Privacy for the Distributed Stochastic Algorithm with Breakouts

Julien Vion[1], **René Mandiau**[1], **Sylvain Piechowiak**[1], **Marius Silaghi**[2]
[1]LAMIH CNRS UMR 8201, Université Polytechnique Hauts de France, F-59313 Valenciennes, France
{first name.family name}@uphf.fr
[2]Florida Institute of Technology, Melbourne, FL, USA
marius.silaghi@fit.edu

Abstract

Privacy has traditionally been a major motivation of distributed problem solving. In this paper, we focus on privacy issues when solving Distributed Constraint Optimization Problems (DCOPs) using a local search approach. Two such popular algorithms exist to find good solutions to DCOP: DSA and GDBA. However, these were not designed with privacy in mind.

In this paper, we propose DSAB, a new algorithm that merges ideas from both algorithms to allow extensive handling of constraint privacy. We also study how algorithms behave when solving Utilitarian DCOPs, where utilitarian agents want to reach an agreement while reducing the privacy loss. We show experimentally that this allows us reductions of domain privacy loss by a factor 2 to 3 with no significant impact on the quality of the solution.

Introduction

The Distributed Constraint Optimization Problem (DCOP) is a general framework used to model and solve distributed NP-hard challenges. To solve a DCOP, agents negotiate to find a solution that satisfies as well as possible a given set of constraints. Privacy is an important issue in a lot of distributed applications. Therefore, besides the constraint costs entailed by the solution, the cost of privacy loss during the process should also be considered (Greenstadt, Pearce, and Tambe 2006; Yokoo et al. 1998; Hamadi, Bessiere, and Quinqueton 1998). Confidentiality is lost when data is exchanged between agents. For example, in distributed scheduling problems, participants may not want to reveal their unavailable time slots because the explanation concerns their private life, and it should not be publicized. We know that the assignment of time slots can be difficult if participants do not want to reveal their constraints (Freuder, Minca, and Wallace 2001; Crépin et al. 2009; Faltings, Léauté, and Petcu 2008). When an agent is concerned with its privacy, it practically associates a cost to the revelation of each piece of information in its local problem. This cost may be embedded into utility driven reasoning (Silaghi and Mitra 2004; Doshi et al. 2008; Savaux et al. 2017). Local search algorithms such as the Distributed Stochastic Algorithm (DSA) (Zhang, Wang, and Wittenburg 2002) and (Generalized) Distributed Breakout Algorithm (GDBA) (Yokoo and Hirayama 1996; Hirayama and Yokoo 2005; Okamoto, Zivan, and Nahon 2016) allow for taking into account utilities quite easily, i. e., without impact on their soundness.

Distributed local search algorithms are prone to *oscillations*, when two agents keep changing their values without ever reaching an agreement. As most traditional local search techniques, they are also at the risk to get stuck in local minima. DSA and GDBA implement strategies to prevent oscillations and/or local minima. Although incomplete, these algorithms allow for finding suboptimal solutions of good quality relatively quickly and have the *anytime* property (Wallace and Freuder 1996; Zivan, Okamoto, and Peled 2014).

In this paper, we propose a new algorithm we call Distributed Stochastic Algorithm with Breakouts (DSAB) that uses strategies inspired from DSA and GDBA to prevent oscillations, to get out of local optima, and to allow extensive handling of constraint privacy. We also study how the algorithms behave when solving Utilitarian DCOPs, where utilitarian agents want to reach an agreement while reducing the privacy loss. Previous works have shown the interest of utilities to improve privacy in complete algorithms (Savaux et al. 2019).

In the following sections, after reviewing background, we introduce DSAB, then show how domain privacy can be enhanced for both DSAB, DSA and GDBA with Utilitarian agents. Then, we observe the impact on both constraint and domain privacy costs on academic and realistic problems.

Background

In this paper, we make the following assumptions, without loss of generality, in order to simplify our descriptions:

- an agent encapsulates exactly one variable,
- domains are finite subsets of integers (any finite and countable set can actually be used),
- a constraint involves at most two variables.

The following definition embraces the Open Constraint Programming model for domains (Faltings and Macho-Gonzalez 2005). This allows to add values to domains dynamically during search. In our context, this means that domains are initially private, and values are revealed only when the search process requires it.

Definition 1. *A Distributed Constraint Optimization Problem (DCOP) is defined as a tuple* $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$*, where:*

$\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ *is a finite set of n agents.*

$\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ *is a set of n variables. Each agent A_i encapsulates the variable x_i.*

$\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$ *is a set of n domains, with $d = \max(|\mathcal{D}_i|)$.*[1] *Each variable x_i can be instantiated to a value $v \in \mathcal{D}_i$. An assignment of $X \subseteq \mathcal{X}$ is a set of instantiations for each variable of X.*

$\mathcal{C} = \{C_1, C_2, \ldots, C_e\}$ *is a finite set of e valued constraints. Each constraint C_i involves variables $\mathcal{X}(C_i) \subseteq \mathcal{X}$, and defines a non-negative cost for each assignment of its variables.*[2] *We denote with $C_i(X)$ the cost of the assignment of $\mathcal{X}(C_i) \subseteq X$ for C_i. We also denote by $\mathcal{C}(A_i)$ the set of constraints that involve x_i, i. e., $\mathcal{C}(A_i) = \{C_j \in \mathcal{C} \mid x_i \in \mathcal{X}(C_j)\}$.*

The objective is to find an optimal solution S, i. e., an assignment for $\mathcal{X}$ that minimizes the total cost of constraints $\sum_{i=1}^{e} C_i(S)$.

