Planification et apprentissage par renforcement avec modèles d'actions compacts

Boris Lesner

université de Caen Basse-Normandie

UNIVERSITÉ de CAEN BASSE-NORMANDIE

U.F.R. : Sciences

ÉCOLE DOCTORALE : SIMEM

THÈSE

présentée par

Boris Lesner

et soutenue

le 8 décembre 2011

en vue de l'obtention du

DOCTORAT de l'UNIVERSITÉ de CAEN

spécialité : Informatique et applications

(Arrêté du 7 août 2006)

Planification et apprentissage par renforcement avec modèles d'actions compacts

MEMBRES du JURY

Yann Chevaleyre	Professeur	U. Paris-Nord	(rapporteur)
Olivier Sigaud	Professeur	U. Pierre et Marie Curie	(rapporteur)
Olivier Buffet	Chargé de Recherche	INRIA Nancy	
Abdel-Illah Mouaddib	Professeur	U. Caen Basse-Normandie	
Bruno Scherrer	Chargé de Recherche	INRIA Nancy	
Bruno Zanuttini	Maître de Conférences HDR	U. Caen Basse-Normandie	(directeur)

Mis en page avec la classe thloria.

Remerciements

Tout d'abord un grand merci à Bruno Zanuttini, pour son encadrement et son soutien sans faille tout au long de ces trois années. Ce travail de thèse est avant tout le fruit d'une collaboration et n'aurait pas été le même sans sa participation. Merci Bruno d'avoir partagé avec moi ta passion pour la recherche, ta détermination et ton savoir-faire.

Merci aussi à Abdel-Illah Mouaddib et François Bourdon mes premiers directeurs de thèse. Je remercie tout particulièrement François pour m'avoir donné l'envie de continuer mes études en informatique et ce, dès ma première année d'IUT où il m'a parlé de systèmes multi-agents.

Je remercie les rapporteurs de cette thèse, Yann Chevalèyre et Olivier Sigaud pour leur investissement et les discussions que nous avons pu avoir. Merci également à Olivier Buffet et Bruno Scherrer pour avoir accepté de faire partie de mon jury.

Une dédicace à Lamia et Arnaud avec qui j'ai partagé le bureau 362 pendant les trois années de ma thèse, sans oublier Nicolas (aka. "Prospect") et Guillaume qui se sont joints à nous pour mettre encore un peu plus d'ambiance dans le bureau. Merci aussi aux autres doctorants/postdocs/MC du labo : Abir, Benoît, Cyril, Grégory, JP, Laetitia, Mathieu, Romain, et aux « anciens » : Hugo, Léo, Matthieu et Simon.

Merci à vous Wafa, Olivier et Lucile pour votre soutien et votre écoute. Merci et bravo à Julien pour m'avoir supporté comme colocataire pendant ces trois ans.

Et enfin, mes derniers remerciements - et pas les moindres - vont à mes parents et à ma petite soeur Morgane.

Table des matières

Introduction générale

Motivations

Qu'il s'agisse d'un robot autonome mis en place dans un environnement inconnu, d'une industrie voulant optimiser sa consommation de ressources et sa production ou même d'un adversaire ordinateur dans un jeu vidéo, tous ces scénarios doivent résoudre le problème de la prise de décision face à l'incertitude d'un environnement. Pour chacun de ces systèmes, il y a plusieurs critères de réussite à maximiser comme la réussite de la mission pour le robot ou l'augmentation de la productivité pour l'industrie. En parallèle il y a aussi des coûts à minimiser, car chaque décision nécessite des ressources pour être exécutée. Toutes ces décisions doivent être prises dans un environnement qui évolue de manière incertaine au fil du temps, que ce soit naturellement ou en conséquence des actions entreprises.

De façon plus générale, l'environnement est décrit sous la forme *d'états*, chaque état décrivant une situation particulière de l'environnement. Un état est une configuration particulière des paramètres décrivant l'environnement. Ainsi pour un robot explorateur ces paramètres peuvent être sa position, son niveau d'énergie, la température extérieure, etc. L'entité prenant des décisions est un *agent*. À chaque instant, au vu de son état courant, l'agent doit choisir une *action* ayant un certain coût mais qui produit aussi une *récompense* qui modélise le degré d'adéquation de cette action à la réussite de ses objectifs. L'incertitude est reflétée par le comportement des actions : pour un état donné, plusieurs exécutions d'une même action peuvent avoir des issues différentes. Dans ce contexte, il faut donc arriver à prendre une « bonne » décision non seulement pour l'instant présent mais aussi arriver à *anticiper* l'incertitude des instants à venir.

La description de tels problèmes rentre dans le cadre formel des *Processus de Décision Markoviens* ou PDM. Dans ce cadre, on décrit l'ensemble des états du système, des actions de l'agent, et des récompenses et coûts induits par ces actions. L'incertitude est représentée de manière probabiliste : on suppose que l'on connaît la probabilité d'arriver dans chaque état en exécutant une action. Si l'on possède une telle description, il est alors possible de calculer des bonnes stratégies de décision qui maximisent l'espérance de récompense sur le long terme, on parle alors de *planification*. Malheureusement, décrire exhaustivement chacun des états et le comportement de chaque action pour ces états est très difficile car le nombre d'états augmente très rapidement en fonction du nombre de paramètres utilisés pour les décrire. Si on désire modéliser un problème concret possédant une trentaine de paramètres, il est déjà quasi-impossible

de stocker sa description dans un ordinateur, sans parler du temps nécessaire pour résoudre le problème de planification.

Cependant, les deux dernières décennies ont vu apparaître les représentations structurées de ces environnements ainsi que diverses méthodes exploitant cette structure pour résoudre plus efficacement les problèmes de planification associés. Ces représentations se basent sur le fait que la dynamique d'un environnement possède le plus souvent une certaine *structure*, des régularités qui permettent une *représentation compacte*. L'observation fondatrice de ces approches est que le comportement de chaque action est indépendant de certains paramètres de l'état, ce qui fait que l'on peut regrouper les états ayant un comportement identique et les traiter comme un tout sans les considérer individuellement. Ces techniques ont permis de traiter des problèmes plus complexes tout en restant à la portée des moyens de calcul actuels.

Les approches ci-dessus, structurées ou non, possèdent néanmoins une importante limitation. Elles supposent en effet que l'on connaît a priori la dynamique de l'environnement, voire même sa structure. Prenons l'exemple d'un robot explorateur sur Mars. Comment être certain d'une assertion telle que « La probabilité de s'enliser dans le sable en avançant est de 0,2. » ? Car il s'agit exactement ce qui est fait lors de la modélisation du problème, alors qu'il n'existe aucune validation empirique de cette valeur, c'est tout au mieux un choix éclairé. Dans une telle situation, quelle confiance accorder à une stratégie de décision dérivée d'un tel modèle, qui est certainement imprécis ? Une réponse à ce problème est donnée par *l'apprentissage par renforcement*. Plutôt que de se baser sur un modèle imprécis, l'idée de l'apprentissage est de mettre l'agent directement en situation et de le laisser découvrir (apprendre) la dynamique de l'environnement pour obtenir par la suite un comportement adapté et sans a priori sur celui-ci. La difficulté centrale à ce problème est de décider si l'agent doit arrêter d'apprendre et *exploiter* ses connaissances, au risque d'utiliser un modèle incomplet, ou au contraire *d'explorer* son environnement pour collecter plus d'informations, avec cette fois pour risque de perdre du temps et de prendre de très mauvaises décisions.

Pour l'apprentissage par renforcement, le problème du grand nombre d'états à considérer est toujours présent. Cependant, il existe très certainement une structure inconnue qui là aussi peut être exploitée pour rendre traitable le problème de décision considéré. C'est pourquoi il est intéressant de considérer à la fois le problème d'apprentissage de la structure de l'environnement ainsi que sa dynamique pour arriver à déterminer un bon comportement pour l'agent. Toutefois, dans un cadre d'apprentissage, l'agent ne dispose que de très peu d'informations initiales. Il lui faut donc un certain temps d'exploration pour acquérir une idée précise de son environnement. C'est pourquoi, en plus des mesures de performances en termes de ressources de calcul nécessaires, un algorithme pour agent apprenant doit aussi minimiser une *complexité d'apprentissage* qui mesure ce temps d'exploration.

Cette thèse s'articule exclusivement autour de ces modèles structurés, à la fois pour les problèmes de planification et d'apprentissage. Nous proposons en effet pour chacun de ces problèmes différentes approches aux caractéristiques très différentes.

Objectifs

Il existe deux grandes familles de description structurées pour les PDM. Une première, les Réseaux Bayésiens Dynamiques, a fait l'objet de nombreux travaux que ce soit pour les problèmes de planification et d'apprentissage. Dans cette thèse, nous traitons plutôt une autre représentation, moins répandue : les *opérateurs STRIPS probabilistes* et PDDL probabiliste, à la formulation plus naturelle mais certainement moins adaptée pour le calcul de stratégies de décision.

Pour la partie planification, notre but est de proposer des algorithmes prenant en entrée une représentation compacte de l'environnement et de produire une stratégie de décision elle aussi compacte. L'enjeu est de maintenir, tout au long du processus de planification, un modèle compact des données intermédiaires générées par les algorithmes. De cette façon, la structure est exploitée pour réduire les temps de calcul et l'espace requis.

En ce qui concerne l'apprentissage, notre objectif est de proposer des méthodes capables d'apprendre à la fois la structure et la dynamique d'un modèle d'actions, si possible avec certaines garanties théoriques sur la complexité d'apprentissage. Pour ce problème, nous devons faire face à l'ambiguïté des observations, qui introduit de l'incertitude quant à la pertinence de l'information collectée par l'agent lors de l'apprentissage.

Contributions

Pour la planification nous proposons deux méthodes. Une première, basée sur les outils de la logique propositionnelle, est capable de résoudre les problèmes de manière approchée. Ensuite, nous proposons une méthode exacte pour traiter des problèmes décrits dans un langage d'actions très riche, le PDDL probabiliste, via l'introduction d'une notion étendue de fonction de valeur d'action. Ces fonctions de valeur « *frameless* » permettent de manipuler directement la description des actions pour calculer leur valeur de manière incrémentale.

En apprentissage, nous proposons aussi deux approches. Une première est basée sur des heuristiques, calculées par programmation linéaire, pour découvrir les effets des actions et en déduire ensuite le comportement à adopter. Notre seconde contribution, plus théorique, bien qu'évaluée empiriquement, se place dans le récent cadre d'apprentissage KWIK qui propose des solutions élégantes pour construire et analyser des algorithmes d'apprentissage par renforcement. Nous identifions certaines propriétés clés sur l'environnement et en dérivons ensuite des algorithmes d'apprentissage pour la structure et la dynamique de l'environnement. Pour ceux-ci, nous avons mené une étude théorique sur le temps d'apprentissage nécessaire à l'obtention d'un modèle précis et montré que, sous certaines conditions, celui-ci est polynomial en le nombre de paramètres décrivant le problème.

Pour toutes nos contributions dans ces deux domaines, nous avons mené des évaluations empiriques sur des problèmes issus de la littérature. Bien qu'il ne s'agisse pas de problèmes

applicatifs réels, ils permettent d'étudier le comportement de nos approches afin d'identifier leurs avantages et inconvénients vis à vis des techniques existantes.

Plan du mémoire

Dans une première partie nous présentons les PDM et leurs représentations compactes (chapitres 1,2 et 3). Ensuite une seconde partie traitera de la planification avec ces représentations : après avoir décrit certaines méthodes existantes nous présentons nos travaux sur l'approximation de PDM (chapitre 5) pour terminer par l'algorithme RBAB pour la résolution de PDM décrits en PDDL probabiliste (chapitre 6). Enfin, la troisième et dernière partie traitera de l'apprentissage. Nous commencerons (chapitre 7) par une présentation des approches existantes ainsi que du cadre formel dans lequel nous travaillerons. Ensuite nous présentons nos deux approches d'apprentissage par renforcement avec un modèle STRIPS probabiliste : un première utilisant la programmation linéaire (chapitre 8) et une seconde dans le cadre KWIK (chapitre 9). Finalement (chapitre 10) nous évaluerons expérimentalement ces deux algorithmes et les comparons au cas de l'apprentissage pour les PDM énumérés. Finalement nous discutons et ouvrons des perspectives pour la suite de ces travaux. En annexes se trouvent quelques lemmes et résultats utiles ainsi qu'une table répertoriant les notations utilisées et leur signification.

Première partie

Représentations

Chapitre 1

Processus de décision markoviens

Sommaire

Ce chapitre présente la notion de processus de décision markovien et les algorithmes d'optimisation associés. Nous introduisons les notations et les propriétés importantes qui seront utilisées tout au long de ce manuscrit.

1.1 Formulation

Dans le cadre d'un processus de décision markovien (PDM), un agent agit sur son environnement via des actions. Le temps est décomposé en instants discrets. À chacun de ces instants, l'agent se situe dans un état particulier et doit effectuer une action. Effectuer une action change l'état de l'agent (transition) et celui-ci reçoit une récompense. Les transitions sont stochastiques, pour un état et une action donnés, l'état résultant est déterminé selon une loi de probabilité qui modélise son incertitude

Définition 1 (processus de décision markovien) *Un processus de décision markovien fini M est composé des éléments suivants :*

- *$\mathcal{S}$ est l'ensemble fini des états possibles du système ;*
- *$\mathcal{A}$ est l'ensemble fini des actions de l'agent ;*
- *$P(\cdot|s,a)$, représente la* fonction de transition *; ainsi $P(s'|s,a)$ représente la probabilité de passer de l'état s à l'état s' en exécutant l'action a ;*

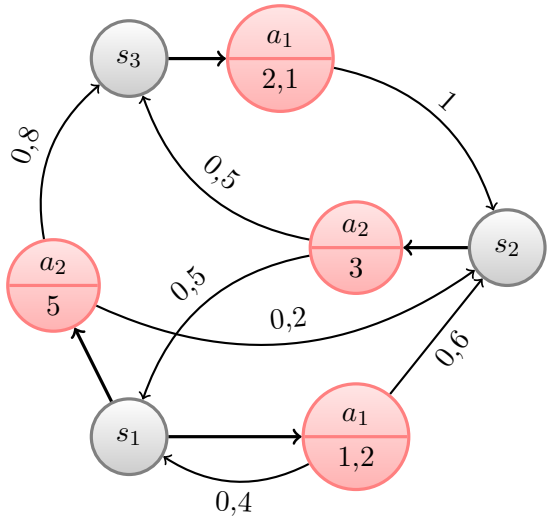


FIGURE 1.1 – Un exemple de PDM à 3 états et 2 actions.

– $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ *représente la* fonction de récompense *; pour un état s et une action a, $R(s,a)$ représente la récompense associée à l'exécution de a dans l'état s.*

À titre d'illustration, un exemple de PDM fini est donné sur la figure 1.1. Il existe de nombreuses autres définitions des PDM, notamment quand les espaces d'états et d'actions sont infinis, ou quand la fonction de transition est non-stationnaire (dépendante de l'historique). Cependant, au cours de cette thèse nous ne traiterons que des PDM correspondant à la définition 1. Une caractérisation exhaustive des possibilités du modèle PDM est donnée dans [Puterman, 1994].

1.1.1 Politiques

Un élément central aux PDM est la notion de *politique*. Une politique π associe à chaque état du système une action à exécuter par l'agent. Pour un état s, nous notons $\pi(s)$ pour l'action à exécuter dans s. Nous n'abordons que ces politiques particulières dites déterministes et stationnaires. D'une manière plus générale, il est possible de définir des politiques plus riches (stochastiques et/ou non-stationnaires).

Pour un PDM M, une politique π et un état s_0 induisent une distribution de probabilités sur des *trajectoires* : des séquences $(s_0,s_1,\dots)$ d'états traversés par l'agent en exécutant π à partir de s_0 :

$$\Pr(s_0,s_1,\dots|\pi,M,s_0) = \prod_{t=1}^{\infty} P(s_t|s_{t-1},\pi(s_{t-1}))$$

Tout au long de ce document, et lorsqu'il n'y a pas d'ambiguïté nous notons s pour un état ($s \in \mathcal{S}$) et a pour une action ($a \in \mathcal{A}$).

1.1.2 Critère d'optimalité

Valeur d'une politique

Étant donné un PDM M, une politique π, et un état s_0, l'exécution de π à partir de s_0 génère une séquence de récompenses aléatoires $r_t, t = 0, \ldots, \infty$. L'*utilité totale pondérée* est définie comme la variable aléatoire :

$$U_\pi(T) = \sum_{t=0}^{T} \gamma^t r_t$$

Intuitivement, ce gain représente la somme pondérée des récompenses obtenues par l'agent lorsqu'il exécute la politique π depuis s_0 pendant T pas de temps. Le facteur $\gamma \in [0,1[$ a pour fonction de pondérer l'importance des récompenses futures. Ainsi, à l'instant t, la récompense r_t obtenue sera pondérée par γ^t.

