Journal of Artificial Intelligence Research 64 (2019) 705-748 Submitted 02/18; published 03/19

Distributed Gibbs: A Linear-Space Sampling-Based DCOP Algorithm

Duc Thien Nguyen DTNGUYEN.2014@PHDIS.SMU.EDU.SG
School of Information Systems
Singapore Management University
80 Stamford Rd., Singapore 178902

William Yeoh WYEOH@WUSTL.EDU
Department of Computer Science and Engineering
Washington University in St. Louis
1 Brookings Dr., St. Louis, MO 63130, USA

Hoong Chuin Lau HCLAU@SMU.EDU.SG
School of Information Systems
Singapore Management University
80 Stamford Rd., Singapore 178902

Roie Zivan ZIVANR@BGU.AC.IL
Department of Industrial Engineering and Management
Ben-Gurion University of the Negev
P.O.B. 653 Beer-Sheva, 8410501 Israel

Abstract

Researchers have used distributed constraint optimization problems (DCOPs) to model various multi-agent coordination and resource allocation problems. Very recently, Ottens *et al.* proposed a promising new approach to solve DCOPs that is based on confidence bounds via their Distributed UCT (DUCT) sampling-based algorithm. Unfortunately, its memory requirement per agent is exponential in the number of agents in the problem, which prohibits it from scaling up to large problems. Thus, in this article, we introduce two new sampling-based DCOP algorithms called Sequential Distributed Gibbs (SD-Gibbs) and Parallel Distributed Gibbs (PD-Gibbs). Both algorithms have memory requirements per agent that is linear in the number of agents in the problem. Our empirical results show that our algorithms can find solutions that are better than DUCT, run faster than DUCT, and solve some large problems that DUCT failed to solve due to memory limitations.

1. Introduction

Distributed Constraint Optimization Problems (DCOPs) are problems where agents need to coordinate their value assignments to maximize the sum of resulting constraint rewards (Modi, Shen, Tambe, & Yokoo, 2005; Petcu & Faltings, 2005a; Fioretto, Pontelli, & Yeoh, 2018). Researchers have used them to model various multi-agent coordination and resource allocation problems such as the distributed scheduling of meetings (Maheswaran, Tambe, Bowring, Pearce, & Varakantham, 2004b; Zivan, Okamoto, & Peled, 2014), the distributed allocation of targets to sensors in a network (Farinelli, Rogers, Petcu, & Jennings, 2008; Yeoh, Varakantham, & Koenig, 2009), the distributed allocation of resources and coordination of mobile agents in disaster evacuation scenarios (Lass, Kopena, Sultanik, Nguyen,

Dugan, Modi, & Regli, 2008; Nguyen, Yeoh, & Lau, 2012), the distributed management of power distribution networks (Kumar, Faltings, & Petcu, 2009), the distributed generation of coalition structures (Ueda, Iwasaki, & Yokoo, 2010), the distributed coordination of logistics operations (Léauté & Faltings, 2011), the distributed assignment of frequency to radio transmitters and receivers (Yeoh & Yokoo, 2012; Fioretto, Le, Yeoh, Pontelli, & Son, 2014), and the distributed coordination of IoT devices in smart homes (Rust, Picard, & Ramparany, 2016; Fioretto, Yeoh, & Pontelli, 2017).

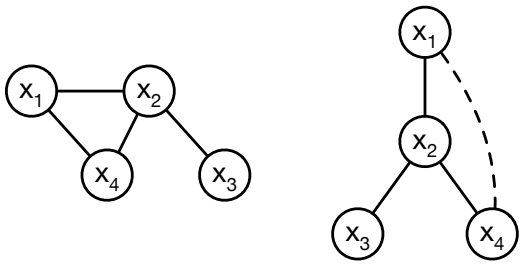
The field has matured considerably over the past decade as researchers continue to develop better and better algorithms (Fioretto et al., 2018). Most of these algorithms fall into one of the following two classes of algorithms: (1) search-based algorithms like ADOPT (Modi et al., 2005) and its variants (Yeoh, Felner, & Koenig, 2010; Gutierrez, Meseguer, & Yeoh, 2011), AFB (Gershman, Meisels, & Zivan, 2009), and MGM (Maheswaran, Pearce, & Tambe, 2004a), where the agents enumerate through combinations of value assignments in a decentralized manner, and (2) inference-based algorithms like DPOP (Petcu & Faltings, 2005a), max-sum (Farinelli et al., 2008), and Action GDL (Vinyals, Rodríguez-Aguilar, & Cerquides, 2011), where the agents use dynamic programming to propagate aggregated information to other agents.

More recently, Ottens, Dimitrakakis, and Faltings (2012, 2017) proposed a promising new approach to solve DCOPs that is based on confidence bounds. They introduced a new sampling-based algorithm called *Distributed UCT*, which is an extension of UCB (Auer, Cesa-Bianchi, & Fischer, 2002) and UCT (Kocsis & Szepesvári, 2006). While the algorithm is shown to outperform competing algorithms,[1] its memory requirement per agent is exponential in the number of agents in the problem, which prohibits it from scaling up to large problems.

Thus, in this article, we introduce two new sampling-based DCOP algorithms called *Sequential Distributed Gibbs* (SD-Gibbs) and *Parallel Distributed Gibbs* (PD-Gibbs), which are distributed extensions of the Gibbs algorithm (Geman & Geman, 1984).[2] Both SD-Gibbs and PD-Gibbs have a linear-space memory requirement – the memory requirement per agent is linear in the number of agents in the problem. While the Gibbs algorithm was designed to approximate joint probability distributions in Markov random fields and solve maximum a posteriori estimation problems, we show how one can map such problems into DCOPs in order for Gibbs to operate directly on DCOPs. Our empirical results show that SD-Gibbs and PD-Gibbs can find solutions that are better than DUCT, run faster than DUCT, and solve some large problems that DUCT failed to solve due to memory limitations.

The structure of this article is as follows: In Section 2, we provide the background for DCOPs. In Section 3, we provide a brief overview of the centralized Gibbs algorithm, which we will extend, and the Distributed UCT (DUCT) algorithm. We then describe the

1. DUCT finds better solutions compared to DSA and MGM when they are all given the same amount of runtime, and finds solutions for large problems that DPOP failed to solve due to memory limitations (Ottens et al., 2012, 2017).
2. This article extends our previous conference paper (Nguyen, Yeoh, & Lau, 2013), which introduced the Distributed Gibbs (D-Gibbs) algorithm, in the following manner: (1) It introduces an improved version of D-Gibbs that uses best-response optimizations; It describes a partial trace on an example DCOP; (3) It includes more detailed theoretical proofs; and (4) It presents more comprehensive empirical evaluations.



x_i	x_j	$F_{ij}(x_i, x_j)$
0	0	-1
0	1	1
1	0	1
1	1	-2

where $i < j$

(a) (b) (c)

Figure 1: Example DCOP

two variants of the Distributed Gibbs algorithm in Section 4 and present the experimental results in Section 5 before concluding in Section 6.

2. Background: DCOP

A *distributed constraint optimization problem* (DCOP) (Modi et al., 2005; Mailler & Lesser, 2004; Petcu & Faltings, 2005a; Fioretto et al., 2018) is defined by $\langle \mathcal{X}, \mathcal{D}, \mathcal{F}, \mathcal{A}, \alpha \rangle$, where

- $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of variables;
- $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of finite domains, where D_i is the domain of variable x_i;
- $\mathcal{F}$ is a set of utility functions, where each utility function $F_i : D_i \times D_j \mapsto \mathbb{R}$ specifies the utility of each combination of values of variables in its *scope* (i.e., $x_{i_1}, \ldots, x_{i_k}$);[3]
- $\mathcal{A} = \{a_1, \ldots, a_p\}$ is a set of agents; and
- $\alpha : \mathcal{X} \to \mathcal{A}$ maps each variable to one agent.

Although the general DCOP definition allows one agent to own multiple variables as well as the existence of k-ary constraints, with $k > 2$, we use a restricted definition where each agent owns *exactly* one variable and that all utility functions are binary functions, where their scopes have *exactly* two variables. We thus use F_{ij} to denote the utility function between variables x_i and x_j in the rest of the paper. One can transform a general DCOP to our restricted DCOP using pre-processing techniques (Yokoo, 2001; Burke & Brown, 2006; Bacchus, Chen, van Beek, & Walsh, 2002). A solution is a value assignment for a subset of variables. Its utility is the evaluation of all utility functions on that solution. A solution is *complete* iff it is a value assignment for all variables. The goal is to find a utility-maximal complete solution. Finally, we assume that the messages sent between agents can be delayed by a finite amount of time but are never lost and they are received in the same order that they were sent.

A *constraint graph* visualizes a DCOP instance, where nodes in the graph correspond to variables in the DCOP and edges connect pairs of variables appearing in the same utility function. A *pseudo-tree* arrangement has the same nodes and edges as the constraint graph and satisfies that (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

3. Feasible value assignments are assigned non-negative utilities and infeasible value assignments are assigned negative utilities.

Algorithm 1: Gibbs$(z_1, \ldots, z_n)$

```
1 for i = 1 to n do
2 |  z_i^0 ← Initialize(z_i)
3 end
4 for t = 1 to T do
5 |  for i = 1 to n do
6 |  |  z_i^t ← Sample(P(z_i | z_1^t, ..., z_{i-1}^t, z_{i+1}^{t-1}, ..., z_n^{t-1}))
7 |  end
8 end
```

edges are called *backedges*. Tree edges connect parent-child nodes, while backedges connect a node with its pseudo-parents and its pseudo-children. A pseudo-tree arrangement can be constructed using a distributed DFS algorithm (Hamadi, Bessière, & Quinqueton, 1998). In this article, we will use N_i to refer to the set of neighbors of variable x_i in the constraint graph, C_i to refer to the set of children of variable x_i in the pseudo-tree, P_i to refer to the parent of variable x_i in the pseudo-tree, and PP_i to refer to the set of pseudo-parents of variable x_i in the pseudo-tree.

Figure 1(a) shows the constraint graph of a simple example DCOP with four variables that can each take on value 0 or value 1. Figure 1(b) shows one possible pseudo-tree for the problem, where variable x_1 has one pseudo-child x_4 (the dotted line is a backedge). Figure 1(c) shows the utility function that is identical for all four functions F_{12}, F_{14}, F_{23}, and F_{24}. For our example DCOP, a utility-maximal complete solution is $x_1 = x_3 = x_4 = 0$ and $x_2 = 1$, which has a utility of 2.

3. Background: Algorithms

In this section, we provide a brief overview of two relevant sampling-based algorithms – the centralized Gibbs algorithm and the Distributed UCT (DUCT) algorithm.

3.1 Gibbs

The Gibbs sampling algorithm (Geman & Geman, 1984) is a Markov chain Monte Carlo algorithm that can be used to approximate joint probability distributions. It generates a Markov chain of samples, each of which is correlated with previous samples. Suppose we have a joint probability distribution $P(z_1, z_2, \ldots, z_n)$ over n variables, which we would like to approximate. Algorithm 1 shows the pseudocode of the Gibbs algorithm, where each variable z_i^t represents the t-th sample of variable z_i. The algorithm first initializes z_i^0 to any arbitrary value of variable z_i (lines 1-3). Then, it iteratively samples z_i^t from the conditional probability distribution assuming that all the other $n-1$ variables take on their previously sampled values, respectively (lines 4-8). This process continues for a fixed number of iterations or until convergence, that is, the joint probability distribution approximated by the samples do not change. It is also common practice to ignore a number of samples at the beginning as it may not accurately represent the desired distribution. Once the joint probability distribution is found, one can easily identify a complete solution with the

maximum likelihood. This problem is called the *maximum a posteriori* (MAP) estimation problem, which is a common problem in many applications such as image processing (Besag, 1986) and bioinformatics (Yanover, Meltzer, & Weiss, 2006; Sontag, Meltzer, Globerson, Jaakkola, & Weiss, 2008).

The Gibbs sampling algorithm is desirable as its approximated joint probability distribution (formed using its samples) will converge to the true joint probability distribution at the limit. While Gibbs cannot be used to solve DCOPs directly, we will later show how one can map MAP estimation problems to DCOPs, and how to extend Gibbs to solve DCOPs in a distributed manner.

3.2 Distributed UCT

The *Upper Confidence Bound* (UCB) (Auer et al., 2002) and *UCB Applied to Trees* (UCT) (Kocsis & Szepesvári, 2006) algorithms are two Monte Carlo algorithms that have been successfully applied to find near optimal policies in large Markov decision processes. The *Distributed UCT* (DUCT) algorithm (Ottens et al., 2012, 2017) is a distributed version of UCT that can be used to find near-optimal cost-minimal complete DCOP solutions. We now provide a brief introduction to the algorithm and refer readers to the original articles (Ottens et al., 2012, 2017) for a more detailed treatment. Additionally, it is important to note that DUCT was designed to solve a *minimization* problem, where the goal is to find a cost-minimal complete solution. We thus describe it as such below and explain how it can be used to solve our *maximization* problem in the experimental results section.

DUCT first constructs a pseudo-tree, by running a Distributed DFS algorithm (Hamadi et al., 1998). After the pseudo-tree is constructed, each agent knows its parent, pseudo-parents, children and pseudo-children. Each agent x_i maintains the following for all possible contexts X and values $d \in D_i$:

- Its current value d_i.
- Its current context X_i, which is initialized to null. It is its assumption on the current values of its ancestors.
- Its cost y_i, which is initialized to ∞. It is the sum of the costs of all cost functions between itself and its ancestors given that they take on their respective values in its context and it takes on its current value.
- Its counter $\tau_i(X, d)$, which is initialized to 0. It is the number of times it has sampled value d under context X.
- Its counter $\tau_i(X)$, which is initialized to 0. It is the number of times it has received context X from its parent.
- Its cost $\hat{\mu}_i(X, d)$, which is initialized to ∞. It is the smallest cost found when it sampled d under context X so far up to the current iteration.
- Its cost $\hat{\mu}_i(X)$, which is initialized to ∞. It is the smallest cost found under context X so far up to the current iteration.

At the start, the root agent chooses its value and sends it down in a CONTEXT message to each of its children. When an agent receives a CONTEXT message, it too chooses its value, appends it to the context in the CONTEXT message, and sends the appended context down in a CONTEXT message to each of its children. Each agent x_i chooses its value d_i

using:

$$d_i = \operatorname*{argmin}_{d \in D_i} B_i(d) \tag{1}$$

$$B_i(d) = f(\delta_i(d), \hat{\mu}_i(X_i, d), \tau_i(X_i, d), B_c) \tag{2}$$

$$\delta_i(d) = \sum_{\langle x_j, d_j \rangle \in X_i} F_{ij}(d, d_j) \tag{3}$$

where its bound $B_i(d)$ is initialized with a heuristic function f that balances exploration and exploitation as well as using bounds B_c that are reported by the agent's children. Additionally, each agent x_i increments the number of times it has chosen its current value d_i under its current context X_i using:

$$\tau_i(X_i, d_i) = \tau_i(X_i, d_i) + 1 \tag{4}$$

$$\tau_i(X_i) = \tau_i(X_i) + 1 \tag{5}$$

This process continues until leaf agents receive CONTEXT messages and choose their respective values. Then, each leaf agent calculates its cost and sends it up in a COST message to its parent. When an agent receives a COST message from each of its children, it too calculates its cost, which includes the costs received from its children, and sends it up to its parent. Each agent x_i calculates its costs y_i, $\hat{\mu}_i(X_i, d)$ and $\hat{\mu}_i(X_i)$ using:

$$y_i = \delta_i(d_i) + \sum_{x_c \in C_i} y_c \tag{6}$$

$$\hat{\mu}_i(X_i, d_i) = \min\{\hat{\mu}_i(X_i, d_i), y_i\} \tag{7}$$

$$\hat{\mu}_i(X_i) = \min\{\hat{\mu}_i(X_i), \hat{\mu}_i(X_i, d_i)\} \tag{8}$$

This process continues until the root agent receives a COST message from each of its children and calculates its own cost. Then, the root agent starts a new iteration, and the process continues until all the agents terminate. An agent x_i terminates if its parent has terminated and the following condition holds:

$$\max_{d \in D_i} \left\{ \hat{\mu}_i(X_i) - \left[\hat{\mu}_i(X_i, d) - \sqrt{\frac{\ln \frac{2}{\Delta}}{\tau_i(X_i, d_i)}} \right] \right\} \leq \epsilon \tag{9}$$

where Δ and ϵ are parameters of the algorithm.

4. Distributed Gibbs

While DUCT has been shown to be very promising, its memory requirement per agent is $O(\hat{D}^T)$, where $\hat{D} = \max_{x_i} D_i$ is the largest domain size over all agents and T is the depth of the pseudo-tree. Each agent needs to store a constant number of variables for all possible contexts and values,[4] and the number of possible contexts is exponential in the number of ancestors. Therefore, this large memory requirement might prohibit the use of DUCT

4. This list of variables are listed in Section 3.2.

in large problems, especially if the agents have large domain sizes as well. Therefore, we now introduce the *Sequential Distributed Gibbs* (SD-Gibbs) and *Parallel Distributed Gibbs* (PD-Gibbs) algorithms, which are distributed extensions of the Gibbs algorithm adapted to solve DCOPs. Additionally, their memory requirement per agent is linear in the number of ancestors that the agent has in the pseudo-tree.

