Privacy Guarantees through Distributed Constraint Satisfaction

Boi Faltings, Thomas Léauté, and Adrian Petcu

Artificial Intelligence Laboratory
Ecole Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland
{boi.faltings,thomas.leaute,adrian.petcu}@epfl.ch
http://liawww.epfl.ch/

LIA-REPORT-2008-012

Abstract. In Distributed Constraint Satisfaction Problems, agents often desire to find a solution while revealing as little as possible about their variables and constraints. So far, most algorithms for DisCSP do not guarantee privacy of this information. This paper describes some simple obfuscation techniques that can be used with DisCSP algorithms such as *DPOP*, and provide sensible privacy guarantees based on the distributed solving process without sacrificing its efficiency.

Key words: Distributed Constraint Satisfaction/Optimization, privacy

1 Introduction

Many practical situations require the coordination of actions of different agents. For example, consider allocation of airport takeoff and landing slots. Currently, each airport allocates slots individually [1]. However, airlines need combinations of slots to operate sequences of flights. Similar situations exist when sharing pipelines, electricity grids, and other infrastructure among competing agents.

A consideration that often prohibits such coordinated decisions is the desire to keep coordination constraints private. For example, in airport slot allocation, airlines do not want competitors to find out what routes they intend to fly, as this can be used to counter their strategy. As security breaches are frequent in modern information systems, they would not entrust a central platform with this information either.

One possibility to nevertheless achieve coordination is to use a distributed algorithm where agents are themselves responsible for applying the coordination constraints. Several authors have proposed distributed algorithms for constraint satisfaction and optimization, such as *ABT* [2], *AWC* [2], *AAS* [3], *ADOPT* [4], *OPTAPO* [5] and *DPOP* [6]. However, none of these algorithms gives privacy guarantees, and any particular information could be leaked in the right circumstances.

There is significant earlier work on how to measure the privacy loss of these algorithms. Franzin et al. [7] measure privacy loss as the reduction in entropy of other agents' preferences. Maheswaran et al. [8] develop a framework called Valuations of Possible States that measures privacy loss as the degree to which the possible states of other agents are reduced. Greenstadt [9] uses this framework to analyze privacy loss of DPOP and ADOPT.

While it is important to *measure* privacy loss, in practice it is important to be able to give guarantees that certain information is not revealed by an algorithm. Brito [10, 11] has developed various methods for reducing privacy loss in search and in particular to protect privacy of a constraint from agents involved in it. Greenstadt [12] proposes an algorithm that uses cryptography to eliminate a major source of privacy loss in DPOP. Some algorithms have been proposed that maintain total privacy, but these rely on secure multiparty computation that is too complex to be used in practical applications [13–15].

In this paper, we show how an existing algorithm for distributed constraint satisfaction, *DPOP*, can be adapted to give the required privacy guarantees with no increase in complexity and thus solve problems of realistic size.

Throughout the paper, we will use the simple example of three airlines A, B and C, interested in two slots y and z at two different airports. Airline A is interested in getting either one of the two slots (OR constraint), airline B also wants either one, but not both (XOR constraint), and airline C wants either both slots or none (equality constraint).

In this example, a solution is for instance to give slot y to airline A, slot z to B, and no slot to C. Note that some information is revealed by the fact that each agent finds out the values its variables take in the final solution: for example C will know that some other agents asked for and obtained either slot y or z, since it did not get its request satisfied. No algorithm can avoid this information loss, so agents that cannot accept it need to avoid it through appropriate modeling or other means.

In a problem with multiple solutions, participants must accept to leak the information associated with *any* of the solutions, since they cannot control which solution will be reached. We call the information that can be inferred from potential solutions the *semi-private* information. Importantly, the techniques in this paper do not protect semi-private information, since agents must accept that this might be leaked by the final solution.

We distinguish four types of privacy guarantees:

– *Agent privacy*: no agent can learn the identity of any other agent unless they share a coordination constraint. For example, airline A cannot discover the identity of airline B.

  Definition 1. *An algorithm for DisCSP preserves* agent privacy *when no agent a_i learns the identity of the agent controlling any variable x_j that does not share a constraint with another variable x_i controlled by agent a_i.*

– *Topology privacy*: no agent can learn anything about topological constructs (constraints, cycles) that do not involve a variable that it has a constraint with. For example, airline A cannot find out that another airline has a constraint over slots y and z.

Definition 2. *An algorithm for DisCSP preserves* topology privacy *if no agent learns about the existence of either constraints or cycles of constraints that do not involve at least one variable it controls.*

- *Constraint privacy*: no agent can learn the nature or contents of constraints in which it is not involved. For example, A cannot find out that B wants y XOR z. In an optimization setting, A cannot discover what value B attaches to obtaining y or z, in case it has any preference for one slot or the other.

