DeciMaxSum: Décimer pour résoudre des DCOP cycliques plus efficacement

Jesús Cerquides[a] cerquide@iiia.csic.es
Rémi Emonet[b] remi.emonet@univ-st-etienne.fr
Gauthier Picard[b,c] gauthier.picard@emse.fr
Juan A. Rodríguez-Aguilar[a] jar@iiia.csic.es

[a]IIIA-CSIC, Campus UAB, 08193 Cerdanyola, Catalonia, Spain

[b]Univ Lyon, UJM-Saint-Etienne, CNRS, Institut d'Optique Graduate School, Laboratoire Hubert Curien UMR 5516, F-42023, SAINT-ETIENNE, France

[c]Mines Saint-Etienne, Univ Lyon, Univ Jean Monnet, IOGS, CNRS, UMR 5516 LHC, Institut Henri Fayol, Departement ISI, F - 42023 Saint-Etienne France

Résumé

Dans le cadre de la résolution des problèmes d'optimisation des contraintes distribués (DCOP), les algorithmes approchés de propagation de croyances (BP) comme Max-Sum sont des candidats de choix. Cependant, lorsque le modèle graphique sous-jacent est très cyclique, ces méthodes de résolution souffrent de mauvaises performances, en raison de la non-convergence et des trop nombreux messages échangés. Afin d'améliorer les performances de Max-Sum sur de tels DCOPs, nous proposons de s'inspirer de la décimation guidée par BP pour résoudre des problème k-SAT. Nous proposons la nouvelle méthode DeciMaxSum, paramétrable par des critères de déclenchement de décimation, de choix de variables à décimer et de valeurs pour ces variables. Sur la base d'une évaluation expérimentale sur le modèle d'Ising, certaines de ces combinaisons de critères présentent de meilleures performances que les algorithmes concurrents.

Mots-clés : *DCOP, Max-Sum, décimation*

Abstract

In the context of solving large distributed constraint optimization problems (DCOP), belief-propagation (BP) like Max-Sum are candidates of choice. However, when the factor graph is very loopy, these solution methods suffer from bad performance, due to non-convergence and many exchanged messages. As to improve performances of Max-Sum when solving such DCOPs, we propose to take inspiration from BP-guided decimation used to solve k-SAT problems. We propose the novel DeciMaxSum method, which is parameterized in terms of policies to decide when to trigger decimation, which variables to decimate, and which values to assign to decimated variables. Based on an empirical evaluation on the Ising model, some of these combinations of policies exhibit better performance than state-of-the-art competitors.

Keywords: *DCOP, Max-Sum, decimation*

1 Introduction

Dans un contexte multi-agent, l'optimisation sous contraintes distribuée (DCOP) est un paradigme pratique pour modéliser des problèmes de coordination auxquels les agents devront faire face, comme l'allocation de ressources. Dans un DCOP, chaque agent gère une ou plusieurs variables auxquelles il doit assigner des valeurs (e.g. un objectif, une décision), tout en tenant compte de contraintes qu'il a avec d'autres agents. Résoudre un DCOP consiste à mettre en interaction les agents afin de minimiser la violation de ces contraintes. Plusieurs méthodes de résolution existent pour résoudre de tels problèmes, allant de solutions complètes et optimales, à des solutions approchées. Dans les cas à large échelle, les algorithmes approchées sont des candidats de choix. En effet, les méthodes complètes, comme ADOPT ou DPOP [11, 16], souffrent de coûts de communication et/ou de calcul exponentiels dans le cas général. En conséquence, dans de nombreux cas applicatifs, les méthodes approchées sont préférées, comme le confirme la très large littérature sur le domaine (voir [1, 4] pour des états de l'art assez complets). Une des difficultés majeures pour résoudre un DCOP est l'existence de cycles dans la structure graphique du problème (graphe de contraintes, pseudo-arbre, ou graphe de facteurs). Parmi les méthodes approchées, les méthodes par inférence, comme Max-Sum [3] et ses extensions [17], ont démontré de très bonnes performances même dans des configurations cycliques. Cependant, il existe certains cas, avec de très nombreux cycles, ou un grand diamètre de graphe (i.e. plus long chemin pour rejoindre n'importe quelle paire du points du graphe), où ces algorithmes exhibent de très mauvaises per-

formances, ce qui se traduit en la génération d'un très grand nombre de messages, un temps excessif de convergence, et/ou une solution de mauvaise qualité (en terme d'optimalité).
Une approche originale pour résoudre la présence de cycles est de « casser » ces cycles en *décimant* des variables en cours de résolution. La décimation est une méthode inspirée de la physique statistique, et appliquée à la propagation de croyances (BP) [5], qui consiste à fixer une valeur à une variable, en utilisant les valeurs marginales comme critère de décision pour sélectionner une variable à décimer [14]. La décimation est effectuée régulièrement après la convergence d'une procédure classique de propagation de croyances. Dans [12], la décimation a été utilisée dans le cadre de la satisfaction de contraintes, pour résoudre des problèmes de k-SAT de manière centralisée. Nous inspirant de ce concept, nous proposons un cadre général pour appliquer la décimation aux DCOPs. D'autres travaux ont proposé Max-Sum_AD_VP afin d'améliorer les performances de Max-Sum sur des graphes cycliques [20]. L'idée est d'effectuer un mécanisme d'inférence sur une version dirigée et acyclique du graphe sous-jacent au DCOP, pour supprimer les cycles, et d'alterner la direction des arcs à fréquence fixe afin d'améliorer la solution sous-optimale trouvée à l'alternance précédente. Un des mécanismes parmi ces extensions, à savoir la propagation de valeurs, peut être vue comme une décimation temporaire.
Au regard de ce constat, l'objectif de ce papier est de proposer un cadre général pour installer un mécanisme de décimation dans Max-Sum. Plus précisément, nous proposons les contributions suivantes :

1. Nous proposons une méthode de résolution paramétrique, appelée DeciMaxSum, pour mettre en œuvre la décimation dans Max-Sum. Elle prend trois paramètres fondamentaux : (i) une politique spécifiant quand déclencher la décimation, (ii) une politique spécifiant quelles variables décimer, et (iii) une politique spécifiant quelles valeurs assigner aux variables décimées.
2. Nous proposons un catalogue de politiques de décimation ; certaines inspirées par l'état de l'art, et d'autres plus originales. Beaucoup de combinaisons sont possibles, en fonction des problèmes à résoudre.
3. Nous évaluons quelques unes de ces combinaisons sur un benchmark classique très cyclique (le modèle d'Ising) en comparaison d'algorithmes de l'état de l'art (notamment Max-Sum and Max-Sum_AD_VP).

