Préserver la confidentialité pour l'algorithme Generalized Distributed Breakout

Julien Vion[1] René Mandiau[1] Sylvain Piechowiak[1] Marius Silaghi[2]

1. LAMIH CNRS UMR 8201, Université Polytechnique Hauts de France
2. Florida Institute of Technology

1. {prénom.nom}@uphf.fr 2. msilaghi@fit.edu

Résumé

Protéger la vie privée des utilisateurs est l'un des enjeux des systèmes distribués. Dans cet article, nous mesurons la fuite d'informations lors de la résolution de problèmes d'optimisation de contraintes distribués (DCOP). Récemment, Okamoto, Zivan et Nahon [19] ont proposé l'algorithme de Breakout distribué généralisé (GDBA pour Generalized Distributed Breakout Algorithm) pour traiter ce type de problème. Cependant, la mesure et le contrôle des données échangées n'ont jamais été réalisés pour cet algorithme.

Nous étudions le comportement de GDBA lors de la résolution de DCOP et de DCOP utilitaristes, une variante dans laquelle les agents cherchent une solution de qualité tout en protégeant les informations qu'ils possèdent. Nous montrons que GDBA peut être amélioré pour préserver la plupart des informations sur les contraintes et réduire la fuite de données sur les domaines d'un facteur 2 à 3 sans impact significatif sur la qualité des solutions.

Abstract

Privacy has traditionally been a major motivation of distributed problem solving. In this paper, we focus on privacy issues when solving Distributed Constraint Optimization Problems (DCOPs). Recently, Okamoto, Zivan et Nahon [19] promoted the Generalized Distributed Breakout Algorithm (GDBA) as a very efficient heuristic strategy to find good solutions to DCOPs.

We study how GDBA behaves when solving Utilitarian DCOPs, where utilitarian agents want to reach an agreement while reducing the privacy loss. We show that GDBA can be improved to allow for extensive handling of constraint privacy, and reduce domain privacy loss by a factor 2~3 with no significant impact on the quality of the solution.

Le problème d'optimisation de contraintes distribué (DCOP pour Distributed Constraint Optimization Problem) est un modèle général en programmation par contraintes (PPC) utilisé pour modéliser et résoudre des problèmes distribués NP-difficiles. Pour résoudre un DCOP, les agents négocient afin de trouver une solution qui satisfait au mieux un ensemble de contraintes. Le respect de la vie privée est un enjeu important dans de nombreuses applications distribuées. En conséquence, en plus de limiter le coût engendré par des contraintes non satisfaites, le coût lié à la fuite d'information devrait également être prise en compte [9, 13, 26]. Par exemple, dans un problème d'ordonnancement distribué, les participants peuvent souhaiter ne pas révéler toutes leurs contraintes et leurs disponibilités. Certaines d'entre elles peuvent être liées à leur vie privée.

Il y a perte de confidentialité dès lors que les agents échangent des données. L'affectation des créneaux horaires peut être difficile si les participants ne révèlent pas leurs contraintes [3, 5, 7]. En pratique, si un agent souhaite contrôler les fuites de données, il peut associer un coût à la révélation de chaque information liée à son problème local. Ce coût peut être pris en compte par un raisonnement orienté utilités [4, 21, 22].

Des algorithmes stochastiques de recherche locale ont été introduits pour traiter les problèmes distribués, notamment Distributed Stochastic Algorithm (DSA) [28] et Generalized Distributed Breakout Algorithm [15, 19, 27] (GDBA). Ils permettent de prendre en compte les utilités relativement facilement, i. e., sans impact sur leur correction. Bien qu'incomplets, ces algorithmes peuvent trouver des solutions sous-optimales de bonne qualité relativement rapidement, dans un contexte *anytime* [25, 29].

Dans cet article, nous étudions les propriétés de GDBA du point de vue de la confidentialité. L'objectif de ce travail est d'améliorer GDBA pour préserver la vie privée de l'agent. Dans les sections suivantes, après

des rappels de notations et de contexte, nous montrons comment la confidentialité des contraintes (section 2) et des domaines (section 3) peuvent être améliorées pour GDBA. Dans la section 4, nous montrons expérimentalement l'impact des coûts des contraintes et de confidentialité des domaines sur des problèmes académiques et réalistes.

1 Contexte

Nous traitons des problèmes discrets ; les domaines sont des sous-ensembles finis de $\mathbb{Z}$, bien que n'importe quel ensemble fini dénombrable puisse être utilisé. Nous faisons quelques suppositions simplificatrices supplémentaires, sans perte de généralité : un agent n'encapsule qu'une seule variable, une contrainte implique au plus deux variables, les coûts des contraintes sont dans $\mathbb{N}$ (on pourrait utiliser $\mathbb{R}$ sans autre difficulté que leur représentation informatique, mais les coûts négatifs ne sont pas intuitifs).

Définition 1.1. Un problème d'optimisation de contraintes distribué (DCOP) est défini par un quadruplet $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, tel que :

$\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ est un ensemble de n agents.

$\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ est un ensemble de n variables. Chaque agent A_i encapsule la variable x_i.

$\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$ est un ensemble de n domaines, tels que $\forall i, |\mathcal{D}_i| \leq d$. Chaque variable x_i peut être instanciée à une valeur $v \in \mathcal{D}_i$. Une affectation de $X \subseteq \mathcal{X}$ est un ensemble d'instanciations pour chaque variable de X.

$\mathcal{C} = \{C_1, C_2, \ldots, C_e\}$ est un ensemble de e constraintes valuées. Chaque contrainte C_i implique les variables $\mathcal{X}(C_i) \subseteq \mathcal{X}$, et définit un coût pour chaque affectation de ses variables. Nous notons $C_i(X)$ le coût de l'affectation de $\mathcal{X}(C_i) \subseteq X$ pour C_i. Nous notons également $\mathcal{C}(A_i)$ l'ensemble des contraintes qui impliquent x_i, i. e., $\mathcal{C}(A_i) = \{C_j \in \mathcal{C} \mid x_i \in \mathcal{X}(C_j)\}$.

L'objectif du problème est de trouver une solution optimale S, i. e., une affectation de $\mathcal{X}$ qui minimise le coût des contraintes $\sum_{i=1}^{e} C_i(S)$.

Les contraintes peuvent être unaires et n'impliquer qu'un seul agent (on parle de contrainte intra-agent), ou binaires et mettre ainsi en jeu deux agents. Ces dernières sont appelées contraintes inter-agent et les deux agents impliqués sont alors voisins. Chaque agent A_i a exactement N_i voisins. Nécessairement, $\sum_{i=1}^{n} N_i$ est égal à deux fois le nombre de contraintes inter-agent du problème.