  Definition 3. *An algorithm for DisCSP preserves* full constraint privacy *if an agent does not learn the cost of any particular tuple in a constraint that does not involve any variable it controls, except for semi-private information. It preserves* limited constraint privacy *if the information that it can learn about such a tuple is bounded by a threshold ϵ and ϵ can be made arbitrarily small by suitable choice of protocol parameters.*

- *Decision privacy*: no agent can discover the outcome of any decision that other agents make in the final solution.

  Definition 4. *An algorithm for DisCSP preserves* full decision privacy *if no agent can learn the values of any variable that it does not control in the final solution, except for semi-private information.*
  An algorithm for DisCSP preserves limited decision privacy *if no agent can learn the values of any variable that it does not control and is not part of the neighbourhood of a variable it controls, except for semi-private information.*

In this paper, we show how the DPOP algorithm can be adapted to provide these four types of privacy guarantees. We achieve agent and topology privacy through the use of codenames, limited constraint privacy through the use of obfuscation, and limited decision privacy through the use of codenames.

2 Preliminaries

2.1 Distributed Constraint Satisfaction Problems

Definition 5 (DisCSP). *A discrete* distributed constraint satisfaction problem *(DisCSP) is a tuple $< \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C} >$:*

- $\mathcal{A} = \{a_1, ..., a_k\}$ *is a set of agents*
- $\mathcal{X} = \{x_1, ..., x_n\}$ *is a set of variables. Each variable x_i is controlled by an agent $a(x_i)$*
- $\mathcal{D} = \{d_1, ..., d_n\}$ *is a set of finite variable domains*
- $\mathcal{C} = \{c_1, ..., c_m\}$ *is a set of constraints, where each constraint c_i is a function of scope $(x_{i_1}, \cdots, x_{i_l})$, $c_i : d_{i_1} \times .. \times d_{i_l} \rightarrow \{0, 1\}$, assigning 0 to feasible tuples, and 1 to infeasible ones. Constraints are known to all agents that control a variable involved in the constraint.*

A solution is a complete assignment such that the sum of constraint values $\sum_{c_i \in \mathcal{C}} c_i = 0$, which is the case exactly when it is consistent with all constraints. We can thus understand these values as a cost. The framework can be extended to partial satisfaction or general constraint optimization by finding an assignment that minimizes the cost and letting constraints be general real functions.

In the following, we assume that all agents know the number n of variables in the problem, that the graph formed by the constraints is connected (otherwise each sub-problem can be solved independently), and that all agents that control variables in a given constraint can communicate securely. We also assume that all constraints are either unary or binary, which is a common assumption in the literature that can be made without loss of generality. In a slight abuse of language, we also sometimes indifferently write "variable x_i" instead of "agent $a(x_i)$."

Slot Allocation as DisCSP: We model the airport slot allocation example by two sets of variables x and $\hat{x}$. x_a^b is controlled by the airport offering slot b and takes value 1 if it allocates slot b to airline a and 0 otherwise. $\hat{x}_a^b$ is a variable controlled by airline a and constrained to be equal to x_a^b. Three types of constraints exist on these variables:

1. Each slot b can only be assigned to at most one airline, so for any pair of variables x_a^b and x_c^b, the combination of assignments where they are both assigned 1 is infeasible.
2. Each airline has private constraints on the feasible combinations of slot assignments. For example, since airline A is interested in getting at least one of the slots, then $c_A(\hat{x}_A^y = 0, \hat{x}_A^z = 0) = 1$ and $c_A = 0$ in all other cases.
3. Corresponding x and $\hat{x}$ must have equal values.

Constraints of type 1 and 2 must be kept private to the corresponding airports and airlines, for they would otherwise reveal important competitive information to competing airlines. As shown in the constraint graph of Figure 1, these can in fact be seen as constraints internal to the agents that define them. The only inter-agent constraints are the constraints of type 3, which should also be kept private to the airport and airline that control the corresponding variables.

2.2 Depth-First Search (DFS) Trees

DPOP works on a Depth-First Search (DFS) traversal of the constraint graph. It constructs a spanning pseudotree consisting of the constraints used to discover the nodes and linking parents to children by edges called *tree edges*. All other constraints become *back edges* that link pseudoparents to pseudochildren. Because of the DFS tree construction, all pseudoparents are also ancestors in the spanning tree.

The *separator* Sep_i of x_i is the set of ancestors of x_i whose removal disconnects the subtree rooted at x_i from the rest of the tree. A node's separator can be determined recursively: for a leaf, it is the union of its parent and all pseudoparents; and for a non-leaf node, the union of its parent, pseudoparents, and its children's separators, minus itself.

