Autonomous Agents and Multi-Agent Systems (2024) 38:8
https://doi.org/10.1007/s10458-024-09636-x

Differentially private multi-agent constraint optimization

Sankarshan Damle[1] · Aleksei Triastcyn[2] · Boi Faltings[2] · Sujit Gujar[1]

Accepted: 18 January 2024 / Published online: 2 March 2024

Abstract
Distributed constraint optimization (DCOP) is a framework in which multiple agents with private constraints (or preferences) cooperate to achieve a common goal optimally. DCOPs are applicable in several multi-agent coordination/allocation problems, such as vehicle routing, radio frequency assignments, and distributed scheduling of meetings. However, optimization scenarios may involve multiple agents wanting to protect their preferences' privacy. Researchers propose privacy-preserving algorithms for DCOPs that provide improved privacy protection through cryptographic primitives such as partial homomorphic encryption, secret-sharing, and secure multiparty computation. These privacy benefits come at the expense of high computational complexity. Moreover, such an approach does not constitute a rigorous privacy guarantee for optimization outcomes, as the result of the computation may compromise agents' preferences. In this work, we show how to achieve privacy, specifically Differential Privacy, by randomizing the solving process. In particular, we present P-Gibbs, which adapts the current state-of-the-art algorithm for DCOPs, namely SD-Gibbs, to obtain differential privacy guarantees with much higher computational efficiency. Experiments on benchmark problems such as Ising, graph-coloring, and meeting-scheduling show P-Gibbs' privacy and performance trade-off for varying privacy budgets and the SD-Gibbs algorithm. More concretely, we empirically show that P-Gibbs provides fair solutions for competitive privacy budgets.

Keywords Distributed constrain optimization · Differential privacy · SD-Gibbs

✉ Sankarshan Damle
sankarshan.damle@research.iiit.ac.in

Aleksei Triastcyn
aleksey.tryastsyn@alumni.epfl.ch

Boi Faltings
boi.faltings@epfl.ch

Sujit Gujar
sujit.gujar@iiit.ac.in

[1] Machine Learning Lab (MLL), IIIT, Hyderabad, Hyderabad, India

[2] Artificial Intelligence Laboratory (LIA), EPFL, Lausanne, Switzerland

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. These include no single-point failure, incremental growth, reliability, open system, parallel computing, and easier management of resources. In this paper, we focus on the distributed analogue of constrained optimization [1], namely *distributed constraint optimization problem* (DCOP), first introduced in [2].

DCOP is a problem where agents collectively compute their value assignments to maximize (or minimize) the sum of resulting *constraint* rewards. In DCOP, constraints quantify each agent's *preference* for each possible assignment. DCOPs help model various multi-agent coordination and resource allocation problems like distributed scheduling of meetings and graph-coloring related applications such as mobile radio frequency assignments. For instance, consider the problem of meeting-scheduling in which several Chief Executive Officers (CEOs) aim to decide a date and time to meet. Each CEO will have a constraint for each date and time slot assignment, quantifying its preference for the assignment. The preferences may depend on the CEOs' availability and favorable slots. In this scenario, the CEOs cannot directly employ a centralized coordinator to decide on an agreeable meeting slot. The coordinator will require information regarding the CEOs' availability—which is often sensitive. Alternatively, the CEOs can generate a suitable schedule by modeling the problem as a DCOP and using any DCOP-solving algorithms. However, researchers show that despite their distributed nature, DCOP algorithms may themselves leak sensitive information [3].

1.1 Privacy in DCOPs

In general, the need to preserve the privacy of an agent's sensitive information in AI/ML solutions is paramount [4–6]. Despite its distributed nature, 'solving' a DCOP instance transfers information across agents, which may leak sensitive information to the other participants, such as the agent's preferences. In the above example, an information leak may involve a CEO inferring critical information about the other participating CEOs during the information exchange. Thus, privacy-preserving solutions to DCOPs are necessary and form the basis of this work. Before discussing the existing privacy-preserving DCOP literature, we first summarize the existing DCOP algorithms.

1.1.1 DCOP algorithms

Solving a DCOP instance is NP-hard [7]. 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 [8], ADOPT [7] and its variants, AFB [9] and MGM [10], where the agents enumerate through sequences of assignments in a decentralized manner; (2) inference-based algorithms like DPOP [11] and Max-Sum [12], where the agents use dynamic programming to propagate aggregated information to other agents; (3) sampling-based algorithms like DUCT [13, 14], where the agents iteratively sample promising assignments. We refer the reader to [15] for a comprehensive survey on DCOP algorithms.

This paper focuses on SD-Gibbs (and its parallel analog PD-Gibbs) [16], the current state-of-the-art algorithms for approximately solving DCOPs. SD-Gibbs is known to run faster (e.g., compared to DUCT [16]), find a better quality of solutions (e.g., compared to MGM and DUCT [14]), and be applicable for larger problems (e.g., compared to DPOP [14] and DUCT [16]).

1.1.2 Privacy-preserving DCOP algorithms

In literature, several algorithms exist to preserve privacy in DCOPs. Unfortunately, we identify that such existing privacy-preserving algorithms have two significant drawbacks. Firstly, these algorithms lack scalability with respect to the number of agents and constraints. Secondly, privacy-preserving complete algorithms for DCOPs converge at the optimal solution. As such, the said solution may be used to infer potentially critical information regarding the DCOP instance. We next discuss these drawbacks in detail.

1.1.2.1 Non-scalability of Private DCOPs As DCOPs are NP-hard, complete DCOP algorithms such as DPOP do not scale as it is. The added complexity of the underlying cryptographic primitives further hits the scalability of its privacy variants: P-DPOP [3], $P^{3/2}$-DPOP [17], and P^2-DPOP [17]. Of these, P-DPOP scales the best, primarily due to its weaker privacy guarantees. Even for P-DPOP, the algorithm is known to only scale up to 12 agents for graph-coloring, and 6 agents for meeting-scheduling – two popular benchmark problems in DCOP literature.

On the other hand, more recent algorithms like P-Max-Sum [18] and P-SyncBB [19] scale better, either in part to the underlying approximate algorithm (P-Max-Sum) or efficient secure multi-party computation protocols (P-SyncBB). However, the algorithms are still computationally intensive. For instance, P-Max-Sum requires a computational overhead ranging from minutes to an hour. Also, the algorithm's run-time increases by a factor of 1000s over its non-private variant [18].

1.1.2.2 Solution Privacy In addition to their lack of scalability, privacy-preserving DCOP algorithms built atop complete algorithms output the optimal assignment (or solution). However, these final assignments cannot be private and, in turn, may leak critical information about agents' preferences [3]. We refer to this information leak as *solution privacy*. For complete DCOP algorithms such as DPOP, their privacy variants built through cryptographic primitives such as P-DPOP [3], $P^{3/2}$-DPOP [17], and P^2-DPOP [17] trivially do not satisfy solution privacy.

1.2 Our goal and approach

In summary, we note that while algorithms exist that realize constraint privacy, their non-scalability hinders their practical use. The cryptographic primitives used to achieve privacy further exaggerate this lack of scalability. Moreover, information leaked through the algorithms' output can be used to extract significant private information, especially when the problem is solved repeatedly. Motivated by these, we aim to construct a scalable DCOP algorithm while providing rigorous and provable privacy guarantees for agents' constraints and one that satisfies solution privacy.

Note that the non-guarantee of solution privacy is an inevitable outcome of a cryptographically secure algorithm. However, it is possible to make the final assignment of a

DCOP algorithm *differentially private* [20]. Consequently, to achieve such a private and scalable algorithm, we focus on the strong notion of *differential privacy* (DP) [20, 21]. In particular, we focus on achieving privacy in SD-Gibbs using DP techniques. Furthermore, we consider a *stronger* local model of privacy [21], which ensures the indistinguishability of any two agents.

1.2.1 Our approach and contributions

Differential privacy (DP) is usually achieved through randomization. This makes it natural to consider randomized algorithms, such as SD-Gibbs [16], which also, at the same time, are much more computationally efficient. However, these algorithms by themselves do not protect privacy, and we develop additional mechanisms to ensure DP of the entire process. More concretely, we consider the following approach to design a scalable DCOP that preserves constraint privacy.

Identifying Privacy Leaks in SD-Gibbs. We first show that SD-Gibbs may leak information about agent constraints during execution. More concretely, during the algorithm's execution, agents send and receive information that directly depends on their utility functions, i.e., functions that quantify the preferences for each constraint. What is more, SD-Gibbs' iterative nature may further lead to a high privacy loss over the iterations.

As such, we are required to construct an algorithm that not only preserves constraint privacy but one which incurs minor privacy leaks across iterations.

P-Gibbs. Towards this, we develop a new differentially private variant of SD-Gibbs. We present a novel algorithm *P-Gibbs*: which crucially differs from SD-Gibbs in three key aspects with respect to preserving constraint privacy.

1. *Sampling through Soft-max with Temperature.* Sampling through Gibbs distribution [22] in SD-Gibbs leaks information about the underlying utilities. This leak is because a value with greater utility is more likely to be sampled. To overcome this, we use *soft-max with temperature* over the Gibbs distribution. This process smooths out sampling distributions in SD-Gibbs.
2. *Adding Gaussian Noise to Relative Utilities.* Each agent in SD-Gibbs sends its relative utility to its immediate parent. These relative utilities are the difference between its previous assignment and its current one. As such, these values leak vital information about the utilities. E.g., if a particular assignment has a high utility for agent j, but low for others (and it is known), an intermediate agent will learn about agent j even from the aggregated utility. To this end, we add *Gaussian* noise to the relative utility in our algorithm. The added noise helps to perturb each agent's relative utilities such that information regarding the utilities is protected.
3. *Subsampling*. As aforementioned, the iterative nature of SD-Gibbs implies that the privacy loss is accumulated over the iterations. To limit this loss, we propose that each agent must sample a new assignment with a subsampling probability q. This limits the information being leaked at each iteration, resulting in bounded privacy loss.

In addition, we provide a refined analysis of privacy within the framework of (ϵ, δ)-DP for P-Gibbs. We simulate P-Gibbs on three benchmark problems in DCOP literature, namely Ising [23], graph-coloring, and meeting-scheduling [24]. Our experiments demonstrate our novel algorithm's practicality and robust performance for a reasonable *privacy budget*, i.e.,

Table 1 Comparing existing literature in privacy-preserving DCOPs with our novel privacy variant, P-Gibbs

Algorithm	Complete	Privacy agent	Topology privacy	Constraint privacy	Decision privacy	Solution privacy	Collusion resistance	No privacy overhead
P-DPOP [3, 17]	✓	✓	∘	∘	∘	✗	✗	✗
$P^{3/2}$-DPOP [17]	✓	✓	∘	∘	✓	✗	✗	✗
P^{2}-DPOP [17]	✓	✓	∘	✓	✓	✗	✗	✗
P-SyncBB [19]	✓	✗	✓	✓	✓	✗	✗	✗
P-Max-Sum [18]	✗	✗	✓	✓	✓	✗	✗	✗
P-RODA [27]	✗	✗	✓	✓	✓	✗	✗	✗
PC-SyncBB [28]	✓	✗	✗	✓	∘	✗	✓	✗
MD-Max-Sum [29]	✗	✗	✓	✓	✓	✗	✓	✗
P-Gibbs	✗	✓†	✓†	✓‡	✗	✓	✓	✓

†: P-Gibbs can support agent and topology privacy through anonymous communication (e.g., using codenames [3]), ‡: Differentially-private guarantee

Here, "✓"denotes the realization of the property, "∘"that the property is realized partially, and "✗"if the property is not realized. Note that the rest of the algorithms provide a cryptographic guarantee outside of P-Gibbs

ϵ, with SD-Gibbs as the baseline. Specifically, we show that P-Gibbs provides only a marginal drop in solution qualities than SD-Gibbs for a desirable privacy budget.

