Prise de décision en temps-réel pour des POMDPs de grande taille

Sébastien Paquet — Ludovic Tobin — Brahim Chaib-draa

Laboratoire DAMAS
Département d'informatique et de génie logiciel
Université Laval, Canada
{spaquet ;tobin ;chaib}@damas.ift.ulaval.ca

RÉSUMÉ. *Cet article présente une méthode d'approximation pour les processus décisionnels de Markov partiellement observables (POMDP) qui est basée sur une recherche en profondeur pour la planification dans un environnement temps-réel dynamique. L'idée de base de notre approche, appelée RTBSS (Real-Time Belief Space Search), est d'éviter de calculer des politiques complètes pour des POMDPs. Cette approche est spécialement utile pour des environnements temps-réel où l'espace d'états est trop grand pour que l'on puisse considérer les algorithmes de résolution hors-lignes des POMDPs. À cet effet, nous proposons une approche en-ligne pour calculer à chaque cycle, l'action qui maximise l'utilité espérée de l'agent. Nous commençons par présenter tout le formalisme à la base de notre méthode. Par la suite, nous présentons les résultats expérimentaux obtenus sur trois environnements : les environnements Tag et RockSample ainsi que la simulation de la RoboCupRescue. Les résultats obtenus montrent la force de notre approche, particulièrement en ce qui concerne la rapidité d'exécution et l'adaptabilité à de nouveaux environnements. Mentionnons par ailleurs que cette approche a été implémentée avec succès pour la compétition mondiale de la RoboCupRescue en 2004 à Lisbonne au Portugal où nous nous sommes classés en deuxième position.*

ABSTRACT. *This paper presents a POMDP approximation method, called RTBSS (Real-Time Belief Space Search), which is based on a look-ahead search in order to plan in a real-time dynamic environment. The basis of our approach is to avoid computing full policies in POMDP problems. Our approach is especially motivated by real-time environments where the state space is too large to consider traditional offline algorithms. We then proceed with an online approach to find at each step, the action that maximize the agent expected utility. To this end, we present the formalism behind our approach. Then, we present how the approach was applied on three different environments: Tag, RockSample and the RoboCupRescue simulation. Let us mention finally, that the approach we present was successfully implemented for the RoboCupRescue 2004 international competition in Lisbon, Portugal where we finished in second position.*

MOTS-CLÉS : *POMDP, temps-réel, prise de décision*

KEYWORDS: *POMDP, real-time, decision making*

1. Introduction

Les processus décisionnels de Markov partiellement observables (POMDPs) procurent un modèle général pour des problèmes de prise de décisions séquentielles dans des environnements partiellement observables. Plusieurs problèmes peuvent être modélisés à l'aide des POMDPs, mais très peu peuvent être résolus dû à la complexité du problème (les POMDPs sont PSPACE-complete [PAP 87]). Bien entendu, une telle complexité ne rend les POMDPs applicables que pour de très petits environnements, ne contenant que quelques centaines d'états tout au plus. Toutefois, la majorité des problèmes d'intérêt ont de très grands espaces d'états, ce qui a pour effet de motiver la recherche d'algorithmes d'approximation [HAU 00]. Ceci est particulièrement le cas dans les systèmes multiagents où il y a souvent un énorme espace d'états avec plusieurs agents interagissant les uns avec les autres.

Plusieurs algorithmes d'approximation pour les POMDPs ont été développés récemment [BRA 04, PIN 03, POU 05, SMI 04, SPA 04]. Ces algorithmes ont tous en commun le fait qu'ils essaient de solutionner le problème hors-ligne. Ces algorithmes peuvent obtenir de très bonnes performances, mais ils ne demeurent applicables que pour des environnements relativement petits.

L'un des problèmes lors du calcul d'une politique hors-ligne est que l'on doit associer une action à chacun des états de croyance (*belief state*) possibles, ce qui est rarement nécessaire. En d'autres mots, une politique hors-ligne spécifie l'action à exécuter pour chacune des situations que l'agent peut rencontrer dans l'environnement. Ceci nous apparaît contre intuitif car, dans la plupart des environnements complexes, plusieurs états ne seront jamais visités ou sont tout simplement inaccessibles. Idéalement, une politique devrait uniquement associer une action aux états qui peuvent éventuellement être atteints. Toutefois, ceci est particulièrement difficile si l'agent évolue dans un environnement hautement dynamique où l'agent ne peut pas savoir à l'avance comment l'environnement évoluera.

Une solution à ce problème est d'adopter une approche en-ligne pour trouver la meilleure séquence d'actions en ne considérant que les états de croyance qui ont une chance d'être atteints à partir de l'état de croyance actuel [MCA 99]. Notre algorithme, appelé RTBSS (Real-Time Belief Space Search), que nous présentons dans cet article est basé sur une recherche locale dans l'espace des états de croyance accessibles à partir de l'état de croyance courant.

Étant donné que notre algorithme doit travailler en-ligne sous certaines contraintes de temps-réel, l'exploration des états de croyance doit se faire le plus rapidement possible. Pour cela, nous tirons profit d'une représentation factorisée du problème et d'une stratégie de « branch and bound ». En élaguant certaines branches de l'arbre de recherche, l'algorithme peut rechercher plus en profondeur tout en respectant les contraintes de temps-réel.

Notre objectif principal était de développer un algorithme ne demandant aucun temps de calcul hors-ligne et pouvant tout de même être exécuté très rapidement. Ceci

dans le but de pouvoir utiliser l'algorithme dans des environnements complexes et dynamiques dans lesquels les approches hors-lignes ne peuvent tout simplement pas tout prévoir. De plus, en évitant les calculs hors-lignes, notre approche demeure applicable immédiatement, même si les conditions initiales de l'environnement changent.

D'autres approches utilisent une recherche en-ligne pour les POMDPs, mais elles ne sont pas immédiatement applicables sans calculs hors-lignes. Par exemple, l'algorithme BI-POMDP a besoin que le MDP sous-jacent soit résolu hors-ligne pour choisir l'ordre d'expansion des noeuds lors de la recherche [WAS 97]. De manière similaire, l'algorithme BEL-RTDP [GEF 98] demande des exécutions successives dans l'environnement en plus de la solution du MDP sous-jacent, par conséquent, il a besoin d'entraînement hors-ligne avant de devenir efficace.

Dans ce qui suit, nous présentons formellement notre approche ainsi que notre algorithme RTBSS (Real-Time Belief Space Search). De plus, nous commentons les résultats expérimentaux que nous avons obtenus sur trois environnements : les environnements *Tag* et *RockSample* ainsi que la simulation de la RoboCupRescue.

2. Motivations

Dans cette section, nous présentons les principales contraintes qui nous ont motivés lors du développement de notre algorithme :

– l'algorithme doit être efficace dans de grands espaces d'états ;

– l'environnement est partiellement observable ;

– un agent utilisant cet algorithme doit être efficace dans des instances de l'environnement qu'il n'a jamais vues.

– le temps de réponse de l'agent doit respecter certaines contraintes temps-réel.

Un exemple de problèmes avec ces contraintes est un robot autonome qui évolue dans des environnements inconnus et où il doit décider de l'action à accomplir dans un temps limité, même si c'est la première fois qu'il voit l'environnement. Un autre exemple est la simulation de la RoboCupRescue [KIT 00] dans laquelle les agents doivent être immédiatement efficaces dans des villes inconnues.

Dans de tels environnements, un algorithme pour les POMDPs doit garantir un temps de réponse très rapide (de l'ordre des 500ms par exemple). Par ailleurs, un agent devrait pouvoir être immédiatement efficace dans n'importe quelles configurations de l'environnement. Cette dernière contrainte élimine toutes les approches hors-lignes, parce qu'un agent n'a pas le temps d'apprendre une politique complète avant son exécution réelle dans l'environnement. Il n'est pas réaliste de laisser un algorithme hors-ligne prendre 2 jours, par exemple, pour calculer une politique complète chaque fois qu'il y a une légère modification dans l'environnement, parce que pendant ce temps, la situation peut se dégrader. Ces contraintes nous ont motivés à développer notre algorithme en-ligne pour la résolution de POMDPs qui permet une réponse rapide dans de grands espaces d'états.

3. POMDP

Dans cette section, nous présentons tout d'abord les POMDPs de base, par la suite, nous introduisons les POMDPs factorisés et nous terminons en présentant notre façon d'utiliser la factorisation de manière à réduire la complexité de calcul d'une politique. Il est à noter qu'il existe plusieurs articles décrivant les POMDPs de manière plus détaillée [KAE 96, ABE 03], mais dans cette section nous nous attardons uniquement à présenter les parties importantes à la compréhension de notre approche. Formellement, un POMDP est décrit comme un uplet $\langle S, A, T, R, \Omega, O \rangle$ où

– S : est l'ensemble de tous les états possibles ;

– A : est l'ensemble des actions que l'agent peut exécuter ;

– $T : S \times A \times S \rightarrow [0,1]$ est la fonction de transition. $P(s' \mid s, a)$ représente la probabilité d'arriver dans l'état s' si l'agent exécute l'action a dans l'état s ;

– $R : S \rightarrow \mathbb{R}$ est la fonction de récompense. $R(s)$ est donc la récompense associée à l'état s.

– Ω : est l'ensemble des observations que l'agent peut faire ;

– $O : S \times A \times \Omega \rightarrow [0,1]$ est la fonction d'observation. $P(o \mid s', a)$ représente la probabilité d'observer o si l'action a est exécutée et que l'on se retrouve dans l'état s'.

La résolution d'un POMDP peut être divisée en deux parties : l'agent doit pouvoir estimer l'état dans lequel il se trouve et il doit par la suite choisir une action. Pour gérer l'incertitude par rapport à l'état dans lequel il se trouve, l'agent peut garder un historique de toutes ses actions et ses observations passées. Il est aussi possible d'apprendre une extension sélective du passé [DUT 03], mais pour cet article, nous allons nous baser sur un fait bien connu que l'historique n'a pas à être représenté explicitement, mais qu'il peut être résumé à l'aide d'une distribution de probabilité, appelée état de croyance (*belief state*) [AST 65].

Un état de croyance b est défini comme une distribution de probabilité sur tous les états possibles de l'environnement. Dans ce cas, $b(s)$ donne la probabilité d'être dans un certain état s. Un agent peut mettre à jour son état de croyance lorsqu'il perçoit de nouvelles informations en utilisant l'équation 4. Cette technique est pratique sur de petits problèmes, mais lorsque l'espace d'états devient très grand, il devient intéressant d'utiliser une représentation plus compacte de l'espace d'état. L'ensemble de tous les états de croyance possibles B est infini et non dénombrable, car il contient toutes les distributions de probabilités possibles sur les états de l'environnement.

