Recherche incrémentale à base de points pour la résolution des DEC-POMDPs

J. S. Dibangoye[a,b] gilles.dibangoye@unicaen.fr
A.-I. Mouaddib[a] mouaddib@info.unicaen.fr
B. Chaib-draa[b] brahim.chaib-draa@ift.ulaval.ca

[a]Greyc - UMR 6072,
Université de Caen, France

[b]Département d'informatique,
Université Laval - Québec Q.C., Canada

Résumé

*Nous nous intéréssons au problème du contrôle d'un processus décisionnel de Markov décentralisé et partiellement observé (*DEC-POMDP*) à horizon fini. Nous introduisons une nouvelle approche heuristique qui s'appuie sur les observations suivantes : (1) l'opération élémentaire de programmation dynamique, consistant à la génération exhaustive et l'évaluation de toutes les politiques jointes, est extrêmement prohibitive ; (2) bon nombre des politiques jointes ainsi générées sont inutiles pour un contrôle optimal ou presqu'optimal. Suivant ces observations, nous proposons la première technique de construction incrémentale de politiques jointes à base d'états de croyance,* PBIP*, permettant d'éviter ces calculs intensifs. L'algorithme* PBIP *surpasse les performances des meilleurs techniques approximatives actuelles sur de nombreux exemples de la littérature.*

Mots-clés : *Planification décentralisée*

Abstract

*Recent scaling up of decentralized partially observable Markov decision process (*DEC-POMDP*) solvers towards realistic applications is mainly due to approximate methods. Of this family,* MEMORY BOUNDED DYNAMIC PROGRAMMING *(*MBDP*), which combines in a suitable manner top-down heuristics and bottom-up value function updates, can solve* DEC-POMDPs *with large horizons. The performance of* MBDP*, however, can be drastically improved by avoiding the systematic generation and evaluation of all possible policies which result from the exhaustive backup. To achieve that, we suggest a heuristic search method, namely* POINT BASED INCREMENTAL PRUNING *(*PBIP*) which is able to distinguish policies with different heuristic estimate. Taking this insight into account,* PBIP *searches only among the most promising policies, finds the useful, and prunes dominated ones. Doing so permits us to reduce clearly the amount of computation required by the exhaustive backup. A theoretical analysis of* PBIP *shows that it is both complete and optimal (with respect to* MBDP*). We also present experiment results, which show how* PBIP *outperforms the performance of* MBDP *and extensions on* DEC-POMDP *examples from the literature.*

Keywords: *Decentralized planning*

1 Introduction

De nombreux problèmes de planification qui impliquent deux ou plusieurs agents coopèrants afin d'optimiser une fonction de récompense, tout en ayant différentes observations locales, peuvent être modélisés comme des DEC-POMDPs. Ces problèmes apparaissent naturellement dans diverses disciplines, incluant l'informatique, par exemple le contrôle de plusieurs robots pour l'exploration de l'espace, l'économie, les chaînes d'approvisionnement décentralisées, ou la recherche opérationnelle, le trafic des réseaux de routage. Malheureusement, trouver une solution optimale ou ε-approximée à de tels problèmes s'est révélée particulièrement difficile [3, 7]. À ce jour, la plupart des algorithmes de résolution des DEC-POMDPs sont estimés incapables de faire face aux problèmes réels [5, 11, 12].

Il y a deux difficultés distinctes, mais étroitement liées, qui expliquent l'incapacité de ces algorithmes à passer à l'échelle. La plus connue des ces difficultés est nommée la *malédiction de la dimension* [2] : dans un problème avec n agents ; $|S|$ états physiques (pour chaque agent) ; et pour Q_{-i} l'ensemble des politiques jointes des partenaires de l'agent i, les algorithmes doivent raisonner sur un espace continu de dimension $|S^n \times Q_{-i}|$ qui croît doublement exponentiellement avec le nombre d'agents et le nombre d'observations. Cela s'explique par le fait que chaque agent doit raisonner sur la base des politiques des autres agents évaluées

sur l'espace joint des états physiques S^n. Cette contrainte indique en partie pourquoi la majorité des algorithmes est incapable de résoudre des DEC-POMDPs avec un grand nombre d'agents et quelques dizaines d'états. L'autre difficulté de la complexité des DEC-POMDPs est la *malédiction de l'historique* : où le nombre d'historiques joints croît doublement exponentiellement avec l'horizon de planification et le nombre d'agents [6]. De récentes tentatives ont été faites pour minimiser l'ensemble des historiques considérés [4, 8, 9, 11], mais jusqu'à présent, les meilleures techniques, y compris MBDP [9], restent toujours entravées par la malédiction de la dimension.

Pour les DEC-POMDPs à horizon fini, MBDP [9] est actuellement l'algorithme approximatif qui connaît le plus de succès. En comparaison à d'autres algorithmes de programmation dynamique (DP), MBDP dispose de deux avantages : d'une part, il est à mémoire limitée, *i.e.*, il ne requiert pas la mémorisation d'un nombre exponentiel de politiques mais seulement un nombre fixe noté $maxTrees$; d'autre part, il sélectionne suivant un ensemble d'heuristiques, ses états de croyance permettant de choisir les meilleurs politiques courantes. Toutefois, un des effets indésirables de cette stratégie est qu'elle nécessite l'énumération exhaustive de toutes les politiques jointes de l'itération courante construites sur la base des politiques jointes de l'itération précédente. Ceci est réalisé par le biais de l'opération dite de *sauvegarde exhaustive*. Malheureusement dans le pire des cas, le nombre de politiques jointes résultantes croît exponentiellement en fonction du nombre d'observations et du nombre d'agents. Ce problème nous pousse à introduire une nouvelle technique réduisant considérable la complexité de la sauvegarde exhaustive.

La principale contribution de ce papier est l'introduction d'une nouvelle technique de sélection des meilleures politiques jointes d'une itération donnée selon un ensemble fini d'états de croyance et les politiques jointes de l'itération précédente. Cette méthode vise à remplacer l'opérateur fondamental de tout algorithme de programmation dynamique à savoir la sauvegarde exhaustive. PBIP contourne le problème de la sauvegarde exhaustive en construisant *incrémentalement* les plus prometteuses des politiques jointes et en élaguant les politiques dominées. Ceci est réalisé au moyen d'une méthode triple : (1) nous identifions tout d'abord une bijection de l'espace des politiques jointes d'un DEC-POMDP vers un sous-espace des politiques du POMDP multi-agent sous-jacent (MPOMDP) ; (2) nous utilisons les états de croyance sélectionnés par chaînage-avant afin de déterminer les contributions des politiques jointes de l'itération précédente dans celles de l'itération courante ; (3) ces contributions sont par la suite utilisées comme estimations heuristiques afin de traverser l'espace exponentiel des politiques jointes du DEC-POMDP vers un espace plus petit correspondant aux politiques pertinentes du MPOMDP sous-jacent.