La section 2 présente les concepts de bases sur les DCOPs, et expose l'algorithme de décimation dont nous nous sommes inspirés pour concevoir DeciMaxSum. La section 3 définie le cadre général de DeciMaxSum, et plusieurs exemples de politiques de décimation. La section 4 présente des résultats expérimentaux et leurs analyses, sur le fonctionnement de DeciMaxSum, avec plusieurs combinaisons de politiques, comparé à des algorithmes de l'état de l'art. Enfin, la section 5 conclut ce papier avec des perspectives.

2 Notions importantes

Cette section expose le cadre des DCOPs et des algorithmes de propagation de croyances de la littérature sont discutés du point de vue de la gestion des cycles dans le graphe sous-jacent.

2.1 Le cadre des DCOPs

Une façon de modéliser des prises de décisions conjointes entre agents est le formalisme des problèmes d'optimisation sous contraintes distribués (ou DCOP).

Définition 1 (DCOP)**.** *Un* problème d'optimisation sous contraintes distribué *(ou DCOP) discret est un tuple* $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, C, \mu \rangle$*, où :* $\mathcal{A} = \{a_1, \ldots, a_{|A|}\}$ *est un ensemble d'agents ;* $\mathcal{X} = \{x_1, \ldots, x_N\}$ *sont les variables de décision ;* $\mathcal{D} = \{\mathcal{D}_{x_1}, \ldots, \mathcal{D}_{x_N}\}$ *est un ensemble de domaines finis, de telle sorte que la variable* x_i *prenne ses valeurs dans* $\mathcal{D}_{x_i} = \{v_1, \ldots, v_k\}$ *;* $C = \{u_1, \ldots, u_M\}$ *est un ensemble de contraintes souples, où chaque* u_i *défini une utilité* $\in \mathbb{R} \cup \{-\infty\}$ *pour chaque combinaison d'affectation de valeurs au sous-ensemble de variables* $\mathcal{X}_i \subseteq \mathcal{X}$ *(une contrainte n'est initialement connue que des agents impliqués) ;* $\mu : \mathcal{X} \rightarrow \mathcal{A}$ *est une fonction assignant les variables à leurs agents respectifs. Une* solution *à un DCOP est une affectation* $\mathcal{X}^* = \{x_1^*, \ldots, x_N^*\}$ *à chaque variable qui maximise la somme globale des utilités* [1] *:*

$$\sum_{m=1}^{M} u_m(\mathcal{X}_m) \tag{1}$$

Comme souligné par [1, 4], les DCOPs ont été largement étudiés, appliqués à des domaines de références, et exhibent des propriétés très intéressantes : (i) un focus important sur des approches décentralisées où les agents négocient une solution conjointe par des protocoles d'envoi de messages ; (ii) des techniques de résolu-

1. La notion d'utilités peut être remplacée par celle de coûts $\in \mathbb{R} \cup \{+\infty\}$. Dans ce cas, résoudre un DCOP consiste à minimiser la somme globale des coûts.

tion exploitant la structure du domaine (en l'encodant comme des contraintes) pour s'attaquer à des problèmes difficiles ; (iii) un très large choix de méthodes de résolution, complètes ou approchées.
Un DCOP dont toutes les contraintes sont binaires (ne font appel qu'à deux variables) peut être représenté comme un graphe de contraintes, où les nœuds représentent les variables, et les arcs représentent les contraintes.Dans le cas de contraintes n-aires, un DCOP peut être représenté par un graphe de facteurs (voir Figure 1).

Définition 2. *Le graphe de facteurs d'un DCOP, comme défini dans la Définition 1, est un graphe bipartite* $FG = \langle \mathcal{X}, C, E \rangle$*, où l'ensemble des nœuds variables correspond à l'ensemble* $\mathcal{X}$*, l'ensemble des nœuds facteurs correspond à l'ensemble des contraintes* C*, et l'ensemble des arcs est* $E = \{e_{ij} \mid x_i \in \mathcal{X}_j\}$.

Lorsque le graphe sous-jacent du DCOP possède au moins un cycle, il est dit *cyclique* ; sinon il est *acyclique*.
Une très large littérature aborde le sujet des algorithmes de résolution de DCOPs qui tombent dans deux catégories. D'un côté les *algorithmes complets* de recherche comme ADOPT et ses extensions [11], ou d'inférence comme DPOP [16] ou ActionGDL [19], sont optimaux, mais souffrent principalement de charges mémoire et/ou de communication élevées (e.g. mémoire exponentielle pour DPOP et communication exponentielle pour ADOPT) – ce qui ne peut ne pas être applicable dans la plupart des cas pratiques ou contraints, comme les réseaux de capteurs. D'un autre côté, les *algorithmes approchés* comme Max-Sum [3] ou MGM [10] ont le grand avantage d'être rapides et de ne nécessiter que peu de mémoire et de messages, au prix d'une perte d'optimalité dans certaines configurations – e.g. Max-Sum est optimal sur des DCOPs acycliques, mais peut ne pas converger dans certains cas difficiles.
Les algorithmes DCOP exploitent principalement le fait que les utilités (ou les coûts des contraintes) dépendent uniquement d'un sous-ensemble limité de variables de décisions, et que la fonction d'utilité globale est la somme des utilités individuelles. Ici, nous nous intéressons plus particulièrement au cas des algorithmes par propagation de croyances (comme Max-Sum), où la notion de valeur marginale (ou fonction marginale) décrit l'influence des valeurs d'une variable sur la fonction globale.

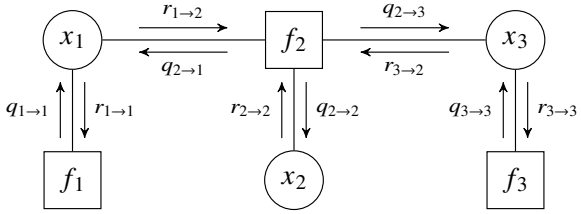


FIGURE 1 – Exemple de graphe de facteurs et de flux de messages de propagation de croyances.

