Privacité dans les DisCSP pour agents utilitaires

Julien Savaux[a] Julien Vion[a] Sylvain Piechowiak[a] René Mandiau[a]

Toshihiro Matsui[b] Katsutoshi Hirayama[c] Makoto Yokoo[d] Marius Silaghi[e]

[a]LAMIH UMR CNRS 8201, Université de Valenciennes et du Hainaut-Cambrésis, France

[b]Nagoya Institute of Technology, Japon

[c]Kobe Naval University, Japon

[d]Kyushu University, Japon

[e]Florida Institute of Technology, USA

Résumé

La privacité a toujours été une motivation majeure pour la résolution de problèmes distribués, et particulièrement dans les CSP distribués (DisCSP). Cependant, même si plusieurs mesures ont été proposées pour la quantifier, aucune d'entre elles n'est largement utilisée. Dans ce papier, nous abordons le problème en supposant que les agents peuvent fonder leur raisonnement sur la base des utilités. Dans ce cadre, nous proposons un modèle de DisCSP basé sur des utilités, appelé UDisCSP. L'utilité de chaque état est traitée comme étant un compromis entre l'utilité de préserver la privacité et celle de parvenir à un accord sur les affectations des variables partagées.

Mots-clés : *Privacité, Problèmes de Satisfaction de Contraintes Distribués, Agents Utilitaires*

Abstract

Privacy has traditionally been a major motivation for distributed problem solving, and particularly for Distributed CSP (DisCSP). However, even though several metrics have been proposed to quantify it, none of them is widely used. In this paper, we approach the problem by assuming that computation is performed among utility-based agents. In this context, we introduce a DisCSP model based on the utility measure, called Utilitarian Distributed Constraint Satisfaction Problems (UDisCSP). The utility of each state is estimated as the summation between the utility of preserved privacy and the utility of reaching an agreement on assignments for shared variables.

Keywords: *Privacy, Distributed Constraint Satisfaction Problems, Utilitarian Agents*

1 Introduction

Dans les problèmes de satisfaction de contraintes distribués (DisCSP), les agents doivent trouver des affectations cohérentes à un ensemble de variables partagées, en respectant des contraintes données. Pour trouver ces affectations, les agents échangent des messages jusqu'à ce qu'une solution soit trouvée ou jusqu'à ce qu'un agent détecte qu'il n'y a pas de solution au problème. Ainsi, au travers des messages échangés, les agents dévoilent des informations au cours du processus de recherche de solution. Cette perte de privacité sur les informations échangées est une préoccupation importante dans les DisCSP [19]. C'est précisément sur le contrôle de la privacité abordée sous l'angle de l'utilité dans les DisCSP que se focalise ce papier.

L'hypothèse habituelle est que les agents utilitaires sont en mesure d'associer à chaque état une valeur d'utilité [14]. Si la préoccupation des agents est de préserver au mieux la privacité, on peut associer une valeur d'utilité à la privacité de chaque information dans la définition des problèmes locaux. Chaque agent doit également être en mesure de quantifier l'utilité pour l'obtention de la solution globale. Dans notre approche, nous abordons le problème en supposant que la privacité a une utilité qui peut être cumulée avec la valeur d'utilité pour résoudre le problème. Nous évaluons la perte de privacité des agents au cours du processus de résolution d'un problème en fonction de l'utilité totale

pour toutes les informations qui ont été révélées. La présence ou l'absence d'une valeur dans le domaine d'une de ses variables est typiquement l'information qu'un agent souhaite garder privée. L'affectation d'une valeur à l'une de ses variables sera associée à un coût qui caractérise le désir de cet agent de conserver cette information privée. Dans les algorithmes existants, les agents participent au processus de recherche jusqu'à ce qu'un accord soit trouvé. Dans cet article, nous considérons qu'un agent puisse décider de ne plus poursuivre sa contribution à la recherche d'une solution globale s'il estime que l'utilité de la privacité qu'il risque de perdre est plus élevée que la récompense qu'il pourrait gagner. Nous proposons donc des extensions aux algorithmes existants afin d'exploiter le modèle utilitaire de la privacité (appelé UDisCSP).

Le papier est organisé comme suit. Après une partie introductive, la partie 2 dresse un état de l'art concernant la privacité pour les algorithmes de DisCSP et présente les limites des approches existantes. La partie 3 introduit le concept de UDisCSP. Après une discussion sur les implications théoriques, nous présentons dans la partie 4 les résultats expérimentaux obtenus sur des problèmes distribués de gestion de réunion entre plusieurs participants. Enfin, nous concluons le papier dans une partie 5.