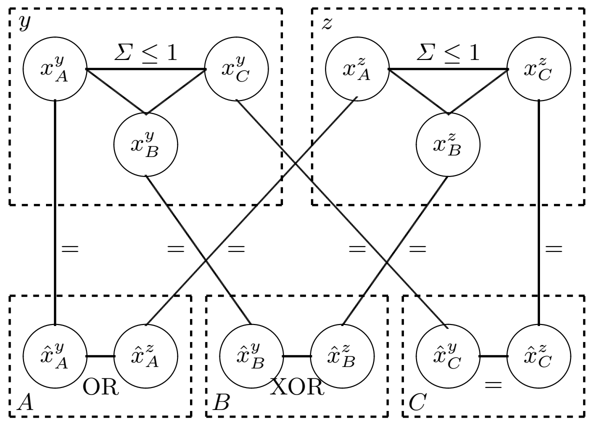

Fig. 1. Constraint graph corresponding to the example in which A wants y OR z, B wants y XOR z, and C wants either both y and z or none.

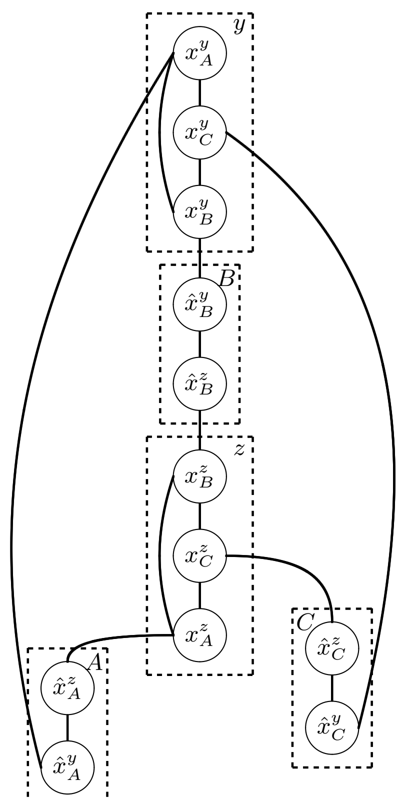

Fig. 2. One possible DFS tree for the example.

2.3 *DPOP*: Dynamic Programming Optimization

DPOP was introduced in [6]. *DPOP* is an instance of the general bucket elimination scheme from [16], which is adapted for the distributed case. *DPOP* has three phases:

Phase 1 – DFS Tree Generation: A DFS tree is obtained from the constraint graph (as in Figure 2) by first running a decentralized leader election algorithm in order to decide upon the root of the pseudotree. Once the root has been identified, it initiates a decentralized DFS traversal of the graph. There are distributed DFS algorithms (for example [6]) that do this by token-passing with a linear number of messages. As a result, each node consistently labels its neighbors as parent/child or pseudoparent/pseudochild. The DFS tree serves as a communication structure for the other two phases of the algorithm: *UTIL* messages (Phase 2) travel bottom-up, and *VALUE* messages (Phase 3) travel top-down, only via tree-edges. Sibling nodes do not exchange any messages.

Phase 2 – *UTIL* Propagation: The agents (starting from the leaves) send *UTIL* messages to their parents. The subtree of a node X_i can influence the rest of the problem only through X_i's separator, Sep_i. Therefore, a *UTIL* message contains the optimal cost obtained in the subtree for each instantiation of Sep_i.

Phase 3 – *VALUE* Propagation: This is a top-down propagation initiated by the root, when Phase 2 has finished. Each node determines its optimal value based on computation from Phase 2 and the *VALUE* message it receives from its parent. Then, it sends this value to its children through *VALUE* messages.

It has been proven in [6] that *DPOP* produces a linear number of messages. Its complexity lies in the size of the *UTIL* messages: the largest one is space-exponential in the induced width of the DFS ordering used.

2.4 Privacy Loss in *DPOP*

In *DPOP*, privacy is lost along all four dimensions listed in the introduction as a result of executing each of *DPOP*'s three phases. Specifically, in the DFS construction phase, agents learn the identity of all their ancestors (even the ones with whom they are not connected), and the existence and identity of back-edges in the DFS. In the example from Figure 2, agent A learns about agent B as its ancestor, and agent z learns about the existence of agent y, and that agent C has a constraint with agent y.

Second, in the *UTIL* propagation phase, all the costs passed in the messages are in clear text, and thus transparent to every agent. Some information is semi-private, for example an agent learns what values are feasible for its own variables under different circumstances. However, the algorithm also leaks consistency information about the variables controlled by other agents: for example, agent z could learn that certain assignments of x_A^y and x_C^y are not feasible for A and C.

Algorithm 1: P-DPOP: DPOP with privacy guarantees.