La *confidentialité* est la propriété qu'ont les agents de garder leurs informations secrètes. C'est un concept assez flou que plusieurs auteurs ont tenté de catégoriser [8, 10, 12]. Pour Grinshpoun [10], la confidentialité des agents peut concerner les domaines, les contraintes, les affectations et les algorithmes. Parmi les trois articles cités ci-dessus, c'est le seul à mentionner la confidentialité des domaines, bien que l'auteur indique qu'il n'est pas possible de la conserver dans le modèle DCOP traditionnel. En effet, Grinshpoun considère que les coûts des contraintes sont représentés en extension, par des matrices qui couvrent tout le produit cartésien des domaines des variables impliquées. Cela implique que les agents ont connaissance des domaines complets de leur voisins avant même que la résolution ne commence. Bien qu'il semble difficile d'implémenter un algorithme de filtrage sans cette connaissance complète des domaines voisins, le modèle de PPC « ouverte » [6] donne des pistes pour le rendre réalisable. Nous rendons possible la non-révélation des domaines lors de l'initialisation par :

— l'utilisation de de contraintes définies *en intention* par une fonction $f_i : \mathbb{Z}^{|\mathcal{X}(C_i)|} \mapsto \mathbb{N}$, i. e., définie sur un sur-ensemble de $\prod_{x_j \in \mathcal{X}(C_i)} \mathcal{D}_j$,

— l'utilisation d'algorithmes stochastiques qui ne révèlent au plus qu'une valeur de domaine par cycle.

GDBA by Okamoto, Zivan et Nahon [19] est une généralisation de DBA [15, 27] aux DCOP valués [1]. GDBA est un algorithme stochastique incomplet conçu pour trouver rapidement de bonnes solutions à un DCOP. Tout comme la plupart des algorithmes incomplets, GDBA ne permet pas de prendre en compte les contraintes « dures », i. e., les coûts infinis [2], et ne peuvent pas prouver qu'une solution est optimale. Cependant, déterminer si une solution est optimale reste un problème NP-difficile et GDBA traite bien mieux les problèmes de grande taille que les algorithmes complets à complexité exponentielle comme ABT, DPOP ou Sync-BB [12, 16, 26].

La plupart des travaux sur la confidentialité pour les DCOP se sont focalisés sur la confidentialité des affectations. Mettre en place une confidentialité complète à ce niveau nécessite l'utilisation d'agents médiateurs ou des protocoles cryptés sécurisés sophistiqués [23, 24]. À notre connaissance, aucun de ces travaux ne traite la confidentialité des domaines. Notre approche est donc différente : au lieu de vouloir garder secrètes les affectations, nous considérons les domaines comme secrets. Au cours de la résolution du problème, un agent essaie d'affecter une valeur du domaine à sa variable, et révèle cette affectation. Cela entraine une perte irré-

1. Pour éviter les confusions avec les variantes que nous introduisons dans cet article, nous nommerons parfois cet algorithme "Vanilla GDBA".

2. En pratique, on peut utiliser une grande constante K pour représenter un coût "infini".

médiable de confidentialité. Notre objectif est de minimiser ces fuites d'information.

GDBA est purement décentralisé, chaque agent raisonne localement sans passer par des médiateurs ni échange de sous-problèmes. L'algorithme fonctionne de manière synchrone : chaque agent commence par choisir un valeur aléatoirement, l'affecte à sa variable et envoie cette information à ses voisins *via* un message de type « ok ? ». À chaque étape, chaque agent choisit de manière « gloutonne » la meilleure valeur à affecter du point de vue des coûts engendrés par ses contraintes. Il calcule alors la différence des coûts des contraintes entre la précédente et la nouvelle affectation. Il transmet ce *delta* à ses voisins *via* un message de type « improve ». Une fois qu'un agent a reçu les deltas de tous ses voisins, il ne change sa valeur que si son propre delta est strictement négatif (c'est-à-dire, dans le cadre d'un problème de minimisation, que le changement de valeur entraine une amélioration de la solution) et le meilleur parmi son voisinage, avec une résolution déterministe des égalités. Si aucun agent ne peut améliorer la solution partielle de son voisinage, i. e., aucun delta n'est négatif, alors l'agent considère qu'il est dans un *quasi minimum local*. Il réalise alors un *breakout*, c'est-à-dire que toutes les contraintes non-satisfaites [3] voient leur pondération incrémentée d'une unité. Cette stratégie présente deux propriétés intéressantes : d'une part, en ne permettant qu'à l'agent présentant le meilleur delta de son voisinage de changer sa valeur, elle interdit les « oscillations » qui pourraient survenir si deux voisins changent de valeur simultanément (cf section 2). D'autre part, la pondération des contraintes permet aux agents de sortir des minima locaux [18].

Si nous revenons aux catégories de Grinshpoun [10] : nous ne considèrerons pas la confidentialité des algorithmes dans cet article, et nous supposons que tous les agents suivent le même algorithme. Cependant, ces travaux n'ont de sens qu'en l'existence d'agents « malveillants » collectant des données pour une utilisation inappropriée. Comme expliqué précédemment, nous traitons la confidentialité des domaines plutôt que des affectations. Notons que ces deux propriétés sont loin d'être indépendantes. Enfin, certaines catégorisations évoquent la confidentialité topologique, c'est-à-dire de la structure des problèmes. GDBA implique que deux agents sont voisins si et seulement si ils partageant une même contrainte. Une confidentialité complète de la topologie n'est pas réalisable, mais reste contrôlée. Dans les sections suivantes, nous traitons la confidentialité au niveau des *coûts* des contraintes, et des domaines.

3. La notion de « contrainte non-satisfaite » dans le cadre des COP valués n'est pas triviale et l'essentiel de l'article de Okamoto, Zivan et Nahon [19] est consacré à cette problématique. Nous y reviendrons dans la section 2 de cet article.

2 Confidentialité des contraintes dans GDBA

Nous souhaitons modéliser et résoudre un problème où un agent A veut organiser un rendez-vous avec les agents B et C simultanément, si possible. Si un créneau commun ne peut être trouvé, A préfèrerait rencontrer B plutôt que C, mais il ne souhaite pas que cette préférence soit révélée, afin de ne pas contrarier C. Avec les variables x_A, x_B et x_C représentant l'heure du rendez-vous du point de vue de chaque agent, on peut modéliser ce problème par un ensemble de contraintes d'égalité ($x_A = x_B$ et $x_A = x_C$) pour exprimer le fait que les participants doivent se réunir à la même heure. On peut modéliser ce problème simple en utilisant des contraintes valuées : le coût est nul si la contrainte est satisfaite, et strictement positif si elle ne l'est pas. On peut traduire les préférences de A en "payant" un coût élevé si la première contrainte n'est pas satisfaite, et plus faible pour l'autre, mais à aucun moment C ne doit être mis au courant des coûts associés à la contrainte $x_A = x_B$ [4].