2 État de l'art

2.1 DisCSP

Le formalisme DisCSP est couramment utilisé pour modéliser des problèmes de satisfaction de contraintes distribués entre plusieurs agents. Il est représenté par un quadruplet $\langle A, V, D, C\rangle$ où :

- A : un ensemble d'agents
- V : un ensemble de variables, chacune d'elles appartient à un seul agent
- D : un ensemble de domaines, chacun définit les valeurs possibles pour la variable correspondante
- C : un ensemble de contraintes, chaque contrainte contient un ensemble d'affectations interdites entre deux variables (par ex., $x_1 \neq x_2$).

La révélation d'une affectation d'un agent à un autre agent a un coût. Une fois l'affectation révélée, nous considérons qu'elle devient publique, par conséquent la révélation d'une affectation à un ou plusieurs agents possède le même coût. Le nombre d'agents qui connaissent la même affectation n'influence pas la mesure de sa privacité.

2.2 Algorithmes de Résolution

Les algorithmes de DisCSP se distinguent classiquement selon que le type de backtracking effectué est synchrone ou asynchrone.

Backtracking Synchrone L'algorithme de base pour les DisCSP est *Backtracking Synchrone* (SyncBT) [20, 22]. SyncBT est une distribution de l'algorithme de backtracking standard. Les agents commencent par déterminer une hiérarchie entre eux. Ensuite, chaque agent de priorité plus élevée envoie une affectation consistante de la variable à l'agent suivant avec un message `ok?`. Le destinataire y ajoute l'affectation de sa variable en respectant ses contraintes, et continue de même. Si un agent est incapable de trouver une affectation compatible avec la solution partielle actuelle qu'il a reçu, il envoie un message `nogood` à l'agent précédent dans la hiérarchie. Le processus se répète jusqu'à l'obtention d'une solution globale s'il en existe une ou jusqu'à l'exploration complète de l'espace de recherche. La principale critique réside dans la séquentialité des messages échangés, qui implique une durée de résolution importante.

Backtracking Asynchrone Le Backtracking Asynchrone (ABT) [20], permet aux agents une exécution simultanée. Chaque agent trouve une instanciation de sa variable et la communique aux autres agents, ayant des contraintes impliquant cette variable. Les agents attendent ensuite les messages entrants. Ils reçoivent des messages `ok?` contenant des affectations de la part des agents de priorité supérieure au début de la résolution et à chaque fois que ces agents modifient leurs affectations afin d'éviter une violation de contrainte. Un agent reçoit finalement les valeurs proposées par les agents auxquels il est connecté par des liens entrants. Ces valeurs constituent un contexte appelé `agent view`. Quand un agent reçoit un message `ok?`, il intègre la solution reçue dans son `agent view` et vérifie si sa solution est compatible avec la solution reçue. Si ce n'est pas

le cas, l'affectation de l'agent est modifiée. Si un agent peut déduire un nogood à partir de ses contraintes et de son `agent view`, la solution de l'agent de plus faible priorité impliqué dans le nogood doit être changée. Un message `nogood` est considéré par son destinataire comme une nouvelle contrainte et peut l'amener à changer de solution et de générer les messages `ok?`, `addlink` ou `nogood` correspondant. Un message `addlink` permet de mettre à jour la liste des voisins du récepteur lorsque celui-ci partage une contrainte avec un agent qui lui était étranger avant la création du nogood.

2.3 Privacité

La privacité (c-à-d. la volonté d'un agent de préserver ses informations confidentielles) est un problème crucial dans de nombreuses applications. Par exemple, lorsque les utilisateurs échangent des informations (plus ou moins confidentielles) via des réseaux sociaux [10]. Un autre exemple dans la gestion d'agenda distribué soulève également des problèmes de confidentialité des informations échangées. En effet, nous savons que l'affectation de créneaux horaires pour des réunions peut s'avérer difficile *a priori* si les participants ne souhaitent pas révéler leur contraintes [3]. Ces décisions coordonnées sont souvent difficiles à mettre en œuvre, d'où la nécessité de maintenir les contraintes privées [5]. Ainsi, la privacité est un aspect important pour les algorithmes de résolution DisCSP. Dans les travaux existants, on peut distinguer deux approches principales pour prendre en compte la privacité. La première utilise des techniques cryptographiques. Le principal problème de ces méthodes est que les protocoles cryptographiques peuvent être beaucoup plus lents, ce qui les rendent souvent impraticables [16]. La seconde approche se base sur l'utilisation de différentes stratégies de recherche pour évaluer la perte de la privacité.

Cryptographie L'approche utilisant la cryptographie décrite dans [21] permet le maintien d'un niveau élevé de privacité. Elle donne plus d'importance en termes de calcul à la privacité qu'à la résolution du problème. Dans cette approche, trois serveurs coopèrent pour trouver une solution cryptée. Cette méthode garantit qu'aucune information d'un agent n'est divulguée à d'autres agents. Lorsque restreinte à une seule variable globale comme dans [16], cette méthode garantit qu'aucune information n'est révélée aux autres agents, excepté ce qui peut être inféré depuis la valeur de la solution acceptée.

Les méthodes suivantes n'utilisent pas de système externe de cryptage pour garantir la privacité.