2 Contexte général

Dans cette section, nous présentons le modèle des DEC-POMDPs et certains des travaux de l'état de l'art.

2.1 Le cadre formel des DEC-POMDPs

Le cadre formel des DEC-POMDPs est un modèle généralisé pour des systèmes d'agents coopératifs, qui opèrent dans des domaines impliquant à la fois des états cachés mais aussi de l'incertitude sur les effets des actions. Un DEC-POMDP à n agents est un tuple $(I, S, \{A_i\}_i, P, R, \{\Omega_i\}_i, O, T, b_0)$.

- Soit $I = \{1, \cdots, n\}$ un ensemble fini d'agents.
- Soit S un ensemble fini d'états.
- Soit $A_i = \{a_1, a_2, \cdots\}$ un ensemble fini d'actions disponibles pour l'agent i, et $A = \otimes_{i \in I} A_i$ l'ensemble fini d'actions jointes a, où $a = (a^1, \cdots, a^n)$, la variable a^i dénote une action exécutée par l'agent $i \in I$.
- Soit $P(s'|s, a)$ la fonction de transition.
- Soit $R(s, a)$ la fonction de récompense.
- Soit $\Omega_i = \{o_1, o_2, \cdots\}$ un ensemble fini d'observations disponibles pour l'agent i, et $\Omega = \otimes_{i \in I} \Omega_i$ l'ensemble fini d'observation jointes $o \in \Omega$, où $o = (o^1, \cdots, o^n)$ et la variable o^i dénote une observation reçue par l'agent $i \in I$.
- Soit $O(o|a, s)$ la fonction d'observation.
- Soit T l'horizon de planification.
- Soit b_0 l'état de croyance initial du système.

Sachant un DEC-POMDP, nous avons pour objectif de trouver un vecteur de politiques dont la récompense à long terme, $E\left[\sum_{t=0}^{T-1} R(s_t, a_t) | \, b_0\right]$, est la plus grande pour l'état de croyance initial b_0. Une politique pour un seul agent, *arbre de politique*, peut-être représentée comme un arbre de décision désigné q_i, où les nœuds sont marqués par une

action $a_i \in A_i$ et les arêtes étiquettées par une observation $o_i \in \Omega_i$. Soit Q_i^t l'ensemble des politiques de l'agent i à l'horizon de planification t. Une solution à horizon t d'un DEC-POMDP peut alors être considérée comme un vecteur d'arbres de politiques à horizon t. On décrit une telle politique en utilisant un vecteur d'arbres de politiques $\vec{q}_t = (q_1^t, \cdots, q_n^t)$, un arbre par agent. L'ensemble des vecteurs d'arbres de politiques est noté $Q^t = \otimes_{i \in I} Q_i^t$. Nous définissons également $V(s, \vec{q}_t)$ comme la valeur espérée à l'état s suite à l'exécution du vecteur d'arbres de politiques $\vec{q}_t$.

$$V(s, \vec{q}_t) = \sum_o P(o|s, \vec{q}_t) \left[\sum_{s'} P(s'|s, \vec{q}_t, o) V(s', \eta(\vec{q}_t, o)) \right] \quad (1)$$

où $\eta(\vec{q}_t, o)$ est le vecteur d'arbres de politiques exécuté par les agents après la reception de l'observation jointe o. Nous utilisons $\alpha(\vec{q}_t)$ afin de désigner le nœud racine (action jointe) du vecteur d'arbres de politiques $\vec{q}_t$. Un vecteur d'arbres de politiques optimales $\vec{q}_{\star T} = (q_1^T, \cdots, q_n^T)$ pour un état de croyance initial b_0 peut alors être déterminé comme suit :

$$\vec{q}_{\star T} = \arg\max_{\vec{q}_T \in Q^T} \sum_s b_0(s) V(s, \vec{q}_T) \quad (2)$$

Un certain nombre d'algorithmes ont été proposés pour la construction de solutions optimales ou approximatives selon deux familles de stratégies : les méthodes de recherche par chaînage-avant [12] et celles par chaînage-arrière [5, 11]. Les difficultés du passage à l'échelle des méthodes précitées ont entraîné le développement d'une vaste variété d'algorithmes approximatifs. Parmi eux, MBDP [9] et ses extensions sont les seules méthodes capables de résoudre des DEC-POMDPs à des horizons élevés.

2.2 L'algorithme MBDP

MBDP [9] est un simple algorithme de programmation dynamique, qui maintient pour chaque agent un nombre limité d'arbres de politiques, *i.e.,* paramètre $maxTrees$. À chaque itération, MBDP sélectionne $maxTrees$ états de croyance en utilisant des heuristiques de recherche par chaînage-avant. Puis, il identifie le meilleur arbre de politiques jointes pour chaque état de croyance, utilisant l'ensemble des politiques jointes résultantes de la sauvegarde exhaustive. À l'itération T, la *meilleure* politique jointe, en fonction de l'état de croyance initial b_0 et des politiques jointes de l'itération précédente, est retournée. Cette combinaison des approches de chaînage-avant et chaînage-arrière est très efficace, et fait de MBDP une méthode différente des précédentes. Cependant, d'un point de vue pratique, MBDP dispose aussi bien de points forts que de points faibles. D'un côté, il réduit considérablement la complexité en pire cas des algorithmes de résolution de DEC-POMDPs notamment par rapport à l'horizon. Cependant, il utilise la sauvegarde exhaustive, et par conséquent, la manipulation et l'évaluation des politiques jointes nécessite un calcul intensif.

L'inconvénient majeur de MBDP est qu'il traverse l'espace exponentiel des politiques jointes afin de déterminer les meilleures d'entre elles selon un ensemble d'états de croyance. Pour mieux comprendre la complexité de MBDP, considérons $maxTrees$ le nombre de politiques sélectionnées pour chaque agent à une itération donnée. La première étape de cette méthode consiste à créer $|A_i| maxTrees^{|\Omega_i|}$ politiques par agent ; la deuxième évalue $\times_{i \in I}(|A_i| maxTrees^{|\Omega_i|})$ politiques jointes pour chaque état de croyance. Par conséquent, MBDP requiert par horizon un temps exponentiel $|S^2||A||\Omega| maxTrees^{|\Omega_i||I|+1}$, où $|S|$, $|\Omega|$ et $|A|$ croissent exponentiellement avec $|I|$. Plus important encore, MBDP procède ainsi même si très peu de ces politiques sont pertinentes pour la réalisation d'un comportement optimal ou presqu'optimal. Toutefois, très peu d'efforts ont été consacrés à l'exploiter cette idée. Actuellement, la quasi-totalité des algorithmes de résolution des DEC-POMDPs fait usage de la sauvegarde exhaustive. Fort de cette observation, nous voulons concevoir un algorithme général qui soit en mesure d'identifier les politiques potentiellement utiles de celles qui ne le sont pas de façon à éviter le problème de la sauvegarde exhaustive.

