DisCSPs with Privacy Recast as Planning Problems for Self-Interested Agents

Julien Savaux,
Julien Vion,
Sylvain Piechowiak,
and René Mandiau
LAMIH UMR CNRS 8201
University of Valenciennes, France

Toshihiro Matsui
Nagoya Institute of Technlogy, Japan

Katsutoshi Hirayama
Kobe University, Japan

Makoto Yokoo
Kyushu University, Japan

Shakre Elmane, Marius Silaghi
Florida Institute of Technology, USA

***Abstract*—Much of the Distributed Constraint Satisfaction Problem (DisCSP) solving research has addressed cooperating agents, and privacy was frequently mentioned as a significant motivation of the decentralization. While privacy may have a role for cooperating agents, it is easier understood in the context of self-interested utility-based agents, and this is the situation considered here. With utility-based agents, the DisCSP framework can be extended to model privacy and satisfaction under the concept of utility. We introduce Utilitarian Distributed Constraint Satisfaction Problems (UDisCSP), an extension of the DisCSP that exploits the rewards for finding a solution and the costs for losing privacy as guidance for the utility-based agents. A parallel can be drawn between Partially Observable Markov Decision Processes (POMDPs) and the problems solved by individual agents for UDisCSPs. Common DisCSP solvers are extended to take into account the utility function. In these extensions we assume that the planning problem is further restricting the set of communication actions to only the ones available in the corresponding solver protocols. The solvers obtained propose the action to be performed in each situation, defining thereby the policy of the agents.**

I. Introduction

In distributed problems, prviacy of intelligent agents is a major concern.. In Distributed Constraint Satisfaction Problems (DisCSP), agents have to find values to a set of shared variables while respecting given constraints, frequently assumed to have unspecified privacy implications. To find such assignments, agents exchange messages until a solution is found or until one agent detects that there is no solution to the problem. Thus, agents commonly reveal information during the solution search process, causing privacy to be a major concern in DisCSPs [1].

The artificial intelligence assumption is that utility-based agents are able to associate each state with a utility value [2]. As such, each action is associated with the difference between initial and final utilities. If a user is concerned about its privacy, then such a user can associate a utility value with the privacy of each piece of information in the definition of their local problem. Since the users are interested in solving the problem, they must be also able to quantify the utility each of them draws from finding the solution. Here we approach the problem by assuming that privacy has a utility that can be aggregated with the utility value of solving the problem. We evaluate how much privacy is lost by the agents during the problem solving process, by the total utility of the information that was revealed. The availability of a value from the domain of a variable for the DisCSP in the presence of the constraints for an agent, is the kind of information that the agents want to keep private. For example, an assignment of a value to a variable has a cost quantifying the desire of the agent to maintain its feasibility private. In traditional algorithms, agents participate in the search process until an agreement is found. We investigate the case where, being utility-driven, an agent may stop its participation if the utility of the privacy expected to be lost overcomes the reward for finding a solution of the problem. Simple extensions to basic algorithms are investigated to exploit the utilitarian model of privacy.

We introduce Utilitarian Distributed Constraint Satisfaction Problems (UDisCSP), an extension of the DisCSPs that exploits the utilities and costs of agreements and privacy as guidance for the utility-based agents, where the utility of each state (aka worth) is estimated as the difference between the expected rewards for agreements on assignments for shared variables, and the expected cost of privacy loss. With the help of the UDisCSP we can now define the problem of each agent as a planning problem. In UDisCSPs, each agent has a planning problem, where solving methods can be seen as consisting both of (a) policies recommending actions for each state, as well as of (b) mechanisms to update belief states. We show that a parallel can be drawn between Partially Observable Markov Decision Processes (POMDPs) and the problems that individual agents have to solve in UDisCSPs. Therefore, a traditional DisCSP with privacy requirements is viewed as a planning problem. Further, we investigate some simple extensions where these solvers can be adjusted to take into account the utility function. In these extensions, we assume that the set of communication actions possible in the planning problems is further restricted to only the communication primitives available in the corresponding solver protocols. The solvers obtained for the new type of problems propose the action (communication/inference) to be performed in each situation, defining thereby the policy of the agents.

We then evaluate and compare synchronous and asynchronous algorithms according to how well they preserve

privacy. To do so, we generate distributed meeting scheduling problems, as described in [3], [4]. In these problems, each agent owns one variable, corresponding to the status of his agreement for the meeting to schedule, and the domain is the same for all variables. In each problem there is a global constraint that requires all the variables to be equal, and also a unary constraint for each agent.

The paper is organized as follow: after a brief introduction in Section I, Section II presents existing research concerning planning, algorithms for solving DisCSPs, as well as approaches to privacy in DisCSPs and their limits. Further, Section III describes the concepts involved in UDisCSPs and the extensions to common DisCSP solvers that exploit the UDisCSP model to preserve privacy. After a discussion on theoretical implications in Section IV, Section V presents our experimental results. Section VI concludes the paper.

