Apprentissage par renforcement hiérarchique dans les MDP factorisés *

Olga Kozlova[1,2], Olivier Sigaud[1] et Christophe Meyer[3]

[1] Institut des Systèmes Intelligents et de Robotique
Université Pierre et Marie Curie - Paris 6, CNRS FRE 2507
4 place Jussieu, F75252 Paris Cedex 05
Olivier.Sigaud@lip6.fr et http://www.isir.fr/

[2] Thales Security Solutions & Services, Simulation
1 rue du Général de Gaulle, Osny BP 226
F95523 Cergy Pontoise Cedex
Olga.Kozlova@thalesgroup.com

[3] Thales Security Solutions & Services, ThereSIS Research and Innovation Office
Route départementale 128
F91767 Palaiseau Cedex
Christophe.Meyer@thalesgroup.com

Résumé : Nous proposons dans cet article une méthode qui permet d'apprendre par renforcement à résoudre un problème markovien de grande taille dont on ne connaît pas a priori la structure en combinant les capacités d'abstraction hiérarchique des Processus Décisionnels Semi-Markoviens et les capacités de factorisation des Processus Décisionnels de Markov factorisés. Nous validons notre approche sur le problème classique du taxi.
Mots-clés : Apprentissage par renforcement, modèles factorisés, options.

1 Introduction

Les Processus Décisionnels de Markov (MDP) constituent un cadre fondamental pour modéliser des problèmes de planification et d'apprentissage par renforcement dans l'incertain. Cependant, les algorithmes exacts conçus dans ce cadre sont inaptes à traiter des problèmes de grande taille à cause de la « malédiction de la dimensionalité » (Sutton & Barto, 1998). Les MDP factorisés (FMDP) (Boutilier *et al.*, 1995) permettent de représenter des problèmes structurés de grande taille de façon compacte. Dans cette approche, un état est décrit implicitement par l'assignation de valeurs à un ensemble de variables aléatoires. Les réseaux bayésiens dynamiques (DBN) (Dean & Kanazawa, 1989) permettent alors d'exploiter les dépendances entre ces variables afin de ne pas énumérer explicitement l'ensemble des états ou des couples (états, actions).

Une autre façon de réduire la taille des problèmes consiste à utiliser les Semi-MDP (SMDP) (Barto & Mahadevan, 2003) dans lesquels le nombre de pas de temps entre une décision et la suivante est une variable aléatoire. Le cadre des SMDP est la base des algorithmes d'apprentissage par renforcement hiérarchique car il permet de décomposer le MDP global en sous-problèmes qui sont plus faciles à résoudre individuellement (Barto & Mahadevan, 2003).

*Ce travail bénéficie du contrat CIFRE - 1032/2006

Dans le cadre des FMDP comme dans celui des SMDP, on fait généralement l'hypothèse que la structure du problème est connue a priori, ce qui est rarement le cas en pratique. Du côté de la structure hiérarchique, (Dietterich, 2000) considère ainsi que le principal problème de l'apprentissage par renforcement consiste à trouver une méthode générale pour découvrir automatiquement la structure hiérarchique d'un MDP. HEXQ (Hengst, 2002) et VISA (Jonsson & Barto, 2006) sont deux algorithmes destinés à résoudre ce problème. Pour les FMDP, lorsque la structure temporelle des fonctions de transition et de récompense n'est pas connue a priori, Degris et al. ont proposé SDYNA pour résoudre des problèmes d'apprentissage par renforcement de grande taille (Degris *et al.*, 2006b).

Dans cette contribution, après avoir présenté les FMDP, les SMDP et la découverte automatique d'options dans la section 2, nous décrivons dans la section 3 notre approche permettant de combiner apprentissage de la structure hiérarchique et apprentissage de la structure temporelle d'un MDP. Dans la section 4, nous comparons notre approche dans trois cas : si l'on apprend le problème sans tenir compte de sa structure hiérarchique, si la structure hiérarchique est donnée a priori et si la structure hiérarchique est apprise. Cette comparaison est réalisée sur le problème du taxi (Dietterich, 1988). Nous discutons à la section 5 de la nécessité d'introduire la gestion de plusieurs niveaux de hiérarchie pour traiter des problèmes de très grande taille et nous indiquons nos perspectives d'application à des problèmes réels de simulation d'entraînement au combat.