P-DPOP($\mathcal{A},\mathcal{X},\mathcal{D},\mathcal{C}$):

Initialization:

1. For each binary constraint $c(x,y) \in \mathcal{C}$, agent $a(x)$ generates a vector of random obfuscating keys $\mathbf{O}(x)$ that it sends to agent $a(y)$, which does likewise
2. For each variable $x_i \in \mathcal{X}$, agent $a(x_i)$ generates a codename $C(x_i)$, and codenames $C(v_1), \ldots, C(v_k)$ for x_i's domain values, and sends them to all agents owning a variable linked to x_i by a constraint

Anonymous DFS construction:

3. Choose root of DFS tree using Algorithm 2
4. Construct DFS labelling using Algorithm 3

***UTIL* propagation protocol:**

5. Wait for *UTIL* messages from all children
6. Partially deobfuscate received *UTIL* messages using known keys and codenames
7. As in *DPOP*, join resulting messages with own unary constraints and binary constraints involving (pseudo-)parents' variables; project x_i out
8. Obfuscate result and send to parent

***VALUE* propagation:**

9. Wait for *VALUE* message from parent; deobfuscate it
10. Compute optimal value v_i^* for x_i
11. Send *VALUE* messages to all children using the codenames $C(x_i)$ and $C(v_i^*)$

Third, in the *VALUE* propagation phase, assignments of variables circulate unencrypted. For example, agent z receives the final assignments for variables x_A^y and x_C^y in clear text, as this is required for z to make its own final decision.

3 *P-DPOP*: Privacy Guarantees for *DPOP*

This section introduces *P-DPOP*, an algorithm for DisCSP that provides privacy guarantees. *P-DPOP* is an extension to *DPOP*, and we show in the following the modifications from *DPOP*, and how they provide privacy guarantees. The *P-DPOP* algorithm is described in Algorithm 1.

3.1 Agent and Topology Privacy in DFS Construction

The DFS tree is created in a distributed fashion following a message exchange protocol in which agents only communicate with their neighbors in the constraint graph. It consists of two phases: first, the variable at the root of the tree must be chosen; second, each variable must identify the nature of its relationships with its neighbors. Both phases guarantee *strong agent privacy* and *topology privacy*.

Algorithm 2: Anonymous leader election to choose the root of the tree

elect_leader($\mathcal{A}$): each agent $a \in \mathcal{A}$ does:

Generate unique obfuscated identifying number `ID`
Initialize known maximum `ID` to $max \leftarrow$ **rand**$(0 \ldots \texttt{ID})$
$nb\_lies = \leftarrow$ **rand**$(n \ldots 2n)$
for nb_lies *times* **do**
  Send max to all neighbors
  Get $max_1 \ldots max_k$ from all neighbors
  $max\_tmp \leftarrow$ **max**$(max, max_1, \ldots, max_k)$
  $max \leftarrow$ **rand**$(max\_tmp \ldots$ **max**$(\texttt{ID}, max\_tmp))$

$max \leftarrow$ **max**$(max, \texttt{ID})$
for $(3n - nb\_lies)$ *times* **do**
  Send max to all neighbors
  Get $max_1 \ldots max_k$ from all neighbors
  $max \leftarrow$ **max**$(max, max_1, \ldots, max_k)$

if $max = \texttt{ID}$ **then**
  Choose any of the agent's variables x_r and mark it as the root of the DFS tree

Anonymous leader election (Algorithm 2): We assume that each agent has a unique identifier, for example its MAC address. It anonymizes this number, for instance by taking it as exponent in a finite field exponentiation, to generate a unique obfuscated identifying number `ID`. The protocol then consists, for each agent, in sending its `ID` number to its neighbors, updating it to the maximum of its `ID` and its neighbors', and then repeating these two steps such that the knowledge of the maximum `ID` propagates progressively to all agents. In the end, only one agent has its initial `ID` number equal to the computed maximum; this agent is the leader, and picks one of its variables x_r to be the root of the tree. This protocol converges in n steps, where n is the maximum distance of any two nodes in the graph, for which the number of nodes in the problem is an upper bound.

A shortcoming of this protocol is that the neighbours of the root agent receive the maximum ID in the first cycle, and can thus tell who the root is. This is avoided by first letting the agents give a random lower number and substituting the true ID at an unknown randomly chosen time. In this way, the protocol does not leak any information about the topology of the constraint graph.