II. Background

A. Planning and POMDPs

Research has already been conducted about the case where multiple agents have synchronous or asynchronous exchanges, resulting in extending Partially Observable Markov Decision Process (POMDP) [5] into event-driven Multiagent POMDP (MPOMDP) [6] as well as with work on Interactive POMDP (I-POMDPs) [7]. A cooperative model for teams solving Distributed Constraint Optimization Problems (DCOPs) without explicit privacy considerations was proposed under the name of Networked Distributed POMDPs (ND-POMDPs) [8]. However, in this work we assume that agents have explicit privacy considerations and are self-interested with personal utilities, and therefore we can see each agent as solving his independent POMDP:

- A multiagent Partially Observable Markov Decision Process (MPOMDP) is a tuple $\langle d, S, A, \Omega, T, O, R\rangle$, where d is the number of agents, S is the set of possible states, A is the set of joint actions, Ω is the joint space of observations, T is the transition function, O is the observation function, and R is the reward function.
- Event-Driven MPOMDPs [6], is an alternative framework for multi-agent decision making under uncertainty, based on the operation of real-time discrete-event systems [9]. In this approach, team decisions are caused by changes in the state of the system. When an agent detects a change, it selects the appropriate action and communicates it to the other agents. This method implies that there are many possible observations to be considered.

B. Backtracking Algorithms

We now present standard algorithms for solving DisCSPs.

1) Synchronous Backtracking: The baseline algorithm for DisCSPs is the Synchronous Backtracking (SyncBT), as presented in [10], [11]. SyncBT is a simple distribution of the standard backtracking algorithm. The agents start by determining a hierarchy between them. The higher priority agent then sends a satisfying assignment of its variable to the next agent with an `ok?` message. The recipient completes the received assignment with an instantiation of its own variable while respecting its constraints, and continues likewise. If an agent is unable to find an instantiation compatible with the current partial assignment it has received, the agent sends a `nogood` message to the previous agent in the hierarchy. The process repeats until a complete solution is built, or until the whole search space is explored. The main efficiency concern is that, since the messages are being sent sequentially, it does not take advantage of possible parallelism.

2) Asynchronous Backtracking: Asynchronous BackTracking (ABT) [10], allows agents to run concurrently. Each agent finds an assignment of its variable and communicates it to the others agent having constraints involving this variable. Agents then wait for incoming messages. They receive an `ok?` message containing an assignment from a related higher priority agent, at the beginning of the resolution and also each time such an agent changes its assignment to avoid constraint violation. An agent eventually receives values proposed by the agents it is connected to by incoming links. These values form a context called `agent view`. When an agent receives an `ok?` message, it integrates the received assignment into its `agent view` and checks if its own solution is consistent with it. If it is not the case, the agent's assignment is changed. The negation of a subset of an `agent view` preventing an agent from finding an assignment that does not violate any of its constraints is called a nogood. If an agent infers a nogood from its constraints and its `agent view`, the assignment of the lowest priority agent involved in the nogood has to be changed. A `nogood` message communicates to that agent the nogood, which is treated by its recipient as a new constraint and can cause it to change its assignment and generate corresponding `ok?`, `addlink`() or `nogood` messages.

C. Privacy

Privacy is an important problem in a lot of applications [12]. In distributed scheduling problems, problems of confidentiality also happen when information is exchanged between agents. Indeed, we know that the assignment of time slots can be difficult if participants do not want to reveal their constraints [13], [14]. Such coordinated decisions are in conflict with the need to keep constraints private [15].

In existing works, several approaches have been developed to deal with privacy in DCOPs. The first approach using cryptographic techniques is [16]. While ensuring privacy [17], [18], cryptographic techniques are usually slower, and sometimes require the use of external servers or computationally intensive secure function evaluation techniques that may not always be available or justifiable for their benefits [19]. Another family of approaches is based on using different search strategies to minimize privacy loss, as defined by certain privacy metrics.

D. Existing Approaches

1) Sample Cryptographic Technique: The approach described in [16], achieves a high level of privacy using encryption, giving more importance to privacy than to the efficiency of the resolution. It consists of using a randomizable public

key encryption scheme. In this algorithm, three servers (value selector, search controller and decryptor) receive encrypted information from agents and cooperate to find an encrypted solution. Relevant parts of the solution are then sent to each agent. When restricted to one single global variable as in [18], this method guarantees that no information is leaked to other agents, except what can be inferred from the value of the agreed solution. We now introduce methods that do not use cryptography.

2) Distributed Private Constraint Satisfaction Problems: A framework called Distributed Private Constraint Satisfaction Problems (DisPrivCSPs), is introduced in [20], modeling the privacy loss for individual revelations. It also models the effect of the privacy loss by assuming that agents may abandon when the incremental privacy loss overcomes the expected gains from finding a solution. Each agent pays a cost if the feasibility of some of its tuple is determined by other agents. The reward for solving the problem is given as a constant. Those concepts were so far used for evaluating qualitatively existing algorithms, but were not integrated as heuristics in the search process.

3) Valuations of Possible States: The Valuations of Possible States (VPS) framework [21], [22], [19] measures privacy loss by the extent to which the possible states of other agents are reduced [23]. Privacy is interpreted as a valuation on the other agents' estimates about the possible states that one lives in. During the search process, agents propose their values in an order of decreasing preference. At the end of the search process, the difference between the presupposed order of preferences and the real one observed during search determines the privacy loss: the greater the difference, the more privacy has been lost.

4) Partially Known Constraints: The Partially Known Constraints (PKC) model [24], uses entropy, as defined in information theory, to quantify privacy and privacy loss. In this method, two variables x_1 and x_2 owned by two different agents may share a constraint. However, not all the forbidden couples (x_1, x_2) are known by both agents. Each agent only knows a subset of the constraints. During the search process, assignment privacy is leaked through `ok?` and `nogood` messages, like in standard algorithms. This problem is solved by not sending the value that is assigned to a variable in a `ok?` message, but the set of values compatible with this assignment. For `nogood` messages, rather than sending the actual assignments, an identifier is used to specify the state of the resolution and is used to check if some assignments are obsolete or not.