1.3 Paper overview

The paper structure is as follows. In Sect. 2, we place P-Gibbs concerning the privacy-preserving DCOP literature. We formally introduce DCOP, describe SD-Gibbs, and define differential privacy (DP) in our context in Sect. 3. We illustrate the nature of privacy leaks in SD-Gibbs with Sect. 4. Section 5 introduces our novel privacy variant P-Gibbs, including a refined analysis of (ϵ, δ)-DP. Next, in Sect. 6, we empirically validate P-Gibbs over several problem instances of benchmark problems in DCOP literature. Our experiments highlight our privacy variant's efficiency. Section 7 concludes the paper along with a discussion on future research directions.

2 Related work

This section places our work concerning the general DCOP literature, focusing on privacy-preserving DCOPs. Table 1 compares the works described in this section with our novel privacy variant, P-Gibbs, regarding their privacy guarantees and scalability.

2.1 Distributed constraint optimization problem (DCOP)

As aforementioned in Sect. 1, despite the computationally hard nature of DCOPs, researchers have proposed various algorithms that aim to solve them either completely or approximately. Outside of the popular algorithms like DPOP [11], ADOPT [7], SyncBB [8], and Max-Sum [12], the field has also seen some recent sampling-based algorithms. Details follow.

Ottens et al. [13] propose Distributed Upper Confidence Tree (DUCT), an extension of UCB [25] and UCT [26]. 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. [16] 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 [22]. 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 empirically show 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 work, we focus on SD-Gibbs. Our results can be trivially extended for PD-Gibbs.

SD-Gibbs [16] is a *sampling-based* algorithm in which the authors use the Gibbs distribution [22] to solve DCOPs. The algorithm can be broadly categorized into the following four phases. *(i)* Initialization: Each agent initializes its algorithm-specific variables. *(ii)* Sampling: Agents sample an assignment to their variable based on the Gibbs distribution and depending on the assignments of their neighboring agents. *(iii)* Backtracking: After each agent has sampled its assignment, they calculate their *relative utilities*. That is, the difference between their previous assignment with their current assignment. The agents then send the utilities to their immediate parents. The parents add their utilities to the ones received, and the process continues till the root agent. This concludes one *iteration*. *(iv)* Deriving Solution: The backtracking process results in the root agent holding the global relative utility. Based on the solution observed thus far, the root throws away or keeps the solution.

In this paper, we focus on SD-Gibbs due to its improved performance in terms of solution quality and computational efficiency.

2.2 Privacy in DCOPs

The existing literature focuses on the following techniques to ensure privacy in DCOPs.

2.2.1 Achieving privacy through cryptosystems

Privacy in DCOPs has focused on using cryptographic primitives, such as *partial homomorphic encryption* and *secret-sharing methods*. To quantify the privacy guarantees, researchers propose the following four notions.

1. *Agent privacy* [3, 17]. No agent must learn the existence of its non-neighboring agents, i.e., agents it does not share constraints with.

2. *Topology privacy* [3, 17]. No agent must discover the existence of topological constructs in the constraint graph, such as nodes (i.e., variables), edges (i.e., constraints), or cycles, unless it owns a variable involved in the construct.
3. *Constraint privacy* [3, 17]. No agent should be able to discover the nature of a constraint that does not involve a variable it owns.
4. *Decision privacy* [3, 17]. No agent should be able to discover the value that another agent's variable takes in the solution chosen for the problem (modulo semiprivate information).

Several privacy-preserving algorithms exist, using *secure multi-party computation* [30] atop existing DCOP algorithms to provide cryptographic privacy guarantees. These include P-DPOP [3], $P^{3/2}$-DPOP [17] , P^2-DPOP [17], which builds on the DPOP algorithm; P-SyncBB [19] and PC-SyncBB [28] over SyncBB; P-Max-Sum [18] over Max-Sum; and P-RODA [27] which is privacy variant for algorithms which fit in Region Optimal DCOP Algorithm (RODA) [27] framework. In Table 1, we provide the known private algorithms and the privacy notions they satisfy; details follow.

To guarantee agent and (partial) topology privacy, the algorithms P-DPOP, $P^{3/2}$-DPOP, and P^2-DPOP use "codenames" (randomly generated numbers) in place of the actual variable names and domains. These codenames are used for information exchange between agents. The agent selected as the root then "decrypts" these values to arrive at the solution. In contrast, P-Max-Sum and PC-SyncBB support topology privacy. These algorithms also use information-hiding public-key encryption and random shifts and permutations.

P^2-DPOP completely preserves constraint privacy. The algorithm uses the partial homomorphic property of ElGamal encryption for the same. This technique is unlike P-DPOP and $P^{3/2}$-DPOP, which merely adds the random numbers communicated by agents, inadvertently leaking privacy. P-Max-Sum also preserves constraint privacy by communicating information through encryptions or random shares.

DCOPs are also solved using region-optimal algorithms such as KOPT [31] or DALO [32]. Grinshpoun et al. present an umbrella setup, namely RODA, that generalizes these region-optimal algorithms. The authors present P-RODA [27], which implements the privacy-preserving implementations of these region-optimal algorithms. P-RODA uses cryptographic primitives such as secret sharing and homomorphic encryptions. As such, P-RODA perfectly simulates RODA but with a significant computational overhead.

The algorithms mentioned above, except PC-SyncBB, assume that agents do *not* collude. Note that any two or more colluding agents can leak sensitive information about the other agents. Using secure multi-party computation, Tassa et al. [28] show that PC-SyncBB is *collusion resistant* as long as the majority of the agents do not collude. Most recently, Kogan et al. [29] introduced MD-Max-Sum, a privacy-preserving, collusion-resistant DCOP algorithm built atop the Max-Sum algorithm. Crucially, MD-Max-Sum uses third parties, namely *mediators*, to guarantee collusion resistance and has a reduced run time compared to PC-SyncBB. The algorithm satisfies constraint, topology, and decision privacy.

2.2.1.1 Solution Privacy Research on privacy-preserving algorithms for DCOP typically focuses on complete algorithms guaranteed to compute the optimal assignment (solution) [3, 17]. Obviously, one cannot keep the solution secret, so the information leaked by knowledge of the solution has generally been considered an inevitable privacy

loss. Moreover, as the optimization outcome cannot be preserved, the computation may compromise agents' preferences, thereby violating constraint privacy.

However, it is possible to make the solution, and therefore any information that can be inferred from it differentially private. We call this property *solution privacy* and add it as an additional objective for privacy-preserving DCOP. We show that our differentially private variant, P-Gibbs, satisfies solution privacy through randomization of the computation process (Table 1).

Solution Privacy and Decision Privacy. In this paper, we follow the classic security principle – "no security through obscurity"– meaning that we cannot assume privacy would be kept by simply hiding decisions from some agents (the server or other agents might reveal them; agents who get your decision can be malicious; decisions may be observed through agents actions by outsiders; and so on). Thus, in our context, the privacy notions of solution privacy and decision privacy are not equivalent.

2.2.1.2 Non-scalability of Existing Private DCOP Algorithm Unlike our privacy variant P-Gibbs (Table 1), cryptographic primitives and the computationally expensive nature of DCOPs results in the algorithms mentioned above not being *scalable* in terms of the number of agents and constraints. More concretely, we say that a private DCOP algorithm admits a *privacy overhead* if it is significantly more computationally expensive compared to its non-private variant.

Our definition implies that private algorithms based on cryptographic primitives incur a significant privacy overhead. E.g., P-DPOP [3] scales up to 12 agents for graph-coloring and 6 agents for meeting-scheduling. Such a lack of scalability is also present in privacy-preserving algorithms built atop approximate algorithms. E.g., P-Max-Sum's run-time increases three-fold in magnitude over its non-private variant [18].

2.2.2 Other privacy notions

2.2.2.1 Information Entropy In a parallel line of work, the authors in [33] use *information entropy* to quantify the privacy loss incurred by an algorithm in solving a distributed constraint problem. The result is later furthered by [34, 35]. Grinshpoun et al. [35] present private local-search algorithms based on the algorithms above. The authors use this quantification to show that their algorithms provide high-quality solutions while preserving privacy. While the privacy loss metric defined in [33] is interesting, it does not offer a worst-case guarantee. Practically, even a minor leak may result in information being revealed completely.

2.2.2.2 Utilitarian DCOPs Savaux et al. [36, 37] propose *Utilitarian* DCOPs (UDCOPs) where privacy leakage is correlated to the quality of the final assignment. They assume that each agent also maintains a privacy cost for each assignment's utility which captures the desire of the agent to preserve that utility's privacy. With this modeling, they can derive the overall privacy cost along with the final solution.

The authors introduce private DCOP algorithms based on this idea (e.g., DBOU and DSAU, which are extensions of DBO and DSA, respectively). The key privacy idea in these algorithms is that agents randomly sample new assignments and only broadcast the information if it positively changes their overall utility.

While such a utility-based privacy cost is another interesting way of quantifying privacy leaks in DCOPs, we believe a (ϵ, δ)-DP approach is a more robust measure of the same. First, the privacy budget used in UDCOPs appears to be agent-specific (i.e., agents may define it in an arbitrary fashion). As such, it may not be applicable in practice as the agents may find it difficult to quantify the privacy cost of revealing information about a certain resource. Furthermore, even if an agent can estimate its cost at one point, privacy implications can change with time. That is, the obtained solution quality may not be useful in the future. In contrast, there is a clear consensus on appropriate values of ϵ and δ in DP, implying quantifiable privacy guarantees.

Second, to the best of our knowledge, while we provide worst-case privacy guarantees for P-Gibbs, similar to the information entropy-based privacy measure, there are no worst-case guarantees for the algorithms in [36, 37]. So, even if we disregard the arbitrary privacy cost assignments by each agent, it is not possible to say if the solution reveals something about the true utilities. And if there is a correlation between privacy costs and utilities, it might even reveal more.

3 Preliminaries

This section formalizes DCOPs, summarizes SD-Gibbs, and defines privacy definitions relevant to our work.

3.1 Distributed constraint optimization problem (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*.

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 $F_{ij} : D_i \times D_j \rightarrow \mathbb{R}$. 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}.
- $\alpha : \mathcal{X} \rightarrow \mathcal{A}$ maps each variable to one agent.

In this work, w.l.o.g [38], we assume that $p = m$, i.e., the number of agents and the number of variables are equal. We also assume $D = D_i = D_j$, $\forall i, j$, 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[1] the total utility, i.e., $F(\mathbf{X}^*) = \max_{\mathbf{X} \in \mathcal{D}} F(\mathbf{X})$.

In DCOPs, a combination of variables (or alternately, 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. The other edges are referred to as *back edges*. Nodes connected via a tree edge are referred to as parent–child nodes. Likewise, back edges connect the pseudo-parent and its pseudo-children. Such an arrangement can be constructed using a distributed-DFS [39].

For the algorithms presented in this paper, let N_i refer to the set of neighbors of x_i in CG. Also, let $\mathcal{C}_i$ 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.

Furthermore, this paper specifically focuses on constraint privacy, formally defined in our context next.

Definition 3 (Constraint Privacy) Given a DCOP defined by $\langle \mathcal{X}, \mathcal{A}, \mathcal{D}, \mathcal{F}, \alpha \rangle$, constraint privacy implies that an agent i learns no information regarding the utility function $\{F_{jk}\}_{k \in N_j}$ of any agent $j \in \mathcal{A} \setminus N_i$. That is, for any agent, it does not share a constraint with.