L'agent doit aussi pouvoir choisir une séquence d'actions à exécuter de manière à maximiser son utilité. À cet effet, on définit une politique $\pi : B \rightarrow A$ qui associe la meilleure action à effectuer selon l'état de croyance de l'agent. Généralement, la politique peut être calculée hors-ligne en utilisant des algorithmes bien connus comme l'algorithme d'énumération [SON 71] et l'algorithme Witness [LIT 96], ou des algorithmes d'approximation développés récemment tels que PBVI [PIN 03] et HSVI [SMI 04]. Lorsque l'agent doit choisir une action, il doit considérer son état

de croyance, car il ne sait pas exactement dans quel état il se trouve. La meilleure séquence d'actions est tout simplement celle qui donne l'utilité espérée maximale.

Plus formellement, la fonction de valeur standard pour un état de croyance dans un POMDP est :

$$V_t(b) = R(b) + \max_a \left[\gamma \sum_{o \in \Omega} P(o \mid b, a) V_{t-1}(\tau(b, a, o)) \right] \tag{1}$$

$$R(b) = \sum_{s \in S} b(s) R(s) \tag{2}$$

$R(b)$ représente la récompense espérée pour l'état de croyance courant. La deuxième partie de l'équation 1 représente la somme escomptée des récompenses espérées futures. $P(o \mid b, a)$ est la probabilité d'observer o si l'action a est exécutée dans l'état de croyance b. Cette probabilité peut être calculée en utilisant l'équation suivante (pour une preuve détaillée, voir [LIT 94]) :

$$P(o \mid b, a) = \sum_{s' \in S} P(o \mid s', a) \sum_{s \in S} P(s' \mid s, a) b(s) \tag{3}$$

De plus, $\tau(b, a, o)$ est la fonction de mise à jour des croyances de l'agent. Elle prend en entrée l'état de croyance courant b, l'action exécutée a et l'observation perçue o et elle retourne le nouvel état de croyance de l'agent. Pour ce faire, cette fonction itère sur tous les états et utilise la fonction suivante pour mettre à jour son état de croyance. Si $b' = \tau(b, a, o)$, alors :

$$b'(s') = \eta P(o \mid s', a) \sum_{s \in S} P(s' \mid s, a) b(s) \tag{4}$$

où η est une constante de normalisation.

Finalement, la politique π peut être calculée comme suit :

$$\pi_t(b) = \operatorname*{argmax}_a \left[R(b) + \gamma \sum_{o \in \Omega} P(o \mid b, a) V_{t-1}(\tau(b, a, o)) \right] \tag{5}$$

3.1. *POMDP factorisé*

Le modèle traditionnel de POMDP n'est pas vraiment adapté pour de grands environnements, parce qu'il utilise des fonctions appliquées sur tous les états. Heureusement, la plupart des environnements, malgré leur énorme espace d'états, peuvent être décrits de manière concise par un ensemble de variables aléatoires.

Posons $X = \{X_1, X_2, \ldots, X_M\}$ comme étant l'ensemble des M variables aléatoires nécessaires à la description d'un état. Puis $D_i = \operatorname{dom} X_i$ comme étant l'ensemble des

valeurs possibles pour la variable X_i. Dans notre cas, nous supposons que toutes les variables ont un domaine de valeurs discret et fini. Un état s peut maintenant être défini par l'assignation d'une valeur à chacune des variables aléatoires : $s = \{X_1 = x_1, X_2 = x_2, \ldots, X_M = x_M\}$ où $x_i \in D_i$. Nous utilisons aussi une forme de représentation plus compacte $s = \{x_i\}_{i=1}^{M}$.

Précédemment, nous avons défini un état de croyance b comme étant une distribution de probabilité sur tous les états possibles. Par conséquent, avec la représentation factorisée, un état de croyance b est défini comme une distribution de probabilité sur toutes les variables aléatoires : $b = \mathbf{P}(X_1, X_2, \ldots, X_M)$. Dans notre approche, nous considérons que toutes les variables sont indépendantes ; c'est pourquoi nous pouvons réécrire l'équation précédente comme : $b = \mathbf{P}(X_1)\mathbf{P}(X_2)\cdots\mathbf{P}(X_M)$. Bien sûr, cela peut sembler être une hypothèse très contraignante. En fait, il n'en est rien et nous discutons à la section 3.3 du cas où certaines variables seraient dépendantes.

Pour l'instant, nous commençons par le cas plus simple où il n'y a pas de dépendance. Ceci permet à l'agent de maintenir ses croyances sur chacune des variables individuelles, au lieu de maintenir ses croyances au niveau des états complets (toutes les combinaisons de valeurs pour toutes les variables). De cette manière, la complexité de la maintenance des croyances de l'agent est réduite de beaucoup.

Il convient de préciser que les équations de base 1, 2, 3 et 4, présentées au début de cette section, deviennent rapidement impraticables lorsque l'espace d'états grandit, car elles sont définies sur tous les états. Il serait intéressant de pouvoir éviter cela de manière à pouvoir gérer de plus grands environnements. C'est à ce niveau que la représentation factorisée peut aider à réécrire ces équations sans les sommations sur tous les états. Dans la prochaine section, nous présentons comment nous avons adapté ces équations de manière à itérer sur des sous-ensembles d'états au lieu de le faire sur l'espace d'états au complet.

3.2. *Utilisation de la représentation factorisée*

Pour illustrer le fonctionnent de la factorisation, supposons un environnement qui peut être décrit par 3 variables aléatoires X_1, X_2 et X_3. Chacune de ces variables peut prendre respectivement 5, 6 et 4 valeurs différentes ; l'environnement a donc 120 états. Supposons également que ces variables sont indépendantes entre elles. Un état de croyance b possible est illustré à la figure 1. Chaque vecteur est associé à une variable aléatoire, et chaque entrée d'un vecteur contient la probabilité que la variable prenne une certaine valeur.

Prenons le cas où l'agent veut calculer la récompense d'être dans l'état de croyance de la figure 1. Avec l'approche traditionnelle qui consiste à parcourir tous les états (équation 2), l'agent doit boucler sur les 120 états. Or, en observant l'état de croyance de l'agent, on se rend compte que plusieurs états sont impossibles. Par exemple, l'agent ne peut être dans aucun des 24 états où la variable X_1 prend la première valeur possible car la probabilité est égale à 0. En fait, il y a seulement 20 états qui sont

$$b = \left(\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.25 \\ 0.05 \\ 0.50 \\ 0 \\ 0.10 \\ 0.10 \end{bmatrix}, \begin{bmatrix} 0.10 \\ 0.30 \\ 0.15 \\ 0.45 \end{bmatrix} \right)$$

Figure 1. *Exemple de la représentation d'un état de croyance.*

possibles à partir de l'état de croyance illustré à la figure 1. Pour calculer le nombre d'états possibles, il suffit de multiplier le nombre de valeurs possibles pour chaque variables ($1 \times 5 \times 4$ pour notre exemple). Pour tenir compte d'une telle factorisation, il convient tout d'abord d'introduire la fonction suivante :

$$\omega(b) = \{\{x_i\}_{i=1}^{M} \mid (\forall\, x_i)\; P_b(X_i = x_i) \neq 0\} \tag{6}$$

Cette fonction retourne tous les états dans lequel l'agent peut être en fonction de l'état de croyance reçu en paramètre. Dans le pire des cas, la fonction retourne S ; c'est-à-dire que tous les états sont possibles. Si les variables sont ordonnées approximativement selon leur degré de certitude, le sous-ensemble d'états peut être calculé plus rapidement puisque chaque fois qu'on rencontre une variable avec une probabilité de zéro, on peut éliminer tous les états correspondants. Par conséquent, l'équation suivante peut être calculée beaucoup plus rapidement que l'équation 2 :

$$R(b) = \sum_{s \in \omega(b)} R(s)b(s) \tag{7}$$

En résumé, la fonction ω retourne l'ensemble des états possibles selon les croyances de l'agent. De plus, le sous-ensemble des états atteignables à partir de ces états possibles peut également être introduit. Pour ce faire, une nouvelle fonction α est introduite. Elle prend en entrée l'état de croyance de l'agent b, l'action exécutée a et l'observation perçue o et elle retourne le sous-ensemble des états atteignables ($\alpha : A \times B \times \Omega \rightarrow \mathbb{P}\, S$) :

$$\alpha(a, b, o) = \{s' \mid (\forall\, s \in \omega(b))\; P(s' \mid s, a) \neq 0 \wedge P(o \mid s', a) \neq 0\} \tag{8}$$

En d'autres mots, la fonction α retourne l'ensemble des états subséquents ayant des probabilités de transition et d'observation différentes de zéro. Il s'ensuit que l'équation suivante, calculant la probabilité d'une certaine observation, peut être calculée beaucoup plus rapidement que l'équation 3, car elle itère sur un nombre réduit d'états :

$$P(o \mid a, b) = \sum_{s' \in \alpha(a,b,o)} P(o \mid s', a) \sum_{s \in \omega(b)} P(s' \mid s, a)b(s) \tag{9}$$

Par ailleurs, à l'aide de la fonction ω, il est aussi possible de redéfinir la fonction de mise à jour des croyances. Si $b' = \tau(b,a,o)$, alors $\forall\, s' \in \alpha(a,b,o)$:

$$b'(s') = \eta P(o \mid s', a) \sum_{s \in \omega(b)} P(s' \mid s, a) b(s) \tag{10}$$

En conséquences, l'équation 1 demeure inchangée, mais les calculs des différentes parties de cette équation sont beaucoup plus rapides. Comme nous venons de le montrer, la rapidité des calculs est due au fait que nous avons remplacé les quatre sommes définies sur tous les états (équations 2, 3 et 4) par des sommes définies sur de plus petits sous-ensembles d'états (équations 6 à 10). Les gains d'une telle approche factorisée sont particulièrement importants lorsque certaines parties de l'environnement sont observables ; c'est-à-dire lorsque l'agent est pratiquement certain de la valeur de certaines variables.

Finalement, avec une représentation factorisée, il est possible de représenter beaucoup plus efficacement le modèle de transition. Avec une approche standard, la probabilité de transition $P(s' \mid s, a)$ doit être définie pour toutes les combinaisons d'états et d'actions. Dans le pire des cas, cela fait $|S|^2 |A|$ probabilités à représenter. En pratique par contre, il y a moyen de compacter la représentation en ne conservant par exemple que les transitions ayant une probabilité non-nulle. Heureusement, lorsque l'on utilise une représentation factorisée, la fonction de transition peut être représentée plus efficacement à l'aide de réseaux bayésiens dynamiques ou autres structures similaires. Nous ne décrivons pas cet aspect dans notre article mais nous référons le lecteur à [BOU 96, HAN 00] pour plus de détails.

