Apprentissage de la structure des processus de décision markoviens factorisés pour l'apprentissage par renforcement

Thomas Degris, Olivier Sigaud et Pierre-Henri Wuillemin
Université Pierre et Marie Curie - Paris 6, LIP6, 4 Place Jussieu, 75252 Paris Cedex 05, France
prenom.nom@lip6.fr

Résumé

Des algorithmes de planification récents issus de la théorie de la décision sont capables de trouver des politiques optimales ou quasi-optimales sur des problèmes de grande taille en utilisant le formalisme des processus de décision markoviens factorisés (FMDPs). Cependant, ces algorithmes ont besoin d'une connaissance a priori de la structure des problèmes qu'ils résolvent. Dans cette contribution, nous proposons SDYNA, un cadre général pour traiter des problèmes d'apprentissage par renforcement (A/R) de grande taille par essais et erreur et sans connaissance a priori de leur structure. SDYNA intègre des algorithmes incrémentaux de planification tirés des FMDPs avec des techniques d'apprentissage supervisé qui construisent une représentation structurée du problème. Nous décrivons en particulier SPITI, une instance de SDYNA, qui utilise un algorithme d'induction incrémentale d'arbres de décision pour apprendre la structure d'un problème et une version incrémentale de l'algorithme *Structured Value Iteration* pour effectuer la planification. Nous montrons que SPITI construit une représentation factorisée d'un problème d'A/R et améliore la politique plus rapidement qu'un algorithme tabulaire en exploitant la propriété de généralisation des algorithmes d'induction d'arbres de décision.

1 Introduction

Les processus de décision markoviens (MDPs) fournissent le cadre théorique principal pour la planification basée sur la théorie de la décision [1] et l'A/R. Quand les fonctions de transition et de récompense sont connues, les méthodes fondées sur la programmation dynamique (DP) résolvent efficacement les problèmes de petite taille, mais ne peuvent être appliquées telles qu'elles aux problèmes de grande taille car elles reposent sur une énumération explicite de tous les états.

Les processus de décision markoviens factorisés (FMDPs) proposés initialement par Boutilier et al. (1995) permettent de représenter les fonctions de transition et de récompense de façon compacte en utilisant les réseaux bayésiens dynamiques (DBNs).

Les méthodes de résolution classiques telles que la programmation dynamique ont été adaptées à la manipulation de ces représentations (Boutilier et al., 2000) et affinées pour résoudre des problèmes de très grande taille (Hoey et al., 1999; Guestrin et al., 2003). Cependant, ces techniques requièrent une connaissance parfaite des fonctions de transition et de récompense, qui n'est généralement pas disponible en pratique.

Sutton et Barto (1998) décrivent deux approches de la résolution de problèmes d'A/R quand les fonctions de transition et de récompense sont inconnues. Les algorithmes d'A/R *direct* construisent une évaluation de la fonction de valeur optimale à partir de laquelle ils dérivent une politique optimale. Les algorithmes d'A/R *indirect* construisent plutôt un modèle des fonctions de transition et de récompense et utilisent des algorithmes de planification sur ces modèles pour trouver la politique optimale. Par exemple, DYNA-Q (Sutton, 1990) apprend une représentation tabulaire des fonctions de transition et de récompense et met à jour la fonction de valeur pour des couples $s \times a$ choisis aléatoirement dans le modèle. Mais, de même que les méthodes de DP, ces techniques sont limitées par la nécessité d'énumérer tous les couples $s \times a$ du problème.

Certaines méthodes d'A/R direct ont été adaptées à la gestion de représentations factorisées (par exemple McCallum (1995) ou Sallans et Hinton (2004)). Mais, à notre connaissance, il n'existe pas d'adaptation de ce type pour des méthodes d'A/R indirect. Une des difficultés d'une telle approche consiste à apprendre la structure des DBNs qui représentent le problème. L'apprentissage de la structure d'un réseau bayésien quelconque est un problème dur (Chickering, 1996). Nous proposons donc ici une méthode supervisée dédiée pour apprendre spécifiquement la structure des DBNs qui sont utilisés pour représenter un FMDP.

Dans cette contribution, nous décrivons *Structured* DYNA (SDYNA), une classe générale d'algorithme d'A/R indirect fondée sur les FMDPs. Pour résoudre un problème dans lequel les fonctions de transition et de récompense sont inconnues, SDYNA apprend incrémentalement une représentation structurée de ces fonctions et utilise des algorithmes incrémentaux de planification pour construire une représentation structurée de la fonction de valeur optimale. Plus précisément, nous nous focalisons sur SPITI, une instanciation de SDYNA qui utilise ITI (Utgoff et al., 1997), un algorithme d'induction incrémentale d'arbres de

[1] DTP, pour *Decision-Theoretic Planning*

décision (Utgoff, 1986), pour apprendre incrémentalement une représentation structurée du problème d'A/R et qui repose sur une version modifiée de *Structured Value Iteration* (SVI) (Boutilier et al., 2000) pour en déduire une représentation factorisée de la fonction valeur du problème.