DisPrivCSP (*Distributed Private Constraint Satisfaction Problems*) modélise la perte de privacité liée aux révélations individuelles [7, 15]. Chaque agent paie un coût lorsqu'il révèle à un autre agent, via un message `ok?` ou `nogood`, si une affectation donnée satisfait ou non son problème local. La récompense pour la solution du problème est donnée comme une constante. Les critères de privacité et de coûts/utilité d'optimisation habituelles ont été combinés en un unique critère [4].

VPS (*Valuation of Possible States*) estime la perte de privacité comme étant la diminution du nombre d'états possibles dans lesquels un agent peut se trouver d'après ses voisins [11]. Au cours du processus de recherche, les agents proposent leurs valeurs dans un ordre décroissant de préférence. A la fin du processus de recherche, la différence entre l'ordre présupposé de préférences et celui observé par les agents voisins lors de la recherche détermine la perte de privacité.

PKC (*Partially Known Constraints*) utilise l'entropie comme définie en théorie de l'information pour mesurer la privacité [2]. Dans cette méthode, deux variables x_1 et x_2 appartenant à deux agents différents peuvent partager une contrainte. Cependant, tous les couples d'affectation interdits ($x_1 = v_1, x_2 = v_2$) ne sont pas connus par les deux agents. Chaque agent ne connaît qu'un sous-ensemble des contraintes. Au cours du processus de recherche, la privacité d'une affectation est perdue à travers les messages `ok?` et `nogood`, comme dans les algorithmes standards.

2.4 Approches Existantes

Exemple 1 *Supposons qu'un professeur et deux étudiants cherchent à se réunir. Ils cherchent tous les trois à déterminer un créneau horaire parmi trois possibles (*8 *h,* 10 *h*

et 14 *h). Nous savons également que Professeur* A_1 *est indisponible à* 14 *h, Étudiant* A_2 *est indisponible à* 10 *h, et Étudiant* A_3 *est indisponible à* 8 *h. Les informations peuvent être confidentielles pour plusieurs raisons. Par exemple, Étudiant* A_2 *ne veut pas révéler son indisponibilité à* 10*h (parce qu'il a pris secrètement un deuxième emploi à cette période). La valeur que Étudiant* A_2 *ne souhaite pas révéler son indisponibilité à* 10 *h est le salaire de son deuxième emploi* (\$2 000)*. L'intérêt de trouver un accord pour chaque étudiant est l'obtention de la bourse pour leurs études* (\$5 000)*. Celui du professeur est de recevoir une prime de* (\$5 000)*. En outre Étudiant* A_3 *s'est récemment vanté à Étudiant* A_2 *qu'à* 8 *h il sera convié à un entretien d'embauche, et il préfère payer une certaine somme* (\$1 000) *que de révéler que ce n'est pas le cas. De même, les participants associent un coût à la révélation de chaque disponibilité et indisponibilité Ainsi, les agents associent un coût de* 1 *à la révélation de leur disponibilité à* 8 *h, un coût de* 2 *à celle de* 10 *h, et de* 4 *à celle de* 14 *h. La récompense pour trouver une solution est de* 4 *pour Professeur* A_1 *et de* 5 *pour Étudiant* A_2*et Étudiant* A_3*.*

DisCSP Cet exemple peut être modélisé sous forme d'un DisCSP. Par souci de simplicité, dans les sections suivantes, nous référencerons ces valeurs possibles par leur identifiant : 1, 2 et 3.

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

Comme on peut le constater, le formalisme initial d'un DisCSP ne prend pas directement en compte les considérations de privacité. Nous allons montrer comment les travaux existants de ce modèle peuvent permettre de les intégrer.

DisPrivCSP Dans DisPrivCSP, les paramètres supplémentaires sont P, pour spécifier le coefficient de privacité de chaque valeur, et R, pour préciser les récompenses de chaque coefficient. Nous pouvons obtenir pour notre exemple :

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

DisPrivCSP modélise efficacement les informations relatives à la privacité décrites dans le problème initial et mesure la perte de privacité lors de l'exécution d'algorithmes de résolution existants. Cependant, ces nouvelles informations ne sont pas utilisées afin de modifier la recherche de solutions. Des extensions aux algorithmes existants pourraient permettre de réduire la perte de privacité.

VPS Dans VPS, les trois participants doivent supposer un ordre de préférence parmi les différentes valeurs possibles. Comme initialement les agents ne savent rien à propos des autres agents, excepté la variable partagée par la contrainte. Ils supposent une répartition égale de toutes les valeurs possibles pour tous les autres agents. Ceci signifie qu'ils n'envisagent pas qu'une valeur puisse être moins secrète, et donc proposée en premier. Dans ce sens il faudrait étendre VPS pour être en mesure de modéliser aussi le type de privacité, que nous avons illustré par notre exemple.