3 Recherche incrémentale

Dans cette section, nous décrivons une version naïve de la méthode proposée, la version optimisée et les principales priopriétés sont discutées plus en profondeur dans la section suivante. Tout d'abord nous allons introduire quelques définitions supplémentaires.

3.1 Espace de recherche

Notre objectif étant de déterminer un vecteur de politiques d'un DEC-POMDP, il paraît évident de faire de l'espace des vecteurs de politiques notre espace de recherche. Cependant, pour des raisons de mise en œuvre de notre approche nous

considérons une représentation legèrement différente de ces politiques. En effet, à la différence de la représentation classique (vecteur d'arbres de politiques), nous considérons un arbre de politique δ, dit *arbre-joint*, pour tout le groupe d'agents. Un arbre-joint est un arbre de décision où la racine $\alpha(\delta) = (\alpha(q_1), \cdots, \alpha(q_n))$ est marqué par une action jointe ; les arcs sont étiquettés par des observations jointes $o \in \Omega$; et les sous-arbres sont des vecteurs d'arbres de politiques $\eta(\delta, o) = (\eta(q_1, o^1), \cdots, \eta(q_n, o^n))$ de l'itération précédente. La Figure 1 illustre parfaitement une telle représentation. Le lecteur notera que cette représentation correspond exactement à celle des politiques jointes du MPOMDP sous-jacent au DEC-POMDP.

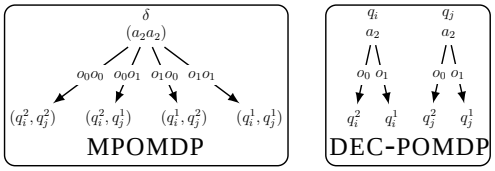

FIG. 1 – Politiques du MPOMDP et du DEC-POMDP.

3.2 Formulation du problème

Un effet désirable découle des observations précédentes : la recherche de la politique jointe δ^t optimale pour un état de croyance b et un ensemble de politiques jointes Q^{t-1} de l'itération précédente, équivaut au problème du déterminisme de l'action jointe $\alpha(\delta^t)$ et des sous-arbres $\eta(\delta^t, o)$ de sorte que l'arbre-joint δ^t résultant soit à la fois *décentralisable* et optimal. Nous dirons qu'un arbre-joint est décentralisable s'il existe un vecteur d'arbres de politique $(q_1, \cdots, q_n)$ qui vérifie les conditions (C1) et (C2) suivantes, dites *conditions de décentralisabilité de l'arbre joint* :
– (C1) $\alpha(\delta) = (\alpha(q_1), \cdots, \alpha(q_n))$,
– (C2) $\eta(\delta, o) = (\eta(q_1, o^1), \cdots, \eta(q_n, o^n))$.
Un arbre-joint est dit décentralisable s'il équivaut à un vecteur de politiques, autrement nous dirons qu'il est non-décentralisable et correspond simplement à une politique jointe du MPOMDP sous-jacent au DEC-POMDP à résoudre.

3.3 L'approche heuristique

A ce niveau, nous nous retrouvons avec deux problèmes : tout d'abord, comment les vecteurs d'arbres de politiques de l'itération précédente peuvent-ils nous aider à estimer le potentiel d'utilité d'un arbre-joint courant ; de plus comment ces estimations peuvent-elles orienter la recherche vers des arbres-joints les plus utiles.

Évaluation des estimations heuristiques. Afin d'estimer le potentiel d'utilité d'un arbre-joint δ, pour un état de croyance b et des sous-arbres, nous devons identifier la contribution de chacun des sous-arbres dans la valeur de l'arbre-joint courant.
– la contribution d'un sous-arbre $\eta(\delta, o)$ est :

$$g^{\delta}_{b,o} = P(o|b, \delta) \left[\sum_{s' \in S} P(s'|s, \delta, o) V(s', \eta(\delta, o))\right] \quad (3)$$

l'évaluation de toutes les contributions requiert un temps polynômial, $O(|S^2||A||\Omega| maxTrees^2)$, négligeable comparativement à la complexité de la sauvegarde exhaustive.
– et la valeur exacte de l'arbre-joint δ est :

$$f^{\star}(b, \delta) = \sum_{s \in S} b(s) R(s, \alpha(\delta)) + \sum_{o \in \Omega} g^{\delta}_{b,o} \quad (4)$$

Il assez facile, cependant, de démontrer que l'arbre-joint dont les sous-arbres offrent individuellement les meilleures contributions est très probablement non-décentralisable. En effet, il correspond à la politique *optimale*, selon les critères précités, du MPOMDP sous-jacent au DEC-POMDP à résoudre. À l'exception de cas particuliers, la valeur de la politique optimale du MPOMDP sous-jacent est une borne supérieure sur la valeur de la meilleure politique jointe du DEC-POMDP à résoudre. En outre, contrairement aux MPOMDPs dans le cadre des DEC-POMDPs les sous-arbres d'un arbre-joint doivent vérifier la condition (C2). Néanmoins, la bonne nouvelle est que nous pouvons utiliser ces estimations pour définir une heuristique sur la valeur réelle des arbres-joints *partiellement ou complètement définis*. On dira qu'un arbre-joint est partiellement défini si tout ou partie des ses sous-arbres reste à définir. L'estimation heuristique d'un arbre-joint est basée sur la décomposition de la fonction d'évaluation (Equation 1) en deux estimations : la première estimation, $g(b, \delta)$, correspond à la somme des contributions des sous-arbres définis et bien sûr sélectionnés conformément à (C2) ; la deuxième estimation, $h(b, \delta)$, est la somme des meilleures contributions des sous-arbres non définis. Nous introduisons Ω^1 comme l'ensemble des observations jointes qui mène à un sous-arbre défini et Ω^2 correspond au reste des observations jointes de sorte que $\Omega = \Omega^1 \cup \Omega^2$. Ceci nous permet de décomposer l'estimation heuristique de tout arbre-joint δ pour un état de croyance donné b en une partie provenant de la contribution exacte des observations jointes Ω^1 et une borne-supérieure

sur les contributions en provenance des autres observations jointes Ω^2 :

$$f(b,\delta) = \underbrace{b \cdot r_{\alpha(\delta)} + \sum_{o \in \Omega^1} g^{\delta}_{b,o}}_{g(b,\delta)} + \underbrace{\sum_{o \in \Omega^2} \bar{g}^{b}_{\alpha(\delta),o}}_{h(b,\delta)} \quad (5)$$

avec $\bar{g}^{b}_{\alpha(\delta),o} = \max_{\delta} g^{\delta}_{b,o}$. Bien entendu, l'estimation heuristique f est une fonction heuristique admissible. En effet, pour tout arbre-joint δ et état de croyance b, on a par construction : $f(b,\delta) \geq f^{\star}(b,\delta)$.

