $p = m$, i.e., the number of agents and the number of variables are equal. Also, let $D = D_i = D_j,\ \forall i, j$, i.e., all variables have the same domain. The total utility in DCOP, for a complete assignment $\mathbf{X} = \{x_i = d_i |\ \forall i\}$, where d_i is the assignment of variable x_i, 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., $F(\mathbf{X}^*) = \max_{\mathbf{X} \in \mathcal{D}} F(\mathbf{X})$.

Constraint Graph 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 $\langle \mathcal{X}, \mathcal{A}, \mathcal{D}, \mathcal{F}, \alpha \rangle$, 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 (Hamadi, Bessiere, and Quinqueton 1998).

For the algorithms presented in this paper, let N_i refer to the set of neighbours 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.

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
X_i and $\bar{X}_i$	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

Procedure 1: Sampling in SD-Gibbs (Nguyen et al. 2019)

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

2.2 Sequential Distributed Gibbs (SD-Gibbs)

We now describe Sequential Distributed Gibbs (SD-Gibbs) as first introduced in (Nguyen et al. 2019). SD-Gibbs uses the well known Gibbs sampling algorithm (Nguyen et al. 2019, Algortihm 1) to solve a DCOP instance as: (i) A DCOP, whose solution is one with the maximum utility, can be mapped to a problem whose solution is one with the maximum likelihood; and (ii) A solution with the maximum utility is also a solution with the maximum likelihood.

The authors then map DCOP to a *maximum a posteriori* (MAP) estimation problem. For this, 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 function, represented by $\theta_{ij}(x_i; x_j)$, is associated with an edge. Let the graph constituting MRF, with nodes and edges, be denoted 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 given by:

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

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

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

Sampling We now describe *sampling* in SD-Gibbs. Let X_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,

$$\begin{aligned} \Pr(x_i | x_j \in \mathcal{X} \setminus \{x_i\}) &= \Pr(x_i | x_j \in N_i) \\ &= \frac{1}{Z} \exp \left[\sum_{\langle x_j, d_j \rangle \in X_i} F_{ij}(d_i, d_j) \right] \end{aligned} \tag{2}$$

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

Algorithm The following steps summarize SD-Gibbs. We present the formal algorithm in the supplement.

S0 The algorithm starts with a construction of the pseudo-tree and each agent initializing each of their variables, from Table 1, to their default values. The root then starts the sampling, as descried in Procedure 1 and sends the VALUE message (line 6) to each of its neighbors.

S1 Upon receiving a VALUE message, the agent i updates its current contexts, X_i and $\bar{X}_i$ with the sender's values. If the message is from agent i's parents, then the agent itself samples, i.e., executes Procedure 1. This *sampling stage* continues until all the leaf agents have sampled.

S2 Each leaf agent j then sends a BACKTRACK message to its parent comprising x_j, Δ_j, and $\bar{\Delta}_j$. When the parent receives the message, it too sends a BACKTRACK message to its parent. The process continues until the root receives the message – concluding one iteration.

S3 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 global relative utility. Based on this, at the end of an iteration, the root decides to keep or throw away the current solution.

2.3 Privacy in DCOPs

Privacy in DCOP algorithms comprises the fact that any given agent's knowledge about the problem will not be revealed to other agents by the messages exchanged during the algorithm execution. Privacy definitions relevant to DCOPs include (Faltings, Léauté, and Petcu 2008): (i) *Agent Privacy* which states that no agent should be able to discover the existence of its non-neighbors; (ii) *Topology Privacy* which implies that agents must not learn topological information of the CG; (iii) *Decision Privacy* which prevents the discovery of an agent's assignments from other agents; and (iv) *Constraint Privacy* which states that no agent must be able to discover the nature of constraint that does not involve a variable it owns. In this work, we focus on constraint privacy to ensure privacy of agent preferences. Since absolute privacy is not an achievable goal (Dwork 2006), we formalise constraint privacy in terms of (ϵ, δ)-DP (Dwork and Roth 2014).

2.4 Differential Privacy

Differential Privacy (DP) is normally defined for *adjacent* databases, i.e., databases differing in a single entry. However, in this instance, not only we want to protect privacy against external adversaries, but also against curious fellow agents. To do so, we consider the local model of privacy (Dwork and Roth 2014). It is defined on individual entries rather than databases, or in our setting, on individual agents. Formally, we want our algorithm for any two utility functions (vectors in $\mathbb{R}^p$) to satisfy the following definition, from (Dwork and Roth 2014),

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

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

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 3).
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.