PKC Dans PKC, chaque participant ne connait que ses indisponibilités personnelles. Seule l'union des informations connues par deux agents sur une contrainte donnée permet de reconstruire l'ensemble du problème. Pour modéliser notre exemple avec PKC, les contraintes sont distribuées entre les agents. A chaque sous-ensemble de contraintes correspond un agent, comme suit :

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

3 Concepts

3.1 Notre Approche : UDisCSP

On peut remarquer que les récompenses et les coûts dans notre problème possèdent des similitudes avec les utilités et les récompenses souvent manipulées par des algorithmes de planification [9]. En tant que tel, nous proposons de définir un cadre qui, tout en étant potentiellement équivalent dans sa représentation aux extensions de DisCSP existants, pourrait néanmoins spécifier explicitement les éléments de la famille correspondants aux problèmes de planification.

Nous introduisons le problème de satisfaction de contraintes distribués utilitaire (UDisCSP). Contrairement aux approches discutées précédemment, nous nous sommes également intéressés au processus de résolution et pas seulement à la gestion de la privacité.

Définition 1 *Un UDisCSP est défini par* $\langle A, V, D, C, U, R\rangle$ *où :*

- $A = \{A_1, \ldots, A_n\}$ *est un ensemble de* n *agents*
- $V = \{x_1, \ldots, x_n\}$ *est un ensemble de* n *variables. Chaque agent* A_i *gère une seule variable* x_i*.*
- $D = \{D_1, \ldots, D_n\}$ *où* $D_i \subseteq \{1, \ldots, d\}$ *est le domaine de la variable* x_i*, connue uniquement par* A_i*.*
- $C = \{C_1, \ldots, C_m\}$ *est un ensemble de contraintes inter-agent.*
- $U = \{u_{1,1}, \ldots, u_{n,d}\}$ *est une matrice de coûts où* $u_{i,j}$ *est le coût pour l'agent* A_i *pour révéler que* $j \in D_i$*.*
- $R = \{r_1, \ldots, r_n\}$ *est un ensemble de récompenses, où* r_i *est la récompense que l'agent* A_i *reçoit si un accord est trouvé.*

L'état de l'agent A_i *considère également le sous-ensemble* D_i *qui est révélé, ainsi que l'accomplissement d'un accord. Le problème est de définir un ensemble d'actions de communication et une politique pour chaque agent de façon à maximiser leur utilité.*

Comme DisPrivCSP, UDisCSP considère les coûts relatifs à la privacité. Cependant, DisPrivCSP utilise uniquement ces coûts pour évaluer la perte de privacité lors de l'exécution d'algorithmes de résolution existants. Seul UDisCSP utilise ces coûts pour permettre aux agents de modifier la recherche de solution afin de diminuer la perte de privacité.

Les participants sont des agents dotés d'un modèle basé sur des utilités [14]. Ces derniers ont pour objectif d'atteindre un état optimal. Nous définissons alors un *accord* comme un ensemble d'affectations pour toutes les variables avec des valeurs de leur domaine, de sorte que toutes les contraintes sont satisfaites. Notons qu'une solution pour un UDisCSP ne suppose pas nécessairement de satisfaire toutes les contraintes.

Exemple 2 *Le DisCSP de l'exemple 1 est étendu en un UDisCSP en spécifiant les paramètres additionnels suivants :*
$U = \{u_{1,1} = 1, u_{1,2} = 2, u_{1,3} = 4,$
$u_{2,1} = 1, u_{2,2} = 2, u_{2,3} = 4,$
$u_{3,1} = 1, u_{3,2} = 2, u_{3,3} = 4\}.$
$R = \langle 4, 5, 5\rangle.$

3.2 Algorithmes

Nous allons maintenant présenter la façon dont les algorithmes de base ABT et SyncBT sont adaptés pour les UDisCSP. L'état d'un agent comprend son `agent view`. Après chaque changement d'état, chaque agent calcule l'utilité estimée de l'état atteint par chaque action possible, et sélectionne l'une des actions conduisant au prochain état dont l'estimation de l'utilité est maximale.

Dans nos algorithmes, une information utilisée par les agents dans leur estimation des utilités attendues est le risque qu'une de leurs affectations soit rejetée. Ce risque pourrait être réévalué à tout moment en fonction des données enregistrées lors de résolutions précédentes sur des problèmes de même dureté (c.-à-d. ayant la même proportion d'instanciations interdites).