III. Concepts

The Distributed Constraint Satisfaction Problem (DisCSP) is the formalism commonly used to model constraint problems distributed between several agents. It is represented by a quadruplet $\langle A, V, D, C \rangle$ where:

- A is a set of agents.
- V is a set of variables, each one being owned by a distinct agent.
- D is a set of domains, each of them defining available values for the corresponding variable.
- C is a set of constraints, each constraint being a relation imposed between variables (*i.e.*, $x_1 = x_2$).

An agent that reveals an assignment to another agent, incurs a cost. For simplicity, for now we consider that once the information is revealed it becomes public, meaning that revealing it to yet another agent will not degrade its privacy.

Example 1. *Suppose a meeting scheduling problem between a professor and two students. They all consider to agree on a time to meet on a given day, to choose between* $8\ am$, $10\ am$ *and* $2\ pm$. *Professor* A_1 *is unavailable at* $2\ pm$, *Student* A_2 *is unavailable at* $10\ am$, *and Student* A_3 *is unavailable at* $8\ am$.

There can exist various reasons for privacy. For example, Student A_2 *does not want to reveal the fact that he is busy at* $10\ am$ *(because he secretly took a second job at that time). The value Student* A_2 *associates with not revealing the* $10\ am$ *unavailability is the salary from the second job (*$\$2,000$*). The utility of finding an agreement for each student is the stipend for their studies (*$\$5,000$*), while for professor it is the a fraction of the value of his project (*$\$4,000$*). This is an example of privacy for absent values or constraint tuples.*

Further Student A_3 *had recently boasted to Student* A_2 *that at* $8\ am$ *he interviews for a job, and he would rather pay* $\$1,000$ *than to reveal that he is not. This is an example of privacy for feasible values of constraint tuples.*

Similarly, participants associate a cost to the revelation of each availability and unavailability. Thus, corresponding agents associate a cost of 1 *to the revelation of their availability at* $8\ am$, *a cost of* 2 *to the one at* $10\ am$, *and a cost of* 4 *to the one at* $2\ pm$. *The reward from finding a solution is* 4 *for* A_1 *and* 5 *for bot Student* A_2 *and Student* A_3.

a) DisCSP: The DisCSP framework models this problem as follow. For simplicity, in the next sections, we will refer to the possible values by their identifier: 1, 2, and 3.

- $A = \{A_1, A_2, A_3\}$
- $V = \{x_1, x_2, x_3\}$
- $D = \{\{1,2,3\}, \{1,2,3\}, \{1,2,3\}\}$
- $C = \{x_1 = x_2, x_2 = x_3, x_1 = x_3,$
  $x_1 \neq 3, x_2 \neq 2, x_3 \neq 1\}$

As it can be observed, DisCSPs cannot model the information concerning privacy. Now we will show how existing extensions model them.

b) DisPrivCSP: The additional parameters are P, to specify the privacy coefficient of each value, and R, to specify the rewards of each coefficient.

- $P = \{P_{A_1}, P_{A_2}, P_{A_3}\} = \{(1,2,4), (1,2,4), (1,2,4)\}$
- $R = \{R_{A_1}, R_{A_2}, R_{A_3}\} = \{(4), (5), (5)\}$

As we see, this framework successfully models all the information described in the initial problem and also measures the privacy loss for each agent. However, it was not yet investigated what is the impact of the interruptions when privacy loss exceeds the revenue threshold, or how agents

could use this information to modify their behavior during the search process to preserve more privacy.

c) VPS: For this problem, with the VPS framework, the 3 participants have to suppose an order of preference between all different possible values for each other agent. Agents initially do not know anything about others agents but the variable they share a constraint with. Agents have no information about others agents privacy requirements. Thus, agents do not expect to receive any value proposal more than another. In this direction one needs to extend VPS to be able to also model the kind of privacy introduced in this example.

d) PKC: With PKC, the individual unavailabilities are only known by the corresponding participant. Only the junction of information known by the two agents over a given constraint can reconstruct the whole problem.

- $A = \{A_1, A_2, A_3\}$
- $V = \{x_1, x_2, x_3\}$
- $D = \{\{1,2,3\}, \{1,2,3\}, \{1,2,3\}\}$
- $C = \{\{x_1 = x_2, x_1 = x_3, x_1 \neq 3\},$
  $\{x_1 = x_2, x_2 = x_3, x_2 \neq 2\},$
  $\{x_1 = x_3, x_1 = x_3, x_3 \neq 1\}\}$

Extensions of PKC can also be proposed to model our example by adding extra parameters for specifying the quantitative information about privacy, as shown below. Next we introduce a framework that can both specify the quantitative input details, and can help agents in their search process.

A. Our Approach: UDisCSP

While some previously described frameworks do model the details of our example, it has until now been an open question as to how they can be dynamically used by algorithms in the solution search process.

We propose to recast a DisCSP as a planning problem. It can be noticed that the rewards and costs in our problem bear similarities with the utilities and rewards commonly manipulated by planning algorithms [25]. As such, we propose to define a framework which, while potentially being equivalent in expressing power to existing DisCSP extensions, would nevertheless explicitly specify the elements of the corresponding family of planning problems. We introduce the Utilitarian Distributed Constraint Satisfaction Problem (UDisCSP) [26]. Unlike previous DisCSP frameworks, besides results, we are also interested in the solution process. A policy is a function that associates each state of an agent with an action that it should perform [2].