Comme expliqué dans la section précédente, GDBA nécessite à chaque étape que tous les agents envoient un message de type « improve » contenant le delta. Ce dernier est la différence entre le coût de l'affectation précédente et le meilleur coût qu'il puisse obtenir en changeant d'affectation. D'une part, cela peut donner des indices sur le coût des contraintes de l'agent, et d'autre part, si on considère un coût lié à la perte d'une information, on peut se demander s'il faut incorporer ce coût de confidentialité dans le calcul du delta. Et dans ce cas, s'il faut aussi contrôler la perte de confidentialité liée aux coûts de confidentialité eux-mêmes (et, pourquoi pas, récursivement).

Nous proposons une variante de GDBA qui résoud ces deux problèmes en supprimant la phase « improve » de l'algorithme. Il reste à trouver des mécanismes alternatifs pour éviter les oscillations, et sortir des minima locaux.

Contribution 1 : prévention des oscillations en l'absence de message « improve ». L'exemple suivant montre le problème lié aux oscillations : considérons par exemple deux agents A_1 et A_2 encapsulant les variables respectives x_1 et x_2, dont les domaines sont tous les deux $\{1, 2\}$, associées par la contrainte $x_1 = x_2$.

4. Le modèle DCOP défini dans la section précédente suppose que la fonction de coût de chaque contrainte est connue des deux agents. Il serait souhaitable de réduire encore la publication de la fonction de coût, par exemple avec un modèle de type PKC [2, 11]. C'est une des perspectives de ce travail.

Initialement, $x_1 = 1$ et $x_2 = 2$, ce qui contredit la contrainte. Individuellement, pour chaque agent, la meilleure action est de changer d'affectation. À l'étape suivante, on aura $x_1 = 2$ et $x_2 = 1$, et la contrainte n'est toujours pas satisfaite. Si aucun mécanisme ne vient prévenir ce changement simultané, les deux agents vont continuer à osciller sans jamais parvenir à la solution.

L'algorithme DSA introduit un paramètre p qui définit la probabilité avec laquelle un agent va changer d'affectation entre deux cycles [28]. Autrement dit, avec une probabilité $1-p$, chaque agent peut « sauter son tour », ce qui permet à l'agent voisin de changer seul d'affectation. Nous avons introduit ce paramètre p dans notre variante de GDBA, que nous appelons dorénavant GDBA-p.

Il faut alors trouver une « bonne » valeur de p pour chaque problème. Si p est trop faible, les agents vont avoir tendance à garder leurs affectations trop longtemps, ce qui rend la recherche lente ; si p est trop élevé, le phénomène d'oscillation réapparait. Cependant, la pratique montre qu'un p relativement élevé, sans pour autant tendre vers 1, convient à une grande variété de problèmes. Dans nos conditions d'expérimentation, $p = 0{,}95$ est très satisfaisant pour l'ensemble des problèmes traités (cf section 4).

Contribution 2 : *Breakout* en l'absence de message « improve ». Nous avons réintroduit le mécanisme de *Breakout* pour échapper aux minima locaux comme suit : quand le delta d'un agent est positif ou nul, il conserve logiquement son affectation précédente, mais il envoie dans ce cas un message « stalled » à ses voisins. Si tous les voisins d'un agent sont dans ce cas, ce dernier détecte un quasi minimum local et peut réaliser le *breakout* en pondérant les contraintes non-satisfaites.

En résumé, un agent A_i exécutant GDBA-p ne réalise qu'une seule action à chaque étape :

- s'il existe une valeur $v' \in \mathcal{D}_i$ qui améliore la fonction de coût, A_i peut *affecter* $x_i = v'$ avec une probabilité p, ou
- si v' existe, avec une probabilité $1-p$, A_i ne change pas sont affectation (il *passe*). Quoi qu'il en soit, il envoie un message « ok ? » à ses voisins pour faire part de la valeur finalement choisie.
- Si v' n'existe pas, A_i est en *plateau* et envoie un message « stalled » à ses voisins. Si tous ses voisins sont dans un tel plateau, un quasi minimum local est détecté et les contraintes non-satisfaites de A_i sont pondérées. Les messages « stalled » permettent de distinguer les *« passes »* des *« plateaux »*.

Algorithme 1 : GDBA-p exécuté par l'agent A_i

```
1  Initialiser les poids des contraintes à 1
2  v ← arg min_{j∈D_i} COST(j, ∅, ∅)
3  Transmettre "ok ?(v)" à tous les voisins
4  tant que condition d'arrêt non rencontrée faire
5      stallCount ← 0
6      Recevoir les messages de tous les voisins :
7          when "ok ?" : mettre à jour agentView et minCosts
8          when "stalled" : incrémenter stallCount
9      v' ← arg min_{j∈D_i} COST(j, agentView, R_i)
10     si COST(v', agentView, R_i) < COST(v, agentView, R_i) alors
11         Avec une probabilité p : v ← v'
12         R_i ← R_i ∪ {v}
13         Envoyer "ok ?(v)" à tous les voisins
14     sinon
15         si stallCount = N_i alors // quasi minimum local
16             pour chaque C_j ∈ C(A_i) | C_j(agentView) > minCosts(C_j) faire
17                 Incrémenter le poids de C_j
18         Envoyer "stalled" à tous les voisins
19 Affecter la meilleure v à x_i
```