Distributed DFS traversal (Algorithm 3): Once the root variable x_r has been marked, the actual DFS construction protocol can be started. In this algorithm, each variable maintains a list of neighbor variables in the constraint graph, defined as those variables that it shares a constraint with. It assigns one of the following labels to each neighbor: *open*, *parent*, *child*, *pseudo-parent* and

Algorithm 3: Distributed DFS traversal to generate the DFS tree

DFS_traversal: for each variable $x \in \mathcal{X}$, a(x) does:

Mark all neighbours as *open*
if *x is marked as the root variable x_r* **then**
  Mark next *open* neighbor as *child* and send it a CHILD token

while *true* **do**
  Wait for incoming token
  if *received very first CHILD token and x is not x_r* **then**
    Mark sender as *parent*

  else if *received CHILD token from* open *neighbor* **then**
    Mark sender as *pseudo-child* and reply with a PSEUDO token

  else if *received PSEUDO token* **then**
    Mark sender as *pseudo-parent*

  if *has* open *neighbor* **then**
    Mark next *open* neighbor as *child* and send it a CHILD token
  else
    Send CHILD token to parent (if x is not x_r) and terminate

pseudo-child. Initially, all neighbors are *open*. The root variable starts by sending a CHILD token that will traverse the constraint graph in DFS order. When a non-root variable receives its very first CHILD token, it marks the sender as its *parent*, and forwards the token to its next *open* neighbor, which it marks as a *child*. When the variable runs out of *open* neighbors, it resends the CHILD token to its parent, hereby telling it that its whole subtree has been explored, and can proceed with the next phase of the P-DPOP algorithm. When a variable receives the CHILD token from the variable it previously sent it to, it knows it can forward it to its next *open* neighbor, which it marks as another *child*. However, if it receives the CHILD token from a different, still *open* neighbor, it can infer there is a cycle in the constraint graph. It then marks the sender as a *pseudo-child*, and replies with a special PSEUDO token to notify it of their pseudo-child/pseudo-parent relationship.

3.2 Limited Constraint Privacy in *UTIL* Phase via Obfuscation

Observe first that if the constraint graph is a tree, then agents only learn about their direct neighbors and so both agent and constraint privacy are guaranteed also during *UTIL* and *VALUE* propagation phases.

However, privacy problems can arise from the presence of a back edge. A tree edge in the DFS tree could be called upon to transmit a multidimensional message that refers to another agent elsewhere in the tree (namely, the root of the back edge). For instance, in *DPOP*, agent A would send the message $UTIL_{A \to z}$ in Table 1 to agent z, containing references to the airport offering slot y, whose identity A might want to hide from z, and also and more importantly from possible competing airlines that could be asked to relay this information to y (such as airline B).

Table 1. Cleartext message $UTIL_{A \to z}$ in *DPOP*.

	$x_A^y = 0$	$x_A^y = 1$
$x_A^z = 0$	1	0
$x_A^z = 1$	0	0

Table 2. Obfuscated message $UTIL_{A \to z}$ in *P-DPOP*.

	$\Gamma = \alpha$	$\Gamma = \beta$
$x_A^z = 0$	12346	23456
$x_A^z = 1$	12345	23456

We identify the corresponding variable's name as well as its different values by codenames that are only known to the agents at both ends of the back edge. In this way, agents in between do not know what variable the message refers to, nor the possible values for this variable. The codenames are be generated by the root of the back edge and communicated through a secure channel to its pseudochild at the other end of the back edge. In this example, the airport offering slot y decides to represent $x_A^y = 0$ by $\Gamma = \alpha$, and $x_A^y = 1$ by $\Gamma = \beta$, and sends these codenames to airline A.

The use of codenames still leaves open the possibility that an agent recognizes the meaning of a message through the values it carries. For example, in a resource allocation problem, an inconsistency usually arises from the non-allocation of a resource. Furthermore, the pattern of consistencies itself can be valuable information that should be protected.

Our solution is to obfuscate constraints by adding large random numbers to their values. If the same number is added uniformly to all values that need to be compared, the results of the comparisons will be unaffected by this obfuscation. This makes it possible to carry out the essential dynamic programming operations on obfuscated numbers while revealing only certain kinds of information.

For instance, when receiving the message $UTIL_{A \to z}$, the airport offering slot z could infer that there is an OR constraint between its variable x_A^z and some other variable (represented by the codename Γ). This is a piece of information that airline A might not want to reveal. To obfuscate it, the airport offering slot y sends through a secure channel a vector of secret, large random numbers $\mathbf{O}(x_A^y) = (12345, 23456)$ to airline A, which is going to add them to all costs in its message corresponding to $x_A^y = 0$ and $x_A^y = 1$, respectively. The resulting obfuscated *UTIL* message is presented in Table 2.

When receiving the obfuscated message $UTIL_{A \to z}$, the airport is no longer able to compare the two columns, because they have been added two different, unknown numbers. However, it is still able to compare the rows, which is necessary to carry out its local operations on the received cost values. In particular, it can compute the join of the obfuscated message $UTIL_{A\to z}$ from airline A with the other obfuscated message $UTIL_{C\to z}$ (Table 3) received from airline C, without de-obfuscating them. The result of this operation is presented in Table 4, and is a doubly-obfuscated message with respect to variables x_A^y and x_C^y.