2.2 De BP à Max-Sum

La propagation de croyances (ou BP, pour *belief-propagation*), correspondant initialement à la méthode par envoi de messages nommée *sum-product*, est un algorithme potentiellement distribuable pour effectuer des inférences sur modèles graphiques (e.g. réseaux bayésiens), et peut fonctionner sur des graphes de facteurs représentant un produit de M facteurs [9] : $F(x) = \prod_{m=1}^{M} f_m(\mathcal{X}_m)$. L'algorithme sum-product fournit une procédure locale par envoi de messages efficace pour calculer les fonctions marginales de toutes les variables simultanément. La fonction marginale $z_n(x_n)$ décrit l'influence de la variable x_n sur la fonction globale $F(x)$: $z_n(x_n) = \sum_{\{x'\}, n' \neq n} F(\mathcal{X}_{n'})$.
BP effectue une propagation itérative de messages notés généralement $m_{i\to j}$ (i.e. tables associant une valeur marginale à chaque valeur possible pour une variable) suivant les arcs du graphe de facteurs, comme illustré dans la figure 1. Ces messages sont soit émis des facteurs vers les variables ($q_{i\to j}$) soit des variables vers les facteurs ($r_{j\to i}$). Lorsque le graphe de facteurs est un arbre (acyclique donc), BP calcule les valeurs marginales exactes et converge en un nombre fini d'étapes de propagation dépendant du diamètre du graphe [9]. *Max-product* est une alternative à *sum-product* qui calcule la valeur maximale plutôt que la somme des marginales. Conçu comme un dérivé de *max-product*, Max-Sum est un algorithme approché de résolution de DCOP [3]. La principale évolution est la façon dont les messages sont calculés, pour passer d'un produit de probabilités à une somme d'utilité (par transformation logarithmique). En conséquence, Max-Sum calcule une affectation $\mathcal{X}^*$ qui maximise l'objectif de l'équation 1. En fonction du DCOP à résoudre, Max-Sum peut être utilisé avec deux règles d'arrêt : (i) continuer jusqu'à convergence (plus aucun message échangé, car lorsque nœuds variables et nœuds facteurs reçoivent deux fois le même message à la suite du même émetteur, ils ne propagent plus) ; (ii) propager les messages un nombre

fixé d'itérations par agent. Max-Sum est optimal sur graphes de facteurs acycliques, est se comporte très bien dans certaines configurations cycliques. Mais il y a certains problèmes pour lesquels il ne converge pas ou converge vers un état sous-optimal. En effet, dans certaines configurations cycliques, [3] identifient les comportements suivants : (i) les agents convergent vers des états stables qui représentent soit leurs solution optimale, soit une solution proche de l'optimale, alors que la propagation a cessé ; (ii) les agents convergent comme précédemment, mais les messages continuent à changer légèrement à chaque envoi, et ainsi continuent à propager dans le réseau ; (iii) ni l'état des agents, ni l'envoi de messages ne convergent, exhibant un comportement répétitif.
Afin d'améliorer les performances de Max-Sum sur des DCOP cycliques, [20] ont proposé deux extensions : (i) Max-Sum_AD qui exécute Max-Sum sur un graphe dirigé acyclique construit à partir du graphe de facteurs originel, et qui alterne les direction des arcs à fréquence fixée (c'est un paramètre de l'algorithme) ; (ii) Max-Sum_AD_VP qui exécute Max-Sum_AD et propage les valeurs courantes des variables en même temps que les messages Max-Sum classiques de telle sorte que les facteurs ne considèrent que la valeur courante d'une variable au lieu de considérer toutes les valeurs possibles du domaine de la variable. Ces deux extensions, surtout la seconde, améliorent grandement la qualité des solutions obtenues : Max-Sum_AD_VP trouve des solutions qui approchent l'optimum à 10% en moyenne. Cependant, l'étude ne considère pas le nombre de messages requis, ou le temps pour converger et permettre à Max-Sum_AD(_VP) de terminer.

2.3 Décimation guidée par BP

Nous proposons de nous inspirer de travaux issus de la physique computationelle [14], afin de faire face à la cyclicité des DCOPs. [7] ont notamment introduit la notion de *décimation* en satisfaction de contraintes, et plus précisément en k-SAT, où les variables sont binaires, $x_i \in \{0, 1\}$, et chaque contrainte nécessite que les valeurs de k variables soient différentes d'un k-uple spécifique. Les auteurs proposent une classe d'algorithmes, appelée *procédure de décimation guidée par envoi de messages*, qui consiste en la répétition des étapes suivantes : (1) exécuter un algorithme par envoi de messages, comme BP ; (2) utiliser le résultat pour choisir une variable x_i, et une valeur x_i^* pour cette variable ; (3) remplacer le problème de satisfaction de contraintes avec celui obtenu en fixant la valeur de x_i à x_i^*.

Algorithme 1 : Décimation guidée par BP [12]

Données : Un graphe de facteurs représentant un problème k-SAT
Résultat : Une affectation réalisable $\mathcal{X}^*$ ou `FAIL`

```
initialiser les messages BP
U ← ∅
pour t = 1, ..., n faire
    exécuter BP jusqu'à ce que le critère de terminaison soit valide
    choisir x_i ∈ X \ U de manière aléatoire uniforme
    calculer la valeur marginale z_i(x_i)
    choisir x_i^* conformément à la distribution induite par z_i
    fixer x_i = x_i^*
    U ← U ∪ {x_i}
    simplifier le graphe de facteurs
    si contradiction alors retourner FAIL
retourner X^*
```

La procédure de décimation quidée par BP est présentée dans l'algorithme 1, dont les performances sont analysées dans [12, 14].
La décimation guidée par BP s'exécute sur un graphe de facteur représentant le problème k-SAT à résoudre. À chaque pas, la variable à décimer est choisie de manière aléatoire parmi les variables non décimées. La valeur de la variable choisie x_i est déterminée par un tirage aléatoire dont la distribution correspond à sa fonction marginale z_i. Après décimation, le graphe de facteur est réduit : des arcs sont supprimés, et des facteurs sont simplifiés (les facteurs étant des tableaux, des colonnes correspondant à la variable décimées sont supprimées). Dans certaines configurations, la décimation guidée par BP peut échouer, si les choix aléatoires affectent une valeur à une variable qui n'est pas consistante avec les autres variables décimées.
Notons que le fait de *se baser sur les valeurs marginales* est au cœur de la nature « guidée par BP » de cette méthode. Les valeurs marginales sont exploitées pour élaguer le graphe de facteurs. De plus, alors que dans les travaux initiaux de [12], cette procédure est utilisées pour résoudre des problèmes k-SAT, cette approche peut facilement être mise en œuvre pour s'attaquer à des problèmes d'optimisation. Pas exemple, la bibliothèque de méthode d'inférence libDAI propose une implémentation de la décimation pour l'inférence discrète approché dans les modèles graphiques [13].