L'évaluation de ce risque peut être en ligne ou hors ligne. Pour l'évaluation hors ligne, l'agent calcule le nombre de messages `ok?` et `nogood` envoyés au cours des exécutions précédentes, appelée *count*. L'agent évalue également le nombre de messages envoyés précédemment conduisant à la terminaison de l'algorithme, dans la variable *agreementCount*. Il calcule le risque qu'une révélation conduise à une solution, appelé *agreementProb* (équation 1). Pour l'évaluation en ligne, les variables *count*, *agreementCount* et *agreementProb* sont mises à jour chaque fois que les événements correspondants se produisent.

$$agreementProb = \frac{agreementCount}{count} \quad (1)$$

Lorsque les messages `ok?` sont envoyés, l'agent a le choix de l'affectation à proposer. Lorsque l'envoi d'un message `nogood` est prévu, les agents ont également le choix sur la façon de l'exprimer. Avant chaque message `ok?` ou `nogood`, les agents vérifient quelle action disponible conduit à l'utilité attendue la plus élevée. Si l'utilité espérée est inférieure

à celle en cours, l'agent annonce un échec. Le résultat est utilisé pour décider de la nouvelle valeur à affecter, du nogood à transmettre, ou de l'interruption de la résolution.

Nous avons appelé ces algorithmes modifiés SyncBTU et ABTU, respectivement. Ces algorithmes sont obtenus en effectuant les modifications mentionnées ci-dessus, dans les pseudo-codes des algorithmes SyncBT [20, 19, 22] et ABT [20, 19, 22]. ABTU dérive de ABT en modifiant les trois procédures `checkAgentView`, `when_nogood` et `backtrack`. Par exemple, la nouvelle procédure `checkAgentView` (voir Algorithme 1) est obtenue en insérant les lignes 8 à 10. Elles testent s'il y a perte de la privacité. La procédure ne se poursuit normalement que si la perte attendue est plus faible que le gain espéré. Par manque de place, nous ne détaillerons pas les versions modifiées des autres procédures de ABTU, car elles sont obtenues de la même manière à partir des procédures de ABT [20, 19]. Les procédures de SyncBT sont modifiées de façon similaire pour obtenir SyncBTU. Nous apportons les modifications au pseudo-code présenté dans [22], aux fonctions `assignCPA`, et `backtrack`.

Algorithm 1: checkAgentView dans 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
```

Pour calculer l'utilité estimée de poursuivre un accord (révélant une alternative), l'agent tient compte de tous les différents scénarios possibles des sous-ensembles de valeurs qui pourraient être révélés dans l'avenir en se basant sur les rejets possibles qu'il a reçus, avec leur probabilité comme le précise l'algorithme 2. Celui-ci prend comme paramètres : (i) $agreementProb$ calculé précédemment (équation 1), (ii) les valeurs possibles D, et (iii) la probabilité de sélectionner une valeur dans D.

L'algorithme calcule ensuite de manière récursive l'utilité des prochains états possibles. Il détermine également si la révélation de la valeur actuelle v conduit à la fin de l'algorithme, les valeurs stockées dans les variables $costRound$ et $costNonTerminal$. L'algorithme renvoie le coût estimé de la perte de privacité pour les futurs états possibles, appelé $estimatedCost$.

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;
```

Exemple 3 *En reprenant l'exemple 1, au début de la résolution, l'agent A_1 doit décider d'une première action à effectuer. Pour décider s'il doit proposer une valeur disponible ou non, il calcule le coût estimé (estimatedCost) correspondant en appelant l'algorithme 2 avec les paramètres : $agreementProb = 0,5$, l'ensemble des messages possibles ($D = \{1, 2, 3\}$) et $probD = 1$.*

Pour chaque valeur possible, cet algorithme calcule de façon récursive le coût pour les deux scénarios correspondants ; à savoir si l'action conduit immédiatement à un accord, ou non. Trois valeurs sont disponibles pour ce problème : 1, 2, *et* 3. *La valeur de estimatedCost retournée est la somme des coûts de chaque valeur, pondérés par la probabilité que leur faisabilité soit révélée dans l'un des scénarios possibles. Cet algorithme équivaut à calculer le coût total pour chaque scénario (la somme des coûts des informa-*

tions qui sont révélées jusqu'à la terminaison de l'algorithme) pondéré par la probabilité que ce scénario se réalise.

Avec cette méthode, présentant plus clairement le calcul du point de vue d'un agent planificateur, trois scénarios sont possibles : L'algorithme peut prendre fin après la proposition des valeurs $\{1\}, \{1,2\}$ *ou* $\{1,2,3\}$*. Une fois la valeur 1 proposée, la probabilité que la valeur suivante (2) doive être proposée est de* $0,5$*. De même, la probabilité que la dernière valeur doive être proposée est de* $(0,5 \times 0,5)$*, soit* $0,25$*. La valeur de estimatedCost obtenue est* $(u_{1,1} \times 0,5 + (u_{1,1} + u_{1,2}) \times 0,25 + (u_{1,1} + u_{1,2} + u_{1,3}) \times 0,25)$*, soit 3.*

La valeur de l'utilité espérée est égale à $(récompense - estimatedCost)$*, soit* $4-2=1$*. Cette utilité étant positive, la première valeur est proposée.*