2 Fondements

Un MDP est un tuple $\langle S, A, R, P\rangle$ où S et A sont un ensemble fini d'états et d'actions ; $R : S \times A \to \mathbb{R}$ est la fonction de récompense immédiate notée $R(s,a)$ et $P : S \times A \times S \to [0,1]$ est la fonction de transition stochastique markovienne notée $P(s'|s,a)$ du MDP. Une politique stationnaire $\pi : S \times A \to [0,1]$ définit la probabilité $\pi(s,a)$ que l'agent effectue l'action a dans l'état s. Le but est de trouver une politique π qui maximise la fonction de valeur $V_\pi(s)$ définie à partir du critère de récompense actualisée : $V_\pi(s) = E_\pi[\sum_{t=0}^{\infty} \gamma^t \cdot r_t | S_0 = s]$, où $0 \leq \gamma < 1$ 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 :

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

La fonction de valeur optimale V^* et la politique optimale π^* sont définies par $V^* = V_{\pi^*}$ et $\forall \pi,\ \forall s : V^*(s) \geq V_\pi(s)$.

2.1 Value Iteration

Quand R et P sont parfaitement connus, la DP propose une famille de méthodes de résolution parmi lesquelles *Value Iteration*, définie sur la figure 1. Value Iteration calcule une approximation de V^* jusqu'à convergence (étape 2), puis construit une politique optimale π^* en choisissant pour chaque état s l'action a dont la valeur $Q(s,a)$ est la plus élevée (étape 3).

Entrée: $P(s'|s,a)$, $R(s',a)$ Sortie: π^*

1. Soit V' une fonction de valeur arbitraire
2. Répéter :
   (a) $V \leftarrow V'$
   (b) $V'(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s'|s,a)$ $(R(s',a) + \gamma V(s'))$

   jusqu'à $||V' - V|| \leq \frac{\epsilon(1-\gamma)}{2\gamma}$ (avec la norme infinie $||X|| = \max_{x \in X} |x|$).
3. Renvoyer π tel que :
   $\pi(s) = arg\max_a \sum_{s' \in S} P(s'|s,a)(R(s',a) + \gamma V'(s'))$

FIG. 1 – L'algorithme Value Iteration

2.2 Apprentissage par renforcement indirect

Dans les problèmes où les fonctions de transition P et de récompense R sont inconnues, les méthodes d'A/R indirect proposent d'apprendre ces fonctions par essais et erreurs et de calculer une politique en appliquant les méthodes de planification décrites précédemment. L'architecture DYNA (Sutton, 1990), conçue pour intégrer la planification, l'action et l'apprentissage, est l'archétype de cette approche. L'algorithme DYNA-Q, décrit sur la figure 2, est une instanciation de DYNA qui utilise Q-learning pour approximer V^*.

Entrée: $\emptyset$ Sortie: π

1. Initialise $Q(s,a)$ et P
2. À chaque étape :
   (a) $s \leftarrow$ état courant (non terminal)
   (b) $a \leftarrow \epsilon\text{-}greedy(s,Q)$
   (c) Exécute a ; observe s' et r
   (d) $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$
   (e) Ajoute $\langle s,a,s',r\rangle$ à P
   (f) Répète N fois :
      i. $s \leftarrow$ état aléatoire déjà observé
      ii. $a \leftarrow$ action aléatoire déjà testée en s
      iii. $s', r \leftarrow P(s,a)$ (en supposant l'environnement déterministe)
      iv. $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$

FIG. 2 – L'algorithme DYNA-Q

La phase d'action de DYNA-Q est composée de trois étapes (2.a à 2.c) dans lesquelles l'agent suit une politique d'exploration ϵ-$greedy$ (Sutton & Barto, 1998) compte tenu de sa connaissance courante du problème. L'étape 2.e représente l'apprentissage supervisé des fonctions R et P. L'étape 2.f représente la phase de planification, dans laquelle les modèles de R et P sont exploités pour mettre à jour les valeurs $Q(s,a)$ pour un couple $s \times a$ choisi aléatoirement. Il faut noter que DYNA-Q avec $N = 0$ est une approche d'A/R direct similaire à Q-learning. Enfin, DYNA-Q utilise une représentation tabulaire de Q, P et R, ce qui le rend inopérant sur des problèmes de grande taille, même si la phase de planification est désactivée avec $N = 0$.

2.3 Processus de décision markoviens factorisés

L'utilisation de processus de décision markoviens factorisés pour exploiter la structure d'un problème a été proposé initialement par Boutilier et al. (1995). Dans un FMDP, les états sont décomposés sous la forme d'un ensemble de variables aléatoires $S = \{X_1, ..., X_n\}$, où chaque X_i prend ses valeurs dans un domaine fini $Dom(X_i)$. Un état est défini par un vecteur de valeurs $s = (x_1, ..., x_n)$ avec $\forall i,\ x_i \in Dom(X_i)$. On note X_i une variable à l'instant t et X'_i la même variable à l'instant $t+1$. Le modèle des transitions T_a pour une action a est constitué d'un graphe de transitions réprésenté par un DBN (Dean & Kanazawa, 1989), qui est un graphe orienté acyclique à deux couches G_a dont les nœuds sont $\{X_1, ..., X_n, X'_1, ..., X'_n\}$ (voir la figure 6). Dans G_a, les parents de X'_i sont notés $Parents_a(X'_i)$. Dans cette contribution, nous supposons que $Parents_a(X'_i) \subseteq X$

(c'est-à-dire qu'il n'y a pas d'arcs synchrones, ou encore d'arcs de X'_i à X'_j). Une distribution de probabilités conditionnelles $CPD^a_{X_i}(X'_i|Parents_a(X'_i))$ est associée à chaque nœud $X'_i \in G_a$. Le modèle des transitions complet du MDP est donc défini par un DBN distinct $T_a = \langle G_a, \{CPD^a_{X_1}, ..., CPD^a_{X_n}\}\rangle$ pour chaque action a.