Description sommaire. Dans un souci de clarté nous proposons une première description naïve de notre méthode nommée PBIP (POINT-BASED INCREMENTAL PRUNING). Cette méthode vise à identifier l'arbre-joint *optimal* selon les critères précités. Pour ce faire, PBIP divise la sélection de l'arbre-joint optimal en la sélection de plusieurs sous-arbres, un pour chaque observation jointe. Malheureusement, comme nous l'avons déjà mentionné, l'arbre-joint résultant doit être optimal et décentralisable, en d'autre termes il doit vérifier les conditions (C1) et (C2). Afin de satisfaire ces exigences, PBIP combine un mécanisme de construction incrémentale (sous-arbre par sous-arbre) de l'arbre-joint à un mécanisme de recherche heuristique classique. La méthode résultante est alors capable de construire progressivement un arbre de recherche dans l'espace des arbres-joints. Le nœud racine de l'arbre de recherche est initialisé par un arbre-joint de racine une action jointe (non traitée) et dont aucun des sous-arbres n'est défini. L'expansion d'un nœud n (de l'arbre de recherche) marqué par l'arbre-joint δ correspond à la construction de tous les arbres-joints qui étendent δ. En d'autres termes, il s'agit de construire tous les arbres-joints où un sous-arbre non défini de δ est remplacé par un sous-arbre sélectionné conformément à (C2), pour chaque arbre-joint δ' ainsi créé, assigner à n un nœud fils marqué par δ'. Lorsqu'un arbre de recherche a été complètement exploré, PBIP garde en mémoire le meilleur arbre-joint et débute un nouvel arbre de recherche avec comme arbre-joint initial un arbre-joint partiel dont la racine est étiquettée par une action jointe non traitée, jusqu'à ce que toutes les actions jointes aient été traitées. La valeur exacte de l'arbre-joint qui est à la fois complètement défini (tous les sous-arbres sont définis) et l'actuel meilleur arbre-joint (disposant de la plus grande valeur), peut-être définie comme borne inférieure notée $\underline{f}(b)$.

3.4 L'algorithme PBIP

La méthode heuristique decrite ci-dessus, de type $A^{\star}$, souffre de trois inconvénients majeurs : elle requiert de nombreux calculs ; consomme beaucoup de mémoire ; et la stratégie d'élagage est inefficace. Afin de faire face à ces inconvénients, nous proposons les optimisations supplémentaires suivantes.

Réduire la quantité de calculs. Tout d'abord l'heuristique ci-dessus nécessite la vérification des conditions (C1) et (C2) pour chaque arbre-joint partiel δ. Toutefois, cette opération exige en pire cas une complexité en temps de $O(n \cdot |\Omega^1| \cdot |Q^{t-1}|)$, où $O(n \cdot |\Omega^1|)$ opérations sont requises pour un seul sous-arbre. Pour contourner ce point, nous nous appuyons sur le fait que seul un nombre restreint de sous-arbres doit être sélectionnés afin de définir complètement un arbre-joint, comme l'illustre l'exemple suivant.

Exemple 1 *Revenons sur l'exemple Figure 1, supposons connu les sous-arbres suivants :* $\eta(\delta,(o_0,o_0)) = (q_i^2,q_j^2)$ *et* $\eta(\delta,(o_1,o_1)) = (q_i^1,q_j^1)$. *On a alors :* $\eta(q_i,o_0) = q_i^2$ *et* $\eta(q_i,o_1) = q_i^1$ *pour l'agent* i*; et* $\eta(q_j,o_0) = q_j^2$ *et* $\eta(q_j,o_1) = q_j^1$ *pour l'agent* j*. Dès lors il est facile de construire l'arbre-joint* δ *correspondant, par exemple* $\eta(\delta,(o_0,o_1)) = (\eta(q_i,o_0),\eta(q_j,o_1)) = (q_i^2,q_j^1)$.

Cette remarque nous permet de réduire les calculs intensifs qui étaient précédemment nécessaires, car (C2) ne sera vérifiée que pour un nombre assez réduit de sous-arbres. Plus formellement, nous introduisons la notion d'*arbre-joint de base* afin de désigner un arbre-joint partiel qui possède le plus petit nombre de sous-arbres définis et suffisants pour construire l'arbre-joint complet. L'ensemble des observations jointes, qui étiquettent les arcs menant aux sous-arbres définis, est dit ensemble d'*observations jointes de base* et noté $\Omega_B = \{o^1,\cdots,o^{\kappa}\}$. Les nœuds associés aux sous-arbres définis sont appelés les nœuds de base. Dans la Figure 1, l'arbre-joint δ où les arcs $\{(o_0,o_1),(o_1,o_0)\}$ sont supprimés est un arbre-joint de base, ses observations jointes de base et nœuds de base sont les ensembles $\{(o_0,o_0),(o_1,o_1)\}$ et $\{(q_i^2,q_j^2),(q_i^1,q_j^1)\}$, respectivement. S'il est relativement simple de déterminer l'arbre-joint complet sachant l'arbre-joint

de base, on peut se demander comment déterminer un arbre-joint de base. Une des stratégies est de déterminer tout d'abord un ensemble d'observations jointes de base Ω_B, puis de sélectionner les sous-arbres pour chacun des nœuds de base. Nous construisons Ω_B par ajout progressif d'une observation jointe $o \in \Omega$ qui soit *indépendante* de celles déjà incluses dans Ω_B. On dira que les deux observations jointes (o_0, o_0) et (o_1, o_1) sont *indépendantes par composante* tandis que (o_0, o_0) et (o_0, o_1) sont simplement indépendantes (car une des composantes est identique). L'ajout d'observations jointes indépendantes se fait de sorte à privilégier les observations jointes indépendantes par composante. La procédure s'arrête lorsque l'ensemble Ω_B est un ensembled de base. La belle propriété des arbres-joints de base est que la vérification de (C2) ne se fait que pour les nœuds de base associés aux observations jointes simplement indépendantes. En effet, cette condition est automatiquement vérifiée dans le cas d'observations indépendantes par composantes. En outre, on peut montrer que le nombre κ de nœuds de base est assez réduit, où $\kappa = \max_{i \in I} |\Omega_i|$ (par construction de Ω_B). En particulier, si tous les agents ont le même nombre d'observations individuelles, il est inutile de vérifier (C2). En effet, dans ce cas, seules les observations jointes indépendantes par composantes sont inclues dans Ω_B.