***DSA* (Zhang, Wang, and Wittenburg 2002) and *GDBA* (Okamoto, Zivan, and Nahon 2016)** are local search (incomplete) synchronized algorithms designed to find good quality solutions to DCOP quickly. As most incomplete algorithms, DSA and GDBA cannot handle hard constraints, i. e., infinite costs,[3] and cannot prove that a given solution is optimal. However, proving optimality is NP-hard and local search algorithms are much more scalable than exponential complete algorithms such as ABT, DPOP or Sync-BB (Hirayama and Yokoo 1997; Yokoo et al. 1998; Petcu and Faltings 2005).

A greedy approach to distributed local search is for each agent to assign the value that entails the lowest cost at each iteration. However, consider for example two agents A_1 and A_2 holding respectively the two variables x_1 and x_2, both with domain $\{1, 2\}$, associated by an inequality constraint ($x_1 \neq x_2$). If x_1 and x_2 are both instantiated to 1, it contradicts the inequality constraint. On the next step, both agents will change their value simultaneously to 2, and the constraint will still be violated. If no strategy is implemented to avoid such *oscillations*, the two agents will keep on changing values at every cycle and no solution will ever be found. DSA avoids oscillations by stochastically "skipping" iterations according to a given parameter p, which allows only the neighbor agent to change its assignment. GDBA adds an additional cycle: agents share the improvement they can get to their neighbors *via* an "improve" message. Only the agent that entails the best improvement will change its assignment.

Another common issue in local search is local optima, where no local change can improve the current solution even if one exists in the search space. GDBA use the "*breakout*" strategy (Morris 1993) to escape local minima: when improvements are sent to neighbors, if all improvements are non-positive, a *quasi* local minima is detected and a *breakout* is performed by incrementing the weight of all violated constraints. DSA does not implement any technique to avoid local minima.

Okamoto, Zivan, and Nahon (2016) (resp. Zhang, Wang, and Wittenburg (2002)) describe several variants of GDBA (resp. DSA). In the following, we will only consider the variant called (M, NM, T) (resp. B), which gave the best results. Adapting DSA for private domains is trivial. For GDBA, it requires to adapt the way that constraints are considered to be unsatisfied in the "NM" variant. This will be further described in the "Constraint Privacy" Section.

Privacy is the property that agents benefit from conserving the secrecy of their personal information. This is a broad concept that several authors tried to categorize (Grinshpoun 2012; Greenstadt, Grosz, and Smith 2007; Léauté and Faltings 2013). According to Grinshpoun (2012), agents' privacy may concern domains, constraints, assignments, and algorithms. From these three papers, it is the only one that mentions domain privacy, although the author states that it cannot be achieved by the traditional DCOP model. Indeed, Grinshpoun takes for granted that constraints costs are represented using extensive tables that covers the full cartesian product of involved variable domains. This entails that each agent is aware of its neighbours' domains even before resolution starts. Here, we consider a more general model, where costs for each constraint C_i are defined by some function $f_i : \mathbb{Z}^{|\mathcal{X}(C_i)|} \mapsto \mathbb{N}$. Note that the function is defined on supersets of variable domains (more details in the "Domain Privacy with DSAB" Section).

Advanced stochastic algorithms rely on *mediator agents* that handle *groups* of agents, and use complete algorithms to prove optimal partial solutions for such groups. Assignment privacy in this context requires sophisticated, cryptographically secure protocols (Léauté and Faltings 2013; Grinshpoun and Tassa 2016; Tassa, Zivan, and Grinshpoun 2016; 2017). However, these approaches do not consider domain privacy. In the following, we consider that initial domains and intra-agent constraints are private, whereas assignments are sent and shared inter-agent constraints are known to neighbors. Every time an agent sends a message, it contains information (i. e., constraint costs or domain values) that entails an irremediable loss of privacy. We want to enhance privacy by minimizing the amount of leaked private information.

We do not consider algorithm privacy in this paper, and assume that all agents follow the same algorithm. In the following sections, we will consider constraint costs and domain privacy.

Distributed Stochastic Algorithm with Breakouts

Our motivating example is an Agent A that wants to schedule a meeting with several peers. It is available at every hour from 8am to 8pm, with variable preferences, and does

[1] d is only used to infer worst-case complexities. If values are added dynamically during search, d considers the "final" domain sizes.