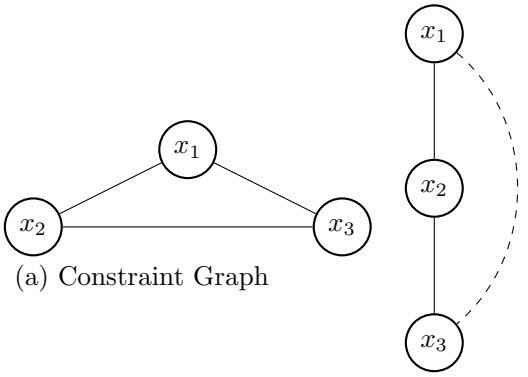
3.1.1 Example

Consider the maximization problem depicted in Fig. 1. Here, $|\mathcal{X}| = |\mathcal{A}|$ s.t. $\mathcal{X} = \{x_1, x_2, x_3\}$ and $D_1 = D_2 = D_3 = \{0, 1\}$. The constraint graph and one possible pseudo tree configuration are presented in Figs. 1(a) and 1(b). In Fig. 1(b), the solid edges represent the tree edges, while the dashed edges represent the back edges. Fig. 1(c) shows the utility function that is identical for $F_{12} = F_{23} = F_{13}$. For this example, a solution is as follows. The optimal assignment is $x_1 = 1, x_2 = 1$, and $x_3 = 1$, with the overall utility 6.

3.2 Sequential distributed Gibbs (SD-Gibbs)

We now describe Sequential Distributed Gibbs (SD-Gibbs) as first introduced in [16]. 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 function, represented by $\theta_{ij}(x_i, x_j)$, is

[1] We can also define a DCOP that minimizes the total utility, i.e., $F(\mathbf{X}^*) = \max_{\mathbf{X} \in \mathcal{D}} -F(\mathbf{X})$.



(a) Constraint Graph

(b) Pseudo Tree

x_i	x_j	F_{ij}
0	0	0
0	1	-1
1	0	-1
1	1	2

(c) Utilities (where $i < j$)

Fig. 1 DCOP Example

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:

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

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

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

3.2.1 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,

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

Let, $\mathbb{P}_i(\mathbf{x}_i) = \{\Pr(x_i | x_j \in \mathcal{X} \backslash \{x_i\}) | x_i = d_i \ \forall d_i \in D_i\}$. That is, $\mathbb{P}_i$ represents SD-Gibbs' probability distribution of each agent i. The relevant notations required for the SD-Gibbs algorithm are presented in Table 2.

Table 2 Variables maintained by each agent x_i in SD-Gibbs

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

Algorithm 1 Sequential Distributed Gibbs [16]

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

Procedure 1 INITIALIZE() [16]

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 if**

Procedure 2 SAMPLE() [16]

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($x_s, d_s, \bar{d}_s, t^*_s, \bar{t}^*_s$) [16]

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

Procedure 4 BACKTRACK($x_s, \Delta_s, \bar{\Delta}_s$) [16]

$\Delta_i \leftarrow \Delta_i + \Delta_s$; $\bar{\Delta}_i \leftarrow \bar{\Delta}_i + \bar{\Delta}_s$
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 if
        SAMPLE()
    end if
end if

3.2.2 Algorithm

Table 2 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 a construction of the pseudo-tree and each agent initializing each of their variables, from Table 2 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 executed Procedure 2.
3. Each leaf agent j then sends a BACKTRACK message to its parent comprising x_j, Δ_j, and $\bar{\Delta}_j$. As described in Procedure 4, when a parent receives such a message, it sends a BACKTRACK message to its parent. The process continues until the root receives the message – concluding one iteration.
4. Each agent i uses its current (Δ_i) and current best-response ($\bar{\Delta}_i$) local utility differences to reach a solution. 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 (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. [3], constraint privacy states that no agent must be able to discover the nature of constraint (i.e., the utilities) that does not involve a variable it owns. Since absolute privacy is not an achievable goal [40], we formalise constraint privacy in terms of (ϵ, δ)-DP [21].

3.3 Differential privacy (DP)

Differential Privacy (DP) [20, 21] is a popular privacy notion that aims to provide a statistical guarantee against a database that the inclusion or exclusion of any single entry will not significantly impact the results of the statistical analysis. The guarantee makes it difficult for an adversary to infer sensitive information about specific individuals present in the database from the said statistical analysis outcome. More concretely, consider any pair of *adjacent* databases, i.e., databases differing in a single entry. We say that a randomized mechanism $\mathcal{M}$ is (ϵ, δ) differentially private if the ratio of the probabilities between adjacent databases as inputs on $\mathcal{M}$ is upper-bounded by ϵ with probability $1 - \delta$. Here, we have $\epsilon > 0$ and $\delta \in [0, 1)$. The smaller the value of ϵ, the higher the privacy protection. Furthermore, the lower the value of δ, the lower the probability of privacy failure.

3.3.1 DP: applications

One of the foremost applications of DP is towards private query releases [21]. Here, the goal is to provide answers to user queries in a differentially private manner. DP has also emerged as the gold standard of privacy in the AI/ML literature. For instance, DP is used to protect user's sensitive information in ML. DP-variants exist for SVMs [41], PCA [42], Multi-armed Bandits (MABs) [43] as well as deep learning-based techniques including DP-SGD [4] and PATE [44]. The mechanism design literature also uses DP to guarantee the privacy of an agent's private information [45, 46]. McSherry and Talwar [47] present

the first such mechanism that uses DP to design an approximately truthful digital goods auction.

3.3.2 DP: our setting

As stated, DP is normally defined for adjacent databases. However, in this instance, not only do we want to protect privacy against external adversaries but also against curious fellow agents, i.e., agents looking to decipher sensitive information. 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 [48]. 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 about agents it shares constraints with and nothing about the rest of the problem.

3.3.3 Local DP

As a result, we consider the *local* model of differential privacy [21]. 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 [21],

Definition 4 (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,

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

Definition 4 states that for any two pairs of inputs, F and F', the output of the randomized mechanism $\mathcal{M}$ is similar up to a factor ϵ with probability at least $1 - \delta$. The smaller the ϵ, the more similar the outputs. In other words, the smaller the ϵ, the harder the adversary finds it to distinguish between F and F'. The typical δ values are of the order $\mathcal{O}(1/m)$ or $\mathcal{O}(1/m^2)$ [21], where m denotes the number of records. In our setting, m denotes the number of variables or agents.

3.3.4 Privacy Loss

We now present another way of interpreting the local-DP definition defined in Eq. 3. For this, we define the privacy loss random variable (PRVL) L, as follows:

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

Now, we can say that a randomized mechanism $\mathcal{M}$ satisfies (ϵ, δ)-DP if with probability at least $1 - \delta$, we have $L^{o}_{\mathcal{M}(F)||\mathcal{M}(F')} \leq \epsilon$ [21]. The PRVL L is often used to analyze the ϵ and δ guarantees of a DP mechanism, e.g., the famous Gaussian mechanism [21] as discussed next.

3.3.5 Gaussian mechanism

As the name suggests, the Gaussian mechanism [21] privatizes a statistic by adding noise sampled by the Gaussian distribution to the statistic analysis outcome. More concretely, given the PDF of the Gaussian distribution $\mathcal{N}(0,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right)$, the Gaussian mechanism defined by:

$$\mathcal{M}(F) \triangleq M(F) + Y,$$

is (ϵ, δ)-DP where $M(F)$ is the (non-private) outcome of the statistic analysis and $Y \sim \mathcal{N}\left(0, 2\ln(\frac{1.25}{\delta})\frac{\tau^2}{\epsilon^2}\right)$ is the Gaussian noise added [21]. The term τ denotes the *sensitivity*, which is the measure of how much the outcome of the statistic analysis changes in response to the addition or removal of a single data entry. In Sect. 5.3, we re-look at the Gaussian mechanism in our context.

4 Privacy leakage in SD-Gibbs

In SD-Gibbs, constraint privacy is compromised in the following two ways:

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 (Sect. 5.2).

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 is 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. We next illustrate the same.

4.1 Illustrating Privacy Leak in SD-Gibbs on Figure 1

Consider the execution of SD-Gibbs (Algorithm 1) on the example provided in Fig. 1. Recall that we aim to preserve constraint privacy in DCOP. The agent x_1 must not learn anything about the nature of the constraint between the agents x_2 and x_3.

Upon execution of Algorithm 1 with each variable initialized with 0, the initial Gibbs distribution for x_2 and x_3 (from Eq. 2) takes the form [0.88, 0.12] and [0.88, 0.45], respectively. As x_1 is the neighboring agent for both x_2 and x_3, it will be aware of their current and best assignments. Moreover, as the number of iterations increases, x_1 observes that its Gibbs distribution converges to [0.002, 0.998]. Further, one can also see that x_2's and x_3's Gibbs distribution changes to [0.002, 0.998]. That is, x_1 can observe that x_2 and x_3 prefer assignment 1 (with high probability) given its context.

Based on the assignment sampling, x_1 already has a qualitative idea of the nature of the constraint x_2–x_3. Since it knows it prefers assignment 1, it can estimate the constraint x_2–x_3 will be such that it is less or equal in value—for any assignment other than their current $x_2 = x_3 = 1$. If not, x_2 and x_3 would have changed their assignments to grab the additional utility.

To get a quantitative estimate, x_1 can observe the relative utilities. In our example, as there are only three constraints, x_1 can use the information on the probable assignments of x_2 and x_3, 1 and 1, and the final utility 6 to derive the value $F_{23}(x_2 = 1, x_3 = 1) = 6 - F_{12}(x_1 = 1, x_2 = 1) - F_{13}(x_1 = 1, x_3 = 1) = 6 - 2 - 2 = 2$. Therefore, x_1 can learn information regarding the constraint F_{23} violating constraint privacy. Applying similar qualitative knowledge of the assignment, on each iteration's Δs, can potentially leak information of the entire utility function.

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.

5 P-Gibbs: preserving constraint privacy in DCOP with SD-Gibbs

In general, for DP, we need to ensure full support of the outcome distribution. If $\Pr[\mathcal{M}(D') = o] = 0$ for some output o, the privacy loss L incurred is infinite, and one cannot bound ϵ. For the specific context of ensuring constraint privacy in SD-Gibbs, this 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 $|F_{ij}(\cdot,\cdot)| > 0, \forall i$. Without these, the probability distributions defined in Eq. (2) may not be bounded as any pair of agents i and j ($i \neq j$) may have $D_i \neq D_j$. As a result, the ϵ with respect to constraint privacy (Definition 3) will not be defined. Formally, consider the following claim.

Claim 1 *With respect to constraint privacy, SD-Gibbs (Algorithm 1) is non-private, i.e., the privacy loss variable L is not defined for SD-Gibbs.*

Proof Consider any two agents $i, j \in \mathcal{A}$ s.t. ($i \neq j$) and $D_i \neq D_j$. W.l.o.g., let $D_i = D_j + \{d\}$. From Definition 3, the privacy loss variable L (Eq. (4)) can be written as,

$$L^d_{\mathbb{P}_i||\mathbb{P}_j} = \ln\left(\frac{\mathbb{P}_i(x_i = d)}{\mathbb{P}_j(x_j = d)}\right) = \ln\left(\frac{v}{0}\right) \text{ where } v > 0,$$

as $d \in D_i$ while $d \notin D_j$. Thus, the privacy loss variable, L, is not defined for SD-Gibbs. □

Claim 1 implies that the privacy budget, ϵ in $(\epsilon, 0)$-DP, is also not-defined for SD-Gibbs. Consequently, to provide meaningful privacy guarantees for constraint privacy in DCOPs, we present P-Gibbs (Sect. 5). In it, we first use soft-max with temperature to bound the SD-Gibbs distributions (Sect. 5.2). 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 (Sect. 5.3). Lastly, to reduce

[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.