Réduire la complexité en espace. Un inconvénient majeur des techniques de type $A^\star$ est qu'en pire cas elles doivent se souvenir d'un nombre exponentiel de nœuds. Malheureusement, cette stratégie requiert une quantité considérable de mémoire. Pour faire face à cela, nous restreignons le nombre de nœuds à développer à chaque niveau de l'arbre de recherche. Plus précisement, les arbres-joints partiels attachés aux feuilles de l'arbre de recherche correspondent uniquement à toutes les associations possibles d'un sous-arbre à un seul et même nœud de base non-défini, comme l'illustre la Figure 2. Cette idée est motivée par le constat selon lequel ces arbres-joints partiels ne diffèrent que par rapport au dernier sous-arbre qui leur est affecté. En conséquence, ils sont traitées dans l'ordre décroissant de la contribution du dernier sous-arbre affecté. Les contributions de chaque sous-arbre selon le nœud de base considéré sont précalculées et stockées dans une file de priorité. Ainsi, lors de l'expansion d'un nœud feuille, il nous suffit de construire le meilleur nœud fils, c'est-à-dire le nœud marqué par un arbre-joint dont le dernier sous-arbre assigné est le premier de la file de priorité. En effet, les autres nœuds ne seront developpés que s'il s'avère que la valeur réelle du nœud developpé est inférieure à l'estimation heuristique du nœud suivant dans la file. Finalement, l'algorithme résultant ne maintient en mémoire qu'un nœud par niveau de l'arbre de recherche, soit en pire cas $(\kappa + 1)$ nœuds.

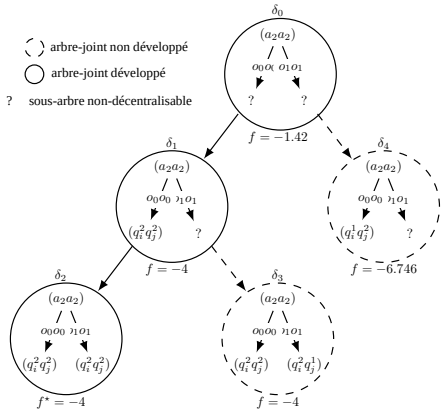

FIG. 2 – Une section d'un arbre de recherche de PBIP.

Algorithm 1 Élagage incrémental à base de points

procedure PBIP($(maxTrees, T, H)$)
  $Select^1 \leftarrow$ initialiser toutes les politiques de profondeur 1
  for all $t = 2, \cdots, T$ **do**
    $Q^{t-1} \leftarrow Select^t$ and $Select^t \leftarrow \emptyset$
    for all $k = 1, \cdots, maxTrees$ **do**
      choisir $h \in H$ et générer l'état de croyance b
      $\delta^t \leftarrow null$ et $\underline{f}(b) \leftarrow -\infty$
      for all $a \in A$ **do**
        RECHERCHE$(b, a, \delta^t, \Omega_B, Q^{t-1}, Select^t)$
      end for
      ajout du meilleur arbre-joint δ^t à $Select^t$
    end for
  end for
  sélectionner l'arbre-joint optimal $\delta^{\star T}$ de $Select^T$
  returner $\delta^{\star T}$
end procedure

Améliorer la stratégie d'élagage. L'utilisation des arbres-joints de base nous offre l'opportunité d'améliorer significativement la stratégie d'élagage en exploitant les propriétés de la fonction heuristique f. Soit $Q(\delta)$ l'ensemble des arbres-joints *successeurs* de l'arbre-joint δ, c'est-à-dire les arbres-joints construits après δ selon l'ordre de priorité défini précédemment. Par exemple, dans la Figure 2, les successeurs de l'arbre-joint δ_1 sont les arbres-joints $\{\delta_2, \delta_3, \delta_4\}$. L'admissibilité de la fonction heuristique f et l'ordre suivant lequel PBIP traite les arbres-joints, nous permet d'énoncer les résultats suivants :

Lemme 1 *Soit δ un arbre-joint attaché au nœud n. Il vient alors que : $\forall \delta' \in Q(\delta)$: $f(b, \delta) \geq f(b, \delta')$.*

Algorithm 2 PBIP sous-routines.

```
1: procedure RECHERCHE((b, a, δ^t, Ω_B, Q^{t-1}, Select^t))
2:     initialiser open ← PILE-VIDE, k ← 0
3:     définir α(δ) ← a
4:     calculer ∀(o, η(δ, o)) ∈ Ω × Q^{t-1} : g^δ_{b,o}, ḡ^b_{a,o}
5:     ÉTENDRE                                   ▷ initialiser l'arbre de recherche
6:     while open ≠ PILE-VIDE do
7:         (o^k, η(δ, o^k)) ← open.PEEK
8:         if f(b) < f(b, δ) then
9:             if o^k = o^κ then
10:                if f*(b, δ) > f(b) and δ ∉ Select^t then
11:                    δ^t ← δ
12:                    f(b) ← f*(b, δ^t)
13:                end if
14:                EXPLORER          ▷ selectionner un nouveau sous-arbre η(δ, o^k)
15:            else ÉTENDRE          ▷ assigner un sous-arbre η(δ, o^{k+1})
16:            end if
17:        else BACKTRACK            ▷ retourne à la position η(δ, o^{k-1})
18:        end if
19:    end while
20: end procedure
```

```
1: procedure ÉTENDRE
2:     k ← k + 1                                 ▷ passer à η(δ, o^{k+1})
3:     η(δ, o^k) ← EXPLORER
4:     open.PUSH(o^k, η(δ, o^k))
5: end procedure
6: procedure BACKTRACK
7:     if open.ISNOTEMPTY then
8:         (o^k, −) ← open.POP
9:         η(δ, o^k) ← −1                        ▷ ré-initialiser la position du pointeur
10:        if o^k ≠ o^1 then
11:            k ← k − 1                         ▷ retourner à η(δ, o^{k-1})
12:            EXPLORER
13:        end if
14:    end if
15: end procedure
16: procedure EXPLORER
17:    INCRÉMENTER (η(δ, o^k))
18:    if η(δ, o^k) > |Q^{t-1}| then
19:        BACKTRACK                             ▷ backtrack à la position η(δ, o^{k-1})
20:    end if
21:    retourner η(δ, o^k)
22: end procedure
```