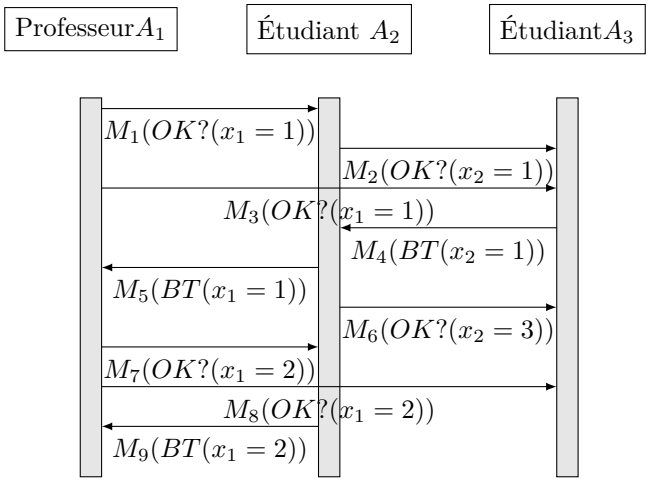

FIGURE 1 – Interactions entre les agents pendant l'exécution de ABT

Exemple 4 *La Figure 1 illustre les échanges de messages entre les agents pour l'algorithme ABT. Elle montre que Étudiant* A_2 *propose* $x_2 = 1$ *dans le message* M_2 *et* $x_2 = 3$ *dans* M_6*. Dans ce cas, la perte de la privacité pour Étudiant* A_2 *est* $u_{2,1} + u_{2,3} = 1 + 4 = 5$*. Cependant, avec ABTU, nous utilisons non seulement l'utilité réelle de l'affectation à être révélée, mais estimons la perte de la privacité en utilisant l'Algorithme 2. Après que Étudiant* A_2 *a envoyé* $x_2 = 1$ *avec* M_2*, il considère l'envoi de* $x_2 = 3$ *avec* M_6*. Si la valeur suivante,* $14h$*, est acceptée, Étudiant* A_2 *atteindra l'état final tout en ayant révélé* $x_2 = 1$ *et* $x_2 = 3$*, pour un coût total de la privacité de* $u_{2,1} + u_{2,3} = 1 + 4 = 5$*. Dans le cas contraire, la non-disponibilité de la dernière valeur* $x_2 = 2$ *devra être révélée afin de poursuivre le processus de recherche, conduisant ainsi à la révélation de l'ensemble de ses affectations pour un coût total de* 7*. Étant donné que ces deux scénarios ont une probabilité de* 50% *de se produire,* $estimatedCost$ *est égal à* $(5 + 7) \times 0,5 = 6$*. L'utilité de* $(récompense - estimatedCost)$ *étant égal à* $5 - 6 = -1$*, Étudiant* A_2 *n'a aucun intérêt à révéler* $x_2 = 3$ *et interrompt la résolution. Sa perte de privacité finale est seulement* $u_{2,1} = 2$*. L'utilité de l'état final obtenue par Étudiant* A_2 *étant de* -2 *avec ABTU et de* -4 *avec ABT, ABTU conserve donc plus les informations privées que ABT pour ce problème.*

Discussion Théorique Le modèle décrit dans UDisCSP peut supposer (sans perte significative de généralité) que les contraintes inter-agent sont publiques. Cela est dû au fait que tout problème de contraintes privées inter-agent (par exemple, PKC), est équivalent à sa représentation duale dans laquelle chaque contrainte devient une variable [1].

En outre, l'hypothèse que chaque agent possède une seule variable n'est pas non plus restrictive. En effet, il est possible de transformer un ensemble de variables en une seule dont le domaine est le produit cartésien. Néanmoins, certains algorithmes peuvent exploiter ces structures sous-jacentes qui ont fait l'objet d'autres travaux de recherches [17, 6, 13].

Proposition 1 *Les UDisCSP sont plus généraux que les DisCSP.*

Preuve 1 *Un DisCSP peut être modélisé comme un UDisCSP avec tous les coûts de la privacité égaux à* 0*. Les UDisCSP obtenus pourraient toujours parvenir à un accord, si c'est possible. Par conséquent, l'objectif d'un UDisCSP coïnciderait également avec l'objectif du DisCSP équivalent.*

L'espace mémoire nécessaire pour ABTU et SyncBTU pour chaque agent est du même ordre de grandeur que pour ABT et SyncBT. L'espace supplémentaire provient (i) du stockage des coûts de privacité associés aux

valeurs des domaines et (ii) des variables *agreementProb*, *agreementCount*, *count* et r_i qui nécessitent une taille constante.

4 Résultats Expérimentaux

Nous évaluons notre approche sur des instances générées aléatoirement de problèmes distribués de planification de réunion (DMS pour *Distributed Meeting Scheduling*) [12, 8]. Dans ces problèmes, tous les agents possèdent une variable, correspondant à la réunion à planifier, et le domaine est le même pour toutes les variables. Un ensemble de contraintes imposent à chaque couple de variables d'être égales, auxquelles s'ajoute une contrainte unaire supplémentaire pour chaque agent.