Table 3. Obfuscated UTIL message $UTIL_{C\to z}$ obtained by using the codenames Δ for variable x_C^y, γ for 0 and δ for 1, and the vector of random numbers $\mathbf{O}(x_C^y) = (34567, 56789)$.

	$\Delta = \gamma$	$\Delta = \delta$
$x_C^z = 0$	34567	56790
$x_C^z = 1$	34568	56789

Table 4. Join of the two UTIL messages $UTIL_{A\to z}$ and $UTIL_{C\to z}$.

		$\Delta = \gamma$		$\Delta = \delta$	
		$\Gamma = \alpha$	$\Gamma = \beta$	$\Gamma = \alpha$	$\Gamma = \beta$
$x_C^z = 0$	$x_A^z = 0$	46913	58023	69136	80246
	$x_A^z = 1$	46912	58023	69135	80246
$x_C^z = 1$	$x_A^z = 0$	46914	58024	69135	80245
	$x_A^z = 1$	46913	58024	69134	80245

The airport offering slot z then joins this matrix with its constraint $x_A^z + x_B^z + x_C^z \leq 1$ (which consists in adding 1 to cost values corresponding to $x_A^z + x_B^z + x_C^z > 1$), and then projects out variables x_A^z and x_C^z. This projection is done by comparing cost values row-wise, which is feasible without de-obfuscation since the rows have been added the same vector of numbers. The result is presented in Table 5, and corresponds to the message that is sent to airline B.

Table 5. Utility message $UTIL_{z\to B}$.

	$\Delta = \gamma$		$\Delta = \delta$	
	$\Gamma = \alpha$	$\Gamma = \beta$	$\Gamma = \alpha$	$\Gamma = \beta$
$\hat{x}_B^z = 0$	46912	58023	69135	80245
$\hat{x}_B^z = 1$	46913	58023	69135	80246

Table 6. *UTIL* message $UTIL_{B\to y}$.

	$\Delta = \gamma$		$\Delta = \delta$	
	$\Gamma = \alpha$	$\Gamma = \beta$	$\Gamma = \alpha$	$\Gamma = \beta$
$x_B^y = 0$	46913	58023	69135	80246
$x_B^y = 1$	46912	58023	69135	80245

Upon receipt of this message, airline B is able to infer that the minimum cost achievable by its subtree depends not only on its decisions (i.e. the value assigned to variable $\hat{x}_B^z$), but also on two other variables represented by the codenames Γ and Δ. However, it does not know what these codenames refer to. Furthermore, for a given value of $\hat{x}_B^z$, it does not know how the minimum achievable cost depends on Γ and Δ, since it cannot compare columns without knowing the secret numbers that were used to obfuscate them.

As described previously, the airline can still carry out its local computations without de-obfuscating the cost values. In particular, it first joins the received *UTIL* message with its XOR constraint, which corresponds to adding 1 to cost values corresponding to $\hat{x}_B^y = \hat{x}_B^z$. Projecting out variable $\hat{x}_B^z$ then yields the message $UTIL_{B\to y}$ (Table 6) that is sent to the airport offering slot y. Again, this projection can be carried out without de-obfuscation, since all costs on the same column have been added the same random number, and the projection is done by comparing costs row-wise.

The airport is then able to de-obfuscate the message simply by subtracting the vector of secret random numbers $\mathbf{O}(x_A^y) = (12345, 23456)$ from the columns corresponding to $(\Gamma = \alpha, \Gamma = \beta)$, and $\mathbf{O}(x_C^y) = (34567, 56789)$ from $(\Delta = \gamma, \Delta = \delta)$. Finally decoding the codenames yields the cost matrix in Table 7. Joining this with the constraint $x_A^y + x_B^y + x_C^y \leq 1$ (i.e. adding 1 to all costs that violate this constraint) yields the final cost matrix in Table 8.

Table 7. De-obfuscated *UTIL* message $UTIL_{B\rightarrow y}$.

	$x_C^y = 0$		$x_C^y = 1$	
	$x_A^y = 0$	$x_A^y = 1$	$x_A^y = 0$	$x_A^y = 1$
$x_B^y = 0$	1	0	1	1
$x_B^y = 1$	0	0	1	0

Table 8. Join of $UTIL_{B\rightarrow y}$ with the constraint $x_A^y + x_B^y + x_C^y \leq 1$.

	$x_C^y = 0$		$x_C^y = 1$	
	$x_A^y = 0$	$x_A^y = 1$	$x_A^y = 0$	$x_A^y = 1$
$x_B^y = 0$	1	0	1	2
$x_B^y = 1$	0	1	2	1

Based on this cost matrix, the airport can choose either one of the two feasible decisions (with a cost of 0), which are to assign its slot y to A or to B, respectively.