4.1 Mapping of MAP Estimation Problems to DCOPs

Recall that the Gibbs algorithm approximates a joint probability distribution over all the variables in a problem when only marginal distributions are available. Once the joint probability distribution is found, it finds the maximum a posteriori (MAP) solution. Gibbs can be used to solve DCOPs if the following two conditions hold: (1) A DCOP, whose solution is one with the maximum utility, can be mapped to a problem whose solution is one with the maximum likelihood; and (2) A solution with the maximum utility is also a solution with the maximum likelihood. Researchers have previously shown that both of these conditions hold and that a DCOP can be mapped to a MAP estimation problem (Dechter, 2003; Sontag, Globerson, & Jaakkola, 2010; Kumar, Yeoh, & Zilberstein, 2011). For completeness, we describe it here again. Consider a MAP estimation problem on a pairwise Markov random field (MRF).[5] An MRF can be visualized by an undirected graph $\langle V, E \rangle$ and is formally defined by:

- A set of random variables $\mathbf{X} = \{x_i \mid \forall i \in V\}$, where each random variable x_i is associated with node $i \in V$. Further, each random variable x_i can be assigned a value d_i from a finite domain D_i.
- A set of potential functions $\boldsymbol{\theta} = \{\theta_{ij}(x_i, x_j) \mid \forall (i,j) \in E\}$, where each potential function $\theta_{ij}(x_i, x_j)$ is associated with edge $(i,j) \in E$. Let the probability $P(x_i = d_i, x_j = d_j)$ be defined as $\exp(\theta_{ij}(x_i = d_i, x_j = d_j))$. For convenience, we will drop the values in the probabilities and use $P(x_i, x_j)$ to mean $P(x_i = d_i, x_j = d_j)$ from now on.

Therefore, a complete assignment $\boldsymbol{x}$ to all the random variables has the probability:

$$P(\boldsymbol{x}) = \frac{1}{Z} \prod_{(i,j) \in E} \exp[\theta_{ij}(x_i, x_j)] \tag{10}$$

$$= \frac{1}{Z} \exp\left[\sum_{(i,j) \in E} \theta_{ij}(x_i, x_j) \right] \tag{11}$$

where Z is the normalization constant. The objective of a MAP estimation problem is to find the most probable assignment to all the variables under $P(\boldsymbol{x})$. This objective is equivalent to finding a complete assignment $\boldsymbol{x}$ that maximizes the function:

$$F(\boldsymbol{x}) = \sum_{(i,j) \in E} \theta_{ij}(x_i, x_j) \tag{12}$$

Maximizing the function in Equation 12 is *also* the objective of a DCOP if each potential function θ_{ij} corresponds to utility function F_{ij}. Therefore, if we use the Gibbs algorithm to

5. We are describing pairwise MRFs so that the mapping to binary DCOPs is clearer.

solve a MAP estimation problem, then the complete solution found for the MAP estimation problem is also a solution to the corresponding DCOP.

Taking our example DCOP shown in Figure 1 as an example, an optimal complete solution $\boldsymbol{x}^*$ of the DCOP is:

$$\boldsymbol{x}^* = \operatorname*{argmax}_{\boldsymbol{x}} \left(F_{12}(x_1, x_2) + F_{14}(x_1, x_4) + F_{23}(x_2, x_3) + F_{24}(x_2, x_4)\right) \tag{13}$$

$$= (x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 0) \tag{14}$$

which is the same solution of a corresponding MAP estimation problem:

$$\boldsymbol{x}^* = \operatorname*{argmax}_{\boldsymbol{x}} P(\boldsymbol{x}) \tag{15}$$

$$= \operatorname*{argmax}_{\boldsymbol{x}} \frac{1}{Z} \prod_{(i,j)\in E} \exp[\theta_{ij}(x_i, x_j)] \tag{16}$$

$$= \operatorname*{argmax}_{\boldsymbol{x}} \frac{1}{Z} \exp\left[\sum_{(i,j)\in E} \theta_{ij}(x_i, x_j)\right] \tag{17}$$

$$= \operatorname*{argmax}_{\boldsymbol{x}} \sum_{(i,j)\in E} \theta_{ij}(x_i, x_j) \tag{18}$$

$$= \operatorname*{argmax}_{\boldsymbol{x}} \left(F_{12}(x_1, x_2) + F_{14}(x_1, x_4) + F_{23}(x_2, x_3) + F_{24}(x_2, x_4)\right) \tag{19}$$

$$= (x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 0) \tag{20}$$

4.2 Sequential Distributed Gibbs

We now describe the *Sequential Distributed Gibbs* (SD-Gibbs) algorithm. Algorithm 2 shows the pseudo-code, where each agent x_i maintains the following:

- Its values d_i and $\hat{d}_i$, which are both initialized to initial value **ValInit**(x_i). They are the agent's value in the current and previous iterations, respectively.
- Its best value d_i^*, which is also initialized to initial value **ValInit**(x_i). It is the agent's value in the best complete solution found so far. Note that each agent maintains its own best value only and does not need to know the best values of other agents. The best complete solution $\boldsymbol{x}^* = (d_1^*, \ldots, d_n^*)$ can then be constructed upon termination.
- Its current context X_i, which is initialized with the tuples of all the neighbors and their initial values. It is the agent's assumption on the current values of the agent's neighbors.
- Its *best-response* value $\bar{d}_i$, which is also initialized to initial value **ValInit**(x_i). It is the agent's value that maximizes its local solution quality (i.e., the sum of utilities over all its constraints) under the assumption that its neighbors take on values according to its current context X_i.
- Its current best-response context $\bar{X}_i$, which is also initialized with the tuples of all the neighbors and their initial values. However, it is the agent's assumption on the *best-response* values, instead of the current values, of the agent's neighbors.
- Its time index t_i, which is initialized to 0. It is the number of iterations the agent has sampled.

Algorithm 2: Sequential Distributed Gibbs()

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

Procedure Initialize()

```
3  d_i ← d̂_i ← d*_i ← d̄_i ← ValInit(x_i)
4  X_i ← X̄_i ← {(x_j, ValInit(x_j)) | x_j ∈ N_i}
5  t_i ← t*_i ← t̄*_i ← 0
6  Δ_i ← Δ̄_i ← 0
7  if x_i is root then
8  |  δ ← δ* ← δ̄ ← 0
9  |  Sample()
10 end
```

- Its time indices t_i^* and $\bar{t}_i^*$, which are both initialized to 0. They are the iteration where the best non-best-response solution and the best best-response solution is found, respectively. Best-response solutions are solutions constructed from the best-response values $\bar{d}_i$ of all agents x_i. Non-best-response solutions are the solutions constructed from the (regular) values d_i of all agents x_i. $\max\{t_i^*, \bar{t}_i^*\}$ is the iteration where the best complete solution is found.
- Its delta value Δ_i, which is initialized to 0. It is the difference in its local solution quality between the solution in the current iteration (i.e., the agent taking on value d_i and all its neighbors take on values according to its context X_i) and the solution in the previous iteration (i.e., the agent taking on value $\hat{d}_i$ and all its neighbors take on values according to its context X_i).
- Its delta value $\bar{\Delta}_i$, which is initialized to 0. It is the difference in local solution quality between the best-response solution in the current iteration (i.e., the agent taking on value $\bar{d}_i$ and all its neighbors take on values according to its context $\bar{X}_i$) and the solution in the previous iteration (i.e., the agent taking on value $\hat{d}_i$ and all its neighbors take on values according to its context $\bar{X}_i$).

Additionally, the root agent maintains the following: The values δ, $\bar{\delta}$, and δ^*, which are all initialized to 0. It is the shifted utility of the current complete solution, the best-response solution, and the best complete solution found so far, respectively. They are shifted by the utility of the initial complete solution, i.e., the solution where all variables x_i are assigned **ValInit**(x_i). For example, if the utility of the initial complete solution is 10 and δ is 5, then the utility of the current solution is 15.

The algorithm starts by constructing a pseudo-tree (line 1) and having each agent initialize its variables to their default values (lines 2-8). The root then starts by sampling, that is, choosing its value d_i based on the probability:

$$P(x_i \mid x_j \in \mathcal{X} \setminus \{x_i\}) = P(x_i \mid x_j \in N_i)$$

Procedure Sample()

11 $t_i \leftarrow t_i + 1$
12 $\hat{d}_i \leftarrow d_i$
13 $d_i \leftarrow$ Sample based on Equation 21
14 $\bar{d}_i \leftarrow \text{argmax}_{d'_i \in D_i} \sum_{\langle x_j, \bar{d}_j \rangle \in \bar{X}_i} F_{ij}(d'_i, \bar{d}_j)$
15 $\Delta_i \leftarrow \sum_{\langle x_j, d_j \rangle \in X_i} \left[F_{ij}(d_i, d_j) - F_{ij}(\hat{d}_i, d_j) \right]$
16 $\bar{\Delta}_i \leftarrow \sum_{\langle x_j, \bar{d}_j \rangle \in \bar{X}_i} \left[F_{ij}(\bar{d}_i, \bar{d}_j) - F_{ij}(\hat{d}_i, \bar{d}_j) \right]$
17 Send VALUE $(x_i, d_i, \bar{d}_i, t^*_i, \bar{t}^*_i)$ to each $x_j \in N_i$

Procedure When Received VALUE$(x_s, d_s, \bar{d}_s, t^*_s, \bar{t}^*_s)$

18 Update $\langle x_s, d'_s \rangle \in X_i$ with (x_s, d_s)
19 **if** $x_s \in PP_i \cup \{P_i\}$ **then**
20 | Update $\langle x_s, d'_s \rangle \in \bar{X}_i$ with $(x_s, \bar{d}_s)$
21 **else**
22 | Update $\langle x_s, d'_s \rangle \in \bar{X}_i$ with (x_s, d_s)
23 **end**
24 **if** $x_s = P_i$ **then**
25 | **if** $\bar{t}^*_s \geq t^*_s$ **and** $\bar{t}^*_s > \max\{t^*_i, \bar{t}^*_i\}$ **then**
26 | | $d^*_i \leftarrow \bar{d}_i$
27 | | $\bar{t}^*_i \leftarrow \bar{t}^*_s$
28 | **else if** $t^*_s \geq \bar{t}^*_s$ **and** $t^*_s > \max\{t^*_i, \bar{t}^*_i\}$ **then**
29 | | $d^*_i \leftarrow d_i$
30 | | $t^*_i \leftarrow t^*_s$
31 | **end**
32 | Sample()
33 | **if** x_i is a leaf **then**
34 | | Send BACKTRACK $(x_i, \Delta_i, \bar{\Delta}_i)$ to P_i
35 | **end**
36 **end**

$$= \frac{1}{Z} \prod_{\langle x_j, d_j \rangle \in X_i} \exp[F_{ij}(d_i, d_j)]$$

$$= \frac{1}{Z} \exp\left[\sum_{\langle x_j, d_j \rangle \in X_i} F_{ij}(d_i, d_j) \right] \tag{21}$$

where Z is the normalization constant (lines 9 and 13). It then computes its best-response value $\bar{d}_i$ (line 14) and sends its current and best-response values d_i and $\bar{d}_i$ in a VALUE message to each of its neighbors (line 17).

When an agent receives a VALUE message, it updates the value of the sender in its current contexts X_i and $\bar{X}_i$ (lines 18-23). If the message is from its parent, then it too samples and sends its current and best-response values in a VALUE message to each of its

Procedure When Received BACKTRACK($x_s, \Delta_s, \bar{\Delta}_s$)

```
37 Δ_i ← Δ_i + Δ_s
38 Δ̄_i ← Δ̄_i + Δ̄_s
39 if Received BACKTRACK messages from all children in this iteration then
40 |  Send BACKTRACK (x_i, Δ_i, Δ̄_i) to P_i
41 |  if x_i is root then
42 |  |  δ̄ ← δ + Δ̄_i
43 |  |  δ ← δ + Δ_i
44 |  |  if δ ≥ δ̄ and δ > δ* then
45 |  |  |  δ* ← δ
46 |  |  |  d_i* ← d_i
47 |  |  |  t_i* ← t_i
48 |  |  else if δ̄ ≥ δ and δ̄ > δ* then
49 |  |  |  δ* ← δ̄
50 |  |  |  d_i* ← d̄_i
51 |  |  |  t̄_i* ← t_i
52 |  |  end
53 |  |  SAMPLE()
54 |  end
55 end
```

neighbors (line 32). This process continues until all the leaf agents sample. Each leaf agent then sends a BACKTRACK message to its parent (lines 33-35). When an agent receives a BACKTRACK message from each child (line 39), it too sends a BACKTRACK message to its parent (line 40). This process continues until the root agent receives a BACKTRACK message from each child, which concludes one iteration.

We now describe how the agents identify if they have found a better solution than the best one found thus far in a decentralized manner without having to know the values of every other agent in the problem. In order to do so, the agents use delta variables Δ_i, $\bar{\Delta}_i$, δ, $\bar{\delta}$, and δ^*, where the upper case delta variables are maintained by each agent and the lower case delta variables are maintained only by the root agent. The two upper case delta variables are sent up the pseudo-tree in BACKTRACK messages (line 40).

After sampling, each agent x_i calculates the difference in its *local* solution quality between the current and previous solutions $\sum_{\langle x_j, d_j\rangle \in X_i} \left[F_{ij}(d_i, d_j) - F_{ij}(\hat{d}_i, d_j)\right]$ and stores that value in Δ_i (line 15). It is a local difference because it takes into account utility functions that involve the agent only. Similarly, the agent also calculates the difference between the current best-response solution and the previous solution $\sum_{\langle x_j, \bar{d}_j\rangle \in \bar{X}_i} \left[F_{ij}(\bar{d}_i, \bar{d}_j) - F_{ij}(\hat{d}_i, \bar{d}_j)\right]$ and stores that value in $\bar{\Delta}_i$ (line 16).

When an agent x_i receives a BACKTRACK message from its child x_s, it adds its child's delta variables Δ_s and $\bar{\Delta}_s$ to its own delta variables Δ_i and $\bar{\Delta}_i$, respectively (lines 37-38). Thus, Δ_i and $\bar{\Delta}_i$ can be seen as a sum of local differences from the leaf agents in its subtree all the way up to the current agent. The delta variables Δ_i and $\bar{\Delta}_i$ of the root agent are thus the differences in the *global* solution quality (see Lemmas 2 and 3).

In each iteration, the root agent accumulates the differences in global solution quality Δ_i and stores this accumulated value in delta variable δ (line 43). Since δ is initially initialized to 0 (line 8), that means that its value is the difference in global solution quality between the current solution and the first initial complete solution (i.e., each variable x_j takes on value $\textbf{ValInit}(x_j)$). Similarly, the root agent also computes the delta value $\bar{\delta}$, which is the same as δ except that it uses the best-response solution instead of the current solution in the current iteration (line 42).

If δ is greater than both $\bar{\delta}$ and the maximum difference δ^*, which means that the current solution is better than both the best-response solution and the best solution found so far, then the root agent updates the maximum difference δ^* to δ and its best value d_i^* to its current value d_i (lines 44-46). Similarly, if $\bar{\delta}$ is greater than both δ and the maximum difference δ^*, which means that the best-response solution is better than both the current solution and the best solution found so far, then the root agent updates the maximum difference δ^* to $\bar{\delta}$ and its best value d_i^* to its best-response value $\bar{d}_i$ (lines 48-50).

After finding a better solution, the root agent needs to inform other agents to update their respective best values. There are the following two cases:

- **Case 1:** $\delta \geq \bar{\delta}$. The root agent needs to inform the other agents to update their best values to their current values. It does so by setting its t_i^* index to the current iteration (line 47) and sends this information down to its children in VALUE messages (line 17). When an agent x_i receives a VALUE message from its parent, it updates its best value d_i^* to its current value d_i (lines 28-29), sets its t_i^* index to the received t_s^* index (line 30), and sends this information down to its children in VALUE messages (line 32 and 17). This process continues until all agents update their best values.
- **Case 2:** $\bar{\delta} \geq \delta$. This case is identical to case 1 except that the agents need to update their best values to their best-response values. Instead of updating its t_i^* index, the root agent updates its $\bar{t}_i^*$ index instead (line 51). When an agent x_i receives a VALUE message from its parent, instead of updating its best value d_i^* to its current value, it updates it to its best-response value $\bar{d}_i$ (lines 25-26). Additionally, instead of updating its t_i^* index, it sets its $\bar{t}_i^*$ index to the received $\bar{t}_s^*$. All other processes are identical to case 1.