Contribution 3 : non-minimalité dans un contexte de PPC ouverte. Le travail de Okamoto, Zivan et Nahon [19] avait consisté à généraliser l'algorithme legacy DBA de Yokoo et Hirayama [27] aux contraintes valuées. Dans ce cadre, la notion de contrainte « non-satisfaite » se révèle insuffisante pour déterminer les contraintes à pondérer. Yokoo et Hirayama ont mesuré les performances de plusieurs variantes de « non-satisfaction ». Parmi celles-ci, la plus prometteuse est celle de non-minimalité (NM). Une contrainte est NM-insatisfaite si le coût de la solution actuelle est plus elevé que le coût minimal pouvant être obtenu par cette contrainte (i. e., pas forcément 0). Cependant, dans le cadre de PPC ouverte, si les domaines des voisins sont inconnus et la fonction de coût définie sur des ensembles infinis, il nous est impossible de connaitre le coût minimal. Nous avons traité ce problème en maintenant une structure de données *minCosts*, qui associe à chaque contrainte C_i le coût le plus bas *connu* pour cette contrainte. Quand un voisin révèle une nouvelle valeur de son domaine, tous les coûts de chaque contrainte incluant cette nouvelle valeur sont évalués et *minCost* est mise à jour en conséquence. Pour maintenir cette structure, chaque affectation n'est évaluée qu'une fois pour chaque contrainte tout au long du processus de recherche, c'est-à-dire qu'il faut effectuer au plus $d^{|\mathcal{X}(C_i)|}$ tests de la contrainte. Comme nous nous sommes restreints au cas des contraintes binaires, nous n'avons pas rencontré d'explosion combinatoire. Généraliser cette technique aux contraintes non-binaires nécessitera probablement la mise en place de fonctions ad-hoc (comme pour les contraintes globales en PPC clas-

sique) ou d'approximations.

La valeur de $minCosts(C_i)$ ne peut que décroitre au cours de la recherche, mais cela signifie que C_i peut être considérée comme minimalement satisfaite au début de la recherche, mais non-minimalement satisfaite plus tard par la même affectation. Cette propriété, bien que contre-intuitive, ne pose pas de difficulté à l'algorithme. La complexité spatiale de *minCosts* est $\Theta(|\mathcal{C}(A_i)|)$ pour chaque agent, soit globalement $\Theta(e)$. Il faut également tenir compte de l'ensemble des valeurs révélées par chaque voisin de chaque agent, ce qui nécessite pour chaque A_i une structure de données de taille $\Theta(N_i \cdot d)$, soit globalement $\Theta(ed)$. GDBA-p a les mêmes complexités temporelle et spatiale que Vanilla GDBA. L'algorithme tourne jusqu'à ce qu'une condition d'arrêt soit rencontrée, généralement une limite sur le nombre de cycles.

GDBA-p est décrit extensivement ci-après (algorithme 1) : v représente l'instantiation actuelle de la variable de l'agent. *agentView* contient une affectation représentant les instantiations connues des voisins. Une de nos contributions mineures consiste à affecter initialement v non pas aléatoirement, mais à la meilleure valeur possible (ligne 2). Même si on ne peut pas à ce stade prendre en compte les contraintes inter-agent, on minimise les coûts des contraintes unaires et de confidentialité décrits dans la section suivante. Cette valeur est transmise aux voisins (ligne 3).

Les lignes 6 à 8 traitent les messages entrants. Les agents GDBA-p envoient et reçoivent exactement un message par voisin à chaque étape (contre deux pour Vanilla GDBA). La ligne 9 sélectionne la meilleure valeur v' pour la fonction COST. Cette fonction prend en compte l'*agentView*, et l'ensemble des valeurs précédemment révélées $\mathcal{R}_i$ (cf section suivante). La ligne 10 vérifie si v' améliore la soluton courante. Si c'est le cas, v' est affectée, avec une probabilité p (ligne 11). C'est ici la principale différence entre GDBA-p et Vanilla GDBA : dans ce dernier, il y a une étape supplémentaire où le delta entre les coûts associés à v et v' est envoyé à tous les voisins, et v' n'est affectée que si l'agent courant permet la meilleure amélioration. Quoi qu'il en soit, la valeur choisie est ajoutée à $\mathcal{R}_i$ (ligne 12) et envoyée aux voisins (ligne 13).

Si v' n'améliore pas la solution courante, la ligne 15 détecte la présence d'un quasi minimum local. Dans ce cas, les lignes 16 et 17 pondèrent les contraintes non-minimalement satisfaites.

3 Confidentialité des domaines dans GDBA

Pour illustrer notre travail, nous voulons modéliser le problème suivant : un agent A organise une réunion avec plusieurs collègues. Il est disponible toute la journée, de 8 h à 20 h, avec des préférences variables, mais ne souhaite par avouer qu'il a autant de disponiblités, afin de ne pas être submergé de demandes de réunions. Il est possible d'obtenir ce résultat en gardant les domaines privés, et en révélant aussi peu de valeurs que possible au cours de la recherche.

Contribution 4 : contrôle des données révélées *via* des utilités. Comme décrit dans la section 1, l'utilisation d'algorithmes stochastiques ainsi que de contraintes définies en intention sur des sur-ensembles des domaines permettent de ne pas dévoiler systématiquement tous les domaines lors de la phase d'initialisation. Cela n'interdit pas pour autant l'existence de contraintes à base de tables : on peut les définir sur des sur-ensembles des domaines, utiliser des valeurs par défaut si la table n'est pas exhaustive, ou encore des représentations compressées (e. g., [17]).

Quand un agent affecte une valeur à sa variable, il transmet le message « ok ? » à ses voisins. Il s'agit d'une perte de confidentialité. Le modèle de DCOP Utilitariste permet d'affecter un coût à cette fuite d'information [4, 21, 22]. Pour contrôler la confidentialité des domaines, nous considérons que la recherche est réalisée par des agents utilitaristes qui prennent leurs décisions pour minimiser leur propre fonction d'utilité. Celle-ci implique de minimiser d'une part le coût global, et d'autre part la perte de confidentialité. Ceci peut être formalisé comme suit :

Définition 3.1. Un DCOP utilitariste (UDCOP) est un quintuplet $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{U} \rangle$, i. e., un DCOP avec une matrice supplémentaire $\mathcal{U}$ de coûts de confidentialité sur les domaines. $\mathcal{U}_{i,j}$ est le coût, positif ou nul, pour A_i de révéler si $j \in \mathcal{D}_i$.

L'objectif est de trouver une solution optimale tout en minimisant la perte de confidentialité $\sum_{i=1}^{n} \sum_{j \in \mathcal{R}_i} \mathcal{U}_{i,j}$, où $\mathcal{R}_i \subseteq \mathcal{D}_i$ est l'ensemble des valeurs que A_i à révélées dans le processus de résolution.