Plus généralement, on évalue une politique en termes *d'espérance* d'utilité. Ainsi lorsque l'agent agit indéfiniment, on parle alors de la *fonction de valeur* de π, notée V^π. La valeur $V^\pi(s)$ correspond à l'espérance d'utilité que l'agent peut obtenir en exécutant la politique π à partir de l'état s :

$$V^\pi(s) = \mathbf{E}\left[U_\pi(\infty) | s_0 = s\right] = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 = s\right]$$

De manière similaire on peut définir la *fonction de valeur d'action* $Q^\pi(s,a)$ qui correspond à l'utilité espérée d'exécuter a dans l'état s puis de suivre indéfiniment π dans les états suivants :

$$Q^\pi(s,a) = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 = s, a_0 = a\right]$$

Les deux fonctions sont liées par l'égalité :

$$V^\pi(s) = Q^\pi(s,\pi(s))$$

Notons aussi que, partant de l'identité $\sum_{t=0}^{\infty} \gamma^t = \frac{1}{1-\gamma}$, nous obtenons la borne suivante sur les fonctions de valeur pour tout état s et politique π :

$$V^\pi(s) \leq \frac{R_{\max}}{1-\gamma} \tag{1.1}$$

avec $R_{\max} = \max_{s,a} R(s,a)$.

Fonctions de valeur et politiques optimales

Étant donné un PDM M, le but d'un agent est de suivre une politique qui maximise l'espérance d'utilité. Il existe toujours [Puterman, 1994] une telle *politique optimale*, notée π^* qui soit déterministe et stationnaire. Les fonctions de valeur et de valeur d'action de telles politiques,

notées respectivement V^* et Q^* sont définies par

$$V^*(s) = \max_{\pi \in \Pi} V^\pi(s)$$
$$Q^*(s,a) = \max_{\pi \in \Pi} Q^\pi(s,a)$$

avec Π l'ensemble des politiques possibles pour M.

Pour une fonction de valeur d'action Q, on définit la politique *gloutonne* π_Q comme celle qui choisit les actions maximisant l'espérance d'utilité :

$$\pi_Q(s) \in \arg\max_{a \in \mathcal{A}} Q(s,a)$$

En particulier la politique π^* est gloutonne pour Q^*.

Calcul de politiques et équations d'optimalité

Le calcul des politiques et des fonctions de valeur optimales peut se faire grâce aux *équations de Bellman* qui donnent les bases de la programmation dynamique [Bellman, 1957].

Définition 2 (opérateur de Bellman) *Pour une action a on note $\mathfrak{B}^a$ l'opérateur de Bellman qui, étant donné une fonction de valeur $V : \mathcal{S} \to \mathbb{R}$, calcule une fonction $\mathfrak{B}^a V$ telle que :*

$$(\mathfrak{B}^a V)(s) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V(s')$$

Similairement, on définit l'opérateur $\mathfrak{B}$ tel que :

$$(\mathfrak{B} V)(s) = \max_{a \in \mathcal{A}} (\mathfrak{B}^a V)(s)$$

L'opérateur de Bellman est *contractant*, ainsi pour deux fonctions de valeur V_1 et V_2, $\gamma < 1$ implique :

$$\|\mathfrak{B} V_1 - \mathfrak{B} V_2\|_\infty \leq \gamma \|V_1 - V_2\|_\infty$$

Cette propriété de contraction garantit que l'application répétée de $\mathfrak{B}$ sur une fonction de valeur V quelconque converge vers un point fixe qui correspond à V^* :

$$\begin{aligned} V^* &= \mathfrak{B} V^* \\ &= \max_{a \in \mathcal{A}} R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s') \end{aligned} \tag{1.2}$$

L'opérateur $\mathfrak{B}$ permet de définir les politiques gloutonnes pour les fonctions de valeurs ; étant donné V, la politique gloutonne π_V est définie comme :

$$\pi_V(s) \in \arg\max_{a \in \mathcal{A}} (\mathfrak{B}^a V)(s)$$

1.2 Calcul de fonctions de valeur et de politiques

1.2.1 Programmation dynamique

Les algorithmes de programmation dynamique pour la résolution de PDM calculent les fonctions de valeur itérativement, par approximations successives. À chaque itération $t = 0,1,\ldots$ de l'algorithme, on calcule la fonction de valeur V^t à t pas de temps restants. Ainsi V^0 représente la valeur espérée quand l'agent n'a plus d'action à effectuer, V^1 quand il lui reste une action, etc. Ainsi étant donné V^t, la valeur $V^{t+1}(s)$ d'un état s est donnée par la relation de récurrence dérivée de l'équation (1.2) :

$$V^{t+1} = \mathfrak{B}\, V^t = \max_{a \in \mathcal{A}} R(s,a) + \gamma \sum_{s' \in \mathcal{S}} V^t(s')$$

Avec $V^0(s) = 0$ pour tout état s, en effet l'agent n'obtiendra aucune récompense quand il n'a plus d'action à effectuer.

La propriété de contraction de l'opérateur $\mathfrak{B}$ garantit alors :

$$\lim_{t \to \infty} V^t(s) = V^*(s)$$

Dans le cas de l'horizon infini, une des propriétés fondamentales de cette série est que lorsque deux valeurs V^t et V^{t+1} sont proches, au sens de la norme infinie, alors V^{t+1} est proche de V^*. Formellement [Williams et Baird, 1994], pour tout $\epsilon > 0$:

$$\left\| V^t - V^{t+1} \right\|_\infty \leq \frac{\epsilon(1-\gamma)}{2\gamma} \quad \Longrightarrow \quad \left\| V^* - V^{t+1} \right\|_\infty \leq \epsilon$$

On dit alors que V^{t+1} est ϵ-optimale. La politique gloutonne $\pi_{V^{t+1}}$ est quand à elle $\frac{2\gamma\epsilon}{1-\gamma}$ optimale [Singh, 1994] :

$$\left\| V^* - V_{\pi_V^{t+1}} \right\|_\infty \leq \frac{2\gamma\epsilon}{1-\gamma}$$

Cette convergence approchée est obtenue après un nombre d'itérations H polynomial en ϵ , $R_{\max}$ et $\frac{1}{1-\gamma}$ [Littman *et al.*, 1995], il suffit que :

$$H \geq \frac{1}{1-\gamma} \ln \frac{R_{\max}}{\epsilon(1-\gamma)}$$

Nous présentons ici les deux algorithmes les plus courants pour calculer une politique et sa fonction de valeur ϵ-optimales pour un $\epsilon > 0$ donné. Leur complexité est polynomiale en $|\mathcal{S}|, |\mathcal{A}|, \frac{1}{\epsilon}, \frac{1}{1-\gamma}$ [Littman *et al.*, 1995].

Itération de valeur

L'algorithme d'itération de valeur (Algorithme 1) transforme une fonction de valeur initiale arbitraire V^0 par applications successives de l'opérateur de Bellman en une fonction de valeur V^t ϵ-optimale. Notons que le critère d'arrêt de l'itération de valeur correspond à une convergence approximative entre V^t et V^{t+1}. Bien que calculer V^H avec H défini comme ci-dessus soit possible, en pratique la convergence arrive bien avant ces H itérations.

Algorithme 1 : Itération de valeur

Entrées : Un PDM M et une fonction V^0 arbitraire

Sorties : Une politique π et une fonction de valeur V^t ϵ-optimales

$t \leftarrow 0$

répéter

  pour chaque *état* $s \in \mathcal{S}$ **faire**

    pour chaque *action* $a \in \mathcal{A}$ **faire**

      $Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^t(s')$

    $V^{t+1}(s) \leftarrow \max_{a \in \mathcal{A}} Q(s,a)$

  $t \leftarrow t + 1$

jusqu'à $\left\|V^t - V^{t-1}\right\|_\infty \leq \frac{\epsilon(1-\gamma)}{2\gamma}$

pour chaque *état* $s \in \mathcal{S}$ **faire**

  $\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}} Q(s,a)$

Itération de politique

Au lieu d'explorer l'espace des fonctions de valeur, l'algorithme *d'itération de politique* [Howard, 1960] (algorithme 2) recherche une politique ϵ-optimale en parcourant l'espace des politiques. Cet algorithme démarre avec une politique arbitraire π, puis l'évalue pour en déduire sa fonction de valeur V^π. Il calcule ensuite une politique π' gloutonne pour V^π. Ces deux étapes d'évaluation et de calcul d'une politique gloutonne sont répétées jusqu'à ce que la politique converge : $\pi' = \pi$.

De manière générale, l'itération de politique requiert moins d'itérations que l'itération de valeur. Cependant, l'étape d'évaluation de la politique (le calcul de V^π) est coûteux en temps de calcul.

1.2.2 Programmation linéaire

Une alternative à la programmation dynamique pour calculer la fonction de valeur optimale d'un PDM consiste à résoudre directement l'équation (1.2) par un programme linéaire :
Variables :

$$V(s), \forall s \in \mathcal{S}$$

Minimiser :

$$\sum_{s \in \mathcal{S}} \alpha(s) V(s)$$

Sous les contraintes : pour tout $s \in \mathcal{S}$ et $a \in \mathcal{A}$

$$V(s) - \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V(s') \geq R(s,a)$$

avec des coefficients $\alpha(s) > 0$ quelconques. Une fois V calculée, la politique gloutonne π_V est optimale pour ce PDM. En pratique il se peut cependant que cette politique ne soit pas tout à

Algorithme 2 : Itération de politique

Entrées : Un PDM M et une politique π arbitraire

Sorties : Une politique π et sa fonction de valeur V_π ϵ-optimale

répéter

`/* Évaluation de` π `*/`

$V_\pi \leftarrow 0$

répéter

$V'_\pi \leftarrow V_\pi$

pour chaque *état* $s \in \mathcal{S}$ **faire**

$V_\pi(s) \leftarrow R(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,\pi(s)) V'_\pi(s')$

jusqu'à $\|V'_\pi - V_\pi\|_\infty \leq \frac{\epsilon(1-\gamma)}{2\gamma}$

`/* Amélioration de` π `*/`

$\pi' \leftarrow \pi$

pour chaque *état* $s \in \mathcal{S}$ **faire**

$\mathcal{A}^* \leftarrow \arg\max_{a \in \mathcal{A}} R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_\pi(s')$

si $\pi(s) \notin \mathcal{A}^*$ **alors**

choisir $a \in \mathcal{A}^*$ arbitrairement

$\pi(s) \leftarrow a$

jusqu'à $\pi = \pi'$

fait optimale à cause des erreurs de précision numérique inhérentes aux solveurs de programmes linéaires. Notons que l'on peut utiliser la programmation linéaire pour l'évaluation de politique dans l'algorithme d'itération de politique.

1.3 Calcul de politiques approchées

Les algorithmes présentés ci-dessus sont des algorithmes exacts. Cependant, ils sont de complexité quadratique en taille de l'espace d'états ce qui est impraticable pour de grands problèmes. Pour faire face à cette difficulté, on peut avoir recours aux méthodes approchées qui utilisent des approximations des fonctions de valeur pour gagner en efficacité.

L'approche classique pour l'approximation de valeur est une décomposition sous la forme d'une *combinaison linéaire de fonctions de base*. À partir d'un ensemble de fonctions $H = \{h_1, \ldots, h_m\}$ telles que $h_i : \mathcal{S} \to \mathbb{R}$ on peut représenter une fonction de valeur sous la forme

$$\hat{V}(s) = \sum_{i=1}^{m} w_i h_i(s)$$

où $\mathbf{w} = (w_1, \ldots, w_m)^T$ est un vecteur de coefficients réels. En représentant H sous la forme d'une matrice à m lignes et $|\mathcal{S}|$ colonnes on peut écrire $\hat{V}$ sous forme vectorielle :

$$\hat{V} = H\mathbf{w}$$

Il est évident que pour un ensemble de fonctions de base H donné, toute fonction de valeur n'est pas représentable, car H définit un sous-espace $\mathcal{H} \subseteq \mathbb{R}^{|\mathcal{S}|}$. Avec une telle représentation, le problème de la résolution approchée revient alors à optimiser le vecteur $\mathbf{w}$ des coefficients de façon à obtenir une bonne, voire la meilleure, approximation de la fonction de valeur optimale V^* selon une norme $\|\cdot\|$ donnée. Ainsi la meilleure approximation d'une fonction V selon H est

$$\hat{V} = \min_{\mathbf{w}} \|H\mathbf{w} - V\|$$

On peut donc adapter l'itération de valeur avec cette représentation, à chaque itération on optimisera le vecteur $\mathbf{w}$ plutôt que V directement. L'algorithme 3 montre l'itération de valeur approchée. On remarque qu'il travaille uniquement à horizon fini T, car selon la norme utilisée il n'est pas garanti qu'un point fixe existe, même en appliquant un facteur de pondération $\gamma < 1$. Telle qu'elle est décrite, l'itération de valeur approchée doit tout de même parcourir l'ensemble d'états pour calculer V et $\mathbf{w}^t$. Cependant, on peut se restreindre à un petit nombre d'états seulement pour simplifier le calcul. Dans ce cas, on introduit une approximation supplémentaire mais on obtient toujours un vecteur de coefficients utilisable pour évaluer n'importe quel état.

Algorithme 3 : Itération de valeur approchée.

$\mathbf{w^0} \leftarrow \vec{0}$
pour $t = 1, \ldots, T$ **faire**
  Calculer $V = \mathfrak{B}(H\mathbf{w}^{t-1})$
  $\mathbf{w}^t = \arg\min_{\mathbf{w}} \|H\mathbf{w} - V\|$
retourner Une politique gloutonne pour $H\mathbf{w}^T$

Tout comme l'itération de valeur possède son équivalent approché, il existe aussi des versions approchées de la résolution par programmation linéaire qui ont suscité beaucoup d'intérêt dans la littérature [de Farias, 2002, de Farias et Van Roy, 2003, Guestrin *et al.*, 2003, de Farias et Van Roy, 2004, Adelman, 2004, de Farias et Van Roy, 2006].

Chapitre 2

Représentations compactes de PDM

Sommaire

Dans ce chapitre, nous introduisons tout d'abord des structures pour la représentation compacte de connaissances. Ces structures peuvent être vues comme des outils pour construire des représentations compactes des PDM. Plus précisément, nous abordons les formules de la logique propositionnelle, avec leurs opérateurs, quelques formes normales et des approximations.

Nous présentons ensuite les représentations des PDM compacts à proprement parler. La caractéristique principale des PDM compacts est que les états sont représentées sous la forme *d'affectations à un ensemble de variables*, où chaque variable représente une caractéristique de l'état. C'est cette notion de caractérisation des états qui donne aux variables une *sémantique* et donc une *structure* qui peut être exploitée pour manipuler efficacement un PDM.

Nous étudions les trois formalismes majeurs de représentation compacte que sont les *réseaux bayésiens dynamiques*, les *opérateurs STRIPS probabilistes* et enfin le *PDDL probabiliste*. En particulier les deux derniers utilisent des formules logiques comme un moyen de représenter

(a) Tables de vérité.

ϕ	ψ	$\phi \wedge \psi$	$\phi \vee \psi$	$\phi \rightarrow \psi$	$\phi \leftrightarrow \psi$	$\phi \oplus \psi$
0	0	0	0	1	1	0
1	0	0	1	0	0	1
0	1	0	1	1	0	1
1	1	1	1	1	1	0

(b) Équivalences entre opérateurs.

$$\phi \rightarrow \psi \iff \neg\phi \vee \psi$$
$$\phi \oplus \psi \iff (\neg\phi \wedge \psi) \vee (\phi \wedge \neg\psi)$$
$$\phi \wedge \psi \iff \neg(\neg\phi \vee \neg\psi)$$
$$\phi \vee \psi \iff \neg(\neg\phi \wedge \neg\psi)$$

TABLE 2.1 – Sémantiques des opérateurs de la logique propositionnelle.

efficacement, *en intension*, des sous-ensembles de l'espace d'états d'un PDM. Ce sont autour de ces deux représentations que s'articulent l'ensemble des travaux présentés.

2.1 Notions de logique propositionnelle

Pour un PDM compact, où les états sont représentés par des affectations à un ensemble de variables $\mathcal{X} = \{x_1, \ldots, x_n\}$, il est naturel de considérer la logique propositionnelle pour représenter des ensembles d'états. En effet, une formule ϕ de la logique propositionnelle sur les variables $\mathcal{X}$ décrit de manière compacte un ensemble de *modèles*, noté $\mathcal{M}(\phi)$ correspondant à toutes les affectations à $\mathcal{X}$ telles que ϕ soit vraie. Les opérateurs sur les ensembles ont des équivalents dans la logique propositionnelle ce qui permet à la fois une représentation et une manipulation efficace tout en maintenant une information sémantique sur ces ensembles.

Cette section décrit tout d'abord formellement la sémantique et les notations utilisées, puis présente la forme normale conjonctive et ses sous-classes traitables.

2.1.1 Formules et sémantique

Les expressions de la logique propositionnelle sont les *formules*, notées par des lettres grecques (le plus souvent, ϕ, ψ, Γ). Toute formule ϕ porte sur un ensemble de variables noté $Var(\phi)$; ces variables peuvent prendre la valeur 0 pour **faux** et 1 pour **vrai**. Les variables d'une formule sont connectées par les opérateurs $\wedge$ (conjonction), $\vee$ (disjonction), $\neg$ (négation), $\rightarrow$ (implication), $\leftrightarrow$ (équivalence) et $\oplus$ (ou-exclusif) et sont régis par la sémantique standard de la logique propositionnelle résumée dans la table 2.1.

Les formules les plus simples sont les *littéraux* qui sont soit une simple variable (littéral *positif*) soit la négation d'une variable (littéral *négatif*). Ainsi pour une variable x, on peut former les littéraux x et $\neg x$. Les littéraux seuls sont le plus souvent notés ℓ et leur complémentaire est noté $\bar{\ell}$. Par exemple si $\ell = \neg x$, son complémentaire est $\bar{\ell} = x$.