Therefore, when a better solution is found, all agents in the SD-Gibbs algorithm update their best values by the end of the next iteration. The algorithm can either terminate after a given number of iterations or when no better solution is found for a given number of consecutive iterations.[6]

4.2.1 Example Partial Trace

We now describe one possible partial trace of SD-Gibbs on our example DCOP with the pseudo-tree shown in Figure 1. Table 1 shows the trace of the update of all variables of all agents. References to line numbers are with respect to the pseudocode shown in Algorithm 2 and its subsequent procedures.

6. One can slightly optimize the algorithm by having the agents (1) send their current values in BACKTRACK messages instead of VALUE messages to their parents; and (2) send smaller VALUE messages, which do not contain time indices, to all pseudo-children. We describe the unoptimized version here for ease of understanding.

	Itera-	Iteration 1					Iteration 2					...
	tion 0	1	2	3	4	5	1	2	3	4	5	
X_1	$\langle x_2, 0\rangle$	$\langle x_2, 0\rangle$	$\langle x_2, 0\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	...
	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 1\rangle$	$\langle x_4, 1\rangle$	$\langle x_4, 1\rangle$	$\langle x_4, 1\rangle$	$\langle x_4, 1\rangle$	$\langle x_4, 1\rangle$	$\langle x_4, 1\rangle$	...
$\bar{X}_1$	$\langle x_2, 0\rangle$	$\langle x_2, 0\rangle$	$\langle x_2, 0\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	...
	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	...
t_1	0	1	1	1	1	1	2	2	2	2	2	...
$\hat{d}_1$	0	0	0	0	0	0	0	0	0	0	0	...
d_1	0	0	0	0	0	0	0	0	0	0	0	...
$\bar{d}_1$	0	1	1	1	1	1	0	0	0	0	0	...
d_1^*	0	0	0	0	0	1	1	1	1	1	0	...
Δ_1	0	0	0	0	0	2	0	0	0	0	3	...
$\bar{\Delta}_1$	0	4	4	4	4	5	0	0	0	0	4	...
δ	0	0	0	0	0	2	2	2	2	2	5	...
$\bar{\delta}$	0	0	0	0	0	5	5	5	5	5	6	...
t_1^*	0	0	0	0	0	0	0	0	0	0	0	...
$\bar{t}_1^*$	0	0	0	0	0	1	1	1	1	1	2	...
X_2	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	...
	$\langle x_3, 0\rangle$	$\langle x_3, 0\rangle$	$\langle x_3, 0\rangle$	$\langle x_3, 0\rangle$	$\langle x_3, 1\rangle$	$\langle x_3, 1\rangle$	$\langle x_3, 1\rangle$	$\langle x_3, 1\rangle$	$\langle x_3, 1\rangle$	$\langle x_3, 0\rangle$	$\langle x_3, 0\rangle$	...
	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 1\rangle$	$\langle x_4, 1\rangle$	$\langle x_4, 1\rangle$	$\langle x_4, 1\rangle$	$\langle x_4, 1\rangle$	$\langle x_4, 1\rangle$	$\langle x_4, 1\rangle$	...
$\bar{X}_2$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 1\rangle$	$\langle x_1, 1\rangle$	$\langle x_1, 1\rangle$	$\langle x_1, 1\rangle$	$\langle x_1, 1\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	...
	$\langle x_3, 0\rangle$	$\langle x_3, 0\rangle$	$\langle x_3, 0\rangle$	$\langle x_3, 0\rangle$	$\langle x_3, 0\rangle$	$\langle x_3, 0\rangle$	$\langle x_3, 0\rangle$	$\langle x_3, 0\rangle$	$\langle x_3, 0\rangle$	$\langle x_3, 0\rangle$	$\langle x_3, 0\rangle$	...
	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	$\langle x_4, 0\rangle$	...
t_2	0	0	1	1	1	1	1	2	2	2	2	...
$\hat{d}_2$	0	0	0	0	0	0	0	1	1	1	1	...
d_2	0	0	1	1	1	1	1	1	1	1	1	...
$\bar{d}_2$	0	0	1	1	1	1	1	1	1	1	1	...
d_2^*	0	0	0	0	0	0	0	1	1	1	1	...
Δ_2	0	0	6	6	2	2	2	0	0	3	3	...
$\bar{\Delta}_2$	0	0	1	1	1	1	1	0	0	4	4	...
t_2^*	0	0	0	0	0	0	0	0	0	0	0	...
$\bar{t}_2^*$	0	0	0	0	0	0	0	1	1	1	1	...
X_3	$\langle x_2, 0\rangle$	$\langle x_2, 0\rangle$	$\langle x_2, 0\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	...
$\bar{X}_3$	$\langle x_2, 0\rangle$	$\langle x_2, 0\rangle$	$\langle x_2, 0\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	...
t_3	0	0	0	1	1	1	1	1	2	2	2	...
$\hat{d}_3$	0	0	0	0	0	0	0	0	1	1	1	...
d_3	0	0	0	1	1	1	1	1	0	0	0	...
$\bar{d}_3$	0	0	0	0	0	0	0	0	0	0	0	...
d_3^*	0	0	0	0	0	0	0	0	0	0	0	...
Δ_3	0	0	0	-3	-3	-3	-3	-3	3	3	3	...
$\bar{\Delta}_3$	0	0	0	0	0	0	0	0	3	3	3	...
t_3^*	0	0	0	0	0	0	0	0	0	0	0	...
$\bar{t}_3^*$	0	0	0	0	0	0	0	0	1	1	1	...
X_4	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	...
	$\langle x_2, 0\rangle$	$\langle x_2, 0\rangle$	$\langle x_2, 0\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	...
$\bar{X}_4$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 1\rangle$	$\langle x_1, 1\rangle$	$\langle x_1, 1\rangle$	$\langle x_1, 1\rangle$	$\langle x_1, 1\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	$\langle x_1, 0\rangle$	...
	$\langle x_2, 0\rangle$	$\langle x_2, 0\rangle$	$\langle x_2, 0\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	$\langle x_2, 1\rangle$	...
t_4	0	0	0	1	1	1	1	1	2	2	2	...
$\hat{d}_4$	0	0	0	0	0	0	0	0	1	1	1	...
d_4	0	0	0	1	1	1	1	1	1	1	1	...
$\bar{d}_4$	0	0	0	0	0	0	0	0	0	0	0	...
d_4^*	0	0	0	0	0	0	0	0	0	0	0	...
Δ_4	0	0	0	-1	-1	-1	-1	-1	0	0	0	...
$\bar{\Delta}_4$	0	0	0	0	0	0	0	0	1	1	1	...
t_4^*	0	0	0	0	0	0	0	0	0	0	0	...
$\bar{t}_4^*$	0	0	0	0	0	0	0	0	1	1	1	...

Table 1: Partial Trace of the SD-Gibbs Variable Updates

We describe the trace in terms of *iterations*, where in each iteration (except for the first), the agents complete a single VALUE phase that propagates information from the root to the leaf agents and a single BACKTRACK phase that propagates information from the leaf agents back to the root. Each iteration is then divided into five sequential *steps*, where in each step, the agents process incoming messages (if any), perform local computations and updates based on the information received in the incoming messages, and send outgoing messages to their neighbors.

Iteration 0: All the agents initially create a pseudo-tree, for example, by running a Distributed DFS algorithm (Hamadi et al., 1998) (line 1). After the pseudo-tree is constructed, each agent x_i initializes all its values to $\mathbf{ValInit}(x_i)$ (line 3), all its contexts to $\{(x_j, \mathbf{ValInit}(x_j)) \mid x_j \in N_i\}$ (line 4), all its time indices to 0 (line 5), and all its delta values to 0 (line 6). In this trace, assume that $\mathbf{ValInit}(x_j) = 0$ for all agents x_j. Then, the root starts the next iteration.

Iteration 1: As mentioned above, to ease readability, we break down the actions of the agents in this iteration into five sequential steps:

- Step 1:
  - **Agent x_1:** After initializing the initialization phase in Iteration 0, the root agent calls the Sample procedure (line 9), which updates its previous value $\hat{d}_1$ to its current value $d_1 = \mathbf{ValInit}(x_1) = 0$ (line 12) and samples its new value d_1 based on Equation 21 (line 13). Assume that the new sampled value $d_1 = 0$. Thus, the agent's value did not change in this iteration. The agent also computes its best-response value $\bar{d}_1 = 1$ (line 14).

    The agent then computes its delta values Δ_1 and $\bar{\Delta}_1$ (lines 15-16). The value Δ_1 is the difference in the utility of the agent between its solution in the previous and current iterations. Since the context X_1 and value d_1 of the agent is the same in both iterations, the difference is $\Delta_1 = 0$. The value $\bar{\Delta}_1$ is the difference in the utility of the agent between its best-response solution in the previous and current iterations. As the agent changed its best-response value from 0 to 1, the difference is $\bar{\Delta}_1 = [F_{12}(1,0) - F_{12}(0,0)] + [F_{14}(1,0) - F_{14}(0,0)] = [1 - (-1)] + [1 - (-1)] = 4$.

    Finally, the agent sends the following messages (line 17):
    - a VALUE message $(x_1, d_1 = 0, \bar{d}_1 = 1, t_1^* = 0, \bar{t}_1^* = 0)$ to its neighbor x_2; and
    - a VALUE message $(x_1, d_1 = 0, \bar{d}_1 = 1, t_1^* = 0, \bar{t}_1^* = 0)$ to its neighbor x_4.
  - **Agents x_2, x_3, and x_4:** The agents are idle in this step.
- Step 2:
  - **Agents x_1 and x_3:** The agents are idle in this step.
  - **Agent x_2:** After receiving a VALUE message from its parent x_1, agent x_2 updates the current value of its parent in its context X_2 with the received current value (line 18) and updates the best-response value of its parent in its context $\bar{X}_2$ with the received best-response value (line 20).

    The agent then calls the Sample procedure (line 32), which updates its previous value $\hat{d}_2$ to its current value $d_2 = \mathbf{ValInit}(x_2) = 0$ (line 12) and samples its new

value d_2 based on Equation 21 (line 13). Assume that the new sampled value $d_2 = 1$. Thus, the agent's value changed in this iteration. The agent also computes its best-response value $\bar{d}_2 = 1$ (line 14), which is coincidentally the same value as its current value $d_2 = 1$. Note that the computation of the agent's current value is based on context X_2 while the computation of the agent's best-response value is based on context $\bar{X}_2$, and the value of parent x_1 is different in the two contexts – it is $d_1 = 0$ in X_2 and $\bar{d}_1 = 1$ in $\bar{X}_2$.

The agent then computes the delta values Δ_2 and $\bar{\Delta}_2$ (lines 15-16), where $\Delta_2 = [F_{12}(0,1) - F_{12}(0,0)] + [F_{23}(1,0) - F_{23}(0,0)] + [F_{24}(1,0) - F_{24}(0,0)] = [1-(-1)] + [1-(-1)] + [1-(-1)] = 6$ and $\bar{\Delta}_2 = [F_{12}(1,1) - F_{12}(1,0)] + [F_{23}(1,0) - F_{23}(0,0)] + [F_{24}(1,0) - F_{24}(0,0)] = [-2-1] + [1-(-1)] + [1-(-1)] = 1$.

Finally, the agent sends the following messages (line 17):

- a VALUE message $(x_2, d_2 = 1, \bar{d}_2 = 1, t_2^* = 0, \bar{t}_2^* = 0)$ to its neighbor x_1;
- a VALUE message $(x_2, d_2 = 1, \bar{d}_2 = 1, t_2^* = 0, \bar{t}_2^* = 0)$ to its neighbor x_3; and
- a VALUE message $(x_2, d_2 = 1, \bar{d}_2 = 1, t_2^* = 0, \bar{t}_2^* = 0)$ to its neighbor x_4.

- **Agent x_4:** After receiving a VALUE message from its pseudo-parent x_1, agent x_4 updates the current value of its pseudo-parent in its context X_4 with the received current value (line 18) and updates the best-response value of its pseudo-parent in its context $\bar{X}_4$ with the received best-response value (line 20).

- Step 3:

  - **Agent x_1:** After receiving a VALUE message from its child x_2, agent x_1 updates the current and best-response values of its child in its contexts X_1 and $\bar{X}_1$, respectively (lines 18 and 22).
  - **Agent x_2:** The agent is idle in this step.
  - **Agent x_3:** After receiving a VALUE message from its parent x_2, agent x_3 updates the current and best-response values of its parent in its contexts X_3 and $\bar{X}_3$, respectively (lines 18 and 20).

    The agent then calls the Sample procedure (line 32), which updates its previous value $\hat{d}_3$ to its current value $d_3 = \textbf{ValInit}(x_3) = 0$ and samples its new value d_3 (lines 12-13). Assume that the new sampled value $d_3 = 1$. The agent also computes its best-response value $\bar{d}_3 = 0$ (line 14). The agent then computes the delta values Δ_3 and $\bar{\Delta}_3$ (lines 15-16), where $\Delta_3 = F_{23}(1,1) - F_{23}(1,0) = -2 - 1 = -3$ and $\bar{\Delta}_3 = F_{23}(1,0) - F_{23}(1,0) = 1 - 1 = 0$.

    Finally, the agent sends the following messages (lines 17 and 34):

    - a VALUE message $(x_3, d_3 = 1, \bar{d}_3 = 0, t_3^* = 0, \bar{t}_3^* = 0)$ to its neighbor x_2; and
    - a BACKTRACK message $(x_3, \Delta_3 = -3, \bar{\Delta}_3 = 0)$ to its parent x_2.

  - **Agent x_4:** After receiving a VALUE message from its parent x_2, agent x_4 updates the current and best-response values of its parent in its contexts X_4 and $\bar{X}_4$, respectively (lines 18 and 20).

    The agent then calls the Sample procedure (line 32), which updates its previous value $\hat{d}_4$ to its current value $d_4 = \textbf{ValInit}(x_4) = 0$ and samples its new value d_4

(lines 12-13). Assume that the new sampled value $d_4 = 1$. The agent also computes its best-response value $\bar{d}_4 = 0$ (line 14). The agent then computes the delta values Δ_4 and $\bar{\Delta}_4$ (lines 15-16), where $\Delta_4 = [F_{14}(0,1) - F_{14}(0,0)] + [F_{24}(1,1) - F_{24}(1,0)] = [1-(-1)] + [-2-1] = -1$ and $\bar{\Delta}_4 = [F_{14}(1,0) - F_{14}(1,0)] + [F_{24}(1,0) - F_{24}(1,0)] = [1-1] + [1-1] = 0$.

Finally, the agent sends the following messages (lines 17 and 34):

- a VALUE message $(x_4, d_4 = 1, \bar{d}_4 = 0, t_4^* = 0, \bar{t}_4^* = 0)$ to its neighbor x_1;
- a VALUE message $(x_4, d_4 = 1, \bar{d}_4 = 0, t_4^* = 0, \bar{t}_4^* = 0)$ to its neighbor x_2; and
- a BACKTRACK message $(x_4, \Delta_4 = -1, \bar{\Delta}_4 = 0)$ to its parent x_2.

- Step 4:
  - **Agent x_1:** After receiving a VALUE message from its pseudo-child x_4, agent x_1 updates the current and best-response values of its pseudo-child in its contexts X_1 and $\bar{X}_1$, respectively (lines 18 and 22).
  - **Agent x_2:** After receiving a VALUE message from each of its children x_3 and x_4, agent x_2 updates the current and best-response values of its children in its contexts X_2 and $\bar{X}_2$, respectively (lines 18 and 22). After receiving a BACKTRACK message from each of its children x_3 and x_4, agent x_2 updates its delta values $\Delta_2 = \Delta_2 + \Delta_3 + \Delta_4 = 6 + (-3) + (-1) = 2$ and $\bar{\Delta}_2 = \bar{\Delta}_2 + \bar{\Delta}_3 + \bar{\Delta}_4 = 1 + 0 + 0 = 1$ (lines 37-38).[7]

    The agent then sends the following message (line 40):

    - a BACKTRACK message $(x_2, \Delta_2 = 2, \bar{\Delta}_2 = 1)$ to its parent x_1.
  - **Agents x_3 and x_4:** The agents are idle in this step.

- Step 5:
  - **Agent x_1:** After receiving a BACKTRACK message from its child x_2, agent x_1 updates its delta values $\Delta_1 = \Delta_1 + \Delta_2 = 0 + 2 = 2$ and $\bar{\Delta}_1 = \bar{\Delta}_1 + \bar{\Delta}_2 = 4 + 1 = 5$ (lines 37-38).