2.4 État d'un graphe de facteurs

L'algorithme précédent s'exécute sur un graphe de facteurs représentant le problème.

S'« exécute » signifie que l'algorithme crée une structure de donnée représentant le graphe de facteurs qui évolue avec le temps : les valeurs marginales changent, des variables disparaissent, des messages sont envoyés/reçus, etc. Généralement la représentation d'un graphe de facteurs est un ensemble de nœuds connectés en fonction du problème. Chacun de ces nœuds a un état qui stocke certaines valeurs utiles.
Notons, FG^t l'état courant au temps t du graphe de facteurs $FG = \langle \mathcal{X}, C, E \rangle$ est la composition de tous les états courants des structures de données utilisées par l'algorithme basé sur BP pour s'exécuter sur le graphe de facteurs, notamment les valeurs marginales z_i, les messages $m_{i \to j}$, l'ensemble des variables décimées $\mathcal{U}$, et tout autre donnée spécifique à l'algorithme.
Nous pouvons considérer que pour un problème donné, plusieurs états de graphe de facteurs peuvent exister. Nous notons $\mathfrak{S}$ l'ensemble des états de graphe de facteurs possibles, et $\mathfrak{S}(FG) \subset \mathfrak{S}$ l'ensemble des états possibles pour le graphe FG.

3 DeciMaxSum : étendre Max-Sum avec la décimation

Alors que cette méthode a été initialement conçue pour des problèmes k-SAT, la décimation guidée par BP peut être utilisée pour résoudre des DCOP avec très peu de modifications. Dans la limite de nos connaissances, cette approche, que nous appelons DeciMaxSum, n'a jamais été proposée pour améliorer les performances de Max-Sum.

3.1 Principes

Notre idée est d'étendre et généraliser la décimation guidée par BP de [12] afin de définir un cadre plus général dans lequel les autres algorithmes basé sur BP pourront s'intégrer. Tout d'abord, l'aspect principal est la *décimation*, ce qui signifie assigner une valeur à une variable afin de la supprimer du problème. Comme le nom le suggère, il n'y a pas de retour en arrière possible lorsqu'une variable a été décimée – contrairement aux algorithmes de recherche, où les affectations de valeurs aux variables sont révisées par des mécanismes comme le *backtrack*. Par conséquent, déclencher la décimation est une décision importante. C'est pourquoi, notre cadre est principalement abordé suivant trois questions décisives : (i) Quand déclencher la décimation ? (ii) Quelles variables décimer ? (iii) Quelles valeurs affecter aux variables décimées ?
Plusieurs critères de décision peuvent être définis pour répondre à ces questions, et DeciMaxSum spécifie ces critères par des *politiques de décimation*, qui sont des paramètres fondamentaux de la procédure.

Définition 3 (Politique de décimation)**.** *Une* politique de décimation *est un tuple* $\pi = \langle \Theta, \Phi, \Upsilon, \Lambda \rangle$ *où :*

- $\Theta : \mathfrak{S} \to \{0, 1\}$ *est la condition de déclenchement de la décimation (ou* politique de déclenchement*),*
- $\Phi : \mathfrak{S} \to 2^{\mathcal{X}}$ *est une* politique de filtrage *qui sélectionne des variables candidates à la décimation,*
- $\Upsilon : \mathcal{X} \times \mathfrak{S} \to \{0, 1\}$ *est une condition pour effectuer la décimation sur une variable, appelée* politique de condition*,*
- $\Lambda : \mathcal{X} \times \mathfrak{S} \to \mathcal{D}_{\mathcal{X}}$ *est une* politique d'affectation*, qui assigne une valeur à une variable donnée.*

Une grande famille d'algorithmes basés sur la décimation peut être construite dans ce cadre par combinaison de politiques. Par exemple, nous pourrions considérer une instance de DeciMaxSum qui (i) déclenche la décimation une fois que BP a convergé, (ii) choisit aléatoirement la variable à décimer parmi tout l'ensemble des variables non décimées, et (iii) qui tire la valeur de la variable décimée au hasard en fonction de ses valeurs marginales. Ce faisant, nous obtenons l'algorithme classique de décimation guidée par BP de [12]. Cependant, autant d'autres politiques de décimation peuvent être définies et combinées.

3.2 DeciMaxSum comme algorithme

Nous pouvons résumer le fonctionnement de DeciMaxSum par l'algorithme 2. C'est une reformulation de l'algorithme 1, paramétrée par une politique de décimation. Ici, la décimation n'est pas nécessairement déclenchée à convergence de BP. Le critère Θ peut se baser sur d'autres composants de l'état du graphe de facteur. Contrairement à la décimation guidée par BP classique, il peut y avoir plusieurs variables décimées en même temps et ces variables peuvent être sélectionnées de manière informée non aléatoire, en utilisant le critère Υ. Les valeurs affectées aux variables décimées ne sont pas nécessairement déterminées de manière stochastique, mais sont affectées grâce à la fonction Λ qui peut être déterministe (tout en dépendant de l'état du graphe de facteurs). Comme, ici nous ne sommes pas dans le cas k-SAT, mais dans le cadre de l'optimisation, il n'y a pas de cas d'échec, mais une éventuelle sous-optimalité. Enfin, une fois toutes