Table 3 Notations

Symbol	Definition
τ	Sensitivity
c	Clipping constant
$\mathbb{P}_i$	SD-Gibbs probability distribution for Agent i
p_i	Soft-max function on $\mathbb{P}_i$ for Agent i
γ	Soft-max temperature parameter
q	Sub-sampling probability
T	Number of iterations in P-Gibbs
$\mathcal{N}(0, \sigma^2)$	Gaussian distribution with mean zero and variance σ^2
(ϵ_s, δ)	Privacy parameters of the Sampling stage
(ϵ_n, δ)	Privacy parameters for the Relative utilities

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). Table 2 and Table 3 provide a reference point for the notations used in this section.

Procedure 5 P-Gibbs SAMPLE()

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

5.1 P-Gibbs: algorithm

Recall that privacy leak in SD-Gibbs is due to the qualitative and quantitative information loss due to communicating the sampled value d and the relative utilities Δ, respectively (Sect. 4). Our privacy variant, P-Gibbs, preserves this information loss through its novel sampling procedure. We formally provide the sampling in P-Gibbs with Procedure 5. The differences, compared to SD-Gibbs' sampling procedure, are summarized as follows:

1. To preserve constraint privacy loss due to sampling (Procedure 5, Lines 3–9):
   - P-Gibbs uses soft-max function over SD-Gibbs distributions for sampling d_i's, $\forall i$. As shown later in Proposition 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 (Procedure 5, Lines 13–16):
   - In P-Gibbs, we sanitize the relative utilities with calibrated *Gaussian Noise*.
   - To bound the sensitivity (see Sect. 5.3), we "clip" the relative utilities by $\pm c$, where c is the *clipping constant* (Procedure 5, Lines 16 and 17).

In the next 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.

5.2 P-Gibbs: 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.,

$$p_i(\mathbf{x}_i, \gamma) = \left\{ \frac{\exp\left(\mathbb{P}_i(x_i = d_k)/\gamma\right)}{\sum_{d_l \in D} \exp\left(\mathbb{P}_i(x_i = d_l)/\gamma\right)}; \forall d_k \in D \right\} \tag{5}$$

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. $x_i = d_k, \forall d_k \in D$. This observation ensures that the scenario of an unbounded privacy loss due to $p_i(x_i, \gamma) = 0$, described earlier with Claim 1, will not occur for P-Gibbs.

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 $1/e \leq \exp(p_i(x) - p_j(x)) \leq e$.

Proposition 1 *For two probability distributions using soft-max, p_i and p_j defined by Eq. (5), 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 Because p_i and p_j are soft-max distributions, we have,

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

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

$$\ln\left[\frac{p_i(x_i = d, \cdot)}{p_j(x_j = d, \cdot)}\right] \leq \ln\left[\frac{\frac{\exp\left(\mathbb{P}_i(x_i=d)/\gamma\right)}{\sum_{d_l \in D} \exp\left(\mathbb{P}_i(x_i=d_l)/\gamma\right)}}{\frac{\exp\left(\mathbb{P}_j(x_j=d)/\gamma\right)}{\sum_{d_l \in D} \exp\left(\mathbb{P}_j(x_j=d_l)/\gamma\right)}}\right] \leq \ln\left[\frac{\exp\left(1/\gamma(\mathbb{P}_i - \mathbb{P}_j)\right))}{N_1/N_2}\right] \tag{6}$$

Here, $N_1 = \sum_{d_l \in D} \exp\left(\mathbb{P}_i(x_i = d_l)/\gamma\right)$ and $N_2 = \sum_{d_l \in D} \exp\left(\mathbb{P}_j(x_j = d_l)/\gamma\right)$. Now, in Eq. (6) observe that both the numerator and denominator in r.h.s of Eq. (6) are positive. Further, as $\ln(x)$ is an increasing function x, this implies that r.h.s of Eq. (6) is maximum when the numerator is maximum, and the denominator is minimum. Thus 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 Eq. (6) is at-most $\exp(1/\gamma) = e^{1/\gamma}$.

The denominator in r.h.s of Eq. (6) 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 $N_2 = |D| \cdot e^{1/\gamma}$. Using these values in Eq. (6) completes the claim. □

5.2.1 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 Eq. (5), 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 Proposition 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 Proposition 1 is desirable and hence use it to construct the sampling distribution in P-Gibbs.

5.2.2 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 [4] for non-Gaussian mechanisms. Following derivations by [49],

$$\Pr[L \geq \epsilon_s] \leq \max_{p_i, p_j} e^{\lambda \mathcal{D}_{\lambda+1}[p_i||p_j] - \lambda\epsilon_s} \tag{7}$$

Here, L is the privacy loss between any two agents and $\mathcal{D}_\lambda(p_i||p_j) = \frac{1}{\lambda-1}\log \mathbb{E}_{d\sim p_j}\left(\frac{p_i(d)}{p_j(d)}\right)^\lambda$ is Rényi divergence [50] of order $\lambda \in \mathbb{N}^+, \lambda > 1$. From [49], the choice of the hyperparameter λ is arbitrary since the bound holds for any feasible value of λ. Note that the value of λ determines how tight the bound is.

Also from [49], we borrow the notion of *privacy cost* $c_t(\lambda)$. By trivial manipulation, for each iteration t,

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

where Eq. (8) is due to monotonicity $\mathcal{D}_\lambda(P||Q) \leq \mathcal{D}_{\lambda+1}(P||Q) \leq \mathcal{D}_\infty(P||Q),\ \forall \lambda \geq 0$.

Subsampling. The privacy cost c_t in Eq. (8) can be further reduced by subsampling agents with probability $q << 1$. Balle et al. [51] show that the privacy guarantees of a DP mechanism can be amplified by applying the mechanism to a small random subsample of records of any database.

Reproducing the steps of the sampled Gaussian mechanism analysis by [49] 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*

$$c_t^{(s)}(\lambda) = \ln \mathbb{E}_{k\sim B(\lambda+1,q)}\left[e^{k\cdot 2/\gamma}\right], \tag{9}$$

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 [49, Theorem 3]. □

Unlike the analysis in [49, 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 2 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.

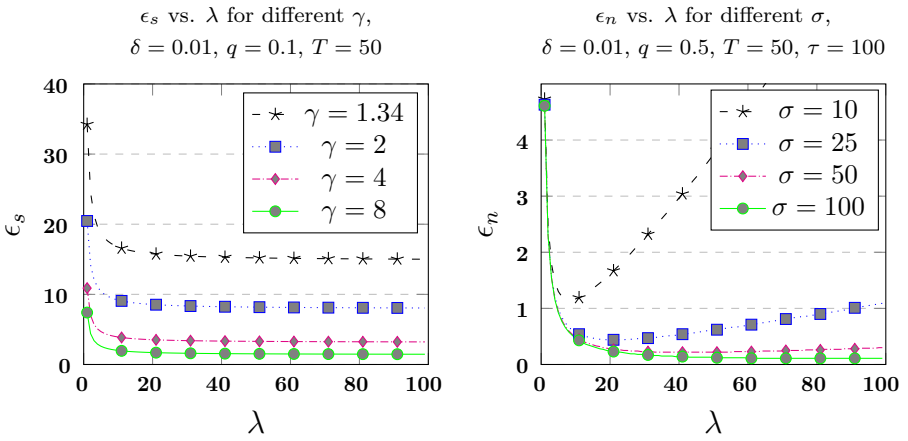


Fig. 2 Variation of ϵ_s and ϵ_n vs. λ

5.2.3 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 Proposition 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 a *uniform* distribution, as more information on SD-Gibbs sampling distribution is lost. To distinguish this extreme case, we refer to P-Gibbs with $\gamma \to \infty$ as P-Gibbs$_\infty$.

5.3 P-Gibbs: privacy of relative utilities (Δ)

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. In order to achieve DP for these 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 5 (Sensitivity (τ)) It is the maximum absolute difference between any two relative utility values Δ and Δ', i.e.,

$$\tau = \max_{\Delta, \Delta'} |\Delta - \Delta'| \tag{11}$$

As we clip the relative utilities with a constant c (see Procedure 5, Line 16–17), trivially from Eq. 11, $\tau = 2 \cdot c$.

Table 4 Per-iteration and final (ϵ, δ) bounds

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

Next, we must sanitize the relative utilities so as to preserve privacy fully. We achieve this through the *Gaussian noise mechanism* (Sect. 3.3 and [21]) 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 [21, Theorem A.1] or the moments accountant [4]. The latter can be unified with the accounting for the sampling stage by using:

$$c_t^{(n)}(\lambda) = \ln \mathbb{E}_{k\sim B(\lambda+1,q)}\left[e^{k\mathcal{D}_{\lambda+1}[\mathcal{N}(0,\tau^2\sigma^2)||\mathcal{N}(\tau,\tau^2\sigma^2)]}\right]. \tag{12}$$

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 Fig. 2, 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 [49, 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 the procedures are the same as provided with Algorithm 1. Table 4 summarises expressions for per-iteration and total ϵ values for P-Gibbs and P-Gibbs$_\infty$.

5.3.1 Collusion Resistance

Recall that in the private DCOP literature, an algorithm is collusion resistant if *no* subset of agents can collude to gain additional information about the remaining agents. We remark that P-Gibbs trivially satisfies collusion-resistance. This is because each agent in P-Gibbs locally adds noise or randomness to its utility and assignment sampling. Due to the post-processing property of DP, no subset of the agent will be able to infer any additional information outside of the (ϵ, δ)-DP guarantee.

6 Experiments

We now empirically evaluate the performance of our novel algorithm, P-Gibbs w.r.t. to SD-Gibbs. This section first describes our experimental setup and benchmark problems (Sect. 6.1). Next, in Sect. 6.2, we present the results for P-Gibbs' performance in terms of solution w.r.t. SD-Gibbs. Section 6.3 presents criteria to empirically explain the privacy protection in P-Gibbs with regard to changes in the privacy budget. Section 6.4 provides a general discussion of the results presented and summarizes the advantages of our DP-based approach compared to the existing cryptographic approach for privacy-preserving DCOP algorithms.

6.1 Benchmark problems and experimental setup

We now describe the DCOP benchmark problems and illustrate our experimental setup.

6.1.1 Benchmark problems

We construct the following problem instances to test our novel differentially private variant, P-Gibbs. These are standard benchmarks in the DCOP literature.

6.1.1.1 Ising [23] We generate 20 sample Ising problems. For this, the constrained graph is a rectangular grid with each agent/variable connected to its four nearest neighbors. The problems are such that the number of agents/variables lie between [10, 20). Each agent's domain is binary, i.e., $D_i = \{0, 1\}, \forall i$. The constraints are of two types: (i) binary constraints whose strength is sampled from $\mathcal{U}[\beta, \beta]$ where $\beta \in [1, 10)$ and (ii) unary constraints whose strength is sampled from $\mathcal{U}[-\rho, \rho]$ where $\rho \in [0.05, 0.9)$. Ising is a *minimization* problem.

6.1.1.2 Graph-Coloring (GC) We generate 20 sample graph-coloring problems. The problems are such that the number of agents/variables lies between [30, 100) and agents' domain size between [10, 20). Each constraint is a random integer taken from (0, 10). Graph-coloring is a *minimization* problem.

6.1.1.3 Meeting-Scheduling (MS) [24] We generate 20 sample meeting-scheduling problems. The problems are such that the number of agents and variables lies between [1, 75) with the number of slots, i.e., the domain for each agent randomly chosen from [30, 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.

While meeting-scheduling is a concrete problem [24], even abstract problems like graph-coloring can model real-world scenarios. E.g., Radio Frequency or Wireless Network Assignment can be modeled as a graph-coloring problem [52]. The Ising model is also a widely used benchmark in statistical physics [23].