L'objectif est maintenant double. Cependant, la perte de confidentialité dépend du processus de recherche lui-même et pas uniquement du problème et de sa solution. On ne peut donc pas utiliser les techniques traditionnelles de résolution multi-objectif. Les deux objectifs sont contradictoires : la perte de confidentialité ne peut que s'accroitre au cours de la recherche, au fur et à mesure que de nouvelles valeurs sont révélées, alors que les coûts sur les contraintes vont descendre alors que de nouvelles solutions sont découvertes. Nous voulons contrôler le processus de recherche pour parvenir à un compromis. Dans des travaux précédents et dans le cadre d'algorithmes complets, nous avons utilisé des éléments de théorie des jeux [20]. Ici, nous

Algorithme 2 : $\text{cost}(v, agentView, \mathcal{R}_i)$

1 $constraintCost \leftarrow \sum_{C_j \in \mathcal{C}(A_i)} weight(C_j) \times C_j(agentView \cup \{x_i = v\})$

2 $privacyCost \leftarrow \left(\max_{C_j \in \mathcal{C}(A_i)} weight(C_j) \right) \times \sum_{j \in R_i \cup \{v\}} \mathcal{U}_{i,j}$

3 **retourner** $constraintCost + privacyCost$

avons choisi de nous baser sur la *somme* entre les coûts des contraintes et la perte de confidentialité.

Cette technique peut être appliquée à DSA, Vanilla GDBA ou GDBA-p. Une des difficultés avec GDBA et ses variantes est que la pondération des contraintes peut, à terme, rendre les coûts de confidentialité négligeables. Nous avons observé expérimentalement que donner autant de poids aux coûts de confidentialité qu'à la contrainte la plus pondérée donnait de très bons résultats. L'algorithme 2 résume la fonction COST finale que nous avons utilisée. La fonction renvoie le coût (contraintes et confidentialité) qu'entrainerait l'affectation de la valeur v à l'étape courante, étant donnés *agentView* et $\mathcal{R}_i$. Pour prendre en compte les contraintes unaires et des coûts de confidentialité lors de l'initialisation de l'algorithme, un coût par défaut (0) est renvoyé si un *agentView* incomplet ne permet pas de déterminer le coût d'une contrainte.

La valeur *privacyCost* calculée à la ligne 2 représente ce que nous appelons le *contrôle de la confidentialité des domaines* (DPC pour Domain Privacy Cost). Cette valeur peut être mémorisée entre deux appels à la fonction et mise à jour (en temps constant) quand le poids de la contrainte la plus pondérée est incrémenté, ou qu'une nouvelle valeur est révélée. Ainsi, la DPC n'a pas d'impact sur la complexité temporelle de GDBA.

4 Expérimentations

Nous avons implémenté DSA, GDBA et GDBA-p avec et sans DPC sur la plateforme Akka 2.5 [1]. Akka est une implémentation moderne et performante du modèle Acteurs [14] pour la machine virtuelle Java. Elle convient parfaitement pour simuler des agents DCOP dans notre contexte expérimental.

Nous avons testé les algorithmes sur 150 instances des benchmarks suivants, et avons mesuré la moyenne des coûts *anytime* à chaque cycle. Les valeurs aléatoires non-négatives ont été générées, sauf exception, d'après une distribution log-normale de moyenne μ et écart-type σ donnés, arrondie à l'entier le plus proche.

MultiDMS est un problème d'ordonnancement de réunions distribué : 40 réunions doivent être programmées sur $d = 30$ crénaux horaires. 100 individus sont impliqués dans ces réunions : pour chacune d'elles, chaque personne peut être sélectionnée pour participer avec une probabilité de 5 %. Chaque réunion a une durée ($\mu = 2{,}5; \sigma = 1{,}12$). De plus, les participants on besoin de temps ($\mu = 1{,}5; \sigma = 0{,}5$) pour passer d'une réunion à la suivante. Si le planning d'une personne nécessite d'être présent à deux réunions à la fois, ou que la contrainte de distance n'est pas respectée, une pénalité de $K = 100$ est imposée. Une contrainte unaire est générée pour chaque variable, donnant à chaque valeur un coût ($\mu = 20; \sigma = 6{,}06$). Un coût de confidentialité est généré pour chaque valeur ($\mu = 10; \sigma = 3$).

RDO est un problème binaire aléatoire de $n = 200$ variables avec un domaine de $d = 30$ valeurs. Un graphe de contraintes aléatoire d'une densité de 2 % est généré ($e = 398$). Chaque contrainte impose un coût ($\mu = 50; \sigma = 50$) pour chaque affectation possible de ses variables. Un coût de confidentialité est généré pour chaque valeur ($\mu = 10; \sigma = 10$).

WGC est une variante de coloration de graphe pondérée sur $n = 200$ variables avec $d = 30$ couleurs. Un graphe de contraintes aléatoire est généré avec une densité de 20 % ($e = 3\,980$). Ces contraintes binaires sont des contraintes d'inégalité "dures" ($K = 1\,000$). Une contrainte unaire de coût est générée pour chaque variable, imposant un coût à chaque valeur ($\mu = 50; \sigma = 50$). Un coût de confidentialité est généré pour chaque valeur ($\mu = 10; \sigma = 10$).

Nous avons choisi les paramètres de MultiDMS pour correspondre grossièrement à notre application cible, i. e., générer un planning hebdomadaire pour notre établissement. Le nombre de participants à une réunion suit une distribution binomiale de moyenne 5. Chaque paire (réunion, participant) nécessite une variable, soit 200 variables en moyenne pour chaque instance. Pour ce problème, notre implémentation peut exécuter environ 500 cycles de GDBA par seconde (temps *wall-clock*) sur une VM Serveur Java OpenJDK 11 64-Bit exécutée sur un processeur à 4 cœurs Intel i7-5600U @ 2,6 GHz avec 4 GiB de mémoire de tas allouée. Pour les deux autres classes de problèmes, plus académiques, nous avons choisi les paramètres pour obtenir des problèmes de taille similaire (variables et domaines) et un temps d'exécution similaire.