Une *affectation* d'un ensemble fini de variables $\mathcal{X}$ est une application de $\mathcal{X}$ dans l'ensemble $\{0,1\}$. L'ensemble des affectations à $\mathcal{X}$ est noté $2^{\mathcal{X}}$. Pour une variable $x_i \in \mathcal{X}$ son affectation à 1 est notée x_i et son affectation à 0, $\bar{x}_i$.

Les *modèles* d'une formule ϕ sont toutes les affectations $m \in 2^{Var(\phi)}$ qui satisfont ϕ, dans ce cas on écrit $m \models \phi$. L'ensemble de ces modèles est noté $\mathcal{M}(\phi)$. Par exemple pour $\phi = x \vee \neg y$, ses modèles sont $\mathcal{M}(\phi) = \{x\bar{y}, xy, \bar{x}\bar{y}\}$. Une formule dont l'ensemble de modèles est vide est une *contradiction*, notée $\bot$, à l'inverse si son ensemble de modèles est $2^{Var(\phi)}$ c'est une *tautologie* notée $\top$. Une formule qui possède au moins un modèle est dite *satisfaisable*. De manière générale, décider la satisfaisabilité d'une formule (le test SAT) est un problème NP-complet.

Pour deux formules ϕ et ψ, on dit que ϕ *implique* ψ, noté $\phi \models \psi$, si tous les modèles de ϕ sont aussi des modèles de ψ :

$$\phi \models \psi \iff \mathcal{M}(\phi) \subseteq \mathcal{M}(\psi)$$

Si lorsque $\phi \models \psi$ et $\psi \models \phi$, ces deux formules sont *équivalentes* et on note $\phi \equiv \psi$.

Un *terme* est une formule constituée seulement d'une conjonction de littéraux. De manière duale, une *clause* est une disjonction de littéraux. Par souci de simplicité nous considérons un terme à la fois comme une formule ou un ensemble de littéraux ou encore comme une affectation. Pour un terme $t = \ell_1 \cdots \ell_k$ on note $\bar{t}$ pour le terme $\bar{\ell}_1 \cdots \bar{\ell}_k$. Pour un terme t et un ensemble de variables $X \subseteq \mathcal{X}$, on note $t[X]$ pour le sous-terme de t restreint aux variables de X.

Définition 3 (inconsistance mutuelle) *Deux formules ϕ et ψ sont mutuellement inconsistantes si et seulement si elles n'ont pas de modèles en commun, c'est-à-dire que leur conjonction est insatisfaisable. Plus généralement, un ensemble de formules mutuellement inconsistantes est un ensemble de formules inconsistantes deux à deux.*

Une opération importante sur les formules que nous utiliserons souvent est la notion *d'oubli de variables*. Intuitivement, étant donnés un ensemble de variables X et une formule ϕ, il s'agit de calculer une forme propositionnelle de la formule $\exists X, \phi$. La définition formelle est la suivante :

Définition 4 (oubli de variables) *Soient X un ensemble de variables propositionnelles et ϕ une formule. Alors* l'oubli *de X dans ϕ, noté $Oubli(X,\phi)$ est défini récursivement comme :*

- $Oubli(\emptyset, \phi) = \phi$
- $Oubli(\{x\}, \phi) = \phi_{x \leftarrow 0} \vee \phi_{x \leftarrow 1}$
- $Oubli(\{x\} \cup X, \phi) = Oubli(\{x\}, Oubli(X,\phi))$

L'oubli peut s'interpréter comme une projection de la formule sur un sous-ensemble de variables. L'ensemble des modèles d'une formule après oubli est composé des modèles de la formule initiale sans considérer les littéraux sur les variables oubliées comme le montre l'exemple suivant.

Exemple 5 *Soit $\phi = x \vee y$, les modèles de ϕ sont $\mathcal{M}(\phi) = \{xy, \bar{x}y, x\bar{y}\}$. En oubliant x dans ϕ nous obtenons alors $\psi = Oubli(\{x\},\phi) = 0 \vee y \vee y \vee 1 = \top$ et donc en considérant les modèles de ψ sur $\{x,y\} \setminus \{x\} = \{y\}$ nous avons $\mathcal{M}(\psi) = \{y, \bar{y}\}$.*

2.1.2 Forme normale conjonctive et classes traitables

Toute formule propositionnelle ϕ possède une représentation en Forme Normale Conjonctive (CNF). Une formule en CNF est une conjonction de clauses. Par souci de lisibilité nous considérerons parfois une formule CNF comme l'ensemble de ses clauses.

Exemple 6 *La formule* $\phi = (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3 \vee \neg x_4)$ *est sous forme normale conjonctive.*

Dans les chapitres suivants, nous nous intéresserons plus particulièrement à des sous-classes syntaxiques de la CNF. En introduisant des contraintes sur la syntaxe des formules, il est alors possible de définir des *classes* aux propriétés intéressantes. Les deux paragraphes suivants s'intéressent à des restrictions sur les clauses des formules CNF. Ces langages restreints ont la propriété particulière de disposer d'algorithmes polynomiaux pour le test SAT. Ce sont cependant des langages *incomplets*, c'est-à-dire que toute formule n'a pas nécessairement d'équivalent dans ces classes.

Clauses de Horn

Une clause de Horn est une clause qui ne contient au plus qu'un littéral positif. Une formule CNF de Horn est une conjonction de clauses de Horn. Une des propriétés intéressantes de ces clauses est qu'elles peuvent toutes être réécrites sous la forme de règles :

$$\neg x_{i_1} \vee \neg x_{i_2} \vee \cdots \vee \neg x_{i_k} \vee \ell \equiv (x_{i_1} \wedge x_{i_2} \wedge \cdots \wedge x_{i_k}) \rightarrow \ell$$

Le test de satisfaisabilité pour Horn est en temps linéaire [Dowling, W.F. and Gallier, 1984].

Exemple 7 *La formule* $\phi = (\neg x_1 \vee x_2 \vee \neg x_4 \vee \neg x_6) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_5)$ *est une CNF de Horn.*

Clauses de taille 2

Le langage des CNF restreintes aux clauses de taille au plus 2 (la 2-CNF) est lui aussi un langage traitable qui permet le test de satisfaisabilité en temps linéaire. Cette propriété vient du fait que toute clause de taille 2 correspond à deux implications :

$$x_i \vee x_j \equiv \neg x_i \rightarrow x_j \equiv \neg x_j \rightarrow x_i$$

C'est l'interprétation de ces implications sous la forme d'un graphe orienté entre les littéraux qui permet de construire un algorithme efficace pour le test de satisfaisabilité basé sur la recherche de composantes fortement connexes [Aspvall *et al.*, 1979].

Exemple 8 *La formule* $\phi = (\neg x_1 \vee x_2) \wedge (x_4 \vee \neg x_6) \wedge (\neg x_2 \vee \neg x_3) \wedge \neg x_5$ *est une 2-CNF.*

2.1.3 Approximation de formules dans des classes traitables

Bien que toute formule ϕ n'ait pas nécessairement d'équivalent en CNF de Horn ou en 2-CNF, [Kautz *et al.*, 1995] proposent des algorithmes *d'approximation* de ϕ dans les classes 2-CNF et Horn. L'approximation consiste à construire deux formules ϕ_{lb} et ϕ_{ub} chacune dans une classe traitable et telles que

$$\phi_{lb} \models \phi \models \phi_{ub}$$

La formule ϕ_{lb} est un *minorant* de ϕ et ϕ_{ub} est un *majorant* de ϕ. Si lors de l'approximation de ϕ dans une classe $\mathcal{C}$ il n'existe pas de formule $\phi' \in \mathcal{C}$, $\phi' \neq \phi_{lb}$ telle que $\phi_{lb} \models \phi' \models \phi$ alors ϕ_{lb} est un *minorant maximal* de ϕ. Dualement, ϕ_{ub} est un *majorant minimal* s'il n'existe pas de $\phi' \in \mathcal{C}$, $\phi' \neq \phi_{ub}$ tel que $\phi \models \phi' \models \phi_{ub}$

Exemple 9 *Soit une CNF* $\phi = (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_3) \wedge (x_1 \vee x_2)$. *Les formules* $x_1 \wedge x_2 \wedge x_3$, $x_1 \wedge x_3$ *et* $x_2 \wedge x_3$ *sont des minorants de Horn pour* ϕ *; de plus les deux derniers sont des minorants maximaux. La formule* $(\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_3)$ *est un majorant et c est un majorant minimal.*

$$x_1 \wedge x_2 \wedge x_3 \models x_1 \wedge x_3 \models \phi \models c \models (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_3)$$

Nous utiliserons ces techniques d'approximation pour simplifier la description des conditions d'actions décrites en STRIPS probabiliste (section 2.5). En effet, par la suite (chapitre 4) nous présentons un algorithme pour résoudre des PDM décrits de cette façon. Cet algorithme nécessitant de nombreux tests SAT, ces approximations permettent de réaliser ces tests en temps polynomial.

2.2 Diagrammes de décision

Nous présentons à présent les diagrammes de décision. Ceux-ci sont une façon particulière de représenter et d'implémenter des fonctions dont l'ensemble de départ est défini par les affectations à un ensemble de variables propositionnelles $\mathcal{X}$. Nous distinguons deux types de diagrammes de décision : les *binaires* pour représenter des fonctions de la forme $\{0,1\}^{\mathcal{X}} \to \{0,1\}$, c'est-à-dire les formules logiques et les diagrammes *algébriques*, plus généraux, pour représenter les fonctions $\{0,1\}^{\mathcal{X}} \to \mathbb{R}$. Le bon compromis entre compacité et efficacité de manipulation offert par les diagrammes de décision en font un outil de référence pour l'implémentation de solveurs de PDM efficaces.

2.2.1 Diagrammes de décision binaires

Les diagrammes de décision binaires (BDD) [Bryant, 1986] sont une représentation sous forme de graphe orienté acyclique de formules logiques. Ces diagrammes sont une version plus compacte des arbres de décision où les nœuds correspondent aux deux valeurs possibles d'une variable et les feuilles sont 0 et 1. La compacité des BDD par rapport aux arbres de décision vient du fait que les sous-diagrammes communs sont partagés et non pas dupliqués. À titre

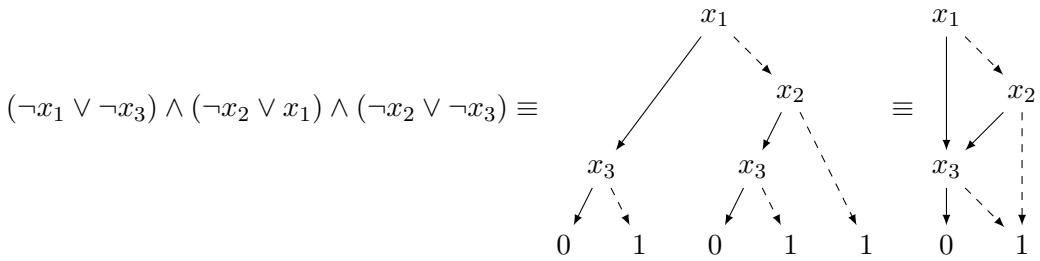


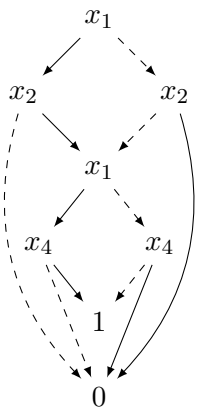
FIGURE 2.1 – Différentes représentations d'une même formule : CNF, Arbre de décision et BDD.

d'exemple, la figure 2.1 montre les différences entre les représentations CNF, arbre de décision et BDD d'une même formule.

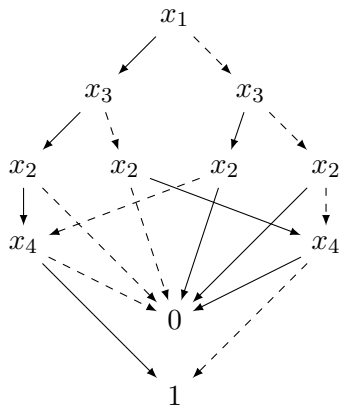
En imposant un ordre total sur l'ordre d'apparition des variables dans le diagramme, les BDD permettent alors une représentation canonique de chaque formule. Ce qui permet notamment de pouvoir tester l'équivalence entre deux formules simplement par comparaison des diagrammes. Le test SAT est aussi très simple, la formule représentée par un BDD est satisfaisable si et seulement si ce BDD n'est pas la feuille 0. De plus, l'ordre sur les variables permet d'obtenir des implémentations très efficaces en termes de mémoire via l'utilisation de caches [Brace *et al.*, 1991]. Cet ordre permet aussi d'avoir des algorithmes polynomiaux en la taille des BDD pour les opérations telles que la conjonction ou la disjonction de deux diagrammes. La négation peut s'effectuer en temps linéaire en inversant les deux feuilles 0 et 1, certaines implémentations permettent même la négation en temps constant.

Tous les avantages computationnels obtenus en imposant un ordre sur les variables se font au prix d'une perte de compacité dans la taille de la représentation. En effet, un BDD peut être très compact pour un ordre donné et ensuite être exponentiellement plus grand avec un ordre différent. Déterminer l'ordre optimal pour un BDD ou un ensemble de BDDs est un problème NP-complet qui, dans la pratique, se résout avec des méthodes approchées [Rudell, 1993] capables de balancer le compromis entre le temps de recherche d'un ordre et sa compacité.

Exemple 10 *Pour la fonction $\phi = x_1 \leftrightarrow x_2 \wedge x_3 \leftrightarrow x_4$ sa représentation en BDD avec l'ordre $x_1 \prec x_2 \prec x_3 \prec x_4$ est*



tandis qu'avec l'ordre $x_1 \prec x_3 \prec x_2 \prec x_4$ le BDD obtenu est :



De manière plus générale, la formule $x_0 \leftrightarrow x_1 \wedge \cdots \wedge x_{2n} \leftrightarrow x_{2n+1}$ se représente avec un BDD contenant $O(n)$ nœuds avec l'ordre $x_0 \prec x_1 \prec \cdots \prec x_{2n+1}$. Tandis que l'ordre $x_0 \prec x_2 \prec \cdots \prec x_{2n} \prec x_1 \prec x_3 \prec \cdots \prec x_{2n+1}$ conduit à des BDD contenant $O(2^n)$ nœuds.

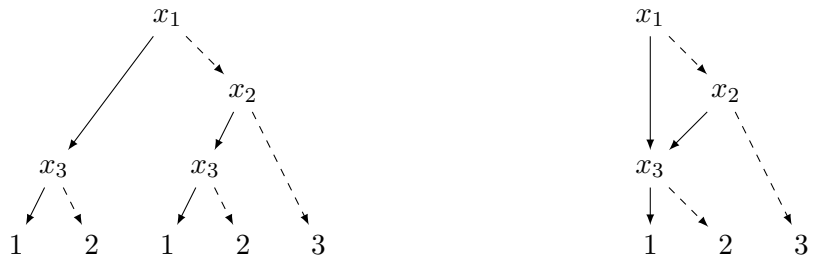
Avantages et inconvénients liés à l'utilisation des BDD

Dans nos travaux nous avons utilisé les BDD en tant que formalisme de *représentation* de formules logiques lors de *l'implémentation* de nos algorithmes. Cette représentation des formules n'est de loin pas la plus compacte en effet, certaines formules nécessite une représentation en BDD de taille exponentielle en comparaison à une utilisation sans restriction [1] des connecteurs logiques [Darwiche et Marquis, 2002]. Cependant, les BDD disposent d'algorithmes efficaces pour la *manipulation* des formules. Ainsi le test de satisfaisabilité, la conjonction de deux BDD, la négation d'un BDD... sont réalisables en temps polynomial, ce qui est loin d'être le cas pour d'autres représentations plus compactes. Pour plus de détails sur la compacité et la manipulabilité des diverses représentations des formules propositionnelles, nous renvoyons le lecteur à [Darwiche et Marquis, 2002, Fargier et Marquis, 2008] qui proposent une étude détaillée de la compilation de connaissances propositionnelles.

2.2.2 Diagrammes de décision algébriques

Les diagrammes de décision algébriques (ADD) [Bahar *et al.*, 1997] sont une représentation compacte de fonctions de la forme $\{0,1\}^n \to \mathbb{R}$. Ils sont similaires aux BDD avec la différence que les feuilles sont des nombres réels. En effet, ces fonctions associent une valeur réelle aux affectations à un ensemble de variable binaires. Les ADD sont une représentation plus compacte que les arbres de décisions binaires comme le montre la figure 2.2. Leur structure est particulièrement compacte pour représenter les fonctions pour lesquelles un grand nombre d'antécédents ont la même image. Nous verrons par la suite qu'ils sont utilisés pour représenter efficacement des Tables de Probabilités Conditionnelles comme pour les arbres de décision de la figure 2.4.

1. À l'exception de la négation $\neg$.



(a) Arbre de décision. (b) ADD équivalent.