We define an *agreement* as a set of assignments for all the variables with values from their domain, such that all the constraints are satisfied.

Definition 1. *A UDisCSP is formally defined as a tuple* $\langle A, V, D, C, U, R\rangle$ *where:*

- $A = \langle A_1, \ldots, A_n\rangle$ *is a vector of* n *agents*
- $V = \langle x_1, \ldots, x_n\rangle$ *is a vector of* n *variables. Each agent* A_i *controls the variable* x_i*.*
- $D = \langle D_1, \ldots, D_n\rangle$ *where* D_i *is the domain for the variable* x_i*, known only to* A_i*, and a subset of* $\{1, \ldots, d\}$*.*
- $C = \{C_1, \ldots, C_m\}$ *is a set of interagent constraints.*
- $U = \{u_{1,1}, \ldots, u_{n,d}\}$ *is a matrix of costs where* $u_{i,j}$ *is the cost of agent* A_i *for revealing whether* $j \in D_i$*.*
- $R = \langle r_1, \ldots, r_n\rangle$ *is a vector of rewards, where* r_i *is the reward agent* A_i *receives if an agreement is found.*

The state *of agent* A_i *includes the subset of* D_i *that it has revealed, as well as the achievement of an agreement. The problem is to define a set of communication actions and a policy for each agent such that their utility is maximized.*

Note that the solution of a UDisCSP does not necessarily include an agreement. In principle the set of available actions for agents consists of any communication operator, as well as any local inference computation. The participants are utility-based agents [2] and try to reach the optimal state.

Example 2. *The DisCSP in Example 1 is extended to UDisCSP by specifying the additional parameters* U *and* R*:*
$U = \{u_{1,1} = 1, u_{1,2} = 2, u_{1,3} = 4,$
$u_{2,1} = 1, u_{2,2} = 2, u_{2,3} = 4,$
$u_{3,1} = 1, u_{3,2} = 2, u_{3,3} = 4\}.$
$R = \langle 4, 5, 5\rangle.$

B. Algorithms

Now we discuss how the basic ABT and SyncBT algorithms are adjusted to UDisCSPs. The state of an agent includes the `agent view`. After each state change, each agent computes the estimated utility of the state reached by each possible action, and selects randomly one of the available actions leading to the a state with the maximum expected utility.

In our algorithms, agents compute the risk of one of their assignments to be rejected to estimate the expected utilities. This risk can be re-evaluated at any moment based on data recorded during previous runs on problems of similar tightness (*i.e.*, having the same proportion of forbidden instantiations). Learning from previous experience has been extensively studied in [27]. The learning can be online or offline. For offline learning one calculates the number of messages `ok?` and `nogood` sent during previous executions, called $count$. One also counts how many messages previously sent lead to the termination of the algorithm, in the variable $agreementCount$. The risk for a solution to not lead to the termination of the algorithm, is called $agreementProb$. For online learning, one can update the variables $count$, $agreementCount$ and $agreementProb$ dynamically whenever the corresponding events happen. When previous experiences are not available, the value of $agreementProb$ is set to 0.5 by default.

$$agreementProb = \frac{agreementCount}{count} \tag{1}$$

When `ok?` messages are sent, the agent has the choice of which assignment to propose. When a `nogood` message is scheduled to be sent, agents also have choices of how to express them. Before each `ok?` or `nogood` message, the agents check which available action leads to the highest expected utility. If the highest expected utility is lower than the

current one, the agent announces failure. The result is used to decide between proposing assignments, a nogood, or declaring failure.

We called these modified algorithms SyncBTU and ABTU, respectively. Algorithms SyncBTU and ABTU are obtained by the same modifications of the pseudocodes of SyncBT [10], [1], [11] and of ABT [10], [1], [11].

SyncBTU is obtained by restricting the set of communication actions to the standard communication acts of SyncBT, namely `ok?` and `nogood` messages. The procedures of a solver like SyncBT define a policy, since they uniquely identify a set of actions (inferences and communications) to be performed in each state. A state of an agent in SyncBT is defined by an agent-view and a current assignment of the local variable. The local inferences in SyncBTU are obtained from the ones of SyncBT by a simple extension exploiting the utility information available. The criteria in this research was not to guarantee an optimal policy but to use utility with a minimal change to the original behavior of SyncBT when reinterpreted as a policy. In SyncBTU, the state is extended to also contain a history of revelations of one's values defining an accumulated privacy loss, and a probability to reach an agreement with each action. Similar modifications are done to ABT to obtain ABTU: the restricted set of communication of ABTU is composed of `ok?`, `addlink` and `nogood`. The state and local inferences of ABTU are the same as SyncBTU, while also containing the set of nogoods.

For ABTU, there are three procedures of ABT that have to be modified: `checkagentview`, `when nogood`, and `backtrack`. The new procedure `checkagentview` is shown in Algorithm 1 and is obtained by inserting Lines 8 to 10. They test the privacy loss and only continue as usual if the expected loss is smaller than the expected reward.

For lack of space, we do not include here the modified versions of the other two procedures of ABTU, since they are obtained in the same way from the procedures of ABT in [1], procedure `when nogood`, 7^{th} line, and procedure `backtrack`, 7^{th} line. For SyncBTU, its procedures are obtained from the procedures of SyncBT in an identical way as for ABTU and ABT. Since [10] does not provide pseudocode for SyncBT, we refer the modifications to the pseudocode presented in [11], function `assignCPA`, before Line 7, and function `backtrack`, before Line 6.