Une première expérimentation compare DSA-0.95, GDBA et GDBA-p pour plusieurs valeurs de p. DPC est désactivé en supprimant la ligne 2 de l'algorithme 2, seul les coûts de contrainte sont observés. La figure 1 présente les résultats. Ces graphiques

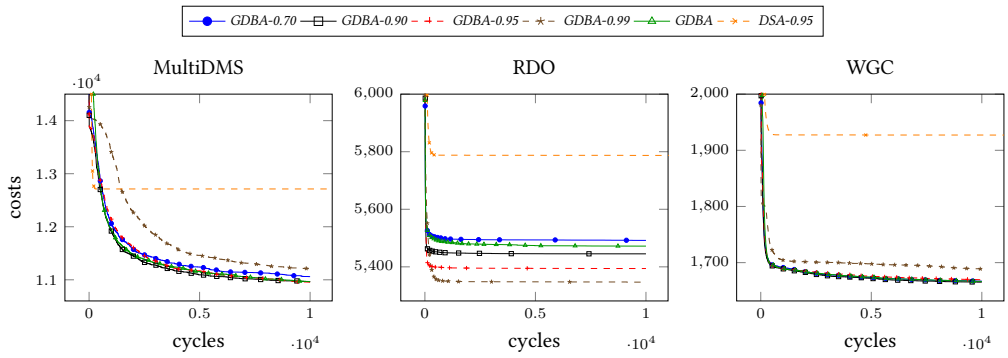

FIG. 1 – Coûts de contraintes *anytime* pour chaque cycle avec GDBA et GDBA-p

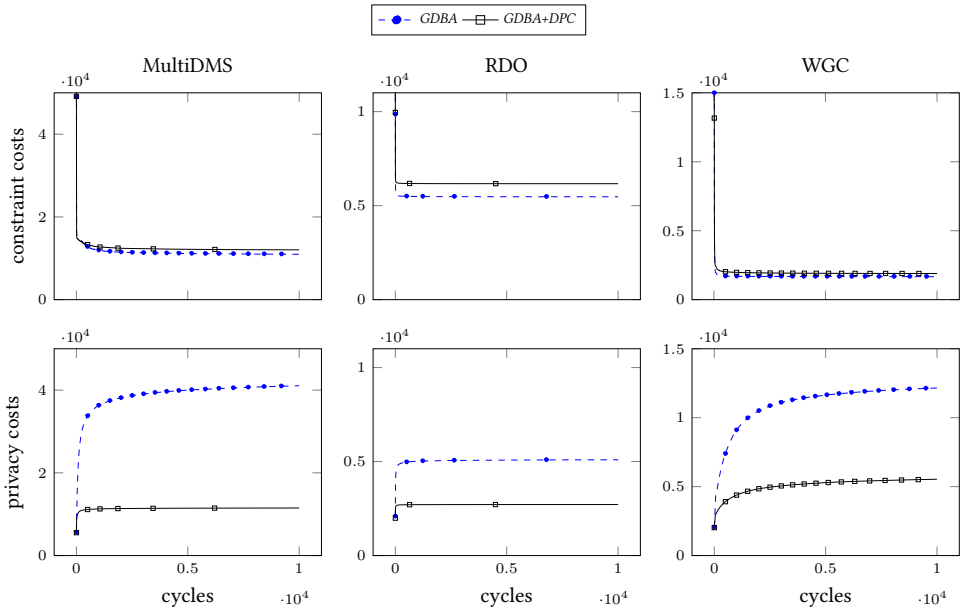

FIG. 2 – Impact de DPC sur les coûts de contrainte et de confidentialité *anytime* pour GDBA

montrent que GDBA-0.90 est toujours meilleur que Vanilla GDBA sur ces benchmarks. Cependant, GDBA-0.95 est nettement meilleur que GDBA-0.90 sur RDO, et la différence est presque impercetible sur les autres. GDBA-0.99 fonctionne particulièrement bien sur les problèmes RDO : les oscillations ne surviennent probablement pas sur un problème si homogène. Pour DSA, nous avons réalisé une expérimentation préliminaire pour déterminer le meilleur paramètre, qui s'est révélé être 0.95 également. DSA ne bénéficie d'aucun mécanisme pour échapper aux minima locaux et bloque rapidement sur une solution de mauvaise qualité. Notons que cela implique que peu de valeurs sont révélées.

Une deuxième expérimentation compare les coûts des contraintes et de confidentialité pour Vanilla GDBA et GDBA-p, avec et sans DPC. Nous avons choisi ici d'*inclure* les coûts de confidentialité dans le calcul des deltas pour les message "improve". Cela signifie que de l'information sur les coûts de confidentialité a pu être transmise, mais les prendre en compte réduit les coûts de confidentialité eux-mêmes. Les résultats sont présentés sur la figure 2. On peut observer que DPC permet de réduire les coûts de confidentialité d'un facteur 2 à 3 en moyenne, tout en ayant un impact relativement faible sur les coûts des contraintes. Des résultats similaires sont obtenus avec GDBA-0.95 (figure 3). On remarque également que GDBA-0.95 tend à révéler plus de valeurs que Vanilla GDBA. Pour mieux observer ce phénomène avec plusieurs valeurs de p, nous avons ajouté l'expérimentation suivante.

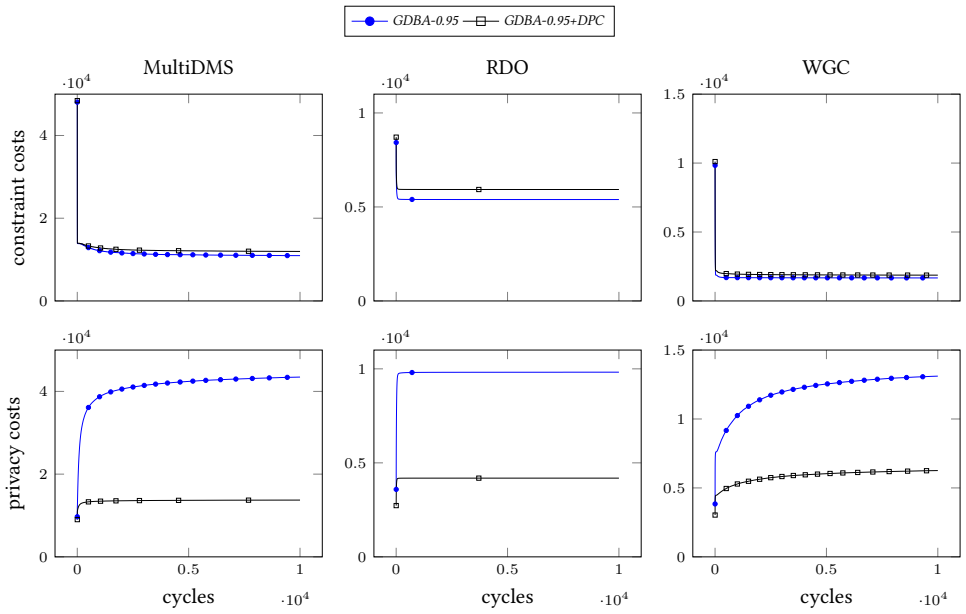

FIG. 3 – Impact de DPC sur les coûts de contrainte et de confidentialité *anytime* pour GDBA-0.95

La troisème expérimentation compare les coûts de confidentialité pour Vanilla GDBA et GDBA-p pour différents p, avec DPC désactivé ou activé. La figure 4 montre que Vanilla GDBA ou GDBA-p avec de petites valeurs de p réduit le nombre de valeurs révélées, ce qui fait sens dans la mesure où un p élevé implique des réaffectations plus aggressives. Ce comportement s'observe également quand DPC est activé, mais dans une moindre mesure. Pour WGC avec DPC désactivé et $p = 0{,}99$, i. e., une valeur trop élevée, l'impact des oscillations est tellement négatif sur la qualité de la résolution que peu de valeurs sont révélées.

