Apprentissage par renforcement exploitant la structure additive des MDP factorisés

Thomas Degris, Olivier Sigaud et Pierre-Henri Wuillemin

Université Pierre Marie Curie Paris 6 ; ISIR (CNRS FRE 2507) et LIP6 (CNRS UMR7606) ; 104, avenue du Président Kennedy F-75016 PARIS
prenom.nom@lip6.fr

Résumé :

SDYNA est un cadre algorithmique générique pour traiter des problèmes d'apprentissage par renforcement factorisés de grande taille dont la structure est inconnue selon une approche à base de modèles. L'approche consiste à construire incrémentalement les fonctions de transition et de récompense d'un FMDP tout en utilisant des techniques de planification propres aux FMDP pour déterminer une politique efficace. Des instanciations précédentes de SDYNA utilisaient *Structured Value Iteration* pour planifier, mais n'exploitaient pas la structure additive d'un FMDP. Dans cette contribution, nous présentons une nouvelle instanciation reposant sur la programmation linéaire, capable de traiter des problèmes de 10^{11} états dont la structure est inconnue en exploitant leur structure additive. De plus, nous proposons un nouvel algorithme d'apprentissage qui accélère la construction du modèle par SDYNA.

1 Introduction

Les Processus Décisionnels de Markov (MDP) constituent un cadre privilégié pour modéliser des problèmes de planification et d'apprentissage par renforcement dans l'incertain. Les méthodes de base conçues dans ce cadre sont inaptes à traiter des problèmes de grande taille parce qu'elles imposent d'énumérer explicitement l'ensemble des états ou des couples états/actions. Les MDP factorisés (FMDP), proposés par Boutilier *et al.* (1995), supposent que l'espace d'états se décompose en un ensemble de variables aléatoires. En exploitant les dépendances entre ces variables, spécifiées par des réseaux bayésiens dynamiques (DBN) (Dean & Kanazawa, 1989), ils permettent de représenter de façon compacte les fonctions de transition et de récompense de grands problèmes structurés.

Des méthodes de planification dédiées ont été développées pour traiter les cas où les fonctions de transition et de récompense sont spécifiées a priori. Elles reposent sur deux techniques classiques, à savoir la programmation dynamique et la programmation linéaire. *Structured Policy Iteration* (SPI), *Structured Value Iteration* (SVI) et *Stochastic Planning Using Decision Diagrams* (SPUDD) étendent la programmation dynamique

(Boutilier *et al.*, 2000; Hoey *et al.*, 1999) et exploitent des indépendances contextuelles sous la forme d'arbres de décision ou de diagrammes de décision. Par ailleurs, Guestrin *et al.* (2003) proposent des algorithmes fondés sur la programmation linéaire, qui exploitent la décomposition additive de la fonction de récompense et une approximation de la décomposition additive de la fonction de valeur. Nous utiliserons le terme de *structure additive* d'un problème pour regrouper ces deux concepts.

Lorsque la structure des fonctions de transition et de récompense n'est pas connue a priori, (Degris *et al.*, 2006b) ont proposé *Structured* DYNA (SDYNA) qui combine des algorithmes d'apprentissage supervisé avec les méthodes de planification précédentes pour résoudre des problèmes d'apprentissage par renforcement de grande taille. SPITI est une instance de SDYNA qui utilise un algorithme d'induction incrémentale d'arbres de décision combiné avec une version incrémentale de SVI. Par conséquent, SPITI utilise l'indépendance contextuelle mais ne peut pas exploiter d'autres régularités des FMDP telles que la structure additive.

Dans cette contribution, après avoir présenté les FMDP, les méthodes de programmation linéaire et SDYNA dans la section 2, nous décrivons deux nouvelles instances de SDYNA, nommées ULP et UNATLP, qui reposent sur la programmation linéaire et exploitent la structure additive des problèmes. De plus, UNATLP utilise un nouvel algorithme qui représente différemment la structure du FMDP. Nous montrons dans la section 4 que ces deux instances traitent des problèmes de 10^{11} états, ce qui était, à notre connaissance, hors de portée de toutes les méthodes d'apprentissage par renforcement existantes basées sur la construction d'un modèle. Par ailleurs, nous montrons la supériorité de UNATLP sur SPITI et ULP sur de tels problèmes.

2 Cadre théorique

Un MDP est défini par un tuple $\langle S, A, R, P\rangle$ où S est un ensemble fini d'états, A un ensemble fini d'actions, R une fonction de récompense immédiate avec $R : S\times A \to \mathbb{R}$ et P une fonction de transition $P(s'|s,a)$ avec $P : S \times A \times S \to [0,1]$. Une politique stationnaire π déterministe est une fonction $S \to A$ où $\pi(s)$ définit l'action à réaliser dans l'état s.

Une fonction de valeur $V_\pi(s)$ est définie avec un 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 et r_t la récompense obtenue à l'instant t. Une politique π est optimale si $\forall s \in S, \forall \pi' : V_\pi(s) \geq V_{\pi'}(s)$. La fonction de valeur d'une politique optimale π^* est appelée « fonction de valeur optimale » et notée V^*. La fonction de valeur d'action $Q_a^V(s)$ pour une action a et une fonction de valeur $V(s)$ est définie par $Q_a^V(s) = R(s,a) + \gamma \sum_{s'\in S} P(s'|s,a)V(s')$.