[2] We chose to avoid negative constraint costs because they are not intuitive, but they would not affect the algorithms described in this paper.

[3] In practice, we use some large constant K to represent "infinite" costs.

not want the nature of these preferences to be known. The DCOP framework allows modeling these preferences as constraint costs, and using a distributed environment allows for keeping these constraints private. However, to avoid oscillations and escape local minima, GDBA requires agents to send "improve" messages, containing the delta between the cost of its current assignment and the best cost it can achieve by changing the value of its variable. This has two drawbacks: firstly, although the delta is the result of the sum of several costs, it can give clues about the constraint costs of the agent. It is well known that even such statistical information can be used to derive initial data (Adam and Worthmann 1989). Secondly, it brings questions about whether and how privacy costs should be incorporated in the computation of deltas (privacy costs will be detailed in the next section).

With the Distributed Stochastic Algorithm with Breakouts (DSAB), we propose to use the stochastic strategy of DSA to avoid oscillations, and implement local minima detection and escape using breakouts. For the stochastic part, DSAB allows agents to randomly "yield" every now and then, so that alternative assignments can be eventually tried to reach the best agreement. This "yield" technique is controlled by a parameter $0 \leq p \leq 1$, as for DSA. p defines the probability to change the value, which means that $p = 1$ corresponds to the "greedy" approach, and $p = 0$ means that agents keep their initial assignment forever. p is not an excessively crucial parameter and experiments show that $p = 0.95$ is good for most applications.

If an agent cannot improve its current solution, it sends a "stalled" message to detect quasi local minima and perform breakouts. In a nutshell, an agent A_i running DSAB-p performs one action at each step:

- if there exists a value v' in $\mathcal{D}_i$ that improves the current cost function, A_i may *change* its instantiation to $x_i = v'$ with a probability p, or
- if a better value exists, with a probability $1 - p$, A_i *yields* its chance to switch its value. Either way, it sends an "ok?" message to its acquaintances containing the final value.
- If no better value exists, A_i *stalls* and sends a "stalled" message to its acquaintances. If all its neighbors have stalled on the previous step, a quasi local minimum is detected and constraints $\mathcal{C}(A_i)$ are weighted. "Stalled" messages are required to distinguish yields from stalls.

Our algorithm is further described on Algorithm 1: v represents the agent's current instantiation. $agentView$ is an assignment containing the known instantiations of the neighbors. v is initialized to the best possible value (Line 2) and sent to neighbors (Line 3). As $agentView$ is empty at this step, the cost evaluation can only consider unary constraints and domain privacy costs (cf. next section).

Lines 6–8 process incoming messages. DSAB-p agents send and receive only one message per neighbor at each step. Line 9 selects the best value v' according to the COST function, taking into account the $agentView$ and the set of revealed values $\mathcal{R}_i$ (cf. next section). Line 10 checks whether v' improves the current solution. In this case, v' is assigned with a probability p (Line 11). The chosen value is added to $\mathcal{R}_i$ (Line 12) and sent to the neighbors (Line 13).

Algorithm 1: DSAB-p running on agent A_i

```
1  Initialize constraint weights to 1
2  $v \leftarrow \arg\min_{j \in D_i} \text{COST}(j, \emptyset, \emptyset)$
3  Send $v$ to all neighbors
4  while termination condition not met do
5      $stallCount \leftarrow 0$
6      Receive messages from all neighbors:
7          when "ok?": update $agentView$ and $minCosts$
8          when "stalled": increment $stallCount$
9      $v' \leftarrow \arg\min_{j \in D_i} \text{COST}(j, agentView, \mathcal{R}_i)$
10     if $\text{COST}(v', agentView, \mathcal{R}_i) < \text{COST}(v, agentView, \mathcal{R}_i)$ then
11         With a probability $p$: $v \leftarrow v'$
12         $\mathcal{R}_i \leftarrow \mathcal{R}_i \cup \{v\}$
13         Send "ok?($v$)" to all neighbors
14     else
15         if $stallCount = |\mathcal{N}_i|$ then // all neighbors have stalled
16             foreach $C_j \in \mathcal{C}(A_i) \mid C_j(agentView) > minCosts(C_j)$ do
17                 Increase $C_j$ weight
18             Send "stalled" to all neighbors
19 Assign best $v$ to $x_i$
```

If v' does not improve the current solution, Line 15 controls whether a quasi local minima has been encountered ($|\mathcal{N}_i|$ is the number of neighbors). In this case, Lines 16–17 weight the non-minimally (NM) violated constraints. Following Okamoto, Zivan, and Nahon (2016), a constraint is NM-violated if the cost corresponding to the current solution is higher than the best cost that can be obtained with this constraint. However, the "best cost" of the constraint is not necessarily known when the algorithm starts. Indeed, the best cost for constraint C_i is defined as the smallest cost entailed by the cartesian product of the domains of $\mathcal{X}(C_i)$. If $\mathcal{X}(C_i)$ contains variables from neighbor agents, their domain is unknown. To settle this, we maintain $minCosts$, which maps for each constraint C_i the lowest known cost that the constraint may entail. When a neighbor reveals a new value on Line 7, $minCosts$ is partly recomputed. The same change must be implemented in GDBA in order to allow dynamic or private domains.