Importantly, we perform our experiments on much larger problems than earlier complete algorithms (e.g., [3]) can handle.[3] Concerning the infeasibility of a DCOP solution,

[3] For e.g., DPOP, a non-private, complete algorithm timed-out after 24 h of computing (i) an Ising instance with 10 variables, (ii) a graph-coloring instance with 12 variables and $|D| = 8$, (iii) a meeting-scheduling instance with 25 variables and $|D| = 20$. For details, refer to Appendix 1.

we remark that incomplete (or random) algorithms like MGM, DUCT, and SD-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 [16], we focus on soft constraints; thus, infeasible solutions will not occur.

6.1.2 Experimental setup

Our experimental setup is as follows.

Implementation. pyDCOP [53] 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, and metrics collection, among others. 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. With this, we generate instances for our benchmark problems, i.e., Ising, graph-coloring, and meeting-scheduling. 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 10 times for a fair comparison and use the subsequent average utility for our results.

The complete codebase is available at: github.com/magnetar-iiith/PGiBBS.

Performance Measure. We measure P-Gibbs' performance w.r.t. SD-Gibbs using the following performance measure.

Definition 6 (Solution Quality (SQ)) Solution quality $\text{SQ}_{\mathcal{A}}$ of an algorithm $\mathcal{A}$ is defined as

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

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. $\text{SQ} \approx 1$ indicates that utility does not deteriorate than SD-Gibbs. On the other hand, $\text{SQ} \approx 0$ means little utility as compared to the SD-Gibbs solution. It is possible that $\text{SQ} > 1$ due to randomness and privacy noise acting as simulated annealing[4] [56].

[4] We remark that this behavior is different from the Distributed Simulated Annealing (DSAN) algorithm for DCOPs [54, 55]. DSAN is an iterative optimization algorithm with a *temperature* parameter that aims to control the likelihood of accepting worse solutions. DSAN consists of an annealing schedule that determines the change in the temperature parameter over time. As the parameter decreases, DSAN becomes more selective and explores the solution space more effectively. Instead of selecting the next assignment through a specific, utility-based distribution like in SD-Gibbs (Eq. 2), in DSAN, an agent randomly chooses its next assignment. E.g., by uniform sampling or by swapping values with neighboring agents. DSAN is neither complete nor private.

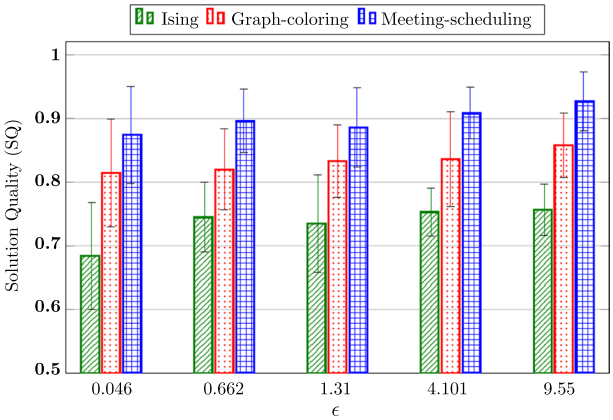


Fig. 3 The average and standard deviation of P-Gibbs' Solution Quality (SQ) for different privacy budgets. Note that, the case with $\epsilon = 0.046$ corresponds to P-Gibbs$_\infty$ as $\gamma = \infty$

6.2 Results

In this subsection, we present (i) the overall trend for change in P-Gibbs' SQ vs. ϵ, and (ii) the effect of hyperparameters (σ, γ, q) and the problem size on P-Gibbs' SQ.

6.2.1 General trends for solution quality

We now provide general trends w.r.t. ϵs and SQs. More specifically, we focus on the change in the SQ with the change in ϵ (aka privacy budget).

(ϵ, δ)-**bounds**. Throughout these experiments, we choose $\delta = 10^{-2}$, $T = 50$, $\tau = 50$ and λs as 100. As standard, our choice of δ is such that $\delta < 1/m$ [21]. We calculate ϵ using different permutations of $\gamma \in \{4, 8, 20, \infty\}, q \in \{0.1, 0.2\}$, and $\sigma \in \{7, 10, 25, 1000\}$. With these, we obtain the following ϵ values: $\epsilon = 0.046$ where $\sigma = 1000, \gamma = \infty$ and $q = 0.1$, (ii) $\epsilon = 0.662$ where $\sigma = 25, \gamma = 20$ and $q = 0.1$, (iii) $\epsilon = 1.31$ where $\sigma = 25, \gamma = 20$ and $q = 0.2$, (iv) $\epsilon = 4.101$ where $\sigma = 10, \gamma = 8$ and $q = 0.2$, (v) $\epsilon = 9.55$ where $\sigma = 7, \gamma = 4$ and $q = 0.2$.

Note that the case with $\epsilon = 0.046$ corresponds to P-Gibbs$_\infty$ as $\gamma = \infty$.

6.2.1.1 Results Fig. 3 presents the overall change in the SQ concerning an increase in ϵ. We plot SQ scores averaged across all problems. For all three benchmarks, the average SQ improves between $\epsilon \in [0.046, 9.55]$. This behavior is expected as greater ϵs imply an increase in the subsampling probability and a decrease in the noise added (σ). The increase in the probability of subsampling allows an agent to explore more values in its domain. That is an increase in the chance of encountering better assignments for itself.

We observe that the average solution quality is least for Ising and the highest for MS. The quality for GC is slightly lower than that of MS. For all three benchmarks, the quality increases sufficiently with increasing privacy budget, i.e., ϵ. P-Gibbs' performance for meeting-schedule is strong, especially for higher ϵs. Note that $\epsilon < 1$ is typically desirable. We consider $\epsilon \geq 1$ for illustrative purposes. P-Gibbs also provides good solution qualities for $\epsilon < 1$. Specifically, for Ising, the average quality crosses 0.75. For GC, the average quality remains above 0.8 and crosses 0.92 for MS.

Coefficient of Variation. The Coefficient of Variation (CoV) is a statistical measure equal to the ratio between the standard deviation and the average. Note that lower CoVs

imply a lower extent of variability with the average solution quality. We compute the CoV values for each ϵ based on the reported average and standard deviations (refer to Fig. 3).

For Ising, for varying ϵ, we observe a maximum CoV of 0.123 and a minimum of 0.050. Likewise, for GC, we observe a minimum CoV of 0.059 and a maximum of 0.104. For meeting-scheduling, we have 0.044 (minimum) and 0.086 (maximum). Notably, for all three benchmarks, the maximum CoV corresponds to $\epsilon = 0.046$. For graph-coloring, the minimum CoV corresponds to $\epsilon = 9.55$ and for Ising and meeting-scheduling to $\epsilon = 4.101$. As expected, the maximum CoV corresponds to the lowest ϵ since the amount of noise (or the loss in SD-Gibbs' distribution) is highest. In contrast, higher ϵ infuse lesser noise, and consequently, the CoVs are lower.

The above observation supports the improvement in solution quality with increasing ϵ in Fig. 3. As ϵ increases, the decrease in CoV values denotes a lower extent of variability concerning the average solution quality. That is, as ϵ increases, P-Gibbs is likelier to output a solution quality closer to the average quality reported.

6.2.2 Effect of specific parameters on solution quality

We now study the specific effect of parameters σ, γ, and q on the quality of P-Gibbs' solution. First, we vary σ while fixing the other parameters and observing SQ changes. Then, we likewise vary γ followed by q and observe the change in SQ for these. We conduct these experiments on the same 20 benchmark problem instances as earlier and report the average values across 20 runs.

Effect of the Noise Parameter (σ). Similar to our previous subsection experiments, we let λs be 100 and $\delta = 10^{-2}$. Further, we fix $\gamma = \infty$, $\tau = 50$, and $q = 0.1$. We vary σ from the set $\{1000, 100, 50, 25, 10\}$. As σ decreases, the privacy budget ϵ increases. Intuitively, we expect the solution quality to improve with a decrease in noise added.

Figure 4 presents the change in SQ w.r.t to the change in σ. We derive the ϵ values using Table 4. As expected, we observe an overall increase in the solution quality of P-Gibbs as σ decreases. However, the increase is marginal for graph-coloring, while the quality significantly improves for Ising and meeting-scheduling.

Interestingly, the solution quality for meeting-scheduling for $\sigma = 10$ ($\epsilon = 0.32$) is similar to the earlier reported quality for $\epsilon = 9.55$ (Fig. 3). In contrast, from Fig. 3, the SQ for graph-coloring is comfortably better for $\epsilon = 9.55$.

Effect of the Temperature Parameter (γ). We now turn our attention to the effect of γ on the solution qualities of P-Gibbs. For this, we fix $\tau = 50, \sigma = 100$ and $q = 0.1$ while varying $\gamma = \{\infty, 16, 8, 4, 2\}$. Similar to the case of σ, as the temperature parameter γ decreases, the privacy budget ϵ increases. As γ decreases, greater information of the original SD-Gibbs distribution is retained.

Figure 4 presents the results. We observe an increase in SQs as γ decreases. This increase is because an increase in γ implies that P-Gibbs' sampling distribution tends to the original SD-Gibbs' distribution. As such, the resulting solution also tends towards that of SD-Gibbs.

Effect of the subsampling Probability (q). To study the effect of subsampling probability q, we vary the value from the set $q \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. We fix the other parameters, i.e., $\gamma = 16$, $\sigma = 100$, and $\tau = 50$. As the probability q increases, the privacy budget ϵ increases.

Figure 4 presents our results. Similar to our previous results, we see an increase in solution quality for graph-coloring as the probability of sampling increases. This increase is

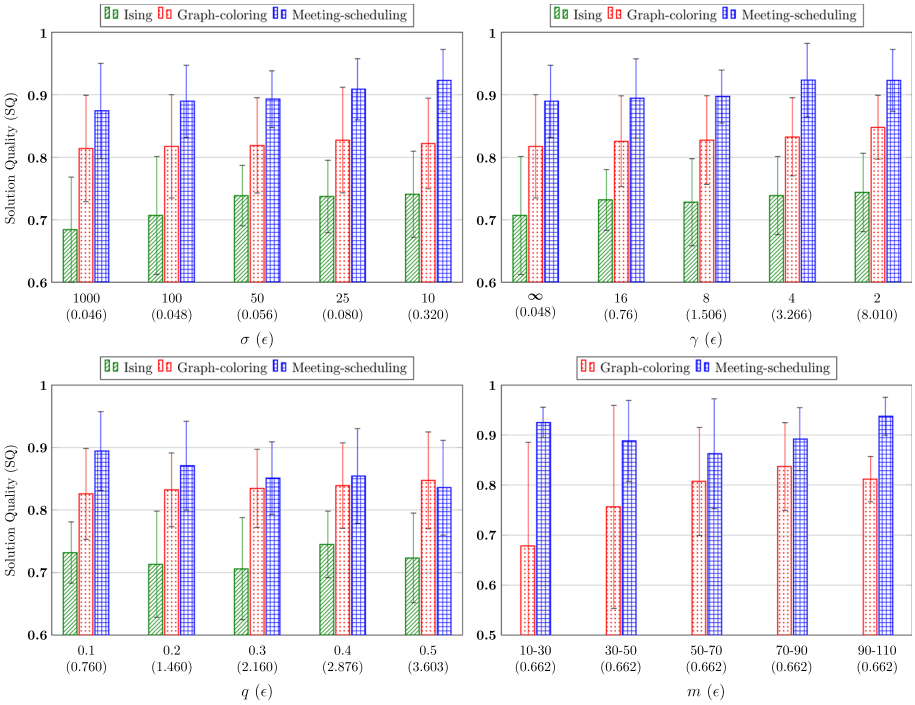


Fig. 4 The average and standard deviation of P-Gibbs' Solution Quality (SQ) for different hyperparameters (σ, γ and q) and the problem size, m

because an increase in the subsampling probability implies an increase in each agent's probability of sampling a better assignment. However, for meeting-scheduling, we do not observe any such trend.