These follow by observing what critical information gets transferred by each agent i in S0-S3. We ignore t^* and $\bar{t}^*$ because these are simply functions of utility, i.e., will be private by post-processing property once utility are 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)**.** *Sensitivity, denoted by τ, 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 Differential Privacy in 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 $F_{ij}(\cdot, \cdot) > 0, \forall i$. Further, in order to bound privacy loss of the algorithm by a reasonably small ϵ, we introduce the following assumption about utility functions.

3.1 Privacy Assumption

Recall the notion of *max-divergence*, a worst-case analogue of KL divergence, for two probability distributions P and Q:

$$\mathcal{D}_{\infty}(P||Q) = \max_{x} \left(\ln \frac{P(x)}{Q(x)} \right). \tag{6}$$

Absolute continuity is trivially satisfied by the initial assumptions introduced above. With this in mind, we now present the main privacy assumption.

Assumption 1 (Bounded Sampling Divergence)**.** *For any two agents i and j, with value domains $D = D_i = D_j$ and $i \neq j$, max-divergence between SD-Gibbs sampling distributions is bounded by a constant Γ_{∞}. Formally, $\forall i, j$, s.t. $i \neq j$ and $\forall X_i, X_j$, we have,*

$$\mathcal{D}_{\infty}(\mathbb{P}_i||\mathbb{P}_j) \leq \Gamma_{\infty}. \tag{7}$$

3.2 Bounding Sensitivity with Assumption 1

We now prove that Assumption 1 leads to bounded sensitivity for relative utility Δ. Specifically, Claim 1 presents Γ_{∞} as the maximum difference of sums of utility, for any pair of agents in SD-Gibbs. Then, Theorem 1 uses this difference to bound the sensitivity.

Claim 1. *In SD-Gibbs, $\forall i, j$ s.t $i \neq j$ and $D = D_i = D_j$, from Assumption 1, we have,*

$$\max_{d \in D} \left(\sum_{X_i} F_{ik} - \sum_{X_j} F_{jl} \right) \leq \Gamma_{\infty}$$

Proof. Observe that,

$$\begin{aligned} &\max_{d \in D} \left(\sum_{X_i} F_{ik} - \sum_{X_j} F_{jl} \right) \\ &\quad = \max_{d \in D} \ \ln \left(\frac{1/Z \cdot \exp \left[\sum_{X_i} F_{ik}(d, d_k) \right]}{1/Z \cdot \exp \left[\sum_{X_j} F_{jl}(d, d_l) \right]} \right) \\ &\quad = \max_{d \in D} \ \ln \left(\frac{\mathbb{P}_i(x_i = d)}{\mathbb{P}_j(x_j = d)} \right) \leq \Gamma_{\infty}. \end{aligned} \tag{8}$$

This proves the claim. □

Theorem 1. *For any two agents i and j, $i \neq j$, under Assumption 1, the relative utility sensitivity in SD-Gibbs is bounded by $\tau = 2 \cdot \Gamma_{\infty}$.*

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

Proof. We know,

$$\Delta_i \leftarrow \sum_{\langle x_k, d_k\rangle \in X_i} \Big[F_{ik}(d_i, d_k) - F_{ik}(\hat{d_i}, d_k)\Big].$$

For brevity, let $F_i(d)$ represent $F_{ik}(d, d_k), \forall \langle x_k, d_k\rangle \in X_i$. Similarly, $F_j(d)$ represent $F_{jl}(d, d_l), \forall \langle x_l, d_l\rangle \in X_j$. Now from (5), let $\kappa = \big|\Delta_i - \Delta_j\big|$, i.e.,

$$\kappa = \left|\sum_{X_i} F_i(d) - \sum_{X_i} F_i(\hat{d}) - \left(\sum_{X_j} F_j(d) - \sum_{X_j} F_j(\hat{d})\right)\right|$$
$$= \left|\sum_{X_i} F_i(d) - \sum_{X_j} F_j(d) - \left(\sum_{X_i} F_i(\hat{d}) - \sum_{X_j} F_j(\hat{d})\right)\right|$$

From Claim 1 and (5), we get, $0 \leq \kappa \leq 2 \cdot \Gamma_\infty$, and thus, $\tau = 2 \cdot \Gamma_\infty$. □

3.3 Enforcing Assumption 1 with Softmax

In order to enforce Assumption 1, we apply a *softmax* function to sampling distributions. Let the softmax distribution with *temperature* γ over $\mathbb{P}_i$ be given by p_i:

$$p_i(\mathbf{x}_i, \gamma) = \left\{ \frac{\exp(\mathbb{P}_i(x_i = d_k)/\gamma)}{\sum_{d_l \in D} \exp(\mathbb{P}_i(x_i = d_l)/\gamma)}; \forall d_k \in D \right\} \tag{9}$$

The following claim follows from Assumption 1.