Algorithme 2 : DeciMaxSum

Données : $FG = \langle \mathcal{X}, C, E \rangle$, $\pi = \langle \Theta, \Phi, \Upsilon, \Lambda \rangle$
Résultat : $\mathcal{X}^*$

```
initialiser les messages BP
U ← ∅
tant que U ≠ X faire
    exécuter BP jusqu'à déclenchement de la décimation,
      i.e. Θ(FG^t) = 1                          // Sect. 3.3
    choisir les variables à décimer
      X' = {x_i ∈ Φ(FG^t) | Υ(x_i, FG^t)}       // Sect. 3.4
    pour x_i ∈ X' faire                         // Sect. 3.5
        x_i ← Λ(x_i, FG^t)
        U ← U ∪ {x_i}
        simplifier FG^t        // supprimer les variables,
          couper les facteurs
retourner X* en décodant U
```

les variables décimées, la sortie de l'algorithme consiste à décoder l'état courant du graphe de facteur FG^t, i.e. récupérer de manière distribuée les valeurs des variables. Ceci signifie que finalement DeciMaxSum effectue la phase de décodage habituel en propagation de croyance tout en résolvant le problème, contrairement aux algorithmes classique de résolution de DCOP comme Max-Sum ou DSA, qui nécessite une phase supplémentaire après résolution pour extraire la solution à partir des valeurs marginales.
Bien que présenté comme un algorithme classique, notons que la décimation a pour but d'être effectuée de manière distribuée et concurrente, en fonction des composants nécessaires à la politique de décimation.

3.3 Déclencher la décimation (critère Θ)

Dans l'approche originale de [12], la décimation est déclenchée une fois que la propagation de croyances a convergé. Dans un contexte distribué avec des algorithmes de diffusion comme BP, ceci peut être mis en œuvre en utilisant des mécanismes de détection de terminaison ou de silence (*quiescence*).

$$\Theta_{\texttt{converge}}(s) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{si } s \text{ est } \textit{quiescent} \\ 0, & \text{sinon} \end{cases} \tag{2}$$

Ce déclenchement consiste à détecter le silence de l'état courant de graphe de facteurs. Ceci signifie qu'aucune action locale n'est en cours et qu'aucun message de propagation n'est en cours de diffusion [8]. Des algorithmes comme *DijkstraScholten* peuvent détecter un tel état global en mettant en œuvre un protocole d'envoi/réception de tokens opéré sur le même graphe que FG [8]. Notons que de telles techniques génèrent une charge réseau supplémentaire à cause des messages dédiés à la détection de terminaison.

A cause du comportement de Max-Sum sur des graphes cycliques, la convergence peut ne pas survenir [20]. La solution de contournement classique est d'exécuter BP pour un nombre d'itérations fixé, au lieu d'attendre la stabilité du système. Fixer cette limite de temps (appelée `LIMIT`) dépend fortement de la nature du problème à résoudre.

$$\Theta_{\texttt{time}}(s) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{si } \textit{time}(s) = \texttt{LIMIT} \\ 0, & \text{sinon} \end{cases} \tag{3}$$

Dans des configurations synchrones (toutes les variables et les facteurs sont exécutés de manière synchrone, pas à pas), le numéro d'itérations de l'état courant du graphe de facteurs, $\textit{time}(s)$, peut être obtenu de manière distribuée, comme cela est habituellement fait dans Max-Sum. Dans le cas asynchrone, on peut soit (i) utiliser une horloge partagée, soit (ii) compter de manière locale les messages sortants au sein de chaque variable, et une fois que la variable a envoyé un nombre limite de messages, la décimation est déclenchée.
Dans certaines configurations très contraintes par le temps ou les capacités de calculs (e.g. réseaux de capteurs [3], ou l'Internet des objets [18]), on ne peut se permettre d'attendre la convergence. En effet, BP peut générer un grand nombre de messages. Par conséquent, on peut considérer une décimation avant convergence, à fréquence régulière (e.g. toutes les 10 itérations), ou en utilisant un budget fixé d'itérations partagé entre les variables (e.g. toutes les 1000 itérations divisées par le nombre de variables). Nous pouvons également considérer une vitesse variable de décimation (e.g. rapide au début, puis plus lente à la fin, comme certains phénomènes observés dans les réseaux de neurones [15]).

$$\Theta_{\texttt{frequency}}(s) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{si } \textit{time}(s) \bmod f(s) = 0 \\ 0, & \text{sinon} \end{cases} \tag{4}$$

où f est une fonction de l'état courant du graphe de facteurs, par exemple :

- $f(s) = \texttt{RATE}$, avec une fréquence fixée de décimation,
- $f(s) = \texttt{BUDGET}/|\mathcal{X}|$, avec un budget prédéfini de calcul,
- $f(s) = 2 \times \textit{time}(s)$, pour une fréquence de décimation décroissante.

Enfin, une autre approche pourrait être de déclencher la décimation à chaque détection de cycle. En effet, la décimation est utilisée pour résoudre les problèmes a priori dus aux cycles, ainsi décimer des variables identifiées comme

appartenant à des cycles semble être une bonne approche.

$$\Theta_{\texttt{loop}}(s) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{si } \exists x_i \in \mathcal{X}, |loop(x_i)| > 1 \\ 0, & \text{sinon} \end{cases} \tag{5}$$

où $loop(x_i)$ est l'ensemble des agents dans le même cycle que x_i vient juste de découvrir. Détecter des cycles dans le graphe de facteurs peut être mis en œuvre dans BP, en ajoutant des informations supplémentaires dans les messages de propagation de croyances.

3.4 Décider des variables à décimer (critères Φ et Υ)

Une fois que le système a déterminé que la décimation devait être déclenchée, la question suivante est de savoir quelles variables décimer. Dans [12], une variable est choisie de manière aléatoire uniforme, Alors que dans [13], c'est la variable avec l'entropie maximale sur ses valeurs marginales (la variable la plus *déterminée*) qui est sélectionnée. Exploiter les valeurs marginales, construites tout au long du processus de propagation semble être une bonne direction.