NOTE. These results show that the loss in sampling information deteriorates the solution quality in graph-coloring, while meeting-scheduling's solution quality largely depends on the amount of noise added. This may be due to differences between graph-coloring and meeting-scheduling [24]. In particular, we believe that abstract problems like graph-coloring better satisfy the SD-Gibbs assumption of statistical independence of variables, while concrete problems like meeting-scheduling do not. Thus, solution quality for graph-coloring depends more on the SD-Gibbs probability distribution than meeting-scheduling.

Effect of Problem Size (m). We now measure the effect of the change in the number of agents (m) on P-Gibbs' SQ. To this end, we generate test cases for the two benchmarks[5] Graph-coloring (GC) and Meeting-scheduling (MS), by varying $m \in \{10\text{–}30\,, 30\text{–}50\,, 50\text{–}70\,, 70\text{–}90\,, 90\text{–}110\,\}$. We fix the hyperparameters in P-Gibbs to derive $\epsilon = 0.662$. We generate 10 problem instances for each m and report the average and standard deviation for P-Gibbs' SQ. Figure 4 depicts the results.

[5] We omit Ising from this set of experiments, as Ising instances with > 20 agents ran out of memory during execution.

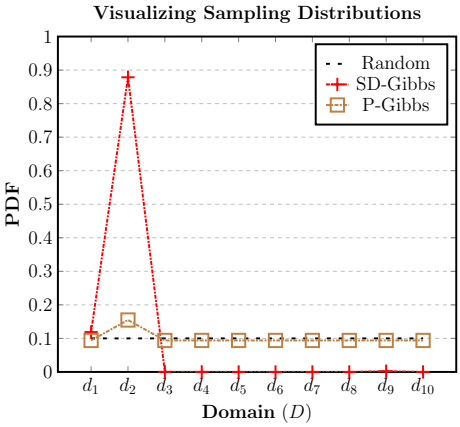


Fig. 5 Visualizing Sampling Distributions for SD-Gibbs and P-Gibbs with a Random Assignment

For GC, the average SQ generally increases as the number of agents increases from 10–30 to 90–110. Further, the standard deviation is more significant when the number of agents is small. When m is 50–70 or more, we observe greater SQ and the standard deviation is lesser than 10–30. Contrarily, for MS, the average SQ is almost similar across different problem sizes. This behavior may be due to the SQs being significantly large for each problem size.

Privacy Leak due to Hyperparameter Tuning. Researchers have shown that hyperparameter tuning of ML models may compromise their privacy [57]. Fortunately, in our case, the tuning only corresponds to DP parameters. These parameters can be tuned via simulating privacy computation in advance, without running the actual problem-solving algorithm, and thus without revealing any information.

6.3 Explaining P-Gibbs' privacy protection for varying ϵ

In this paper, we provide a rigorous (ϵ, δ)-DP guarantee for P-Gibbs. Moreover, in the previous subsection, we provide the empirical quality of P-Gibbs' solution w.r.t. SD-Gibbs for a given privacy budget. We now demonstrate the privacy guarantee of P-Gibbs by comparing the final assignments of P-Gibbs and SD-Gibbs concerning a *random* assignment.

Note that a random final assignment will be *perfectly* privacy-preserving since no information about an agent's utility function will get encoded in the assignment. As such, the *closer* a DCOP algorithm's final assignment is to the random assignment, the *greater* the privacy protection. Figure 5 depicts this observation for a graph-coloring benchmark test instance with the domain $D = \{d_1, \ldots, d_{10}\}$ for any variable/agent i.

We can see that SD-Gibbs' distribution prefers the assignment d_2 with ≈ 0.9 probability. Sampling a value with the SD-Gibbs' distribution will imply that agent i greatly prefers d_2 (i.e., agent i's utility function has a sufficiently greater value for d_2 compared to $D \setminus \{d_2\}$). P-Gibbs's distribution is *closer* to random, thus plugging the information leak. To measure the distance of the assignments, we introduce the metric: *Assignment Distance*.

6.3.1 Assignment distance

As depicted in Fig. 5, we can explain the increased privacy guarantees in P-Gibbs by measuring the distance between the final assignment from P-Gibbs with a random assignment. To measure the distance, we employ the Jensen-Shannon divergence (JSD) [58].[6] Now, consider a DCOP algorithm A and domain $D = \{d_1, \ldots, d_p\}$ such that for each variable/agent i, the assignment distribution after T iterations is given by the vector $\mathbf{p}^i_A = \{p^i_{d_1}, \ldots, p^i_{d_p}\}$. Now, consider the following definition.

Definition 7 (Assignment Distance (AD_A)) We define Assignment Distance (AD_A) of a DCOP algorithm A as the average Jensen-Shannon divergence (JSD) [58] between the vector of the assignment distribution for variable/agent i from algorithm A, i.e., $\mathbf{p}^i_A$, with the vector $\mathbf{r}$ from the random assignment, i.e., $\mathbf{r} = \frac{1}{|D|} \cdot \mathbb{1}_p$. Formally,

$$\text{AD}_A = \frac{\sum_{i \in [p]} \text{JSD}\left(\mathbf{p}^i_A || \mathbf{r}\right)}{p}. \tag{13}$$

From Eq. (13), $\text{AD}_A \in [0, 1]$. When $\text{AD}_A \to 0$, it shows that algorithm A's final assignment is closer to the random assignment, i.e., A is as private as a random assignment. For $\text{AD}_A \to 1$, the assignment is farthest, implying that A encodes the maximum information possible. By comparing the assignment distance, i.e., $\text{AD}_{\text{SD-Gibbs}}$ and $\text{AD}_{\text{P-Gibbs}}$, we can explain the increased privacy of P-Gibbs.

6.3.2 Experimental evaluation

To better study assignment distance, we empirically derive its values for two DCOP benchmarks, graph coloring and meeting scheduling. We omit Ising from this set of experiments as the domain there is binary.

Instance Setup. For graph coloring, we have 30 agents/variables with domain size 10 and each constraint between (0, 10]. For meeting scheduling, we have 30 agents/variables with domain size 10 and each constraint between $[-1000, 10] \backslash \{0\}$.

Results. We run both the benchmark instances 40 times and report the corresponding assignment distance (AD) values in Table 5. For graph-coloring, AD values for P-Gibbs are $\approx 50\%$ less than that for SD-Gibbs. Whereas for meeting-scheduling, it is $\approx 10\%$. This shows that P-Gibbs' final assignment encodes less information compared to SD-Gibbs', in turn, better preserving constraint privacy. Moreover, as ϵ increases, the decrease in the noise added and increase in subsampling probability results in an increase in AP values for P-Gibbs as the algorithm behaves more like SD-Gibbs.

[6] JSD [58] is a statistical method to measure the similarity of two probability distributions. It is based on KL-divergence, but does not require the same support for the distributions.

Table 5 Empirically evaluating Assignment Distance for SD-Gibbs and P-Gibbs. Note that AD → 0 implies greater privacy protection, while AD → 1 implies maximum information leak

Benchmark	Assignment distance (AD)			
	SD-Gibbs	P-Gibbs ($\epsilon = 0.046$)	P-Gibbs ($\epsilon = 0.662$)	P-Gibbs ($\epsilon = 9.55$)
Graph-coloring	0.553	0.253	0.275	0.288
Meeting-scheduling	0.71	0.623	0.624	0.64

6.4 Discussion

Overall, P-Gibbs provides strong solution quality for a competitive privacy budget. Concerning specific hyperparameters, we observe that the amount of noise (σ) added has the most impact on the quality of the solution – especially for meeting-scheduling. In practice, one needs to properly tune the parameters based on the problem at hand [24]. Since ours is the first method of its kind, to the best of our knowledge, we believe the results presented are strong, and future work may further improve the performance.

6.5 Advantages of our DP-based Approach

We present the first differentially private DCOP algorithm with provable guarantees for constraint privacy with P-Gibbs. Here we emphasize the utility of a DP-based solution compared to the existing cryptographic solutions. Notably, as also mentioned in [3], in cryptography-based solutions, the final assignment may leak critical information about the constraints, i.e., existing algorithms do not satisfy solution privacy. Our DP-based approach overcomes this prevalent issue by randomizing the computation and perturbing agents' utility values. In turn, the final assignment may not always be optimal, i.e., with P-Gibbs, there is a drop in solution quality.

Another crucial advantage of DP is the scalability of P-Gibbs. Since our privacy guarantees do not depend on computationally heavy cryptographic primitives, we conduct experiments on much larger problems than existing algorithms (e.g., [3, 17]). P-Gibbs' run time remains the same as SD-Gibbs in contrast to private DCOP algorithms based on cryptographic primitives (e.g., [18, 19, 28]).

7 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. Using 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.

7.1 Future Work

As a first attempt at providing a differential privacy guarantee for DCOPs, we focused on the classical DP notion in this paper. Using the notion of Bayesian DP [49] may further improve the (ϵ, δ) guarantees. More concretely, with Bayesian DP, we may be able to estimate the privacy cost $c_t(\lambda)$ with the agent's actual distribution (instead of the worst-case for all agents). We remark that such an estimation is non-trivial as it may require certain assumptions of other agent's distribution. We leave the analysis for future work.

Alternatively, one can further enrich our DCOP model by relaxing the assumption of domains being the same for each agent while ensuring meaningful privacy guarantees. Concerning P-Gibbs, we can also perform experiments on real-world datasets to further fine-tune the algorithm's hyperparameters.

Appendix 1 Comparing P-Gibbs' quality of solution with DPOP

Complete algorithms like DPOP fail to solve sufficiently large problems. More concretely, in our experiments, the pyDCOP's DPOP solver timed out after 24 h for (i) an Ising benchmark instance with 10 variables/agents, (ii) a graph-coloring benchmark instance with 12 variables/agents and $|D| = 8$, (iii) a meeting-scheduling benchmark instance with 25 variables/agents and $|D| = 30$. Recall that in Sect. 6, we conduct experiments on significantly larger problems than these.

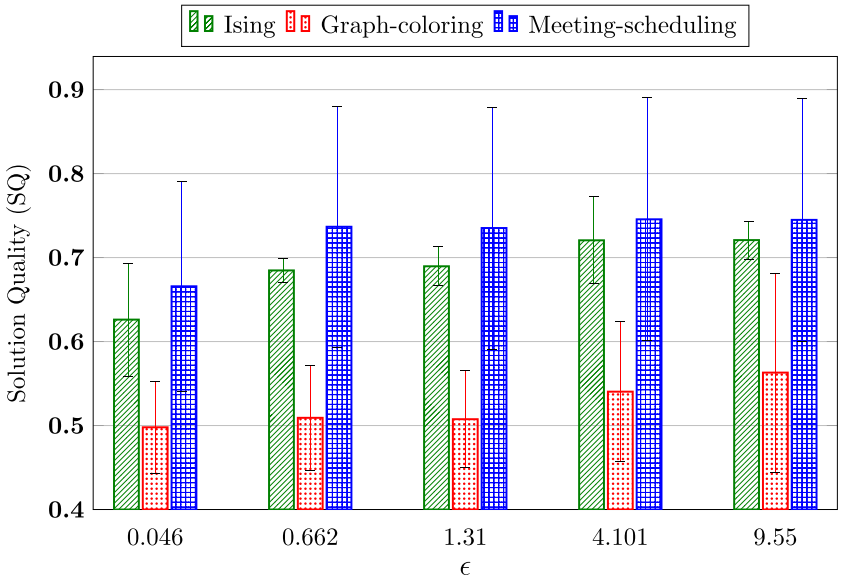


Fig. 6 The average and standard deviation of P-Gibbs' Solution Quality (SQ) for different privacy budgets, compared to DPOP. Note that, the case with $\epsilon = 0.046$ corresponds to P-Gibbs$_\infty$ as $\gamma = \infty$

As private algorithms built atop DPOP (i.e., P-DPOP, $P^{3/2}$-DPOP and P^2-DPOP) use computationally expensive cryptographic primitives, these algorithms are less efficient than DPOP [3, 17]. Given this evidence, we conclude that P-Gibbs is significantly more scalable than the DPOP family of algorithms.