3.3 Limited Decision Privacy through the Use of Codenames

During the *VALUE* propagation phase, decisions are made and sent down the tree. In order to provide the same level of privacy as during the *UTIL* propagation phase, variables and values are referred to by their codenames. For instance, if $x_B^y = 0$ is chosen as the optimal decision, then the airport offering slot y sends the message

$$VALUE_{y\rightarrow B} = \{x_B^y = 0, \Gamma = \beta, \Delta = \gamma\}$$

to airline B, which is then not able to learn that slot y was assigned to the competing airline A.

In most applications, agents will learn the values chosen for variables they have constraints with anyway, so limited decision privacy is sufficient. For instance, in our example, airline B would eventually be able to infer the value of variable x_B^y from the value of its variable $\hat{x}_B^y$ anyway, since they are constrained to be equal. However, we recognize that there may be applications that require full decision privacy.

3.4 Privacy properties

Proposition 1. *Algorithms 1,2 and 3 preserve agent privacy.*

Proof. All variables are identified by codenames. Codenames of variables are only communicated between agents that control variables in the same constraint, and agents have no other way of learning codenames.

Proposition 2. *Algorithms 1,2 and 3 preserve topology privacy.*

Proof. Variables are identified by codenames. Thus, both in the leader election and in the DFS tree generation, an agent does not learn anything other than the codename about agents that it is does not share a constraint with.

Further information can be inferred from the constructed DFS tree. Here, the presence of a backedge shows the existence of a cycle in the constraint graph. Note that, if the backedge occurs in the propagation from variable x_i, it indicates a cycle that involves this variable x_i. Thus, it does not leak information about any topological element that x_i is not involved in, and so topology privacy is preserved.

Proposition 3. *Algorithm 1 preserves limited constraint privacy.*

Proof. Information about constraints is transmitted during the UTIL propagation phase. Privacy could be violated in two ways: by inference from the obfuscated value to a value itself, and by inference from combinations of values obfuscated with the same key. We first prove the protection of the obfuscation.

Let $X \in [0, 2^l]$ identify the cost of a particular tuple (in this case, it is the number of conflicts entailed by that tuple). An eavesdropping agent A that controls k variables observes this cost in at most k obfuscated versions $Y = Y_1, .., Y_k$, each time obfuscated with a different and statistically indepedent $Z_i \in [0, 2^m]$ that arises through the fact that certain costs may be known to be the same and observed several times. Then the information that $Y_1, .., Y_k$ gives about X is:

$$\begin{aligned} I(Y;X) = I(X;Y) &= H(Y) - H(Y|X) \\ &= H(Y) - \sum_{i=1}^{k} H(Y_i|X) = H(Y) - \sum_{i=1}^{k} H(Z_i) \end{aligned}$$

Now $Y_i \in [X, X+2^m]$ and $H(Y_i) \leq log(X+2^m)$, and also $H(Y) \leq k \cdot log(X+2^m)$ because of independence. So we have that

$$\begin{aligned} I(Y;X) &\leq k \cdot (log(X + 2^m) - log(2^m)) \\ &\leq k \cdot (log((2^l + 2^m)/2^m) = k \cdot log(1 + 2^{(l-m)}) \end{aligned}$$

Thus, the bound on the information that the obfuscated observations give about the cost can be made arbitrarily small by increasing m, and we can reach any desired level of privacy.

Further information can be inferred from the fact that the same random number Y_i must be used to obfuscate an entire column of the UTIL message. In particular, the agent that owns a variable x_j can tell the differences in cost of different values for this variable. To use this information for inference about particular tuples would require background information about at least one of the values, for example knowing the minimum or maximum cost value. However, this information is not available to the agent; even the distribution of cost values cannot be used to make inferences about minima or maxima as it is self-similar and so does not reveal anything about the absolute values.

The exception is that in the VALUE propagation phase, the agent finds out that a certain combination of value assignments for other variables leads to a consistent solution, thus allowing it to infer that the minimum value in the corresponding column is zero. Consequently, it can tell the number of conflicts for all these values. Note that the set of values for which there is no conflict is semi-private information, as each of them corresponds to a consistent solution of the CSP and would thus be leaked by that solution. In certain cases, for example for functional constraints, knowing that a certain value has k conflicts can be used to infer that a value of another connected variable has k or $k-1$ conflicts. However, because of topology privacy, the agent does not know what constraints these conflicts result from, and thus cannot infer anything about their tuples. Hence, the only information that can be gained is semi-private information.