Sous-ensemble des variables à décimer. [12] et [13] sélectionnent une seule variable parmi tout l'ensemble des variables non décimées (cf. ligne 5 dans l'algorithme 1). Ainsi, le critère Φ peut être spécifié comme suit :

$$\Phi_{\texttt{all}}(s) \stackrel{\text{def}}{=} \mathcal{X} \setminus \mathcal{U} \tag{6}$$

Cependant, cette sélection sur tout l'ensemble des variables peut être discuté, lorsque le critère de déclenchement est local, comme lors de la détection de cycles. Dans un tel cas, sélectionner les variables parmi les agents impliqués dans la boucle, ou uniquement celui qui la détecte en premier semble plus approprié. Une autre approche est de considérer la sélection de variables en fonction d'états passé du graphe de facteurs. Par exemple, si une variable a été décimée, de bonnes candidates à la décimation pourrait être ses voisines directes dans le graphe de facteurs :

$$\Phi_{\texttt{neighbors}}(s) \stackrel{\text{def}}{=} \{x \in \mathcal{X} \setminus \mathcal{U} \mid neighbors(x) \cap \mathcal{U} \neq \emptyset\} \tag{7}$$

avec
$neighbors(x_i) = \{x_j \in \mathcal{X} | j \neq i, \exists e_{ik}, e_{kj} \in E\}$

Critère pour déclencher la décimation. Maintenant, nous devons spécifier le critère Υ utilisé pour décider quelles candidates décimer. Dans [12], ceci est déterminé de façon aléatoire : cela ne dépend pas de l'état courant des variables. Ceci correspond à faire tirer un nombre aléatoire à chaque variable, et choisir la variable ayant obtenu le plus haut résultat :

$$\Upsilon_{\texttt{max\_rand}}(x_i, s) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{si } \forall x_j \neq x_i \in \mathcal{X}, rand(x_i) > rand(x_j) \\ 0, & \text{sinon} \end{cases} \tag{8}$$

où $rand(x)$ représente la sortie d'un générateur de nombres aléatoires (appelé $sample$) suivant une distribution uniforme (e.g. $U[0, 1]$).
Dans [13], c'est la variable ayant l'entropie sur ses valeurs marginales maximale qui est sélectionnée. Ceci signifie que la variable dont les valeurs marginales semblent les plus informées, au sens de Shannon [9], est choisie :

$$\Upsilon_{\texttt{max\_entropy}}(x_i, s) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{si } \forall x_j \neq x_i \in \mathcal{X}, H(z_i(x_i)) > H(z_j(x_j)) \\ 0, & \text{sinon} \end{cases} \tag{9}$$

avec

$$H(z_k(x_k)) = -\sum_{d \in \mathcal{D}_k} z_k(x_k)(d) \log(z_k(x_k)(d))$$

À partir de là, plusieurs autres critères peuvent être dérivés. Par exemple, au lieu d'utiliser l'entropie, nous pourrions considérer la valeur marginale normalisée maximale :

$$\Upsilon_{\texttt{max\_marginal}}(x_i, s) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{si } \forall x_j \neq x_i \in \mathcal{X}, \\ & \max_{d \in \mathcal{D}_i}(z_i(x_i)(d)) > \max_{d \in \mathcal{D}_j}(z_j(x_j)(d)) \\ 0, & \text{sinon} \end{cases} \tag{10}$$

Si plusieurs variables peuvent être décimées au même moment, nous pourrions également considérer la sélection de l'ensemble des variables une entropie ou une valeur marginale normalisée maximale supérieure à un certain seuil à déterminer, pour décimer les variables « suffisamment » déterminées. Par conséquent, cette approche nécessite un autre paramètre (appelé `THRESHOLD`) :

$$\Upsilon_{\texttt{threshold\_entropy}}(x_i, s) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{si } H(z_i(x_i)) > \texttt{THRESHOLD} \\ 0, & \text{sinon} \end{cases} \tag{11}$$

Bien sûr, de nombreuses combinaisons des critères précédents, ou d'autres critères, peuvent

être considérée dans le cadre DeciMaxSum. Nous ne discutons pas ici de critères comme dans DSA qui ne dépendent pas de valeurs marginales, mais de décisions stochastiques [10].

Sous-ensemble de variables dont la décimation dépend. Ici se pose le problème de la coordination de la sélection de variables. En effet, si le calcul du critère Υ ne dépend pas de la décision des autres variables, la procédure est complètement distribuable à peu de coût de communication, comme pour le critère (11). Au contraire, si la décision nécessite d'avoir connaissance de l'état des autres variables, comme pour (8), (9) et (10), la procédure nécessitera des messages de coordination à l'échelle du système, ou d'un sous-ensemble d'agents. Dans [12] et [13], la décimation concerne toutes les variables, à partir desquelles une seule sera sélectionnée. Ceci requiert une coordination globale, ou un protocole d'élection de leader qui peut nécessiter la mise en place d'un réseau sous-jacent (e.g. un anneau ou un arbre couvrant), comme celui utilisé pour la détection de *quiescence* pour propager les messages d'élection [8].

Dans certains cas, la décision de décimation peut se faire à échelle locale, quand les variables prennent leur décision en fonction des voisins directs, ou des variables dans la même boucle. Dans ce cas, moins de messages de coordination sont nécessaires. Par exemple, si l'on considère la décimation de variables dans un cycle, seules les variables de ce cycle vont mettre en œuvre le protocole d'élection de leader. Tous les critères de (8) à (10), pourraient être étendus de la même manière, en remplaçant $\mathcal{X}$ par $loop(x_i)$, $neighours(x_i)$, ou n'importe quel sous-ensemble de $\mathcal{X}$. Par exemple :

$$\Upsilon_{\texttt{max\_rand\_loop}}(x_i, s) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{si } \forall x_j \neq x_i \in loop(x_i), \\ & rand(x_i) > rand(x_j) \\ 0, & \text{sinon} \end{cases} \quad (12)$$

3.5 Décider des valeurs à affecter aux variables décimées (critère Λ)

Les variables à décimer étant identifiées, il faut choisir quelles valeurs leur attribuer. Habituellement, dans les algorithmes basés sur BP, la façon la plus simple de sélectionner les valeurs des variables après propagation est de prendre la valeurs maximisant la valeur marginale (ou l'utilité). [13] utilise un tel critère pour l'inférence sur modèles graphiques :

$$\Lambda_{\texttt{max\_marginal}}(x_i, s) \stackrel{\text{def}}{=} \operatorname*{argmax}_{d \in \mathcal{D}_i} z_i(x_i)(d) \quad (13)$$

Alors que ce critère est déterministe, dans [12] le choix de la valeur est aléatoire avec distribution de probabilité équivalente aux valeurs marginales :

$$\Lambda_{\texttt{max\_marginal}}(x_i, s) \stackrel{\text{def}}{=} sample(z_i(x_i)) \quad (14)$$

Une fois encore, ce ne sont que quelques exemples de critères exploitant BP, et il est possible d'en créer d'autres.