Next, we want to compare the quality of solutions given by the DPOP family of algorithms and P-Gibbs. As each of P-DPOP, $P^{3/2}$-DPOP, and P^2-DPOP do not any noise/randomness to the computation process, it suffices to compare the solution quality of DPOP and SD-Gibbs (refer Definition 6). We begin by setting up the benchmark problems.

Solution Quality

Benchmarks. As in Sect. 6, the instances are created using pyDCOP's *generate* option.[7]

1. **Ising** [23]. We generate 5 sample Ising problems with variables/agents between [5, 8] and $D_i = \{0, 1\}, \forall i$. The constraints are of two types: (i) binary constraints whose strength is sampled from $\mathcal{U}[\beta, \beta]$ where $\beta \in [1, 10)$ and (ii) unary constraints whose strength is sampled from $\mathcal{U}[-\rho, \rho]$ where $\rho \in [0.05, 0.9)$. Ising is a *minimization* problem.
2. Graph-coloring (GC). We generate 5 sample graph-coloring problems. The problems are such that the number of agents/variables lies between [8, 12] and agents' domain size between [10, 20). Each constraint is a random integer taken from (0, 10). Graph-coloring is a *minimization* problem.
3. Meeting-scheduling (MS). We generate 5 sample meeting-scheduling problems. The problem instances are such that the number of agents and variables lies between [15, 20] with the number of slots, i.e., the domain for each agent randomly chosen from [20, 30]. 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.

Results. Fig. 6 depicts the results. As previously observed in Fig. 4, P-Gibbs' solution quality improves as the problem size (m) increases. As DPOP does not scale, we see that the solution quality remains around 50–75%, with an increase in the quality with an increase in ϵ. Furthermore, as P-DPOP, $P^{3/2}$-DPOP, and P^2-DPOP do not add noise/randomness in the solution process, we argue that these results, although hold for them.

Appendix 2 Comparing P-Gibbs' quality of solution with max-sum

As P-Max-Sum perfectly simulates Max-Sum, i.e., preserves the solution of its underlying non-private counterpart [18, Theorem 4.1], we now compare P-Gibbs' quality of solution with that of Max-Sum. We begin by generating the benchmark instances.

Benchmarks. As in Sect. 6, the instances are created using pyDCOP's *generate* option. We use similarly sized problems as in the experiments in Sect. 6.

1. **Ising** [23]. We generate 5 sample Ising problems with variables/agents between [10, 20) and $D_i = \{0, 1\}, \forall i$. The constraints are of two types: (i) binary constraints whose

[7] Compared to the instances created in Sect. 6, we only scale down the number of variables and the domain size while keeping the nature of the constraints the same.

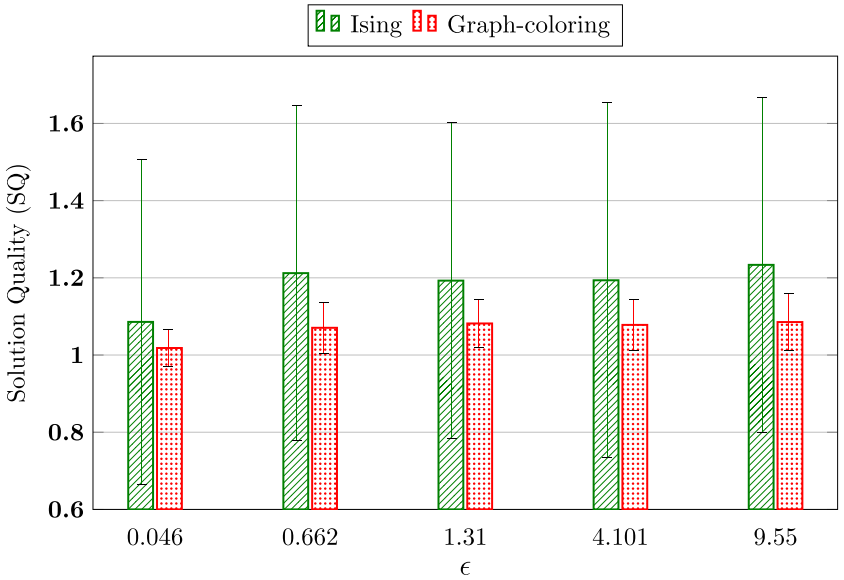


Fig. 7 The average and standard deviation of P-Gibbs' Solution Quality (SQ) for different privacy budgets, compared to Max-Sum. Note that, the case with $\epsilon = 0.046$ corresponds to P-Gibbs$_\infty$ as $\gamma = \infty$

strength is sampled from $\mathcal{U}[\beta, \beta]$ where $\beta \in [1, 10)$ and (ii) unary constraints whose strength is sampled from $\mathcal{U}[-\rho, \rho]$ where $\rho \in [0.05, 0.9)$. Ising is a *minimization* problem.

2. **Graph-coloring (GC).** We generate 5 sample graph-coloring problems. The problems are such that the number of agents/variables lies between [50, 100] and agents' domain size between [10, 20). Each constraint is a random integer taken from (0, 10). Graph-coloring is a *minimization* problem.

We omitted meeting-scheduling from this set of experiments, as Max-Sum did not perform

Results. For both Ising and GC, the solution quality remains > 1. That is, P-Gibbs consistently outputs better solutions than Max-Sum. The quality also improves as the privacy budget, ϵ, increases from 0.046 to 9.55.

Runtime. Here, we show that while P-Max-Sum perfectly simulates Max-Sum, it does so with a significant computational overhead. More concretely, from [18, Section 6.2], we know that P-Max-Sum's computational overhead (compared to Max-Sum), for any iteration and each node is *quadratic* in the domain size. E.g., from [18, Section 6.7], for random graphs, the runtime increases from 100 s for $|D| = 3$, to 242 s for $|D| = 5$ and 450 s for $|D| = 7$.

With this, we conclude that P-Gibbs' performance is comparable to Max-Sum (Fig. 7) and, importantly, without the significant computational overhead of P-Max-Sum.

Comparing P-Gibbs, P-RODA and P-Max-Sum. From Table 1, P-RODA [27] satisfies topology, constraint, and decision privacy. In terms of performance, P-RODA finds better quality solutions than P-Max-Sum [27]. With Fig. 7, we also see that P-Gibbs provides better quality of solution than Max-Sum (and, consequently, P-Max-Sum). Given these observations, we believe that the solutions of P-Gibbs and P-RODA may be comparable.

Furthermore, our differentially private variant is significantly less computationally expensive than P-RODA. For instance, in [27, Figure 8], we see that there is a

non-linear increase in P-RODA's runtime with an increase in the domain size. In fact, P-RODA takes (a minimum of) ≈ 200 seconds for an iteration when the domain size is 25 [27, Figure 8]. In contrast, P-Gibbs takes ≈ 300 and 25 s to complete 50 iterations for randomly generated instances of graph-coloring and meeting-scheduling for the same domain size and 100 agents.

Appendix 3 Explaining P-Gibbs'privacy protection for varying ϵ

To measure the proximity of the assignments between the maximum distribution values, we introduce the metric: *Assignment Proximity*. The critical difference between Assignment (AD) Distance and Assignment Proximity (AP) is that while AD compares the distance between the overall assignment distribution, AP compares the distance between the most probable assignment and a random assignment.

Assignment Proximity

Consider the following definition.

Definition 8 (Assignment Proximity (AP_A)) We define Assignment Proximity (AP_A) of a DCOP algorithm A as the L2-distance between the vector of the most probable assignment for each variable across l runs with the vector of the random assignment. Formally,

$$AP_A = \left\| \left(\frac{x_{f_i} : \texttt{frequent}(x_i^1, \ldots, x_i^l)}{l} \right)_{i \in [p]} - \frac{1}{|D|} \cdot \mathbb{1}_p \right\|_2 \tag{14}$$

where $\mathbb{1}_p$ is a p-dimensional unit vector, x_i^k is the final assignment of variable x_i in the $k \in [l]$ run, $\texttt{frequent}(\cdot)$ is a function which outputs the most frequently occurring value given an input vector and x_{f_i} is the most frequent assignment of variable x_i.

The intuition behind introducing assignment proximity is that given $AP_{\text{SD-Gibbs}}$ and $AP_{\text{P-Gibbs}}$, one can compare the proximity of P-Gibbs' assignment to a random assignment with that of SD-Gibbs. A greater value of $AP_{\text{SD-Gibbs}}$ will imply that with SD-Gibbs, each variable is being assigned a particular domain value with high probability, in turn encoding maximum information. In contrast, a lower value of $AP_{\text{P-Gibbs}}$ will imply that with P-Gibbs, each variable is being assigned a particular domain value with probability closer to random (i.e., $1/|D|$). By comparing the assignment proximity values, we can explain the increased privacy of P-Gibbs.

Experimental Evaluation

To better visualize assignment proximity, we empirically derive the values of it for the two DCOP benchmarks, graph coloring and meeting scheduling.

Instance Setup. For graph coloring, we have 30 agents/variables with domain size 10 and each constraint between (0, 10]. For meeting scheduling, we have 30 agents/variables with domain size 20 and each constraint between $[-1000, 10] \backslash \{0\}$.

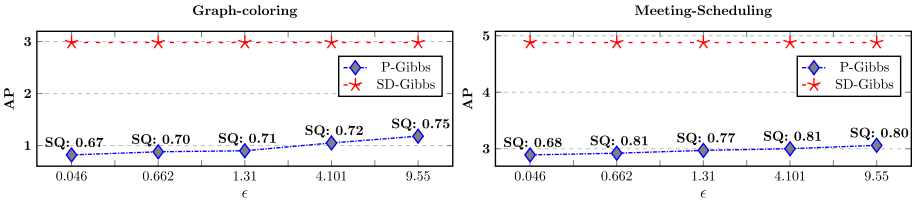


Fig. 8 Assignment Proximity (AP) for SD-Gibbs and P-Gibbs for different ϵs, with the corresponding Solution Quality (SQ) values. The lower the AP, the higher the proximity of an algorithm's final assignment with a perfectly random assignment. Thus, an algorithm with lower AP encodes less information regarding an agent's utility function, preserving constraint privacy

Results. We run both the benchmark instances $l = 20$ times and report the corresponding assignment proximity (AP) values in Fig. 8. From the table, observe that AP value for P-Gibbs is $\approx 50\%$ less than that for SD-Gibbs. This shows that P-Gibbs' final assignment encodes less information compared to SD-Gibbs', in turn, better preserving constraint privacy. Moreover, as ϵ increases, the decrease in the noise added and increase in subsampling probability results in an increase in AP values for P-Gibbs as the algorithm behaves more like SD-Gibbs.

Author contributions All authors contributed to the problem formulation and the design of the algorithm. SD and AT contributed to the theoretical proofs. SD worked on the algorithm's implementation and prepared the figures. SD and AT wrote the paper. All authors reviewed the manuscript.

Funding Not applicable.

Availability of data and materials The implementation is adopted from pyDCOP [53] available at github.com/Orange-OpenSource/pyDcop. We perform our experiments on synthetic data generated using pyDCOP's command line tool. Our codebase is available at: github.com/magnetar-iiith/PGiBBS.

Declarations

Conflict of interest Authors have no competing interests as defined by Springer.