To calculate the estimated utility of pursuing an agreement (revealing an alternative assignment), the agent considers all different possible scenarios of the subsets of values that might have to be revealed in the future based on possible rejections received, together with their probability (see Algorithm 2). The algorithm assumes as parameters:

- $agreementProb$ calculated in Equation 1,
- Set of possible values D, and
- Probability of having to select a value from D.

The algorithm then recursively computes the utility of the next possible states, and whether the revelation of the current value v leads to the termination of the algorithm, values stored in variables $costRound$ and $costNonTerminal$. The algorithm returns the estimated cost of privacy loss for the future possible states currently, called $estimatedCost$.

Algorithm 1: checkAgentView in ABTU

Input: D, $agentView$, $agreementProb$, $reward$
Output: $\varnothing$

```
1  when agentView and currentValue inconsistent do
2      if no value in D is consistent with agentView then
3          backtrack;
4      else
5          select d ∈ D where agentView and d are
             consistent;
6          currentValue = d ;
7          if calculateCost
8          (agreementProb, D, 1) ⩾ reward  then
9              interruptSolving();
10         else
11             send(ok?,(x_i,d)) to outgoing links
```

Algorithm 2: calculateCost

Input: $agreementProb$, D, $probD$
Output: $estimatedCost$

```
1  if only one value is left in the domain then
2      return marginalCost(value) × probD;
   else
3      v = first(D);
4      costRound= calculateCost
           (agreementProb, {v}, probD);
5      costNonTerminal= calculateCost
           (1 − agreementProb, D \ {v},
           (1 − agreementProb) × probD);
6      estimatedCost = costRound + costNonTerminal;
7      return estimatedCost;
```

Example 3. *Continuing with Example 1, at the beginning of the computation agent A_1 has to decide for a first action to perform. We suppose the $agreementProb$ learned from previous solvings is 0,5. To decide whether it should propose an available value or not, it calculates the corresponding $estimatedCost$ by calling Algorithm 2 with parameters: the learned $agreementProb = 0{,}5$, the set of possible messages ($D = \{1, 2, 3\}$) and $probD = 1$.*

For each possible value, this algorithm recursively sums the cost for the two scenarios corresponding to whether the action leads immediately to termination, or not. Given privacy costs, the availability of three possible subsets of D may be revealed in this problem: $\{1\}, \{1, 2\}$, and $\{1, 2, 3\}$. The $estimatedCost$ returned is the sum of the costs for all possible sets, weighted by the probability of their feasibility being revealed if an agreement is pursued. At the call, $costRound = u_{1,1} \times 0{,}5$. In the next recursion for $costNonTerminal$, we get $costRound = (u_{1,1} + u_{1,2}) \times 0{,}25$. In the last

recursion, the algorithm returns $(u_{1,1} + u_{1,2} + u_{1,3}) \times 0{,}25$. *The* $estimatedCost$ *obtained is* $u_{1,1} \times 0{,}5 + (u_{1,1} + u_{1,2}) \times 0{,}25 + (u_{1,1} + u_{1,2} + u_{1,3}) \times 0{,}25$. *The expected utility (*$reward$+*estimatedCost* $= 4 - 3 = 1$*) of pursuing a solution being positive, the first value is proposed.*

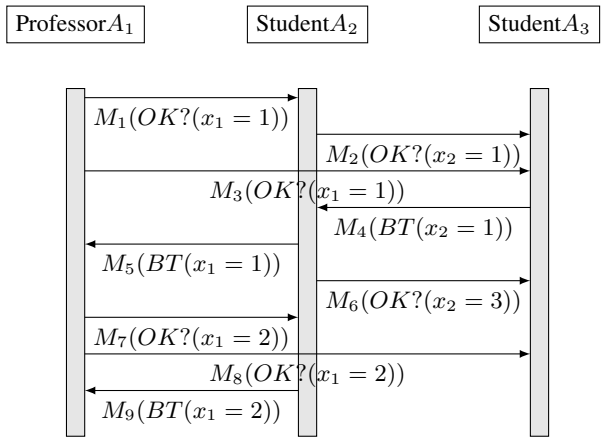

Fig. 1. Interactions between agents during ABT

Next is an illustrative example of other ABTU operations.

Example 4. *With the original ABT, possible obtained traces is depicted in Figure 1, respectively. In Figure 1, we see that Student* A_2 *proposes* $x_2 = 1$ *in Message* M_2 *and* $x_2 = 3$ *in Message* M_6. *In this case, the privacy loss for Student* A_2 *is* $u_{2,1} + u_{2,3} = 1 + 4 = 5$.