4 Expérimentations

Nous évaluons les performances des différentes combinaisons de politiques de décimation dans DeciMaxSum, comparées à Max-Sum [3] et son extensions Max-Sum_AD_VP [20], que nous avons implanté dans notre cadre expérimental.

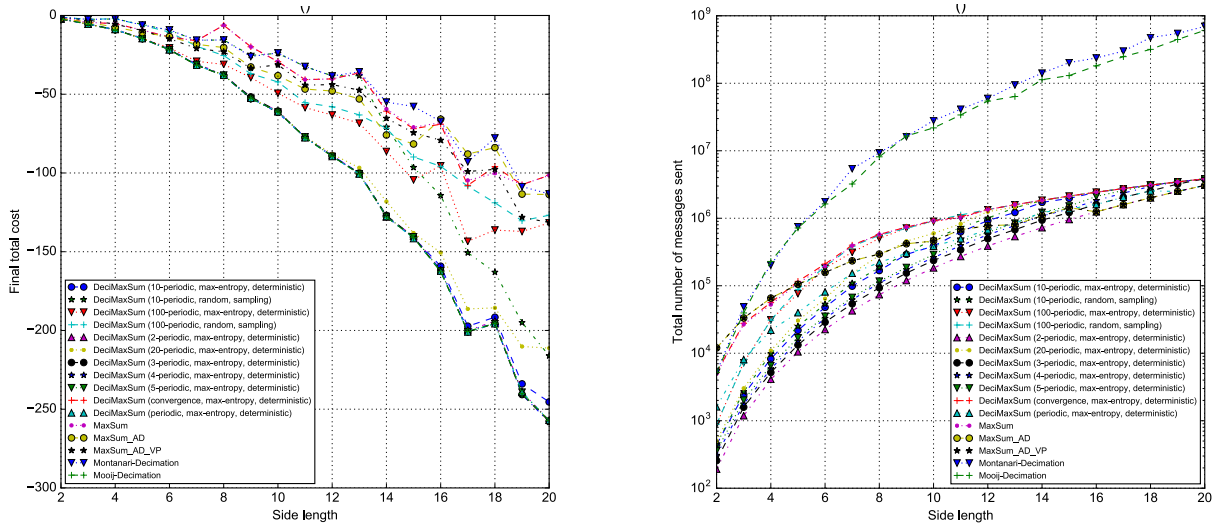
Comme nous sommes intéressés par évaluer notre approche en la présence de fortes dépendances entre les valeurs des variable du DCOP, nous l'évaluons sur le modèle Ising, très cyclique, qui est un étalon reconnu en physique statistique [6]. Ici, le graphe de facteurs est une grille rectangulaire où chaque variable binaire x_i est connectée à ses quatre variables voisines les plus proches (avec liaisons toriques connectant les côtés opposés de la grille), et sont contraintes par un coût unaire r_i. Le poids de chaque contrainte binaire r_{ij} est déterminé par tirage uniforme d'une valeur κ_{ij} dans $U[-\beta, \beta]$, puis en affectant

$$r_{ij}(x_i, x_j) = \begin{cases} \kappa_{ij} & \text{si } x_i = x_j \\ -\kappa_{ij} & \text{sinon} \end{cases}$$

où β contrôle la force moyenne des interactions entre variables. Ici, nous avons fixé $\beta = 1.6$. Le poids des contraintes unaires r_i est déterminé par tirage uniforme de κ_i dans $U[-0.05, 0.05]$, puis en affectant $r_i(0) = \kappa_i$ et $r_i(1) = -\kappa_i$.

Dans cette section nous analysons les résultats de différentes combinaisons de politiques DeciMaxSum pour résoudre des problèmes d'Ising carrés de côté 10 à 20 (e.g. 100 à 400 variables). Nous avons implanté les algorithmes de l'état de l'art suivants : MaxSum [3], MaxSum_AD, as defined in [20], MaxSum_AD_VP, as defined in [20], Montanari-Decimation, as defined in [12], Mooij-Decimation, as defined in [13]. Pour DeciMaxSum, nous avons implanté les combinaisons suivantes :

— critères de déclenchement (Θ) : `converge`, `frequency` basé sur une fréquence fixée (noté 2-periodic, 3-periodic, 5-periodic, 10-periodic, 20-periodic, et 100-periodic), `frequency` basée sur un budget total (noté periodic),



(a) Coût de la solution finale

(b) Nombre total final de messages

FIGURE 2 – Performances de DECIMAXSUM et d'autres méthodes de résolution sur des problèmes de Ising (moyenne sur 10 problèmes par paramètre de génération d'instance, et moyenne sur 3 résolutions par instance de problèmes).

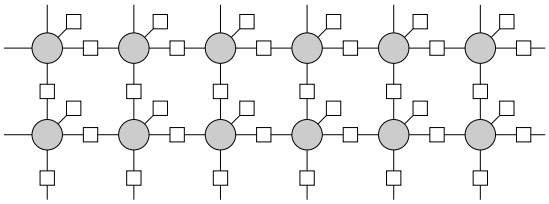


FIGURE 3 – Un exemple de modéle d'Ising 6×2 : les variables x_i en gris et les facteurs unaires (r_i) et binaires (r_{ij}) en blanc

— toutes les variables sont considérées comme candidates (critère $\Phi_{\texttt{all}}$),

— critères de sélection des variables (Υ) : `max_rand` (noté random) et `max_entropy` (noté max_entropy),