Des travaux antérieurs [18] sur les DisCSP ont déjà abordé la question de la privacité dans les problèmes de planification distribués de réunion. Ces travaux considèrent la disponibilité d'un agent à une heure donnée comme étant confidentielle. La perte de privacité est alors évaluée comme étant la différence entre les cardinalités du dernier et du premier ensemble de disponibilités possibles pour un participant. Comme les différentes distributions de contraintes unaires peuvent avoir un impact important sur la privacité, nous proposons de générer deux types de problèmes aléatoires différents :

Uniform où les contraintes unaires sont distribuées uniformément entre les agents

Tail-constrained où les $n/2$ agents de plus haute priorité ont une probabilité 3 fois plus petite de recevoir une contrainte unaire que les $n/2$ agents de plus basse priorité, même si la dureté globale reste la même

Exemple 5 *Supposons un problème où les deux agents de plus basse priorité ont des ensembles disjoints de disponibilités, ce qui signifie que ces agents peuvent détecter à eux seuls que le problème n'a pas de solution. ABTU permet à ces agents d'échanger des messages depuis le début du processus de recherche et donc de l'interrompre rapidement. Par contre, SyncBTU les empêche d'échanger des messages avant que tous les agents plus prioritaires ont construit une solution partielle. Ainsi, SyncBTU nécessite plus de messages échangés et donc une perte de privacité plus élevée que pour ABTU.*

Table 1 – Comparaison des algorithmes

Mesure	SyncBT	ABT	SyncBTU	ABTU
Perte de Privacité	2,5	9,0	1,8	5,3
Messages	2,8	531,3	2,3	150,6
Résolution(%)	30	20	30	20
Interruption(%)	—	—	30	70
Temps CPU(ms)	258	1329	255	910

Un problème est généré en trois étapes :

1. création des différentes variables (une par « agent participant ») et initialisation de leurs domaines (« horaires possibles »)
2. ajout des contraintes d'égalité de tous les couples de variables et ajout des contraintes unaires (respect de la contrainte de dureté)
3. génération d'un coût de révélation uniformément distribué entre 0 et 9, pour chaque valeur de chaque variable.

Les expériences sont réalisées sur un ordinateur sous Windows 7, avec un processeur 2, 16 GHz et 4 Go de mémoire vive. Les problèmes sont paramétrés comme suit : 10 agents, 10 valeurs possibles, l'utilité d'une révélation est un nombre aléatoire compris entre 0 et 9, la récompense pour trouver une solution au problème est de 20. Chaque série d'expérimentations est une estimation moyenne de 50 instances pour les différents algorithmes (c-à-d. SyncBT, ABT, SyncBTU et ABTU).

Discussion sur les expérimentations Nous avons mesuré pour chacun de ces algorithmes, la perte de privacité, le nombre de messages échangés, le nombre de problèmes résolus, le nombre de résolutions interrompues pour préserver la privacité et le temps CPU (Tableau 1). Nous notons ainsi qu'interrompre les résolutions pour préserver la privacité permet non seulement de réduire la perte de privacité de 39% mais aussi de réduire les temps de calcul de 27%, de réduire le nombre de messages échangés de 71% tout en résolvant 98% des problème résolus par les algorithmes standards, les interruptions ayant lieu majoritairement lorsque les problèmes n'ont pas de solution.

La Figure 2 illustre l'estimation moyenne de la perte de privacité par agent, en fonction

de la dureté des contraintes unaires et de la répartition de celles-ci. Nous observons que de manière générale, les algorithmes synchrones sont meilleurs que les algorithmes asynchrones pour préserver la privacité. Par ailleurs, pour une dureté supérieure à 40%, la perte de privacité est plus faible avec des problèmes de type *tail-constrained DMS* que pour des problèmes de type *uniform DMS*, en particulier pour ABTU.

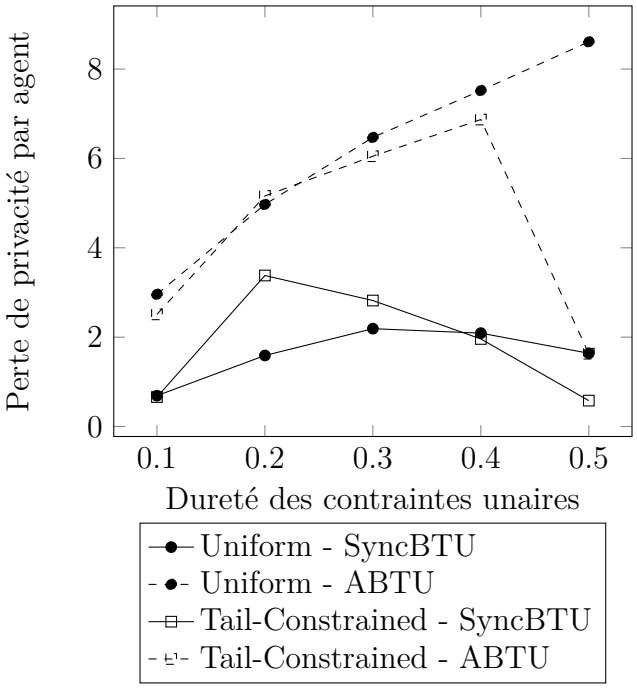

FIGURE 2 – Evaluation de la perte de privacité sur des instances avec différentes répartitions des contraintes