2 Cadre théorique

2.1 MDP et FMDP

Un MDP est un tuple $\langle S, A, R, P\rangle$, où S et A sont des ensembles finis d'états et d'actions d'un agent ; $R : S \times A \rightarrow \mathbb{R}$ est la fonction de récompense immédiate $R(s,a)$ et $P : S \times A \times S \rightarrow [0,1]$ est la fonction décrivant les probabilités de transition $P(s\prime|s,a)$ du MDP. Réaliser l'action $a \in A$ dans l'état $s \in S$ produit la transition du système dans l'état $s\prime \in S$ avec une probabilité $P(s\prime|s,a)$ et donne lieu à la récompense $R(s,a)$. Une politique stationnaire $\pi : S \times A \rightarrow [0,1]$ définit la probabilité $\pi(s,a)$ que l'agent fasse l'action a dans l'état s. Le but considéré ici est de trouver une politique π qui maximise la fonction de valeur $V_\pi(s)$ utilisant le critère de récompense actualisée : $V_\pi(s) = E_\pi[\sum_{t=0}^{\infty} \gamma^t, r_t|s_0 = s]$ où $\gamma \in (0,1]$ est le facteur d'actualisation, r_t la récompense obtenue à l'instant t et s_0 l'état initial, en considérant un horizon infini. La fonction de valeur d'action $Q_\pi(s,a)$ est définie par :

$$Q_\pi(s,a) = \sum_{s\prime \in S} P(s\prime|s,a)(R(s,a) + \gamma V_\pi(s\prime)) \quad (1)$$

Une politique π est optimale si, pour tout $s \in S$ et toute politique $\pi\prime$, $V_\pi(s) \geq V_{\pi\prime}(s)$. La fonction de valeur d'une politique optimale est appelée fonction de valeur optimale et notée V^*.

Un MDP factorisé (FMDP) (Boutilier *et al.*, 1995) est décrit par un ensemble de variables d'états $S = \{X_1, \ldots, X_n\}$, où chaque X_i prend ses valeurs dans un domaine fini $Dom(X_i)$. Un état $s \in S$ assigne une valeur $x_i \in Dom(X_i)$ à chaque variable d'état X_i. Le modèle de transition d'état P_a d'une action a est défini par le graphe de transition G_a representé par un réseau bayésien dynamique (DBN pour *Dynamic Bayes Net*) (Dean & Kanazawa, 1989). G_a est un graphe orienté acyclique à deux couches dont les nœuds sont $\{X_1, ..., X_n, X\prime_1, ..., X\prime_n\}$ où X_i est une variable au temps t et $X\prime_i$ la même variable au temps $t+1$. Nous supposons qu'il n'y a pas d'arcs synchrones, c'est-à-dire pas d'arcs entre des variables au même pas de temps. Un graphe G_a est quantifié par une distribution de probabilité conditionnelle représentée par un arbre de probabilité conditionnelle (CPT) associé à chaque nœud $X\prime_i \in G_a$. Le modèle des transitions P du FMDP est alors défini par un DBN spécifique $P_a = \langle G_a, \{cpt^a_{X_1}, ..., cpt^a_{X_n}\}\rangle$ pour chaque action a. Représenter le problème sous la forme d'un FMDP permet une réduction exponentielle de la taille du modèle.

Des méthodes de planification dédiées aux FMDP ont été développées pour traiter les cas où les fonctions de transition et de récompense sont connues a priori. Nous ne mentionnerons ici que les méthodes de program-

mation dynamique, à savoir *Structured Policy Iteration* (SPI), *Structured Value Iteration* (SVI) (Boutilier *et al.*, 2000) et *Stochastic Planning Using Decision Diagrams* (SPUDD) (Hoey *et al.*, 1999), qui étendent la programmation dynamique et exploitent des fonctions de transition exprimées sous la forme d'arbres de décision ou de diagrammes de décision.

2.2 SMDP

Comme nous l'avons mentionné dans l'introduction, une autre façon de réduire la taille d'un problème est d'avoir recours à des représentations hiérarchiques. Cette méthode consiste à introduire différentes formes d'abstraction en ignorant les états ou les variables non pertinents. Pour tirer profit de ces simplifications, les méthodes d'apprentissage par renforcement hiérarchique (HRL) décomposent la tâche originale en tâches plus petites et permettent l'apprentissage et la planification à tous les niveaux d'abstraction temporelle (Barto & Mahadevan, 2003; Dietterich, 2000). En pratique, on généralise les MDP en SMDP en introduisant entre deux transitions d'état des actions de durée étendue qui correspondent à des sous-tâches.

Parmi les différents formalismes proposés pour les SMDP (Barto & Mahadevan, 2003), nous utilisons le cadre des *options* (Sutton *et al.*, 1999). Les options constituent une généralisation des actions primitives pour inclure des actions abstraites à durée étendue. Là encore, la prise en considération d'options permet de réduire exponentiellement la taille des problèmes d'apprentissage à résoudre. Même si le fait d'ajouter les options aux actions primitives augmente la taille de l'espace des choix d'actions possibles, le nombre de choix à réaliser pour atteindre l'objectif diminue, ce qui simplifie le SMDP à résoudre. Par ailleurs, des options peuvent être réutilisées d'une tâche à une autre.