Claim 2. *For two probability distributions using softmax, p_i and p_j, we have, $\forall i, j$, $\forall d \in D$ and $\forall D$, s.t. $|D| > 1, \gamma \geq 1$*

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

Effect of Softmax. We illustrate the effect of softmax 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 softmax distributions, from (9), 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 softmax distribution is more *uniform* than Gibbs'. This implies that an adversary will be more indifferent towards the domain values the domain values while sampling. For e.g., d_1 and d_2 in $p(\cdot, \gamma = 1)$. Thus softmax allows us to improve on the privacy leak during sampling in SD-Gibbs.

4 Preserving Constraint Privacy in DCOP

We now build upon SD-Gibbs to present two novel, scalable algorithms for DCOPs that preserve constraint privacy.

4.1 P-Gibbs

The difference in P-Gibbs and SD-Gibbs is as follows:

1. P-Gibbs uses softmax function over SD-Gibbs distributions for sampling d_i's, $\forall i$.
2. 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 $1 - q$.

Providing (ϵ, δ)-DP for P-Gibbs We first calculate the privacy parameters of the sampling stage, denoted by ϵ_s and δ, in P-Gibbs. For this, we use an extension of the moments accountant method (Abadi et al. 2016) for non-Gaussian mechanisms. Following derivations by Triastcyn and Faltings (2020),

$$\Pr[L \geq \epsilon_s] \leq \max_{F, F'} e^{\lambda \mathcal{D}_{\lambda+1}\left[\mathcal{M}(F)||\mathcal{M}(F')\right] - \lambda\epsilon_s}. \tag{10}$$

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 Triastcyn and Faltings (2020), we consider the classical DP. Using their notion of Bayesian DP could improve the bounds, but we leave it for future work.

Also from (Triastcyn and Faltings 2020), we borrow the notion of *privacy cost* $c_t(\lambda)$. By trivial manipulation, for each iteration t,

$$c_t(\lambda) = \max_{i,j} \lambda \mathcal{D}_{\lambda+1}\left[p_i(d)||p_j(d)\right] \leq \lambda\Gamma_\infty, \tag{11}$$

where (11) 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 Triastcyn and Faltings (2020) for our mechanism and classical DP, we formulate the following result.

Theorem 2. *Privacy cost $c_t(\lambda)$ at iteration t of a sampling stage of P-Gibbs, with agent subsampling probability q, is*

$$c_t^{(s)}(\lambda) = \ln \mathbb{E}_{k \sim B(\lambda+1, q)}\left[e^{k\Gamma_\infty}\right],$$

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

Finally, merging the results, we can compute ϵ_s, δ across multiple iterations as

$$\left.\begin{array}{l} \ln\delta \leq \sum_{t=1}^{T} c_t^{(s)}(\lambda) - \lambda\epsilon_s \\ \epsilon_s \leq \frac{1}{\lambda}\left(\sum_{t=1}^{T} c_t^{(s)}(\lambda) - \ln\delta\right) \end{array}\right\} \tag{12}$$

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 P-Uniform

We presented P-Gibbs, which uses a softmax function to satisfy Assumption 1 and bound the privacy loss incurred by sampling. Coupled with Theorem 2 and Eq. 12, it allows us to quantify privacy parameters ϵ_s and δ.

A more "extreme" way to prevent this leakage is to always sample uniformly, i.e., $d_i \in \text{Uniform}(D_i)$, for each agent i. This will provide the least information leakage in the sampling stage of SD-Gibbs. We refer to this private version of SD-Gibbs as *P-Uniform*.

Theorem 3. *Privacy loss in P-Uniform for sampling $d \in D$ is 0, for any two agents i and j, $D = D_i = D_j$ and $i \neq j$.*

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

Table 2: Per-iteration and final (ϵ, δ) bounds for P-Gibbs and P-Uniform.