— critères de sélection de valeurs (Λ) : `deterministic` et `sampling`,

La figure 2 présente deux métriques de performance des algorithmes (le coût final de la solution obtenue et le nombre total de messages échangés lors de la résolution). Du point de vue de l'optimalité des solutions finales obtenues par les différents algorithmes et les différentes instances de DECIMAXSUM, il apparaît qu'une décimation très rapide combinée avec une sélection déterministe de la variable la plus déterminée (max_entropy) permet d'obtenir les meilleurs résultats. En effet, les décimations periodic, 2-periodic, 3-periodic, 5-periodic, et 10-periodic avec choix de la variable avec l'entropie maximale et choix de valeur avec marginale la plus élevée produisent toutes des solutions avec des coûts finaux deux fois moins élévés que les solutions de l'état de l'art (Montanari-Decimation, Mooij-Decimation et les variantes de Max-Sum). Concernant les coûts de communication, une décimation très rapide dans DECIMAXSUM implique que peu de messages sont échangés, en comparaison des autres méthodes de décimation (Montanari-Decimation et Mooij-Decimation), car il n'y a pas d'attente de convergence avec le déclenchement de la décimation. Cependant, toutes les autres méthodes de résolution tendent vers un nombre comparable de messages échangés. En résumé, nos premiers résultats sont très prometteurs et semblent indiquer que DECIMAXSUM avec une décimation guidée par valeur marginale rapide et déterministe fournit une très bonne qualité, sans coût supplémentaire de communication, sur des DCOP avec une structure régulière comme Ising.

5 Conclusions

Dans cet article, nous avons étudié comment étendre Max-Sum pour résoudre des problèmes d'optimisation de contraintes distribuées en s'inspirant des mécanismes de décimation utilisés pour résoudre les problèmes k-SAT par propagation de croyances. Nous proposons une méthode paramétrique, à savoir DECIMAXSUM, qui peut être configurée avec différentes politiques de décimation pour décider quand déclencher la décimation, quelles variables décimer, et quelle valeur attribuer aux variables dé-

cimées. Dans cet article, nous proposons un catalogue de politiques qui peuvent être combinées pour produire différentes versions de DeciMaxSum. Nos résultats empiriques sur un benchmark classique montrent que certaines combinaisons de politiques de décimation surpassent l'algorithme Max-Sum classique et son extension, Max-Sum_AD_VP, spécifiquement conçue pour gérer les boucles. DeciMaxSum donne des solutions de meilleure qualité avec une quantité raisonnable de propagation de message.
Nous identifions plusieurs pistes de travaux futurs. Premièrement, nous n'avons exploré qu'un ensemble limité de politiques de décimation. Nous souhaitons étudier des politiques plus complexes, en particulier les politiques qui déclenchent la détection des boucles par les agents. En effet, puisque notre objectif principal est de faire face aux boucles, les détecter au niveau de l'agent semble une approche raisonnable pour initier la décimation dans un réseau cyclique. Cette approche nécessitera que les agents mettent en œuvre un protocole de détection de cycles, en envoyant l'historique des messages, tout en propageant les valeurs marginales. Dans un tel contexte, plusieurs choix de décimation peuvent se produire simultanément dans le système. Deuxièmement, nous aimerions généraliser le cadre DeciMaxSum pour considérer Max-Sum_AD_VP comme un cas particulier de décimation : la décimation itérée. Enfin, nous prévoyons d'appliquer DeciMaxSum sur les applications du monde réel, avec une nature fortement cycliques, comme la coordination d'objets intelligents dans l'IoT [18] ou des marchés décentralisés de l'énergie dans les smart grids [2].

Références

[1] J. Cerquides, A. Farinelli, P. Meseguer, and S. D. Ramchurn. A tutorial on optimization for multi-agent systems. *The Computer Journal*, 57(6) :799–824, 2014.

[2] J. Cerquides, G. Picard, and J.A. Rodríguez-Aguilar. Designing a marketplace for the trading and distribution of energy in the smart grid. In *14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1285–1293, 2015.

[3] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS'08)*, pages 639–646, 2008.

[4] Ferdinando Fioretto, Enrico Pontelli, and William Yeoh. Distributed constraint optimization problems and applications : A survey. *CoRR*, abs/1602.06347, 2016.

[5] D. Koller and N. Friedman. *Probabilistic Graphical Models*. MIT Press, 2009.

[6] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10) :1568–1583, Oct 2006.

[7] F. Krzakala, A. Montanari, F. Ricci-Tersenghi, G. Semerjian, and L. Zdeborova. Gibbs states and the set of solutions of random constraint satisfaction problems. *Proceedings of the National Academy of Science*, 104 :10318–10323, June 2007.

[8] N.A. Lynch. *Disributed Algorithms*. Morgan Kaufmann, 1996.

[9] D. J. C. Mackay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.

[10] R.T. Maheswaran, J.P. Pearce, and M. Tambe. Distributed algorithms for dcop : A graphical-game-based approach. In *Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems (PDCS)*, pages 432–439, 2004.

[11] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. ADOPT : Asynchronous Distributed Constraint Optimization with Quality Guarantees. *Artificial Intelligence*, 161(2) :149–180, 2005.

[12] A. Montanari, F. Ricci-Tersenghi, and G. Semerjian. Solving constraint satisfaction problems through belief propagation-guided decimation. *CoRR*, abs/0709.1667, 2007.

[13] Joris M. Mooij. libDAI : A free and open source C++ library for discrete approximate inference in graphical models. *Journal of Machine Learning Research*, 11 :2169–2173, August 2010.

[14] M. Mézard and A. Montanari. *Information, Physics, and Computation*. Oxford University Press, 2009.

[15] Saket Navlakha, Alison L. Barth, and Ziv Bar-Joseph. Decreasing-rate pruning optimizes the construction of efficient and robust distributed networks. *PLOS Computational Biology*, 11(7) :1–23, 07 2015.

[16] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 266–271, 2005.

[17] A. Rogers, A. Farinelli, R. Stranders, and N.R. Jennings. Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence*, 175(2) :730 – 759, 2011.

[18] P. Rust, G. Picard, and F. Ramparany. Using message-passing DCOP algorithms to solve energy-efficient smart environment configuration problems. In *International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI Press, 2016.

[19] M. Vinyals, J.A. Rodriguez-Aguilar, and J. Cerquides. Constructing a unifying theory of dynamic programming dcop algorithms via the generalized distributive law. *Autonomous Agents and Multi-Agent Systems*, 22(3) :439–464, 2010.

[20] R. Zivan and H. Peled. Max/min-sum distributed constraint optimization through value propagation on an alternating DAG. In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes)*, pages 265–272. IFAAMAS, 2012.