Only assignments containing the new value need to be evaluated. In the worst case, over the full search process, maintaining $minCosts$ requires to evaluate every possible assignment exactly once, i. e., for each constraint C_i, at most $d^{|\mathcal{X}(C_i)|}$ constraint checks are performed. In order to avoid the exponential factor if non-binary intentional or global constraints are to be considered, some polynomial function or approximation to compute $minCosts$ will probably have to be added to the constraint definition.

The value of $minCosts(C_i)$ can only decrease throughout the search. This means that C_i might be considered satisfied early in the search but NM-violated later by the same assignment. The space complexity of $minCosts$ is $\Theta(|\mathcal{C}(A_i)|)$ for each agent. It is also required to keep track of values revealed by acquaintances, which is a $\Theta(|\mathcal{N}_i| \cdot d)$ data structure. DSAB-p has the same time and space complexities of GDBA. The algorithm runs until some termination condition is met (Line 4), usually by setting an upper bound on the number of cycles.

Domain Privacy with Utilities

We take again our motivating example from the previous section. Although agent A is available at every hour from 8am to 8pm, it would like to avoid confessing that he has so much availabilities, so as to avoid being highly solicited in the future. This can be achieved by keeping its initial domain private, and by revealing as few values as possible during search.

The matters presented in this section can be applied to either DSAB-p, GDBA, and DSA, assuming they embrace the Open Constraint Programming principle for domains (Faltings and Macho-Gonzalez 2005). Although DCOP models traditionally assume that constraints are represented using extensive tables, which implies that agents know their neighbors' initial domains, our more general model from Definition 1 allows to define cost functions on *supersets* of variable domains. We do not make assumptions on how the functions are implemented: it might be a table, but also in intention, e. g., a simple arithmetic expression such as $f(x,y) \mapsto (x-y)^2$ for a quadratic soft equality constraint, or by a global property such as the number of different values that appear in the assignment.[4] Further on, a default cost (e. g., 0 or some large constant K) can be provided if an unexpected value is encountered, and even more sophisticated compressed representations can be used (Mairy, Deville, and Lecoutre 2015).

When the agent reveals its instantiation, it reveals that the concerned value belongs to its domain. The Utilitarian DCOP model allows to set a cost to the leak of this information (Doshi et al. 2008; Savaux et al. 2017; 2019). If an agent wishes to protect its privacy, we can consider that the computation is now performed by utility-based agents that are partly self-interested: they make decisions aiming at minimizing a utility function, which involves both minimizing the constraint cost, and the privacy loss. This is formalized as follows:

Definition 2. *A Utilitarian Distributed Constraint Optimization Problem is defined as a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{P} \rangle$, i. e., a DCOP with an additional matrix $\mathcal{P}$ of domain privacy costs. $\mathcal{P}_{i,j}$ is the non-negative cost[5] for A_i to reveal whether $j \in \mathcal{D}_i$.*

[4]This property corresponds to the soft all-equal constraint (Hebrard et al. 2011).

[5]Assigning negative privacy costs implies that the agent has an *interest* in leaking information. Although this is not very intuitive, it does not really prevent the algorithms described in this paper to behave normally. Those values are simply assigned as soon as they do not entail excessive constraint costs.

Algorithm 2: COST(v, $agentView$, $\mathcal{R}_i$)

```
1 $constraintCost \leftarrow \sum_{C_j \in \mathcal{C}(A_i)} weight(C_j) \times C_j(agentView \cup \{x_i = v\})$
2 $privacyCost \leftarrow \left(\max_{C_j \in \mathcal{C}(A_i)} weight(C_j)\right) \times \sum_{j \in R_i \cup \{v\}} \mathcal{P}_{i,j}$
3 return $constraintCost + privacyCost$
```

The objective is to find an optimal solution while minimizing the privacy loss $\sum_{i=1}^{n} \sum_{j \in \mathcal{R}_i} \mathcal{P}_{i,j}$, where $\mathcal{R}_i \subseteq \mathcal{D}_i$ is the set of values that A_i revealed during the solving process.

Note that privacy loss is a function that depends on the search process itself and is not represented in traditional DCOPs. The two objectives are somewhat contradicting: the privacy loss can only increase during the search process as new messages are sent, whereas the constraint costs are likely to decrease as new, hopefully better solutions are found. What we want is to control the search process to find good compromises. In our implementation, the utility function of each agent is based on the *sum* of constraint costs and privacy losses.