5 Conclusions

Bien que plusieurs travaux ont été développés récemment pour traiter la question de la privacité dans la résolution de problèmes distribués, aucun d'entre eux n'est largement utilisé, probablement en raison de la difficulté de modéliser des problèmes communs. Dans cet article, nous avons proposé un paradigme appelé *Utilitarian Distributed Constraint Satisfaction Problems* (UDisCSP). La perte de privacité pour la révélation des contraintes d'un agent y est prise en compte à travers une fonction d'utilité. Nous avons présenté des algorithmes qui permettent aux agents d'estimer dans quelle mesure la privacité sera perdue à la fin du processus de résolution de problème, en utilisant les informations issues de leurs expériences précédentes. Cette estimation est utilisée pour modifier le comportement de l'agent. Ensuite, nous avons montré comment adapter les protocoles synchrones et asynchrones (SyncBTU et ABTU). Les premiers résultats obtenus par une étude expérimentale sur des problèmes aléatoires de planification de réunion ont permis d'identifier certaines familles de problèmes présentant des propriétés particulières concernant la privacité.

À l'avenir, nous voulons évaluer la perte de privacité lors de la résolution des différentes classes de problèmes. Nous prévoyons également d'améliorer la manière dont les agents apprennent de l'expérience précédente, en utilisant non seulement la dureté des problèmes correspondants, mais aussi la densité, le nombre de variables ou le nombre de contraintes inter-agent impliquées.

Remerciements Les auteurs remercient les relecteurs pour leurs remarques constructives permettant d'améliorer notre article.

Ces travaux ont été réalisés alors que le premier auteur était chercheur invité au Florida Institute of Technology.

Références

[1] F. Bacchus and P. Van Beek. On the conversion between non-binary and binary constraint satisfaction problems. In *Proceedings of the 15th National Conference on Artificial Intelligence*, Madison, Wisconsin, 1998.

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

[3] L. Crépin, Y. Demazeau, O. Boissier, and F. Jacquenet. Privacy preservation in a decentralized calendar system. In Yves Demazeau, Juan Pavón, Juan M. Corchado, and Javier Bajo, editors, *7th International Conference on Practical Applications of Agents and Multi-Agent Systems, PAAMS 2009, Salamanca, Spain, 25-27 March 2009*, volume 55 of *Advances in Intelligent and Soft Computing*, pages 529–537. Springer, 2009.

[4] P. Doshi, T. Matsui, M. Silaghi, M. Yokoo, and M. Zanker. Distributed private constraint optimization. In *Web Intelligence and Intelligent Agent Technology,*

2008. WI-IAT'08. IEEE/WIC/ACM International Conference on, volume 2, pages 277–281. IEEE, 2008.

[5] 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*, volume 2, pages 350–358. IEEE, 2008.

[6] F. Fioretto, 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*, pages 1823–1824. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

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

[8] 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.

[9] S. Koenig, D. Furcy, and C. Bauer. Heuristic search-based replanning. In *AIPS*, pages 294–301, 2002.

[10] Y. Krupa and L. Vercouter. Handling privacy as contextual integrity in decentralized virtual communities : The privacias framework. *Web Intelligence and Agent Systems*, 10(1) :105–116, 2012.

[11] R.T. Maheswaran, J.P. 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*, pages 1030–1037. ACM, 2005.

[12] R.T. Maheswaran, M. Tambe, E. Bowring, J.P. 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*, pages 310–317. IEEE Computer Society, 2004.

[13] R. Mandiau, J. Vion, S. Piechowiak, and P. Monier. Multi-variable distributed backtracking with sessions. *Appl. Intell.*, 41(3) :736–758, 2014.

[14] S. Russell and P. Norvig. *Artificial intelligence : a modern approach*. Prentice Hall, 2010.

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

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

[17] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Asynchronous search with aggregations. In *AAAI*, pages 917–922, 2000.

[18] R.J. Wallace and E.C. Freuder. Constraint-based reasoning and privacy/efficiency tradeoffs in multi-agent problem solving. *Artificial Intelligence*, 161(1) :209–227, 2005.

[19] M. Yokoo, E.J. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem : Formalization and algorithms. *Knowledge and Data Engineering, IEEE Transactions on*, 10(5) :673–685, 1998.

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

[21] 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*, pages 387–401. Springer, 2002.

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