    As it is the root, it also updates its delta values $\bar{\delta}$ and δ (lines 42-43), which are the shifted utility[8] of the best-response solution and current solution, respectively. The updated values are $\bar{\delta} = \delta + \bar{\Delta}_1 = 0 + 5 = 5$ and $\delta = \delta + \Delta_1 = 0 + 2 = 2$. As $\bar{\delta} > \delta$, it means that the best-response solution has a larger utility than the current solution.[9] Further, as $\bar{\delta} > \delta^*$, indicating that the best-response solution is better than the best solution found so far,[10] the agent updates $\delta^* = \bar{\delta}$, its best value $d_1^* = \bar{d}_1$ to its best-response value, and the iteration that it found its best best-response solution $\bar{t}_1^* = t_1$ to the current iteration (lines 48-51). The information that the best solution found

7. These updates are actually performed sequentially as each BACKTRACK message is received, but we described them as being aggregated together for brevity.
8. They are shifted by the utility of the initial complete solution, i.e., the solution where all variables x_i are assigned $\mathbf{ValInit}(x_i)$.
9. The (unshifted) utility of the best-response and current solutions is 1 and -2, respectively. They are shifted by the utility of the initial solution, which is -4, to get the $\bar{\delta}$ and δ values. In other words, $5 = 1 - (-4)$ and $2 = -2 - (-4)$.
10. δ^* reflects the shifted utility of the best solution found so far.

so far is the best-response solution found in this iteration will be propagated to the other agents in the next iteration.

- **Agents x_2, x_3, and x_4:** The agents are idle in this step.

Iteration 2: In this iteration, all the agents also execute many of the same operations as in the previous iteration, i.e., they all sample their new values d_i; compute their best-response values $\bar{d}_i$; compute their delta values Δ_i and $\bar{\Delta}_i$ based on their current and best-response values; and update their contexts X_i and $\bar{X}_i$ based on the current and best-response values received from their neighbors. Therefore, we will abbreviate the description of these procedures. Instead, we will focus on how the agents identify that their best solution found so far is the best-response solution found in the previous iteration.

- Step 1:
  - **Agent x_1:** The root agent starts the iteration by sampling its current value $d_1 = 0$; computes its best-response value $\bar{d}_1 = 0$; and computes its delta values $\Delta_1 = \bar{\Delta}_1 = 0$ (lines 13-16). Finally, the agent sends the following messages (line 17):
    - a VALUE message $(x_1, d_1 = 0, \bar{d}_1 = 0, t_1^* = 0, \bar{t}_1^* = 1)$ to its neighbor x_2; and
    - a VALUE message $(x_1, d_1 = 0, \bar{d}_1 = 0, t_1^* = 0, \bar{t}_1^* = 1)$ to its neighbor x_4.
  - **Agents x_2, x_3, and x_4:** The agents are idle in this step.

- Step 2:
  - **Agents x_1 and x_3:** The agents are idle in this step.
  - **Agent x_2:** After receiving a VALUE message from its parent x_1, agent x_2 updates its contexts X_2 and $\bar{X}_2$ (lines 18 and 20). The agent then checks if either of the conditions on lines 25 and 28 is true. These conditions indicate whether the root agent identified that the agents collectively found a better solution. If the condition on line 25 is true, then it means that the best-response solution found in iteration $\bar{t}_1^*$ is the best solution found so far. Similarly, if the condition on line 28 is true, then it means that the current solution found in iteration t_1^* is the best solution found so far. In this trace, the condition on line 25 is true since $1 = \bar{t}_1^* \geq t_1^* = 0$ and $1 = \bar{t}_1 > \max\{t_2^*, \bar{t}_2^*\} = \max\{0, 0\} = 0$. Therefore, the agent updates its best value $d_2^* = \bar{d}_2 = 1$ to its best-response value in the previous iteration (line 26) and updates the iteration that it found that value $\bar{t}_2^* = \bar{t}_1^* = 1$ to the iteration by its parent (line 27).

    The agent then samples its current value $d_2 = 1$; computes its best-response value $\bar{d}_2 = 1$; and computes its delta values $\Delta_2 = 0 = \bar{\Delta}_2 = 0$ (lines 13-16). Finally, the agent sends the following messages (line 17):
    - a VALUE message $(x_2, d_2 = 1, \bar{d}_2 = 1, t_2^* = 0, \bar{t}_2^* = 1)$ to its neighbor x_1;
    - a VALUE message $(x_2, d_2 = 1, \bar{d}_2 = 1, t_2^* = 0, \bar{t}_2^* = 1)$ to its neighbor x_3; and
    - a VALUE message $(x_2, d_2 = 1, \bar{d}_2 = 1, t_2^* = 0, \bar{t}_2^* = 1)$ to its neighbor x_4.
  - **Agent x_4:** After receiving a VALUE message from its pseudo-parent x_1, agent x_4 updates its contexts X_4 and $\bar{X}_4$ (lines 18 and 20).

- Step 3:
  - **Agent x_1:** After receiving a VALUE message from its child x_2, agent x_1 updates its contexts X_1 and $\bar{X}_1$ (lines 18 and 22).
  - **Agent x_2:** The agent is idle in this step.
  - **Agent x_3:** After receiving a VALUE message from its parent x_2, agent x_3 updates its contexts X_3 and $\bar{X}_3$ (lines 18 and 20). And since the condition on line 25 is true – $1 = \bar{t}_2^* \geq t_2^* = 0$ and $1 = \bar{t}_2 > \max\{t_3^*, \bar{t}_3^*\} = \max\{0, 0\} = 0$ – the agent updates its best value $d_3^* = \bar{d}_3 = 0$ to its best-response value in the previous iteration (line 26) and updates the iteration that it found that value $\bar{t}_3^* = \bar{t}_2^* = 1$ to the iteration by its parent (line 27).

    The agent then samples its current value $d_3 = 0$; computes its best-response value $\bar{d}_3 = 0$; and computes its delta values $\Delta_3 = \bar{\Delta}_3 = [F_{23}(1,0) - F_{23}(1,1)] = 1 - (-2) = 3$ (lines 13-16). Finally, the agent sends the following messages (lines 17 and 34):
    - a VALUE message $(x_3, d_3 = 0, \bar{d}_3 = 0, t_3^* = 0, \bar{t}_3^* = 1)$ to its neighbor x_2; and
    - a BACKTRACK message $(x_3, \Delta_3 = 3, \bar{\Delta}_3 = 3)$ to its parent x_2.
  - **Agent x_4:** After receiving a VALUE message from its parent x_2, agent x_4 updates its contexts X_4 and $\bar{X}_4$ (lines 18 and 20). And since the condition on line 25 is true, the agent updates its best value $d_4^* = \bar{d}_4 = 0$ to its best-response value in the previous iteration (line 26) and updates the iteration that it found that value $\bar{t}_4^* = \bar{t}_2^* = 1$ to the iteration by its parent (line 27).

    The agent then samples its current value $d_4 = 1$; computes its best-response value $\bar{d}_4 = 0$; and computes its delta values $\Delta_4 = 0$ and $\bar{\Delta}_4 = 1$ (lines 13-16). Finally, the agent sends the following messages (lines 17 and 34):
    - a VALUE message $(x_4, d_4 = 1, \bar{d}_4 = 0, t_4^* = 0, \bar{t}_4^* = 1)$ to its neighbor x_1;
    - a VALUE message $(x_4, d_4 = 1, \bar{d}_4 = 0, t_4^* = 0, \bar{t}_4^* = 1)$ to its neighbor x_2; and
    - a BACKTRACK message $(x_4, \Delta_4 = 0, \bar{\Delta}_4 = 1)$ to its parent x_2.
  - Steps 4–5: At the end of Step 3, all the agents updated their best values d_i^* to their best-response values in the previous iteration since that is the best solution found so far. In Steps 4 and 5, the operations of the agents are very similar to their operations in Steps 4 and 5 of the previous iteration, except that some of the current, best-response, and delta values may have changed. The key difference is that the root agent will realize that the best-response solution of this iteration is better than the best solution found so far (it is actually an optimal solution), and this information will be propagated to the other agents in the next iteration.

Subsequent Iterations: In the third and subsequent iterations, the operations of the agents are very similar to their operations in the previous iteration, except that some of the current, best-response, and delta values may have changed. The agents will continue to iteratively seek better solutions until their termination condition is reached (e.g., they have reached a time out or a maximum number of iterations).

4.2.2 Theoretical Properties

Like Gibbs, the SD-Gibbs algorithm also samples the values sequentially and samples them based on the same equation (Equation 21). The main difference is that Gibbs samples down

a pseudo-chain (a pseudo-tree without sibling subtrees), while SD-Gibbs samples sibling subtrees in parallel. However, this difference only speeds up the sampling process and does not affect the correctness of the algorithm since agents in sibling subtrees are conditionally independent of each other given the values of their common ancestors. Thus, we will show several properties that hold for centralized Gibbs and, thus, also hold for SD-Gibbs.

Gibbs can be viewed as a variant of the simulated annealing algorithm with a fixed temperature. Therefore, our analysis below follows the analysis of the simulated annealing algorithm (Rajasekaran, 2000).

Lemma 1. *A lower bound on the probability for an agent x_i choosing its value d_i^* in an optimal solution is p_i^*:*

$$p_i^* = \min_{X_i} \frac{\exp\left[\sum_{\langle x_j, d_j\rangle \in X_i} F_{ij}(d_i^*, d_j)\right]}{\sum_{d_i' \in D_i} \exp\left[\sum_{\langle x_j, d_j\rangle \in X_i} F_{ij}(d_i', d_j)\right]} \tag{22}$$

Proof. Given a particular context X_i of an agent x_i, the probability of it choosing d_i^* is: $\frac{\exp\left[\sum_{\langle x_j,d_j\rangle \in X_i} F_{ij}(d_i^*,d_j)\right]}{\sum_{d_i' \in D_i} \exp\left[\sum_{\langle x_j,d_j\rangle \in X_i} F_{ij}(d_i',d_j)\right]}$ according to Equation 21. Therefore, the smallest of these probabilities across all possible contexts must be a lower bound on the probability of the agent choosing d_i^*. □

Theorem 1. *SD-Gibbs converges within k iterations with at least probability $1-(1-p)^k$, where $p = \prod_{i=1}^n p_i^*$ and p_i^* is as defined in Lemma 1.*

Proof. At any sampling step, each agent x_i chooses its optimal value d_i^* with probability at least p_i^* (Lemma 1). Therefore, the joint probability for all agents to collectively choose an optimal solution (i.e., they all choose their respective optimal values) from any current solution is at least $p = \prod_{i=1}^n p_i^*$.

As the probability to choose an optimal solution in each iteration is at least p, the cumulative probability distribution to choose an optimal solution in k iterations is at least the cumulative distribution to have one success in k iterations when generating Bernoulli trials with probability p at each step. Therefore, the cumulative probability distribution of a geometric distribution with success probability p, which is $1-(1-p)^k$, is a lower bound on the cumulative probability distribution of SD-Gibbs to find an optimal solution in k iterations. □

Corollary 1. *The probability that SD-Gibbs converges to an optimal solution approaches 1 as its number of iterations approaches infinity.*

Lemma 2. *In each iteration, after the root agent receives a BACKTRACK message from each of its children and updates its delta variable Δ_i, that updated value is the difference in solution quality between the current solution and the previous solution.*

Proof. The accumulated value of the root agent's delta variable includes the evaluation of $F_{ij}(d_i, d_j) - F_{ij}(\hat{d}_i, d_j)$ by each agent x_i for all utility functions with its neighbors x_j, where d_j is the value of x_j in context X_i (line 15). Thus, that value is accumulated twice – once by each agent involved in the utility function. Let the two agents be agents x_p and x_c, where agent x_p is a parent or pseudo-parent of agent x_c. Also, let t be the current iteration and d_i^t be the value of agent x_i in iteration t. Then, the contribution of the two agents combined is:

$$\begin{aligned}
&F_{pc}(d_p, d_c \mid \langle x_c, d_c\rangle \in X_p) - F_{pc}(\hat{d}_p, d_c \mid \langle x_c, d_c\rangle \in X_p) \\
&\qquad + F_{pc}(d_p, d_c \mid \langle x_p, d_p\rangle \in X_c) - F_{pc}(d_p, \hat{d}_c \mid \langle x_p, d_p\rangle \in X_c) \\
&= F_{pc}(d_p^t, d_c^{t-1}) - F_{pc}(d_p^{t-1}, d_c^{t-1}) + F_{pc}(d_p^t, d_c^t) - F_{pc}(d_p^t, d_c^{t-1}) \\
&= F_{pc}(d_p^t, d_c^t) - F_{pc}(d_p^{t-1}, d_c^{t-1})
\end{aligned}$$

which is the difference in utility of that function between the current solution and the previous solution. The sum of this difference over all utility functions is thus the difference in solution quality between the current solution and the previous solution. □

Lemma 3. *In each iteration, after the root agent receives a BACKTRACK message from each of its children and updates its delta variable $\bar{\Delta}_i$, that updated value is the difference in solution quality between the current best-response solution and the previous solution.*

Proof. The accumulated value of the root agent's delta variable includes the evaluation of $F_{ij}(\bar{d}_i, \bar{d}_j) - F_{ij}(\hat{d}_i, \bar{d}_j)$ by each agent x_i for all utility functions with its neighbors x_j, where $\bar{d}_j$ is the value of x_j in context $\bar{X}_i$ (line 16). Thus, that value is accumulated twice – once by each agent involved in the utility function. Let the two agents be agents x_p and x_c, where agent x_p is a parent or pseudo-parent of agent x_c. Also, let t be the current iteration, d_i^t be the value of agent x_i in iteration t, and $\bar{d}_i^t$ be the best-response value of agent x_i in iteration t. Then, the contribution of the two agents combined is:

$$\begin{aligned}
&F_{pc}(\bar{d}_p, \bar{d}_c \mid \langle x_c, \bar{d}_c\rangle \in \bar{X}_p) - F_{pc}(\hat{d}_p, \bar{d}_c \mid \langle x_c, \bar{d}_c\rangle \in \bar{X}_p) \\
&\qquad + F_{pc}(\bar{d}_p, \bar{d}_c \mid \langle x_p, \bar{d}_p\rangle \in \bar{X}_c) - F_{pc}(\bar{d}_p, \hat{d}_c \mid \langle x_p, \bar{d}_p\rangle \in \bar{X}_c) \\
&= F_{pc}(\bar{d}_p^t, d_c^{t-1}) - F_{pc}(d_p^{t-1}, d_c^{t-1}) + F_{pc}(\bar{d}_p^t, \bar{d}_c^t) - F_{pc}(\bar{d}_p^t, d_c^{t-1}) \\
&= F_{pc}(\bar{d}_p^t, \bar{d}_c^t) - F_{pc}(d_p^{t-1}, d_c^{t-1})
\end{aligned}$$

which is the difference in utility of that function between the current best-response solution and the previous solution. The sum of this difference over all utility functions is thus the difference in solution quality between the current best-response solution and the previous solution. □

Lemma 4. *After the root agent receives a BACKTRACK message from each of its children and updates its delta variables δ and $\bar{\delta}$, those updated values are the difference in solution quality between the current complete solution and the initial complete solution (i.e., all variables x_j are assigned* **ValInit**(x_j)*) and the difference in solution quality between the current complete best-response solution and the initial complete solution, respectively.*

Proof. We will first prove by induction that the root agent's δ variable is the difference in solution quality between the current complete solution and the initial complete solution.

- **Iteration 1:** Δ_i is the difference in the solution quality between the complete solution in iteration 1 and the initial complete solution of iteration 0 (Lemma 2). Therefore, since $\delta = 0 + \Delta_i$ (lines 8 and 43), the lemma holds for iteration 1.
- **Induction Assumption:** Assume that the lemma holds for all iterations up to iteration $k-1$.
- **Iteration k:** Δ_i is the difference in the solution quality between the complete solution in iteration k and the complete solution in iteration $k-1$ (Lemma 2). Additionally, the δ value prior to the execution of line 43 is the difference in the solution quality between the complete solution in iteration $k-1$ and the initial complete solution of iteration 0 (induction assumption). Therefore, the δ value after the execution of line 43 is the difference in the solution quality between the complete solution in iteration k and the initial complete solution of iteration 0.

The proof for the case that the root agent's $\bar{\delta}$ variable is the difference in solution quality between the current complete best-response solution and the initial complete solution follows the same principle. □

Theorem 2. *Upon termination, SD-Gibbs returns the best solution found.*

Proof. There are the following two cases:

- The best solution is a regularly sampled solution: Thus, $\delta \geq \bar{\delta}$ (Lemma 4) and $\delta > \delta^*$. Then, the root agent will update its best value and best time index (lines 46-47) and sends these values in VALUE messages to its children and pseudo-children (lines 53 and 17). When an agent receives this VALUE message, it too updates its best value and best time index (lines 28-31) and sends these values in VALUE messages to its children and pseudo-children. This process propagates down the pseudo-tree until all agents have updated their best values and best time indices.
- The best solution is a best-response sampled solution. Thus, $\bar{\delta} \geq \delta$ (Lemma 4) and $\bar{\delta} > \delta^*$. Then, the root agent will update its best value and best time index (lines 50-51) and sends these values in VALUE messages to its children and pseudo-children (lines 53 and 17). When an agent receives this VALUE message, it too updates its best value and best time index (lines 25-27) and sends these values in VALUE messages to its children and pseudo-children. This process propagates down the pseudo-tree until all agents have updated their best values and best time indices.