FIGURE 2.2 – Arbre de décision binaire et son ADD équivalent.

$$
\begin{aligned}
(f \bullet g)(\mathbf{x}) &= f(\mathbf{x}) \bullet g(\mathbf{x}) \text{ avec } \bullet \in \{+, -, \times, /\} \\
\max(f,g)(\mathbf{x}) &= \max(f(\mathbf{x}), g(\mathbf{x})) \\
(\phi \times f)(\mathbf{x}) &= f(\mathbf{x}) \text{ si } \mathbf{x} \models \phi \text{ sinon } 0 \\
\mathrm{ITE}(\phi,f,g)(\mathbf{x}) &= f(\mathbf{x}) \text{ si } \mathbf{x} \models \phi \text{ sinon } g(\mathbf{x}) \\
(f[t])(\mathbf{x}) &= f(\mathit{apply}(t,\mathbf{x})) \\
(\exists x_i f)(\mathbf{x}) &= (f[x_i])(\mathbf{x}) + (f[\bar{x}_i])(\mathbf{x}) \\
(\forall x_i f)(\mathbf{x}) &= (f[x_i])(\mathbf{x}) \times (f[\bar{x}_i])(\mathbf{x}) \\
(f > g)(\mathbf{x}) &= 1 \text{ si } f(\mathbf{x}) > g(\mathbf{x}) \text{ sinon } 0
\end{aligned}
$$

TABLE 2.2 – Sémantique des opérateurs sur les fonctions représentées par des ADD. Avec f,g : $2^{\mathcal{X}} \to \mathbb{R}$, $\mathbf{x} \in 2^{\mathcal{X}}$, $x_i \in \mathcal{X}$, ϕ une formule sur $\mathcal{X}$ et t un terme sur $\mathcal{X}$.

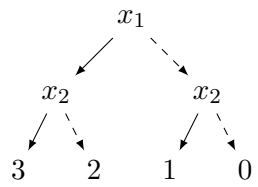
De façon similaire aux BDD, les ADD disposent d'une large palette d'opérateurs sur les fonctions qu'ils représentent. On trouve notamment les opérateurs algébriques ($+$, $-$,$\times$,/,max ,min ,...), les abstractions de variables via divers opérateurs comme la somme, min ou max , le conditionnement (forcer la valeur d'une variable), les opérateurs logiques pour les fonctions booléennes, etc. La sémantique des opérateurs utilisés est décrite dans la table 2.2.

En imposant un ordre total quelconque sur les variables, chaque fonction possède alors une représentation unique en ADD, comme les BDD, ce qui permet de construire des algorithmes efficaces pour les opérations via l'introduction de caches pour stocker les résultats intermédiaires. Cependant, l'ordre des variables a un impact important sur la taille d'un diagramme, ce qui pénalise à la fois la complexité en espace de la représentation et la complexité en temps des opérations.

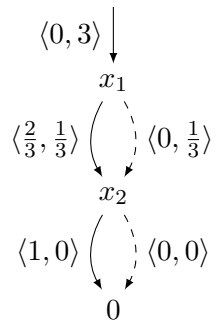
Les ADD affines

Certaines fonctions peuvent bénéficier une représentation encore plus compacte en utilisant des *ADD affines* [Sanner et McAllester, 2005] (AADD). Les AADD sont une extension des ADD qui comportent sur chaque arête une transformation affine représentée par deux valeurs $\langle a, b\rangle$. Ainsi lors de l'évaluation d'une affectation de variables, pour chaque arête traversée, on retourne la valeur $a+bv$ où v est la valeur issue de l'évaluation du sous-diagramme pointé par cette arrête. L'exemple suivant montre la compacité et la méthode d'évaluation des AADD.

Exemple 11 *Considérons l'ADD suivant :*



Cet ADD n'est pas plus compact qu'un arbre de décision. Cependant sa représentation en ADD affine est la suivante :



Ainsi l'évaluation de cet AADD pour $\mathbf{x} = x_1\bar{x}_2$ *s'effectue en propageant les transformations affines depuis la feuille* 0 *jusqu'à la racine en suivant les arêtes indiquées par* $\mathbf{x}$*. On calcule donc*

$$0 + 3 \times \left(\frac{2}{3} + \frac{1}{3} \times (0 + 0 \times 0)\right) = 2$$

La racine d'un AADD étant une arête, on observe que des opérations telles que l'ajout d'une constante ou la multiplication par une constante sont réalisables en temps constant : il suffit de modifier les coefficients de l'arête racine.

L'utilisation des AADD pour la résolution de PDM factorisés a donné de bons résultats en termes de compacité des fonctions de valeur [Sanner et McAllester, 2005]. Leur implémentation reste cependant délicate car la succession d'additions et de multiplications entraîne des erreurs de précision numérique lorsque le nombre de variables augmente. Cette structure de données reste toutefois très intéressante pour la résolution de PDM compacts.

2.3 Enjeux des représentations compactes de PDM

Les PDM tels que présentés dans le chapitre précédent possèdent un espace d'états énuméré. En suivant cette représentation pour modéliser un problème donné il serait donc nécessaire de décrire en *extension* tous les composants du PDM. Pour un problème à N états et M actions une telle représentation « éclatée » du problème nécessite un espace en $O(MN^2 + MN)$ pour décrire les fonctions de transition et de récompense. Cependant l'espace d'états peut être vu comme un ensemble de valuation de descripteurs (ou variables d'états) $x_1, \ldots, x_n$, en conséquence, le nombre d'états N est alors exponentiel en le nombre de ces descripteurs. Historiquement, cette explosion combinatoire a été formulée sous le nom de « *malédiction de la dimensionalité* ».

Cependant, pour les problèmes naturels, il y a le plus souvent une structure sous-jacente qui permet de représenter de manière compacte le processus de décision. Prenons par exemple le cas d'une action « allumer la lumière ». Une description compacte de cette action serait de dire : « avec une certaine probabilité p la lumière devient allumée, le reste du temps son état ne change pas ». Une représentation en extension devrait être spécifiée ainsi : « si la lumière est éteinte et la porte ouverte, alors la lumière sera allumée et la porte sera ouverte avec une probabilité p ; si la lumière est éteinte et la porte fermée alors la lumière sera allumée et la porte sera fermée avec une probabilité p, ... ».

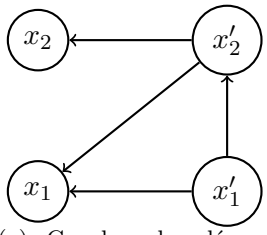
Au delà de cette simplification représentationelle, il est naturel de supposer qu'il est possible de tirer parti de cette structure pour simplifier le calcul de politiques. Notamment, de nombreux états peuvent être équivalents en termes de récompense espérée, ce qui permet de les traiter simultanément lors de la résolution.

Le reste de ce chapitre est consacré à la présentation des formalismes de représentation.

2.4 Réseaux bayésiens dynamiques

La représentation de fonctions de transition par réseaux bayésiens dynamiques [Dean et Kanazawa, 1989] (DBN) suppose que l'évolution de chaque variable d'état selon les actions ne dépend que des valeurs d'un petit nombre d'autres variables. Ainsi, la fonction de transition de chaque action est représentée comme le *produit* de fonctions de transitions partielles. C'est de cette notion de produit que vient le terme de PDM factorisés.

Dans un PDM factorisé, deux ensembles de variables interviennent : $\mathcal{X} = \{x_1, \ldots, x_n\}$ dites les variables *pré-action* et $\mathcal{X}' = \{x'_1, \ldots, x'_n\}$ dites *post-action*. Pour chaque action, chaque variable x_i représente une valeur avant l'action et x'_i représente la valeur de cette même variable après l'exécution de l'action. Un réseau bayésien dynamique exprime les dépendances probabilistes de chaque variable post-action x'_i avec les autres variables. Les dépendances entre x'_i et les variables pré-action sont dites *asynchrones* et celles avec des variables post-action sont dites *synchrones*. Un réseau bayésien pour une action peut alors être représenté par un graphe orienté acyclique ayant pour nœuds $\mathcal{X} \cup \mathcal{X}'$, une arête entre un nœud post-action x'_i et un autre nœud indiquant une dépendance probabiliste (Figure 2.3). Les indépendances sont alors implicites



(a) Graphe de dépendances

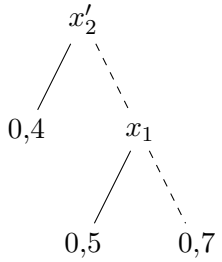
x_1	x'_2	$P(x'_1 = 1 \mid x_1, x'_2)$
0	0	0,7
0	1	0,4
1	0	0,5
1	1	0,4

(b) $CPT(x'_1)$

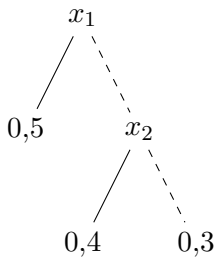
x_1	x_2	$P(x'_2 = 1 \mid x_1, x_2)$
0	0	0,3
0	1	0,4
1	0	0,5
1	1	0,5

(c) $CPT(x'_2)$

FIGURE 2.3 – Un exemple de réseau bayésien dynamique à 2 variables



(a) Arbre de décision pour $CPT(x'_1)$



(b) Arbre de décision pour $CPT(x'_2)$

FIGURE 2.4 – Représentation compacte sous forme d'arbre de décision des tables de probabilités de la figure 2.3

dans ce modèle. Pour chaque x'_i, l'ensemble des variables dont dépend x'_i est noté $Parents_a(x'_i)$. À chaque x'_i est associée une *table de probabilités conditionnelles* $CPT_a(x'_i)$ qui, pour chaque affectation aux variables de $Parents_a(x'_i)$, donne la probabilité que l'action a mette x'_i à 1. Chaque $CPT_a(x'_i)$ définit une distribution de probabilité $P_a(x'_i = 1 | Parents_a(x'_i))$ qui, pour toute affectation $\mathbf{x}$ à $Parents_a(x'_i)$, donne la probabilité que l'action a mette x'_i à 1 dans un état satisfaisant $\mathbf{x}$ [2]. La probabilité de mettre x'_i à 0 est naturellement : $1 - P_a(x'_i = 1 | Parents_a(x'_i))$. La fonction de transition entre états complets du système $P(s'|s,a)$ peut ainsi s'exprimer :

$$P(s'|s,a) = \prod_{i=1}^{n} P_a\left(x'_i = s'[i] \mid s,s'\right)$$

La compacité de cette représentation peut être encore améliorée quand les tables de probabilités conditionnelles exhibent elles aussi une certaine structure. Par exemple, les tables de la figure 2.3 peuvent être compressées sous forme d'arbres de décision (figure 2.4). D'autres représentations encore plus compactes de ces tables sont possibles, dans le chapitre 7 nous présenterons des algorithmes de calcul de politique qui traitent de façon directe ces tables compactes.

2. S'il y a des dépendances synchrones, il faut aussi que l'état d'arrivée satisfasse $\mathbf{x}$.

2.5 Opérateurs STRIPS probabilistes

Un second langage de représentation pour les actions sont les opérateurs STRIPS probabilistes [Kushmerick *et al.*, 1995] (PSO). C'est une variante de STRIPS, une représentation d'actions déterministes pour la planification classique [Fikes et Nilsson, 1971] qui est étendue aux actions probabilistes. Les éléments centraux des actions sont les *effets*, qui indiquent la valeur de certaines variables après exécution de l'action, les autres restant inchangées. Dans le cas des variables d'état propositionnelles, un effet est donc un ensemble consistant de littéraux de $\mathcal{X}$.

Définition 12 (application d'un effet à un état) *Soit $s \in 2^{\mathcal{X}}$ un état et e un effet. L'application de e à s, notée $apply(e,s)$ correspond à un état défini comme :*

$$apply(e,s) = e \cup \left(s \setminus \left\{\ell \middle| \bar{\ell} \in e\right\}\right)$$

C'est en grande partie le fait de ne pas préciser explicitement la persistance des variables inchangées qui confère aux PSO leur compacité, contrairement aux DBN qui doivent encoder cette persistance.

Dans le cadre probabiliste, les actions sont divisées en *conditions*, représentant un ensemble d'états, chacune comportant une distribution de probabilités sur un ensemble possible d'effets. Dans le cas des variables d'état propositionnelles, une condition peut être une formule logique, dont les modèles correspondent alors aux états dans lesquels les effets s'appliquent.

Exemple 13 (Le domaine Coffee-Robot**)** *Le domaine* Coffee-Robot *[Boutilier et Dearden, 1994] met en scène un robot dont le but est de sortir d'un bureau pour aller chercher un café pour un utilisateur. Le café se situant dans un autre bâtiment, il doit donc passer à l'extérieur au risque de se faire mouiller quand il pleut. Le robot dispose de quatre actions : changer de bâtiment pour passer du bureau au café et vice-versa (***move***) ; prendre un parapluie pour éviter d'être mouillé quand il pleut (***get-umbrella***) ; acheter un café (***buy-coffee***) ; et donner un café à l'utilisateur (***give-coffee***). Chacune de ces actions a une probabilité d'échouer : par exemple* **move** *peut laisser le robot à sa position initiale, ou encore donner le café peut mouiller le robot si celui-ci le renverse. L'objectif du robot est donc d'être sec (récompense* 0,2*) et que l'utilisateur ait un café (récompense* 0,8*).*

Ce problème est très facile à représenter en PSO, comme le montre la figure 2.5 qui présente l'action **move** *pour ce problème.*

Plus formellement, une action a est un ensemble $\{c_i\}_i$ de conditions mutuellement exclusives. À chacune d'elles est associée une distribution de probabilités $P_i^a(\cdot)$ sur un ensemble fini d'effets E_i^a. Nous notons $Eff(a,s)$ pour l'ensemble des effets associés à un état s par une action a, c'est-à-dire l'ensemble E_i^a tel que $s \models c_i^a$. De cette façon la fonction de transition entre états est définie par :

$$P(s'|s,a) = \sum \{P_i^a(e) \mid e \in Eff(a,s), apply(e,s) = s'\} \tag{2.1}$$

Condition	Effet	Prob.
$InOffice \wedge Raining \wedge \neg HasUmbrella$	$\neg InOffice, IsWet$	0.81
	$\neg InOffice$	0.09
	$IsWet$	0.09
	$\emptyset$	0.01
$InOffice \wedge (\neg Raining \vee HasUmbrella)$	$\neg InOffice$	0.9
	$\emptyset$	0.1
$\neg InOffice \wedge Raining \wedge \neg HasUmbrella$	$InOffice, IsWet$	0.81
	$InOffice$	0.09
	$IsWet$	0.09
	$\emptyset$	0.01
$\neg InOffice \wedge (\neg Raining \vee HasUmbrella)$	$InOffice$	0.9
	$\emptyset$	0.1

FIGURE 2.5 – Exemple de PSO

Notons qu'une somme apparaît dans l'équation 2.1 car, pour certains états s, il se peut que l'application de *plusieurs* effets produisent le même s' une fois appliqués à s. Nous formaliserons cette observation dans le chapitre 7.

La fonction de récompense se présente quant à elle sous la forme d'une liste de formules sur $\mathcal{X}$, chacune étant adjointe d'un nombre réel représentant la récompense associée aux états satisfaisant cette formule. Plus formellement on note $R_a = \{\phi_i, r_i\}_i$ l'ensemble fini de couples formules-réels représentant la fonction de récompense d'une action a qui est calculée de manière additive :

$$R(s,a) = \sum \{r_i | (\phi_i, r_i) \in R_a,\ s \models \phi_i\}$$

Exemple 14 (fonction de récompense) *Dans le domaine* COFFEE-ROBOT*, le robot obtient* 0,8 *dans tous les états où l'utilisateur a du café, à cela on ajoute* 0,2 *s'il n'est pas mouillé. La fonction de récompense peut alors s'écrire comme*

$$R = \{(UserHasCoffee, 0{,}8), (\neg Wet, 0{,}2)\}$$

2.6 PDDL probabiliste

Le langage de description de domaines de planification PDDL (planning domain definition language) est un standard largement utilisé par la communauté de planification déterministe. Il propose une représentation compacte des actions sous la forme d'effets imbriqués. Nous étudions ici la variante probabiliste de PDDL (PPDDL) qui permet de décrire des actions aux issues incertaines et en particulier n'importe quel PDM [Younes et Littman, 2004]. Ce langage peut

être vu comme une généralisation des opérateurs STRIPS probabilistes car il n'impose pas une structure aussi stricte que les PSO (conditions + distribution d'effets) et permet parfois des représentations exponentiellement plus compactes (cf. exemple 15).

PPDDL permet la description de domaines de planification au premier ordre. C'est-à-dire qu'il est possible d'utiliser des prédicats sur des objets au lieu de simples propositions. Dans ce cas on parle de schémas d'action qui, une fois que l'on a précisé les objets du domaine, seront instanciés pour former des actions qui, elles, portent sur un domaine propositionnel.

2.6.1 Actions et effets

En PPDDL, une action a est constituée de deux éléments : une pré-condition ϕ_a qui est une formule représentant les états dans lesquels l'action peut être exécutée et un effet e_a qui décrit les changements que produit l'action sur l'état et la récompense obtenue. La norme PPDDL 1.0 décrit cinq types d'effets différents, répartis en deux catégories : les effets terminaux et les effets d'agrégation. Les effets terminaux sont ceux qui soit agissent directement sur les variables d'état en forçant leur valeur, soit modifient la récompense obtenue par l'agent (Table 2.3 (a)). On dispose alors des effets suivants :

- Les effets *simples* : x ou $\neg x$ qui forcent la valeur d'une variable x.
- Les effets de *mise-à-jour* : $\uparrow r$ qui permettent d'augmenter la récompense perçue par l'agent de r (r peut être négatif).