3 L'architecture SDYNA

Par analogie avec DYNA, nous proposons l'architecture *Structured* DYNA (SDYNA) pour intégrer la planification, l'action et l'apprentissage en utilisant des représentations factorisées plutôt que tabulaires. Cette architecture est conçue pour résoudre des problèmes d'A/R de grande taille dont la structure est inconnue a priori, en utilisant des versions incrémentales de techniques de planification fondées sur les FMDPs.

L'algorithme 3 décrit SDYNA. $Fact(F)$ désigne une représentation factorisée de la fonction F. Différents types de représentations factorisées sont possibles pour exploiter des régularités de F, telles que des règles, des arbres de décision, des diagrammes de décision binaires ou algébriques, etc.

Entrée: $Acting, LearnR, LearnT, Plan, Fact$
Sortie: $Fact(\pi)$

1. Initialisation
2. À chaque pas de temps t, faire :
   (a) $s \leftarrow$ état courant (non terminal)
   (b) $a \leftarrow Acting(s, \{\forall a \in A : Fact(Q^{\pi}_{t-1}{}'^{*}(s,a))\})$
   (c) Exécute a ; observe s' et r
   (d) $Fact(T_t) \leftarrow LearnT(\langle s, a, s'\rangle, Fact(T_{t-1}))$
   (e) $Fact(R_t) \leftarrow LearnR(\langle s, a, r\rangle, Fact(R_{t-1}))$
   (f) $\{Fact(V_t), \{\forall a \in A : Fact(Q^{\pi}_{t}{}'^{*}(s,a))\}\} \leftarrow Plan(Fact(R_t), Fact(T_t), Fact(V_{t-1}))$

FIG. 3 – L'algorithme SDYNA

La phase d'action de SDYNA (étapes 2.a à 2.c) est similaire à celle de DYNA et des politiques d'exploration telles que ϵ-$greedy$ peuvent être utilisées sans modification. Au contraire, les phases d'apprentissage et de planification dépendent directement de la représentation factorisée utilisée. Contrairement à DYNA-Q, SDYNA ne marche pas si l'on désactive la phase de planification.

Dans la suite, nous nous focalisons sur SPITI, une instanciation de SDYNA qui utilise des arbres de décision pour représenter les fonctions manipulées. La représentation arborescente d'une fonction F est notée $Tree(F)$. La phase d'action de SPITI est instanciée par une politique d'exploration ϵ-$greedy$. Sa phase d'apprentissage (étapes 2.d et 2.e de SDYNA) est fondée sur l'induction incrémentale d'arbres de décision décrite à la section 3.1. Sa phase de planification (étape 2.f) est fondée sur une version incrémentale modifiée de *Structured Value Iteration* (SVI), un algorithme proposé par Boutilier et al. (2000) et décrit à la section 3.2.

3.1 SPITI : apprentissage d'un modèle structuré

Quand un agent exécute une action a dans un état s, il peut exploiter en retour deux informations : son nouvel état s' et sa récompense r. Des algorithmes incrementaux de classification apprennent une fonction à partir d'un ensemble d'exemples $\langle \mathcal{A}, \varsigma\rangle$ où $\mathcal{A}$ est un ensemble d'*attributs* ν_i et ς est la *classe* de l'exemple. Par conséquent, à partir de l'observation de l'agent $\langle s, a, s', r\rangle$ avec $s = (x_1, ..., x_n)$ et $s' = (x'_1, ..., x'_n)$, nous proposons d'apprendre la récompense $R(s,a)$ et les modèles de transition $CPD^a_{X_i}$ à partir des exemples $\langle \mathcal{A} = (x_1, ..., x_n, a), \varsigma = r\rangle$ et $\langle \mathcal{A} = (x_1, ..., x_n), \varsigma = x'_i\rangle$, respectivement.

Les exemples successifs reçus par l'agent dans son environnement constituent un flux qui doit être appris de façon incrémentale. Dans SPITI, nous utilisons ITI (Utgoff et al., 1997), un algorithme d'induction incrémentale d'arbres de décision. ITI, noté $ITI(Tree(F), \mathcal{A}, \varsigma)$, est capable de construire un arbre de décision $Tree(F)$ à partir d'un flux d'exemples $\langle \mathcal{A}, \varsigma\rangle$. Nous renvoyons le lecteur à Utgoff et al. (1997) pour une description complète d'ITI.

Entrée: s, a, r Sortie: $\emptyset$

1. $\mathcal{A} \leftarrow \{x_1, ..., x_n, a\}$
2. $\varsigma \leftarrow r$
3. $ITI(Tree(R), \mathcal{A}, \varsigma)$

FIG. 4 – SPITI (1) : l'algorithme $LearnR(s, a, r)$.