3.3. *Variables Dépendantes*

L'approche que nous venons de présenter repose sur l'hypothèse que toutes les variables sont indépendantes. Il se peut toutefois qu'il y ait des dépendances entre certaines variables. S'il y a une dépendance entre certaines variables, il devient alors impossible de maintenir les croyances sur chaque variable indépendamment. Une solution est de regrouper les variables ayant une dépendance entre elles à l'intérieur d'un seul vecteur. De cette manière, un état de croyance est défini par un ensemble de vecteurs contenant un vecteur pour chaque sous-ensemble de variables qui est indépendant des autres sous-ensembles.

Pour rendre cela plus clair, reprenons notre petit exemple avec trois variables X_1, X_2, et X_3. Supposons que les deux dernières variables sont dépendantes entre elles mais que X_1 est totalement indépendante des deux autres variables. On peut alors séparer l'état de croyance en deux vecteurs : le premier étant pour la variable X_1 et le deuxième pour toutes les combinaisons possibles des variables X_2 et X_3 (voir Figure 2). Avec cet exemple, il faut 29 entrées dans l'état de croyance tandis qu'il n'y en avait que 15 lorsque les variables étaient indépendantes.

$$b = \left(\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} P(X_2 = x_1 \wedge X_3 = x_1) \\ P(X_2 = x_1 \wedge X_3 = x_2) \\ \cdots \\ \cdots \\ \cdots \\ P(X_2 = x_6 \wedge X_3 = x_4) \end{bmatrix} \right)$$

Figure 2. *Exemple de la représentation d'un état de croyance lorsqu'il existe des dépendances entre certaines variables.*

Cependant, même si les variables sont dépendantes, il est quand même possible de factoriser l'état de croyance de manière à minimiser la dégradation de la solution. Certaines méthodes ont été développées pour générer automatiquement les regroupements de variables qui trouvent un compromis entre la compacité de la représentation et l'efficacité de l'agent [BOY 98, MCA 99, POU 01].

4. La prise de décision

Comme nous l'avons mentionné précédemment, les approches utilisant la programmation dynamique ne s'appliquent pas vraiment lorsque l'espace d'états devient trop imposant. C'est pourquoi, même en utilisant la factorisation, l'équation 1 demeure non applicable pour des environnements contenant un nombre élevé d'états. Il existe plusieurs méthodes efficaces permettant de trouver des approximations très près de la solution optimale, mais elles sont encore limitées à des problèmes de taille raisonnable. Plutôt que de calculer hors-ligne la politique optimale, notre approche permet plutôt d'évaluer à chaque cycle la meilleure action à faire pour l'agent. L'avantage d'une telle méthode est qu'elle peut être appliquée quelle que soit la taille du POMDP. Cela permet donc d'avoir un modèle pour la prise de décision dans des environnements stochastiques, dont le nombre d'états est très grand.

Dans cette section, nous allons donc décrire en détail le fonctionnement de notre approche en-ligne de prise de décision dans les POMDPs. L'idée de base consiste à construire un arbre où les sommets sont des états de croyance de l'agent et où les arcs sont une combinaison d'une action et d'une observation (voir Figure 3).

4.1. *Approximation de la valeur d'un état de croyance*

Dans la section 3, nous avons décrit comment il est possible de calculer la valeur exacte d'un état de croyance de l'agent en utilisant la programmation dynamique (équation 1). Dans cette section, nous allons plutôt démontrer comment nous avons estimé la valeur d'un état de croyance en utilisant une technique de recherche en profondeur. Pour ce faire, nous avons défini une nouvelle fonction $\delta : B \times \mathbb{N} \rightarrow \mathbb{R}$ qui ressemble beaucoup à l'équation 1 mais qui se base plutôt sur une recherche en pro-

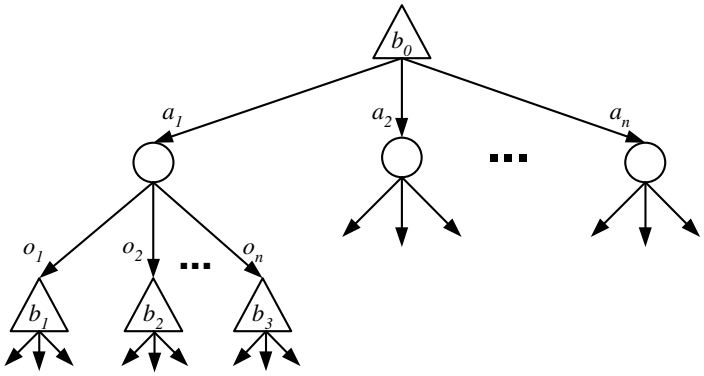

Figure 3. *Un arbre de recherche dans l'espace des états de croyance.*

fondeur. Elle prend en entrée un état de croyance b et elle retourne une estimation de la valeur de b selon une recherche de profondeur p. Lors du premier appel, p est initialisée à P, la profondeur maximale permise pour la recherche.

$$\delta(b,p) = \begin{cases} U(b) & \text{, si } p = 0 \\ R(b) + \gamma \max\limits_{a} \sum\limits_{o \in \Omega} (P(o \mid b,a) \times \delta(\tau(b,a,o), p-1)) & \text{, si } p > 0 \end{cases} \quad (11)$$

où $R(b)$ est évaluée selon l'équation 7 et $P(o \mid b,a)$ est calculée selon l'équation 9.

Dans le cas où $p = 0$, cela veut dire que l'on a atteint une feuille de notre arbre de recherche. Dans ce cas, la valeur de cet état de croyance est donnée par une fonction d'utilité $U(b)$. Cette fonction $U(b)$ permet d'avoir un aperçu de la valeur réelle d'être dans cet état de croyance (si la fonction $U(b)$ était parfaite, alors il serait inutile de faire une recherche). Cette fonction d'utilité doit être définie pour chaque problème.

Dans le cas où $p > 0$, la valeur d'un état de croyance pour une profondeur $P-p$ est tout simplement la récompense courante plus les récompenses escomptées des états de croyance plus bas dans l'arbre.

Finalement, pour trouver la politique dans un certain état de croyance il suffit de faire une recherche en profondeur pour trouver quelle action effectuer.

$$\pi_t(b,P) = \operatorname*{argmax}_{a} \sum_{o \in \Omega} P(o \mid b,a) \delta(\tau(b,a,o), P-1) \quad (12)$$

Il faut noter que cette équation n'est pas utilisée pour choisir l'action à effectuer pour tous les états de croyance, mais uniquement pour l'état de croyance courant de l'agent lorsque vient le temps de prendre une décision.

4.2. *Algorithme RTBSS*

Dans cette section, nous présentons en détails notre algorithme RTBSS (Real-Time Belief Space Search) que nous utilisons pour construire l'arbre de recherche ainsi que pour trouver la meilleure action. Il est important de noter que, contrairement à la grande majorité des approches de résolution de POMDPs, notre approche est appliquée en-ligne. Cela veut donc dire que notre algorithme doit être exécuté à chaque fois que l'agent doit faire une action dans l'environnement. L'algorithme RTBSS (voir Algorithme 1) est basé sur l'équation 11 que nous venons de présenter. Nous détaillons ici une version en pseudo-code de l'algorithme qui donne un aperçu plus détaillé de l'implémentation de notre approche.

Notre algorithme est basé sur une recherche en profondeur d'abord à laquelle on applique une technique de « *Branch and Bound* » afin d'élaguer certaines branches de l'arbre. La façon dont l'algorithme fonctionne est qu'il explore d'abord un chemin dans l'arbre jusqu'à la profondeur de recherche maximale P. Ensuite, il calcule la valeur de cette branche qui servira par la suite de borne inférieure sur la valeur espérée maximale.

Lors de la recherche, l'algorithme évalue, à chacun des sommets de l'arbre, s'il est possible d'améliorer la borne inférieure en continuant la recherche jusqu'à une profondeur de P. C'est la fonction ÉLAGAGE, à la ligne 10, qui décide si la recherche doit être poursuivie. Cette fonction évalue, à l'aide d'une heuristique, s'il est possible d'obtenir une meilleure valeur que la borne inférieure courante. La fonction heuristique retourne une estimation de la meilleure valeur d'utilité qui pourrait être trouvée si la recherche était poursuivie jusqu'à une profondeur de P. Par conséquent, si l'heuristique retourne une valeur supérieure à la borne, alors la recherche va se poursuivre car il y a une certaine probabilité non nulle que la meilleure solution réside dans ce sous-arbre. Dans le cas contraire, l'algorithme va pouvoir effectuer un retour arrière et tout le sous-arbre sera ignoré.

Par ailleurs, la fonction heuristique utilisée par la fonction ÉLAGAGE doit être définie pour chacun des problèmes. De plus, afin de bien fonctionner, la fonction heuristique doit toujours surestimer la véritable valeur d'utilité espérée de l'état de croyance courant. Si l'heuristique sous-estime la vraie valeur, cela pourrait avoir pour effets d'élaguer certaines parties de l'arbre contenant la meilleure solution. D'un autre côté, si l'heuristique surestime, alors seuls des sous-arbres non-intéressants seront ignorés et la solution retournée sera la même que celle trouvée par force brute. Finalement, lorsque l'algorithme arrive sur une feuille et que la valeur espérée est plus grande que la borne inférieure, alors la borne inférieure est mise à jour.

Toutefois, il est important de noter que l'heuristique n'est pas essentielle au bon fonctionnement de l'algorithme. Si pour un problème donné, il est trop coûteux d'utiliser une heuristique ou s'il est impossible d'en définir une, alors on n'a qu'à faire en sorte que la fonction ÉLAGAGE retourne toujours faux. À ce moment, l'algorithme de recherche dans l'arbre devient un algorithme de recherche en profondeur d'abord limitée. Bien entendu, s'il n'y a pas d'élagage, l'algorithme est plus lent, mais il retourne

1: **Procédure** RTBSS(*b, p , rAcc*)

Entrées : *b* : L'état de croyance de l'agent.
p : la profondeur restante à la recherche.
rAcc : les récompenses accumulées jusqu'au noeud courant.
Statiques : *P* : La profondeur maximale de la recherche.
meilleureValeur : La meilleure valeur rencontrée lors de la recherche.
action : La meilleure action trouvée.

2: **Si** $p = 0$ **Alors**
3: $valeurFinale \leftarrow rAcc + \gamma^P \times U(b)$
4: **Si** $valeurFinale > meilleureValeur$ **Alors**
5: $meilleureValeur \leftarrow valeurFinale$
6: **Fin Si**
7: **retourner** *valeurFinale*
8: **Fin Si**
9: $rAcc \leftarrow rAcc + \gamma^{P-p} \times R(b)$
10: **Si** ÉLAGAGE(*b, rAcc, p*) **Alors**
11: **retourner** $-\infty$
12: **Fin Si**
13: $listeAction \leftarrow$ TRIER(*b, A*)
14: $max \leftarrow -\infty$
15: **Pour tout** $a \in listeAction$ **Faire**
16: $rEspérées \leftarrow 0$
17: **Pour tout** $o \in \Omega$ **Faire**
18: $b' \leftarrow \tau(b, a, o)$
19: $rEspérées \leftarrow rEspérées + \gamma^{P-p} \times P(o \mid a, b) \times \text{RTBSS}(b', p-1, rAcc)$
20: **Fin Pour**
21: **Si** $(p = P \wedge rEspérées > max)$ **Alors**
22: $max \leftarrow rEspérées$
23: $action \leftarrow a$
24: **Fin Si**
25: **Fin Pour**
26: **retourner** *max*

Algorithme 1: Notre algorithme RTBSS pour la prise décision en temps-réel.

tout de même les mêmes résultats. La complexité en pire cas d'un tel algorithme est de $O((\mid A \mid \times \mid \Omega \mid)^p)$.