Les effets d'agrégation (Table 2.3 (b)) permettent de combiner ou de restreindre l'occurrence d'autres effets. Parmi ces effets d'agrégation on distingue :

- Les effets *conditionnels* : $\phi \triangleright e$ qui restreignent l'occurrence d'un effet e aux états satisfaisants la formule ϕ. Ils sont équivalents aux conditions des PSO.
- Les effets *conjonctifs* : $e_1 \wedge \cdots \wedge e_k$ qui permettent de décrire l'occurrence simultanée de plusieurs effets $e_1, \ldots, e_k$, avec comme contrainte que les effets simples soient consistants, c'est-à-dire qu'ils ne forcent pas une variable et sa négation. Quand $k = 0$ on parle alors de l'effet vide, noté $\top$ qui ne modifie pas l'état.
- Les effets *probabilistes* : $p_1 e_1 | \ldots | p_k e_k$ qui décrivent une distribution de probabilités entre plusieurs effets $e_1, \ldots, e_k$, terminaux ou d'agrégation. On requiert que $\sum_i p_i = 1$, dans le cas où cette somme est inférieure à 1, on considère que l'effet $\top$ peut se produire avec probabilité $1 - \sum_i p_i$.

La table 2.3 présente les différents effets de PPDDL avec leur syntaxe et la notation que nous utiliserons pour les représenter formellement. La table 2.4 [Rintanen, 2003] présente les différentes équivalences entre effets. L'action **move** du problème Coffee-Robot (exemple 13) décrite en PPDDL avec sa notation formelle est présentée par la figure 2.6.

Remarquons que les opérateurs STRIPS probabilistes correspondent au PPDDL restreint aux actions de la forme

$$\phi_1 \triangleright (p_{1,1}\ e_{1,1} | \ldots | p_{1,q_1}\ e_{1,q_1}) \wedge \cdots \wedge \phi_k \triangleright (p_{k,1}\ e_{k,1} | \ldots | p_{k,q_k}\ e_{k,q_k})$$

(a) Les effets terminaux de PPDDL

Syntaxe	**Notation**	**Description**
(x)	x	Force la valeur de la variable x à **vrai**
$(\texttt{not}\ \ x)$	$\neg x$	Force la valeur de la variable x à **faux**
$(\texttt{increase}\ (\texttt{reward})\ \ r)$	$\uparrow r$	Augmente la récompense de l'agent de $r \in \mathbb{R}$

(b) Les effets d'agrégation de PPDDL

Syntaxe	**Notation**	**Description**
$(\texttt{when}\ \ \phi\ e)$	$\phi \triangleright e$	L'effet e n'est appliqué que dans les états satisfaisant la formule ϕ
$(\texttt{probabilistic}\ \ p_1\ e_1\ \ldots\ p_k\ e_k)$	$p_1\ e_1 \mid \ldots \mid p_k\ e_k$	Chaque effet e_i peut se produire avec probabilité p_i
$(\texttt{and}\ \ e_1\ \ldots\ e_k)$	$e_1 \wedge \cdots \wedge e_k$	Les effets $e_1, \ldots, e_k$ sont consistants et se produisent simultanément
$(\texttt{and})$	$\top$	Effet vide, ne change pas l'état dans lequel il est appliqué

Table 2.3 – Les effets de PPDDL

$$
\begin{aligned}
c \triangleright (e_1 \wedge \cdots \wedge e_k) &\equiv (c \triangleright e_1) \wedge \cdots \wedge (c \triangleright e_k) \\
c \triangleright (c' \triangleright e) &\equiv c \wedge c' \triangleright e \\
c \triangleright (p_1 e_1 | \cdots | p_k e_k) &\equiv p_1 (c \triangleright e_1) | \cdots | p_k (c \triangleright e_k) \\
(c \triangleright e) \wedge (c' \triangleright e) &\equiv (c \vee c') \triangleright e \\
e \wedge (c \triangleright e) &\equiv e \\
e &\equiv \top \triangleright e \\
e \wedge (p_1 e_1 | \cdots | p_k e_k) &\equiv p_1 (e \wedge e_1) | \cdots | p_k (e \wedge e_k) \\
p_1 (p'_1 e'_1 | \cdots | p'_k e'_k) | p_2 e_2 | \cdots | p_n e_n &\equiv (p_1 p'_1) e'_1 | \cdots | (p_1 p'_k) e'_k | p_2 e_2 | \cdots | p_n e_n \\
p_1 (e' \wedge (c \triangleright e_1)) | p_2 e_2 | \cdots | p_k e_k &\equiv (c \triangleright (p_1 (e' \wedge e_1) | p_2 e_2 | \cdots | p_k e_k)) \wedge (\neg c \triangleright (p_1 e' | p_2 e_2 | \cdots | p_k e_k))
\end{aligned}
$$

Table 2.4 – Équivalences entre effets [Rintanen, 2003].

```
(:action move :effect
(and
  (when (in-office)                (probabilistic 0.9 (not (in-office))))
  (when (not (in-office))          (probabilistic 0.9 (in-office)))
  (when (and (raining)
             (not (has-umbrella))) (probabilistic 0.9 (is-wet)))
  (when (user-has-coffee)          (increase (reward) 0.8))
  (when (not is-wet)               (increase (reward) 0.2))))
```

(a) Code PPDDL

$$
\begin{aligned}
& \mathit{InOffice} \triangleright (0{,}9\ \neg \mathit{InOffice} | 0{,}1\ \top) \\
\wedge\ & \neg \mathit{InOffice} \triangleright (0{,}9\ \mathit{InOffice} | 0{,}1\ \top) \\
\wedge\ & (\mathit{Raining} \wedge \neg \mathit{HasUmbrella}) \triangleright (0{,}9\ \mathit{IsWet} | 0{,}1\ \top) \\
\wedge\ & \mathit{UserHasCoffee} \triangleright \uparrow 0{,}8 \\
\wedge\ & \neg \mathit{IsWet} \triangleright \uparrow 0{,}2
\end{aligned}
$$

(b) Notation

FIGURE 2.6 – Représentation PPDDL de l'action **move** du problème COFFEE-ROBOT

où chaque effet $e_{i,j}$ est une conjonction d'effets simples et les formules ϕ_i sont inconsistantes deux à deux. C'est en effet la possibilité de pouvoir exprimer des effets conjonctifs qui contiennent des conditions mutuellement consistantes qui confère à PPDDL la possibilité de représenter certaines actions de manière exponentiellement plus compacte que les PSO, comme le montre l'exemple suivant.

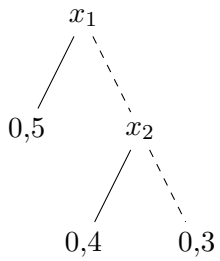
Exemple 15 *(PPDDL vs. PSO) Considérons l'effet PPDDL :*

$$e = x_1 \triangleright y_1 \wedge \cdots \wedge x_n \triangleright y_n$$

Chacune des conditions $x_1, \ldots, x_n$ étant consistantes deux à deux, une représentation en opérateurs STRIPS probabilistes devrait énumérer les 2^n conditions possibles sur les variables $x_1, \ldots, x_n$ car chacune des affectations à ces variables mène à un effet différent.

Un autre exemple de la puissance d'expressivité de PPDDL est qu'il permet de représenter les tables de probabilité conditionnelles d'un réseau bayésien dynamique sans arcs synchrones en taille linéaire en leur description sous forme d'arbre de décision.

Exemple 16 *(PPDDL vs. DBN) Considérons l'arbre de décision suivant, qui représente une table de probabilités conditionnelles pour une variable y' dépendant des variables x_1 et x_2.*



Une description en PPDDL de cet arbre est alors l'effet

$$x_1 \triangleright (0{,}5\ y|0{,}5\ \neg y) \wedge \neg x_1 \triangleright (x_2 \triangleright (0{,}4\ y|0{,}6\ \neg y) \wedge \neg x_2 \triangleright (0{,}3\ y|0{,}7\ \neg y))$$

qui est de taille linéaire en celle de l'arbre de décision. Une action complète peut donc être représentée par la conjonction des effets construits à partir de chaque arbre de probabilités.

De manière générale, pour l'arbre de probabilités d'une variable v' on réalise la transformation suivante. Chaque feuille étiquetée par une probabilité p est transformée en l'effet $(p\ v|(1-p)\ \neg v)$ et chaque nœud avec une variable x est transformé en l'effet $x \triangleright e_1 \wedge \neg x \triangleright e_0$ où e_1 (resp. e_0) est la transformation du sous-arbre ou x est à vrai (resp. à faux). Notons que dans le cas d'un ADD (cf. section 2.2.2) représentant ces probabilités, cette transformation n'est pas nécessairement linéaire en la taille du diagramme.

2.6.2 Interprétation d'un PDM décrit en PPDDL

La définition suivante caractérise pour chaque état s et un effet e quelconque la distribution de probabilités sur les effets basiques (au sens des PSO) et les récompenses induites par l'application de e dans s.

Définition 17 (distribution induite) *Pour tout état s, un effet PPDDL e induit une distribution de probabilité $D(e,s)$ sur des couples $\langle t,r\rangle$ où t est un effet basique (un ensemble de littéraux consistants) et r une récompense. $D(e,s)$ est définie récursivement par :*

Effet e	Distribution $D(e,s)$
$\top$	$\{\langle\emptyset, 0\rangle : 1\}$
x	$\{\langle x, 0\rangle : 1\}$
$\neg x$	$\{\langle \bar{x}, 0\rangle : 1\}$
$\uparrow r$	$\{\langle\emptyset, r\rangle : 1\}$
$\phi \triangleright e'$	$D(e',s)$ si $s \models \phi$ sinon $\{\langle\emptyset, 0\rangle : 1\}$
$p_1\ e_1 \mid \ldots \mid p_k\ e_k$	$\bigcup_{i=1}^{k}\{\langle t,r\rangle : p_i p \mid \langle t,r\rangle : p \in D(e_i,s)\}$ †
$e_1 \wedge \cdots \wedge e_k$	$\{\langle\bigcup_{i=1}^{k} t_i, \sum_{i=1}^{k} r_i\rangle : \prod_{i=1}^{k} p_i \mid \langle t_1, r_1\rangle : p_1 \in D(e_1,s), \ldots, \langle t_k, r_k\rangle : p_k \in D(e_k,s)\}$

† *Pour les effets probabilistes on suppose $\{\langle t,r\rangle : p\} \cup \{\langle t,r\rangle : p'\} = \{\langle t,r\rangle : p + p'\}$.*

À partir de cette définition, il est donc possible de retrouver les fonctions de récompense et de transition des actions d'un PDM :

$$P(s'|s,a) = \sum \{p|\langle t, r\rangle : p \in D(e_a, s),\ \mathit{apply}(t,s) = s'\}$$

$$R(s,a) = \mathop{\mathbf{E}}_{\langle t,r\rangle \sim D(e_a,s)}[r] = \sum_{\langle t,r\rangle : p \in D(e_a,s)} p \cdot r$$

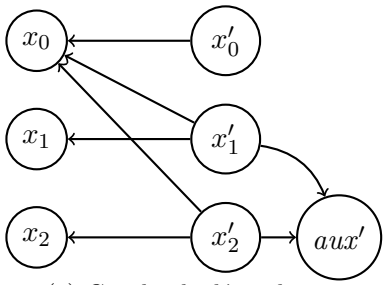
Plus généralement, la notion de distribution induite par l'effet d'une action a permet de décrire l'application de l'opérateur de Bellman à une fonction de valeur V de la manière suivante :

$$\begin{aligned}(\mathfrak{B}^a V)(s) &= \mathop{\mathbf{E}}_{\langle t,r\rangle \sim D(e_a,s)}[r + \gamma V(\mathit{apply}(t,s))] \\ &= \sum_{\langle t,r\rangle : p \in D(e_a,s)} p \cdot (r + \gamma V(\mathit{apply}(t,s)))\end{aligned} \tag{2.2}$$

Étant donné une représentations d'un PDM sous la forme d'actions PPDDL il est alors possible de construire la distribution induite de ces actions et de calculer une politique en utilisant l'équation (2.2) dans un algorithme d'itération de valeur. Cependant, les effets conjonctifs peuvent conduire à une distribution induite de taille exponentielle en la description des actions. Dans le chapitre 5 nous présentons une méthode de calcul de politique qui exploite au maximum la compacité de PPDDL afin d'éviter de calculer et de manipuler directement des effets PPDDL pour le calcul de valeur d'action, sans passer par la distribution induite.

2.6.3 Traduction en réseaux bayésiens dynamiques

Un PDM décrit en PPDDL peut être traduit en réseaux bayésiens dynamiques au prix d'une augmentation polynomiale de la taille de représentation [Younes et Littman, 2004]. Informellement, on introduit des *variables auxiliaires multivaluées* pour chaque effet probabiliste. Chaque valeur possible de cette variable représente alors l'occurrence d'un effet en particulier. Ces variables n'ont pas de parent dans le graphe de dépendances, ce sont au contraire les variables modifiées par ces effets qui vont en dépendre. La table de probabilités conditionnelles associée à ces variables indique la probabilité de l'effet correspondant à chaque valeur possible. Un exemple est donné sur la figure 2.7 pour l'effet $x_0 \triangleright (0{,}4\ x_1 | 0{,}6\ \neg x_2)$. Si la variable auxiliaire aux' est à 0 cela signale que l'effet x_1 s'est produit, sinon il s'agit de l'effet $\neg x_2$. Le symbole $*$ dans les tables de probabilités indique que la valeur de la variable considérée n'influe pas sur la probabilité.



(a) Graphe de dépendances

x_0	$P(x'_0 = 1)$
0	0
1	1

(b) $CPT(x'_0)$

x_0	x_1	aux'	$P(x'_1 = 1)$
0	0	$*$	0
0	1	$*$	1
1	0	0	0
1	0	1	1
1	1	$*$	1

(c) $CPT(x'_1)$

x_0	x_2	aux'	$P(x'_2 = 1)$
0	0	$*$	0
0	1	$*$	1
1	0	0	0
1	$*$	1	0
1	1	0	1

(d) $CPT(x'_2)$

$P(aux' = 1)$
0,6

(e) $CPT(aux')$

FIGURE 2.7 – Représentation en DBN de l'effet $x_0 \triangleright (0{,}4\ x_1 | 0{,}6\ \neg x_2)$.

Deuxième partie

Résolution compacte

Introduction de la partie

Plutôt que de représenter un PDM en extension, nous avons vu dans la partie précédente qu'il existe des représentations compactes, ou en intension, des actions et des fonctions de récompense.

Au delà de ces bénéfices purement représentationels, il est primordial d'exploiter cette compacité de représentation pour améliorer l'efficacité du calcul de fonctions de valeurs et de politiques. En effet, l'existence de représentations compactes pour certains problèmes suggère l'existence de régularités dans les fonctions de valeur et les politiques. Il arrive souvent que des ensembles d'états soient équivalents, que ce soit en termes de récompense espérée (valeur identique) ou en termes de décision (politique identique). De plus, ces groupes d'états exhibent le plus souvent des propriétés communes, comme par exemple le fait que seules les valeurs d'un petit nombre de variables d'état permettent de les caractériser.

Cette partie est consacrée aux algorithmes de résolution compacte des PDM. Ces algorithmes, directement basés sur l'itération de valeur ou de politique, construisent incrémentalement des représentations en intention des fonctions de valeur sous forme de partitions de l'espace d'états. Au fur et à mesure de la résolution, les raffinements nécessaires sont introduits dans ces partitions afin de les maintenir aussi petites que possibles.

Dans cette partie, nous présenterons en premier lieu une partie des approches existantes traitant la résolution factorisée de PDM représentées en réseaux bayésiens dynamiques ainsi que des préliminaires sur les techniques approchées et du premier ordre. Ensuite, nous donnons nos deux contributions, une première traitant les opérateurs STRIPS probabilistes dans un cadre de résolution approchée, et une seconde pour la résolution optimale de PDM décrits en PPDDL.

Chapitre 3

Méthodes de résolution de PDM compacts

Sommaire

3.1 Programmation Dynamique Structurée

Les techniques d'optimisation de PDM factorisés reposent sur la *programmation dynamique structurée*. Ces algorithmes réalisent une itération de valeur en manipulant des représentations compactes de fonctions de valeur, telles que des arbres de décision pour les algorithmes SPI et SVI [Boutilier *et al.*, 1995, Boutilier *et al.*, 2000], ou des ADD pour l'algorithme SPUDD [Hoey *et al.*, 1999]. En contraste à ces algorithmes, il existe les approches de réduction de modèle [Dearden et Boutilier, 1997, Dean et Givan, 1997, Dean *et al.*, 1997, Givan *et al.*, 2003]. Celles-ci prennent en entrée une description compacte d'un PDM (sous forme de STRIPS probabiliste ou de DBN) pour en déduire un PDM énuméré plus simple qui peut être utilisé pour résoudre le problème original. Les algorithmes de programmation dynamique structurée sont en fait équivalents à appliquer une réduction de modèle puis résoudre le PDM réduit [Dean et Givan, 1997]. Une étude unifiée des différentes techniques de minimisation de modèle est donnée dans [Li *et al.*, 2006].