À partir d'une fonction de valeur V donnée, il est possible de définir une politique gloutonne relativement à V, notée Greedy_V, en choisissant pour chaque état s l'action ayant la plus grande valeur d'action :

$$\text{Greedy}_V(s) = \arg\max_a [R(s,a) + \gamma \sum_{s'\in S} P(s'|s,a)V(s')] = \arg\max_a [Q_a^V(s)] \quad (1)$$

La politique gloutonne relativement à V^* est une politique optimale définie telle que $\pi^*(s) = \text{Greedy}_{V^*}(s)$.

2.1 Processus Décisionnels de Markov factorisés

Nous supposons à présent que les états sont constitués d'un ensemble de variables aléatoires $X = \{X_1, \ldots, X_n\}$. Un état est alors défini par un vecteur $s = (x_1, \ldots, x_n)$ avec $\forall i, x_i \in \text{Dom}(X_i)$. Les FMDP exploitent cette structure de la façon suivante (Boutilier *et al.*, 1995). Pour chaque action a, le modèle des transitions du FMDP est défini par un DBN distinct $T_a = \langle G_a, \{P^a_{X_1}, \ldots, P^a_{X_n}\}\rangle$. G_a est un graphe orienté acyclique composé de deux couches correspondant à deux ensembles $\{X_1, \ldots, X_n\}$ et $\{X'_1, \ldots, X'_n\}$ de nœuds où X_i est une variable au temps t et X'_i la même variable au temps $t+1$. Les parents de X'_i sont notés $\text{Parents}_a(X'_i)$. Nous supposons que $\text{Parents}_a(X'_i) \subseteq X$, ce qui signifie qu'il n'y a pas d'arcs synchrones. Le modèle des transitions T_a est quantifié par des *distributions de probabilités conditionnelles* (CPD) $P^a_{X_i}(X'_i|\text{Parents}_a(X'_i))$ associées à chaque nœud $X'_i \in G_a$.

On peut représenter de façon similaire la fonction de récompense. On définit la notion de fonction localisée (Guestrin *et al.*, 2003) : une fonction f a un *scope* $\text{Scope}(f) = C \subseteq X$ si $f : \text{Dom}(C) \mapsto \mathbb{R}$. Si $\text{Scope}(f) = Y$ avec $Y \subseteq X$, on note $f(x)$ pour $f(y)$ où y est la partie de l'instanciation x correspondant aux variables appartenant à Y. La fonction de récompense est décrite comme la somme $\sum_{j=1}^{r} R_j(s) \in \mathbb{R}$ où chaque fonction R_j est une fonction localisée dont le scope $\text{Scope}(R_j)$ est limité à un petit nombre de variables. Il peut y avoir une décomposition R^a_j différente pour chaque action a.

2.2 Méthodes fondées sur la programmation linéaire

La fonction de valeur optimale d'un MDP peut être calculée en formulant celui-ci sous la forme d'un programme linéaire (Manne, 1960) :

Pour les variables: $V(s), \forall s \in S$;
Minimiser: $\sum_s \alpha(s)V(s)$;
Avec les contraintes : $V(s) \geq R(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s') \forall s \in S, \forall a \in A$. (LP 1)

où $\alpha(s)$ est la pondération d'intérêt de l'état s. Cette formulation se heurte à deux inconvénients majeurs. Elle suppose plusieurs énumérations explicites des états, et elle impose de connaître a priori les fonctions de transition et de récompense. Nous traitons ces deux points dans les deux sections qui suivent.

2.2.1 Approximation des MDP par programmation linéaire

Une approche pour traiter des problèmes de grande taille consiste à approcher la fonction de valeur sous la forme d'une combinaison linéaire de fonctions de base $H = \{h_1, \ldots, h_k\}$. Une fonction de valeur linéaire sur H est une fonction $\mathcal{V}$ qui peut être écrite $\mathcal{V}(s) = \sum_{i=1}^{k} w_i h_i(s)$ avec des coefficients $(w_1, \ldots, w_k)$ où chaque h_i est une fonction localisée. (LP 1) peut alors être ré-écrit sous une forme approchée dans H (Schweitzer & Seidmann, 1985) :

$$\begin{array}{ll}\text{Pour les variables:} & w_1, \ldots, w_k \text{ ;} \\ \text{Minimiser:} & \sum_s \alpha(s) \sum_{i=1}^k w_i h_i(s) \text{ ;} \\ \text{Avec les contraintes :} & \sum_{i=1}^k w_i h_i(s) \geq R(s,a) + \gamma \sum_{s'} P(s'|s,a) \sum_{i=1}^k w_i h_i(s') \\ & \forall s \in S, \forall a \in A. \end{array} \quad \text{(LP 2)}$$

La fonction de valeur est ainsi remplacée par son approximation $\mathcal{V}(s) = \sum_{i=1}^k w_i h_i(s)$. Il est garanti qu'une solution existe à (LP 2) si une fonction de base constante, notée h_0, est incluse à l'ensemble des fonctions de base. Nous supposerons que c'est le cas pour la suite de l'article.