SD-Gibbs thus returns the best solution in either of the above two cases. □

Theorem 3. *Each SD-Gibbs iteration takes a finite amount of time.*

Proof. In each iteration, each agent sends exactly one VALUE message to each of its neighbors and sends exactly one BACKTRACK message to its parent. The root agent starts the VALUE propagation phase by sending the first VALUE message and the leaf agents start the BACKTRACK propagation phase by sending BACKTRACK messages upon receiving VALUE messages. Since messages are never lost, the root will eventually receive a BACKTRACK message from each of its children, which ends the current iteration. □

Theorem 4. *The memory complexity of each agent in SD-Gibbs is $O(|\mathcal{X}|)$, that is, it is linear in the number of agents in the problem.*

Proof. Each agent x_i needs to store a context X_i and a context $\bar{X}_i$, which contain agent-value pairs of all neighboring agents $x_j \in N_i$. Additionally agent x_i needs to store the values d_i, $\hat{d}_i$, d_i^*, $\bar{d}_i$; time indices t_i, t_i^*, $\bar{t}_i^*$; and delta variables Δ_i, $\bar{\Delta}_i$. If it is the root, then it also needs to store delta variables δ, δ^*, and $\bar{\delta}$. Each of these variables is a single value. Therefore, its memory complexity is $O(|\mathcal{X}|)$. □

Theorem 5. *The amount of information passed around in the network per iteration by SD-Gibbs agents is $O(|\mathcal{X}|^2)$, that is, it is polynomial in the number of agents in the problem.*

Proof. Each agent x_i needs to send exactly one VALUE message to each neighbor and exactly one BACKTRACK message to its parent in each iteration, and each message contains a constant number of values (each VALUE message contains 5 values and each BACKTRACK message contains 3 values). Thus, the amount of information passed around in the network per iteration is $O(|\mathcal{X}|^2)$. □

4.3 Parallel Distributed Gibbs

We now describe the *Parallel Distributed Gibbs* (PD-Gibbs) algorithm. Algorithm 3 shows the pseudo-code. The main differences between PD-Gibbs and SD-Gibbs, its sequential counterpart, are the following:

- In SD-Gibbs, the agents along the same branch of the pseudo-tree sample sequentially in each iteration, while, in PD-Gibbs, multiple agents along the same branch of the pseudo-tree can sample in parallel in each iteration. However, an agent can only sample if all its neighbors are not sampling in the same iteration. In order to enforce this restriction, each agent maintains two additional variables: p_i and p_{max}, where p_i is the priority of that agent in sampling (the smaller the value of p_i, the higher its priority to sample) and $p_{max} = \max_{x_j \in \mathcal{X}} p_j$ is the largest priority value over all agents in the problem. The agents initialize p_i and p_{max} through PRIORITY, PMAXUP, and PMAXDOWN messages, which we will describe in detail later.
- Since agents in PD-Gibbs can sample in parallel, each agent now needs to maintain its current value $d_i^{t_i}$, best-response value $\bar{d}_i^{t_i}$, and delta variables $\Delta_i^{t_i}$, $\bar{\Delta}_i^{t_i}$, and δ^{t_i} for multiple past iterations so that it can retrieve its value in a past iteration where the best solution is found. These variables are thus indexed by the agent's time index t_i in superscript. Additionally, this removes the need to maintain the previous value and previous best-response value like in SD-Gibbs.
- Like in SD-Gibbs, each agent in PD-Gibbs also keeps track of its best-response value. However, unlike in SD-Gibbs, where the best-response value is computed under the assumption that the other ancestors all take on their respective best-response values, in PD-Gibbs, the best-response value is computed under the assumption that all the other agents take on their respective regularly sampled values. Therefore, the best solution found will *always* be a best-response solution, unlike in SD-Gibbs, where the best solution can be a non-best-response solution.

Algorithm 3: Parallel Distributed Gibbs()

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

Procedure Initialize()

3 $p_{max} \leftarrow 0$
4 $X_i \leftarrow \{\langle x_j, \textbf{ValInit}(x_j)\rangle \mid x_j \in N_i\}$
5 $t_i \leftarrow t_i^* \leftarrow 0$
6 $d_i^{t_i} \leftarrow \bar{d}_i^{t_i} \leftarrow d_i^* \leftarrow \textbf{ValInit}(x_i)$
7 $\Delta_i^{t_i} \leftarrow \bar{\Delta}_i^{t_i} \leftarrow 0$
8 **if** x_i is root **then**
9 $\quad \delta^{t_i} \leftarrow \delta^* \leftarrow 0$
10 $\quad p_i \leftarrow 0$
11 $\quad$ Send PRIORITY $(x_i, p_i, d_i^{t_i})$ to each $x_j \in N_i$
12 **end**

- Finally, when a best solution is found by the root, it propagates BEST messages, which contains the iteration where the best solution is found. Agents receiving BEST messages can thus update their best value to their best-response value of the iteration in those messages.

All the other variables – X_i, t_i^*, and δ^* – are identical to those defined in SD-Gibbs.

Like in SD-Gibbs, the algorithm starts by constructing the pseudo-tree (line 1) and having each agent initialize its variables to their default values (lines 2-9). The root that then initializes its own priority value p_i to 0 (indicating that it has the highest priority among its neighbors to sample) and sends this value as well as its current value in a PRIORITY message to each of its neighbors (lines 10-11).

When an agent receives a PRIORITY message, it updates its context with the value of the neighbor contained in the message (line 13). If the agent has received a PRIORITY message from each of its parent and pseudo-parents (indicating that each of its neighboring ancestors have already chosen their priority values), then it too chooses its priority value to be the smallest value not yet taken by any of its neighboring ancestors (line 15) and sends a PRIORITY message to each of its neighbors (line 16). This process continues propagating down the pseudo-tree until it reaches all the leafs, at which point all the agents have chosen their priority values. This mechanisms ensures that no two neighboring agents have the same priority value, with preference for agents higher up in the pseudo-tree compared to agents lower down in the pseudo-tree.

When a leaf agent receives a PRIORITY message from all its parent and pseudo-parents, it also updates its p_{max} value to its p_i value (lines 3 and 17) and sends this value in a PMAXUP message to its parent (lines 18-20). When an agent receives a PMAXUP message, it updates its p_{max} value to the maximum of its current p_{max} value and the p_{max} value in the message (line 22). If the agent has received a PMAXUP message from each of its children, it too sends a PMAXUP message containing its updated p_{max} value to its

Procedure When Received PRIORITY(x_s, p_s, d_s)

```
13 Update ⟨x_s, d'_s⟩ ∈ X_i with ⟨x_s, d_s⟩
14 if Received PRIORITY messages from parent and all pseudo-parents then
15 |  p_i ← smallest non-negative priority value not taken by any x_j ∈ P_i ∪ PP_i
16 |  Send PRIORITY (x_i, p_i, d_i^{t_i}) to each x_j ∈ N_i
17 |  p_max ← p_i
18 |  if x_i is a leaf then
19 |  |  Send PMAXUP (p_max) to P_i
20 |  end
21 end
```

Procedure When Received PMAXUP(p_s)

```
22 p_max ← max{p_s, p_max}
23 if Received PMAXUP messages from all children then
24 |  if x_i is root then
25 |  |  Send PMAXDOWN (p_max) to each x_j ∈ C_i
26 |  |  SAMPLE()
27 |  else
28 |  |  Send PMAXUP (p_max) to P_i
29 |  end
30 end
```

parent (line 27-29). This process continues propagating up the pseudo-tree until it reaches the root, at which point the root's $p_{max} = \max_{x_j \in \mathcal{X}} p_j$ value equals is the largest priority value over all agents in the problem. It thus needs to inform all the other agents of this value, and it does so by sending its p_{max} value in a PMAXDOWN message to each of its children (line 25). When an agent receives a PMAXDOWN message, it updates its p_{max} value to the value in the message (line 31) and sends its updated value in a PMAXDOWN message to each of its children (line 32). This process continues propagating down the pseudo-tree until it reaches all the leafs and all agents have the correct p_{max} value. The agents use this p_{max} value to synchronize their sampling process so that an agent samples only if all its neighbors are not sampling in that iteration. We will describe later how the agents do so.

After an agent sends a PMAXDOWN message to each of its children, it starts the sampling process (lines 26 and 33). It increments its current iteration (line 34) and checks if it is its turn to sample (line 35). The condition $((t_i - 1) \mod p_{max}) = p_i$ ensures that in the first iteration ($t_i = 1$), only the agents whose priority values p_i equal 0 can sample. In the second iteration ($t_i = 2$), only the agents whose priority values equal 1 can sample. This process continues until all the agents whose priority values equal p_{max} samples, at which point all the agents have sampled once. The whole process repeats again starting with the agents whose priority values equal 0 sample. Since no two neighboring agents have

Procedure When Received PMAXDOWN(p_s)

31 $p_{max} \leftarrow \max\{p_s, p_{max}\}$
32 Send PMAXDOWN (p_{max}) to each $x_j \in C_i$
33 SAMPLE()

Procedure Sample()

34 $t_i \leftarrow t_i + 1$
35 **if** $((t_i - 1) \mod p_{max}) = p_i$ **then**
36 $\quad d_i^{t_i} \leftarrow$ Sample based on Equation 21
37 $\quad \bar{d}_i^{t_i} \leftarrow \text{argmax}_{d_i'} \sum_{\langle x_j, d_j \rangle \in X_i} F_{ij}(d_i', d_j)$
38 **else**
39 $\quad d_i^{t_i} \leftarrow d_i^{t_i - 1}$
40 $\quad \bar{d}_i^{t_i} \leftarrow d_i^{t_i - 1}$
41 **end**
42 $\Delta_i^{t_i} \leftarrow \sum_{\langle x_j, d_j \rangle \in X_i} [F_{ij}(d_i^{t_i}, d_j) - F_{ij}(d_i^{t_i - 1}, d_j)]$
43 $\bar{\Delta}_i^{t_i} \leftarrow \sum_{\langle x_j, d_j \rangle \in X_i} [F_{ij}(\bar{d}_i^{t_i}, d_j) - F_{ij}(d_i^{t_i - 1}, d_j)]$
44 Send VALUE $(x_i, d_i^{t_i})$ to each $x_j \in N_i$

the same priority value, this process ensures that no two neighboring agents are sampling in the same iteration.

If it is not an agent's turn to sample, then it takes its value in the previous iteration as its value and its best-response value in the current iteration (lines 38-41). If it is the agent's turn to sample, identical to SD-Gibbs, it samples according to Equation 21 (line 35-36). It then chooses its best-response value under the assumption that all its neighbors take on their respective value in its current context X_i (line 37). Note that this assumption is different than that in SD-Gibbs, which assumes that all the ancestors take on their respective best-response values.

In both cases, the agent also updates its delta values $\Delta_i^{t_i}$ and $\bar{\Delta}_i^{t_i}$ (lines 42-43). The update of $\Delta_i^{t_i}$ is identical to that in SD-Gibbs but the update of $\bar{\Delta}_i^{t_i}$ is different than that in SD-Gibbs. Here, $\bar{\Delta}_i^{t_i}$ is calculated under the assumption that the neighbors take on their regularly sampled values in current context X_i. In contrast, this delta value is calculated under the assumption that the neighbors take on their best-response value in SD-Gibbs. Finally, similar to SD-Gibbs, the agent sends its current value in a VALUE message to each of its neighbors (line 44).

When an agent receives a VALUE message, it updates the value of the sender in its current context X_i (line 45). If it has received a VALUE message from each of its neighbors in the current iteration, then it too samples (lines 46-47). If it is a leaf agent, then it sends its time index and its delta values in a BACKTRACK message to its parent (lines 48-50). When an agent receives a BACKTRACK message, it adds the received delta variables to its own delta variables (lines 52-53). If it has received a BACKTRACK message from all its children in a particular iteration t, then it sends the time index t and the delta variables

Procedure When Received VALUE(x_s, d_s)

45 Update $\langle x_s, d'_s \rangle \in X_i$ with $\langle x_s, d_s \rangle$
46 **if** Received VALUE messages from all neighbors in this iteration **then**
47 | Sample()
48 | **if** x_i is a leaf **then**
49 | | Send BACKTRACK $(x_i, t_i, \Delta_i^{t_i}, \bar{\Delta}_i^{t_i})$ to P_i
50 | **end**
51 **end**

Procedure When Received BACKTRACK($x_s, t, \Delta_s^t, \bar{\Delta}_s^t$)

52 $\Delta_i^t \leftarrow \Delta_i^t + \Delta_s^t$
53 $\bar{\Delta}_i^t \leftarrow \bar{\Delta}_i^t + \bar{\Delta}_s^t$
54 **if** Received BACKTRACK messages from all children in iteration t **then**
55 | **if** x_i is root **then**
56 | | $\delta^t \leftarrow \delta^t + \Delta_i^t$
57 | | **if** $\delta^{t-1} + \bar{\Delta}_i^t > \delta^*$ **then**
58 | | | $\delta^* \leftarrow \delta^{t-1} + \bar{\Delta}_i^t$
59 | | | $t_i^* \leftarrow t$
60 | | | $d_i^* \leftarrow \bar{d}_i^t$
61 | | | Send BEST(t) to each $x_j \in C_i$
62 | | **end**
63 | **else**
64 | | Send BACKTRACK $(x_i, t, \Delta_i^t, \bar{\Delta}_i^t)$ to P_i
65 | **end**
66 **end**

of that time index in a BACKTRACK message to its parent (line 62-64). This process continues propagating up the pseudo-tree until it reaches the root, at which point the root's delta variables indicate the differences in the global solution quality (see Lemmas 5 and 6). It thus updates its delta variable δ^t (line 56) and if it has found a better solution (line 57), then it updates the maximum difference δ^*, its best iteration index t_i^*, and its best value d_i^* (lines 58-60). Finally, it sends its best iteration index in a BEST message to each of its children (line 61).

When an agent receives a BEST message, it updates its best iteration index and its best value and sends a BEST message to each of its children (lines 67-69). This process continues propagating down the pseudo-tree until all leafs update their best iteration index and best values, which means that each agent knows its value in the best solution.

4.3.1 Theoretical Properties

Like Gibbs and SD-Gibbs, the PD-Gibbs algorithm also samples based on the same equation (Equation 21). The main difference is that, in SD-Gibbs, the agents along the same branch of the pseudo-tree sample sequentially in each iteration, while, in PD-Gibbs, multi-

Procedure When Received(BEST(t))

67 $t_i^* \leftarrow t$
68 $d_i^* \leftarrow \bar{d}_i^t$
69 Send BEST(t) to each $x_j \in C_i$

ple agents along the same branch of the pseudo-tree can sample in parallel in each iteration. The restriction is that no two neighboring PD-Gibbs agents sample in the same iteration. Therefore, this does not affect the correctness of the algorithm since non-neighboring (sampling) agents are conditionally independent of each other given the values of their neighboring (non-sampling) agents. Thus, the properties of Gibbs, which apply to SD-Gibbs, also apply to PD-Gibbs. We now describe the theoretical properties that apply specifically to PD-Gibbs.

Lemma 5. *In each iteration, after the root agent receives a BACKTRACK message from each of its children in iteration t and updates its delta variable Δ_i^t, that updated value is the difference in solution quality between the solution in iteration t and the solution in iteration $t-1$.*

Proof. The accumulated value of the root agent's delta variable includes the evaluation of $F_{ij}(d_i^t, d_j) - F_{ij}(d_i^{t-1}, d_j)$ by each agent x_i for all utility functions with its neighbors x_j, where d_j is the value of x_j in context X_i (line 42). Thus, that value is accumulated twice – once by each agent involved in the utility function. Let the two agents be agents x_p and x_c. Then, the contribution of the two agents combined is:

$$\begin{aligned} &F_{pc}(d_p^t, d_c \mid \langle x_c, d_c\rangle \in X_p) - F_{pc}(d_p^{t-1}, d_c \mid \langle x_c, d_c\rangle \in X_p) \\ &\quad + F_{pc}(d_p, d_c^t \mid \langle x_p, d_p\rangle \in X_c) - F_{pc}(d_p, d_c^{t-1} \mid \langle x_p, d_p\rangle \in X_c) \\ &= F_{pc}(d_p^t, d_c^{t-1}) - F_{pc}(d_p^{t-1}, d_c^{t-1}) + F_{pc}(d_p^{t-1}, d_c^t) - F_{pc}(d_p^{t-1}, d_c^{t-1}) \end{aligned}$$

There are the following three cases:

- **Case 1:** It is neither x_p nor x_c's turn to sample in iteration t (i.e., $((t-1) \mod p_{max}) \neq p_p$ and $((t-1) \mod p_{max}) \neq p_c$). Thus, $d_p^t = d_p^{t-1}$ and $d_c^t = d_c^{t-1}$ (line 39). Substituting these equalities in the equation above, we get:

$$\begin{aligned} &F_{pc}(d_p^t, d_c^{t-1}) - F_{pc}(d_p^{t-1}, d_c^{t-1}) + F_{pc}(d_p^{t-1}, d_c^t) - F_{pc}(d_p^{t-1}, d_c^{t-1}) \\ &\quad = F_{pc}(d_p^{t-1}, d_c^{t-1}) - F_{pc}(d_p^{t-1}, d_c^{t-1}) + F_{pc}(d_p^{t-1}, d_c^{t-1}) - F_{pc}(d_p^{t-1}, d_c^{t-1}) \\ &\quad = 0 \end{aligned}$$

  which is the correct difference in utility since both agents did not sample new values.
- **Case 2:** It is x_p's turn to sample in iteration t (i.e., $((t-1) \mod p_{max}) = p_p \neq p_c$). Thus, $d_c^t = d_c^{t-1}$ (line 39). Substituting this equality in the equation above, we get:

$$\begin{aligned} &F_{pc}(d_p^t, d_c^{t-1}) - F_{pc}(d_p^{t-1}, d_c^{t-1}) + F_{pc}(d_p^{t-1}, d_c^t) - F_{pc}(d_p^{t-1}, d_c^{t-1}) \\ &\quad = F_{pc}(d_p^t, d_c^t) - F_{pc}(d_p^{t-1}, d_c^{t-1}) + F_{pc}(d_p^{t-1}, d_c^{t-1}) - F_{pc}(d_p^{t-1}, d_c^{t-1}) \\ &\quad = F_{pc}(d_p^t, d_c^t) - F_{pc}(d_p^{t-1}, d_c^{t-1}) \end{aligned}$$

which is the difference in utility of that function between the current solution and the previous solution.