Pour faire un lien entre l'algorithme et les équations présentées, mentionnons que la ligne 19 de l'Algorithme 1 correspond à $\sum_{o \in \Omega} P(o \mid b, a)\delta(\tau(b, a, o), p-1)$ dans l'équation 11. Par ailleurs, la fonction $\tau(b, a, o)$, ligne 18, retourne le nouvel état de croyance si l'observation o est observée après que l'agent ait exécuté l'action a dans l'état de croyance b.

De plus, notons que la fonction TRIER à la ligne 13 trie les actions de manière à essayer les actions les plus prometteuses en premier car cela risque d'engendrer plus de coupures dans l'arbre de recherche. Encore une fois, comme pour l'élagage, le tri n'est pas essentiel, mais s'il est possible de trier les actions, alors cela permet d'accélérer la recherche en trouvant de meilleures bornes plus rapidement.

Avec cet algorithme, nous sommes en mesure, à chaque cycle, de trouver l'action ayant la valeur espérée maximale jusqu'à un horizon de P. Évidemment, la performance de notre algorithme dépend fortement de la profondeur de la recherche effectuée. C'est pourquoi nous suggérons l'utilisation de l'élagage jumelé au tri des actions pour accélérer la recherche et permettre ainsi une recherche plus profonde dans le temps alloué.

La complexité de la recherche en pire cas est de $O((\mid A \mid \times \mid \Omega \mid)^p)$. Avec un bon heuristique toutefois, il est possible de faire beaucoup mieux. Notre algorithme est efficace si le nombre d'actions et d'observations restent petits. Sinon, le facteur de branchement devient trop grand et la recherche ne peut pas être effectuée assez profondément. S'il y a plusieurs observations, il est possible d'échantillonner les observations de manière à explorer seulement les observations les plus probables [MCA 99]. Toutefois, pour cet article, nous avons uniquement considéré l'exploration de toutes les observations.

5. Expérimentations

Cette section présente d'abord les résultats sur deux problèmes que l'on retrouve dans la littérature des POMDPs : *Tag* [PIN 03] et *RockSample* [SMI 04]. Bien que notre but premier soit de développer une méthode qui peut être appliquée dans l'environnement de simulation de la RoboCupRescue, il est très difficile de bien évaluer les performances de notre approche sur un environnement aussi complexe. Pour cette raison, nous comparons d'abord notre approche avec les meilleurs algorithmes actuels sur des environnements de plus petite taille.

5.1. *Tag*

Nous avons expérimenté notre algorithme sur *Tag*, présenté pour la première fois dans [PIN 03]. Cet environnement a aussi été utilisé très récemment dans [POU 03,

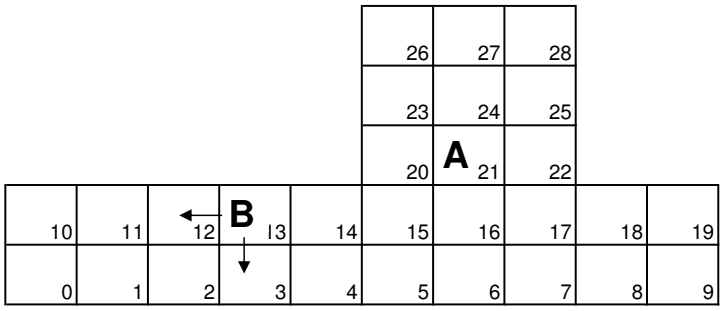

Figure 4. *Le problème de Tag.*

VLA 04, SPA 04, SMI 04, BRA 04]. Cet environnement, de grandeur moyenne (870 états), demande l'utilisation d'une méthode de résolution approximative des POMDPs. Il nous permet de pouvoir comparer notre approche à plusieurs autres approches récentes. Une telle comparaison ne serait pas possible sur de trop grands POMDPs car la majorité des autres approches ne seraient pas applicables comme on le verra plus tard. Comme cet environnement est tout de même de grandeur relativement petite, les comparaisons de cette section ne font pas ressortir à sa juste mesure la performance de notre algorithme qui est lui applicable à des environnements beaucoup plus grands. Toutefois, les résultats montrent que notre algorithme peut obtenir des résultats de bonne qualité et ce dans des temps très courts .

5.1.1. *Description de l'environnement*

L'environnement *Tag* consiste en un agent *A* qui poursuit un autre agent *B* afin de l'attraper. La configuration de l'environnement est présentée à la Figure 4. Une particularité de ce problème est que l'agent *A* ne peut pas percevoir son adversaire, sauf si les deux agents occupent la même position. Le but est donc pour l'agent *A* de se promener dans l'environnement vers les endroits où il pense que l'agent *B* pourrait se trouver et s'ils sont sur la même case, alors *A* n'a qu'à toucher *B* pour gagner.

Un état de ce problème est le produit de deux variables : $X_1 = \{0 \dots 28\}$ est l'état de l'agent *A* et $X_2 = \{0 \dots 28$ et touché$\}$ est l'état de l'agent *B*. L'agent peut choisir parmi 5 actions : *Nord*, *Sud*, *Est*, *Ouest* et *Toucher*. La position initiale des deux agents est décidée aléatoirement (en excluant le cas trivial où les 2 agents partent sur la même case). Dans ce problème, les fonctions de transition et d'observation sont déterministes, c'est-à-dire qu'il n'y a pas d'incertain au niveau des actions et des observations de l'agent *A*. Cependant, l'environnement demeure fortement partiellement observable car l'agent *A* ne sait pas où se trouve son adversaire.

L'agent *B*, de son coté, a une vision parfaite et s'éloigne de *A* avec une probabilité de 0.8 et reste sur place avec une probabilité de 0.2. Par exemple, si *B* peut faire 2 actions pour s'éloigner de *A* (voir exemple sur la Figure 4), alors chacune des ces 2 actions auraient une probabilité de 0.4 d'être exécutées, et toujours 0.2 de rester sur

Méthode	Récompense	Temps hors-ligne (s)	Temps en-ligne (s)
Q_{MDP}	-16,75	11,8	-
RTBSS	**-10,56**	**0**	**0,23**[1]
PBVI [PIN 03]	-9,18	180880	-
BBSLS [BRA 04]	$\backsim$ -8.3	$\backsim$ 100000	-
BPI [POU 05]	-6,65	250	-
HSVI [SMI 04]	-6,37	10113	-
Perséus [SPA 04]	-6,17	1670	-

Tableau 1. *Comparaison de notre approche pour le problème de Tag. Pour RTBSS, la récompense et le temps de calcul sont des moyennes sur 5000 simulations.*

place. Donc malgré le fait que *A* soit pratiquement aveugle, il est quand même en mesure d'avoir une bonne idée de la position de son adversaire sachant que celui-ci n'a comme seul but que de s'éloigner.

Finalement, l'agent reçoit une récompense de -1 pour chaque déplacement qu'il effectue dans l'environnement et une récompense spéciale pour l'action de *Toucher* : une récompense de $+10$ si l'action réussit et -10 si l'adversaire n'est pas présent. Les récompenses sont additionnées suivant un facteur d'escompte $\gamma = 0.95$.

5.1.2. *Résultats*

En comparant notre approche avec les approches existantes (voir Tableau 1), on constate que notre algorithme s'exécute très rapidement. Il ne requiert aucun temps hors-ligne et à chaque cycle il ne prend que quelques dixièmes de secondes pour choisir l'action que l'agent devrait effectuer. Les résultats obtenus se rapprochent des meilleurs résultats des autres approches, mais avec un temps de calcul beaucoup plus petit.

On constate également que notre algorithme s'exécute plus rapidement que Q_{MDP} et permet d'obtenir un résultat considérablement meilleur. Il convient de préciser que Q_{MDP} consiste à résoudre le problème comme s'il était totalement observable. Cela revient donc à résoudre le PDM (Processus Décisionnel de Markov) sous-jacent au POMDP et la résolution des PDMs peut se faire assez rapidement pour des problèmes de taille raisonnable.

Toutefois, l'environnement *Tag* ne rend pas tout à fait justice à notre algorithme car on pourrait affirmer qu'il est préférable d'appliquer un algorithme hors-ligne afin d'obtenir un meilleur résultat, même s'il faut attendre quelques dizaines de minutes avant d'obtenir une solution. Le problème de ces algorithmes, qui s'avèrent particulièrement efficaces sur le problème de *Tag*, est qu'ils ne pourraient pas être appliqués

1. Ce temps correspond au temps moyen en-ligne pour choisir une action.

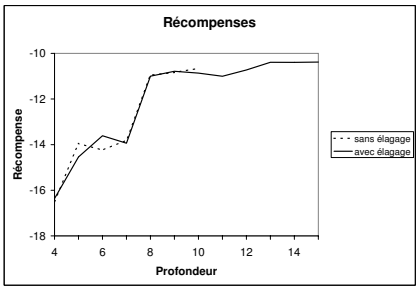

Figure 5. *Récompense moyenne pour différentes profondeurs de recherche.*

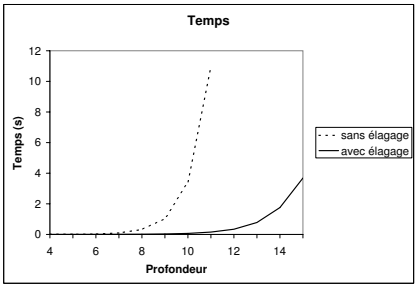

Figure 6. *Temps de délibération moyen pour une décision selon la profondeur de recherche.*

à des problèmes avec plusieurs dizaines de milliers d'états alors que notre algorithme peut être appliqué sans difficulté sur des problèmes de très grande taille. À vrai dire, la vitesse d'exécution de notre algorithme ne dépend pas du nombre d'états mais plutôt du nombre d'actions et d'observations possibles. Le but de cette comparaison était plutôt de faire ressortir que notre algorithme, quoiqu'il ne puisse pas rivaliser avec les meilleurs algorithmes au niveau de la qualité de la solution pour de petits environnements, permet tout de même d'obtenir rapidement des résultats intéressants. D'ailleurs, RTBSS permet de trouver une solution bien meilleure que celle trouvée par la populaire méthode heuristique Q_{MDP} et ce en un temps quasiment instantané.