La formulation de (LP 2) réduit le nombre de variables libres de $|S|$ à k. Cependant, le nombre de contraintes est toujours de $|S| \times |A|$, chaque contrainte s'exprimant potentiellement par une somme de $|S|$ termes et la fonction objectif s'exprimant toujours comme une somme de $|S|$ termes. Nous décrivons maintenant la méthode proposée par Guestrin *et al.* (2003) pour réduire la taille de ce programme linéaire en exploitant la structure du problème.

2.2.2 Approximation des FMDP par programmation linéaire

La représentation compacte de (LP 2) est constituée d'un ensemble de fonctions de valeur linéaires reposant sur la base $h_1, \ldots, h_k$, où chaque fonction linéaire h_i est restreinte à un petit nombre de variables d'état (Koller & Parr, 1999). Pour réaliser l'approximation du FMDP, la première opération consiste à calculer l'espérance de la valeur, notée $g_i^a(s)$, de la fonction de base h_i pour une action a, G_a étant fixé :

$$g_i^a(s) = \sum_{s'} P(s'|s,a) h_i(s') \quad (2)$$

Sachant que le scope d'une fonction h_i est petit, le calcul de $g_i^a(s)$ peut être simplifié (Koller & Parr, 1999). Le scope de $g_i^a(s)$ est $\text{Scope}(g_i^a) = \cup_{X'_j \in \text{Scope}(h_i)} \text{Parents}_a(X'_j)$. Par conséquent, le coût du calcul dépend linéairement de $|\text{Dom}(\text{Scope}(g_i^a))|$, qui dépend du scope de h_i et de la complexité de G_a. Une fois ces calculs effectués pour tous les $h_i \in H$ et toutes les actions $a \in A$, on peut réécrire les contraintes de (LP 2) :

$$\begin{array}{rl} & \sum_{i=1}^k w_i h_i(s) \geq R(s,a) + \gamma \sum_{s'} P(s'|s,a) \sum_{i=1}^k w_i h_i(s') \\ \iff & \sum_{i=1}^k w_i h_i(s) \geq R(s,a) + \gamma \sum_{i=1}^k w_i g_i^a(s) \\ \iff & 0 \geq R(s,a) + \sum_{i=1}^k w_i [\gamma g_i^a(s) - h_i(s)] \\ \iff & 0 \geq \sum_{j=1}^r R_j^a(s) + \sum_{i=1}^k w_i [\gamma g_i^a(s) - h_i(s)] \end{array}$$

sachant que la fonction de récompense du FMDP est décomposée comme une somme de fonctions localisées. Ces contraintes peuvent alors être simplifiées grâce au scope restreint des fonctions g_i^a, h_i et R_j^a.

Les pondérations d'intérêt $\alpha(s)$ de la fonction objectif de (LP 2) peuvent être considérés comme une distribution sur les états vérifiant $\sum_s \alpha(s) = 1$. Par conséquent, on peut appliquer une réorganisation similaire (Guestrin *et al.*, 2003) :

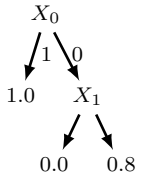
$$\sum_s \alpha(s) \sum_{i=1}^k w_i h_i(s) = \sum_{i=1}^k w_i \sum_{c_i \in C_i} \alpha(c_i) h_i(c_i)$$

où $C_i = \text{Dom}(\text{Scope}(h_i))$ et $\alpha(c_i)$ la pondération d'intérêt α pour le scope C_i. Nous utilisons une distribution uniforme définie telle que $\alpha(s) = \frac{1}{|S|}$. Par conséquent, nous obtenons $\alpha(c_i) = \frac{1}{|C_i|}$. En utilisant la réorganisation des contraintes et en calculant à l'avance $\alpha_i = \sum_{c_i \in C_i} \alpha(c_i) h_i(c_i)$, le programme linéaire devient alors :

$$\begin{array}{ll} \text{Pour les variables:} & w_1, \ldots, w_k \text{ ;} \\ \text{Minimiser:} & \sum_{i=1}^{k} w_i \alpha_i \text{ ;} \\ \text{Avec les contraintes :} & 0 \geq \sum_{i=1}^{k} w_i [\gamma g_i^a(s) - h_i(s)] + \sum_{j=1}^{r} R_j^a(s) \\ & \forall s \in S, \forall a \in A. \end{array} \quad \text{(LP 3)}$$

(LP 3) n'utilise cette fois plus que k variables, k termes dans la fonction objectif, et chaque contrainte peut être calculée sur un scope restreint. Cependant, le nombre de contraintes reste exponentiel. Comme décrit par Guestrin *et al.* (2003), ce nombre peut être réduit en ayant recours à un algorithme d'élimination de variables exploitant la structure du problème. Nous noterons FactoredALP l'algorithme qui construit (LP 3) puis le résout. Nous renvoyons à Guestrin *et al.* (2003) pour une description complète de cet algorithme.