L'algorithme d'apprentissage de la récompense est décrit sur la figure 4. Un exemple est composé du vecteur $s = (x_1, ..., x_n)$ et de l'action a, tandis que sa classe ς est définie par la récompense reçue par l'agent. L'exemple est intégré dans l'arbre par ITI.

Entrée: s, a, s' Sortie: $\emptyset$

1. $\mathcal{A} \leftarrow \{x_1, ..., x_n\}$
2. $\forall i \in |X|$:
   (a) $\varsigma \leftarrow x'_i$
   (b) $ITI(Tree(CPD^a_{X_i})), \mathcal{A}, \varsigma)$

FIG. 5 – SPITI (2) : l'algorithme $LearnT(s, a, s')$.

Le modèle des transitions T_a de l'action a est composé du graphe G_a et de l'ensemble $CPD^a = (CPD^a_{X_1}, ..., CPD^a_{X_n})$. Nous proposons d'apprendre separément chaque $CPD^a_{X_i}$ sous la forme d'un arbre de décision sans essayer de construire le DBN G_a. L'algorithme $LearnT(s, a, s')$ correspondant est décrit à la figure 5. On utilise un arbre par $CPD^a_{X_i}$. L'arbre est construit en apprenant des exemples composés du vecteur $s = \{x_1, ..., x_n\}$ tandis que la classe ς est définie par l'instanciation des X'_i dans l'état s' du système. Cette méthode est valide car nous considérons des problèmes dépourvus d'arcs synchrones, donc $X'_i \perp\!\!\!\perp X'_j \,|\, X_1, \cdots, X_n$.

Pour déterminer quel test installer sur chaque nouveau nœud de l'arbre de décision, les algorithmes d'induction

incrémentale d'arbres de décision utilisent des mesures tirées de la théorie de l'information, qui sont calculées pour chaque attribut $\nu_i \in \mathcal{A}$. Différentes mesures ont été proposées, telles que *gain ratio*, *Kolmogorov-Smirnoff* ou χ^2. Dans SPITI, nous utilisons χ^2 car la mesure peut aussi être utilisée en tant que test d'indépendance entre deux distributions de probabilité, comme le suggère Quinlan (1986).

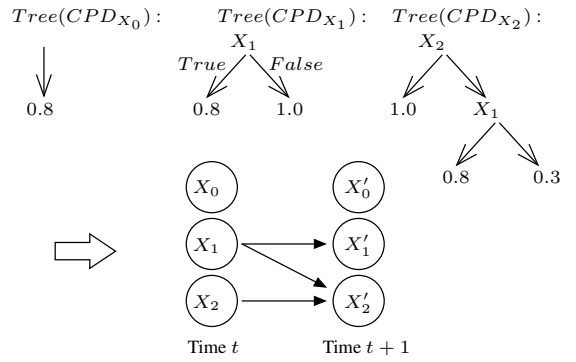

FIG. 6 – Structure du DBN G obtenu à partir d'un ensemble d'arbres $\{Tree(CPD_{X_i})\}$. Dans $Tree(CPD_{X_2})$, la feuille dont le label est 0.3 indique que la probabilité que X'_2 soit vrai est $P(X'_2|X_1 = False, X_2 = False) = 0.3$.

Ces tests sont destinés à empêcher la sur-spécialisation et sont nécessaires pour apprendre le modèle des transitions pour des problèmes stochastiques. La probabilité $CPD^a_{X_i}(X'_i|Parents_a(X'_i))$ est calculée pour chaque feuille de $Tree(CPD^a_{X_i})$ à partir des exemples stockés dans cette feuille. De plus, on pourrait reconstituer le DBN G_a en assignant à $Parents_a(X'_i)$ l'ensemble des variables X_i utilisées dans les tests installés dans chaque $Tree(CPD^a_{X_i})$ (voir la figure 6 pour un exemple).

3.2 SPITI : planification factorisée

L'algorithme de planification de SPITI est une version incrémentale modifiée de l'algorithme Structured Value Iteration (SVI) proposé par Boutilier et al. (2000). SVI est l'adaptation de Value Iteration aux représentations factorisées. Il est décrit à la figure 7.

Entrée: $\langle G_a|P_a\rangle, Tree(R)$ Sortie: $Tree(\pi^*)$

1. $Tree(V) \leftarrow Tree(R)$
2. Répéter jusqu'à l'arrêt :
   (a) $\forall a \in A : Tree(Q^V_a) \leftarrow Regress(Tree(V), a)$
   (b) $Tree(V) \leftarrow Merge(\{Tree(Q^V_a) : \forall a \in A\})$ (en utilisant la maximisation des valeurs comme fonction de combinaison).
3. $\forall a \in A : Tree(Q^V_a) \leftarrow Regress(Tree(V), a)$
4. $Tree(\pi) \leftarrow Merge(\{Tree(Q^V_a) : \forall a \in A\})$ (en utilisant la maximisation des valeurs comme fonction de combinaison et en plaçant les actions comme label des feuilles).
5. Renvoyer $Tree(\pi)$

FIG. 7 – L'algorithme Structured Value Iteration (SVI).