An issue we have with this strategy in GDBA and DSAB-p is that when constraints are heavily weighted, then the privacy costs may become negligible. We found out that weighting the privacy costs as much as the most weighted constraint, i. e., give as much importance to privacy as to the most weighted constraint, brings good results.[6] For DSA the issue does not exist as constraints are not weighted ($\forall j, weight(C_j) = 1$). We summarize our COST function on Algorithm 2. The function returns the total cost (both constraint and privacy) that would be entailed if the value v was instantiated during the current step, given current $agentView$ and $\mathcal{R}_i$. The $privacyCost$ variable implements *domain privacy control* (DPC). Note that $privacyCost$ does not need to be recomputed from scratch at every call to COST function. We simply maintain the maximum constraint weight (constant-time operation every time a constraint is weighted), as well as the sum of the privacy costs of already revealed values (constant-time every time a new value is revealed), to compute $privacyCost$ in constant time. This allows DPC to have no impact on the worst-case complexity of algorithms.

This strategy also works when agents encapsulate several variables. Although sending values between internal variables does not actually entail privacy loss, there is actually two cases for a given variable x_i: 1. x_i is purely internal, and we can define $\forall j, \mathcal{P}_{i,j} = 0$ so that privacy is ignored, or 2. x_i is an *interface* variable, i.e., is involved by at least one inter-agent constraint. In this case, data is leaked every time its value is changed, so penalty costs must be considered anyway.

[6]Preliminary experiments where we weighted privacy costs as much as the *average* of constraint weights were inconclusive.

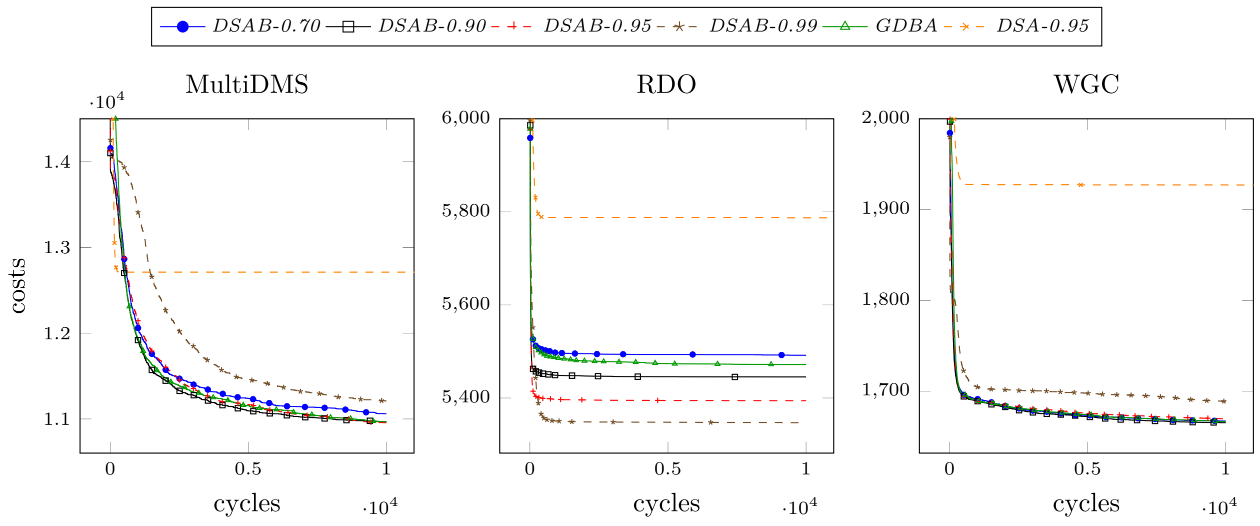

Figure 1: *Anytime* total constraint costs over cycles for GDBA and DSAB-p

Experiments

We implemented DSAB-p, GDBA, and DSA with and without domain privacy control (DPC) over the Akka 2.5 platform (Bonér and the Akka Team at Lightbend 2009). Akka is a modern, performant implementation of the Actor Model (Hewitt, Bishop, and Steiger 1973) for the Java Virtual Machine, suitable for managing DCOP agents in our experimental context. We used the B variant of DSA described in (Zhang, Wang, and Wittenburg 2002), that gave best results on graph coloring problems.

We test the algorithms on 150 instances of the following benchmarks and compute the average of *anytime* costs at each cycle. Unless noted, we used a log-normal probability distribution with given mean and standard deviation (std. dev.), rounded to the nearest integer, to generate non-negative values.

MultiDMS is a distributed meeting scheduling problem: 40 meetings must be scheduled over $d = 30$ time slots. A pool of 100 people are involved in these meetings. For each meeting, each individual may be selected to participate in the meeting with a probability of 5 %. Each event has a duration with mean $=$ 2.5 and std. dev. $=$ 1.12. Moreover, participants must have time to travel from a meeting to another. For each pair of meetings, a generated distance of mean $=$ 1.5 and std. dev. $=$ 0.5 must be respected. For each pair of attendees of a meeting, a penalty of $K = 100$ is given if the two participants are not scheduled at the same time. Another penalty of $K = 100$ is given for each attendee if two meetings overlap or do not respect the distance constraint.

A unary cost constraint is generated for each variable, giving each domain value a cost with mean $=$ 20 and std. dev. $=$ 6.06. A privacy domain cost is generated for each value, with mean $=$ 10 and std. dev. $=$ 3.