Une option est un tuple $\langle \mathcal{I}, \pi, \beta \rangle$, où $\mathcal{I} \subseteq S$ est un *ensemble d'initialisation*, c'est-à-dire le sous-ensemble des états à partir desquels il est possible d'atteindre la sortie de l'option o, $\pi : S \times A \rightarrow [0,1]$ est la politique suivie lorsqu'on exécute l'option o et $\beta : S \rightarrow [0,1]$ est une fonction de probabilité d'arrêt qui représente la probabilité de stopper o dans chaque état. Une action primitive $a \in A$ du MDP est aussi une option, appelée option à un pas, avec $\mathcal{I} = \emptyset$ et $\beta(s) = 1$. Si l'option est exécutée, ses sous-options sont sélectionnées conformément à sa politique interne π jusqu'à ce qu'elle se termine dans l'état $s\prime$ avec une probabilité $\beta(s\prime)$. Quand l'option se termine, l'agent peut choisir une autre option. Le modèle SMDP est donc représenté par une hiérarchie d'options dans laquelle les options d'un niveau font appel à des options d'un niveau inférieur.

Dans cette perspective, une option o peut être vue comme une sous-tâche du SMDP $\mathcal{M}_o = \langle S_o, O_o, \Psi, R_o, P_o \rangle$ où $S_o \in S$ est l'ensemble des états de l'option, O_o est l'ensemble des sous-options de o, Ψ est l'ensemble des couples (état,option) admissibles, c'est-à-dire l'ensemble des couples dans lesquels on peut atteindre la sortie de l'option à partir de l'état considéré, déterminé par les ensembles d'initialisation des options de O_o, R_o est la fonction de récompense de l'option et P_o est la fonction de probabilité de transition.

2.3 SDYNA

Structured DYNA (SDYNA) (Degris *et al.*, 2006b) est un algorithme général d'apprentissage par renforcement indirect dans le cadre des FMDP. De même que les architectures DYNA (Sutton, 1991), SDYNA combine la planification, l'action et l'apprentissage pour résoudre par essais et erreurs des problèmes d'apprentissage par renforcement stochastiques dont les fonctions de transition et de récompense ne sont pas connues a priori. En particulier, SDYNA n'a pas besoin de construire explicitement la structure globale du DBN et apprend directement la structure locale d'un problème en utilisant un algorithme d'induction incrémentale d'arbres de décision nommé ITI (*Incremental Tree Inducer* (Utgoff, 1989)). L'algorithme construit une représentation structurée des FMDP sous la forme $\mathcal{F} = \{\forall X_i \in X : Tree[P(X_i\prime|s,a)], \forall R_i \in R : Tree[R_i]\}$ à partir d'un flux d'exemples $\langle \mathcal{A}, \sigma \rangle$ où $\mathcal{A} = (x_i, ..., x_n, a)$ est un ensemble d'attributs et σ la classe de l'exemple. Ces représentations sont exploitées simultanément avec leur construction par des

versions incrémentales de techniques de planification fondées sur les FMDP. Par exemple, SPITI, une instanciation de SDYNA, utilise une version incrémentale modifiée de SVI. Nous renvoyons à (Degris *et al.*, 2006a,b) pour une description détaillée des diverses instanciations de SDYNA.

2.4 HEXQ et VISA

Un algorithme de référence pour résoudre les SMDP est MAXQ Dietterich (2000), qui exploite une hiérarchie de tâches donnée a priori avec un algorithme d'apprentissage par renforcement direct. HEXQ (Hengst, 2002) est un algorithme d'apprentissage par renforcement hiérarchique plus récent qui décompose et résout automatiquement un FMDP sans connaître son modèle. HEXQ découvre les abstractions d'état et les options en identifiant des sous-structures qui se répètent dans l'environnement.

Pour réaliser la décomposition hiérarchique du FMDP, HEXQ détermine un ordre des variables d'état sur la base de la fréquence avec laquelle elles changent de valeur : plus elles changent, plus elles sont en fin de classement. Pour chaque variable d'état, HEXQ identifie des couples de *sortie* $\langle s_e, a \rangle$ signifiant que réaliser l'action a dans l'état s_e produit une transition imprévisible, où l'action a fait : (i) changer la valeur de la variable d'état suivante dans l'ordre déterminé ou (ii) se terminer l'option. Ces sorties constituent les frontières entre les régions qui représentent les sous-tâches du MDP global. HEXQ définit un MDP dans chaque région en associant à la sortie une transition vers un état absorbant. La solution de ce sous-MDP est une politique sur la région qui amène l'agent à la sortie à partir de n'importe quel état initial. Dans HEXQ, les politiques des sous-MDP sont apprises en ligne et deviennent les actions abstraites du niveau supérieur. Comme les actions abstraites peuvent nécessiter des durées variables, la tâche globale est alors un SMDP qui fait appel à moins de variables que le MDP original et qui n'utilise que des actions abstraites.