SVI repose sur deux opérateurs : $Merge(\{T_1, ..., T_n\})$ et $Regress(Tree(V), a)$. $Merge$ produit un arbre unique qui résulte de la fusion des arbres $T_1, ..., T_n$ et dont le label des feuilles est calculé au moyen d'une *fonction de combinaison* des labels des feuilles impliquées dans la fusion des arbres originaux[2]. $Regress$ produit $Tree(Q^V_a)$, qui représente l'utilité espérée de l'action a contenu de la fonction de valeur $Tree(V)$. Il itère l'équation (1) pour tous les états en s'appuyant sur la représentation arborescente. Nous renvoyons à Boutilier et al. (2000) pour une description plus détaillée des opérateurs $Merge$ et $Regress$.

Utiliser SVI tel quel dans SPITI est possible, mais peu efficace pour deux raisons. Tout d'abord, SVI améliorerait la fonction de valeur jusqu'à convergence, malgré un modèle incomplet. Ensuite, la sortie de SVI est une politique gloutonne qui peut s'avérer inadaptée pour un problème dans lequel les fonctions de transition et de récompense sont inconnues. Dans un tel cadre, les agents doivent essayer des actions sous-optimales pour acquérir de l'information. Nous proposons donc dans SPITI la version incrémentale modifiée de SVI décrite sur la figure 8.

Entrée: $Tree(R_t), Tree(T_t), Tree(V_{t-1})$
Sortie: $Tree(V_t), \{\forall a \in A : Tree(Q_t(s,a))\}$

1. $\forall a \in A : Tree(Q_t(s,a)) \leftarrow Regress(Tree(V_{t-1}), a)$
2. $Tree(V_t) \leftarrow Merge(\{Tree(Q_t(s,a)) : \forall a \in A\})$ (en utilisant la maximisation des valeurs comme fonction de combinaison).
3. Renvoyer $\{Tree(V_t), \{\forall a \in A : Tree(Q_t(s,a))\}\}$

FIG. 8 – SPITI (3) : l'algorithme $Plan$ fondé sur SVI.

À chaque pas de temps, l'algorithme $Plan$ effectue une itération de SVI et met à jour pour chaque action a la représentation structurée de sa fonction de valeur $Tree(Q_t(s,a))$. La politique gloutonne est construite en recherchant la meilleure action dans l'état s en fonction des valeurs de $Tree(Q_t(s,a))$. Cette information est ensuite utilisée par la phase d'action pour réaliser une politique ϵ-$greedy$ (Sutton & Barto, 1998).

La fonction de valeur $Tree(V_t)$ est calculée en fusionnant les fonctions de valeur d'action $\{Tree(Q_t(s,a)) : \forall a \in A\}$ (en utilisant la maximisation des valeurs comme fonction de combinaison). $Tree(V_t)$ est ensuite réutilisé à l'instant $t+1$ pour calculer les fonctions de valeur d'action $Tree(Q_{t+1}(s,a))$. Si les modèles des transitions et des récompenses appris par l'agent sont stationnaires, $Tree(V_t)$ converge vers la fonction de valeur optimale $Tree(V^*)$ sous-tendue par la connaissance actuelle de l'agent (Boutilier et al., 2000). Cependant, tant que les modèles appris par l'agent ne sont pas assez précis, $Tree(V^*)$ peut différer significativement de la fonction de valeur optimale du problème d'A/R à résoudre.

[2] La fonction de combinaison des labels utilisée durant le processus de fusion est noté dans la description des algorithmes qui l'utilisent.

4 Résultats

Nous évaluons SPITI empiriquement sur deux problèmes[3], à savoir *Coffee Robot* et *Process Planning*, définis dans (Boutilier et al., 2000). Pour chaque problème, nous comparons SPITI à DYNA-Q. Pour chaque algorithme, on fixe $\gamma = 0.9$ et on utilise une politique d'exploration ϵ-$greedy$ avec ϵ fixé à 0.1. Dans DYNA-Q, on utilise $\alpha = 1.0$ et N vaut deux fois la taille du modèle. La fonction $Q(s, a)$ est initialisée de façon optimiste à une valeur élevée ce qui favorise l'exploration. Dans SPITI, le seuil τ_{χ^2} utilisé pour détecter l'indépendance entre deux distributions est fixé à 10.6, ce qui correspond à une probabilité d'indépendance supérieure à 0.99.

À des fins de comparaison, nous faisons aussi tourner deux agents, notés RANDOM et OPTIMAL, qui exécutent respectivement une politique aléatoire et la politique optimale. L'agent RANDOM apprend les modèles des transitions et de la récompense à l'aide des algorithmes $LearnR$ et $LearnT$ de SPITI. La politique de l'agent OPTIMAL est calculée hors-ligne en utilisant SVI avec un critère d'arrêt fondé sur la norme infinie avec un seuil à 0.01 et avec un modèle des transitions et de la récompense parfaits donnés a priori.

Dans le but d'effectuer des essais successifs au sein d'un même problème, on ajoute à leur définition un ensemble d'états initiaux et un ensemble d'états terminaux. Dès qu'un agent atteint un état terminal, il est réinitialisé dans un état choisi aléatoirement avec une distribution uniforme parmi les états initiaux. Chaque agent est évalué durant T pas de temps.