RDO is a randomly generated problem with $n = 200$ variables with a domain of $d = 30$ values. A random graph with variables as vertices and binary constraints as edges is generated with a density of 2 % ($e = 398$). For each constraint, a constraint cost is enforced on each pair of possible instantiations with mean $=$ 50 and std. dev. $=$ 50.

A privacy domain cost is generated for each value, with mean $=$ 10 and std. dev. $=$ 10.

WGC is a variant of weighted random coloring problem over $n = 200$ variables with $d = 30$ colors. A random graph with variables as vertices and binary constraints as edges is generated with a density of 20 % ($e = 3.980$). Binary constraints are "hard" ($K = 1{,}000$) inequality constraints. A unary cost constraint is generated for each variable, with mean $=$ 50 and std. dev. $=$ 50.

A privacy domain cost is generated for each value, with mean $=$ 10 and std. dev. $=$ 10.

We chose the parameters for MultiDMS to roughly match our target problem of generating a weekly time table for our organization. The number of attendees of a meeting follows a binomial distribution with a mean of $100 \times 5\,\% = 5$. Each (meeting, attendee) pair requires one variable. It results in 200 variables on average for each instance.

For this problem, our implementation can run 10,000 cycles of GDBA in about 20 seconds wallclock time on a Java OpenJDK 11 64-Bit Server VM running on a 4-core Intel i7-5600U CPU @ 2.6 GHz with 4 GiB allowed heap space. For the two other problem classes, which are more academic, we tuned the parameters to obtain the same number of variables and domain size, and similar running times.

A first experiment compares DSA-0.95, GDBA and DSAB-p for various p. Domain privacy control is disabled by removing Line 2 of Algorithm 2, only constraint costs are observed. Results are illustrated in Figure 1. These