Variable Influence Structure Analysis (VISA) (Jonsson & Barto, 2006), décompose dynamiquement un FMDP en niveaux hiérachiques. VISA détermine des relations causales entre les variables d'états en construisant un graphe d'influence sur la base d'un modèle fourni a priori du FMDP. Pour définir des options, VISA utilise un formalisme proche de celui de HEXQ mais, au lieu de classer les variables en fonction de leur fréquence, il se fonde sur les relations du graphe d'influence. VISA utilise des algorithmes d'apprentissage par renforcement pour trouver la politique associée aux options, sans avoir recours à un modèle des probabilités de transition.

3 Méthode

Dans cette contribution, nous présentons une approche qui consiste à découper un MDP global en un ensemble d'options découvertes automatiquement et à utiliser simultanément SVI sur chaque option pour apprendre la politique locale associée. Ces options font à leur tour appel à des options plus simples qui sont les actions primitives du MDP. L'approche permet d'accélérer l'apprentissage de la politique globale, ce que nous vérifierons expérimentalement.

Outre l'apprentissage du modèle des transitions qui est réalisé incrémentalement et globalement par induction d'arbres de décision exactement comme dans SPITI, l'algorithme global de planification se décompose en deux parties : la découverte des options et de leur hiérarchie et l'exécution globale de la politique. Nous présentons successivement ces deux parties.

3.1 Découverte des options et de leur hiérarchie

Notre approche de la découverte des options est similaire à celle de HEXQ : nous définissons un ensemble de « sorties » (*exits*) correspondant à des changements de valeur pour des variables liées à la fonction de

récompense, nous associons à chacune de ces sorties un contexte correspondant à l'ensemble des états dans lequel le changement de valeur correspondant se produit et nous associons une option à chaque sortie.

La découverte des changements de valeur pour les variables considérées se fait à partir des arbres représentant le modèle des transitions, comme indiqué dans l'algorithme 1.

3.1.1 Hiérarchie des sorties

Algorithme 1 : *UpdateExitsHierarchy*($\mathcal{F}_t$) $\rightarrow \mathcal{M}_e$

entrée : $\mathcal{F}_t$, le modèle des transition courant
sortie : $\mathcal{M}_e$, la hiérarchie des sorties
pour chaque $\mathcal{F}_t.Tree[P(X_i{}'|s,a)]$ **faire**
  pour chaque *feuille* $l_b \in \mathcal{F}_t.Tree[P(X_i{}'|s,a)]$ **faire**
    si x'_i *dans* $l_b \neq x^b{}_i$ *contenu dans la branche* b *(i.e. si l'action* a *modifie la valeur de* X_i*)* **alors**
      si $a \notin \mathcal{M}_e$.*node*(X_i).*ListChilds* **alors**
        $\mathcal{M}_e$.*node*(X_i).*AddChild*(a);
      si *le nombre de variables* V *dans le contexte* c *de* $l_b > 1$ **alors**
        $v_{ch} \leftarrow \langle x^b{}_i, x'{}_i \rangle$;
        $\mathcal{M}_e$.*node*(a).*AddChild*(v_{ch});
        pour chaque *variable* $v \in c$ **faire**
          si $v \neq X_i$ **alors**
            $\mathcal{M}_e$.*node*(v_{ch}).*AddChild*(v);
        Ajouter dans $\mathcal{M}_e$ l'exit $e \leftarrow \langle X_i, a, v_{ch}, c \rangle$;
      sinon
        Ajouter dans $\mathcal{M}_e$ l'exit $e \leftarrow \langle X_i, a, \emptyset, \emptyset \rangle$;

Contrairement à HEXQ, où les sorties sont des couples (état, action), nous définissons une sortie comme un tuple $\langle v, a, v_{ch}, c \rangle$, où v est la variable dont la valeur est modifiée par cette sortie, $c = \{x_1, ..., x_n\}$ est le contexte, c'est-à-dire l'ensemble des contraintes (i.e. assignations de valeurs à un sous-ensemble de variables d'états) qui conditionne cette sortie, a est l'action de sortie qui modifie la valeur de v et v_{ch} est une modification de la valeur de la variable, i.e. un couple $\langle x, x' \rangle$ où x et $x\prime$ sont les valeurs de la variable avant et après l'exécution de a respectivement[1]. On note que les actions primitives ont un contexte vide.

Cette représentation étendue des sorties rend possible l'organisation hiérarchique globale de l'ensemble des sorties $\mathcal{M}_e$ sous la forme d'un arbre de décision conformément à l'algorithme 1, qui tire parti du fait que le modèle des transitions est lui-même organisé sous la forme d'un ensemble d'arbres. La structuration des sorties en arbres est compacte et facilite la mise à jour incrémentale locale des seules parties de l'arbre qui sont concernées par un changement.

A noter que, lorsqu'on calcule le contexte d'une sortie, on exclut de ce contexte la variable dont la modification définit la sortie, ce qui a pour effet d'éviter de créer des dépendances croisées entre des options.

$\mathcal{M}_e$ est mis à jour à chaque fois que le modèle du FMDP change. Un exemple de hiérarchie des sorties dans le cas du problème du taxi est donné à la figure 2.