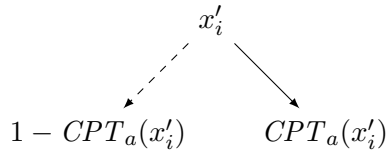
3.1.1 Représentation compacte des actions

Tous les algorithmes énoncés ci-dessus utilisent des actions représentées par des réseaux bayésiens dynamiques. Pour chaque action a et chaque variable $x'_i \in \mathcal{X}'$ ces algorithmes manipulent une représentation compacte des tables de probabilité conditionnelles $CPT_a(x'_i)$. Chacune de ces tables peuvent en effet être représentées par un arbre de décision ou un ADD sur les variables $Parents_a(x'_i)$. Ces techniques permettent d'exploiter les régularités dans ces tables.

Plus précisément, ces algorithmes construisent les représentations compactes $T_a(x'_i)$ des distributions de probabilités $P_a(x'_i|Parents_a(x'_i))$. Ces distributions sont des fonctions dont le domaine est l'ensemble des affectations aux variables $\{x'_i\} \cup Parents_a(x'_i)$. Par exemple, pour une représentation en ADD, $T_a(x'_i)$ est défini comme :

$$T_a(x'_i) = \text{ITE}(x'_i, CPT_a(x'_i), 1 - CPT_a(x'_i))$$

Pour les arbres de décision on utilisera

$T_a(x'_i) =$ 

Le choix entre les arbres de décision et les ADD dépend de deux critères importants et antagonistes. En effet, l'usage des ADD restreint l'application à des problèmes comportant des variables propositionnelles uniquement. Malgré leur efficacité limitée, les arbres de décision ont l'avantage de proposer une représentation naturelle pour les variables portant sur un domaine de taille supérieure à 2. Bien qu'il soit toujours possible d'encoder une variable n-aire à l'aide de $\lceil \log_2 n \rceil$ variables propositionnelles pour profiter de l'efficacité des ADD, cette augmentation du nombre de variables à traiter peut nuire à la compacité et donc à l'efficacité des algorithmes. Cette perte de compacité peut s'expliquer du fait que la sémantique (et donc la structure) apportée par cette variable est « dispersée » sur plusieurs variables qui, une fois considérées individuellement, n'apportent aucune information sur la structure. Cet encodage a aussi pour contrepartie d'introduire des états interdits. Pour chaque variable n-aire et son encodage en $\lceil \log_2 n \rceil$ variables binaires, il y a $2^{\lceil \log_2 n \rceil} - n$ combinaisons de valeurs pour ces variables qui ne représentent aucun état du problème original. Un algorithme de résolution factorisé devra cependant calculer une fonction de valeur et une politique pour ces états fictifs et par conséquent perdre inutilement du temps.

3.1.2 Représentation compacte de la fonction de récompense

La fonction de récompense R est quant à elle une fonction de $2^{\mathcal{X}}$ dans $\mathbb{R}$. Elle peut être décrite de diverses façons, soit de manière directe, soit comme la somme de plusieurs fonctions $R_1, \ldots, R_k$. Cette seconde description de la récompense a l'avantage d'être très compacte tout en gardant sa sémantique explicite. En fonction de l'algorithme utilisé, ces fonctions sont représentées elles aussi sous forme d'arbre de décision ou d'ADD.

3.1.3 Calcul de politiques

L'élément central à ces approches est d'opérer la *régression d'action* d'une façon qui exploite la compacité des fonctions de valeur. Plus précisément, cette étape consiste à calculer le résultat de l'application de l'opérateur de Bellman $\mathfrak{B}^a$ à une fonction de valeur compacte V pour obtenir la valeur d'action $Q(\cdot,a) = \mathfrak{B}^a V$ *en préservant au maximum la compacité des résultats intermédiaires.*

$$Q(s,a) = (\mathfrak{B}^a V)(s) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a)V(s')$$

Le calcul d'une représentation structurée de $Q(\cdot,a)$ à partir d'une fonction de valeur $V : 2^{\mathcal{X}} \rightarrow \mathbb{R}$ comporte quatre étapes :

1. Transformer V en une fonction V' des variables post-action $\mathcal{X}'$, car V' représente la valeur des états *après* l'exécution de l'action a. Chaque variable x_i de V est donc remplacée par x'_i pour obtenir V'.
2. V' est alors successivement multipliée par les fonctions $T_a(x'_i)$ pour $i = 1,\ldots,n$. La fonction W obtenue dépend à la fois des variables post-action (introduites par V') et des variables pré-action introduites par les fonctions T_a.
3. L'espérance $\sum_{s' \in \mathcal{S}} P(s'|s,a)V(s')$ est alors calculée en éliminant les variables post-action de W. Ces variables sont éliminées en sommant sur toutes les variables post-action. Dans le cas d'un arbre de décision, chaque nœud x'_i est remplacé par la somme de ses sous-arbres. Pour les ADD cela revient à calculer la fonction :

   $$\exists x'_1,\ldots,x'_n \left[T_a(x'_1) \times \cdots \times T_a(x'_n) \times V'\right]$$

   L'intuition justifiant cette étape est que la valeur pour x'_i a déjà été pondérée par sa probabilité lors de l'étape précédente, il faut donc sommer sur les différentes valeurs de cette variable pour obtenir une espérance sur les états post-action. Dans le cas où le DBN de l'action a ne comporte pas d'arcs synchrones, l'expression :

   $$\exists x'_1 \left[T_a(x'_1) \times \exists x'_2 \left[T_a(x'_2) \times \ldots \exists x'_n \left[T_a(x'_n) \times V'\right] \ldots\right]\right]$$

   est équivalente. Cet ordre d'opération permet *pour certains problèmes* de limiter la taille des résultats intermédiaires et donc d'accélérer le calcul, notamment quand $T_a(x'_1) \times \cdots \times T_a(x'_n)$ est peu compact.
4. Finalement, cette espérance est multipliée par le facteur γ et la fonction de récompense est ajoutée pour obtenir une représentation compacte de $Q(\cdot,a)$.

La suite de l'algorithme reste alors la même que pour l'itération de valeur : la nouvelle fonction de valeur est le maximum sur les actions des fonctions $Q(\cdot,a)$ et l'algorithme continue jusqu'à convergence.

Une politique compacte peut alors être déduite des fonctions $Q(\cdot,a_1),\ldots,Q(\cdot,a_m)$. Pour chaque action a_i, $i = 1,\ldots,m$ on calcule une représentation compacte de la fonction booléenne

Algorithme 4 : Stochastic Planning Using Decision Diagrams (SPUDD)

Input : $\{T_a\}_{a \in \mathcal{A}}$ L'ensemble des ADD de transitions.

Output : π : ADD de politique, chaque état correspond à un indice d'action

$V \leftarrow 0$

répéter

  Remplacer chaque variable x_i présente dans V avec sa version primée x'_i pour obtenir un diagramme V' sur les variables post-action uniquement.

  $W \leftarrow -\infty$

  pour chaque *action a* **faire**

    $Q \leftarrow V'$

    pour chaque *variable prime* x'_i **faire**

      $Q \leftarrow \exists x'_i[T_a(x'_i) \times Q]$

    $Q \leftarrow \gamma \times Q + R_a$

    $W \leftarrow \max(W, Q)$

  converge $\leftarrow ||V - W||_\infty \leq \frac{\epsilon(1-\gamma)}{2\gamma}$

  $V \leftarrow W$

jusqu'à *converge*

`/* Extraction de la politique` π `*/`

$W \leftarrow -\infty$

Remplacer chaque variable x_i présente dans V avec sa version primée x'_i pour obtenir un diagramme V'.

$\pi \leftarrow 0$

pour chaque *action* a_i, $i = 1, \ldots, |A|$ **faire**

  $Q \leftarrow V'$

  pour chaque *variable prime* x'_i **faire**

    $Q \leftarrow \exists x'_i[T_a(x'_i) \times Q]$

  $Q \leftarrow \gamma \times Q + R_a$

  $G \leftarrow Q > W$

  $\pi \leftarrow \max(\pi, G \times i)$

  $W \leftarrow$ `ITE`(G, Q, W) ; `/* Équivalent à` $W \leftarrow \max(Q,W)$ `mais plus efficace. Voir page 22 pour la définition de ITE */`

return π

$\chi_i(s)$ qui vaut 1 dans les états s tels que $Q(s,a_i) = \max_a Q(s,a)$ et 0 dans les autres états. Une politique compacte est alors la fonction :

$$\pi = \max(1 \times \chi_1, \ldots, m \times \chi_m)$$

Ainsi, pour chaque état, $\pi(s)$ correspond à l'indice d'une action optimale dans l'état s. L'algorithme 4 décrit SPUDD, un algorithme d'itération de valeur factorisé. L'algorithme SVI est identique à l'exception que ce sont des arbres de décision qui sont manipulés au lieu de diagrammes de décision algébriques.

Remarque 18 *L'utilisation de* max *dans le calcul de π vient du fait que pour certains états, plusieurs actions peuvent être optimales. L'opérateur* max *permet de ne garder pour ces états qu'une seule action (celle de plus grand indice), en opposition à une somme qui pourrait donner un résultat inconsistant. Un moyen correct de calculer une politique avec une somme serait :*

$$\pi = 1 \times \chi_i + 2 \times (1 - \chi_1) \times \chi_2 + \cdots + m \times (1 - \chi_1 \wedge \cdots \wedge \chi_{m-1}) \times \chi_m$$

3.2 Résolution approchée de PDM factorisés

Les approches factorisées pour la représentation de PDM permettent en effet d'obtenir des descriptions compactes de la fonction de transition et des récompenses. Il n'y a cependant aucune garantie que la structure présente dans la représentation reste présente dans les fonctions de valeur et des politiques [Koller et Parr, 1999, Liberatore, 2002]. Pour pallier à ce problème on peut alors avoir recours aux fonctions de valeurs *approchées* qui permettent alors une représentation compacte. Pour cela on utilise une approximation de la fonction de valeur par des fonctions de base (comme présenté dans la section 1.3). Dans un cadre factorisé, on fait l'hypothèse que ces fonctions de base dépendent seulement d'un petit nombre de variables. La localité de ces fonctions est ensuite exploitée lors de la génération du programme linéaire pour projeter la fonction de valeur dans l'espace linéaire. Puisque nous nous intéressons à la résolution *exacte* dans cette thèse, pour plus de détails nous renvoyons le lecteur à [Schuurmans et Patrascu, 2001, Guestrin *et al.*, 2003].

3.3 Approches relationnelles et du premier ordre

3.3.1 Enjeux

Une approche plus générale aux PDM structurés est d'utiliser des méthodes relationnelles, ou du premier ordre, pour la représentation et le calcul de politiques. Par exemple, le PPDDL permet la représentation de problèmes directement dans le premier ordre, où les états sont des prédicats sur des objets de l'environnement. L'avantage d'une telle représentation est évident. Par exemple, la définition d'un but tel que $\exists o : P(o)$ (« il existe un objet o tel que $P(o)$ est

vrai ») se traduirait en propositionnel par la disjonction $P(o_1) \vee \cdots \vee P(o_k)$ ou chaque objet o_i est donné par la description du domaine.

En évitant de « propositionaliser » le domaine, ces techniques de planification ont pour avantage d'avoir une complexité qui ne dépend que du nombre de prédicats et qui est indépendante du nombre d'objets utilisés pour les instancier. Par conséquent, une politique symbolique est elle aussi indépendante du nombre objets du domaine dans lequel elle est exécutée et peut même s'appliquer pour un nombre *infini* d'objets. Notons que la résolution de PDM relationnels ou du premier ordre constitue une perspective importante pour l'algorithme de résolution RBAB que nous présentons dans le chapitre 5.

Il existe cependant quelques limites aux possibilités offertes par ces approches. Par exemple, en présence de *récompenses universellement quantifiées*, il n'est pas toujours possible d'être indépendant du nombre d'objets pouvant instancier un prédicat. Si, par exemple, on obtient une récompense de 1 dans les états tels que *tous* les objets o_i, $i = 1, \ldots, k$ vérifient un prédicat $P(o_i)$, la valeur espérée sera différente selon le nombre d'objets o_i de l'état tels que $P(o_i)$ est vrai.

3.3.2 Algorithmes

Une grande diversité d'algorithmes ont été proposés pour la résolution de PDM relationnels ou du premier ordre. Ils diffèrent notamment par le langage utilisé pour représenter les actions et les formules ainsi que dans leur représentation des fonctions de valeur et des politiques. Ces approches diffèrent principalement dans leur façon de gérer le compromis entre un langage simple, peu expressif mais facile à manipuler et un langage plus riche, très expressif mais dont la manipulation peut s'avérer très complexe.

Une première série d'approches est basée sur le langage du *Situation Calculus* (SC), un langage du premier ordre pour l'axiomatisation de mondes dynamiques. Celles-ci sont apparues avec les travaux de [Boutilier *et al.*, 2001] qui ont introduit la notion de « First-Order Decision Theoretic Regression » pour le calcul de valeur d'action (Q) depuis une fonction de valeur (V). Ces fonctions de valeur sont représentées par des tables contenant une partition de l'espace d'états sous forme de formules du premier ordre munies chacune d'une valeur associée.

Par la suite, l'algorithme ReBel [Kersting *et al.*, 2004] utilise un principe de régression similaire mais en se restreignant à un langage relationnel plus simple (sans quantification universelle par exemple). La plus grande simplicité de ce langage a permis une évaluation en pratique avec des premiers résultats encourageants. Avec encore un autre langage de représentation d'actions, on trouve FOVIA [Skvortsova et Hölldobler, 2004], qui cette fois travaille en *Probabilistic Fluent Calculus* (PFC), un langage d'action dont l'expressivité est à mi-chemin entre le SC et le langage utilisé par ReBel. [Skvortsova et Hölldobler, 2004] décrivent également comment traduire un domaine PPDDL en PFC ainsi que des évaluations empiriques.

On trouve ensuite toute une série de travaux par Scott Sanner *et al.* sur les actions décrites en Situation Calculus. Notamment avec l'introduction de fonction de valeur approchées par une

combinaison linéaire de fonctions de base (toujours sous formes de tables) et de l'utilisation de la programmation linéaire. Ainsi un premier algorithme, FOALP [Sanner et Boutilier, 2005] utilise ces notions, qui on été par la suite étendues dans [Sanner et Boutilier, 2006]. FOAPI, une version Itération de Politique de FOALP, ainsi qu'une méthode de décomposition des récompenses universelles sont présentées dans [Sanner et Boutilier, 2006]. Avec un langage toujours plus expressif, apparaît la notion de PDM du premier ordre *factorisé*, les fFOMDP, avec l'algorithme fFOLAP [Sanner et Boutilier, 2007]. Les améliorations apportées par les fFOMDP sont notamment la possibilité de représenter des fonctions de récompenses additives ainsi que des transitions factorisées qui peuvent dépendre du nombre d'objets du domaine.

Pour les PDM relationnels, les travaux de [Wang et Joshi, 2008] introduisent les diagrammes de décision du premier ordre (FODD). Ils les utilisent pour obtenir une variante de l'itération de valeur basée sur les outils de [Boutilier *et al.*, 2001] qui manipulent cette fois non pas des tables, mais des FODD pour représenter les fonctions de valeur. Cette représentation est à la fois plus efficace mais nécessite aussi moins d'espace. L'expressivité des FODD n'est pas aussi grande que celle du Situation Calculus mais reste néanmoins plus importante que celle de ReBel.

Chapitre 4

Approximations dans les biais logiques

Sommaire

Nous abordons dans ce chapitre la résolution de PDM représentés en opérateurs STRIPS probabilistes. Nous présentons tout d'abord l'algorithme général et en étudions sa complexité. Ensuite, nous décrivons différentes approches pour l'approximation des actions et de la fonction de récompense. Ces travaux ont été présentés dans [Lesner, 2009, Lesner et Zanuttini, 2010].

4.1 Résolution factorisée

L'algorithme que nous décrivons diffère des algorithmes classiques tels que SPUDD ou SVI sur deux points. Tout d'abord, le problème est défini sous la forme d'actions STRIPS probabilistes. Ensuite la représentation des fonctions de valeur manipulées est différente. Au lieu

d'utiliser des arbres de décision comme SVI ou des ADD comme SPUDD, nous les représentons sous la forme de partitions de l'espace d'états où chaque élément de la partition est une formule. C'est cette représentation avec des formules qui nous permet d'utiliser les techniques d'approximation de connaissances présentées dans la section 2.1.3.

4.1.1 Algorithme

La résolution de PDM ainsi formulés se base sur l'algorithme *d'itération de valeur*, dont le but est de calculer une fonction de valeur optimale. Une fonction de valeur V est représentée de manière compacte sous la forme d'une *partition* : un ensemble de couples (formule, valeur) où les formules sont mutuellement inconsistantes et l'union de leurs ensembles de modèles couvre $2^{\mathcal{X}}$. Ainsi, pour tout état $s \in 2^{\mathcal{X}}$ il existe exactement un couple $(\phi,v) \in V$ tel que $s \models \phi$ et v est sa valeur associée selon V. Par abus de notation nous noterons $V(\phi)$ les éléments de V qui représentent les valeurs possibles selon V pour les états représentés par ϕ :

$$V(\phi) = \{(\psi,v) \in V \mid \phi \wedge \psi \text{ est satisfaisable}\}$$

L'algorithme de résolution se déroule en trois étapes : initialiser une première partition de valeur grossière à partir de la récompense ; raffiner itérativement cette partition et ses valeurs comme le fait l'itération de valeur ; une fois que les valeurs de la partition ont convergé, en déduire la politique associée.