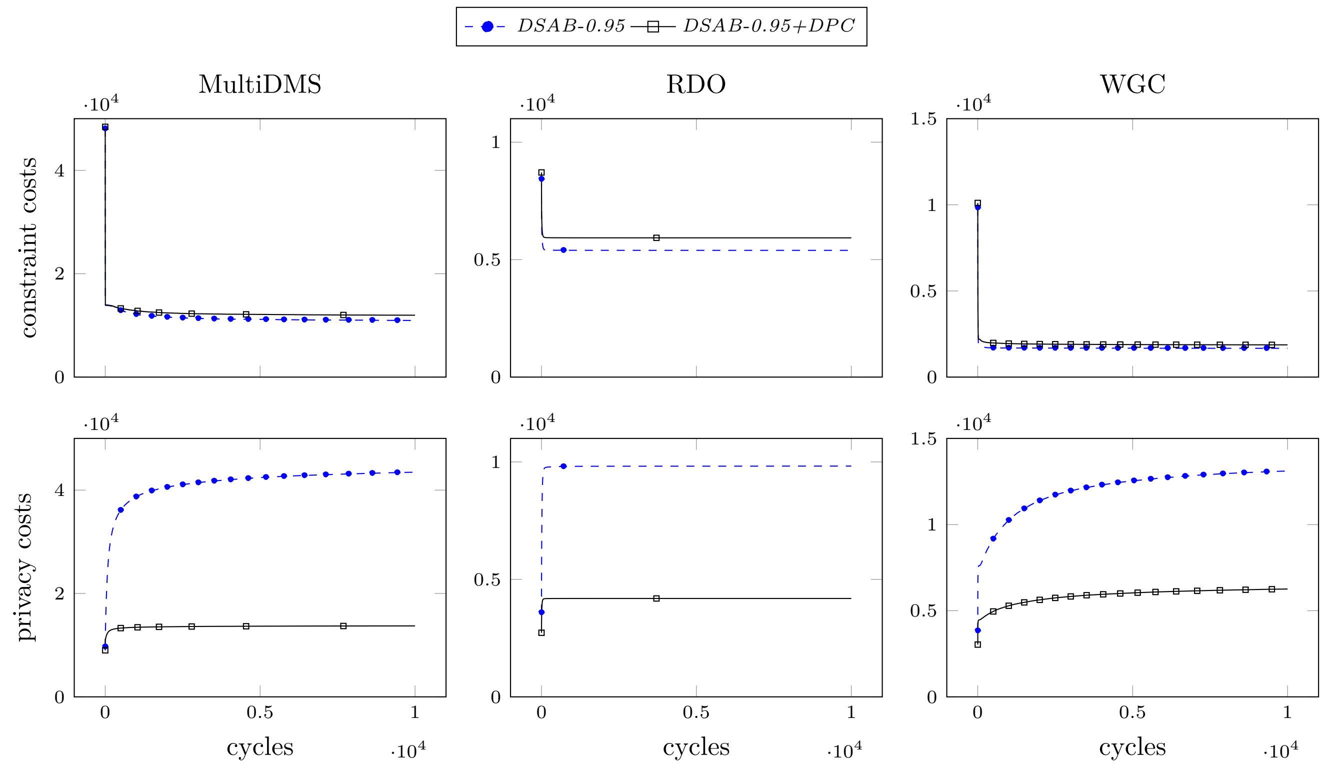

Figure 2: Impact of DPC on *anytime* constraint and privacy costs for DSAB-0.95

plots show that DSAB-0.90 is always better than GDBA on these benchmarks. However, DSAB-0.95 is better than DSAB-0.90 by a relatively large margin on RDO problems, and the difference is nearly imperceptible on the others. DSAB-0.99 peforms surprisingly well on RDO problems. Probably oscillations are not really an issue on such homogeneous problems. For DSA, we ran a preliminary experiment to devise the best p parameter, and it turned it was 0.95 as well. DSA does not have any mechanism to escape local minima and quickly jams on a bad solution. Note that this results in very few leaked domain values.

A second experiment compares constraint and privacy costs for DSAB-p, with and without domain privacy control We chose to *include* the domain privacy costs when computing and transmitting deltas for the "improve" messages of GDBA. This means that (unmeasured) information on privacy costs is leaked, but it reduces the domain privacy costs themselves. Results are displayed in Figure 2. We can observe that DPC allows to reduce privacy costs by a factor of 2 to 3 on average, while having a low impact on the constraint costs. We obtained similar results with GDBA (omitted due to space constraints). We also note that DSAB-0.95 tends to increase the number of values revealed w.r.t. GDBA.

Conclusion and Outlook

In this paper, we proposed a new algorithm, called Distributed Stochastic Algorithm with Breakouts (DSAB-p), which uses a stochastic technique to avoid oscillations and breakouts to escape local minima. Compared to the legacy GDBA algorithm, DSAB-p does not send any information about constraint costs. As a second contribution, we embraced the Open Constraint Programming model for domains (Faltings and Macho-Gonzalez 2005), implemented and experimented domain privacy control for DSA, GDBA and DSAB-p, exploiting the Utilitarian DCOP model proposed by (Doshi et al. 2008; Savaux et al. 2017). Our results show that implementing privacy control has a low impact on the quality of solutions w.r.t. constraint costs, but allows for reducing domain privacy loss by a factor of two to three.

Future work may consider agents encapsulating more than one variable and non-binary constraints. A better evaluation of the quality of the obtained solutions w.r.t the optimal would allow to improve local search algorithms, while keeping privacy in mind. Heuristics more sophisticated than the sum of constraint and privacy costs should be considered, e. g., weighted sums or products, dynamic weights, games, etc. Also, it would be interesting to introduce a malevolent agent that exploits the solving protocol in order to gather more information than required to solve the problem. A more formal study of what can be derived from data sent by the various algorithms should be performed. Event though DSAB-p hides constraint costs, the order in which values are sent is meaningful. This may be countered by preserving the k-anonymity of the constraint relations (Sweeney 2002).

In our experiments, agents know the constraints that they share with their neighbors and their associated cost function. This is not a hard requirement in the algorithms we studied. Models similar to Partially Know Constraints (PKC) (Brito et al. 2009; Grinshpoun et al. 2013), where the actual cost function of a constraint is held by only one agent, could easily be designed. The behavior of the algorithms on such models should be investigated. For example, if a constraint is one-sided, one agent will not have to reveal its assignment to its neighbour. This will improve domain privacy and reintroduce a form of assignment privacy.

References

Adam, N. R., and Worthmann, J. C. 1989. Security-control methods for statistical databases: A comparative study. *ACM Comput. Surv.* 21(4):515–556.

Bonér, J., and the Akka Team at Lightbend. 2009. Akka: Build powerful reactive, concurrent, and distributed applications more easily. akka.io.

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

Crépin, L.; Demazeau, Y.; Boissier, O.; and Jacquenet, F. 2009. Privacy preservation in a decentralized calendar system. In *Proc. 7th Itl Conf on Practical Applications of Agents and Multi-Agent Systems (PAAMS)*, volume 55 of *Advances in Intelligent and Soft Computing*, 529–537.

Doshi, P.; Matsui, T.; Silaghi, M.; Yokoo, M.; and Zanker, M. 2008. Distributed private constraint optimization. In *Proc. IEEE/WIC/ACM Itl Conf on Intelligent Agent Technology*, 277–281. IEEE Computer Society.

Faltings, B., and Macho-Gonzalez, S. 2005. Open constraint programming. *Artificial Intelligence* 161(1):181 – 208. Distributed Constraint Satisfaction.

Faltings, B.; Léauté, T.; and Petcu, A. 2008. Privacy guarantees through distributed constraint satisfaction. In *Proc. IEEE/WIC/ACM Itl Conf on Intelligent Agent Technology*, volume 2, 350–358. IEEE Computer Society.

Freuder, E. C.; Minca, M.; and Wallace, R. J. 2001. Privacy/efficiency tradeoffs in distributed meeting scheduling by constraint-based agents. In *IJCAI Workshop on Distributed Constraint Reasoning*, 63–72.