3.1.2 Hiérarchie des options

La hiérarchie $\mathcal{M}_o$ des options est calquée sur la hiérarchie des sorties $\mathcal{M}_e$ grâce à l'algorithme 2. La méthode $o.AddSubOptions(\mathcal{M}_e)$ ajoute à une option ses sous-options de la façon suivante : pour chaque

[1] Dans le cas d'un problème stochastique, on a en fait une distribution de probabilités sur des v_{ch} pour une même sortie.

Algorithme 2 : *UpdateOptionsHierarchy*($\mathcal{F}_t, \mathcal{M}_e$) $\rightarrow \mathcal{M}_o$

entrée : $\mathcal{M}_e, \mathcal{F}_t$
sortie : $\mathcal{M}_o$
pour chaque *exit* $e \in \mathcal{M}_e$ **faire**
  Chercher l'option o contenant l'exit e;
  si $o \notin \mathcal{M}_o$ **alors**
    Initialiser une option o avec e;
    Initialiser la politique locale $o.\pi$ avec $\mathcal{F}_t$;
  else
    Mettre à jour o avec e et $\mathcal{F}_t$;
  $o.AddSubOptions(\mathcal{M}_e)$;
  ComputeInitiationSet(o);
Mettre à jour $\mathcal{M}_o$ et $\mathcal{M}_e$ en supprimant les nœuds non pertinents;

variable du contexte de l'option, s'il y a dans $\mathcal{M}_e$ une sortie qui modifie la valeur de cette variable, on ajoute une sous-option correspondant à cette sortie.

La suppression des nœuds non pertinents est rendue nécessaire par le fait que $\mathcal{M}_o$ et $\mathcal{M}_e$ évoluent en fonction de la structure du modèle des transitions qui peut être largement erronée au début de l'apprentissage, ce qui peut conduire à la construction d'options inappropriées. Le processus de destruction des options non pertinentes est fondée sur le fait qu'une option inappropriée cesse d'être mise à jour une fois que le modèle des transitions se stabilise. A la création de chaque option, son nombre de mises à jour est initialisé à la moyenne des nombres de mises à jours de toutes les options présentes dans la hiérarchie. Par la suite, les options dont le nombre de mises à jour s'écarte de plus de 60% par rapport à la moyenne sont considérées comme non pertinentes et supprimées.

Un exemple de hiérarchie des options dans le cas du problème du taxi est donné à la figure 3.

3.1.3 Ensemble d'initialisation et condition d'arrêt

Algorithme 3 : *ComputeInitiationSet*(o) $\rightarrow I_o$

entrée : o
sortie : I_o
$I_o \leftarrow \emptyset$;
pour chaque $v \in exit.c$ **faire**
  $I_o \leftarrow v$
pour chaque $o_i \in$ *SubOptions(o)* **faire**
  si o_i *est une action primitive* **alors**
    $I_o \leftarrow o_i.exit.v$ (avec toutes les valeurs possibles de v);
  sinon
    $I_o \leftarrow o_i.exit.v$;
    pour chaque $v_i \in o_i.exit.c$ **faire**
      $I_o \leftarrow v_i$

L'ensemble d'initialisation I_o d'une option o est construit conformément à l'algorithme 3. Cet algorithme est déclenché à chaque création ou mise à jour d'une option. L'ensemble d'initialisation I_o d'une option o contient les ensembles d'initialisation I_i de chacune de ses sous-options i. Par convention, une sous-option qui ne possède pas d'ensemble d'initialisation est déclenchable partout. C'est le cas notamment pour les options correspondant aux actions primitives. Par conséquent, toutes les valeurs des variables associées aux sorties de telles options sont acceptées. Ainsi, I_o regroupe tous les états à partir desquels l'état de sortie de l'option o est accessible.

Une option se termine en exécutant son action de sortie a dès qu'elle atteint le contexte c de sa sortie e ou dès qu'elle ne peut plus atteindre c. On sait qu'on ne peut plus atteindre la sortie dès que l'état ne figure plus parmi l'ensemble d'initialisation de l'option. La probabilité $\beta(s)$ d'arrêt d'une option o vaut donc 1 pour les états s qui vérifient les contraintes de c et pour les états $s \notin I_o$ dans lesquels le processus ne peut plus rejoindre la sortie. Sinon, $\beta(s) = 0$.