2.2.3 Indépendance contextuelle



(a)

$$\left| \begin{array}{ll} X_0 = 1 \wedge X_0' = 1 & : 1.0 \\ X_0 = 0 \wedge X_1 = 1 \wedge X_0' = 0 & : 1.0 \\ X_0 = 0 \wedge X_1 = 0 \wedge X_0' = 0 & : 0.2 \\ X_0 = 0 \wedge X_1 = 0 \wedge X_0' = 1 & : 0.8 \end{array} \right|$$

(b)

FIG. 1 – Représentation de CPD $P(X_0'|s)$ sous forme (a) d'arbre de décision $\mathsf{Tree}[P](X_0'|s)$ et (b) de règles $\mathsf{Rules}[P](X_0'|s)$.

De façon analogue à SPI, SVI et SPUDD (Boutilier *et al.*, 2000; Hoey *et al.*, 1999), Guestrin *et al.* (2003) utilisent une représentation à base de règles (Zhang & Poole, 1999) pour exploiter les indépendances contextuelles dans le calcul des $g_i^a(s)$ et pour réduire d'avantage le nombre de contraintes dans (LP 3).

Deux contextes $c \in \text{Dom}(C)$ et $b \in \text{Dom}(B)$ avec $C \subseteq \{X, X'\}$ et $B \subseteq \{X, X'\}$ sont dits *consistants* si les variables de $C \cap B$ ont les mêmes valeurs dans ces contextes. Une *règle de probabilité* $\eta = |c : p|$ est une fonction $\eta : \{X, X'\} \mapsto [0, 1]$, où le contexte $c \in \text{Dom}(C)$, $C \subseteq \{X, X'\}$ et $p \in [0, 1]$ et tel que $\eta(s, x') = p$ si s et x' sont consistants avec c et $\eta(s, x') = 1$ sinon. Une *distribution de probabilités conditionnelles à base de règles* est une fonction $P_a : (\{X_i'\} \cup X) \mapsto [0, 1]$ composée d'un ensemble de règles de probabilités $\{\eta_1, \ldots, \eta_n\}$ dont les contextes forment une partition. Une probabilité est définie par $P_a(x_i'|s) = \eta_j(s, x_i')$ où η_j est l'unique règle de P_a pour laquelle c_j est consistant avec s et x_i'. Enfin, on impose $\sum_{x_i'} P_a(x_i'|s) = 1 \; \forall s \in S$.

Une *règle de valeur* $\rho = |c : v|$ est une fonction $\rho : X \rightarrow \mathbb{R}$ telle que $\rho(s) = v$ quand s est consistant avec c et 0 sinon. Une *fonction à base de règles* $f : X \mapsto \mathbb{R}$ est composée d'un ensemble de règles $\{\rho_1, \ldots, \rho_n\}$ telles que $f(x) = \sum_{i=1}^{n} \rho_i(x)$. Les fonctions de base et la fonction de récompense peuvent être représentées sous forme de fonctions à base de règles.

On peut donc représenter les fonctions de transition, de valeur et de récompense d'un FMDP sous forme de règles. Guestrin *et al.* (2003) utilisent cette représentation pour reformuler le calcul des $g_i^a(s)$ et pour représenter (LP 3) de façon plus compacte en exploitant les indépendances contextuelles.

2.3 STRUCTURED DYNA et SPITI

La méthode de programmation linéaire décrite précédemment suppose que la structure du problème est connue, ce qui n'est généralement pas le cas. Récemment, Degris *et al.* (2006b) ont proposé *Structured* DYNA (SDYNA) qui apprend cette structure au fil de l'expérience (voir la figure 2, où $\mathsf{Fact}[F]$ désigne une représentation factorisée d'une fonction F).

Input: Acting, Learn, Plan, Fact
Output: $\emptyset$

1. Initialiser le FMDP $\hat{\mathcal{F}}_0$
2. A chaque instant t, faire :
   (a) $s \leftarrow$ l'état courant (non-terminal)
   (b) $a \leftarrow \text{Acting}(s, \{\mathsf{Fact}[Q_{t-1}^a], \forall a \in A\})$
   (c) Exécuter a ; observer s' et r
   (d) $\hat{\mathcal{F}}_t \leftarrow \text{Learn}(\hat{\mathcal{F}}_{t-1}, \langle s, a, s', r\rangle)$
   (e) $\{\mathsf{Fact}[V_t], \{\mathsf{Fact}[Q_t^a], \forall a \in A\}\} \leftarrow \text{Plan}(\hat{\mathcal{F}}_t, \mathsf{Fact}[V_{t-1}])$

FIG. 2 – L'algorithme SDYNA

SPITI est une instanciation de SDYNA qui apprend incrémentalement des représentations structurées des fonctions de transition et de récompense en s'appuyant sur un algorithme d'induction incrémentale d'arbres de décision. L'algorithme d'apprentissage de SPITI est décrit figure 3, où $\mathsf{Tree}[F]$ désigne la représentation sous forme d'arbre de la fonction F. L'algorithme $\text{UpdateTree}(\mathsf{Tree}[F], \mathcal{A}, \varsigma)$ met à jour $\mathsf{Tree}[F]$ en insérant l'exemple $\langle \mathcal{A}, \varsigma \rangle$, où $\mathcal{A}$ est l'ensemble d'attributs de l'exemple, c'est-à-dire un vecteur de variables instanciées, et ς la classe de l'exemple. Dans SPITI, l'algorithme UpdateTree repose sur l'algorithme ITI (Utgoff *et al.*, 1997) en utilisant χ^2 comme mesure d'information. Un nœud de décision est installé lorsque la valeur du χ^2 pour une variable donnée dépasse un seuil τ_{χ^2} (Degris *et al.*, 2006a).