La première étape constitue à créer la partition V^0. Elle est calculée de manière à distinguer toutes les situations de récompense initiale. Notons la fonction de récompense $R = \{(\psi_1,r_1),\ldots,(\psi_{|R|},r_{|R|})\}$ et, pour toute formule ψ, $\psi^0 \equiv \neg\psi$ et $\psi^1 \equiv \psi$. Alors

$$V^0 = \left\{\left(\phi = \psi_1^{\epsilon_1} \wedge \cdots \wedge \psi_{|R|}^{\epsilon_{|R|}} \;, \epsilon_1 r_1 + \cdots + \epsilon_{|R|} r_{|R|}\right) \mid \forall i, \epsilon_i \in \{0,1\}, \phi \text{ est sat.}\right\}$$

Exemple 19 *Soit* $R = \{(x,1),(\neg x \vee y,3)\}$ *une fonction de récompense compacte. Alors* $V^0 = \{(x \wedge y,4),(x \wedge \neg y,1),(\neg x,3)\}$

La partition V^0 est la plus grossière possible permettant de distinguer les états ayant la même récompense immédiate, aux coïncidences près. À chaque itération de la résolution, ce seront les formules de V^0 qui seront raffinées selon la partition de valeur de l'itération précédente pour former la nouvelle partition. Le raffinement se déroule comme suit : pour chaque formule ϕ de V^0, calculer les ensembles d'états résultant de l'application de tous les effets de chaque action sur les états représentés par ϕ. Si, pour les états d'un ensemble ainsi calculé, les valeurs diffèrent selon la fonction de valeur de l'itération précédente, alors cet ensemble est décomposé en autant de sous-ensembles disjoints qu'il y a de valeurs possibles. Ces opérations sont réalisables de manière compacte, en maintenant la représentation des états sous la forme de formules dont les modèles ne sont pas énumérés.

Par abus de notation, la formule (l'ensemble d'états) résultant de l'application d'un effet e sur une formule ϕ est notée $\mathit{apply}(e,\phi)$. Elle est telle que :

$$\mathcal{M}(\mathit{apply}(e,\phi)) = \{\mathit{apply}(e,s) \mid s \in \mathcal{M}(\phi)\}$$

Lemme 20 *Pour toute conjonction $\phi_1 \wedge \phi_2$, l'application d'un effet e implique la conjonction des applications :*

$$\mathit{apply}(e, \phi_1 \wedge \phi_2) \models \mathit{apply}(e,\phi_1) \wedge \mathit{apply}(e,\phi_2)$$

Démonstration. Par définition de l'implication logique, il faut que les modèles de la partie de gauche soient inclus dans ceux de la partie de droite. Prenons $s \in \mathcal{M}(\mathit{apply}(e,\phi_1 \wedge \phi_2))$, s est le résultat de l'application de s à un modèle commun à ϕ_1 et ϕ_2 ; c'est-à-dire $\exists m$, $m \in (\mathcal{M}(\phi_1) \cap \mathcal{M}(\phi_2))$ tel que $s = \mathit{apply}(e,m)$.

En appliquant maintenant l'effet e directement aux modèles de ϕ_1 et de ϕ_2 on obtient : $s \in (\{\mathit{apply}(e,m_1) \mid m_1 \in \mathcal{M}(\phi_1)\} \cap \{\mathit{apply}(e,m_2) \mid m_2 \in \mathcal{M}(\phi_2)\}))$. Ce qui signifie :

$$s \in \mathcal{M}(\mathit{apply}(e,\phi_1) \wedge \mathit{apply}(e,\phi_2))$$

□

Proposition 21 *L'application d'un effet peut être calculée via* l'oubli de variables *(définition 4) :*

$$\mathit{apply}(e,\phi) \equiv e \wedge \mathit{Oubli}(\mathrm{Var}(e), \phi)$$

Démonstration. Soit $\mathcal{L}(V) = \{v,\neg v \mid v \in V\}$ l'ensemble des littéraux pouvant être formés sur un ensemble de variables V. On a :

$$\begin{aligned}\mathcal{M}(\mathit{apply}(e,\phi)) &= \{(m \setminus \mathcal{L}(\mathrm{Var}(e))) \cup e \mid m \in \mathcal{M}(\phi)\} \\ &= \{m \cup e \mid m \in \mathcal{M}(\mathit{Oubli}(\mathrm{Var}(e),\phi))\} \\ &= \mathcal{M}(e \wedge \mathit{Oubli}(\mathrm{Var}(e), \phi))\end{aligned}$$

□

Raffiner une formule ϕ selon une partition de valeur V et une action a consiste à calculer un ensemble de formules $\phi \wedge \delta_1, \ldots, \phi \wedge \delta_k$ partitionnant ϕ telles que la Q-valeur selon a des états représentés par chaque $\phi \wedge \delta_i$ soit unique.

Lemme 22 *Soient une formule ϕ, un effet e et une partition de valeurs V. Alors, pour tout couple $(\psi_i,v_i) \in V(\mathit{apply}(e,\phi))$ on a :*

$$V(\mathit{apply}(e,\mathit{Oubli}(\mathrm{Var}(e),\psi_i \wedge e) \wedge \phi)) = \{(\psi_i,v_i)\}$$

Démonstration. Soit δ tel que $\delta = \mathit{Oubli}(\mathrm{Var}(e),\psi_i \wedge e)$. Alors d'après le lemme 20 nous avons $\mathit{apply}(e,\phi \wedge \delta) \models \mathit{apply}(e,\phi) \wedge \mathit{apply}(e,\delta)$. Or, $\mathit{apply}(e,\delta) \equiv \delta \wedge e$ car $\mathrm{Var}(\delta) \cap \mathrm{Var}(e) = \emptyset$ et $\delta \wedge e \equiv \psi_i \wedge e$ par définition de δ. Finalement, $\mathit{apply}(e,\phi) \wedge \mathit{apply}(e,\delta) \equiv \mathit{apply}(e,\phi) \wedge \delta \wedge e \equiv \mathit{apply}(e,\phi) \wedge \psi_i \wedge e \models \psi_i$, donc $\mathit{apply}(e,\phi \wedge \delta) \models \psi_i$ et ψ_i est alors la seule formule de V satisfaisable avec $\mathit{apply}(e,\phi \wedge \delta)$. □

À partir du lemme 22 nous pouvons construire l'algorithme 5 qui, étant données une formule ϕ, une partition de valeurs V et une condition d'action, partitionne ϕ comme ci-dessus. Il procède en appliquant récursivement chaque effet à ϕ. Si, après une application d'un effet e sur une formule φ, plusieurs valeurs sont possibles selon V, il calcule une formule de *distinction* δ_l pour chacune de ces valeurs, ainsi les états résultant de l'application de e à $\phi \wedge \delta_l$ ont tous la même valeur selon V.

Algorithme 5 : Raffiner

Entrées : V : partition de valeur
Entrées : ϕ : formule partitionnant V^0
Entrées : c_i^a : condition
si $\phi \wedge c_i^a$ *est NON SAT* **alors retourner** $\emptyset$
$I \leftarrow \{(\phi \wedge c_i^a, 0)\}$
pour chaque *effet* $e \in E_i^a$ **faire**
  $I' \leftarrow \emptyset$
  pour chaque $(\phi', v) \in I$ **faire**
    pour chaque $(\psi, v') \in V(apply(e, \phi'))$ **faire**
      $\delta_l \leftarrow Oubli(\mathrm{Var}(e), \psi \wedge e)$
      `/* Si la disjonction est disponible et/ou efficace pour la classe de formules considérée, on peut procéder à une fusion (par disjonction) des formules ayant la même valeur */`
      $I' \leftarrow I' \cup \{(\phi' \wedge \delta_l,\ v + P_i^a(e) \times v')\}$
  $I \leftarrow I'$
retourner I

Parfois l'algorithme 5 peut produire différentes formules avec des valeurs identiques. Dans ce cas, il est possible de fusionner de telles formules grâce à une disjonction pour réduire au maximum la taille de la partition résultat. Il faut cependant s'assurer que la disjonction est disponible dans la classe de formules considérée et si c'est le cas qu'elle est efficace. Il est en effet parfois plus raisonnable de laisser deux formules simples plutôt que d'utiliser leur disjonction qui peut être de taille sensiblement plus importante.

Exemple 23 (raffinement de partition par l'algorithme 5) *La figure 4.1.1 montre un exemple de partition initiale V^0 et une partition V^{n-1}. Supposons $c_i^a = x$, un unique effet $e = \neg x \wedge z$. Soit $\phi = x$. La formule $c_i^a \wedge \phi$ étant satisfaisable nous pouvons appliquer e : $apply(e, c_i^a \wedge \phi) = \neg x \wedge z$. On a $V^{n-1}(\neg x \wedge z) = \{(\neg x \wedge \neg y \wedge z, 10), (\neg x \wedge y, 4)\}$, il y a plus d'une valeur possible, il faut donc raffiner $c_i^a \wedge \phi$ en deux formules $c_i^a \wedge \phi \wedge \delta_1$ et $c_i^a \wedge \phi \wedge \delta_2$:*

- *$\delta_1 = Oubli(\mathrm{Var}(e), \neg x \wedge \neg y \wedge z \wedge e) \equiv \neg y$: nous obtenons $V^{n-1}(apply(e, c_i^a \wedge \phi \wedge \delta_1)) = \{(\neg x \wedge \neg y \wedge z, 10)\}$, il y a bien une unique valeur pour ces états. La Q-valeur $\mathfrak{B}^a V^{n-1}$ contiendra donc une formule $c_i^a \wedge \phi \wedge \delta_1$ munie de la valeur $2 + \gamma 10$.*

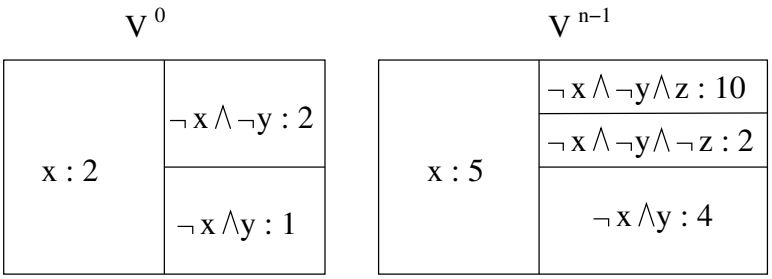


FIGURE 4.1 – Exemple de partitions de valeur

– $\delta_2 = \mathit{Oubli}(\mathrm{Var}(e), (\neg x \wedge y \wedge e) \equiv y$ *: nous obtenons* $V^{n-1}(\mathit{apply}(e, c_i^a \wedge \phi \wedge \delta_2)) = \{(\neg x \wedge y, 4)\}$, $\mathfrak{B}^a V^{n-1}$ *contiendra* $(c_i^a \wedge \phi \wedge \delta_2, 2 + \gamma 4)$.

En appliquant pour chaque action l'algorithme 5 sur chacune des formules partitionnant V^0, nous obtenons une partition représentant la Q-valeur $\mathfrak{B}^a V^{n-1}$ de chaque action. De manière similaire à l'itération de valeur, il faut maintenant maximiser les Q-valeurs, c'est-à-dire calculer une partition de valeurs maximales.

Pour le calcul d'une telle partition maximale nous proposons deux algorithmes différents. Le premier (algorithme 6) calcule une partition maximale en prenant les intersections de tous les éléments des Q-valeurs données, en gardant pour chacune d'elles la valeur maximale. Pour ce faire il utilise *seulement* la conjonction de formules et le test de satisfaisabilité. Cependant, il génère un grand nombre d'intersections candidates dont beaucoup se révèlent être vides.

Le second algorithme (algorithme 7) utilise à la fois la disjonction, la négation, et la conjonction sur les formules. Il choisit d'abord l'ensemble d'états ayant la Q-valeur maximale parmi toutes les partitions. Il choisit ensuite le second et n'en retient que la partie non couverte par le premier et ainsi de suite, jusqu'à couvrir toute la partition demandée (paramètre Φ de l'algorithme 7).

L'algorithme 8 calcule donc la fonction de valeur optimale sous la forme d'une partition. Avant de raffiner V^0 pour calculer une Q-valeur, il initialise celle-ci en donnant une récompense de $-\infty$ pour les états où l'action courante n'est pas exécutable. Ainsi, cette action ne sera pas choisie lors de la maximisation. Après cette étape, les états où aucune action n'est possible ont une valeur de $-\infty$ qu'il faut remplacer par la récompense immédiate. Notons que pour plus d'efficacité nous pouvons maximiser seulement les raffinements de chaque formule ϕ_i de V^0 selon chaque action puisque les raffinements des $\phi_j, j \neq i$ sont nécessairement insatisfaisables avec ceux de ϕ_i.

4.1.2 Génération d'une politique

Une politique est simplement représentée par un ensemble de $|\mathcal{A}|$ formules, chacune associée à une action. L'action d'une telle formule est optimale pour tous les états la satisfaisant.

Algorithme 6 : Maximiser

Entrées : $\mathcal{Q}$: ensemble de Q-valeurs
Entrées : Φ : couverture de $\mathcal{Q}$
Prendre Q_{a_0}, la Q-valeur de l'action a_0 dans $\mathcal{Q}$
$V \leftarrow \{(\phi, v, a_0) \mid (\phi,v) \in Q_{a_0}\}$
pour chaque *action* $a_i \in \mathcal{A}$, $i > 0$ **faire**
  $V' \leftarrow \emptyset$
  Prendre Q_{a_i} la Q-valeur de a_i dans $\mathcal{Q}$
  pour chaque $(\phi,v) \in Q_{a_i}$ **faire**
    pour chaque $(\phi',v', a) \in V$ **faire**
      $\psi \leftarrow \phi \wedge \phi'$
      si ψ *est SAT* **alors**
        si $v > v'$ **alors**
          $V' \leftarrow V' \cup \{(\psi, v, a_i)\}$
        sinon
          $V' \leftarrow V' \cup \{(\psi, v', a)\}$
  $V \leftarrow V'$
$\pi_a \leftarrow 0 \; \forall a \in \mathcal{A}$
$M \leftarrow \emptyset$
pour chaque $(\phi,v,a) \in V$ **faire**
  $\pi_a \leftarrow \pi_a \vee \phi$
  `/* Note : pour éviter la disjonction, on peut garder la politique telle qu'elle est dans` V`. */`
  $M \leftarrow M \cup \{(\phi,v)\}$
$\Pi \leftarrow \{\pi_a\}_{a \in \mathcal{A}}$
retourner (M, Π)

Une politique optimale pour les Q-valeurs courantes peut être calculée au cours de la maximisation, comme montré dans les algorithmes 6 et 7. À tout moment dans l'exécution de ces algorithmes, V est correcte (mais incomplète). On remarque en effet que seuls des éléments maximaux y sont insérés. Sachant que chaque élément inséré provient d'une unique action, il est possible de mettre directement à jour la formule de la politique de cette action.

4.1.3 Complexité

Nous cherchons à mesurer la complexité de notre algorithme en termes d'opérations logiques : test de satisfaisabilité, oubli de variables et conjonction.

Notons e, le nombre maximal d'effets par condition et $|V|$ la taille de la partition de valeur courante. Pour l'algorithme 5, au plus $O(|V|^e)$ oublis de variables sont nécessaires. Sachant que vérifier les valeurs pour une formule donnée nécessite $O(|V|)$ tests de satisfaisabilité, il faudra

Algorithme 7 : Maximiser

Entrées : $\mathcal{Q}$: ensemble de Q-valeurs
Entrées : Φ : couverture de $\mathcal{Q}$
$V \leftarrow \emptyset$
$\Psi \leftarrow 0$ `// rien n'est couvert`
$\pi_a \leftarrow 0 \; \forall a \in \mathcal{A}$
tant que $\Psi \not\equiv \Phi$ **faire**
  $(\phi, v)_a \leftarrow$`retirer-max` $(\mathcal{Q})$
  si $\phi \models \neg\Psi$ `//` ϕ `est totalement non couvert`
  alors
    $V \leftarrow V \cup \{(\phi, v)\}$
    $\Psi \leftarrow \Psi \vee \phi$
    $\pi_a \leftarrow \pi_a \vee \phi$
  sinon si $\phi \not\models \Psi$ `//` ϕ `est partiellement non couvert`
  alors
    $V \leftarrow V \cup \{(\neg\Psi \wedge \phi, v)\}$
    $\Psi \leftarrow \Psi \vee \phi$
    $\pi_a \leftarrow \pi_a \vee (\neg\Psi \wedge \phi)$
retourner $(V, \{\pi_a\}_{a \in \mathcal{A}})$

Algorithme 8 : Itération de valeur sur un espace d'états factorisé

Initialiser V^0 avec la récompense
$n \leftarrow 0$
répéter
  $n \leftarrow n + 1$
  $V^n \leftarrow \emptyset$
  pour chaque $(\phi, r) \in V^0$ **faire**
    pour chaque $a \in \mathcal{A}$ **faire**
      $Q_{a,\phi} \leftarrow \{(\neg(c^a_1 \vee \cdots \vee c^a_{|a|}) \wedge \phi, -\infty)\}$
      pour chaque $c^a_i \in a$ **faire**
        $Q_{a,\phi} \leftarrow Q_{a,\phi} \cup \{(\psi, r + \gamma \cdot v) \mid (\psi, v) \in$ `Raffiner`$(V^{n-1}, \phi, c^a_i)\}$
    $(V_\phi, \Pi_\phi) \leftarrow$ `Maximiser`$(\{Q_{a,\phi}\}_{a \in \mathcal{A}}, \phi)$
    Remplacer les $-\infty$ de V_ϕ par r
    $V^n \leftarrow V^n \cup V_\phi$
    $\Pi_a \leftarrow \Pi_a \vee \pi_a, \; \pi_a \in \Pi_\phi, \; \forall a \in \mathcal{A}$
jusqu'à V^n satisfaisante ou horizon atteint

dans le pire des cas $O(|V|^e)$ tests SAT. On peut montrer qu'il s'agit du nombre de raffinements à

générer dans le pire des cas. L'exemple 24 montre un cas particulier où il faut générer un nombre exponentiel de distinctions.