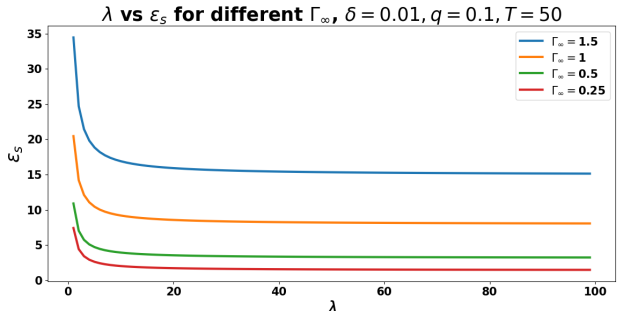

Figure 1: Variation of ϵ_s with λ

Proof. Observe that the probability of sampling any $d \in D$ will the be the same for any two distinct agents i and j. As a result, from Eq. 4, the privacy loss incurred is 0. □

This implies that $\epsilon_s = 0$ in P-Uniform. Further, observe that the temperature parameter in P-Gibbs may be tuned to decrease the overall privacy budget (Figure 1). As $\gamma \to \infty$, we have $p_i = p_j \implies \epsilon_s \to 0$. Thus, increasing γ, or equivalently, decreasing Γ_∞, leads to P-Gibbs performing more similarly to P-Uniform, as more information of SD-Gibbs sampling distribution is lost.

4.3 Privacy of Δ in P-Gibbs and P-Uniform

Sections 4.1 and 4.2 deal with the privacy loss occurring due to sampling in P-Gibbs and P-Uniform, respectively. However, as aforementioned, the values Δ and $\bar{\Delta}$ also leak information about agents' constraints. As a result, we must sanitize these values so as to fully preserve privacy. We achieve this by employing the *Gaussian noise mechanism* (Dwork and Roth 2014) 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 (Dwork and Roth 2014, Theorem A.1) or the moments accountant (Abadi et al. 2016). The latter can be unified with the accounting for the sampling stage by using:

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

Table 2 summarises expressions for per-iteration and total ϵ values for P-Gibbs and P-Uniform.

5 Experimental Evaluation

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

Setup *pyDCOP* (Rust, Picard, and Ramparany 2019) is a *Python* module that provides implementations of many DCOP algorithms (DSA, MGM, MaxSum, DPOP, etc.). It also allows the implementation of one's own DCOP algorithm easily, by providing all the required infrastructure: agents, messaging system, metrics collection, etc. We use pyDCOP's implementation of the SD-Gibbs algorithm to run our experiments. In addition, we also implement our privacy variants – P-Gibbs and P-Uniform.

Generating Test-cases pyDCOP allows for generating random test-cases for various problems through its command line interface using the *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 25 such randomly generated problems.

Method We consider the utility given by SD-Gibbs' solution as one of the two baselines. The second baseline is the "one-shot" random assignment, i.e., an algorithm that randomly assigns values to each agent and outputs the utility based on it. As there is no exchange of information in such an one-shot assignment, trivially with respect to constraint privacy, (i) one-shot: $\epsilon = 0$; and (ii) SD-Gibbs: $\epsilon = \infty$.

Further, note that all these methods, including our P-Gibbs and P-Uniform, are random algorithms. Hence, for a fair comparison, we run each benchmark problem 20 times and use the subsequent average utility for our calculations.

Throughout our experiments we fix: $\gamma = 4$, $\lambda = 100$ (for sampling), $\lambda = 7$ (for noise), $q = 0.1$, $\delta = 10^{-2}$, and $T = 50$. The choice of δ is such that $\delta < 1/m$. We let $\gamma = 4$ which implies $\Gamma_\infty = 0.5$. Finally, we consider $\epsilon \in \{1, 5, 7.5, 10\}$, calculated using the results in Table 2.