Preuve D'après l'ordre de traitement des nœuds cette assertion est vraie pour tous les arbres-joints successeurs et attachés aux nœuds frères de n. En ce qui concerne les autres arbres-joints, il convient de procéder par induction. Premièrement le cas de base consiste à montrer que cette assertion est vraie pour le cas d'un nœud fils de n : $f(b,\delta) \geq f(b,\delta_1)$, où δ_1 est l'arbre-joint attaché au nœud fils. Il suffit alors de différentier δ_1 de δ. La principale différence réside sur le fait que δ_1 dispose d'un sous-arbre défini de plus que δ, disons le sous-arbre associé à l'observation jointe de base o^1 :

$$\begin{aligned} f(b,\delta_1) &= f(b,\delta) - \bar{g}^b_{\alpha(\delta),o^1} + g^{\delta_1}_{b,o^1} \\ &= f(b,\delta) - \bar{g}^b_{\alpha(\delta_1),o^1} + g^{\delta_1}_{b,o^1} \quad (\alpha(\delta) = \alpha(\delta_1)) \\ &\leq f(b,\delta) \end{aligned}$$

La dernière inégalité tient car $\bar{g}^b_{\alpha(\delta_1),o^1} \geq g^{\delta_1}_{b,o^1}$ par définition d'une borne supérieure. En répétant cet argument pour toute paire consécutive $(\delta_k,\delta_{k+1}) \in Q(\delta) \times Q(\delta)$, on a : $f(b,\delta_k) \geq f(b,\delta_{k+1}),\ \forall k = 1,\cdots,|Q(\delta)|$ ∎

Intuitivement, il est évident que cette propriété est plus restrictive que la simple *monotonie* car $Q(\delta)$ inclut non seulement les nœuds fils mais aussi les nœuds frères de n. L'usage de cette propriété et l'ordre suivant lequel PBIP traite les nœuds, nous permet d'énoncer le théorème suivant. Ce théorème décrit une propriété utile pour éviter le développement des arbres-joints avec une valeur heuristique faible.

Théorème 1 *Soit δ un arbre-joint avec $f(b,\delta)$ sa valeur heuristique, et $f^\star(b,\delta')$ la valeur exacte du meilleur arbre-joint courant δ'. Si $f(b,\delta) \leq f^\star(b,\delta')$, alors on a : $f^\star(b,\delta') \geq f^\star(b,\delta_k),\ \forall\delta_k \in Q(\delta)$.*

Preuve En accord avec le Lemme 1, nous avons :

$$f(b,\delta) \geq f(b,\delta_k),\ \forall\delta_k \in Q(\delta)$$

On obtient alors :

$$\begin{aligned} f^\star(b,\delta') &\geq f(b,\delta) && \text{(par hypothèse)} \\ &\geq f(b,\delta_k),\ \forall\delta_k \in Q(\delta) && \text{(Lemme 1)} \\ &\geq f^\star(b,\delta_k),\ \forall\delta_k \in Q(\delta) && \text{(déf. de la borne sup.)} \end{aligned}$$

∎

Ce théorème indique que si un arbre-joint δ a une valeur heuristique inférieure ou égale à la borne inférieure courante, PBIP ne developpe pas le sous-espace $Q(\delta)$. Cette stratégie améliore considérablement le mécanisme d'élagage de la version naïve précédente. En effet, elle ne requiert pas la génération et l'évaluation de tous les nœuds successeurs. D'un point de vue pratique, si la valeur heuristique de δ est inférieure ou égale à la borne inférieure actuelle (*condition de backtrack*), alors il existe deux possibilités : (a) si le nœud parent de δ est le nœud racine, alors on met fin à la recherche ; (b) sinon, la recherche *retourne* vers le nœud parent de δ et teste la condition de backtrack.

Exemple 2 *La Figure 2 décrit différentes étapes de l'algorithme* PBIP. *Le premier nœud en partant du nœud racine δ_0 est δ_1, car il dispose de la plus grande valeur heuristique -4. Ce nœud à son tour génère un seul nœud fils δ_2. Comme l'arbre-joint résultant ne dispose plus de nœuds de base non définis,* PBIP *calcule sa valeur exact -4 et l'utilise comme borne inférieure.* PBIP *retourne ensuite vers le nœud δ_1. Fort du fait que la valeur heuristique -4 de δ_1 est égale à la borne inférieure courante,* PBIP *ne construit pas ses nœuds successeurs δ_3 et δ_4. Enfin, la recherche se termine car le nœud parent δ_0 de δ_1 est la racine de l'arbre de recherche. Ainsi, l'arbre-joint δ_2 est le meilleur pour cet arbre de recherche.*

3.5 Implémentation spécifique

Dans la même veine que MBDP, PBIP combine heuristiques de recherche d'états de croyance par chaînage-avant et sélection de politiques par chaînage-arrière comme l'indique l'Algorithme 1. Cependant, ce qui distingue PBIP de MBDP est sa capacité à éviter la génération exhaustive de toutes les politiques jointes à chaque itération. En effet, une seule itération de PBIP peut-être résumée comme suit : d'abord l'algorithme défini l'ensemble Q^{t-1}. des politiques jointes de l'itération précédente ; puis, il choisit un portefeuille d'heuristiques H de recherche d'états de croyance par chaînage-avant, afin d'identifier $maxTrees$ états de croyance ; ensuite, il utilise une sous-routine nommée RECHERCHE pour le déterminisme de l'ensemble des politiques jointes optimales $Select^t$, une pour chaque état de croyance ; finalement, à l'itération T, la meilleure politique jointe, en fonction de l'état de croyance initial b_0, est retournée.

Dans ce qui suit, nous attirons l'attention sur le traitement d'un seul arbre de recherche , avec pour données en entrée : un état de croyance b ; une action jointe a ; l'ensemble des observations jointes de base Ω_B ; les sous-arbres Q^{t-1} possibles provenant de l'itération précédente ; et finalement, the meilleures politiques jointes courantes $\delta^t \in Select^t$ (voir Algorithm 2). Les contributions exactes ou optimistes des sous-arbres dans Q^{t-1} sont pré-calculées et stockées dans une file de priorité. Nous utiliserons $\eta(\delta, o)$ à la fois comme sous-arbre et comme sa position dans la file de façon inter-changeable.

4 Propriétés Théoriques

Dans cette section, nous allons présenter quelques propriétés théoriques additionnelles qui garantissent les performances de PBIP, y compris la complétude, l'*optimalité*, et la complexité.