À la figure 5 on peut voir la performance de notre algorithme sur le problème de *Tag* en fonction de la profondeur de recherche utilisée. On constate que les récompenses sont sensiblement les mêmes, que l'on utilise l'élagage ou non. Ceci montre que le fait d'utiliser notre heuristique pour l'élagage ne diminue en rien les performances de notre agent. Par ailleurs, comme on peut le voir, à partir d'une certaine profondeur, l'agent gagne très peu à rechercher plus profondément. Par conséquent, il devient important d'analyser la profondeur la plus utile pour un problème. Souvent, comme c'est le cas dans le problème de *Tag*, le temps de calcul d'une recherche plus en profondeur ne vaut pas nécessairement la peine par rapport au temps supplémentaire de recherche qu'une telle profondeur impose. En d'autres mots, pour chaque problème, il faut trouver le bon compromis entre performance et vitesse de calcul.

À la figure 6, nous comparons notre algorithme RTBSS par rapport à une version sans élagage. On constate qu'implanter une heuristique pour élaguer l'arbre de recherche peut grandement améliorer la performance. Le temps de calcul de notre algorithme croît toujours de façon exponentielle étant donné qu'il s'agit d'une recherche dans un arbre, mais la croissance est de beaucoup atténuée. Notre algorithme est jusqu'à 50 fois plus rapide pour un même problème que la version sans élagage, ce qui est particulièrement souhaitable dans un environnement temps-réel où les décisions doivent être rendues très rapidement.

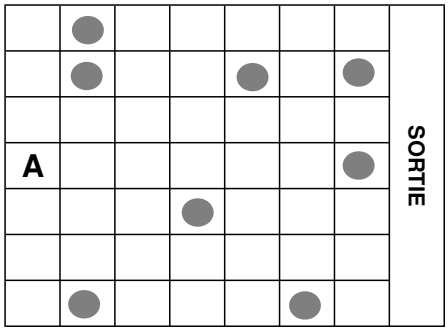

Figure 7. *RockSample[7,8].*

5.2. *RockSample*

Nous avons également testé notre approche sur le problème de *RockSample*, présenté pour la première fois dans [SMI 04]. Pour ce problème, nous effectuons une comparaison uniquement avec l'algorithme HSVI (qui était un des plus performants sur l'environnement de *Tag*).

5.2.1. *Description du problème*

RockSample modélise un problème d'un robot qui doit explorer son environnement et ramasser des roches (voir Figure 7), un peu comme un vrai robot devrait faire sur la planète Mars. Le robot peut percevoir des récompenses en échantillonnant des roches ou en sortant de l'environnement (en traversant l'environnement à la droite). Certaines roches ont une bonne valeur scientifique et d'autres non ; le robot doit donc échantillonner seulement celles qui sont intéressantes. Le robot connaît la position de chacune des roches mais ignore totalement la valeur scientifique de celles-ci. On considère qu'une roche peut être bonne ou mauvaise. Afin de vérifier si une roche est bonne, le robot peut utiliser un capteur imparfait. Le capteur permet d'avoir une certaine idée de la qualité d'une roche.

On définit *RockSample*$[n, k]$ comme étant une instance du problème de *RockSample* où la grille de l'environnement est d'une taille de $n \times n$ et où il y a k roches. Un état du problème est donc caractérisé par $k + 1$ variables : *Position* $= (1, 1), (1, 2), \ldots, (n, n)$ et k variables *Roche*$_i$ $= \{$*bonne*, *mauvaise*$\}$. Il y a aussi un état final supplémentaire à la droite de l'environnement. Cela fait donc que pour *RockSample*$[n, k]$, il y a $n^2 \times 2^k$ états. Par ailleurs, l'agent peut effectuer $k+5$ actions : {*Nord*, *Sud*, *Est*, *Ouest*, *Échantillonner*, *Vérifier*$_1$, $\ldots$, *Vérifier*$_k$}. Les actions qui consistent à se déplacer sont totalement déterministes. Par contre, les actions *Vérifier*$_i$ sont stochastiques et dépendent de la distance du robot par rapport à la distance de la roche qu'il désire vérifier avec son capteur. C'est-à-dire que plus le robot est près d'une roche, plus son capteur est précis.

Problème	Taille	Récompense	Temps hors-ligne (s)	Temps en-ligne (s)
RockSample[4,4]	257s,9a,2o			
RTBSS		16.2	0	0.1[2]
PBVI [SMI 04][3]		17.1	$\sim$ 2000	-
HSVI [SMI 04]		18.0	577	-
RockSample[5,5]	801s,10a,2o			
RTBSS		18.7	0	0.1[3]
HSVI [SMI 04]		19.0	10208	-
RockSample[5,7]	3201s,12a,2o			
RTBSS		22.6	0	0.1[3]
HSVI [SMI 04]		23.1	10263	-
RockSample[7,8]	12545s,13a,2o			
RTBSS		20.1	0	0.2[3]
HSVI [SMI 04]		15.1	10266	-

Tableau 2. *Comparaison de notre approche pour le problème de RockSample. La taille est indiquée en fonction du nombre d'états, d'actions et d'observations. Par exemple, (257s,9a,2o) signifie que l'environnement a 257 états, 9 actions et 2 observations. Pour RTBSS, les récompenses et les temps de calcul sont des moyennes sur 1000 simulations.*

Une récompense de 10 est allouée au robot lorsqu'il échantillonne une bonne roche alors qu'une pénalité de -10 est donnée s'il échantillonne une mauvaise roche, d'où l'importance d'utiliser son capteur afin d'avoir une idée de la qualité de la roche. Finalement, il y a un facteur d'escompte $\gamma = 0.95$.

5.2.2. *Résultats*

Nous avons comparé notre approche avec HSVI sur quatre instances du problème de *RockSample*. Les résultats sont illustrés au Tableau 2. On constate que nous sommes capables d'atteindre des résultats très similaires et ce sans utiliser de temps de calcul hors-ligne. RTBSS offre de meilleures performances particulièrement lorsque le nombre d'états est très grand. La différence entre HSVI et RTBSS sur la plus grande instance provient du fait que HSVI n'a pas pu converger dans le temps alloué. Ceci montre un grand avantage de notre approche sur les grands environnements. Il est plus avantageux d'utiliser un algorithme en-ligne qu'un algorithme hors-ligne qui n'a pas eu le temps de converger. Plus l'environnement est grand, plus notre approche se démarque des approches hors-lignes.

2. Ces temps correspondent au temps moyen en-ligne pour choisir une action, contrairement au temps hors-ligne pour les autres approches.
3. L'algorithme PBVI a été présenté dans [PIN 03], mais le résultat sur l'environnement *RockSample* a été publié dans [SMI 04]

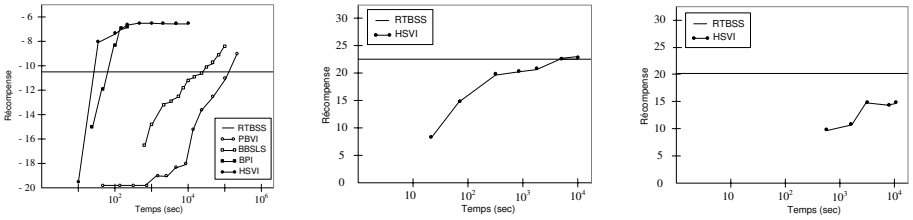

Figure 8. *Qualité de la solution en fonction du temps de calcul hors-ligne pour trois problèmes de complexité grandissante : Tag(gauche), RockSample[5,7](centre) et RockSample[7,8](droite). La ligne représentant les performances de RTBSS est différente des autres, car RTBSS n'utilise aucun temps de calcul hors-ligne. Ces graphiques montrent le temps nécessaire aux approches hors-lignes pour rattraper (si possible) les performances que RTBSS peut obtenir immédiatement.*

5.3. *Temps de calcul hors-ligne*

Un autre avantage de notre algorithme est qu'il peut facilement être appliqué si certains paramètres de l'environnement changent. Par exemple, dans *RockSample*, la position des roches est fixée et connue de l'agent. C'est-à-dire que HSVI, qui procède hors-ligne, trouve une politique pour une seule configuration de roches. Si on veut changer les roches d'endroit, il faut recalculer une nouvelle politique. Avec notre approche en-ligne, il suffit de relancer notre agent dans l'environnement et cela n'aura aucun impact sur ses performances. Par conséquent, notre algorithme RTBSS est plus adapté aux environnements où les configurations initiales peuvent changer et lorsque les agents doivent être déployés rapidement. Par exemple, lors d'opération de secours, les agents doivent être déployés rapidement et ils n'ont pas le temps d'apprendre une bonne politique.

La Figure 8 montre que notre algorithme RTBSS est le meilleur s'il y a très peu de temps alloué aux calculs hors-lignes. Sur le plus petit environnement *Tag*, les approches hors-lignes rattrapent rapidement les performances de RTBSS, mais sur le plus grand environnement (*RockSample*[7,8]), l'algorithme hors-ligne n'est pas capable de rattraper RTBSS. Sur de grands environnements, les approches hors-lignes demandent beaucoup de temps de calcul avant d'obtenir de bons résultats, alors que RTBSS peut obtenir de bons résultats immédiatement. Ceci devient très important si les agents doivent être déployés rapidement dans un environnement qui peut changer d'une exécution à l'autre.

6. Extension pour des environnements complexes

Dans cette section, nous allons présenter une extension de notre méthode de base permettant de l'appliquer efficacement dans des environnements très complexes. Plus particulièrement, nous nous attardons ici sur des environnements ayant un espace

d'états énorme, voire même infini. Dans de tels environnements, les fonctions de transition et d'observation sont difficilement définissables, car il y a trop de choses qui bougent dans l'environnement. L'agent ne peut tout simplement pas tenir compte de tout ce qui peut changer dans l'environnement. Par conséquent, nous considérons que l'agent n'a pas de modèles complets et qu'il ne peut se fier qu'à un modèle partiel de l'environnement pour prendre ses décisions. Le meilleur exemple d'un environnement de ce type est le monde réel, mais pour nos tests, nous avons plutôt utilisé l'environnement de simulation de la RoboCupRescue.

Un des avantages d'avoir une représentation structurée est qu'elle nous permet de spécifier les effets des actions sur les différentes variables aléatoires représentant l'environnement. Une action a très peu de chance d'affecter toutes les variables utilisées pour décrire un état du système. Par conséquent, pour chaque action nous pouvons spécifier les variables de l'environnement qui ont des chances d'être modifiées. Nous pouvons donc définir la fonction de transition uniquement sur ces variables, ce qui a comme effet de diminuer considérablement les probabilités à définir.