Les résultats portent sur la récompense actualisée R^{disc} obtenue par l'agent et la taille du modèle des transitions construit durant les expériences. R_t^{disc} est définie par $R_t^{disc} = r_t + \gamma' R_{t-1}^{disc}$, où r_t est la récompense immédiate reçue par l'agent et $\gamma' = 0.99$[4]. Nous ne montrons pas les résultats relatifs à la taille du modèle de la récompense car ils sont similaires à ceux obtenus pour le modèle des transitions, ni ceux relatifs à la taille de la fonction de valeur ou au temps de calcul car Boutilier et al. (2000) fournit des résultats exhaustifs sur ces points.

4.1 Le problème *Coffee Robot*

Le problème *Coffee Robot* [5] est un problème stochastique de petite taille dans lequel un robot doit se rendre dans un café, y acheter un café et le remettre à sa propriétaire, sans se faire mouiller en chemin. Le problème est composé de 4 actions et 6 variables booléennes, définissant $2^6 * 4 = 256$ couples $s \times a$. Les états terminaux sont ceux dans lesquels la propriétaire a un café, tout état non terminal est un état initial possible. Nous faisons tourner 30 expériences pour chaque agent avec $T = 4000$.

La figure 9 montre la récompense actualisée obtenue par chaque agent. DYNA-Q et SPITI se comportent de façon similaire sur ce petit problème et convergent en moins de 1000 pas de temps vers une politique quasi-optimale. Cependant, ils n'atteignent pas la politique optimale à cause du facteur d'exploration ϵ qui ne décroît pas.

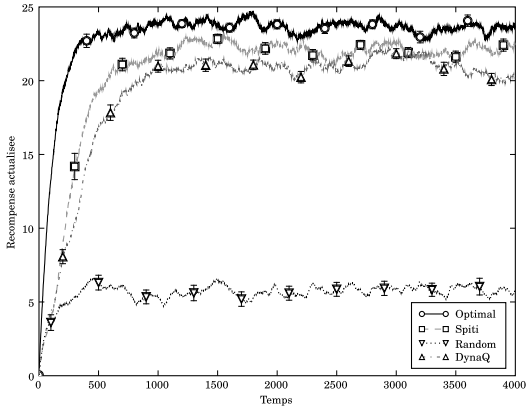

FIG. 9 – Récompense actualisée sur le problème *Coffee Robot*.

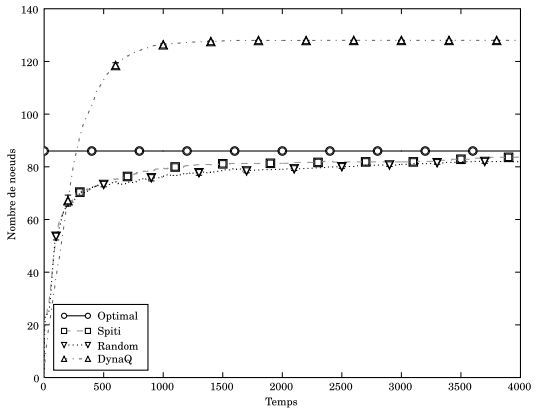

FIG. 10 – Taille du modèle des transitions sur le problème *Coffee Robot*.

La figure 10 permet de comparer les tailles des modèles des transitions appris par SPITI, DYNA-Q et RANDOM. Le nombre de nœuds du modèle construit par DYNA-Q est égale au nombre de transitions du problème (128 transitions, sachant que les états terminaux n'ont pas de transition). Les modèles construits par SPITI et RANDOM sont significativement plus petits, avec moins de 90 nœuds, car ils construisent une représentation structurée du problème. La récompense actualisée obtenue par SPITI (figure 9) montre que cette représentation est exploitée correctement par l'algorithme de planification pour construire incrémentalement une politique quasi-optimale. De plus, puisqu'ils sont obtenus avec le même algorithme d'appren-

[3] Une définition détaillée de ces problèmes est disponible sur le site de SPUDD `http ://www.cs.ubc.ca/spider/jhoey/spudd/spudd.html`.

[4] $\gamma' = 0.99$ rend les résultats plus lisibles que $\gamma' = 0.9$.

[5] Appelé "*Coffee*" sur le site de SPUDD

tissage, les modèles des transitions construits par RANDOM et SPITI sont de taille voisine malgré la différence de politique.

4.2 Le problème *Process Planning*

Le problème *Process Planning* [6] est un problème stochastique de grande taille composé de 14 actions et 17 variables booléennes, définissant $2^{17} * 14 = 1835008$ couples $s \times a$. Un objet doit être produit en rassemblant deux composants manufacturés. On peut produire des composants de qualité élevée en utilisant des actions telles que peindre à la main ou riveter ou des composants de faible qualité en utilisant des actions telles que peindre au jet ou encoller. L'agent doit produire des composants de qualité élevée ou faible en fonction d'une demande variable. Les états terminaux sont les états dans lesquels les deux composants sont rassemblés. Les états initiaux sont tous les états à partir desquels on peut atteindre un état terminal. Nous faisons tourner 20 expériences pour chaque agent avec $T = 10000$.