Tout d'abord, chaque CPD $\mathsf{Tree}[\hat{P}_{X_i}^a]$ de l'action a est mise à jour pour chaque variable X_i en utilisant l'état courant comme ensemble d'attributs et la valeur de X_i' comme classe (étape 1). Ensuite, la fonction de récompense $\mathsf{Tree}[\hat{R}]$ est mise à jour en utilisant l'état courant et l'action comme exemple et la récompense observée comme classe (étape 2).

Input: un FMDP $\hat{\mathcal{F}}$, une observation $\langle s, a, s', r\rangle$ Output: $\emptyset$

1. Pour tout $X_i \in X$: UpdateTree($\mathsf{Tree}[\hat{P}^a_{X_i}], \{x_1, \ldots, x_n\}, x'_i$)
2. UpdateTree($\mathsf{Tree}[\hat{R}], \{x_1, \ldots, x_n, a\}, r$)

   avec $x_i \in s$, $x'_i \in s'$ et, $\hat{P}^a_{X_i}$ et $\hat{R}$ respectivement la CPD $P^a_{X_i}$ de la variable X_i pour l'action a et la fonction de récompense R de $\hat{\mathcal{F}}$.

FIG. 3 – L'algorithme Learn($\hat{\mathcal{F}}, \langle s, a, s', r\rangle$) dans SPITI.

SPITI utilise une version incrémentale de l'algorithme SVI (Boutilier *et al.*, 2000) durant la phase de planification pour mettre à jour l'ensemble $\{\mathsf{Tree}[Q^a_{t-1}], \forall a \in A\}$ des fonctions de valeur d'action. Les fonctions de transition, de récompense et de valeur sont représentées avec des arbres dans SPITI, comme dans SVI. Enfin, SPITI utilise une politique d'exploration ϵ-$greedy$.

3 Exploitation de la structure additive

SPITI se heurte à deux limitations pour exploiter la structure additive d'un problème d'apprentissage par renforcement. Tout d'abord, sa planification ne peut pas exploiter la structure additive d'un problème. Ensuite, il ne prend pas en compte une éventuelle décomposition additive de la fonction de récompense. Nous traitons ces deux points en décrivant deux nouvelles instances de SDYNA, à savoir ULP et UNATLP. ULP est similaire à SPITI mis à part le fait qu'il utilise une méthode de programmation linéaire. UNATLP est similaire à ULP, mais il utilise une décomposition différente de la fonction de transition du FMDP appris.

3.1 Planification et action

SDYNA peut agir sans disposer d'une représentation explicite de la politique. Cependant, il a besoin d'une représentation des fonctions de valeur d'action, pour choisir à tout instant la meilleure action. Nous notons $\mathsf{Rules}[F]$ la représentation sous forme de règles d'une fonction F. À partir de l'équation 1, le système détermine la meilleure action en calculant :

$$\text{Greedy}_{\mathcal{V}}(s) = \arg\max_a [\mathsf{Rules}[Q^{\mathcal{V}}_a](s)]$$

ULP et UNATLP utilisent $\text{Greedy}_{\mathcal{V}}(s)$ combinée avec une loi d'exploration ϵ-$greedy$, qui consiste à choisir une action aléatoire avec une probabilité ϵ et de l'action de meilleure valeur sinon.

L'algorithme FactoredALP utilise des règles plutôt que des arbres comme représentation structurée. Comme le montre la figure 1, à partir de $\mathsf{Tree}[P](X'|s)$, on peut construire une représentation à base de règles $\mathsf{Rules}[P](X'|s)$ en construisant l'ensemble de règles de probabilités tel que $\mathsf{Rules}[P](X'|s) = \{|c_i \wedge X' = x : p_i| \text{ tel que } x \in \text{Dom}(X), p_i(X' = x|s) \neq 0, \forall l_i \in \mathsf{Tree}[P](X'|s)\}$ où c_i est le contexte d'une feuille l_i et p_i la probabilité $P(X' = x|s)$ dans l_i.

A l'aide de ces conversions, on peut appliquer des méthodes de programmation linéaire au FMDP construit incrémentalement par SPITI. Nous pouvons donc définir l'algorithme de planification incrémentale qui apparaît sur la figure 4, capable d'exploiter la structure additive d'un problème, que ce soit avec ULP ou UNATLP. L'idée principale est d'éviter de résoudre le programme linéaire à chaque pas de temps en réutilisant la solution précédente du dernier programme linéaire généré par FactoredALP tant qu'elle est valable. Nous supposons que les fonctions de base sont connues a priori.