3.2 Calcul de la politique

Algorithme 4 : TeXDYNA

initialisation : FMDP $\mathcal{F}_0$, SMDP $\mathcal{M}_0$;
initialiser $\mathcal{M}_e$ en ajoutant au nœud (**Root**) un nœud-enfant pour chaque variable d'état X_i
pour chaque *pas de temps* t **faire**
  si *il n'y a pas option en cours* **alors** $o \leftarrow$ *GetAdmissibleOption*$(\mathcal{M}_{t-1})$;
  sinon si *o n'est pas une action primitive* **alors**
    si *TerminationFunction(o)* **alors**
      $a \leftarrow o.exit.a$;
    sinon
      $a \leftarrow$ *Acting*$(o.\pi)$;
    Exécuter a ; observer $s\prime$ et r;
    $o.\pi \leftarrow$ *Planning*$(\mathcal{F}_t)$;
  sinon si *o est une action primitive* **alors**
    $a \leftarrow o.exit.a$;
    Exécuter a ; observer $s\prime$ et r;
  $\mathcal{F}_t \leftarrow$ *UpdateFMDP*$(\mathcal{F}_{t-1})$ avec (s,a,$s\prime$,r);
  option en cours $\leftarrow o$;
  $\mathcal{M}_e \leftarrow$ *UpdateExitsHierarchy*$(\mathcal{F}_t)$;
  $\mathcal{M} \leftarrow$ *UpdateOptionsHierarchy*$(\mathcal{F}_t, \mathcal{M}_e)$;

L'algorithme global, que nous nommons TeXDYNA[2], décrit dans l'algorithme 4, réalise simultanément l'apprentissage de la structure du SMDP, celle du FMDP et le calcul d'une politique associée. L'algorithme SVI qui repose sur des modèles des transitions et des récompenses appris par expérience n'est appliqué que localement au calcul des options de niveau intermédiaire dans la hiérarchie, la racine n'étant pas considérée comme une option.

La résolution de chaque sous-MDP, représenté par une option, suit les trois phases de l'algorithme SDYNA : action (méthode *Acting*()), apprentissage (méthode *UpdateFMDP*()) et planification (méthode *Planning*()). La méthode *Acting*() choisit l'action selon la politique interne de l'option $o.\pi$ en appliquant une politique d'exploration ϵ *-greedy*. Pendant la phase d'apprentissage, les représentations structurées des transitions et des récompenses sont mises à jour au niveau du MDP global par la méthode *UpdateFMDP*() qui implémente l'algorithme ITI. La phase de planification, locale à chaque option, utilise une version incrémentale modifiée de SVI qui construit $o.\pi$ sous la forme d'un arbre de décision.

De plus, pour assurer la propagation des récompenses externes aux politiques locales des options, lorsque une option de haut niveau est découverte, une récompense supplémentaire, appelée « récompense interne » r_i (par opposition aux récompenses externes reçues de l'environnement), est attribuée à son action de sortie. En pratique, on fixe $r_i = \frac{r_{max}}{2}$, où r_{max} est la plus grande récompense immédiate externe possible. Cette heuristique s'inspire de l'idée de *« salient event »* proposée dans (Singh *et al.*, 2005).

Enfin, la méthode *GetAdmissibleOption*() retourne les options o dont l'ensemble d'initialisation I_o contient l'état courant s. En pratique, les options de haut niveau forment une partition de l'espace d'état, si bien que *GetAdmissibleOption*() renvoie une seule option et qu'il n'est pas nécessaire d'apprendre quelle option déclencher dans quel état.