However, with ABTU, we do not only use the actual utility of the next assignment to be revealed, but estimate privacy loss using Algorithm 2. After Student A_2 *has already sent* $x_2 = 1$ *with* M_2, *it considers sending* $x_2 = 3$ *with* M_6. *This decision making process is depicted in Figure 2. If the next value,* $2pm$, *is accepted, Student* A_2 *will reach the final state while having revealed* $x_2 = 1$ *and* $x_2 = 3$, *for a total privacy cost of* $u_{2,1}+u_{2,3} = 1+4 = 5$. *If it is not, the unavailability of the last value* $x_2 = 2$ *will have to be revealed to continue the search process, leading to the revelation of all its assignments for a total cost of* 7. *Since both these scenarios have a probability of* 50% *to occur, the* $estimatedCost$ *equals* $((5+7)/2) = 6$. *The utility (*$reward-$*estimatedCost) being equal to* $5 - 6 = -1$, *Student* A_2 *has no interest in revealing* $x_2 = 3$ *and interrupts the solving. Its final privacy loss is only* $u_{2,1} = 2$. *The utility of the final state reached by Student* A_2 *being* 2 *with ABTU, and* 4 *with ABT, ABTU preserves more privacy than ABT in this problem.*

IV. Theoretical Discussion

The assumption that each agent owns a single variable is also not restrictive. Multiple variables in an agent can be aggregated into a single variable by Cartesian product. Nevertheless some algorithms can exploit these underlying structures for efficiency, and this has been the subject of extensive research [30], [31], [32].

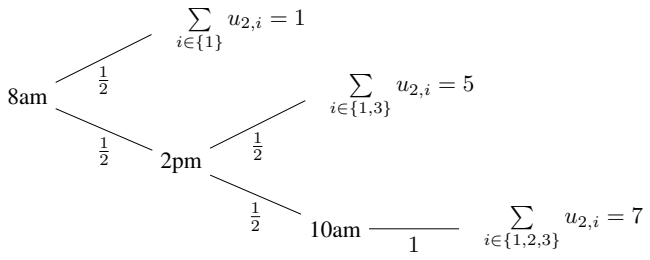

Fig. 2. Calculation of cost from Student A_2 for all scenarios during ABTU

We now discuss how UDisCSP can be interpreted as a planning problem.

a) Comparison with POMDPs: The problem that each agent in UDisCSPs has to solve in UDisCSPs has similarities to a Partially Observable Markov Decision Problem (POMDP). Given ways to approximate observation and transition conditional probabilities, these problems could be reduced to POMDPs. We remind that a POMDP is defined as the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, O, \gamma \rangle$, where $\mathcal{S}$ is the set of possible states of the agent, A is the set of possible actions, $\mathcal{T}$ is the conditional transition probabilities between states, $\mathcal{R}$ is a reward function given states and actions, Ω is the set of possible observations and O is a set of conditional observation probabilities based on states, while γ is a discount factor. A POMDP agent regularly reasons in terms of belief (probability distribution over the states), and tries to build a policy, namely a recommendation of each action to be executed as function of current belief.

For UDisCSPs, the set $\mathcal{S}$ of states of the agent is defined by the possible contents of its `agent view`, of the nogoods stored by the agent, and of the knowledge the agent gathers about the secret elements of the UDisCSP unknown to it.

Set $\mathcal{A}$ of actions available to an agent consist in local reasoning actions as well as communication actions that are a function of the selected protocols (*i.e.*, communication language and ontology). For example, in ABT these communications actions can have as payloads assignment announcements (in `ok?` messages) and nogoods in (`nogood` messages). It is commonly assumed that the set of communication actions are determined by the agents before the actual start of the planning and execution.

Set T of transition probabilities between states given actions can be trained for UDisCSPs in the same way in which we have trained the $agreementProb$.

Set $\mathcal{R}$ of rewards of the POMDP is the same as R for the corresponding UDisCSP. Set Ω of possible observations is given by the possible payloads of the communication actions available. The set of conditional observation probabilities can be trained or assumed. In reported experiments, it is assumed that the message payloads truthfully reveal the corresponding elements of the states of the other agents, while the probability of the remaining elements have to be inferred by the agent.

In this work, we have not taken into consideration the time and therefore we had γ set to 1.

b) Comparison with DisCSPs: The introduced UDisCSP framework can assume without significant loss of generality that interagent constraints are public. This is due to the fact that any problem with private interagent constraints, is equivalent with its dual representation where each constraint becomes a variable [28]. However, note that the privacy of domains mentioned in [33] is not modeled by privacy of constraints.

UDisCSP mainly differs from DisCSP from the perspective of how solution is defined. It does not define solution as an agreement on a set of assignments but as a policy that could eventually reach such an agreement. As such, their comparison is not trivial, as one compares different aspects.

Theorem 1. *UDisCSPs planning and execution is at least as general as DisCSPs solving.*

Proof. A DisCSP can be modeled as a UDisCSPs with all privacy costs equal 0. The obtained UDisCSPs would always reach an agreement, if possible. Therefore the goal of a UDisCSP would also match with the goal of the modeled DisCSP. This implies a tougher class of complexity. □

The space complexity required by ABTU and SyncBTU in each agent is identical with the one of ABT and SyncBT, since the only additional structures are:

- the privacy costs associated with its values, constituting a constant factor increases for domain storage.
- the variables $agreementProb$, $agreement$, $count$ and r_i that require a constant space.

V. Experimental Results

We evaluate our framework and algorithms on randomly generated instances of *distributed meeting scheduling (DMS) problems*. Previous work [34] in distributed constraint satisfaction problems has already addressed the question of privacy in distributed meeting scheduling by considering the information on whether an agent can attend a meeting to be private. They evaluate the privacy loss brought by an action as the difference between the cardinalities of the final set and of the initial set of possible availabilities for a participant.