Links to Own Prior Work Some text passages of this manuscript (e.g., preliminaries) have been drawn from our prior work published as a conference paper [59]. The current manuscript differs from the conference paper as follows: We provide a concrete example to highlight the privacy leak in SD-Gibbs (the state-of-the-art DCOP algorithm). We introduce a novel privacy metric, namely solution privacy, to study the additional information leak in privacy-preserving DCOP algorithms. In [59], we only provide proof sketches. The current manuscript provides formal proofs for each result presented. We provide an additional set of experiments, including (i) an additional benchmark and (ii) concerning P-Gibbs' hyperparameters to study the specific impact of each hyperparameter on the quality of P-Gibbs' solution and the privacy budget. We introduce a novel metric, namely assignment distance, to explain the privacy protection in P-Gibbs compared to SD-Gibbs.

References

1. Rosser, M. (2003). *Basic Mathematics for Economists*. Routledge.
2. Yokoo, M., Durfee, E. H., Ishida, T., & Kuwabara, K. (1998). The distributed constraint satisfaction problem: formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering, 10*(5), 673–685.

3. Faltings, B., Léauté, T., & Petcu, A. (2008). Privacy guarantees through distributed constraint satisfaction. In *2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology* (Vol. 2, pp. 350–358). IEEE.
4. Abadi, M., Chu, A., Goodfellow, I., McMahan, H.B., Mironov, I., Talwar, K., & Zhang, L.(2016) Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on CCS* (pp. 308–318).
5. Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., & Seth, K. (2017) Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1175–1191).
6. Huai, M., 0015, D.W., Miao, C., Xu, J., & Zhang, A. (2019) Privacy-aware synthesizing for crowdsourced data. In *IJCAI* (pp. 2542–2548).
7. Modi, P. J., Shen, W.-M., Tambe, M., & Yokoo, M. (2005). Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence, 161*(1–2), 149–180.
8. Hirayama, K., & Yokoo, M. (1997) Distributed partial constraint satisfaction problem. In*International Conference on Principles and Practice of Constraint Programming* (pp. 222–236). Springer
9. Gershman, A., Meisels, A., & Zivan, R. (2009). Asynchronous forward bounding for distributed cops. *Journal of Artificial Intelligence Research, 34*, 61–88.
10. Maheswaran, R.T., Pearce, J.P., & Tambe, M. (2006) In Scerri, P., Vincent, R., Mailler, R. (eds.) *A Family of Graphical-Game-Based Algorithms for Distributed Constraint Optimization Problems* (pp. 127–146). Springer.
11. Petcu, A., & Faltings, B. (2005) DPOP: A scalable method for multiagent constraint optimization. In*IJCAI 05* (pp. 266–271).
12. Farinelli, A., Rogers, A., Petcu, A., & Jennings, N.R. (2008) Decentralised coordination of low-power embedded devices using the Max-Sum algorithm. In *AAMAS* (pp. 639–646).
13. Ottens, B., Dimitrakakis, C., & Faltings, B. (2012) DUCT: An upper confidence bound approach to distributed constraint optimization problems. In*AAAI* (pp. 528–534).
14. Ottens, B., Dimitrakakis, C., & Faltings, B. (2017). DUCT: An upper confidence bound approach to distributed constraint optimization problems. *ACM Transactions on Intelligent Systems and Technology (TIST), 8*(5), 1–27.
15. Fioretto, F., Pontelli, E., & Yeoh, W. (2018). Distributed constraint optimization problems and applications: A survey. *Journal of Artificial Intelligence Research, 61*, 623–698.
16. Nguyen, D. T., Yeoh, W., Lau, H. C., & Zivan, R. (2019). Distributed gibbs: A linear-space sampling-based DCOP algorithm. *Journal of Artificial Intelligence Research, 64*, 705–748.
17. Léauté, T., & Faltings, B. (2013). Protecting privacy through distributed computation in multi-agent decision making. *Journal of Artificial Intelligence Research, 47*, 649–695.
18. Tassa, T., Grinshpoun, T., & Zivan, R. (2017). Privacy preserving implementation of the Max-Sum algorithm and its variants. *Journal of Artificial Intelligence Research, 59*, 311–349.
19. Grinshpoun, T., & Tassa, T. (2016). P-SyncBB: A privacy preserving branch and bound DCOP algorithm. *Journal of Artificial Intelligence Research, 57*, 621–660.
20. Dwork, C., McSherry, F., Nissim, K., & Smith, A. (2006). Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference* (pp. 265–284). Springer.
21. Dwork, C., & Roth, A. (2014). The algorithmic foundations of differential privacy. *Theoretical Computer Science, 9*(3–4), 211–407.
22. Liao, J. (1998). Variance reduction in Gibbs sampler using quasi random numbers. *Journal of Computational and Graphical Statistics, 7*(3), 253–266.
23. Cerquides, J., Rodríguez-Aguilar, J. A., Emonet, R., & Picard, G. (2021). Solving highly cyclic distributed optimization problems without busting the bank: A decimation-based approach. *Logic Journal of the IGPL, 29*(1), 72–95.
24. Maheswaran, R., Tambe, M., Bowring, E., Pearce, J., & Varakantham, P. (2004). Taking DCOP to the real world: Efficient complete solutions for distributed event scheduling. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems* (pp 310–317).
25. Fischer, W., & Muller, B. W. (1991) Method and apparatus for the manufacture of a product having a substance embedded in a carrier. Google Patents. US Patent 5,043,280.
26. Gelly, S., & Silver, D. (2007) Combining online and offline knowledge in UCT. In *Proceedings of the 24th International Conference on Machine Learning* (pp. 273–280).
27. Grinshpoun, T., Tassa, T., Levit, V., & Zivan, R. (2019). Privacy preserving region optimal algorithms for symmetric and asymmetric DCOPs. *Artificial Intelligence, 266*, 27–50.
28. Tassa, T., Grinshpoun, T., & Yanai, A. (2021). PC-SyncBB: A privacy preserving collusion secure DCOP algorithm. *Artificial Intelligence, 297*, 103501.

29. Kogan, P., Tassa, T., & Grinshpoun, T. (2022) Privacy preserving DCOP solving by mediation. In *Cyber Security, Cryptology, and Machine Learning - 6th International Symposium, CSCML*. Lecture Notes in Computer Science (Vol. 13301, pp. 487–498).
30. Yao, A.C. (1982) Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science (SFCS 1982)* (pp. 160–164). IEEE
31. Katagishi, H., & Pearce, J. P. (2007) KOPT: Distributed DCOP algorithm for arbitrary k-optima with monotonically increasing utility. DCR-07
32. Kiekintveld, C., Yin, Z., Kumar, A., & Tambe, M. (2010) Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In *AAMAS* (vol. 10, pp. 133–140).
33. Maheswaran, R. T., Pearce, J. P., Bowring, E., Varakantham, P., & Tambe, M. (2006). Privacy loss in distributed constraint reasoning: A quantitative framework for analysis and its applications. *Autonomous Agents and Multi-Agent Systems, 13*(1), 27–60.
34. Brito, I., Meisels, A., Meseguer, P., & Zivan, R. (2009). Distributed constraint satisfaction with partially known constraints. *Constraints, 14*(2), 199–234.
35. Grinshpoun, T., Grubshtein, A., Zivan, R., Netzer, A., & Meisels, A. (2013). Asymmetric distributed constraint optimization problems. *Journal of Artificial Intelligence Research, 47*, 613–647.
36. Savaux, J., Vion, J., Piechowiak, S., Mandiau, R., Matsui, T., Hirayama, K., Yokoo, M., Elmane, S., & Silaghi, M. (2017) Utilitarian approach to privacy in distributed constraint optimization problems. In*FLAIRS Conference* (pp. 454–459).
37. Savaux, J., Vion, J., Piechowiak, S., Mandiau, R., Matsui, T., Hirayama, K., Yokoo, M., Elmane, S., & Silaghi, M. (2020). Privacy stochastic games in distributed constraint reasoning. *Annals of Mathematics and Artificial Intelligence, 88*, 691–715.
38. Yokoo, M., Etzioni, O., Ishida, T., & Jennings, N. (2001). *Distributed constraint satisfaction: Foundations of cooperation in multi-agent systems*. Springer.
39. Hamadi, Y., Bessiere, C., & Quinqueton, J. (1998) Distributed intelligent backtracking. In *ECAI* (pp. 219–223).
40. Dwork, C. (2006) Differential privacy. In *33rd International Colloquium on Automata, Languages and Programming, Part II (ICALP 2006)*. Lecture Notes in Computer Science (vol. 4052, pp. 1–12).
41. Rubinstein, B. I., Bartlett, P. L., Huang, L., & Taft, N. (2009) Learning in a large function space: Privacy-preserving mechanisms for SVM learning. arXiv preprint arXiv:0911.5708
42. Chaudhuri, K., Sarwate, A., & Sinha, K. (2012) Near-optimal differentially private principal components. Advances in neural information processing systems **25**
43. Basu, D., Dimitrakakis, C., & Tossou, A. (2019) Differential privacy for multi-armed bandits: What is it and what is its cost? arXiv preprint arXiv:1905.12298
44. Papernot, N., Abadi, M., Erlingsson, U., Goodfellow, I., & Talwar, K.(2017) Semi-supervised knowledge transfer for deep learning from private training data. In *ICLR*. openreview.net/forum?id=HkwoSDPgg
45. Roth, A. (2012). Buying private data at auction: the sensitive surveyor's problem. *ACM SIGecom Exchanges, 11*(1), 1–8.
46. Pai, M. M., & Roth, A. (2013). Privacy and mechanism design. *ACM SIGecom Exchanges, 12*(1), 8–29.
47. McSherry, F., & Talwar, K. (2007) Mechanism design via differential privacy. In 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07) (pp. 94–103). IEEE.
48. Huang, Z., Mitra, S., & Vaidya, N. (2015) Differentially private distributed optimization. In *Proceedings of the 2015 International Conference on Distributed Computing and Networking* (pp. 1–10).
49. Triastcyn, A., & Faltings, B. (2020) Bayesian differential privacy for machine learning. In *Proceedings of the 37th International Conference on Machine Learning*.
50. Rényi, A. (1961) On measures of entropy and information. In Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics (vol. 4, pp. 547–562). University of California Press.
51. Balle, B., Barthe, G., & Gaboardi, M. (2018) Privacy amplification by subsampling: Tight analyses via couplings and divergences. Advances in Neural Information Processing Systems **31**
52. Aardal, K. I., Van Hoesel, S. P., Koster, A. M., Mannino, C., & Sassano, A. (2007). Models and solution techniques for frequency assignment problems. *Annals of Operations Research, 153*(1), 79–129.
53. Rust, P., Picard, G., & Ramparany, F. (2019) pyDCOP: A DCOP library for dynamic IoT systems. In *International Workshop on Optimisation in Multi-Agent Systems*.
54. Arshad, M., & Silaghi, M.C. (2004). Distributed simulated annealing. Distributed constraint problem solving and reasoning in multi-agent systems **112**
55. Chapman, A. C., Rogers, A., & Jennings, N. R. (2011). Benchmarking hybrid algorithms for distributed constraint optimisation games. *Autonomous Agents and Multi-agent Systems, 22*, 385–414.

56. Kirkpatrick, S., Gelatt, C. D., Jr., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671–680.
57. Papernot, N., & Steinke, T. (2021) Hyperparameter tuning with Renyi differential privacy. arXiv preprint arXiv:2110.03620
58. Liese, F., & Vajda, I. (2018) Convex statistical distances. Statistical Inference for Engineers and Data Scientists
59. Damle, S., Triastcyn, A., Faltings, B., & Gujar, S. (2021) Differentially private multi-agent constraint optimization. In *IEEE/WIC/ACM WI-IAT* (pp. 422–429).