En exploitant la similitude de PBIP à $A^\star$ il est facile de montrer que : l'algorithme PBIP est complet ; et il est optimal en fonction de l'état de croyance et des sous-arbres. Tous les résultats précédemment énoncés visent à garantir que PBIP offre des performances au moins aussi bonnes que celle de MBDP et ses extensions. Mieux, ils soulignent le fait que PBIP peut significativement faciliter le passage à l'échelle comparativement à MBDP et ses extensions. Néanmoins, la complexité en pire cas de PBIP reste exponentielle suivant le nombre d'agents et κ. Dans les meilleurs cas, cependant, la complexité en temps est en $O(\kappa)$, cela correspond à la longueur du chemin nécessaire pour la construction d'un arbre-joint complet. Néanmoins la complexité en temps globale peut-être fortement influencée par le pré-calcul des contributions des sous-arbres, $O(|S^2||A||\Omega|maxTrees^2)$. Bien que PBIP s'inspire de l'algorithme $A^\star$, il requiert bien moins de mémoire. En effet, la sous-routine RECHERCHE nécessite une mémoire linéaire, c'est-à-dire la longueur du plus long chemin de la racine à un arbre-joint complet soit $O(\kappa)$. Cela est possible car nous calculons toutes les contributions de tous les sous-arbres pour chaque paire action-observation jointe et cela avant que la recherche ne commence. En conséquence, la complexité globale en mémoire de PBIP est $O(|A||\Omega|maxTrees)$.

5 Expérimentations

Nous avons testé notre algorithme sur plusieurs bancs d'essais de DEC-POMDPs à 2 agents : y compris le problème du *broacast* sur un canal de communication à accès multiple (MABC) ; le problème du tigre multi-agent (TIGER) et le problème BOX-PUSHING considéré comme le plus large de la littérature. Nous avons comparé les performances de notre approche à celles des meilleurs algorithmes approximatifs connus : MBDP, IMBDP et MBDP-OC. Tous ont été exécutés sur la même machine en utilisant les mêmes paramêtres : portefeuille d'heuristiques ; $maxTrees$ et $maxObs$ en ce qui concerne IMBDP et MBDP-OC. Une sélection de ces résultats est rassemblée dans le Tableau 1 et dans la Figure 3. Ces résultats confirment la réduction drastique de la complexité de la sauvegarde exhaustive, notamment pour des paramêtres $maxTrees$ élevés (voir Figure 3) ou des problèmes de plus grande taille (voir Tableau 1). PBIP surpasse tous les autres algorithmes selon tous les critères et les domaines testés. Il construit ses politiques jointes jusqu'à 800 fois plus vite que IMBDP et MBDP-OC tout en offrant des politiques jointes de meilleures qualités. Par exemple, pour le problème BOX-PUSHING à horizon 100, PBIP construit une politique jointe de valeur 786.4 tandis que celles de IMBDP et MBDP-OC sont de 72.8 et 503.8 respectivement.

5.1 Travaux connexes

De nombreuses techniques, d'utilisation des techniques par chaînage-avant afin de résoudre

TIGER problem $maxTrees = 20$									
Algorithm	MBDP		IMBDP		MBDP-OC		PBIP		
T	AEV	CPU (sec.)	AEV	CPU (sec.)	AEV	CPU (sec.)	AEV	CPU (sec.)	%
10	12.5 ± 2.9	67.1 ± 0.13	N.A.		N.A.		13.6 ± 1.11	5.55 ± 0.88	7.83
20	25.8 ± 2.1	159 ± 0.16	N.A.		N.A.		26.8 ± 1.50	15.1 ± 2.68	12.4
50	73.9 ± 0.7	438 ± 0.86	N.A.		N.A.		74.2 ± 2.76	45.3 ± 6.49	15.3
100	149	901	N.A.		N.A.		147 ± 5.84	94.2 ± 9.53	12.9
COOPERATIVE BOX-PUSHING problem $maxTrees = 3$ $maxObs = 3$									
10	N.A.		99.59	7994.6	89.94	7994.4	128.4 ± 6.46	11.8 ± 3.61	10.9
20	N.A.		102.6	17031	135.0	17194	198.0 ± 10.8	25.0 ± 4.21	10.3
50	N.A.		82.99	44708	272.70	44210	422.2 ± 20.8	61.6 ± 10.6	9.07
100	N.A.		73.88	89471	440.08	90914	786.4 ± 31.9	113 ± 20.8	9.62
A.E.V = valeur escomptée moyenne N.A. = pas applicable % = pourcentage d'arbres-joints developpés									

TAB. 1 – Performances de PBIP, MBDP, IMBDP et MBDP-OC.

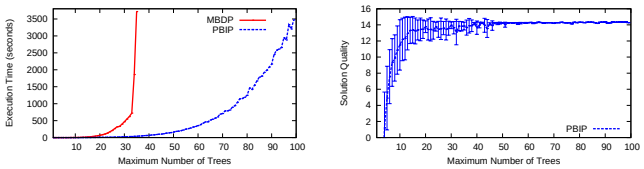

FIG. 3 – Performances pour TIGER à $T = 10$.

les DEC-POMDPs, ont été développées [13, 12, 10]. En principe, ces méthodes de recherche heuristique peuvent traiter de larges domaines de façon efficace, notamment au moyen de la combinaison des bornes supérieures et/ou inférieures à la connaissance des informations initiales du système, par exemple l'état de croyance initial. Szer *et al.* [12] ont fourni le premier algorithme de recherche heuristique pour la résolution des DEC-POMDPs à horizon fini, à savoir MAA*. Cet algorithme est basé sur la combinaison des heurisiques classiques, $A^\star$ par exemple, à la théorie du contrôle décentralisé. MAA* cherche à déterminer la solution optimale pour un horizon prédéfini. Bien que MAA* soit un grand pas en avant par rapport aux algorithmes précédents, il souffre d'un inconvénient majeur : sa capacité à restreindre l'espace de recherche dépend de la précision des bornes (inférieure et supérieure). Malheureusement, la recherche de bornes plus serrées requiert bien souvent un coût additif non-négligeable. Bien que cet algorithme ne parvient pas à résoudre des DEC-POMDPs à horizon supérieur à 4, ses résultats démontrent le potentiel des techniques de recherche heuristique. Varakantham *et al.* [13] suggèrent une version similaire à MAA*, nommée SPIDER, qui exploite l'interaction entre les coéquipiers afin de contourner les limitations de MAA*. Plus précisément, SPIDER améliore la politique jointe en modifiant la politique d'un agent à la fois et suivant l'ordre des interactions. En procédant ainsi, SPIDER est capable de résoudre des problèmes décentralisés disposant d'un nombre d'agents *relativement* grand. Bien qu'intéressante, l'utilisation de SPIDER dépend de l'hypothèse selon laquelle l'on dispose d'une structure d'interations entre coéquipiers et que de plus ces interactions sont faibles. En ce qui concerne le cas général des DEC-POMDPs à horizon fini, où les interactions sont soit inconnues soit fortement couplées, SPIDER n'offre aucun bénéfice.