Proposition 4. *Algorithm 1 preserves limited decision privacy.*

Proof. During VALUE propagation, agents learn decisions for variables that they either share a constraint with or that are part of a backedge that is part of the separator they control. When they share a constraint with the variable, limited decision privacy is not violated. Variables involved in backedges are identified by codenames, as are the values that they take. Unless they share a constraint, the agent does not know the meaning of any codename, and thus cannot infer either the variable nor the value that it has taken.

Because of constraint privacy, the decision on one variable does not allow to infer the decision on another variable. Thus, limited decision privacy is preserved.

4 Conclusions

Distributed constraint satisfaction and optimization has been studied for many years, and privacy has often been cited as a primary reason for using distributed algorithms. In this paper, we show how an existing algorithm for distributed constraint satisfaction, *DPOP* [6], can be adapted to give the required privacy guarantees with no increase in algorithmic complexity. Like *DPOP*, it can thus solve problems of realistic size.

We note that the methods described here work equally well for partial constraint satisfaction or even general optimization when the scope of constraint valuations is extended from $\{0, 1\}$ to the real numbers. However, such settings pose additional issues of self-interest as agents will have an interest to drive solutions towards those that satisfy their interest best. These can be addressed with economic mechanisms but are beyond the scope of this paper.

The methods described here can be also be applied in the context of other distributed constraint satisfaction algorithms. For example, obfuscation and codenames could be used with search. However, as pointed out in [14], such algorithms leak information through the runtime of the solving process itself, so the value of privacy guarantees is not as clear.

References

1. International Air Transport Association (IATA): Worldwide Scheduling Guidelines. 12th edn. (December 2005)
2. Yokoo, M., Hirayama, K.: Algorithms for distributed constraint satisfaction: A review. Autonomous Agents and Multi-Agent Systems **3**(2) (2000) 185–207
3. Silaghi, M.C., Sam-Haroud, D., Faltings, B.: Asynchronous search with aggregations. In: AAAI/IAAI, Austin, Texas (2000) 917–922
4. Modi, P.J., Shen, W.M., Tambe, M., Yokoo, M.: ADOPT: Asynchronous distributed constraint optimization with quality guarantees. AI Journal **161** (2005) 149–180
5. Mailler, R., Lesser, V.: Solving distributed constraint optimization problems using cooperative mediation. Proceedings of Third International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2004) **1** (2004) 438–445
6. Petcu, A., Faltings, B.: DPOP: A scalable method for multiagent constraint optimization. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI-05, Edinburgh, Scotland (Aug 2005) 266–271
7. Franzin, M.S., Freuder, E.C., Rossi, F., Wallace, R.: Multi-agent constraint systems with preferences: Efficiency, solution quality, and privacy loss. Computational Intelligence **20**(2) (May 2004) 264—286
8. Maheswaran, R.T., Pearce, J.P., Bowring, E., Varakantham, P., Tambe, M.: Privacy loss in distributed constraint reasoning: A quantitative framework for analysis and its applications. Autonomous Agents and Multi-Agent Systems (JAAMAS) **13**(1) (July 2006) 27–60
9. Greenstadt, R., Pearce, J.P., Tambe, M.: Analysis of privacy loss in distributed constraint optimization. In: Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI'06), Boston, Massachusetts, U.S.A., AAAI Press (July 16–20 2006) 647–653
10. Brito, I., Meseguer, P.: Distributed forward checking. In: Proceedings of the Nineth International Conference on Principles and Practices of Constraint Programming (CP'03). Volume 2833., Kinsale, Ireland, Springer Berlin / Heidelberg (September 29–October 3 2003) 801–806
11. Brito, I., Meseguer, P.: Distributed forward checking may lie for privacy. In: Proceedings of the Nineth International Workshop on Distributed Constraint Reasoning (CP-DCR'07), Providence, RI, USA (September 23 2007)
12. Greenstadt, R., Grosz, B., Smith, M.D.: SSDPOP: Using secret sharing to improve the privacy of DCOP. In: Proceedings of the Nineth International Workshop on Distributed Constraint Reasoning (CP-DCR'07), Providence, RI, USA (September 23 2007)
13. Suzuki, K., Yokoo, M.: Secure generalized vickrey auction using homomorphic encryption. Financial Cryptography **2742** (2003) 239–249
14. Silaghi, M.C., Mitra, D.: Distributed constraint satisfaction and optimization with privacy enforcement. In: Proceedings of the 2004 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'04), Washington, DC, USA, IEEE Computer Society (2004) 531–535
15. Silaghi, M.C., Faltings, B., Petcu, A.: Secure combinatorial optimization simulating DFS tree-based variable elimination. In: 9th Symposium on Artificial Intelligence and Mathematics, Ft. Lauderdale, Florida, USA (Jan 2006)
16. Dechter, R.: Constraint Processing. Morgan Kaufmann (2003)