Pour cela, nous avons modifié la règle de mise à jour des croyances de l'agent (équation 4) en nous basant sur l'hypothèse que l'agent n'a pas à maintenir activement ses croyances sur toutes les variables. L'agent met à jour ses croyances en se basant sur ses observations et sur ses connaissances à propos de la dynamique de l'environnement. Pour pouvoir raisonner rapidement, l'agent considère certaines parties du monde comme étant statiques. Donc, l'agent ne maintient activement ses croyances que sur les variables directement affectées ou qui affectent directement ses actions. Pour toutes les autres variables, l'agent se fie uniquement à ses observations pour mettre à jour leur valeur. L'agent considère qu'elles ont toujours la même valeur jusqu'à ce qu'il reçoive un nouveau message ou perçoive une nouvelle valeur.

En résumé, nous pouvons voir cette approche comme une façon pour l'agent de porter son attention sur les parties les plus importantes de son environnement.

7. Expérimentations sur l'environnement RoboCupRescue

Dans cette section, nous présentons d'abord l'environnement de simulation de la RoboCupRescue et, par la suite, la façon dont nous avons appliqué notre approche à cet environnement très complexe. Le but de la simulation RoboCupRescue est de construire un simulateur d'équipes d'urgence agissant dans un large environnement urbain ravagé par un tremblement de terre [KIT 00]. Au début d'une simulation, un large tremblement de terre survient dans une ville et suite à ce désastre, plusieurs édifices prennent feu ou bien s'effondrent, les civils sont enfouis sous les décombres et plusieurs routes de la ville sont bloquées rendant la circulation très difficile. Le but du projet, qui prend la forme d'une compétition internationale annuelle, est de concevoir des agents de sorte à minimiser les dommages tels que les pertes de vies et les édifices brûlés. Dans une simulation typique, il y a environ de 30 à 40 agents de six types différents :

Agents pompiers Il y a entre 0 et 15 agents de ce type. Leur but est d'éteindre les feux qui se sont développés suite au tremblement de terre. Chaque agent pompier est en contact radio avec les autres pompiers ainsi qu'avec la centrale de pompier.

Agents policiers Il y a entre 0 et 15 agents de ce type. Leur but est de débloquer les routes afin que la circulation puisse se faire normalement. Chaque agent policier est en contact radio avec les autres policiers ainsi que la centrale de police.

Agents ambulanciers Il y a entre 0 et 8 agents de ce type. Leur but est de parcourir les édifices à la recherche de civils blessés et ensuite de les transporter à l'hôpital. Chaque ambulancier est en contact radio avec les autres ambulanciers ainsi que la centrale d'ambulance.

Agents centres Il y a trois types d'agents centres : la centrale de pompier, la centrale de police et la centrale d'ambulance. Les seules actions que ces agents peuvent faire sont d'envoyer et de recevoir des messages. Les centrales sont en contact avec leurs agents mobiles ainsi qu'avec les autres agents centres.

Dans la simulation, chaque agent ne peut percevoir qu'une faible partie de l'environnement autour de lui. Aucun agent n'a donc une vision complète de l'environnement. De plus, la RoboCupRescue est un environnement que l'on peut caractériser de *collectivement partiellement observable* [NAI 02]. Cela signifie que même si tous les agents mettent leurs connaissances en commun, ils n'auront pas une connaissance complète de l'environnement. Cette incertitude sur l'environnement complique donc de beaucoup le problème de prise de décision dans un tel environnement. Les agents peuvent également communiquer entre eux afin de s'entraider à avoir une meilleure connaissance de l'environnement. Cependant, la communication est limitée de sorte qu'un agent ne peut pas partager toute l'information qu'il connaît.

Un point important de la simulation est que tous les agents connaissent leur position exacte dans la ville, comme s'ils étaient munis d'un GPS (Global Positioning System). Ils ignorent quelles routes sont bloquées et quels édifices sont en feu, mais ils savent se déplacer parfaitement dans la ville.

L'approche présentée dans cet article a été appliquée seulement pour les agents policiers. Leur tâche est de déblayer les routes les plus importantes et ce le plus rapidement possible. Cette tâche est cruciale pour permettre aux agents pompiers et ambulanciers d'accomplir leurs tâches. Cependant, il n'est pas facile de décider comment les agents policiers doivent se déplacer étant donné qu'ils ne possèdent que très peu d'information sur l'environnement. De plus, certaines routes sont plus importantes que d'autres, par exemple une route près d'un feu devrait être débloquée rapidement car il y a de fortes chances que des agents pompiers veulent emprunter cette route.

Dans cette section, nous allons donc présenter comment nous avons appliqué notre approche dans le cas de la RoboCupRescue. Il convient de préciser que nous nous sommes en fait restreints au sous-problème suivant de la RoboCupRescue : Ayant une connaissance partielle des routes qui peuvent être bloquées ou non, des édifices en feu et de la position des autres agents, quelle séquence d'actions un agent policier devrait-t-il effectuer ?

7.1. *RoboCupRescue modélisé en POMDP*

Comme nous l'avons déjà précisé, nous avons seulement modélisé le problème des agents policiers comme un POMDP. Dans ce contexte, les différentes actions qu'un agent policier peut faire sont : *Nord*, *Sud*, *Est*, *Ouest* et *Débloquer*. Un état de l'environnement peut être décrit par un ensemble de variables aléatoires (environ 1500 variables dans une simulation typique).

Routes : Il y a environ 800 routes dans une simulation et chaque route peut être bloquée ou dégagée. Cela fait 2^{800} configurations possibles.

Édifices : Il y a environ 700 édifices dans une simulation. Nous considérons qu'un édifice peut être en feu ou non. Cela fait 2^{700} configurations possibles.

La position des agents : Un agent peut être sur une des 800 routes et il y a environ de 30 à 40 agents. Cela fait au moins 800^{30} configurations possibles.

Donc si nous estimons le nombre possibles d'états, nous obtenons $2^{800} \times 2^{700} \times 800^{30}$ états. Cependant, une grande majorité de ces états sont inaccessibles au fur et à mesure que les agents perçoivent leur environnement. Comme on peut voir, l'espace d'états d'un tel environnement temps-réel est beaucoup trop volumineux pour espérer trouver une politique optimale. Il faut plutôt se rabattre sur une méthode permettant de trouver une solution acceptable à l'intérieur d'un très court délai.

De plus, comme les cartes changent d'une exécution à l'autre, les agents se doivent d'être efficaces même s'ils n'ont jamais vu la carte. Lors des compétitions, il n'y a pas de temps pour apprendre avant l'exécution. Les agents sont exécutés une seule fois dans une ville qu'ils ne connaissent pas et ils doivent être efficaces immédiatement. C'est pourquoi notre algorithme RTBSS est très intéressant, parce qu'il permet de bonnes performances sans calculs hors-lignes.

7.2. *Application de notre approche*

Cette section présente comment nous avons appliqué RTBSS dans la simulation de la RoboCupRescue. Dans cet environnement, la recherche en-ligne dans l'espace des états de croyance représente une recherche dans les chemins possibles qu'un agent policier peut prendre. Dans l'arbre, la probabilité de passer d'un état de croyance à un autre dépend de la probabilité que la route soit bloquée. Une spécificité de ce problème est que l'on doit retourner un chemin au simulateur, donc l'algorithme RTBSS a été légèrement modifié pour retourner la meilleure branche de l'arbre au lieu d'uniquement la première action.

Par ailleurs, l'aspect clef de notre approche est que l'on considère plusieurs variables de l'environnement comme étant fixes durant la recherche dans le but de minimiser le nombre d'états considérés. Par exemple, supposons que l'agent doit évaluer un état de croyance. En considérant le problème général, l'agent devrait itérer sur énormément d'états, parce que plusieurs états sont possibles. Par conséquent, l'ensemble retourné par l'équation 6 serait encore très grand.

Notre idée consiste à réduire encore plus ce sous-ensemble d'états. Pour se faire, l'agent va supposer que certaines parties de l'environnement ne changent pas durant sa recherche dans l'arbre des états de croyance. Dans la RoboCupRescue, toutes les variables sont considérées comme fixes à l'exception de la position de l'agent et des variables décrivant les routes. Pour toutes les variables fixes, comme la position des autres agents et la position des feux, l'agent considère quelles conservent la dernière valeur perçue. Par conséquent, toutes ces variables fixes sont représentées dans l'état de croyance par un vecteur ne contenant que des zéros à l'exception de la dernière valeur perçue qui a une probabilité de 1. Il s'ensuit que la fonction ω (équation 6) ne va retourner qu'un faible sous-ensemble d'états.

Plus précisément, les croyances de l'agent ne sont maintenues que pour les variables décrivant les routes. Ces variables sont les plus importantes pour les décisions de l'agent. En d'autres mots, l'agent se concentre sur les variables les plus importantes en maintenant ses croyances le plus précisément possible. De plus, il abstrait les autres variables en considérant quelles sont fixes et en se fiant uniquement à ses observations pour maintenir ses croyances.

Les variables fixes ne sont pas ignorées, elles sont considérées durant la recherche, mais l'agent se fie aux dernières valeurs perçues et il n'essaie pas de les faire évoluer. Par exemple, si un agent pompier est considéré comme étant sur la route r_3, il va rester à cet endroit tout au long de la recherche. Nous savons qu'en pratique il va sûrement bouger, mais pour simplifier la recherche, nous considérons qu'il reste à la même position. La valeur des variables fixes ne sont mises à jour que lorsque l'agent perçoit une nouvelle valeur. Dans notre modèle, nous considérons les perceptions comme étant à la fois les perceptions directes de l'agent et les informations reçues par messages. Nous sommes dans le cadre d'un système multiagent coopératif, donc les agents se font confiance et toutes les informations reçues sont considérées comme véridiques.

Dans le cas de la RoboCupRescue, nous aurions pu essayer d'estimer la position de tous les autres agents en se basant sur leur dernière position perçue, sur leurs tâches et sur leur vitesse de déplacement, mais comme nous n'avons pas de modèle de transition, ces probabilités ne seraient que très approximatives. Par ailleurs, le temps de calculs pour avoir de bonnes approximations serait prohibitif. Par conséquent, l'amélioration minime des performances qui aurait pu en résulter ne justifierait pas du tout le temps de calcul nécessaire.

Dans des environnements complexes et dynamiques, il est souvent plus efficace de se fier aux observations de l'agent plutôt que d'essayer de tout prévoir. Il y a tout simplement trop de chose qui bougent dans la simulation. Par conséquent, l'agent devrait concentrer ses efforts sur les parties les plus importantes de l'environnement. Pour tenir compte efficacement de toutes les autres parties de l'environnement, l'agent peut réduire sa boucle de perception et d'action pour maintenir ses croyances à jour. Ceci est possible, car notre algorithme RTBSS peut trouver une action très rapidement. Par conséquent, l'agent fait plus fréquemment des observations, donc il n'a pas besoin d'un modèle pour les parties les moins importantes de l'environnement, car elles n'auront pas le temps de se déplacer énormément entre les observations.