Solution Quality (SQ) We use the following metric:

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

$$\text{SQ}_A = \frac{U_A - U_R}{U_S - U_R},$$

for utilities of $\mathcal{A}$*, SD-Gibbs, and one-shot:* U_A, U_S, U_R*.*

SQ allows us to place the utility of P-Gibbs and P-Uniform in the context of SD-Gibbs and one-shot random solutions. $\text{SQ}_A \approx 1$ indicates that utility does not deteriorate compared to SD-Gibbs. On the other hand, $\text{SQ}_A \approx 0$ means that there is little gain compared to a purely random

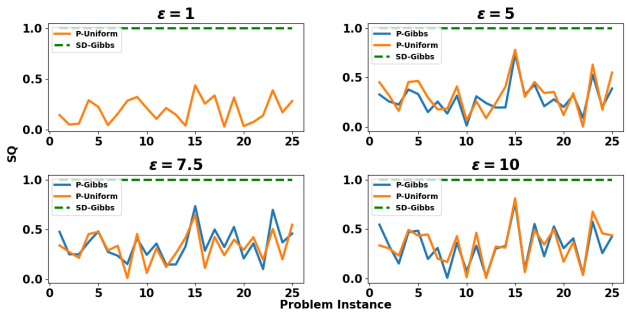

Figure 2: Graph Coloring Benchmark

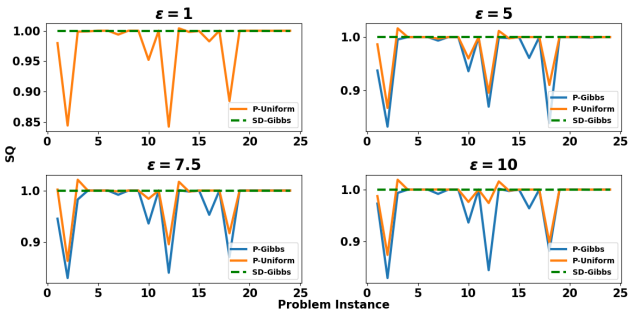

Figure 3: Meeting Scheduling Benchmark

Algorithm	ϵ	SQ (mean $\pm$ std)
P-Gibbs	5	0.281 ± 0.143
	7.5	0.348 ± 0.159
	10	0.324 ± 0.192
P-Uniform	1	0.190 ± 0.119
	5	0.322 ± 0.180
	7.5	0.321 ± 0.153
	10	0.341 ± 0.195

Table 3: Graph coloring, average SQ across problems.

Algorithm	ϵ	SQ (mean $\pm$ std)
P-Gibbs	5	0.973 ± 0.0513
	7.5	0.973 ± 0.051
	10	0.975 ± 0.0498
P-Uniform	1	0.979 ± 0.0475
	5	0.985 ± 0.0372
	7.5	0.987 ± 0.0373
	10	0.990 ± 0.0328

Table 4: Meeting scheduling, average SQ across problems.

assignment. It is possible that $\text{SQ}_A > 1$ due to randomness or privacy noise acting as a simulated annealing.

5.1 Graph-Coloring (GC)

We generate 25 sample graph-coloring problems. The problems are such that the number of agents/variables lie between $[30, 50)$ and agents' domain size between $[10, 20)$. Each constraint is a random integer taken from $(0, 10)$.

Figure 2 gives SQ per problem instance, per ϵ. Table 3 provides SQ scores averaged across all problems. For both algorithms, the average SQ improves between the minimum and the maximum privacy budgets, although not evenly across the range. Overall, private SQ is evidently lower in this benchmark. However, since it is the first method of its kind, to the best of our knowledge, we believe it is still a strong result, and future work will improve the performance.

5.2 Meeting Schedule (MS)

We generate 25 sample meeting scheduling problems. The problems are such that the number of agents/variables lie between $[1, 50)$ with 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 occupy $[1, 5]$ slots randomly. Figure 3 gives the comparison.

Analogously to the graph-coloring example, Figure 3 provides SQ scores per problem, with different plots for each ϵ, and Table 4 average SQ over all problems. Notably, both methods perform nearly as well as SD-Gibbs.

In this problem, performance is very strong and there is virtually no loss of utility. The average SQ increases only marginally with the privacy budget, which could be attributed to the fact that the SQ is high to begin with.

In both problems, P-Gibbs and P-Uniform perform very similarly, which is not intuitive at a first glance. However, we can attribute this behavior to the following fact. For equal values of ϵ, P-Uniform adds less noise in Δ, thereby alleviating the lack of information in the sampling stage.

Discussion. We believe that the improved performance of our privacy variants in MS over SD-Gibbs is because the Gibbs sampling in SD-Gibbs assumes statistical independence of the variables. This is given in random GC instances but not in MS. Thus, the performance of SD-Gibbs is not significantly better than our privacy variants in MS. Our variants also outperform SD-Gibbs in some cases (Figure 3) which can be attributed to the noise added which acts as simulated annealing.