Greenstadt, R.; Grosz, B.; and Smith, M. D. 2007. SS-DPOP: improving the privacy of DCOP with secret sharing. In *Proc. of the 6th Itl joint conference on Autonomous agents and multiagent systems*, 171. IFAAMAS.

Greenstadt, R.; Pearce, J. P.; and Tambe, M. 2006. Analysis of privacy loss in distributed constraint optimization. In *Proc 21st Nat. Conf. on Artificial Intelligence and the 18th Innovative Applications of Artificial Intelligence Conf.*, 647–653. AAAI Press.

Grinshpoun, T., and Tassa, T. 2016. P-syncbb: A privacy preserving branch and bound DCOP algorithm. *J. Artif. Intell. Res. (JAIR)* 57:621–660.

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

Grinshpoun, T. 2012. When you say (DCOP) privacy, what do you mean? - categorization of DCOP privacy and insights on internal constraint privacy. In Filipe, J., and Fred, A. L. N., eds., *Proc. of the 4th Itl Conference on Agents and Artificial Intelligence (ICAART)*, 380–386. SciTePress.

Hamadi, Y.; Bessiere, C.; and Quinqueton, J. 1998. Backtracking in distributed Constraint Networks. In *Proc. of 13th European Conf. on Artificial Intelligence (ECAI)*, 219–223.

Hebrard, E.; Marx, D.; O'Sullivan, B.; and Razgon, I. 2011. Soft constraints of difference and equality. *J. Artif. Int. Res.* 41(2):97–130.

Hewitt, C.; Bishop, P.; and Steiger, R. 1973. A universal modular ACTOR formalism for artificial intelligence. In *Proc. 3rd IJCAI*, 235–245.

Hirayama, K., and Yokoo, M. 1997. Distributed partial constraint satisfaction problem. In Smolka, G., ed., *Principles and Practice of Constraint Programming - CP97, Third International Conference, Linz, Austria, October 29 - November 1, 1997, Proceedings*, volume 1330 of *Lecture Notes in Computer Science*, 222–236. Springer.

Hirayama, K., and Yokoo, M. 2005. The distributed breakout algorithms. *Artificial Intelligence* 161(1-2):89–115.

Léauté, T., and Faltings, B. 2013. Protecting privacy through distributed computation in multi-agent decision making. *Journal Artificial Intelligence Research (JAIR)* 47:649–695.

Mairy, J.-B.; Deville, Y.; and Lecoutre, C. 2015. The smart table constraint. In Michel, L., ed., *Proc. 12th CPAIOR*, 271–287. Springer International Publishing.

Morris, P. 1993. The breakout method for escaping from local minima. In *Proceedings of AAAI'93*, 40–45.

Okamoto, S.; Zivan, R.; and Nahon, A. 2016. Distributed breakout: Beyond satisfaction. In *Proc. 25th IJCAI*, 447–453.

Petcu, A., and Faltings, B. 2005. A scalable method for multiagent constraint optimization. In Kaelbling, L. P., and Saffiotti, A., eds., *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, 266–271. Professional Book Center.

Savaux, J.; Vion, J.; Piechowiak, S.; Mandiau, R.; Matsui, T.; Hirayama, K.; Yokoo, M.; Elmane, S.; and Silaghi, M. 2017. Utilitarian approach to privacy in distributed constraint optimization problems. In *Proc. 30th FLAIRS*, 454–459.

Savaux, J.; Vion, J.; Piechowiak, S.; Mandiau, R.; Matsui, T.; Hirayama, K.; Yokoo, M.; Elmane, S.; and Silaghi, M. 2019. Privacy stochastic games in distributed constraint reasoning. *Annals of Mathematics and Artificial Intelligence* (to appear).

Silaghi, M., and Mitra, D. 2004. Distributed constraint satisfaction and optimization with privacy enforcement. In *Intelligent Agent Technology (IAT)*, 531–535. IEEE.

Sweeney, L. 2002. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10(05):557–570.

Tassa, T.; Zivan, R.; and Grinshpoun, T. 2016. Preserving privacy in region optimal DCOP algorithms. In *Proc. 25th IJCAI*, 496–502.

Tassa, T.; Zivan, R.; and Grinshpoun, T. 2017. Privacy preserving implementation of the Max-Sum algorithm and its variances. *JAIR* 59:311–349.

Wallace, R. J., and Freuder, E. C. 1996. Anytime algorithms for constraint satisfaction and SAT problems. *ACM SIGART Bulletin* 7(2):7–10.

Yokoo, M., and Hirayama, K. 1996. Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *Proc. of the 2nd Itl Conf on Multi-Agent Systems*, 401–408. AAAI Press.

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

Zhang, W.; Wang, G.; and Wittenburg, L. 2002. Distributed stochastic search for constraint satisfaction and optimization: Parallelism, phase transitions and performance. In *Proceedings of AAAI Workshop on Probabilistic Approaches in Search*.

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