Input: $\hat{\mathcal{F}}_t$, $\mathsf{Rules}[\mathcal{V}_{t-1}]$
Output: $\mathsf{Rules}[\mathcal{V}_t]$, $\{\mathsf{Rules}[Q_a^{\mathcal{V}_t}], \forall a \in A\}$

1. Si (Structure de $\hat{\mathcal{F}}_t \neq$ Structure de $\hat{\mathcal{F}}_{t-1}$) alors :
   (a) lastModif $\leftarrow t$
   (b) Réinitialiser la solution du programme linéaire précédent
2. Si $((t -$ lastModif $> T_M)$ ou $(t -$ lastPlanning $< T_P))$ et $(t -$ lastPlanning $> T_{MIN})$ alors :
   (a) lastPlanning $\leftarrow t$
   (b) $\{\mathsf{Rules}[\mathcal{V}_t], \{\mathsf{Rules}[Q_a^{\mathcal{V}_t}], \forall a \in A\}\} \leftarrow$ FactoredALP($\hat{\mathcal{F}}_t$) en utilisant la solution du programme linéaire précédent s'il n'a pas été réinitialisé.

   sinon :

   $\{\mathsf{Rules}[\mathcal{V}_t], \{\mathsf{Rules}[Q_a^{\mathcal{V}_t}], \forall a \in A\}\} \leftarrow \{\mathsf{Rules}[\mathcal{V}_{t-1}], \{\mathsf{Rules}[Q_a^{\mathcal{V}_{t-1}}], \forall a \in A\}\}$
3. Renvoyer $\mathsf{Rules}[\mathcal{V}_t]$ et $\{\mathsf{Rules}[Q_a^{\mathcal{V}_t}], \forall a \in A\}$

FIG. 4 – L'algorithme Plan($\hat{\mathcal{F}}_t, \mathsf{Rules}[\mathcal{V}_{t-1}]$) dans ULP et UNATLP

3.2 Apprentissage de la structure

Nous traitons à présent le second point, à savoir le fait que SPITI n'utilise pas la décomposition additive de la fonction de récompense. Nous faisons à présent l'hypothèse que la récompense n'est plus un scalaire unique $r \in \mathbb{R}$, mais un vecteur $r = (r_1, \ldots, r_r) \in \mathbb{R}^r$ où chaque r_j est la récompense associée à une fonction localisée R_j, ce qui revient à supposer que cette décomposition est connue a priori, sans que soient connues la structure de chaque R_j. L'algorithme Learn($\hat{\mathcal{F}}, \langle s, a, s', r\rangle$), adapté à une telle décomposition de la fonction de récompense, est décrit dans la figure 5.

Input: un FMDP $\hat{\mathcal{F}}$, une observation $\langle s, a, s', r\rangle$ Output: $\emptyset$

1. Pour tout $X_i \in X$:
   UpdateTree($\mathsf{Tree}[\hat{P}^a_{X_i}], \{x_1, \ldots, x_n\}, x'_i$)
2. Pour tout $\hat{R}_j \in \hat{R}$:
   UpdateTree($\mathsf{Tree}[\hat{R}_j], \{x_1, \ldots, x_n, a\}, r_j$)

FIG. 5 – L'algorithme Learn($\hat{\mathcal{F}}, \langle s, a, s', r\rangle$) dans ULP.

La fonction de transition est mise à jour dans ULP comme dans SPITI (étape 1). Cependant, on met à jour un arbre $\mathsf{Tree}[\hat{R}_j]$ différent pour chaque fonction R_j dans $R(s) = \sum_{j=1}^{r} R_j(s)$, en utilisant l'état courant comme exemple et la récompense observée comme classe (étape 2).

SPITI et ULP utilisent un arbre $\mathsf{Tree}[\hat{P}^a_{X_i}]$ pour chaque variable et chaque action afin de représenter chaque CPD $\hat{P}^a_{X_i}$ dans un FMDP $\hat{\mathcal{F}}$. De même que pour la fonction de récompense, il est possible de représenter la fonction de transition avec seulement une CPD $\hat{P}_{X_i}$ par variable X_i du problème, en incluant l'action parmi les variables (Boutilier & Goldszmidt, 1996). L'algorithme Learn$(\hat{\mathcal{F}}, \langle s, a, s', r\rangle)$ utilisé par UNATLP qui apprend cette représentation apparaît sur la figure 6.

Input: un FMDP $\hat{\mathcal{F}}$, une observation $\langle s, a, s', r\rangle$ Output: $\emptyset$

1. Pour tout $X_i \in X$UpdateTree($\mathsf{Tree}[\hat{P}_{X_i}], \{x_1, \ldots, x_n, a\}, x'_i$)
2. Pour tout $\hat{R}_j \in \hat{R}$UpdateTree($\mathsf{Tree}[\hat{R}_j], \{x_1, \ldots, x_n, a\}, r_j$)

FIG. 6 – L'algorithme Learn$(\hat{\mathcal{F}}, \langle s, a, s', r\rangle)$ dans UNATLP. Il utilise un seul arbre $\mathsf{Tree}[\hat{P}_{X_i}]$ par variable, au lieu d'un arbre $\mathsf{Tree}[\hat{P}^a_{X_i}]$ par variable et par action dans SPITI et ULP.