- **Case 3:** It is x_c's turn to sample in iteration t, which is symmetrical to case 2.

The sum of this difference over all utility functions is thus the difference in solution quality between the current solution and the previous solution. □

Lemma 6. *In each iteration, after the root agent receives a BACKTRACK message from each of its children in iteration t and updates its delta variable $\bar{\Delta}_i^t$, that updated value is the difference in solution quality between the best-response solution in iteration t and the solution in iteration $t-1$.*

Proof. The accumulated value of the root agent's delta variable includes the evaluation of $F_{ij}(\bar{d}_i^t, d_j) - F_{ij}(d_i^t, d_j)$ by each agent x_i for all utility functions with its neighbors x_j, where d_j is the value of x_j in context X_i (line 43). Thus, that value is accumulated twice – once by each agent involved in the utility function. Let the two agents be agents x_p and x_c. Then, the contribution of the two agents combined is:

$$\begin{aligned}
&F_{pc}(\bar{d}_p^t, d_c \mid \langle x_c, d_c\rangle \in X_p) - F_{pc}(d_p^{t-1}, d_c \mid \langle x_c, d_c\rangle \in X_p) \\
&\quad + F_{pc}(d_p, \bar{d}_c^t \mid \langle x_p, d_p\rangle \in X_c) - F_{pc}(d_p, d_c^{t-1} \mid \langle x_p, d_p\rangle \in X_c) \\
&= F_{pc}(\bar{d}_p^t, d_c^{t-1}) - F_{pc}(d_p^{t-1}, d_c^{t-1}) + F_{pc}(d_p^{t-1}, \bar{d}_c^t) - F_{pc}(d_p^{t-1}, d_c^{t-1})
\end{aligned}$$

There are the following three cases:

- **Case 1:** It is neither x_p nor x_c's turn to sample in iteration t (i.e., $((t-1) \mod p_{max}) \neq p_p$ and $((t-1) \mod p_{max}) \neq p_c$). Thus, $\bar{d}_p^t = d_p^{t-1}$ and $\bar{d}_c^t = d_c^{t-1}$ (line 40). Substituting these equalities in the equation above, we get:

$$\begin{aligned}
&F_{pc}(\bar{d}_p^t, d_c^{t-1}) - F_{pc}(d_p^{t-1}, d_c^{t-1}) + F_{pc}(d_p^{t-1}, \bar{d}_c^t) - F_{pc}(d_p^{t-1}, d_c^{t-1}) \\
&\quad = F_{pc}(d_p^{t-1}, d_c^{t-1}) - F_{pc}(d_p^{t-1}, d_c^{t-1}) + F_{pc}(d_p^{t-1}, d_c^{t-1}) - F_{pc}(d_p^{t-1}, d_c^{t-1}) \\
&\quad = 0
\end{aligned}$$

  which is the correct difference in utility since both agents did not sample new values.

- **Case 2:** It is x_p's turn to sample in iteration t (i.e., $((t-1) \mod p_{max}) = p_p \neq p_c$). Thus, $\bar{d}_c^t = d_c^{t-1}$ (line 40). Substituting this equality in the equation above, we get:

$$\begin{aligned}
&F_{pc}(\bar{d}_p^t, d_c^{t-1}) - F_{pc}(d_p^{t-1}, d_c^{t-1}) + F_{pc}(d_p^{t-1}, \bar{d}_c^t) - F_{pc}(d_p^{t-1}, d_c^{t-1}) \\
&\quad = F_{pc}(\bar{d}_p^t, d_c^t) - F_{pc}(d_p^{t-1}, d_c^{t-1}) + F_{pc}(d_p^{t-1}, d_c^{t-1}) - F_{pc}(d_p^{t-1}, d_c^{t-1}) \\
&\quad = F_{pc}(\bar{d}_p^t, d_c^t) - F_{pc}(d_p^{t-1}, d_c^{t-1})
\end{aligned}$$

  which is the difference in utility of that function between the current best-response solution and the previous solution.

- **Case 3:** It is x_c's turn to sample in iteration t, which is symmetrical to case 2.

The sum of this difference over all utility functions is thus the difference in solution quality between the current best-response solution and the previous solution. □

Lemma 7. *After the root agent receives a BACKTRACK message from each of its children in iteration t and updates its delta variable δ^t, that updated value is the difference in solution quality between the complete solution in iteration t and the initial complete solution, i.e., all variables x_j are assigned* $\mathbf{ValInit}(x_j)$.

Proof. We prove the above lemma by induction.

- **Iteration 1:** Δ_i^1 is the difference in the solution quality between the complete solution in iteration 1 and the initial complete solution of iteration 0 (Lemma 5). Therefore, since $\delta^1 = 0 + \Delta_i^1$ (lines 9 and 56), the lemma holds for iteration 1.
- **Induction Assumption:** Assume that the lemma holds for all iterations up to iteration $k-1$.
- **Iteration k:** Δ_i^k is the difference in the solution quality between the complete solution in iteration k and the complete solution in iteration $k-1$ (Lemma 5). Additionally, the δ^k value prior to the execution of line 56 is the difference in the solution quality between the complete solution in iteration $k-1$ and the initial complete solution of iteration 0 (induction assumption). Therefore, the δ^k value after the execution of line 56 is the difference in the solution quality between the complete solution in iteration k and the initial complete solution of iteration 0.

Thus, the proof concludes. □

Lemma 8. *After agent x_i receives a BACKTRACK message from each of its children in iteration t and updates its delta variables Δ_i^t and $\bar{\Delta}_i^t$, then the updated $\bar{\Delta}_i^t$ is no smaller than the updated Δ_i^t, i.e., the best-response solution is better than the regularly sampled solution.*

Proof. We prove the lemma by induction on the level of the agent in the pseudo-tree:

- **Leaf agent x_i:** Leaf agents do not receive BACKTRACK messages. Thus, their delta values are based only on the computations on lines 42-43. Therefore,

$$\begin{aligned}
\bar{\Delta}_i^t - \Delta_i^t &= \sum_{\langle x_j, d_j\rangle \in X_i} [F_{ij}(\bar{d}_i^t, d_j) - F_{ij}(d_i^{t-1}, d_j)] - \sum_{\langle x_j, d_j\rangle \in X_i} [F_{ij}(d_i^t, d_j) - F_{ij}(d_i^{t-1}, d_j)] \\
&= \sum_{\langle x_j, d_j\rangle \in X_i} F_{ij}(\bar{d}_i^t, d_j) - F_{ij}(d_i^{t-1}, d_j) - F_{ij}(d_i^t, d_j) + F_{ij}(d_i^{t-1}, d_j) \\
&= \sum_{\langle x_j, d_j\rangle \in X_i} F_{ij}(\bar{d}_i^t, d_j) - F_{ij}(d_i^t, d_j) \\
&\geq 0
\end{aligned}$$

- **Induction Assumption:** Assume that the lemma holds for all agents up to the children of the root agent.
- **Root agent x_i:** Prior to receiving any BACKTRACK messages and prior to executing lines 52-53, the condition $\bar{\Delta}_i^t \geq \Delta_i^t$ holds for the same reason as it held for the leaf agents. After receiving a BACKTRACK message from child $x_s \in C_i$ and after executing lines 52-53, the condition still hold since $\bar{\Delta}_s^t \geq \Delta_s^t$ by induction assumption.

Thus, the proof concludes. □

Theorem 6. *The best complete solution of PD-Gibbs is a best-response complete solution.*

Proof. The theorem trivially holds due to Lemma 8. □

Theorem 7. *Upon termination, PD-Gibbs returns the best solution found.*

Proof. If a better complete solution is found in iteration t, then the solution must be a best-response complete solution (Theorem 6). Additionally, the difference in solution quality between the best-response solution in iteration t and the initial complete solution of iteration 0 (i.e., $\delta_{t-1} + \bar{\Delta}_i^t$) must be greater than the largest difference thus far (i.e., δ^*). The root agent will then update its largest difference, its best time index, and its best value (lines 57-60) and sends these values in BEST messages to its children. When an agent receives a BEST message, it too updates its best value and best time index (lines 68-69) and sends these values in BEST messages to its children. This process propagates down the pseudo-tree until all agents have updated their best values and best time indices. □

Theorem 8. *Each PD-Gibbs iteration takes a finite amount of time.*

Proof. We first prove that the zero-th iteration of each agent takes a finite amount of time. This iteration starts at the start of the algorithm and ends immediately before the agent calls the Sample procedure. In this iteration, PRIORITY messages propagate down the pseudo-tree starting from the root agent to all leaf agents. This is followed by the propagation of PMAXUP messages up the pseudo-tree, which is followed by the propagation of PMAXDOWN messages down the pseudo-tree. Since messages are received in the order that they were sent and are never lost, the propagation of these messages take a finite amount of time. The root agent calls the Sample procedure immediately after receiving a PMAXUP message from each of its children and every other agent calls the Sample procedure immediately after receiving a PMAXDOWN message. Thus, the zero-th iteration takes a finite amount of time.

We now prove that the subsequent iterations of each agent takes a finite amount of time. These iterations start when the agent calls the Sample procedure and ends immediately before the agent calls the procedure again. Therefore, we will now prove that each agent will eventually call the Sample procedure after a finite amount of time unless it terminates first. This is equivalent to proving that an agent receives a VALUE message from each of its neighbors in each iteration in finite amount of time since each agent calls the Sample procedure only after that condition holds.

Each agent sends a VALUE message to each of its neighbors each time it calls the Sample procedure. Since all agents will eventually call the procedure for the first time (see above proof for the zero-th iteration) and all messages are received in the order that they were sent and are never lost, all agents will eventually receive a VALUE message from their neighbors in a finite amount of time. □

Lemma 9. *For each agent $x_i \in \mathcal{X}$, $p_i \leq p_{max}$ after all priority value updates.*

Proof. For each agent x_i, p_{max} is initialized to 0 (line 3) and updated for the first time to $\max\{p_i, 0\}$ (line 17). Each subsequent update of p_{max} can only make it larger (lines 22 and 31). Furthermore, p_i is not updated after it is initialized (lines 10 or 15). Therefore, $p_i \leq p_{max}$. □

Lemma 10. *For each agent $x_i \in \mathcal{X}$, $p_{max} = \max_{x_j \in \mathcal{X}} p_j$ after all priority value updates.*

Proof. All agents initialize their p_{max} priority values to their p_i priority values (line 17). Leaf agents send their p_{max} priority values up to their parent in PMAXUP messages. Upon receiving these messages, the parent updates their p_{max} priority values to the maximum between their current p_{max} values and the received p_{max} values. After receiving these messages from all children, each agent sends its p_{max} priority values up to its parent. Therefore, for each agent x_i, after it has updated its p_{max} priority value upon receiving a PMAXUP message from each of its children, its p_{max} value is the largest priority value p_j over all agents x_j in the subtree rooted at x_i. Therefore, the p_{max} value of the root agent equals is the largest priority value over all agents in the problem.

The root agent then sends this value down to its children in PMAXDOWN messages. Upon receiving this message, the child updates its p_{max} value to the received p_{max} value (line 31) and sends this updated value down to its children. This process continues until it reaches the leaf agents, at which point all agents have updated their p_{max} values to the root's value, which is the largest priority value over all agents in the problem. □

Lemma 11. *For all agents in the problem, $p_{max} < |\mathcal{X}|$, that is, it is upper bounded by the number of agents in the problem.*

Proof. We prove the lemma by induction on the height h of the pseudo-tree:

- **Base Case** [$h = 2$]**:** At this height, there are 2 agents in the problem. Let x_p be the parent (also the root) and x_c be the child. Agent x_p initializes its priorities p_{max} and p_p to 0 and sends its priority p_p to its child (lines 3, 10, and 11). Upon receiving this message, the child initializes its priority p_c and updates its priority p_{max} to 1 and sends its priority p_{max} to its parent (lines 15, 17, and 19). Upon receiving this message, the parent updates its p_{max} to 1 and sends it down to its child. Upon receiving this message, the child updates its priority p_{max} to 1 as well. Thus, p_{max} for both agents equal $1 < |\mathcal{X}| = 2$.
- **Induction Assumption:** Assume that the lemma holds for all heights up to height $k - 1$.
- **Induction Step** [$h = k$]**:** Let $\mathcal{X}_{k-1}$ be the set of agents in the pseudo-tree up to height $k - 1$. According to the induction assumption and Lemma 9, the priority value p_i of each agent $x_i \in \mathcal{X}_{k-1}$ is at most $|\mathcal{X}_{k-1}| - 1$. Furthermore, we know that the priority of all agents in the k-th level of the pseudo-tree must be (1) identical, since they are not constrained with each other, and (2) the smallest non-negative priority value not taken by any agents parent or pseudo-parent. Therefore, the priority value p_i of all agents x_i in the k-th level of the pseudo-tree must be at most $|\mathcal{X}_{k-1}| - 1 + 1 = |\mathcal{X}_{k-1}| < |\mathcal{X}|$. Consequently, the priority value p_{max} of all agents are also bounded by the same bound (Lemma 10).

Thus, the proof concludes. □

Theorem 9. *The memory complexity of each agent in PD-Gibbs is $O(|\mathcal{X}|)$, that is, it is linear in the number of agents in the problem.*

Proof. Each agent x_i needs to store context X_i, which contains agent-value pairs of all neighboring agents $x_j \in N_i$. Additionally, agent x_i needs to store a vector of values $d_i^{t_i}$, $\bar{d}_i^{t_i}$ and a vector of delta variables $\Delta_i^{t_i}$, $\bar{\Delta}_i^{t_i}$. If it is the root, it also needs to store a vector

of delta variables δ^{t-1}. The length of these vectors is $2h$, where h is the height of the pseudo-tree, because it can take up to h iterations for the best solution to propagate from a leaf agent to the root agent and up to h iterations for this solution to propagate down to the leaf agents after being confirmed by the root agent. Finally, agent x_i also needs to store the best value d_i^*; time indices t_i, t_i^*; and priority variables p_i, p_{max}. If it is the root, then it also needs to store delta variable δ^*. Each of these variables is a single value. Therefore, since $h = O(|\mathcal{X}|)$, its total memory complexity is $O(|\mathcal{X}|)$. □

Theorem 10. *The amount of information passed around in the network per iteration by PD-Gibbs agents is $O(|\mathcal{X}|^3)$, that is, it is polynomial in the number of agents in the problem.*

Proof. In the zero-th iteration, each agent sends exactly one PRIORITY message to each neighbor, exactly one PMAXUP message to its parent, and exactly one PMAXDOWN message to each child. Each message contains a constant number of values (each PRIORITY message contains 3 values, each PMAXUP message contains 1 value, and each PMAXDOWN message contains 1 value). In the subsequent iterations, each agent needs to send exactly one VALUE message to each neighbor, up to p_{max} BACKTRACK messages to its parent, and up to p_{max} BEST messages to each child in each iteration. Each message contains a constant number of values (each VALUE message contains 2 values, each BACKTRACK message contains 4 values, and each BEST message contains 1 value). Therefore, since $p_{max} = O(|\mathcal{X}|)$ (Lemma 11), the amount of information passed around in the network per iteration is $O(|\mathcal{X}|^3)$. □

5. Experimental Results

We now compare both versions of Distributed Gibbs – Sequential Distributed Gibbs (SD-Gibbs) and Parallel Distributed Gibbs (PD-Gibbs) – to DPOP (Petcu & Faltings, 2005a), MGM (Maheswaran et al., 2004a), DUCT (Ottens et al., 2012, 2017),[11] and "Rand", a variant of SD-Gibbs that samples uniformly. In terms of network load, that is, the amount of information passed around the network, DPOP sends an exponential amount of information in total while MGM, DUCT, Rand, SD-Gibbs, and PD-Gibbs send a polynomial amount of information in each iteration.