[2]pour *Temporally Extended* SDYNA

4 Expérimentations

4.1 Le problème du taxi

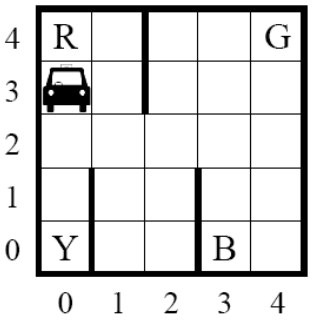

FIG. 1 – Problème du taxi

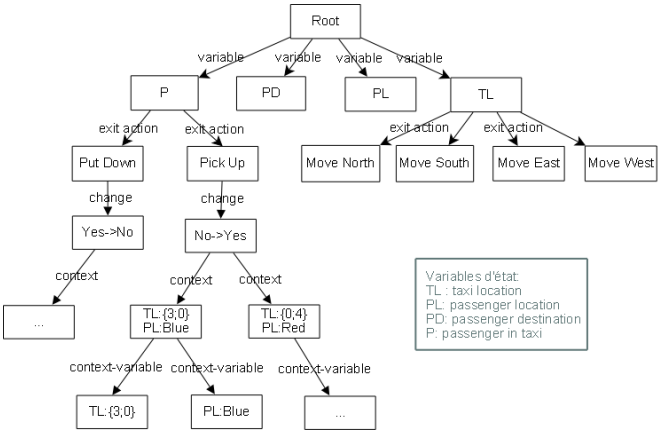

FIG. 2 – Sorties associées au problème du taxi

Le problème du taxi, représenté sur la figure 1, a été proposé par (Dietterich, 1988). La représentation du problème est la suivante : un taxi se déplace sur une grille 5×5 qui comporte quatre zones spécifiques, notées R, G, Y et B, définissant la localisation et/ou la destination d'un passager. Le problème du taxi se décompose en épisodes. A chaque nouvel épisode, le taxi se trouve à une position choisie au hasard, un passager se trouve dans l'une des quatre zones spécifiques et veut être transporté dans une autre zone choisie également au hasard. Un épisode se termine quand le passager est déposé à la destination ou lorsque le nombre de pas de temps prédéfini a été dépassé. A chaque pas de temps, le taxi peut exécuter l'une des 6 actions primitives : aller au nord, au sud, à l'est ou à l'ouest, prendre le passager ou déposer le passager. Dans la version stochastique, par opposition à la version déterministe, il y a une probabilité de 0.2 à chaque pas de temps pour que le taxi exécute une action aléatoire plutôt que celle qu'il a choisie. Le taxi reçoit une récompense égale à 20 s'il dépose le passager à l'endroit correct. Si le taxi exécute l'action de prendre ou déposer le passager dans une zone incorrecte, il reçoit une punition de -10. Une punition de -1 est attribuée pour chaque action primitive. La récompense interne dans le cas d'apprentissage avec options vaut 10. Enfin, il y a 500 états possibles.

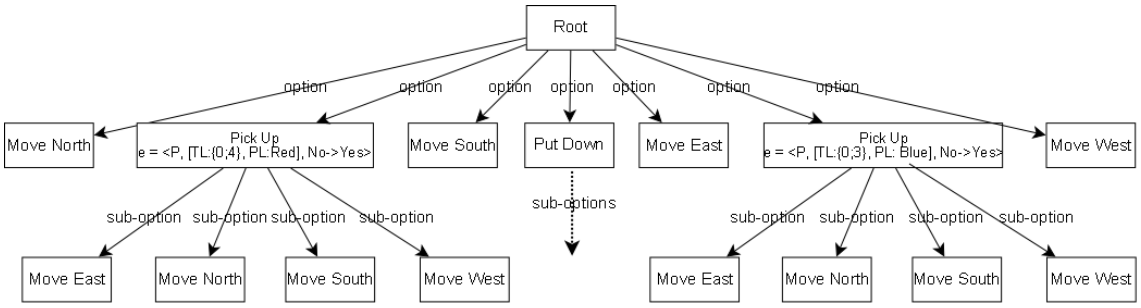

FIG. 3 – Options associées au problème du taxi

4.2 Résultats

Les figures 4 et 5 sont obtenues en faisant une moyenne sur 20 essais sur le problème du taxi dans ses versions déterministe et stochastique respectivement. Pour lisser le résultat, on effectue une moyenne glissante, les valeurs sont pondérées par les dix valeurs voisines. On donne les résultats de SVI sans options (*action*

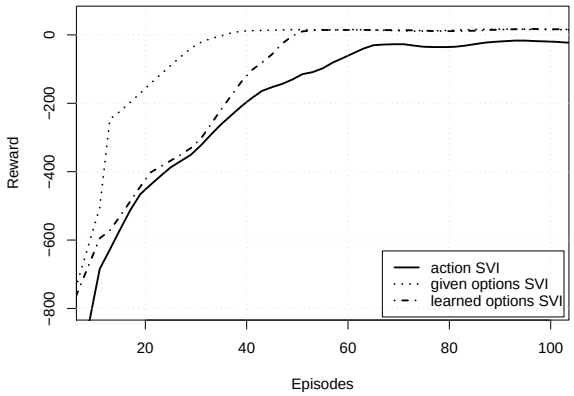

FIG. 4 – Problème de taxi déterministe

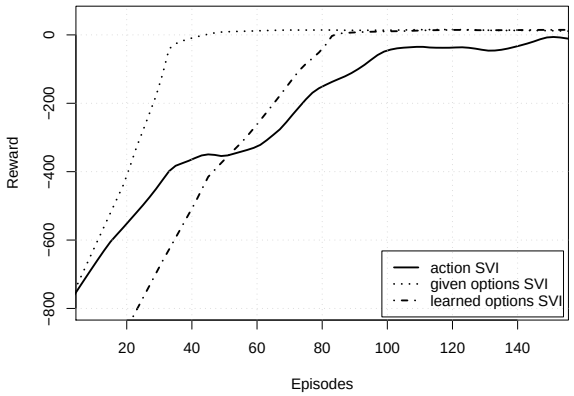

FIG. 5 – Problème de taxi stochastique

SVI), avec des options données a priori (*given options* SVI) et avec les options apprises par notre méthode (*learned options* SVI). Le nombre d'actions pour chaque épisode est limité à 300.

On constate que notre modèle a besoin d'environ 50 épisodes pour converger dans le cas déterministe et 80 épisodes dans le cas stochastique. Comme attendu, cette performance est intermédiaire dans les deux cas entre la performance d'un système auquel on donne les options a priori et SVI qui fonctionne sans option. Par ailleurs, dans le cas stochastique, notre modèle se compare très favorablement à MAXQ, qui a besoin de 115 épisodes environ pour converger et à HEXQ qui a besoin de 160 épisodes environ. On constate en outre que MAXQ et HEXQ convergent tous les deux moins vite que SVI seul, ce qui s'explique pour le premier par le fait qu'il n'utilise pas explicitement la structure factorisée du problème et pour le second par le fait qu'il perd un grand nombre d'épisodes à estimer correctement l'ordre des variables à partir desquelles il construit sa hiérarchie.