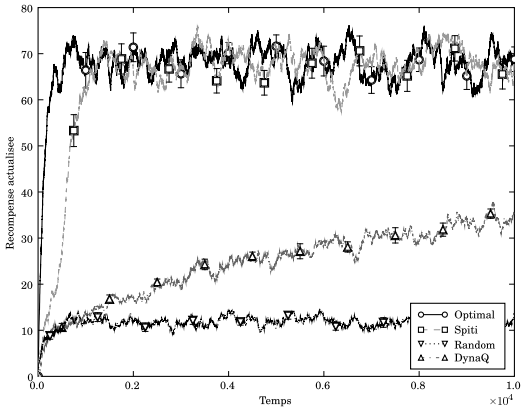

FIG. 11 – Récompense actualisée sur le problème *Process Planning*.

La figure 11 montre la récompense actualisée obtenue par chaque agent. SPITI converge vers une politique quasi-optimale en 2000 pas de temps environ contrairement à DYNA-Q qui progresse toujours après 10000 pas de temps. Cette différence illustre le fait qu'utiliser un algorithme d'induction incrémentale d'arbres de décision apporte une capacité de généralisation. Dans les problèmes de grande taille, visiter tous les couples $s \times a$ est rarement possible. Grâce à la généralisation, l'agent dirigé par SPITI peut choisir une action adéquate pour des états qu'il n'a pas encore visités. Au contraire, DYNA-Q doit essayer chaque couple $s \times a$ avant de les prendre en compte dans la planification.
La figure 12 permet de comparer les tailles des modèles des transitions appris par les agents. De même que dans le problème *Coffee Robot*, la taille du modèle de DYNA-Q croît linéairement avec le nombre d'états et d'actions

[6] Appelé "*Factory*" sur le site de SPUDD

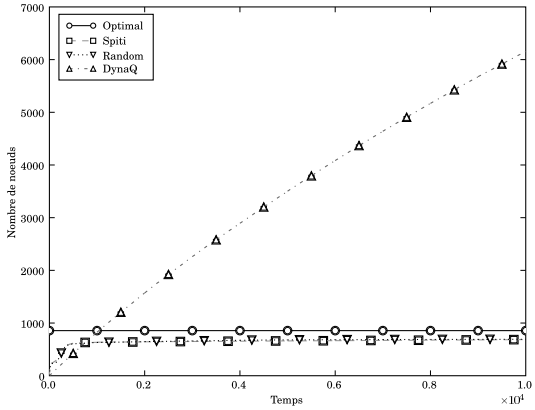

FIG. 12 – Taille du modèle des transitions sur le problème *Process Planning*.

du problème, tandis que SPITI construit rapidement une représentation compacte du problème suffisamment précise pour que la politique qui en découle soit proche de l'optimum (figure 11).

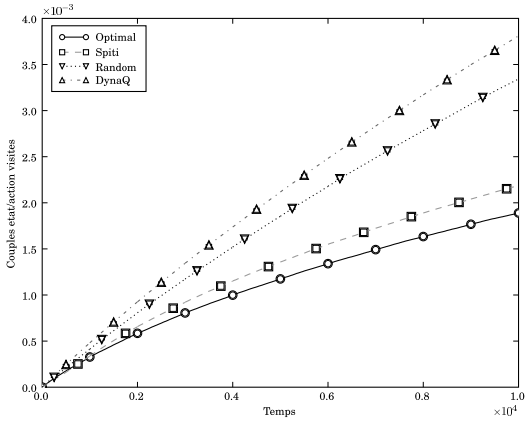

FIG. 13 – Proportion de couples $s \times a$ visités sur le nombre de couples du problème *Process Planning*

La figure 13 montre le nombre de couples $s \times a$ visités par chaque agent. Elle fait ressortir la politique d'exploration intensive de RANDOM et DYNA-Q par comparaison avec SPITI ou OPTIMAL. Contrairement à DYNA-Q et malgré un nombre plus important de couples $s \times a$ visités, le modèle construit par RANDOM est d'une taille similaire à celle du modèle obtenu par SPITI, ce qui montre que la taille du modèle dépend plus de la structure du problème que du nombre de couples $s \times a$ visités.

5 Discussion

La première contribution de cet article consiste en la proposition de SDYNA, une approche générale pour traiter des problèmes d'A/R de grande taille par essais et erreur et sans

connaissance a priori de leur structure. En particulier, nous avons décrit une instanciation de SDYNA appelée SPITI, qui combine une version modifiée de SVI avec ITI, un algorithme d'induction incrémentale d'arbres de décision qui permet d'apprendre la structure d'un problème d'A/R.

Des algorithmes de planification alternatifs à SVI ont été proposés, tels que SPUDD (St-Aubin et al., 2000) ou des techniques fondées sur la programmation linéaire (Guestrin et al., 2003). Ces techniques apparaissent plus rapides et moins gourmandes en mémoire que SVI. Notre version incrémentale de SVI était la méthode de planification la plus facile à mettre en œuvre pour valider notre approche, mais nous nous efforçons à présent d'intégrer une méthode plus efficace dans SDYNA. Ceci dit, le recours à ces algorithmes de planification plus efficaces ne changera pas fondamentalement les résultats présentés ici. Comme nous l'avons montré ci-dessus, la taille du modèle ne dépend pas strictement de la politique de l'agent. De plus, étant donné un modèle à l'instant t, deux politiques gloutonnes calculées par deux algorithmes de planification différents ont des fonctions de valeur similaires. Par contre, le recours à ces méthodes plus efficaces doit nous permettre de traiter des problèmes d'A/R de plus grande taille.