We use publicly-available implementations of MGM, DUCT, and DPOP, which are all implemented on the FRODO framework (Léauté, Ottens, & Szymanek, 2009). As DUCT was designed to solve a minimization problem, the code provided by the authors allows it to solve maximization problems by pre-processing the input files to flip the signs of the utilities (e.g., from positive to negative) and adding a positive constant to them so that the smallest utility is 0. We run our experiments on a quad-core Intel Xeon 2.40 GHz E5620 CPU with 2GB of memory per run.

We measure runtime using the simulated runtime metric (Sultanik, Lass, & Regli, 2007) and evaluate the algorithms on two types of problems: Problems without hard constraints and problems with hard constraints. For all problems, we set the DUCT parameters $\Delta = \epsilon = 0.05$, similar to the settings used in the original article (Ottens et al., 2012) unless

11. We used DUCT-D in our experiments.

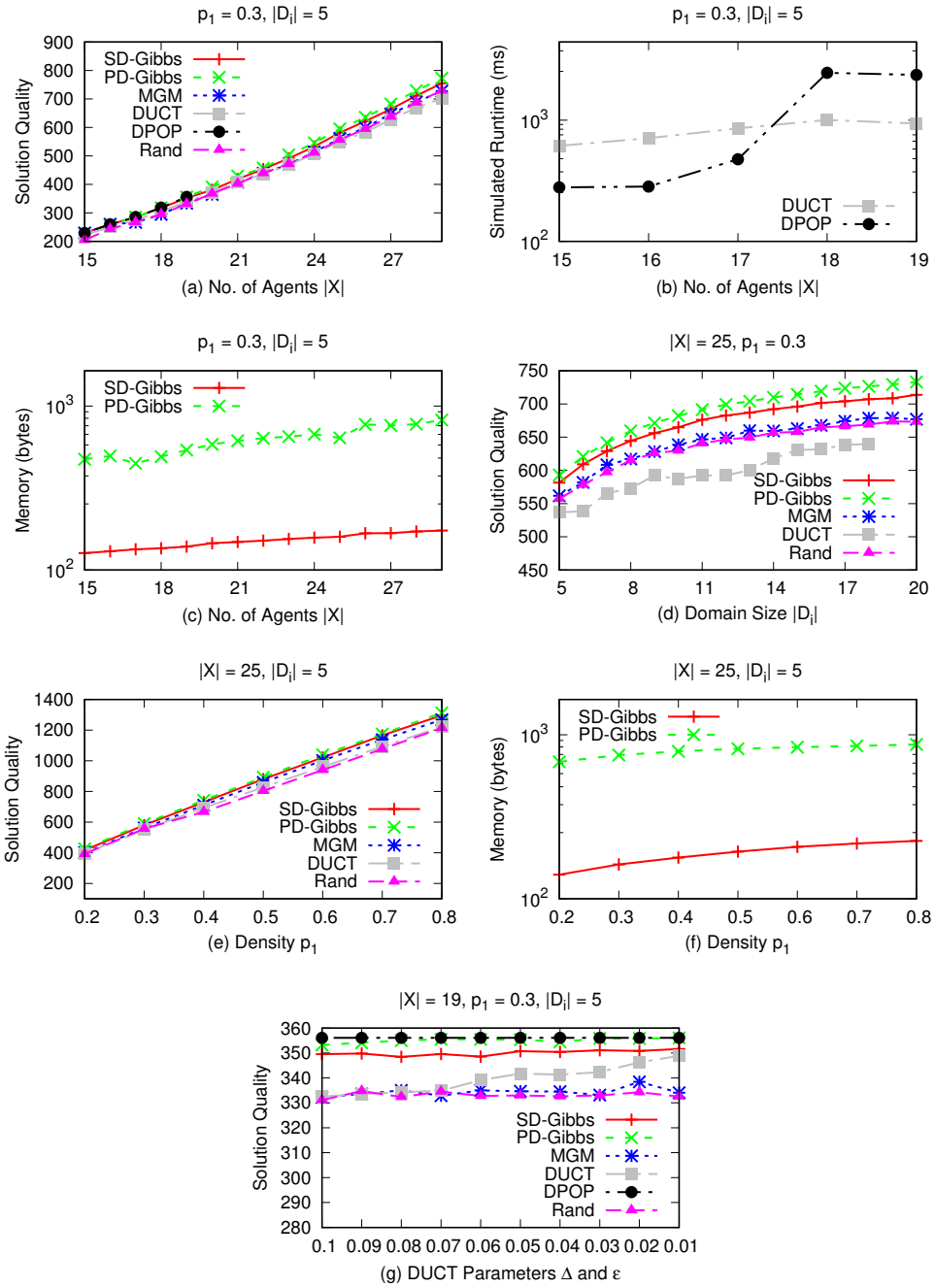


Figure 2: Results for Graph Coloring Problems without Hard Constraints

mentioned otherwise. We also let MGM and both versions of D-Gibbs run for as long as DUCT did for fair comparisons.[12] Each data point is averaged over 50 instances.

12. Exceptions are when DUCT failed to find a solution due to insufficient memory. For domain size $|D_i| = 19$ and 20 in Figure 2(b), we let the other algorithms run for as long as DUCT did for domain size $|D_i| = 18$.

5.1 Problems without Hard Constraints

For problems without hard constraints, that is, all solutions in the problem are feasible, we used graph coloring problems and sensor network problems as examples.

5.1.1 Graph Coloring Problems

We used the random graph coloring problem generator provided in the FRODO framework (Léauté et al., 2009) to generate our problems. We varied the size of the problem by increasing the number of agents $|\mathcal{X}|$ from 18 to 29, the graph density p_1[13] from 0.2 to 0.8 and the domain size $|D_i|$ of each agent x_i from 5 to 20, and we chose the constraint utilities uniformly from the range (0, 10) at random if the neighboring agents have different values and 0 if they have the same value. Figure 2 shows our results, where we varied the number of agents $|\mathcal{X}|$ in Figures 2(a), 2(b), and 2(c), the domain size $|D_i|$ in Figure 2(d), the density p_1 in Figures 2(e) and 2(f), and the DUCT parameters Δ and ϵ in Figure 2(g). DPOP ran out of memory for problems with 20 agents and above, and DUCT ran out of memory for problems with domain sizes 19 and 20.

Overall, DPOP found better solutions (when it did not run out of memory) than PD-Gibbs, which found better solutions than SD-Gibbs. Both versions of D-Gibbs generally found better solutions than MGM and DUCT. The difference is clearer in Figure 2(d). PD-Gibbs found better solutions than SD-Gibbs because it was able to explore more of the search space due to its parallel sampling operations. Interestingly, Rand was quite competitive – the solutions that it found have qualities that are very similar to those found by MGM.

Figure 2(b) shows the simulated runtimes of DUCT and DPOP. We omit results from the other incomplete algorithms as we let them run for as long as DUCT in all our experiments. DPOP is faster than DUCT when the problems are small, and vice versa when the problems are large. The reason is because DUCT requires a reasonably large number of samples to have the necessary confidence to terminate. Thus, when the problems are small, the necessary computation for all the samples is larger than solving the problem exactly with DPOP. As the problems become larger, the difference decreases.

As the amount of memory used by SD- and PD-Gibbs grows with the height of their pseudo-trees, we compare their memory consumption by varying the number of agents in Figure 2(c) and the density in Figure 2(f). We did not vary the domain size for this experiment because the domain size does not affect the height of the pseudo-tree. The results show that both versions of Gibbs require more memory as the number of agents or the density of the problem increase. The reason is because the height of the pseudo-tree increases with these two parameters. Additionally, in both cases, SD-Gibbs require at least five times less memory than PD-Gibbs, showing that while PD-Gibbs is able to find better solutions, it comes at a cost of increased memory consumption.

Finally, the Δ and ϵ values of DUCT correspond to its error tolerance. As those values decrease, its runtime will increase. Therefore, one can interpret Figure 2(g) as a graph that plots the quality of solutions found with increasing runtimes, since we let all algorithms except DPOP run for as long as DUCT. Not surprisingly, DUCT finds better solutions with increasing runtimes. However, interestingly, the quality of solutions found by SD-Gibbs, PD-

13. Defined as the ratio between the number of constraints and the maximum number of constraints.

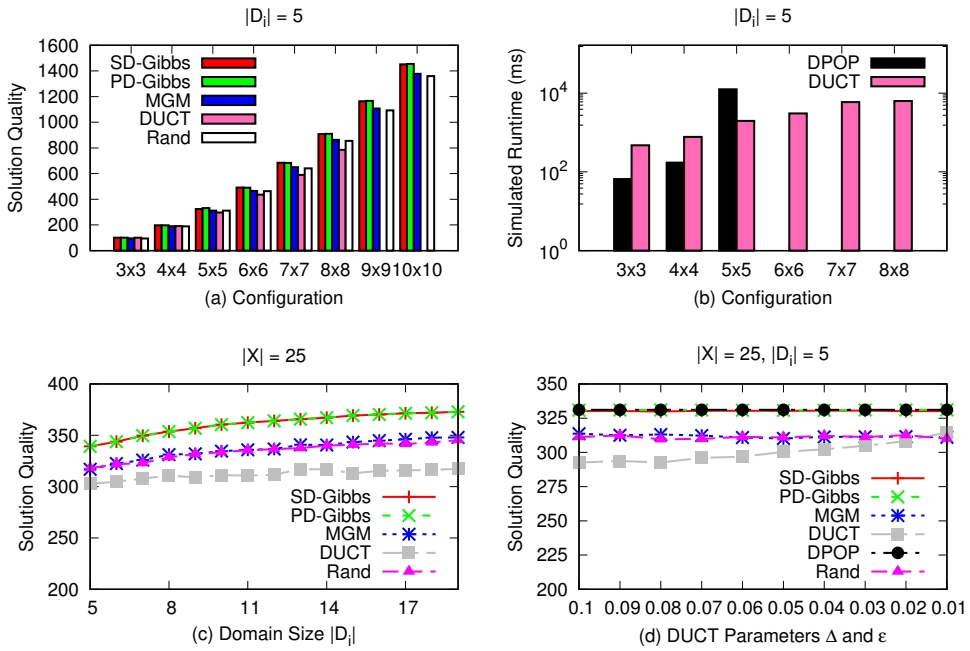


Figure 3: Results for Sensor Network Problems without Hard Constraints

Gibbs, MGM, and Rand remained relatively unchanged despite given more runtime, which means that they converged to their solutions very early on. On average, both SD- and PD-Gibbs found better solutions faster (when $\Delta = \epsilon = 0.1$) than DUCT (when $\Delta = \epsilon = 0.01$). However, it is important to note that DUCT provides quality guarantees on its solutions found while SD- and PD-Gibbs do not.

5.1.2 Sensor Network Problems

We use the same sensor network coordination problem as Nguyen et al. (2012). The sensors are arranged in a grid and each sensor can move along its 2D plane or stay stationary. Thus, the values of each sensor correspond to discretized directions of movements of that sensor. For example, if a sensor has 5 possible values, then it can move in the four cardinal directions or stay stationary. Additionally, sensors are constrained with all of their neighboring sensors. We varied the size of the problem by increasing the number of sensors $|\mathcal{X}|$ in the grid, where they are arranged in a square from 3×3 (i.e., $|\mathcal{X}| = 9$) to 10×10 (i.e., $|\mathcal{X}| = 100$). We also varied the domain size $|D_i|$ of each agent x_i from 5 to 19, and we chose the constraint utilities uniformly from the range [0, 10] at random. Figure 3 shows our results, where we varied the number of agents $|\mathcal{X}|$ in Figures 3(a) and 3(b), the domain size $|D_i|$ in Figure 3(c), and the DUCT parameters Δ and ϵ in Figure 3(d). DPOP ran out of memory for problems with configurations larger than 5×5. However, we omitted it from Figure 3(a) so that the figure is more easily readable. DUCT ran out of memory for problems with configurations larger than 8×8.

The trends in these graphs are consistent with the trends for the previous graph coloring problems except that the quality of solutions found by SD- and PD-Gibbs are very similar

to each other. The reason is because these problems are inherently simpler than the graph coloring problems due to the grid structure of the sensor network. The fact that both SD- and PD-Gibbs found solutions with qualities that are *very* close to optimal (see Figure 3(c)) asserts the simplicity of this class of problems. In these sensor network problems, each agent is constrained with exactly its four neighboring sensors. Therefore, there is a locality to these interactions and backedges only constrain two agents that are at most 4 hops away on the regular edges of the pseudo-tree (i.e., the longest backedge is between an agent and its great-great-grandparent on the pseudo-tree). In contrast, there is no such structure in graph coloring problems and backedges may constrain two agents that are $|\mathcal{X}|-1$ hops away on the regular edges of the pseudo-tree (as in the case if the constraint graph is a loop).

5.2 Problems with Hard Constraints

For problems with hard constraints, that is, some solutions in the problem are infeasible, we used graph coloring problems and radar coordination problems as examples. Additionally, as all the algorithms evaluated, except for DPOP, are incomplete algorithms that are not designed to handle hard constraints, we report two measures that are relevant in these problems – percentage of feasible solutions found and average number of violated constraints in solutions found. These two measures combined give readers a sense of how well an algorithm handles hard constraints in problems. We also omit DPOP from the results since it always finds feasible solutions with no constraints violated.

5.2.1 Hyper-Parameter Tuning

In the DCOP literature, the utilities of hard constraints are commonly set to $-\infty$ so that the utility of an infeasible solution is $-\infty$. If we do so, the probability for an agent of either version of D-Gibbs to take on an infeasible value is $e^{-\infty} = 0$ (see Equation 21). While this result may appear to be desirable at first glance, it, unfortunately, creates the following problem: The solution space of the problem will be segmented into "islands" of feasible solutions that have non-zero probabilities, and these islands are separated by infeasible solutions that have zero probabilities. Consequently, it is not possible for agents to probabilistically jump from one island to another, and the space of possible solutions explored is restricted to the initial island that the agents started on (i.e., the island that contains the initial randomly assigned solution).

Therefore, to overcome this limitation, we have to set the utilities of hard constraints a finite negative value $-U_{hard}$. Further, to better distinguish the soft constraints from the hard ones, we scale the utilities of soft constraints by multiplying them with a scaling factor C. As these two hyper-parameters will affect the quality of solutions found, we empirically evaluate both versions of D-Gibbs on a small set of problem instances[14] to find the best-performing hyper-parameters before using them on the larger sets of experiments. We tabulate the results for SD-Gibbs in Table 2. Results for PD-Gibbs are omitted as they were very similar. From the hyper-parameters that were swept, we chose to set $-U_{hard} = -1$ as the utility of hard constraints and $C = 10$ as the scaling factor for the utilities of soft constraints for our experiments below. The reason is that that combination of hyper-

14. We used 50 instances of random graphs with $|\mathcal{X}| = 19$, $|D_i| = 5$ for each agent x_i, and $p_1 = p_2 = 0.3$. Utilities for soft constraints were randomly sampled from the range $(0, 10)$.

(a) Fraction of Feasible Solutions Found

C	$-U_{hard}$									
	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10
1	0.10	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08
2	0.14	0.12	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08
3	0.18	0.14	0.08	0.08	0.06	0.08	0.08	0.08	0.08	0.08
4	0.30	0.20	0.12	0.08	0.08	0.08	0.08	0.08	0.08	0.08
5	0.24	0.28	0.20	0.20	0.08	0.08	0.08	0.08	0.08	0.08
6	0.50	0.34	0.30	0.14	0.20	0.08	0.08	0.08	0.08	0.08
7	0.60	0.54	0.36	0.32	0.26	0.16	0.08	0.08	0.08	0.08
8	0.60	0.54	0.42	0.40	0.24	0.26	0.18	0.08	0.08	0.08
9	0.62	0.60	0.48	0.50	0.38	0.28	0.24	0.12	0.08	0.08
10	0.68	0.58	0.64	0.46	0.52	0.34	0.36	0.14	0.14	0.08

(b) Average Number of Constraint Violations per Instance

C	$-U_{hard}$									
	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10
1	2.56	2.66	2.68	2.62	2.62	2.62	2.62	2.62	2.62	2.62
2	2.36	2.36	2.66	2.62	2.62	2.62	2.62	2.62	2.62	2.62
3	1.76	2.00	2.46	2.62	2.66	2.62	2.62	2.62	2.62	2.62
4	1.34	1.34	1.80	2.38	2.62	2.62	2.64	2.62	2.62	2.62
5	1.58	1.26	1.32	1.64	2.40	2.62	2.62	2.62	2.62	2.62
6	0.88	0.98	1.08	1.60	1.76	2.36	2.62	2.62	2.68	2.62
7	0.70	0.68	0.96	1.04	1.32	1.80	2.44	2.62	2.66	2.62
8	0.70	0.76	0.80	0.94	1.18	1.22	1.60	2.48	2.62	2.62
9	0.74	0.56	0.90	0.64	0.90	1.16	1.20	1.88	2.60	2.62
10	0.52	0.74	0.70	0.76	0.72	1.04	1.12	1.74	1.62	2.38

Table 2: Hyper-Parameter Tuning Results for SD-Gibbs

parameters resulted in the largest fraction of feasible solutions found as well as the smallest average number of constraint violations.