5 Discussion et conclusion

Nos travaux de recherche doivent à terme être appliqués à la génération automatique de comportements pour des entités animées (civils, guerilleros, militaires, voire véhicules) d'un logiciel de simulation du théâtre opérationnel.

Sur le problème du taxi, nous avons montré que notre approche qui combine factorisation et abstraction se comporte mieux que des algorithmes de référence de ces domaines. Pour ce qui est de la factorisation, nous bénéficions de la puissance de la méthode employée dans SPITI qui consiste à apprendre incrémentalement le modèle des fonctions de transition et de récompense sous la forme d'arbres de décision, fonctionnant ainsi sans connaissance a priori sur la structure du problème. Pour ce qui est de l'abstraction, en revanche, notre méthode peut être largement améliorée. En effet, en l'état, notre algorithme ne construit qu'une hiérarchie à deux niveaux, le premier niveau étant constitué des actions primitives et le second d'un ensemble d'options mutuellement exclusives. Si cette méthode suffit à traiter efficacement le problème du taxi, elle nous semble insuffisante pour aborder les problèmes de beaucoup plus grande taille qui constituent nos objectifs applicatifs. Face à de tels problèmes, nous aimerions disposer d'une hiérarchie d'options dont le nombre de niveaux ne soit pas contraint a priori et dont l'option racine soit une option comme les autres. On apprendrait ainsi la politique de l'option racine de la même façon que pour les autres options. Par ailleurs, notre méthode de suppression des options inadaptées est provisoire, elle fait appel à deux paramètres qu'il faut régler empiriquement. Il sera nécessaire de proposer une méthode mieux fondée pour traiter ce point.

Au delà de ces améliorations locales de notre algorithme, nos perspectives concernent la prise en compte

du caractère non markovien des simulations complexes que nous allons aborder et la mise au point d'une méthodologie pour introduire la connaissance de l'utilisateur là où c'est nécessaire.

Références

BARTO A. & MAHADEVAN S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Systems Journal*, **13**, 41–77. Special Issue on Reinforcement Learning.

BOUTILIER C., DEARDEN R. & GOLDSZMIDT M. (1995). Exploiting structure in policy construction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, p. 1104–1111.

BOUTILIER C., DEARDEN R. & GOLDSZMIDT M. (2000). Stochastic dynamic programming with factored representations. *Artificial Intelligence*, **121**(1-2), 49–10.

DEAN T. & KANAZAWA K. (1989). A model for reasoning about persistence and causation. *Computational Inteligence*, **5**, 142–150.

DEGRIS T., SIGAUD O. & WUILLEMIN P.-H. (2006a). Chi-square tests driven method for learning the structure of factored MDPs. In *Proceedings of the 22nd Conference Uncertainty in Artificial Intelligence*, p. 122–129, Massachusetts Institute of Technology, Cambridge : AUAI Press.

DEGRIS T., SIGAUD O. & WUILLEMIN P.-H. (2006b). Learning the structure of factored markov decision processes in reinforcement learning problems. In *Proceedings of the 23rd International Conference on Machine Learning*, p. 257–264, Pittsburgh, Pennsylvania : ACM.

DIETTERICH T. (1988). The MAXQ method for hierarchical reinforcement learning. In M. KAUFMANN, Ed., *In 15th International Conference on Machine Learning*.

DIETTERICH T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, **13**, 227–303.

HENGST B. (2002). Discovering hierarchy in reinforcement learning with HEXQ. In *Proceedings of the 19th International Conference on Machine Learning*, p. 243–250.

HOEY J., ST-AUBIN R., HU A. & BOUTILIER C. (1999). SPUDD : Stochastic planning using decision diagrams. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, p. 279–288, Stockholm.

JONSSON A. & BARTO A. (2006). Causal graph based decomposition of factored MDPs. *Journal of Machine Learning Research*, **7**, 2259–2301.

SINGH S., BARTO A. & CHENTANEZ N. (2005). Intrinsically motivated reinforcement learning. *Advances in Neural Information Processing Systems*, **18**, 1281–1288.

SUTTON R., PRECUP D. & SINGH S. (1999). Between MDPs and semi-MDPs : A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, **112**, 181–211.

SUTTON R. S. (1991). DYNA, an integrated architecture for learning, planning and reacting. In *Working Notes of the AAAI Spring Symposium on Integrated Intelligent Architectures*.

SUTTON R. S. & BARTO A. G. (1998). *Reinforcement Learning : An Introduction*. Cambridge, MA : MIT Press.

UTGOFF P. E. (1989). Incremental induction of decision trees. *Machine Learning*, **4**, 161–186.