6 Conclusion

In this paper, we address the problem of privacy-preserving distributed constraint optimization. We employ a rigorous standard of differential privacy to guarantee constraint and utility privacy for agents. As we use the local model of DP, our approach protects not only against external adversaries but also against other agents within the same system.

We propose two novel, private DCOP algorithms based on SD-Gibbs. For these methods, we perform theoretical privacy analysis and experimental evaluation on such benchmark problems as graph coloring and meeting scheduling. Our experiments demonstrate that the developed techniques can achieve strong privacy guarantees while maintaining high utility, especially in meeting scheduling problems.

References

Abadi, M.; Chu, A.; Goodfellow, I.; McMahan, H. B.; Mironov, I.; Talwar, K.; and Zhang, L. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 308–318.

Brito, I.; Meisels, A.; Meseguer, P.; and Zivan, R. 2009. Distributed constraint satisfaction with partially known constraints. *Constraints* 14(2): 199–234.

Damle, S.; Faltings, B.; and Gujar, S. 2019. A Truthful, Privacy-Preserving, Approximately Efficient Combinatorial Auction For Single-minded Bidders. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 1916–1918.

Dwork, C. 2006. Differential Privacy. In *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*, volume 4052 of *Lecture Notes in Computer Science*, 1–12. Springer Verlag. ISBN 3-540-35907-9. URL https://www.microsoft.com/en-us/research/publication/differential-privacy/.

Dwork, C.; McSherry, F.; Nissim, K.; and Smith, A. 2006. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, 265–284. Springer.

Dwork, C.; and Roth, A. 2014. The Algorithmic Foundations of Differential Privacy. *Theoretical Computer Science* 9(3-4): 211–407.

Faltings, B.; Léauté, T.; and Petcu, A. 2008. Privacy guarantees through distributed constraint satisfaction. In *2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, volume 2, 350–358. IEEE.

Farinelli, A.; Rogers, A.; Petcu, A.; and Jennings, N. R. 2008. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2*, 639–646. International Foundation for Autonomous Agents and Multiagent Systems.

Fioretto, F.; Pontelli, E.; and Yeoh, W. 2018. Distributed constraint optimization problems and applications: A survey. *Journal of Artificial Intelligence Research* 61: 623–698.

Fischer, W.; and Muller, B. W. 1991. Method and apparatus for the manufacture of a product having a substance embedded in a carrier. US Patent 5,043,280.

Gelly, S.; and Silver, D. 2007. Combining online and offline knowledge in UCT. In *Proceedings of the 24th international conference on Machine learning*, 273–280.

Gershman, A.; Meisels, A.; and Zivan, R. 2009. Asynchronous forward bounding for distributed COPs. *Journal of Artificial Intelligence Research* 34: 61–88.

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

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

Hamadi, Y.; Bessiere, C.; and Quinqueton, J. 1998. Distributed Intelligent Backtracking. In *ECAI*, 219–223.

Hirayama, K.; and Yokoo, M. 1997. Distributed partial constraint satisfaction problem. In *International Conference on Principles and Practice of Constraint Programming*, 222–236. Springer.

Huang, Z.; Mitra, S.; and Vaidya, N. 2015. Differentially private distributed optimization. In *Proceedings of the 2015 International Conference on Distributed Computing and Networking*, 1–10.

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

Li, Q.; Wu, Z.; Wen, Z.; and He, B. 2020. Privacy-Preserving Gradient Boosting Decision Trees. *Proceedings of the AAAI Conference on Artificial Intelligence* 34(01): 784–791. doi: 10.1609/aaai.v34i01.5422. URL https://ojs.aaai.org/index.php/AAAI/article/view/5422.

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

Lu, Y.; Tang, Q.; and Wang, G. 2018. ZebraLancer: Private and Anonymous Crowdsourcing System atop Open Blockchain. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 853–865. doi: 10.1109/ICDCS.2018.00087.

Maheswaran, R. T.; Pearce, J. P.; and Tambe, M. 2006. A family of graphical-game-based algorithms for distributed constraint optimization problems. In *Coordination of large-scale multiagent systems*, 127–146. Springer.

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

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

Ottens, B.; Dimitrakakis, C.; and Faltings, B. 2012. DUCT: An upper confidence bound approach to distributed constraint optimization problems. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Petcu, A.; and Faltings, B. 2005. DPOP: A scalable method for multiagent constraint optimization. In *IJCAI 05*, CONF, 266–271.