5.2.2 Graph Coloring Problems

Similar to Section 5.1.1, we used the same random graph coloring problem generator provided in the FRODO framework (Léauté et al., 2009) to generate our problems. However, the difference is that each variable now also has a unary constraint that prohibits some value assignments. The expected fraction of values prohibited is determined by the constraint tightness p_2. In our experiments, we vary this parameter from 0.1 to 0.9. Figure 4 shows our results, where we set the number of agents $|\mathcal{X}|$ to 19, the domain size $|D_i|$ of each agent x_i to 5, and the constraint density p_1 to 0.3. Utilities for soft constraints were randomly sampled from the range $(0, 10)$ and utilities for hard constraints were set to $-\infty$, except for D-Gibbs, where they were set to $-U_{hard} = -1$.

As expected, the fraction of feasible solutions found decreases and the average number of violated constraints increases with increasing constraint tightness. Not surprisingly, DPOP

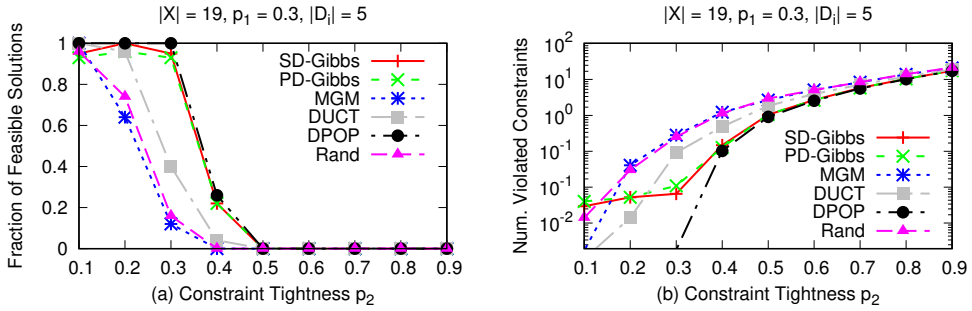


Figure 4: Results for Graph Coloring Problems with Hard Constraints

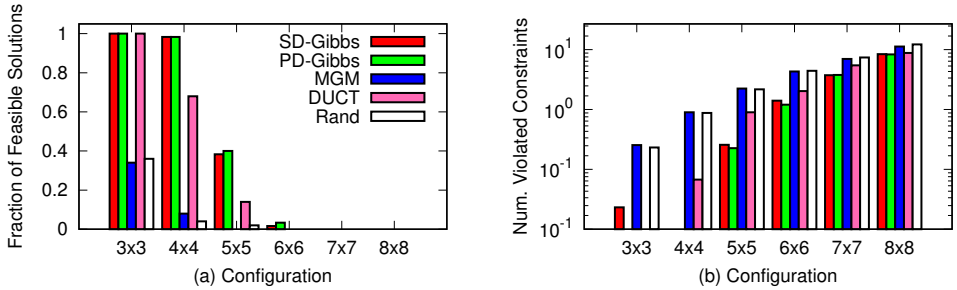


Figure 5: Results for Radar Coordination Problems with Hard Constraints

found the most number of feasible solutions. Both SD- and PD-Gibbs found slightly fewer feasible solutions, followed by DUCT, and then by both MGM and Rand.

In general, these dominance trends similarly translated when looking at the average number of violated constraints. The exception is on problems with small constraint tightness, where both SD- and PD-Gibbs performed worse than DUCT when $p_2 \leq 0.2$, and also worse than MGM when $p_2 = 0.1$. At $p_2 = 0.1$, they found solutions with similar number of violated constraints as Rand. Therefore, DUCT is preferred for solving problems with small constraint tightness, and either SD- or PD-Gibbs is preferred for solving problems with large constraint tightness.

5.2.3 Radar Coordination Problems

We use the same radar coordination problem as Fioretto, Yeoh, and Pontelli (2016). The problem models a set of radars that collect real-time data on the location and importance of atmospheric phenomena. Each phenomenon is characterized by size and weight (i.e., importance). Radars have limited sensing ranges, which determine their scanning regions. The goal is to find a radar configuration that maximizes the utility associated with the scanned phenomena. The radars are arranged in a grid like in the sensor network problems and they can scan their four cardinal directions. Phenomena are randomly generated across the grid until the underlying constraint graph is connected. Each phenomena must be scanned by at least p radars, where the value of p is randomly sampled from the range $[1, 4]$.

Figure 5 shows our results, where we varied the size of the problem by increasing the number of radars $|\mathcal{X}|$ in the grid. The radars are arranged in a square from 3×3 (i.e., $|\mathcal{X}| = 9$) to 8×8 (i.e., $|\mathcal{X}| = 64$). DPOP successfully found feasible solutions for all radar coordination problem instances. We thus omitted it from the graphs so that we can better compare the incomplete algorithms evaluated. In general, the trends in these graphs are consistent with the trends in the previous graph coloring problems, that is, both SD- and PD-Gibbs are better than DUCT, which is better than MGM and Rand.

6. Conclusions and Future Work

Researchers have not investigated sampling-based approaches to solve DCOPs until very recently, where Ottens *et al.* introduced the *Distributed UCT* (DUCT) algorithm, which uses confidence-based bounds. However, one of its limitation is its memory requirement per agent, which is *exponential* in the number of agents in the problem. This large requirement prohibits it from scaling up to large problems. Therefore, in this article, we introduce two new sampling-based DCOP algorithms called *Sequential Distributed Gibbs* (SD-Gibbs) and *Parallel D-Gibbs* (PD-Gibbs), whose memory requirements per agent is *linear* in the number of agents in the problem. It is a distributed extension of Gibbs, which was originally designed to approximate joint probability distributions in Markov random fields. We experimentally show that both SD- and PD-Gibbs found better solutions compared to an algorithm that samples uniformly, as well as competing local search algorithms like MGM in addition to DUCT. These two sampling-based algorithms thus improve the scalability of DCOP approaches to scale to larger problems of practical interest. In some cases, PD-Gibbs also found better solutions compared to SD-Gibbs because it is able to explore more of the search space due to its parallel sampling operations. However, this comes at a cost of larger memory requirement and message complexity compared to SD-Gibbs.

Future work includes a better study of the impact of different communication assumptions (e.g., introduction of delayed messages, message losses, and out of order messages) on the various DCOP algorithms presented in this paper. Initial studies by researchers have found that certain changes (e.g., message delays) can have significant impacts on some algorithms (Fernàndez, Béjar, Krishnamachari, & Gomes, 2002) but not others (Tabakhi, Tourani, Natividad, Yeoh, & Misra, 2017). It is uncertain if either version of Gibbs will always be preferable given network delays and unreliable communication. Thus, a more thorough investigation is necessary to better understand the dependencies of the various algorithms on the different communication assumptions.

We would like to also more thoroughly investigate the use of other MAP estimation algorithms to solve DCOPs. Message-passing algorithms like Expectation Maximization (EM) and Max-Product Linear Programming (MPLP) are ideal candidates for DCOPs given their use of messages to disseminate information between the different variables/agents (Wainwright, Jaakkola, & Willsky, 2002; Globerson & Jaakkola, 2007; Kumar & Zilberstein, 2010). Researchers have very recently adapted EM to solve a DCOP variant with resource constraints (Ghosh, Kumar, & Varakantham, 2015), and we suspect there are other synergistic opportunities in this intersection of DCOPs and MAP estimation problems.

Finally, we also plan to generalize our algorithms to solve dynamically changing problems. Such problems can be modeled as dynamic DCOPs, which are sequences of canonical

DCOPs with changes between subsequent problems (Petcu & Faltings, 2005b). Researchers have extended search- and inference-based DCOP algorithms to exploit information on how the problems may change over time when such information is available (Hoang, Fioretto, Hou, Yokoo, Yeoh, & Zivan, 2016; Hoang, Hou, Fioretto, Yeoh, Zivan, & Yokoo, 2017) as well as combined them with reinforcement learning and incremental heuristic search approaches when such information is unavailable (Nguyen, Yeoh, Lau, Zilberstein, & Zhang, 2014; Yeoh, Varakantham, Sun, & Koenig, 2015). It is likely that sampling-based DCOP algorithms may also be adapted to solve such problems.

Acknowledgment

Duc Thien Nguyen and Hoong Chuin Lau are partially supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office. William Yeoh is partially supported by the National Science Foundation under awards 1540168 and 1550662. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or the U.S. government.

References

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, *47*(2–3), 235–256.

Bacchus, F., Chen, X., van Beek, P., & Walsh, T. (2002). Binary vs. non-binary constraints. *Artificial Intelligence*, *140*(1–2), 1–37.

Besag, J. (1986). On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society, Series B*, *48*(3), 259–279.

Burke, D., & Brown, K. (2006). Efficiently handling complex local problems in distributed constraint optimisation. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pp. 701–702.

Dechter, R. (Ed.). (2003). *Constraint Processing*. Morgan Kaufmann.

Farinelli, A., Rogers, A., Petcu, A., & Jennings, N. (2008). Decentralised coordination of low-power embedded devices using the Max-Sum algorithm. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 639–646.

Fernàndez, C., Béjar, R., Krishnamachari, B., & Gomes, C. (2002). Communication and computation in distributed CSP algorithms. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)*, pp. 664–679.

Fioretto, F., Le, T., Yeoh, W., Pontelli, E., & Son, T. C. (2014). Improving DPOP with branch consistency for solving distributed constraint optimization problems. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)*, pp. 307–323.

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

Fioretto, F., Yeoh, W., & Pontelli, E. (2016). Multi-variable agent decomposition for DCOPs. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 2480–2486.

Fioretto, F., Yeoh, W., & Pontelli, E. (2017). A multiagent system approach to scheduling devices in smart homes. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 981–989.

Geman, S., & Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *6*(6), 721–741.

Gershman, A., Meisels, A., & Zivan, R. (2009). Asynchronous Forward-Bounding for distributed COPs. *Journal of Artificial Intelligence Research*, *34*, 61–88.

Ghosh, S., Kumar, A., & Varakantham, P. (2015). Probabilistic inference based message-passing for resource constrained DCOPs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 411–417.

Globerson, A., & Jaakkola, T. (2007). Fixing Max-Product: Convergent message passing algorithms for MAP LP-relaxations. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*, pp. 553–560.

Gutierrez, P., Meseguer, P., & Yeoh, W. (2011). Generalizing ADOPT and BnB-ADOPT. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 554–559.

Hamadi, Y., Bessière, C., & Quinqueton, J. (1998). Distributed intelligent backtracking. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pp. 219–223.

Hoang, K. D., Fioretto, F., Hou, P., Yokoo, M., Yeoh, W., & Zivan, R. (2016). Proactive dynamic distributed constraint optimization. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 597–605.

Hoang, K. D., Hou, P., Fioretto, F., Yeoh, W., Zivan, R., & Yokoo, M. (2017). Infinite-horizon proactive dynamic DCOPs. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 212–220.

Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *Proceedings of the European Conference on Machine Learning (ECML)*, pp. 282–293.

Kumar, A., Faltings, B., & Petcu, A. (2009). Distributed constraint optimization with structured resource constraints. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 923–930.

Kumar, A., Yeoh, W., & Zilberstein, S. (2011). On message-passing, MAP estimation in graphical models and DCOPs. In *Proceedings of the Distributed Constraint Reasoning Workshop*, pp. 57–70.

Kumar, A., & Zilberstein, S. (2010). MAP estimation for graphical models by likelihood maximization. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*, pp. 1180–1188.

Lass, R., Kopena, J., Sultanik, E., Nguyen, D., Dugan, C., Modi, P., & Regli, W. (2008). Coordination of first responders under communication and resource constraints (Short Paper). In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1409–1413.

Léauté, T., & Faltings, B. (2011). Coordinating logistics operations with privacy guarantees. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2482–2487.

Léauté, T., Ottens, B., & Szymanek, R. (2009). FRODO 2.0: An open-source framework for distributed constraint optimization. In *Proceedings of the Distributed Constraint Reasoning Workshop*, pp. 160–164.

Maheswaran, R., Pearce, J., & Tambe, M. (2004a). Distributed algorithms for DCOP: A graphical game-based approach. In *Proceedings of the International Conference on Parallel and Distributed Computing Systems (PDCS)*, pp. 432–439.

Maheswaran, R., Tambe, M., Bowring, E., Pearce, J., & Varakantham, P. (2004b). Taking DCOP to the real world: Efficient complete solutions for distributed event scheduling. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 310–317.

Mailler, R., & Lesser, V. (2004). Solving distributed constraint optimization problems using cooperative mediation. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 438–445.

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

Nguyen, D. T., Yeoh, W., & Lau, H. C. (2012). Stochastic dominance in stochastic DCOPs for risk-sensitive applications. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 257–264.

Nguyen, D. T., Yeoh, W., & Lau, H. C. (2013). Distributed Gibbs: A memory-bounded sampling-based DCOP algorithm. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 167–174.

Nguyen, D. T., Yeoh, W., Lau, H. C., Zilberstein, S., & Zhang, C. (2014). Decentralized multi-agent reinforcement learning in average-reward dynamic DCOPs. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1447–1455.

Ottens, B., Dimitrakakis, C., & Faltings, B. (2012). DUCT: An upper confidence bound approach to distributed constraint optimization problems. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 528–534.

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*, *8*(5), 69:1–69:27.

Petcu, A., & Faltings, B. (2005a). A scalable method for multiagent constraint optimization. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1413–1420.

Petcu, A., & Faltings, B. (2005b). Superstabilizing, fault-containing multiagent combinatorial optimization. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 449–454.

Rajasekaran, S. (2000). On simulated annealing and nested annealing. *Journal of Global Optimization*, *16*(1), 43–56.

Rust, P., Picard, G., & Ramparany, F. (2016). Using message-passing DCOP algorithms to solve energy-efficient smart environment configuration problems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 468–474.

Sontag, D., Globerson, A., & Jaakkola, T. (2010). *Introduction to Dual Decomposition for Inference*. MIT Press.

Sontag, D., Meltzer, T., Globerson, A., Jaakkola, T., & Weiss, Y. (2008). Tightening LP relaxations for MAP using message passing. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 503–510.

Sultanik, E., Lass, R., & Regli, W. (2007). DCOPolis: a framework for simulating and deploying distributed constraint reasoning algorithms. In *Proceedings of the Distributed Constraint Reasoning Workshop*.

Tabakhi, A. M., Tourani, R., Natividad, F., Yeoh, W., & Misra, S. (2017). Pseudo-tree construction heuristics for DCOPs and evaluations on the ns-2 network simulator. In *Proceedings of the International Conference on Tools with Artificial Intelligence (ICTAI)*.

Ueda, S., Iwasaki, A., & Yokoo, M. (2010). Coalition structure generation based on distributed constraint optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 197–203.

Vinyals, M., Rodríguez-Aguilar, J., & Cerquides, J. (2011). Constructing a unifying theory of dynamic programming DCOP algorithms via the generalized distributive law. *Autonomous Agents and Multi-Agent Systems*, *22*(3), 439–464.

Wainwright, M., Jaakkola, T., & Willsky, A. (2002). MAP estimation via agreement on (hyper)trees: Message-passing and linear programming approaches. *IEEE Transactions on Information Theory*, *51*, 3697–3717.

Yanover, C., Meltzer, T., & Weiss, Y. (2006). Linear programming relaxations and belief propagation – an empirical study. *Journal of Machine Learning Research*, *7*, 1887–1907.

Yeoh, W., Felner, A., & Koenig, S. (2010). BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. *Journal of Artificial Intelligence Research*, *38*, 85–133.

Yeoh, W., Varakantham, P., & Koenig, S. (2009). Caching schemes for DCOP search algorithms. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 609–616.

Yeoh, W., Varakantham, P., Sun, X., & Koenig, S. (2015). Incremental DCOP search algorithms for solving dynamic DCOPs. In *Proceedings of the International Conference on Intelligent Agent Technology (IAT)*, pp. 257–264.

Yeoh, W., & Yokoo, M. (2012). Distributed problem solving. *AI Magazine*, *33*(3), 53–65.

Yokoo, M. (Ed.). (2001). *Distributed Constraint Satisfaction: Foundation of Cooperation in Multi-agent Systems*. Springer.

Zivan, R., Okamoto, S., & Peled, H. (2014). Explorative anytime local search for distributed constraint optimization. *Artificial Intelligence*, *212*, 1–26.