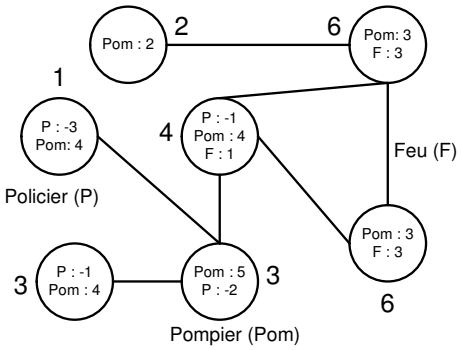

Figure 9. *Graphe pour la fonction de récompense.*

7.3. *Coordination des agents*

Dans un autre ordre d'idée, nous avons aussi défini une fonction de récompense dynamique retournant la récompense pour le déblaiement d'une route, en fonction de la position des autres agents et des feux. Ceci permet à l'agent de calculer efficacement la récompense espérée selon son état de croyance courant, sans avoir à emmagasiner explicitement toutes les récompenses pour tous les états possibles. Cette fonction de récompense permet aussi aux agents de coordonner leurs actions.

Un agent policier doit assigner une récompense espérée à chacune des routes de la ville qui sont représentées comme des noeuds dans un graphe (voir Figure 9). Les récompenses évoluent en fonction de la position des agents et des feux, donc l'agent policier doit recalculer les récompenses à chaque tour. Pour calculer les récompenses, l'agent propage des récompenses sur le graphe, en partant des routes où il y a des agents ou des feux à proximité. Par exemple, si un agent pompier est sur la route r_1, alors cette route reçoit une récompense de 5, les routes adjacentes à r_1 dans le graphe reçoivent une récompense de 4, les routes adjacentes aux routes adjacentes de r_1 reçoivent 3, ainsi de suite. Les routes à l'intérieur d'un certain périmètre autour d'un feu reçoivent aussi une récompense.

Sur la figure 9, les noeuds représentent les routes et les provenances des récompenses sont identifiées dans les noeuds. Le nombre à proximité d'un noeud est la récompense globale pour cette route, qui est tout simplement la somme des récompenses identifiées dans le noeud.

Ce qui est intéressant avec cette fonction de récompense est qu'elle peut être utilisée pour coordonner les agents policiers. La coordination est nécessaire pour empêcher les agents d'aller tous sur la même route. Il faut donc maintenir un certain degré de dispersion entre les agents policiers. Pour ce faire, l'agent propage des récompenses négatives à partir des positions des autres agents policiers. En d'autres mots, les agents policiers se repoussent. Avec cette modification de la fonction de récompense, il était

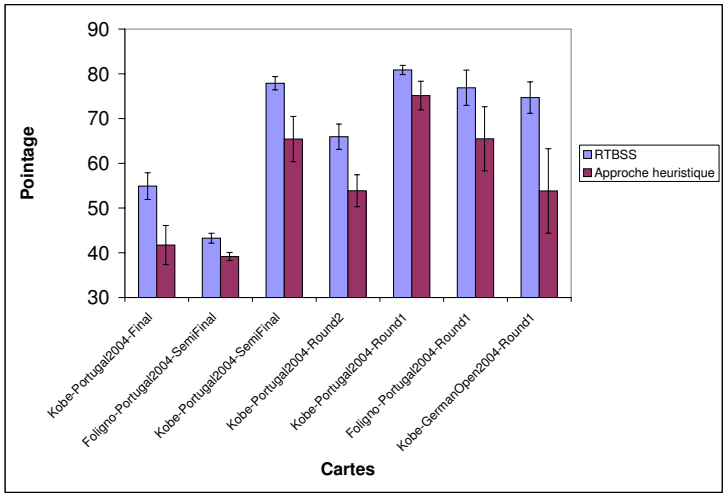

Figure 10. *Pointage moyen sur sept cartes différentes de la RoboCupRescue avec et sans notre approche avec POMDP pour les agents policiers.*

possible de coordonner efficacement 15 agents interagissant dans un environnement très dynamique.

Comme nous pouvons le voir sur la figure 9, les routes autour de l'agent pompier reçoivent des récompenses positives, alors que les routes autour de l'agent policier reçoivent des récompenses négatives. Donc, dans cet exemple, l'agent aura tendance à aller vers les routes proches du feu et non d'aller aider directement l'agent pompier, car il y a déjà un agent policier à proximité. Par conséquent, les agents réussissent à se coordonner d'une manière très flexible en propageant des récompenses positives et négatives. Ceci est une méthode intéressante pour coordonner des agents dans un POMDP multiagent résolu en-ligne.

7.4. *Résultats et discussion*

Pour démontrer l'efficacité de notre approche, nous l'avons comparée avec notre ancienne approche pour les policiers dans la RoboCupRescue. Notre ancienne approche était une approche intuitive dans laquelle, les agents déblayaient les routes selon certaines priorités. Chaque agent se voyait attribuer un secteur au début de la simulation et il était responsable du déblaiement de ce secteur. Dans ce cadre, les agents commençaient par déblayer les routes demandées par les autres agents, par la suite, les routes autour des refuges et des feux et finalement, toutes les routes de leur secteur.

Les résultats que nous avons obtenus sur sept cartes différentes sont présentés sur la Figure 10. Ce graphique présente les différences entre les performances moyennes

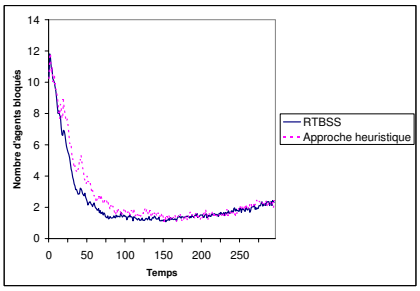

Figure 11. *Comparaison du nombre d'agents bloqués à chaque cycle dans la simulation RoboCupRescue.*

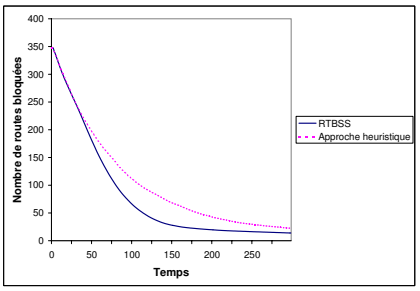

Figure 12. *Comparaison du nombre de routes bloquées à chaque cycle dans la simulation.*

des cartes utilisées lors du German Open 2004 et sur certaines cartes de la compétition mondiale RoboCupRescue 2004 qui avait lieu au Portugal. L'amélioration en moyenne sur ces sept cartes est de 11 points. Cette différence est très importante, surtout en considérant que la seule différence entre les deux programmes étaient les policiers ; les pompiers et les ambulanciers étaient les mêmes. Pour montrer à quel point une différence de 11 points est très importante, mentionnons que nous avons concédé la première place lors de la compétition mondiale de 2004 par une marge de seulement 0,4 point. Sur le graphique, nous avons illustré un intervalle de confiance de 95%. On constate donc en plus d'améliorer considérablement le pointage, l'approche que nous avons présentée offre des résultats beaucoup plus stables.

La figure 11 présente une comparaison entre le nombre d'agents qui sont bloqués à chaque tour. Comme nous l'avons mentionné plus haut, un des rôles des agents policiers est de venir en aide aux autres agents qui se retrouvent face à une route bloquée. Moins il y a d'agents qui sont bloqués, plus les performances seront bonnes. Le graphique montre qu'avec notre approche, il y a en moyenne un ou deux agents de moins de bloqués au début de la simulation, ce qui signifie que ces agents peuvent secourir des civils au lieu d'attendre d'être libérés.

La figure 12 montre le nombre de routes qui sont bloquées en moyenne dans une simulation selon le temps. Le but des agents policiers est donc de débloquer ces routes aussi vite que possible. On constate donc qu'encore une fois, l'approche qui utilise RTBSS permet une grande amélioration.

Notre algorithme RTBSS permet donc à nos agents policiers de déblayer les routes les plus importantes pour les autres agents. De cette manière, les pompiers et les ambulanciers peuvent être plus efficaces dans leurs tâches, car ils peuvent se déplacer plus librement. L'application de notre méthode a d'ailleurs permis à nos policiers d'être parmi les meilleurs au monde lors de la compétition internationale de la RoboCupRescue 2004, qui s'est tenue à Lisbonne au Portugal. Lors de cette compétition, la performance de nos policiers nous a aidés à nous classer deuxième au monde.

8. Travaux connexes

Nous avons comparé notre approche avec cinq autres approches utilisées pour trouver des solutions approximatives [PIN 03, POU 03, SPA 04, SMI 04, BRA 04]. Ces approches sont très intéressantes car elles permettent de résoudre des POMDPs de plus grande taille que les algorithmes optimaux, tout en conservant, pour certains, une estimation de la distance de leur solution avec la solution optimale. Toutefois, ces méthodes sont encore limitées à des problèmes relativement petits, car elles sont exécutées hors-lignes, ce qui requière beaucoup de temps de calcul sur les grands environnements.

Certains chercheurs ont travaillé sur des approches en-lignes pour la résolution de POMDPs. Par exemple, [MCA 99] utilisent aussi une exploration en-ligne de l'espace des états de croyance, mais ils n'utilisent pas de techniques d'élagage pour accélérer l'algorithme. Par ailleurs, [WAS 97] a développé l'algorithme BI-POMDP qui génère un arbre similaire au notre, mais à l'aide de l'algorithme AO* auquel il a ajouté des calculs de bornes pour choisir l'ordre dans lequel les noeuds sont développés. Pour ses bornes, le MDP sous-jacent doit être résolu hors-ligne, ce qui n'est pas acceptable pour nous, voir section 2. [KEA 00] utilisent une exploration de trajectoires dans une structure d'arbre pour tester différentes politiques. Leur objectif est de choisir une politique parmi une classe de politiques en utilisant un simulateur du POMDP. Dans notre cas, nous ne considérons pas la disponibilité d'un simulateur pour tester différentes politiques. Nous considérons plutôt que l'agent doit travailler directement dans l'environnement.

Un autre algorithme en-ligne est BEL-RTDP qui apprend une estimation des valeurs des différents états de croyance visités en effectuant plusieurs essaies dans l'environnement [GEF 98]. Les principales différences de cette approche est qu'elle n'effectue pas de recherche dans l'arbre des états de croyance et qu'elle a besoin de temps hors-ligne pour calculer les valeurs de base pour les états de croyance en se basant sur l'approche Q_{MDP}. De plus, comme il apprend des estimations pour tous les états de croyances visités, il doit discrétiser l'espace des états de croyance pour obtenir un nombre fini d'états de croyance. On peut également citer [FOR 95] qui ont modélisé le problème de conduire une voiture comme un POMDP et qui ont proposé une politique conçue à la main à l'aide de différents critères pour retourner la meilleure action à faire. Notre approche par contre va beaucoup plus loin car elle propose un modèle formel plus général qui peut être utilisé pour la prise de décision dans les POMDPs.