DMS are generated as follow:

1) creation for the variables and their domains
2) creation of the global and unary constraints
3) generation of the privacy costs

The experiments are carried out on a computer under Windows 7, using a 1 core 2.16 GiHz CPU and 4 GiB of RAM. Figure 3 shows the total amount of privacy lost by all agents, averaged over 50 problems, function of the density of unary constraints. The problems are parametrized as follows: 10 agents, 10 possible values, the utility of a revelation is a random number between 0 and 9, the reward for finding a solution to the problem is 20. Each set of experiments is an average estimation of 50 instances for the different algorithms (*i.e.*, SyncBT, ABT, SyncBTU, ABTU).

TABLE I
General comparison for algorithms along multiple metrics

Measure	SyncBT	ABT	SyncBTU	ABTU
Privacy Loss	2,5	9,0	1,8	5,3
Messages	2,8	531,3	2,3	150,6
Solving(%)	30	20	30	20
Interruption(%)	—	—	30	70
CPU Time(ms)	258	1329	255	910

a) Discussion on Experiments: For each algorithm, we have measured in Table I the privacy loss, the number of exchanged messages, the number of problems solved, the number of solvings interrupted to preserve privacy and CPU time. Figure 3 shows that synchronous algorithms are better than asynchronous ones at preserving privacy. Moreover, SyncBTU and ABTU are better than SyncBT and ABT at preserving privacy, respectively.

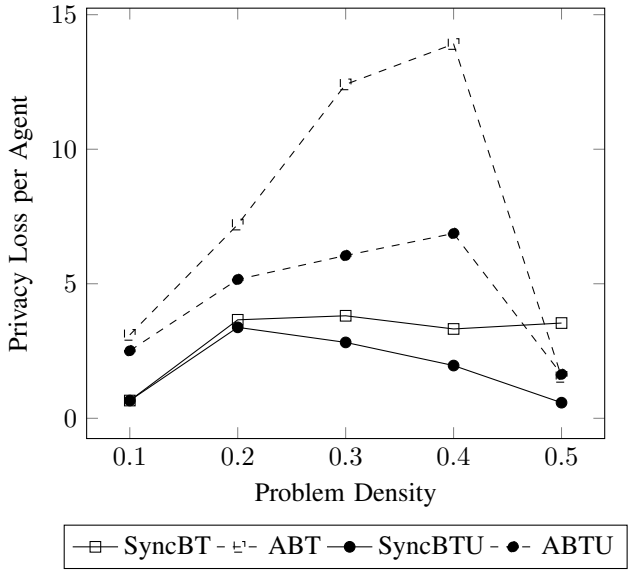

Fig. 3. Evaluation of privacy loss on different random problems

We notice in Table I that interrupting the solvings to preserve privacy lets agents not only reduce privacy loss by 39 % but also reduces the calculation times by 27 %. It reduces the number of messages exchanged by 71 % while still solving 98 % of the problems solved by standard algorithms, since the interruptions happen mostly when the problems have no solution.

VI. Conclusions

While many frameworks have been developed recently for coping with privacy in distributed problem solving, none of them is widely used, likely due to the difficulty in modeling common problems. In this article we propose a framework called Utilitarian Distributed Constraint Satisfaction Problem (UDisCSP). It models the privacy loss for the revelation of an agent's constraints as a utility function. We then use this utilitarian framework to cast the DisCSP into a planning

problem for utility-based agents. The individual problem of each agent can be seen as having similarities to a Partially Observable Markov Decision Process (POMDP), where the objective is now to decide for the best action to perform at each state of the problem, the available actions being the communications available in the chosen DisCSP solver, and inferences based on the probabilities being learned from previous solvings. We then show how adapted synchronous and asynchronous protocols (SyncBTU and ABTU) behave and compare them on different distributed meeting scheduling problems. The comparison shows that SyncBTU can maintain more privacy on random problems, as compared to both ABTU and original SyncBT and ABT.

Acknowledgements: Authors gratefully acknowledge the reviewers for their constructive remarks, allowing to improve our article. This work was realized while the first author was a visiting researcher at Florida Institute of Technology.

REFERENCES

[1] M. Yokoo, E. Durfee, T. Ishida, and K. Kuwabara, "The distributed constraint satisfaction problem: Formalization and algorithms," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 10, no. 5, pp. 673–685, 1998.

[2] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 ed., 2003.

[3] R. Maheswaran, M. Tambe, E. Bowring, J. Pearce, and P. Varakantham, "Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pp. 310–317, IEEE Computer Society, 2004.

[4] A. Gershman, A. Grubshtein, A. Meisels, L. Rokach, and R. Zivan, "Scheduling meetings by agents," in *Proc. 7th International Conference on Practice and Theory of Automated Timetabling (PATAT 2008). Montreal (August 2008)*, 2008.

[5] D. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, "The complexity of decentralized control of markov decision processes," *Mathematics of operations research*, vol. 27, no. 4, pp. 819–840, 2002.

[6] J. Messias, M. Spaan, and P. Lima, "Asynchronous execution in multiagent POMDPs: Reasoning over partially-observable events," in *AAMAS*, vol. 13, pp. 9–14, 2013.

[7] P. Gmytrasiewicz and P. Doshi, "A framework for sequential planning in multi-agent settings," *Journal of Artificial Intelligence Research*, pp. 49–79, 2005.

[8] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo, "Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs," in *AAAI*, vol. 5, pp. 133–139, 2005.

[9] C. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

[10] M. Yokoo, T. Ishida, E. Durfee, and K. Kuwabara, "Distributed constraint satisfaction for formalizing distributed problem solving," in *Distributed Computing Systems, 1992., Proceedings of the 12th International Conference on*, pp. 614–621, IEEE, 1992.