La principale différence réside lors de l'étape 1 où chaque CPD $\mathsf{Tree}[\hat{P}_{X_i}]$ est mise à jour pour chaque variable X_i en utilisant l'état courant et l'action comme attributs de l'exemple et la valeur de X'_i comme classe (étape 1). Cette représentation de la fonction de transition n'a pas d'effet sur la méthode de planification. Si l'action n'est pas testée dans $\hat{P}_{X_i}$, $\hat{P}_{X_i} = \hat{P}^a_{X_i}$ pour tout a, sinon $\hat{P}^a_{X_i}$ est extrait de $\hat{P}_{X_i}$ en remplaçant chaque nœud de décision testant l'action dans $\hat{P}_{X_i}$ par le sous-arbre correspondant à l'action a.

4 Evaluation expérimentale

Dans cette section, nous présentons une évaluation empirique de ULP et UNATLP sur le problème SysAdmin défini par Guestrin *et al.* (2003), avec une topologie en anneau unidirectionnel composé de $N = 40$ machines. La taille du problème est $2^{40} \times 41 \approx 4 \cdot 10^{12}$ couples état/action. A chaque pas de temps, un administrateur système peut rebooter une machine, lui donnant une forte probabilité de fonctionner au pas de temps suivant. Il reçoit une récompense de 1 pour chaque machine en fonctionnement (sauf pour l'une des machines qui rapporte 2, afin de rendre le problème asymétrique). La fonction de valeur est composée de h_0 et par N fonctions de base h_i (définie dans la figure 7) correspondant à chaque machine i.

Nous avons utilisé le solveur de programmes linéaires `glpsol`[1] avec : $\epsilon = 0.1$ dans la politique d'exploration ϵ-$greedy$, un facteur d'actualisation $\gamma = 0.99$, un seuil $\tau_{\chi^2} = 30$ dans UpdateTree et $T_M = 100$, $T_P = 1500$ et $T_{MIN} = 50$ pour l'algorithme de planification (figure 4). Nous avons fait tourner 10 expériences de 20.000 pas de temps. Pour des raisons d'implémentation, nous utilisons un algorithme d'induction d'arbre similaire à ID4 (Schlimmer & Fisher, 1986) plutôt que ITI. Par ailleurs, les

[1] http ://www.gnu.org/sdetware/glpk/glpk.html

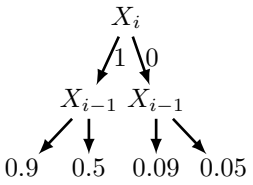


FIG. 7 – Fonction de base pour une variable X_i dans le problème SysAdmin.

$g_i^a(s)$ sont calculés en utilisant des arbres de décision (Boutilier *et al.*, 2000). Nous renvoyons aux articles cités pour une description complète de ces algorithmes.

De plus, nous utilisons un critère des moindres carrés (Breiman & Breiman, 1984) pour apprendre les fonctions de récompense dans l'algorithme UpdateTree. Enfin, nous ne comparons pas ULP et UNATLP à SPITI car celui-ci est inadéquat dans un cadre où les fonctions de récompense et de valeur font apparaître une structure additive (Boutilier *et al.*, 2000; Hoey *et al.*, 1999; Guestrin *et al.*, 2003). En revanche, nous indiquons la performance de deux agents, appelés RANDOM et OPTIMAL, qui exécutent à chaque pas de temps, respectivement, une action aléatoire et l'action optimale, calculée hors-ligne en utilisant FactoredALP avec les fonctions de base de la figure 7.

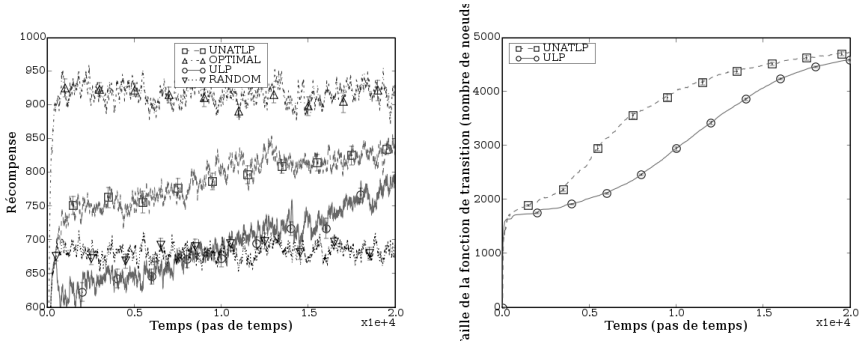


FIG. 8 – (a) Performances sur le problème SysAdmin. ULP et UNATLP améliorent leur politique malgré la taille du problème et le nombre de couples état/action visités. (b) Taille de la fonction de transition du FMDP appris par ULP et UNATLP.

La figure 8(a) montre la performance de l'agent, définie par $R_t^{disc} = r_t + \gamma R_{t-1}^{disc}$ où r_t est la récompense immédiate obtenue par chaque agent. ULP et UNATLP améliorent substantiellement leur politique par rapport à RANDOM. De plus, UNATLP progresse plus vite que ULP. La figure 8(b) compare les tailles des différentes fonctions de transition apprises par ULP et UNATLP. Nous observons que, pour une taille de fonction de transition comparable, UNATLP est plus performant que ULP. De plus, la fonction de transition construite par UNATLP croît plus vite que celle construite par ULP.