La seconde contribution de cet article réside dans la mise en évidence de l'intérêt de la propriété de généralisation des algorithmes d'induction d'arbres de décision pour la résolution de problèmes de décision séquentiels dans le cadre des FMDP. Dans SPITI, cette propriété dépend de la valeur du seuil τ_{χ^2} utilisé dans le test du χ^2 pour détecter l'indépendance entre deux distributions. Pour un problème stochastique, ce paramètre maintien les processus de création et de fusion de nœuds dans les arbres de décision entre deux extrêmes : $Parents_a(X'_i) = \emptyset$ et $Parents_a(X'_i) = \{\forall i, X_i\}$. Dans les études expérimentales décrites dans cet article, nous avons fixé τ_{χ^2} à 10.6, ce qui correspond à une probabilité d'indépendance supérieure à 0.99. Nous étudions par ailleurs les relations entre la valeur de τ_{χ^2}, la taille du modèle appris et la qualité de la politique obtenue au bout du compte.

De plus, nous avons fait ressortir l'utilité de la propriété de généralisation pour définir des politiques d'exploration efficaces dans des problèmes de grande taille, car cette propriété permet d'éviter une exploration exhaustive de l'environnement. Des recherches récentes sur la gestion du compromis entre exploration et exploitation ont permis de mettre au point des algorithmes efficaces pour le cadre factorisé tels que $Factored\ E^3$ et $Factored\ R_{max}$ (Guestrin et al., 2002). Ces algorithmes bénéficient de garanties théoriques de convergence intéressantes, mais imposent des contraintes fortes sur les FMDP qu'ils peuvent résoudre. Par conséquent, des travaux supplémentaires sont nécessaires pour intégrer ces algorithmes dans SDYNA sans remettre en question la construction incrémentale du modèle des transitions et des récompenses.

6 Conclusion

Dans cet article, nous avons présenté SDYNA, une architecture conçue pour intégrer la planification, l'action et l'apprentissage en utilisant les représentations factorisées des FMDP plutôt que des représentations tabulaires. Au travers d'une instanciation de SDYNA appelée SPITI, nous avons montré que cette architecture peut construire une représentation compacte d'un problème d'A/R dont la structure n'est pas connue à l'avance. De plus, nous avons montré que, pour un ensemble donné d'états visités, la propriété de généralisation des algorithmes d'induction d'arbres de décision accélère l'amélioration de la politique vis-à-vis des méthodes fondées sur des représentations tabulaires.

Références

Boutilier, C., Dearden, R., & Goldszmidt, M. (1995). Exploiting Structure in Policy Construction. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)* (pp. 1104–1111). Montreal.

Boutilier, C., Dearden, R., & Goldszmidt, M. (2000). Stochastic Dynamic Programming with Factored Representations. *Artificial Intelligence*, *121*, 49–107.

Chickering, D. M. (1996). Learning Bayesian Networks is NP-complete. *Learning from Data : Artificial Intelligence and Statistics V*.

Dean, T., & Kanazawa, K. (1989). A Model for Reasoning about Persistence and Causation. *Computational Intelligence*, *5*, 142–150.

Guestrin, C., Koller, D., Parr, R., & Venkataraman, S. (2003). Efficient Solution Algorithms for Factored MDPs. *Journal of Artificial Intelligence Research*, *19*, 399–468.

Guestrin, C., Patrascu, R., & SchuurmansKoller, D. (2002). Algorithm-Directed Exploration for Model-Based Reinforcement Learning in Factored MDPs. *ICML-2002 The Nineteenth International Conference on Machine Learning* (pp. 235–242).

Hoey, J., St-Aubin, R., Hu, A., & Boutilier, C. (1999). SPUDD : Stochastic Planning using Decision Diagrams. *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence* (pp. 279–288). Morgan Kaufmann.

McCallum, A. K. (1995). *Reinforcement Learning with Selective Perception and Hidden State*. Doctoral dissertation, Department of Computer Science, University of Rochester, USA.

Quinlan, J. R. (1986). Induction of Decision Trees. *Machine Learning*, *1*, 81–106.

Sallans, B., & Hinton, G. E. (2004). Reinforcement Learning with Factored States and Actions. *Journal of Machine Learning Research*, *5*, 1063–1088.

St-Aubin, R., Hoey, J., & Boutilier, C. (2000). APRICODD : Approximate Policy Construction Using Decision Diagrams. *NIPS* (pp. 1089–1095).

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 216–224). San Mateo, CA. Morgan Kaufmann.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning : An Introduction*. MIT Press.

Utgoff, P. (1986). Incremental Induction of Decision Trees. *Machine Learning*, *4*, 161–186.

Utgoff, P. E., Nerkman, N. C., & Clouse, J. A. (1997). Decision Tree Induction Based on Efficient Tree Restructuring. *Machine Learning*, *29*, 5–44.