Rust, P.; Picard, G.; and Ramparany, F. 2019. pyDCOP: a DCOP library for Dynamic IoT Systems. In *International Workshop on Optimisation in Multi-Agent Systems*.

Sharma, S.; Xing, C.; and Liu, Y. 2019. Privacy-preserving deep learning with SPDZ. In *The AAAI Workshop on Privacy-Preserving Artificial Intelligence*.

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

Triastcyn, A.; and Faltings, B. 2020. Bayesian Differential Privacy for Machine Learning. In *Proceedings of the 37th International Conference on Machine Learning*.

Vinyals, M.; Rodriguez-Aguilar, J. A.; and Cerquides, J. 2009. Generalizing DPOP: Action-GDL, a new complete algorithm for DCOPs. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, 1239–1240. International Foundation for Autonomous Agents and Multiagent Systems.

Yokoo, M. 2012. *Distributed constraint satisfaction: foundations of cooperation in multi-agent systems*. Springer Science & Business Media.

Yokoo, M.; Durfee, E. H.; Ishida, T.; and Kuwabara, K. 1998. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on knowledge and data engineering* 10(5): 673–685.

Algorithm 1: Sequential Distributed Gibbs (Nguyen et al. 2019)

Create pseudo-tree
Each agent x_i calls INITIALIZE()

Procedure 1: INITIALIZE() (Nguyen et al. 2019)

$d_i \leftarrow \hat{d}_i \leftarrow d_i^* \leftarrow \bar{d}_i \leftarrow \text{ValInit}(x_i)$
$X_i \leftarrow \bar{X}_i \leftarrow \{(x_j, \text{ValInit}(x_j)) | x_j \in N_i\}$
$t_i \leftarrow t_i^* \leftarrow \bar{t}_i^* \leftarrow 0$
$\Delta_i \leftarrow \bar{\Delta}_i \leftarrow 0$
if x_i *is root* **then**
  $t_i \leftarrow t_i^* \leftarrow \bar{t}_i^* \leftarrow 0$
  SAMPLE()
end

Procedure 2: SAMPLE() (Nguyen et al. 2019)

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

A SD-Gibbs

Algorithm 1 presents the complete SD-Gibbs algorithm.

B Proofs

We now restate the results presented in the main paper and provide their formal proofs.

B.1 Proof of Claim 2

Claim. *For two probability distributions using softmax, p_i and p_j, we have, $\forall i, j$, $\forall d \in D$ and $\forall D$, s.t. $|D| > 1, \gamma \geq 1$*

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

Proof. Because p_i and p_j are soft-max distributions, we have,

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

$$p_j(\mathbf{x}_j, \gamma) = \left\{ \frac{\exp(\mathbb{P}_j(x_j = d_k)/\gamma)}{\sum_{d_l \in D} \exp(\mathbb{P}_j(x_j = d_l)/\gamma)} \; ; \; \forall d_k \in D \right\}.$$

Procedure 3: VALUE($x_s, d_s, \bar{d}_s, t^*_s, \bar{t}^*_s$) (Nguyen et al. 2019)

Update $\langle x_s, d'_s \in X_i \rangle$ with (x_s, d_s)
if $x_s \in PP_i \cup \{P_i\}$ **then**
  Update $\langle x_s, d'_s \in \bar{X}_i \rangle$ with $(x_s, \bar{d}_s)$
else
  Update $\langle x_s, d'_s \in X_i \rangle$ with $(x_s, \bar{d}_s)$
end
if $x_s = P_i$ **then**
  if $\bar{t}^*_s \geq t^*_s \textbf{and} \bar{t}^*_s > max\{t^*_i, \bar{t}^*_i\}$ **then**
    $d^*_i \leftarrow \bar{d}_i; \bar{t}^*_i \leftarrow \bar{t}^*_s$
  else if $t^*_s \geq \bar{t}^*_s \textbf{and} \bar{t}^*_s > max\{t^*_i, \bar{t}^*_i\}$ **then**
    $d^*_i \leftarrow \bar{d}_i; t^*_i \leftarrow t^*_s$
  end
  SAMPLE()
  if x_i *is a leaf* **then**
    Send BACKTRACK($x_i, \Delta_i, \bar{\Delta}_i$) to P_i
  end
end

Procedure 4: BACKTRACK($x_s, \Delta_s, \bar{\Delta}_s$) (Nguyen et al. 2019)