5 Discussion

Nos résultats montrent que ULP et UNATLP sont capables de construire rapidement un FMDP représentant le problème d'apprentissage par renforcement à résoudre. De plus, les méthodes de planification basées sur la programmation linéaire qui exploitent la structure additive des problèmes démontrent leur efficacité dans SDYNA pour traiter des problèmes dont la structure est inconnue a priori. En effet, ULP et UNATLP traitent des problèmes de très grande taille en exploitant la propriété de généralisation des algorithmes d'induction d'arbres de décision. Cela est confirmé par le fait, que ULP et UNATLP améliorent sensiblement leur politique après avoir visité moins de $5 \cdot 10^{-8}\%$ des couples état/action existants. On retrouve là une propriété déjà mise en évidence pour SPITI (Degris *et al.*, 2006a,b).

Par ailleurs, nos résultats illustrent la différence entre les représentations de la fonction de transition dans ULP et dans UNATLP. La figure 8 (a) montre que UNATLP apprend plus vite que ULP. Cela s'explique principalement par le fait que dans ULP, les nouveaux exemples ne mettent à jour que la CPD de la dernière action, tandis qu'ils participent à la mise à jour de toute la fonction de transition dans UNATLP. De plus, Boutilier & Goldszmidt (1996) suggèrent que la seconde représentation est plus compacte quand la valeur d'une variable persiste quelle que soit l'action, or ce cas ne se présente pas dans le problème que nous avons étudié.

Enfin, l'algorithme $\text{Plan}(\hat{\mathcal{F}}_t, \mathsf{Rules}[\mathcal{V}_{t-1}])$ dans ULP et UNATLP est une adaptation incrémentale de l'algorithme FactoredALP encore très perfectible, qui repose sur des heuristiques naïves pour déterminer quand on peut améliorer la politique. Des progrès significatifs pourront être accomplis en améliorant ce point. De façon plus générale, ni ULP ni UNATLP n'atteignent la performance optimale. Des améliorations importantes sont à rechercher du côté de l'emploi de politiques d'exploration dirigée (Guestrin *et al.*, 2002; Strehl, 2007) plutôt qu'aléatoire.

6 Conclusion

Nous avons décrit ULP et UNATLP, deux nouvelles instances de SDYNA. Ces deux instances utilisent des méthodes de planification reposant sur la programmation linéaire pour exploiter la structure additive des problèmes. Notre première contribution consiste à avoir montré que ces méthodes peuvent être combinées avec l'induction d'arbres de décision pour traiter des problèmes d'apprentissage par renforcement de très grande taille dont la structure est inconnue. Notre seconde contribution est un nouvel algorithme d'apprentissage plus rapide, en particulier sur les problèmes très grands, sans perdre sur la taille de la fonction de transition. Enfin, ULP et UNATLP sont capables de traiter des problèmes précédemment hors de portée.

Remerciements

Nous remercions Christophe Marsala et Ron Parr pour des discussions fructueuses.

Références

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), 49–107.

BOUTILIER C. & GOLDSZMIDT M. (1996). The Frame Problem and Bayesian Network Action Representations. In *Proceedings of the Eleventh Biennial Canadian Conference on Artificial Intelligence*, p. 69–83.

BREIMAN B. & BREIMAN L. (1984). *Classification and Regression Trees*. Chapman & Hall/CRC.

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

DEGRIS T., SIGAUD O. & WUILLEMIN P. (2006a). Chi-square Tests Driven Method for Learning the Structure of Factored MDPs. In *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence*, p. 122–129.

DEGRIS T., SIGAUD O. & WUILLEMIN P. (2006b). Learning the Structure of Factored Markov Decision Processes in Reinforcement Learning Problems. In *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, p. 257–264, Pittsburgh, Pennsylvania, USA.

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. & SCHUURMANS D. (2002). Algorithm-Directed Exploration for Model-Based Reinforcement Learning in Factored MDPs. In *Proceedings of the 19th International Conference on Machine Learning*, p. 235–242.

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 : Morgan Kaufmann.

KOLLER D. & PARR R. (1999). Computing Factored Value Functions for Policies in Structured MDPs. In *Proceedings 16th International Joint Conference on Artificial Intelligence*, p. 1332–1339.

MANNE A. S. (1960). *Linear Programming and Sequential Decisions*. Cowles Foundation for Research in Economics at Yale University.

SCHLIMMER J. & FISHER D. (1986). A Case Study of Incremental Concept Induction. *Proceedings of the Fifth National Conference on Artificial Intelligence*, p. 496–501.

SCHWEITZER P. & SEIDMANN A. (1985). Generalized Polynomial Approximations in Markovian Decision Processes. *Journal of Mathematical Analysis and Applications*, **110**, 568–582.

STREHL A. (2007). Model-Based Reinforcement Learning in Factored MDPs. In *Proceedings of the IEEE Symposium on Approximate Dynamic Programming, to appear*.

UTGOFF P. E., NERKMAN N. C. & CLOUSE J. A. (1997). Decision Tree Induction Based on Efficient Tree Restructuring. *Machine Learning*, **29**(1), 5–44.

ZHANG T. & POOLE D. (1999). On the Role of Context-specific Independence in Probabilistic Reasoning. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, p. 1288–1293.