D'autres méthodes effectuent des recherches heuristiques pour la résolution de POMDPs, mais dans leur cas, la recherche est effectuée dans l'espace des politiques qui sont représentées à l'aide de machines à états finis [HAN 98, MEU 99]. Par conséquent, encore une fois, ils essaient de construire une politique complète hors-ligne, ce qui diffère donc de notre méthode.

Il y a certains travaux similaires au niveau des MDPs, mais ceux-ci ne gèrent pas les observations, car ils sont dans le cas complètement observable. Notons les travaux de [PéR 04] qui utilisent une méthode de recherche en-ligne pour les MDPs.

Notons aussi LAO*, un algorithme hors-ligne qui utilise une recherche pour résoudre des MDPs et qui retourne une politique complète représentée comme un graphe cyclique [HAN 01].

On peut également faire des rapprochements entre notre approche en-ligne pour la prise de décision dans les grands POMDPs avec les recherches effectuées dans le domaine de la planification probabiliste [BLY 99, BOU 99, BON 00]. La planification probabiliste consiste à trouver un plan, c'est-à-dire une séquence d'actions, que l'agent pourra effectuer afin d'atteindre un certain but. Pour ce faire, une recherche en profondeur est souvent utilisée afin de parcourir l'espace des distributions de probabilités sur les états. Il y a deux différences majeures entre la planification probabiliste et les POMDPs. Tout d'abord, en planification l'agent cherche à atteindre un état but alors que ce n'est pas nécessairement le cas dans un POMDPs. Deuxièmement, la planification ne tient pas compte de l'exécution. C'est-à-dire qu'en planification traditionnelle, on suppose que l'agent n'aura aucune interaction avec son environnement. Ainsi, il suffit de trouver un plan, c'est-à-dire une séquence d'actions. Du côté des POMDPs par contre, le modèle tient compte du fait qu'à chaque cycle, l'agent a des observations.

9. Conclusion et travaux futurs

Dans cet article, nous avons présenté une approche pour la prise de décision en-ligne pour des POMDPs avec un grand nombre d'états et avons démontré que cela peut mener à des résultats très satisfaisants avec un temps de calcul très bas. Notre approche a comme principal avantage de pouvoir être appliquée sur des problèmes de tailles très imposantes.

Nous avons testé notre approche sur trois problèmes : *Tag*, *RockSample* et RoboCupRescue. Les deux premiers environnements nous ont permis de nous comparer à plusieurs algorithmes actuels tandis que la RoboCupRescue permet d'illustrer les façons d'étendre notre approche à des environnements très complexes. Dans les résultats sur l'environnement *RockSample*, nous avons pu constater que lorsque l'environnement devient assez grand, les approches hors-lignes ont de la difficulté à converger et lorsque cela se produit, notre approche en-ligne devient plus efficace. Partant de cette constatation, nous pouvons donc affirmer que notre approche est beaucoup mieux adaptée aux grands environnements.

Par ailleurs, notre approche est beaucoup plus flexible concernant la structure de l'environnement. La même implémentation de notre algorithme demeure applicable immédiatement même si la configuration physique de l'environnement change. Pour la RoboCupRescue, ceci est vital dans la mesure où les agents se doivent d'être efficaces dans n'importe quelle ville simulée lors de la compétition et les agents ne disposent pas de temps pour faire un apprentissage hors-ligne de la politique. De plus, nous avons présenté une fonction de récompense dynamique permettant de coordonner plusieurs agents dans des environnements dynamiques.

Dans nos travaux futurs, nous aimerions améliorer notre algorithme en-ligne en réutilisant l'information calculée précédemment dans la simulation. Ceci permettrait d'accélérer les calculs permettant ainsi d'explorer plus en profondeur tout en respectant les contraintes de temps-réel.

10. Bibliographie

[ABE 03] ABERDEEN D., « A (Revised) Survey of Approximate Methods for Solving Partially Observable Markov Decision Processes », rapport, 2003, National ICT Australia.

[AST 65] ASTROM K. J., « Optimal Control of Markov Decision Processes with Incomplete State Estimation », *Journal of Mathematical Analysis and Applications*, vol. 10, 1965, p. 174-205.

[BLY 99] BLYTHE J., « Decision-theoretic planning », *AI Magazine*, , summer 1999.

[BON 00] BONET B., GEFFNER H., « Planning with Incomplete Information as Heuristic Search in Belief Space », *Proc. 5th International Conf. on Artificial Intelligence Planning and Scheduling*, Breckenridge, Colorado, 2000, AAAI Press, p. 52–61.

[BOU 96] BOUTILIER C., POOLE D., « Computing Optimal Policies for Partially Observable Decision Processes Using Compact Representations », *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, Portland, Oregon, USA, 1996, AAAI Press / The MIT Press, p. 1168–1175.

[BOU 99] BOUTILIER C., DEAN T., HANKS S., « Decision-Theoretic Planning : Structural Assumptions and Computational Leverage », *Journal of Artificial Intelligence Research*, vol. 11, 1999, p. 1-94.

[BOY 98] BOYEN X., KOLLER D., « Tractable Inference for Complex Stochastic Processes », *In Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, 1998, p. 33-42.

[BRA 04] BRAZIUNAS D., BOUTILIER C., « Stochastic Local Search for POMDP Controllers », *The Nineteenth National Conference on Artificial Intelligence (AAAI-04)*, 2004.

[DUT 03] DUTECH A., SAMUELIDES M., « Apprentissage par renforcement pour les processus décisionnels de Markov partiellement observés », *Revue d'Intelligence Artificielle*, vol. 17(4), 2003.

[FOR 95] FORBES J., HUANG T., KANAZAWA K., RUSSELL S. J., « The BATmobile : Towards a Bayesian Automated Taxi », *IJCAI*, 1995, p. 1878-1885.

[GEF 98] GEFFNER H., BONET B., « Solving large POMDPs using real time dynamic programming », 1998.

[HAN 98] HANSEN E. A., « Solving POMDPs by Searching in Policy Space », *Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, Madison, Wisconsin, 1998, p. 211–219.

[HAN 00] HANSEN E. A., FENG Z., « Dynamic Programming for POMDPs Using a Factored State Representation », *Artificial Intelligence Planning Systems*, 2000, p. 130-139.

[HAN 01] HANSEN E. A., ZILBERSTEIN S., « LAO : a Heuristic Search Algorithm that Finds Solutions with Loops », *Artificial Intelligence*, vol. 129, n° 1-2, 2001, p. 35-62, Elsevier Science Publishers Ltd.

[HAU 00] HAUSKRECHT M., « Value-Function Approximations for Partially Observable Markov Decision Processes », *Journal of Artificial Intelligence Research*, vol. 13, 2000, p. 33-94.

[KAE 96] KAELBLING L. P., LITTMAN M. L., CASSANDRA A. R., « Planning and Acting in Partially Observable Stochastic Domains », rapport n° CS-96-08, 1996, Brown University.

[KEA 00] KEARNS M., MANSOUR Y., NG A. Y., « Approximate Planning in Large POMDPs via Reusable Trajectories », SOLLA S., LEEN T., MULLER K.-R., Eds., *Advances in Neural Information Processing Systems 12*, MIT Press, 2000.

[KIT 00] KITANO H., « RoboCup Rescue : A Grand Challenge for Multi-Agent Systems », *Proceedings of ICMAS 2000*, Boston, MA, 2000.

[LIT 94] LITTMAN M. L., « The Witness Algorithm : Solving Partially Observable Markov Decision Processes », rapport n° CS-94-40, 1994, Brown University.

[LIT 96] LITTMAN M. L., « Algorithms for Sequential Decision Making », PhD thesis, Brown University, 1996.

[MCA 99] MCALLESTER D., SINGH S., « Approximate Planning for Factored POMDPs using Belief State Simplification », *In Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, Stockholm, 1999, p. 409-416.

[MEU 99] MEULEAU N., KIM K.-E., KAELBLING L. P., CASSANDRA A. R., « Solving POMDPs by Searching the Space of Finite Policies », *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI-99)*, San Francisco, 1999, Morgan Kaufmann, p. 417–426.

[NAI 02] NAIR R., TAMBE M., MARSELLA S., « Team Formation for Reformation in Multiagent Domains like RoboCupRescue », KAMINKA G., LIMA P.AND ROJA R., Eds., *Proceedings of RoboCup-2002 International Symposium*, 2002.

[PAP 87] PAPADIMITRIOU C., TSISIKLIS J. N., « The complexity of Markov decision processes », *Mathematics of Operations Research*, vol. 12, n° 3, 1987, p. 441–450, INFORMS.

[PIN 03] PINEAU J., GORDON G., THRUN S., « Point-based value iteration : An anytime algorithm for POMDPs », *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-03)*, Acapulco, Mexico, 2003, p. 1025-1032.

[POU 01] POUPART P., BOUTILIER C., « Vector-Space Analysis of Belief-State Approximation for POMDPs », *In Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI)*, Seattle, 2001, p. 445-452.

[POU 03] POUPART P., BOUTILIER C., « Bounded finite state controllers », *Advances in Neural Information Processing Systems 16 (NIPS-2003).*, Vancouver, Canada, 2003.

[POU 05] POUPART P., « Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes », PhD thesis, University of Toronto, 2005, à paraître.

[PéR 04] PÉRET L., GARCIA F., « On-line search for solving large Markov decision processes », *In Proceedings of the 16th European Conference on Artificial Intelligence (ECAI2004)*, Valence, Spain, 2004.

[SMI 04] SMITH T., SIMMONS R., « Heuristic Search Value Iteration for POMDPs », *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence(UAI-04)*, Banff, Canada, 2004.

[SON 71] SONDIK E. J., « The Optimal Control of Partially Observable Markov Processes », PhD thesis, Stanford University, 1971.

[SPA 04] SPAAN M. T. J., VLASSIS N., « A Point-Based POMDP Algorithm for Robot Planning », *In Proceedings of the IEEE International Conference on Robotics and Automation*, New Orleans, Louisiana, 2004, p. 2399-2404.

[VLA 04] VLASSIS N., SPAAN M. T. J., « A fast point-based algorithm for POMDPs », *Benelearn 2004 : Proceedings of the Annual Machine Learning Conference of Belgium and the Netherlands*, Brussels, Belgium, janvier 2004, p. 170–176, (Also presented at the NIPS-16 workshop 'Planning for the Real-World', Whistler, Canada, Dec 2003).

[WAS 97] WASHINGTON R., « BI-POMDP : Bounded, Incremental Partially-Observable Markov-Model Planning », *Proceedings of the 4th European Conference on Planning*, vol. 1348 de *Lecture Notes in Computer Science*, Toulouse, France, 1997, Springer, p. 440-451.