[11] R. Zivan and A. Meisels, "Synchronous vs asynchronous search on DisCSPs," in *Proceedings of the First European Workshop on Multi-Agent Systems (EUMA)*, 2003.

[12] Y. Krupa and L. Vercouter, "Handling privacy as contextual integrity in decentralized virtual communities: The privacias framework," *Web Intelligence and Agent Systems*, vol. 10, no. 1, pp. 105–116, 2012.

[13] E. Freuder, M. Minca, and R. Wallace, "Privacy/efficiency tradeoffs in distributed meeting scheduling by constraint-based agents," in *Proc. IJCAI DCR*, pp. 63–72, 2001.

[14] L. Crépin, Y. Demazeau, O. Boissier, and F. Jacquenet, "Privacy preservation in a decentralized calendar system," in *7th International Conference on Practical Applications of Agents and Multi-Agent Systems, PAAMS 2009, Salamanca, Spain, 25-27 March 2009* (Y. Demazeau, J. Pavón, J. Corchado, and J. Bajo, eds.), vol. 55 of *Advances in Intelligent and Soft Computing*, pp. 529–537, Springer, 2009.

[15] B. Faltings, T. Léauté, and A. Petcu, "Privacy guarantees through distributed constraint satisfaction," in *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT'08. IEEE/WIC/ACM International Conference on*, vol. 2, pp. 350–358, IEEE, 2008.

[16] M. Yokoo, K. Suzuki, and K. Hirayama, "Secure distributed constraint satisfaction: Reaching agreement without revealing private information," in *Principles and Practice of Constraint Programming-CP 2002*, pp. 387–401, Springer, 2002.

[17] M. Hirt, U. Maurer, and B. Przydatek, "Efficient secure multi-party computation," in *Advances in Cryptology-ASIACRYPT 2000*, pp. 143–161, Springer, 2000.

[18] M.-C. Silaghi, "Meeting scheduling system guaranteeing n/2-privacy and resistant to statistical analysis (applicable to any DisCSP)," in *WI*, pp. 711–715, 2004.

[19] R. Greenstadt, J. Pearce, and M. Tambe, "Analysis of privacy loss in distributed constraint optimization," in *AAAI*, vol. 6, pp. 647–653, 2006.

[20] M. Silaghi and B. Faltings, "A comparison of distributed constraint satisfaction techniques with respect to privacy," in *Proceedings of the Distributed Constraint Reasoning Workshop at AAMAS*, Citeseer, 2002.

[21] R. Maheswaran, J. Pearce, P. Varakantham, E. Bowring, and M. Tambe, "Valuations of possible states (VPS): a quantitative framework for analysis of privacy loss among collaborative personal assistant agents," in *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pp. 1030–1037, ACM, 2005.

[22] R. Maheswaran, J. Pearce, E. Bowring, P. Varakantham, and M. Tambe, "Privacy loss in distributed constraint reasoning: A quantitative framework for analysis and its applications," *Autonomous Agents and Multi-Agent Systems*, vol. 13, no. 1, pp. 27–60, 2006.

[23] R. Wallace and E. Freuder, "Constraint-based multi-agent meeting scheduling: Effects of agent heterogeneity on performance and privacy loss," in *DCR*, pp. 176–182, 2002.

[24] I. Brito, A. Meisels, P. Meseguer, and R. Zivan, "Distributed constraint satisfaction with partially known constraints," *Constraints*, vol. 14, no. 2, pp. 199–234, 2009.

[25] S. Koenig, D. Furcy, and C. Bauer, "Heuristic search-based replanning.," in *AIPS*, pp. 294–301, 2002.

[26] J. Savaux, J. Vion, S. Piechowiak, R. Mandiau, T. Matsui, K. Hirayama, M. Yokoo, and M. Silaghi, "Privacité dans les DisCSP pour agents utilitaires (to appear)," in *JFSMA 16 - Vingt-quatrièmes Journées Francophones sur les Systèmes Multi-Agents, Saint Martin du Vivier (Rouen), France, October 5th-October 7st, 2016*, 2016.

[27] A. Arbelaez, Y. Hamadi, and M. Sebag, "Continuous search in constraint programming," in *Autonomous Search*, pp. 219–243, Springer, 2011.

[28] F. Bacchus and P. Van Beek, "On the conversion between non-binary and binary constraint satisfaction problems," in *National Conference on Artificial Intelligence*, 1998.

[29] I. Brito and P. Meseguer, "Distributed forward checking," in *Principles and Practice of Constraint Programming–CP 2003*, pp. 801–806, Springer, 2003.

[30] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings, "Asynchronous search with aggregations," in *AAAI*, pp. 917–922, 2000.

[31] F. Ferdinando, W. Yeoh, and E. Pontelli, "Multi-variable agents decomposition for DCOPs to exploit multi-level parallelism," in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pp. 1823–1824, International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[32] R. Mandiau, J. Vion, S. Piechowiak, and P. Monier, "Multi-variable distributed backtracking with sessions," *Appl. Intell.*, vol. 41, no. 3, pp. 736–758, 2014.

[33] I. Brito and P. Meseguer, "Distributed forward checking," in *CP*, 2003.

[34] R. Wallace and E. Freuder, "Constraint-based reasoning and privacy/efficiency tradeoffs in multi-agent problem solving," *Artificial Intelligence*, vol. 161, no. 1, pp. 209–227, 2005.