Bien que, sous certaines conditions, Vanilla GDBA soit parfois plus efficace que GDBA-0.95, nous rappelons ici que contrairement à Vanilla GDBA, GDBA-p ne transmet aucune information sur les coûts des contraintes, ce qui n'apparait pas sur les graphiques.

5 Conclusion et perspectives

Dans cet article, nous avons proposé une variante de l'algorithme de Breakout distribué généralisé (GDBA), que nous appelons GDBA-p. Cette variante supprime les messages « improve » de l'algorithme original. Ceci a pour double avantage de réduire le nombre de messages échangés d'un facteur 2, et de supprimer toute transmission d'information sur les coûts des contraintes. GDBA-p met en place des stratégies alternatives pour limiter le phénomène d'oscillation et de pièges dans des minima locaux.

Une deuxième série de contributions concerne la mise en place d'un contrôle de la confidentialité des domaines (DPC) pour DSA, GDBA et GDBA-p, en exploitant le modèle de DCOP utilitariste proposé par DOSHI et al. [4] et SAVAUX et al. [21].

Nos résultats montrent que la DPC a un impact relativement faible sur la qualité des solutions, mais permet de réduire la fuite de données sur les domaines d'un facteur deux à trois.

Des perspectives ont été évoquées au cours de l'article : en premier lieu, il sera nécessaire d'expérimenter nos approches sur des problèmes présentant plusieurs variables par agent, des contraintes non-binaires, ou aux coûts exprimés dans $\mathbb{R}$. Les problèmes multi-variables sont particulièrement intéressants dans la mesure où il n'y a pas de perte de confidentialité lorsqu'un agent utilise des valeurs pour ses raisonnements locaux. D'autre part, aucune généralisation de DSA et GDBA aux problèmes multi-variables n'a encore été proposée.

Il serait également intéressant de comparer la qualité des solutions obtenues par les algorithmes stochastiques utilisés ici aux solutions optimales des problèmes. En travaillant sur des instances de plus petite taille, on pourra trouver des solutions optimales et observer la marge de progression à trouver pour améliorer ces algorithmes. La littérature sur les algorithmes incomplets (méta-heuristiques) est très riche dans le cas des problèmes centralisés, et il y a sans aucun doute beaucoup de choses à expérimenter sur des problèmes distribués.

Du point de vue de la confidentialité, on a évoqué au cours de l'article l'idée d'améliorer la confidentia-

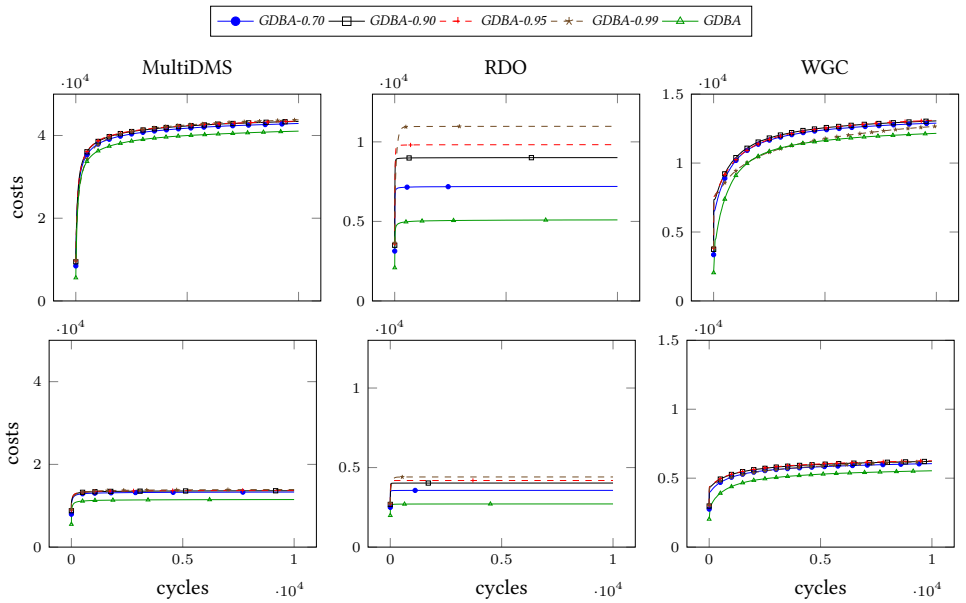

FIG. 4 – Coûts de confidentialité pour GDBA et GDBA-p : DPC désactivé (au dessus) et activé (en dessous)

lité topologique par l'utilisation des contraintes asymétriques/PKC [2, 11]. De plus, si la fonction d'évaluation d'une contrainte n'est connue que d'un agent, il n'a pas besoin de transmettre son affectation à son voisin. Cela réintroduit une forme de confidentialité des affectations.

Nous avons évoqué le fait que le problème UDCOP est un problème à deux objectifs. Nous avons fait le choix d'utiliser la somme des coûts des contraintes et des coûts de confidentialité dans cet article, mais d'autres associations doivent être envisagés. Il n'est pas possible d'utiliser un front de Pareto puisque les coûts de confidentialité ne peuvent que croître au cours de la recherche, mais on peut envisager des produits, rapports, pondérations, des jeux, etc.

Enfin, une autre approche pour tester les performances des algorithmes pour la confidentialité serait de concevoir des agents malveillants qui ont pour stratégie de collecter le maximum de données de leurs voisins. Quelles garanties permettent DSA, GDBA, GDBA-p et les utilités dans ce cadre ?

Références

[1] J. BONÉR et THE AKKA TEAM AT LIGHTBEND. *Akka : Build powerful reactive, concurrent, and distributed applications more easily*. akka.io. 2009.

[2] I. BRITO, A. MEISELS, P. MESEGUER et R. ZIVAN. "Distributed constraint satisfaction with partially known constraints". In : *Constraints* 14.2 (2009), p. 199-234.

[3] L. CRÉPIN, Y. DEMAZEAU, O. BOISSIER et F. JACQUENET. "Privacy preservation in a decentralized calendar system". In : *Proc. 7th Itl Conf on Practical Applications of Agents and Multi-Agent Systems (PAAMS)*. T. 55. Advances in Intelligent and Soft Computing. 2009, p. 529-537.