$\Delta_i \leftarrow \Delta_i + \Delta_s; \bar{\Delta}_i \leftarrow \bar{\Delta}_i + \bar{\Delta}_s$
if Received BACKTRACK from all children in this iteration **then**
  Send BACKTRACK($x_i, \Delta_i, \bar{\Delta}_i$) to P_i
  if x_i *is root* **then**
    $\bar{\Omega} \leftarrow \Omega + \bar{\Delta}_i; \Omega \leftarrow \Omega + \Delta_i$
    if $\Omega \geq \bar{\Omega}$ ***and*** $\Omega > \Omega^*$ **then**
      $\Omega^* \leftarrow \Omega; d^*_i \leftarrow d_i; t^*_i \leftarrow t_i$
    else if $\bar{\Omega} \geq \Omega$ ***and*** $\bar{\Omega} > \Omega^*$ **then**
      $\Omega^* \leftarrow \bar{\Omega}; d^*_i \leftarrow \bar{d}_i; \bar{t}^*_i \leftarrow t_i$
    end
    SAMPLE()
  end
end

Note that, $\mathbb{P}_i$ and $\mathbb{P}_j$ are the probability distributions through SD-Gibbs sampling. With this, observe,

$$\ln\left[\frac{p_i(x_i = d, \cdot)}{p_j(x_j = d, \cdot)}\right] \leq \ln\left[\frac{\frac{\exp(\mathbb{P}_i(x_i=d)/\gamma)}{\sum_{d_l \in D}\exp(\mathbb{P}_i(x_i=d_l)/\gamma)}}{\frac{\exp(\mathbb{P}_j(x_j=d)/\gamma)}{\sum_{d_l \in D}\exp(\mathbb{P}_j(x_j=d_l)/\gamma)}}\right] \leq \ln\left[\frac{\exp(1/\gamma(\mathbb{P}_i - \mathbb{P}_j)))}{N_1/N_2}\right] \quad \text{(A2)}$$

Here, $N_1 = \sum_{d_l \in D}\exp(\mathbb{P}_i(x_i = d_l)/\gamma)$ and $N_2 = \sum_{d_l \in D}\exp(\mathbb{P}_j(x_j = d_l)/\gamma)$. Now, in (A2) observe,

1. Both the numerator and denominator in r.h.s of (A2) are positive.
2. As $\ln(x)$ is an increasing function x, this implies that r.h.s of (A2) is maximum when the numerator is maximum and denominator is minimum.
3. The difference, $\mathbb{P}_i(x_i = d) - \mathbb{P}_j(x_j = d)$, can be at-most 1. Therefore, numerator in r.h.s of (A2) is at-most $\exp(1/\gamma) = e^{1/\gamma}$.
4. The denominator in r.h.s of (A2) is minimum when N_1 is minimum and N_2 is maximum. Note that, N_1 is minimum when $\mathbb{P}_i(x_i = d) = 0, \forall d$, i.e., minimum $N_1 = |D|$. But, N_2 is maximum when $\mathbb{P}_j(x_j = d) = 1, \forall d$, i.e., maximum $N_2 = |D| \cdot e^{1/\gamma}$.
5. Using these values in (A2) completes the claim.

□

B.2 Proof of Theorem 2

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

$$c^{(s)}_t(\lambda) = \ln \mathbb{E}_{k \sim B(\lambda+1, q)}\left[e^{k\Gamma_\infty}\right],$$

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 Γ_∞ in place of the ratio of normality distributions in (Triastcyn and Faltings 2020, Theorem 3). □

Discussion.

1. The theorem follows from Assumption 1.
2. Unlike the analysis in (Triastcyn and Faltings 2020, Theorem 3), we do not have $c^L_t(\lambda)$ and $c^R_t(\lambda)$, 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., Γ_∞.

B.3 Proof of Theorem 3

Theorem. *Privacy loss in P-Uniform for sampling $d \in D$ is 0, for any two agents i and j, $D = D_i = D_j$ and $i \neq j$.*

Proof. For any two agents i and j, $D = D_i = D_j$ and $i \neq j$, from (4),

$$\begin{aligned} L &= \ln\left(\frac{\Pr(x_i \in D)}{\Pr(x_j \in D)}\right) \\ &\leq \max_{d \in D} \ \ln\left(\frac{\Pr(x_i = d)}{\Pr(x_j = d)}\right) \\ &\leq \ln\left(\frac{1/|D|}{1/|D|}\right) = \ln(1) = 0. \end{aligned}$$

This proves the theorem. □