Au contraire, PBIP offre un certain nombre d'avancées majeures en comparaison à MAA* et SPIDER. Tout d'abord, PBIP souligne le lien qui existe entre la représentation des politiques en DEC-POMDPs et celle en MPOMDPs. Cette observation est cruciale car elle permet d'adapter les techniques développées dans la communauté des MPOMDPs à celle des DEC-POMDPs. Ainsi, la résolution d'un DEC-POMDP peut désormais être vue comme la résolution d'un MPOMDP avec des contraintes additionnelles sur les politiques. En poussant plus loin cette idée, nous proposons une adaptation de la meilleure technique de construction incrémentale de politiques en MPOMDPs au cadre décentralisé. Ceci permet l'introduction de la première technique de construction incrémentale (sous-arbre par sous-arbre) de politiques jointes. La différence essentielle entre PBIP et MAA* réside dans le fait que MAA* associe à chaque nœud de l'arbre de recherche une politique jointe complète tandis que PBIP y associe un sous-arbre de la politique jointe recherchée. Cette différence offre à PBIP une très grande flexibilité dans l'exploration de l'espace de recherche comme le démontrent les résultats expérimentaux.

Le deuxième ensemble de techniques consiste en des méthodes de chaînage-arrière ou mixtes [11, 9, 8, 4]. Conformément à Szer *et al.*, la principale limitation des approches de chaînage-arrière (y compris MBDP, IMBDP [8] et MBDP-

OC [4]) est le nombre de calculs dû à l'énumération exhaustive. En effet, chaque itération de ces techniques requiert un appel à la sauvegarde exhaustive, avant que ne commence l'étape d'élagage. Bien que ce phénomène soit similaire en MPOMDPs, il apparaît bien dévastateur en DEC-POMDPs. En effet, les fonctions de valeur en DEC-POMDPs sont définies sur un espace de politiques bien plus large. Les récents algorithmes de cette classe tels que MBDP [9], IMBDP [8] et MBDP-OC [4] tentent de mitiger cet handicap au moyen de : (1) la sélection d'un petit nombre ($maxTrees$) de politiques par agent et par horizon ; (2) la réduction de la croissance exponentielle dûe en partie aux observations, notamment en les échantillonnant. Cependant, le choix du paramètre $maxTrees$ n'est pas trivial et son ajustement de façon dynamique est coûteux. De plus, l'efficacité de la seconde proposition dépendra du domaine, comme le prouve les expérimentations cette proposition réduit considérablement la qualité de la solution retournée. Finalement, en pire cas la complexité de ces algorithmes croît de façon exponentielle selon le nombre d'observations selectionnées ($maxObs$) et le nombre d'agents $|I|$, soit $|S^2||A||\Omega|maxTrees^{maxObs|I|+1}$. Au contraire, bien qu'en théorie notre approche souffre des mêmes contraintes de complexité en pire cas, en pratique elle n'explore qu'un petit sous-espace de l'espace entier des politiques jointes et élague automatiquement les sous-espaces inutiles.

Enfin, Aras *et. al* [1] ont récemment proposé un programme linéaire mixte (MILP-DEC) pour la résolution exacte des DEC-POMDPs à horizon fini. Cette approche s'est construite sur l'observation selon laquelle un DEC-POMDP équivaut à un POMDP sous contraintes structurelles. Bien que nous ayons abouti, de façon indépendante, à cette même observation, les solutions proposées et les performances de nos méthodes (MILP-DEC et PBIP) sont radicalement différentes. Cela s'explique par le fait que PBIP utilise à son avantage les dites contraintes, notamment en observant lesquelles peuvent être systématiquement satisfaites sous certaines conditions.

6 Conclusion

Nous avons présenté la première technique de construction incrémentale de politiques jointes d'un DEC-POMDP sachant un état de croyance et des sous-arbres (PBIP). Cette approche permet de remplacer avantageusement un opérateur de programmation dynamique jusqu'ici indispensable mais extrêmement coûteux, à savoir la sauvegarde exhaustive. Par ce biais, PBIP facilite grandement le passage à l'échelle de nombreux algorithmes de programmation dynamique, y compris ceux de la famille des algorithmes approximatifs de type MBDP. Nous travaillons actuellement sur une extension de PBIP qui nous permettra de résoudre des problèmes généraux (sans hypothèses restrictives) de contrôle décentralisé de plusieurs ($>> 2$) agents coopératifs.

Remerciements

Nous aimerions remercier Sven Seuken et Alan Carlin pour avoir rendu disponible leur code source des algorithmes MBDP, IMBDP et MBDP-OC. Nous remercions également Hamid R. Chinaei et Camille Besse pour leurs commentaires sur ce travail.

Références

[1] Raghav Aras, Alain Dutech, and François Charpillet. Mixed integer linear programming for exact finite-horizon planning in decentralized pomdps. *CoRR*, 2007.

[2] Richard Ernest Bellman. *Dynamic Programming*. Dover Publications, Incorporated, 1957.

[3] Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of markov decision processes. *Math. Oper. Res.*, 27(4), 2002.

[4] Alan Carlin and Shlomo Zilberstein. Value-based observation compression for DEC-POMDPs. In *AAMAS*, 2008.

[5] Eric A. Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *AAAI*, pages 709–715, 2004.

[6] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration : An anytime algorithm for POMDPs. In *IJCAI*, 2003.

[7] Zinovi Rabinovich, Claudia V. Goldman, and Jeffrey S. Rosenschein. The complexity of multiagent systems : the price of silence. In *AAMAS*, pages 1102–1103, 2003.

[8] Sven Seuken and Shlomo Zilberstein. Improved Memory-Bounded Dynamic Programming for DEC-POMDPs. In *UAI*, 2007.

[9] Sven Seuken and Shlomo Zilberstein. Memory-bounded dynamic programming for DEC-POMDPs. In *IJCAI*, pages 2009–2015, 2007.

[10] Daniel Szer and François Charpillet. An optimal best-first search algorithm for solving infinite horizon DEC-POMDPs. In *ECML*, pages 389–399, 2005.

[11] Daniel Szer and François Charpillet. Point-based dynamic programming for DEC-POMDPs. In *AAAI*, pages 16–20, July 2006.

[12] Daniel Szer, Francois Charpillet, and Shlomo Zilberstein. Maa* : A heuristic search algorithm for solving decentralized pomdps. In *UAI*, pages 568–576, 2005.

[13] Pradeep Varakantham, Janusz Marecki, Milind Tambe, and Makoto Yokoo. Letting loose a spider on a network of pomdps : Generating quality guaranteed policies. In *AAMAS*, May 2007.