[4] P. DOSHI, T. MATSUI, M. SILAGHI, M. YOKOO et M. ZANKER. "Distributed private constraint optimization". In : *Proc. IEEE/WIC/ACM Itl Conf on Intelligent Agent Technology*. IEEE Computer Society, 2008, p. 277-281.

[5] B. FALTINGS, T. LÉAUTÉ et A. PETCU. "Privacy guarantees through distributed constraint satisfaction". In : *Proc. IEEE/WIC/ACM Itl Conf on Intelligent Agent Technology*. T. 2. IEEE Computer Society. 2008, p. 350-358.

[6] B. FALTINGS et S. MACHO-GONZALEZ. "Open constraint programming". In : *Artificial Intelligence* 161.1 (2005). Distributed Constraint Satisfaction, p. 181-208. ISSN : 0004-3702.

[7] E. C. FREUDER, M. MINCA et R. J. WALLACE. "Privacy/efficiency tradeoffs in distributed meeting scheduling by constraint-based agents". In : *IJCAI Workshop on Distributed Constraint Reasoning*. 2001, p. 63-72.

[8] R. Greenstadt, B. Grosz et M. D Smith. "SSDPOP : improving the privacy of DCOP with secret sharing". In : *Proc. of the 6th Itl joint conference on Autonomous agents and multiagent systems*. IFAAMAS, 2007, p. 171.

[9] R. Greenstadt, J. P. Pearce et M. Tambe. "Analysis of privacy loss in distributed constraint optimization". In : *Proc 21st Nat. Conf. on Artificial Intelligence and the 18th Innovative Applications of Artificial Intelligence Conf.* AAAI Press, 2006, p. 647-653.

[10] T. Grinshpoun. "When You Say (DCOP) Privacy, What do You Mean ? - Categorization of DCOP Privacy and Insights on Internal Constraint Privacy". In : *Proc. of the 4th Itl Conference on Agents and Artificial Intelligence (ICAART)*. Sous la dir. de Joaquim Filipe et Ana L. N. Fred. SciTePress, 2012, p. 380-386.

[11] T. Grinshpoun, A. Grubshtein, R. Zivan, A. Netzer et A. Meisels. "Asymmetric distributed constraint optimization problems". In : *Journal of Artificial Intelligence Research* 47 (2013), p. 613-647.

[12] T. Grinshpoun et T. Tassa. "P-SyncBB : A Privacy Preserving Branch and Bound DCOP Algorithm". In : *J. Artif. Intell. Res. (JAIR)* 57 (2016), p. 621-660.

[13] Y. Hamadi, C. Bessiere et J. Quinqueton. "Backtracking in distributed Constraint Networks". In : *Proc. of 13th European Conf. on Artificial Intelligence (ECAI)*. Brighton, UK, 1998, p. 219-223.

[14] C. Hewitt, P. Bishop et R. Steiger. "A Universal Modular ACTOR Formalism for Artificial Intelligence". In : *Proc. 3rd IJCAI*. Stanford, USA, 1973, p. 235-245.

[15] K. Hirayama et M. Yokoo. "The distributed breakout algorithms". In : *Artificial Intelligence* 161.1-2 (2005), p. 89-115.

[16] T. Léauté et B. Faltings. "Protecting Privacy through Distributed Computation in Multiagent Decision Making". In : *Journal Artificial Intelligence Research (JAIR)* 47 (2013), p. 649-695.

[17] J.-B. Mairy, Y. Deville et C. Lecoutre. "The Smart Table Constraint". In : *Proc. 12th CPAIOR*. Sous la dir. de L. Michel. Springer International Publishing, 2015, p. 271-287. isbn : 978-3-319-18008-3.

[18] P. Morris. "The breakout method for escaping from local minima". In : *Proceedings of AAAI'93*. 1993, p. 40-45.

[19] S. Okamoto, R. Zivan et A. Nahon. "Distributed Breakout : Beyond Satisfaction". In : *Proc. 25th IJCAI*. 2016, p. 447-453.

[20] J. Savaux, J. Vion, S. Piechowiak, R. Mandiau, T. Matsui, K. Hirayama, M. Yokoo, S. Elmane et M. Silaghi. "Privacy Stochastic Games in Distributed Constraint Reasoning". In : *Annals of Mathematics and Artificial Intelligence* (to appear) (2019).

[21] J. Savaux, J. Vion, S. Piechowiak, R. Mandiau, T. Matsui, K. Hirayama, M. Yokoo, S. Elmane et M. Silaghi. "Utilitarian Approach to Privacy in Distributed Constraint Optimization Problems". In : *Proc. 30th FLAIRS*. 2017, p. 454-459.

[22] M. Silaghi et D. Mitra. "Distributed constraint satisfaction and optimization with privacy enforcement". In : *Intelligent Agent Technology (IAT)*. IEEE. 2004, p. 531-535.

[23] T. Tassa, R. Zivan et T. Grinshpoun. "Preserving Privacy in Region Optimal DCOP Algorithms". In : *Proc. 25th IJCAI*. 2016, p. 496-502.

[24] T. Tassa, R. Zivan et T. Grinshpoun. "Privacy preserving implementation of the Max-Sum algorithm and its variances". In : *JAIR* 59 (2017), p. 311-349.

[25] R. J. Wallace et E. C. Freuder. "Anytime algorithms for constraint satisfaction and SAT problems". In : *ACM SIGART Bulletin* 7.2 (1996), p. 7-10.

[26] M. Yokoo, E. H. Durfee, T. Ishida et K. Kuwabara. "The distributed constraint satisfaction problem : Formalization and algorithms". In : *IEEE Transactions on knowledge and data engineering* 10.5 (1998), p. 673-685.

[27] M. Yokoo et K. Hirayama. "Distributed breakout algorithm for solving distributed constraint satisfaction problems". In : *Proc. of the 2nd Itl Conf on Multi-Agent Systems*. AAAI Press, 1996, p. 401-408.

[28] W. Zhang, G. Wang et L. Wittenburg. "Distributed stochastic search for constraint satisfaction and optimization : Parallelism, phase transitions and performance". In : *Proceedings of AAAI Workshop on Probabilistic Approaches in Search*. Alberta, Canada, juil. 2002.

[29] R. Zivan, S. Okamoto et H. Peled. "Explorative anytime local search for distributed constraint optimization". In : *Artificial Intelligence* 212 (juil. 2014), p. 1-26.