Exemple 24 *Supposons que la partition courante V^{n-1} contienne :*

$$\neg x_1 \wedge x_2 \wedge x_3 \wedge \cdots \wedge x_k \wedge y_1 : 1 \quad \neg x_1 \wedge x_2 \wedge x_3 \wedge \cdots \wedge x_k \wedge \neg y_1 : 0$$
$$\cdots$$
$$x_1 \wedge x_2 \wedge x_3 \wedge \cdots \wedge \neg x_k \wedge y_k : 2^k \quad x_1 \wedge x_2 \wedge x_3 \wedge \cdots \wedge \neg x_k \wedge \neg y_k : 0$$

Pour chaque i, tous les x_j, $j \neq i$ sont positifs sauf x_i, et une puissance de 2 comme valeur selon que y_i est vrai ($V^{n-1} = 2^i$) ou faux ($V^{n-1} = 0$). On a donc $|V^{n-1}| = 2k + 1$ en comptant une dernière formule qui regroupe tous les autres états et de valeur quelconque.

Soit maintenant une action a avec une seule condition toujours vraie, et k effets : x_1, x_2, …, x_k, chacun avec la même probabilité $1/k$. Supposons $(x_1 \wedge x_2 \wedge \cdots \wedge x_k, v) \in V^0$. Alors on peut voir que toute combinaison de y_i donne une valeur espérée différente : $Q_a(x_1 \wedge \cdots \wedge x_k \wedge y_1^{\epsilon_1} \wedge \cdots \wedge y_k^{\epsilon_k}) = v + \gamma \times 1/k \times (\sum \epsilon_i 2^i), \forall \epsilon_i \in \{0,1\}$.

Il faut donc bien 2^k distinctions, à comparer à $2k + 1$ pour $|V^{n-1}|$ et k pour e.

Malgré des complexités dans le pire des cas exponentielles, l'objectif de la résolution compacte est de tirer parti du fait que e est typiquement petit par rapport au nombre de variables $|\mathcal{X}|$ et que $2^{|\mathcal{X}|}$ états sont à énumérer pour une résolution non compacte.

L'algorithme 7 (maximiser les Q-valeurs) demande quant à lui $O(|Q||\mathcal{A}|)$ tests de satisfaisabilité. La version ne nécessitant que la conjonction (algorithme 6) requiert quant à elle $O(|Q|^{|\mathcal{A}|})$ tests SAT.

Le théorème suivant nous permet de tirer parti de la faible complexité des opérations logiques [3] de certaines classes de formules en section 4.2.

Théorème 25 *Si les formules de V^0 et les conditions d'action sont dans une classe $\mathcal{C}$ stable pour la conjonction et l'oubli de variables, alors les formules de V^n sont dans $\mathcal{C}$ pour tout horizon n, lorsque l'algorithme 8 maximise avec l'algorithme 6.*

Démonstration. Par construction, l'algorithme exécute pour seules manipulations de formules la conjonction et l'oubli de variables. Si V^0 et les conditions sont dans une classe stable pour ces opérations, alors l'algorithme est stable pour cette classe. □

4.2 Approximation de problèmes

Nous avons vu dans la section précédente que la complexité de notre algorithme repose sur une grande quantité d'opérations logiques telles que le test de satisfaisabilité et l'oubli de variables. Dans le cas général, ces deux opérations sont difficiles, cependant, certaines classes

3. Nous ignorons pour le moment la formule $\neg(c_1^a \vee \cdots \vee c_{n_a}^a)$: cf. discussion en section 4.5.

de formules permettent de les réaliser en temps polynomial. Ces classes *efficaces* étant incomplètes, nous proposons donc une méthode *d'approximation* de problèmes factorisés dans de telles classes. Le théorème 25 nous garantit de rester dans la même classe si celle-ci est stable pour la conjonction et l'oubli de variables.

4.2.1 Encadrement de formules propositionnelles

Comme présenté dans [Selman et Kautz, 1996], toute formule en logique propositionnelle peut être approchée dans des classes incomplètes telles que les formules de Horn, 2-CNF ou encore affines. L'approximation consiste à encadrer une formule initiale par deux formules, l'une étant logiquement plus forte et l'autre plus faible. Pour une formule ϕ exprimée dans un langage quelconque, elle peut être encadrée par deux formules ϕ_{lb} et ϕ_{ub} telles que $\phi_{lb} \models \phi \models \phi_{ub}$. Du point de vue des modèles (les états représentés par une formule) nous avons $\mathcal{M}(\phi_{lb}) \subseteq \mathcal{M}(\phi) \subseteq \mathcal{M}(\phi_{ub})$. La formule ϕ_{lb} (respectivement ϕ_{ub}) est un *minorant* (resp. *majorant*) de ϕ.

Nous proposons d'approximer des PDM factorisés par des PDM utilisant une classe de formules efficace pour notre algorithme de résolution, afin d'obtenir un encadrement facilement calculable de la valeur optimale du problème initial, dans l'esprit des travaux de [Selman et Kautz, 1996] qui utilisent les majorants et minorants pour répondre à des requêtes d'implication et de satisfaisabilité.

Nous proposons donc d'encadrer un PDM factorisé quelconque P par deux problèmes approchés $P_\downarrow$ et $P_\uparrow$ dont les fonctions de valeurs optimales respectives ${V_\downarrow}^n$ et ${V_\uparrow}^n$ satisfont en tout état s et horizon n

$${V_\downarrow}^n(s) \leq V^n(s) \leq {V_\uparrow}^n(s)$$

Dans les sections suivantes nous montrons la construction de $P_\downarrow$ et $P_\uparrow$ par approximation logique de la fonction de récompense et des actions de P.

4.2.2 Encadrement de la fonction de récompense

Dans ce cadre, pour un PDM P et sa fonction de récompense compacte R, le problème approché $P_\downarrow$ est obtenu à partir de P en renforçant les formules de récompense positive et en affaiblissant les négatives (et inversement pour $P_\uparrow$). Intuitivement, la récompense de chaque état de $P_\downarrow$ est plus faible que dans P. Plus formellement, soit $P = (\mathcal{X},\mathcal{A},R)$ un PDM factorisé, $P_\downarrow = (\mathcal{X},\mathcal{A},R_\downarrow)$ est une *approximation inférieure* de P par la récompense si

$$R_\downarrow = \{(\phi_{lb}, r) \mid (\phi, r) \in R, r \geq 0\} \cup \{(\phi_{ub}, r) \mid (\phi, r) \in R, r < 0\}$$

où $\phi_{lb} \models \phi \models \phi_{ub}$. Les *approximations supérieures* $P_\uparrow$ sont définies de manière duale.

Théorème 26 *Soit P un PDM factorisé, $P_\downarrow$ et $P_\uparrow$ des approximations inférieures et supérieures de P. Alors, leurs fonctions de valeur optimales vérifient ${V_\downarrow}^n(s) \leq V^n(s) \leq {V_\uparrow}^n(s)$ pour tout état $s \in \mathcal{S}$ et horizon n.*

Démonstration. Par symétrie, nous montrons seulement $V_{\downarrow}^{\,n}(s) \leq V^n(s)$ car P peut être considéré comme une approximation inférieure de $P_{\uparrow}$. Nous procédons par récurrence sur l'horizon n. On a par définition :

$$\begin{aligned} V_{\downarrow}^{\,0}(s) = R_{\downarrow}(s) = & \sum\{r \mid (\phi_{lb},r) \in R_{\downarrow}, s \models \phi_{lb}, r \geq 0\} \\ + & \sum\{r \mid (\phi_{ub},r) \in R_{\downarrow}, s \models \phi_{ub}, r < 0\} \end{aligned}$$

Or, par définition des bornes, si $s \models \phi_{lb}$, alors $s \models \phi$ et si $s \models \phi$, alors $s \models \phi_{ub}$. Donc

$$\begin{aligned} \{r \mid (\phi_{lb},r) \in R_{\downarrow}, s \models \phi_{lb}, r \geq 0\} &\subseteq \{r \mid (\phi,r) \in R, s \models \phi, r \geq 0\} \\ \{r \mid (\phi_{ub},r) \in R_{\downarrow}, s \models \phi_{ub}, r < 0\} &\supseteq \{r \mid (\phi,r) \in R, s \models \phi, r < 0\} \end{aligned}$$

On en déduit finalement $V_{\downarrow}^{\,0}(s) \leq V^0(s)$.

À horizon $n+1$: en utilisant l'hypothèse de récurrence $V_{\downarrow}^{\,n}(s) \leq V^n(s)$ et $R_{\downarrow}(s) \leq R(s)$ (montré ci-dessus). On déduit, pour toute action a et avec i tel que la condition $c_i^a \in a$ vérifie $s \models c_i^a$

$$R_{\downarrow}(s) + \gamma \sum_{e \in E_i^a} P_i^a(e) \cdot V_{\downarrow}^{\,n}(\mathit{apply}(e,s)) \;\leq\; R(s) + \gamma \sum_{e \in E_i^a} P_i^a(e) \cdot V^n(\mathit{apply}(e,s)).$$

C'est-à-dire $\mathfrak{B}_{\downarrow}{}^a V^n(s) \leq \mathfrak{B}^a V^n(s)$. D'où, pour tout état s, $V_{\downarrow}^{\,n+1}(s) \leq V^{n+1}(s)$. □

4.2.3 Encadrement des conditions d'actions

Pour un PDM factorisé P muni d'un ensemble d'actions $\mathcal{A}$, les problèmes approchés $P_{\downarrow}$ et $P_{\uparrow}$ sont obtenus à partir de P en remplaçant l'ensemble d'actions $\mathcal{A}$ par des ensembles $\mathcal{A}_{\downarrow} = \{a_{\downarrow} \mid a \in \mathcal{A}\}$ et $\mathcal{A}_{\uparrow} = \{a_{\uparrow} \mid a \in \mathcal{A}\}$, avec $a_{\downarrow} = \{(c_{i\,lb}^a, E_i^a, P_i^a) \mid c_i^a \in \mathcal{C}^a\}$ et $a_{\uparrow} = \{(c_{i\,ub}^a, E_i^a, P_i^a) \mid c_i^a \in \mathcal{C}^a\}$.

Nous nous restreignons dans cette section à des problèmes dont les récompenses sont positives. Intuitivement, il s'agit donc de rendre les actions exécutables dans plus ou moins d'états : si pour un état donné il y a moins d'actions possibles, alors sa valeur ne peut être plus grande, et inversement si plus d'actions sont possibles pour cet état. Dans la suite de cette section, nous allons discuter le cas des approximations inférieures et supérieures séparément.

Dans le cas des approximations inférieures, certains états peuvent n'avoir aucune action possible. Dans ce cas, en conformité avec l'équation 1.2, nous définissons la valeur espérée en cet état comme sa récompense immédiate, quel que soit l'horizon. La preuve suivante tient bien compte de ce cas.

Théorème 27 *Pour un PDM factorisé P dont toutes les récompenses sont positives ou nulles et son approximation inférieure sur les conditions d'actions $P_{\downarrow}$, la relation $V_{\downarrow}^{\,n}(s) \leq V^n(s)$ est vérifiée pour tout état s.*

Démonstration. Après initialisation par la récompense nous avons, pour tout état s, ${V_\downarrow}^0(s) = V^0(s) = R(s)$. Pour toute action a et son approximation $a_\downarrow$ nous avons

$$\{s \in \mathcal{S} \mid \mathit{Eff}(a_\downarrow,s) \neq \emptyset\} \subseteq \{s \in \mathcal{S} \mid \mathit{Eff}(a,s) \neq \emptyset\}$$

C'est-à-dire que $a_\downarrow$ est applicable sur moins d'états que a. L'espérance de récompense pour s en exécutant $a_\downarrow$ est identique à celle en exécutant a, ou nulle (donc inférieure à celle pour a puisque les récompenses sont positives).

Prenons maintenant pour hypothèse de récurrence $V_\downarrow^n(s) \leq V^n(s)$ alors :

$$R(s) + \gamma \sum_{e \in \mathit{Eff}(a_\downarrow,s)} P_i^a(e) {V_\downarrow}^n(\mathit{apply}(e_{i,j}^a,s)) \;\leq\; R(s) + \gamma \sum_{e \in \mathit{Eff}(a,s)} P_i^a(e) V^n(\mathit{apply}(e_{i,j}^a,s))$$

Et donc, pour tout état s, $V_\downarrow^{n+1}(s) \leq V^{n+1}(s)$. □

Notons que pour un problème $P = (\mathcal{S}, \mathcal{A}, T, R)$ et un autre $P_C = (\mathcal{S}, \mathcal{A}, T, R')$ tel que $R'(s) = R(s) + C$, C étant une constante, ainsi que leurs fonctions de valeur respectives V^n et V_C^n à horizon n nous avons l'égalité suivante :

$$V_C^n(s) = C \sum_{i=0}^{n-1} \gamma^i + V^n(s) = C \frac{1-\gamma^n}{1-\gamma} + V^n(s)$$

S'il existe une récompense négative, il est donc possible d'ajouter une constante C telle que :

$$C = -\min\{r \mid (\phi,r) \in R\}$$

à toutes les récompenses afin qu'elles soient positives et donc que le problème rentre dans le cadre d'application du théorème 27. Dans ce cas la politique reste inchangée, mais il faut soustraire $C\frac{1-\gamma^n}{1-\gamma}$ à la fonction de valeur obtenue pour retrouver celle du problème avec les récompenses originales.

Pour la borne supérieure $P_\uparrow$ sur les conditions d'actions d'un problème P, il se peut que $P_\uparrow$ ne soit pas un PDM. En effet, en relaxant les conditions, il se peut que celles-ci ne soient plus mutuellement exclusives. Dans une telle situation, pour une action a et un état s, plusieurs conditions de a peuvent satisfaire s, ce qui aurait pour conséquence que les probabilités de transitions à partir de s auraient une somme supérieure à 1.

Ceci n'est pas nécessairement un obstacle, car ce problème peut être contourné en décomposant les actions : on crée de nouvelles actions en séparant les conditions dont les bornes ne sont pas mutuellement inconsistantes. Ce problème est alors résolu normalement, puis à la génération de la politique, les actions précédemment introduites sont regroupées pour revenir à l'espace d'actions originel.

Avec une preuve similaire à celle du théorème 27 nous avons le résultat suivant :

Théorème 28 *Pour un PDM factorisé P et son approximation supérieure sur les conditions d'actions $P_\uparrow$, telles que les conditions de chaque action de $P_\uparrow$ soient mutuellement exclusives*[4]*, la relation $V^n(s) \leq {V_\uparrow}^n(s)$ est vérifiée pour tout état s.*

4. Dans ce théorème il n'y a pas de restriction sur le signe des récompenses.

4.2.4 Approximation des effets

Il semble naturel qu'après l'approximation de la fonction de récompense et des conditions d'actions, nous nous intéressions au cas des effets. Cependant, de manière générale, il n'est pas possible de déduire un encadrement de la fonction de valeur à partir d'une approximation des effets, comme le montre l'exemple 29.

Exemple 29 *Soit un effet $e = x \wedge \neg y$ et sa borne supérieure $e_\uparrow = x$, et une fonction de valeur initiale V^0 telle que $V^0(xyz) = 1$; $V^0(x\overline{y}\overline{z}) = 2$; $V^0(x\overline{y}z) = 3$; $V^0(xy\overline{z}) = 4$.*

Pour les deux états $\overline{x}yz$ et $xy\overline{z}$ nous obtenons les inégalités de valeur suivantes :

$$V^0(apply(e,\overline{x}yz)) = 3 \quad > \quad V^0(apply(e_\uparrow,\overline{x}yz)) = 1$$
$$V^0(apply(e,xy\overline{z})) = 2 \quad < \quad V^0(apply(e_\uparrow,xy\overline{z})) = 4$$

4.3 Application à la 2-CNF

La classe des formules 2-CNF (des conjonctions de clauses contenant au plus 2 littéraux) est intéressante pour la résolution compacte de PDM. En effet, pour cette classe, le test de satisfaisabilité est linéaire en la taille de la formule [Aspvall *et al.*, 1979]. Pour l'oubli de variables, il suffit de faire la conjonction de tous les impliqués premiers de la formule ne mentionnant pas de variable à oublier [Lang *et al.*, 2003] ; leur nombre étant quadratique pour une formule en 2-CNF, il en résulte que l'oubli est en temps polynomial. Les formules en 2-CNF étant stables pour la conjonction, le théorème 25 garantit que les formules seront maintenues en 2-CNF tout au long de la résolution.

Cette classe n'est pas stable pour la négation, ni la disjonction. Il en résulte que l'initialisation des Q-valeurs de chaque action a dans l'algorithme 8 avec une formule $\phi = \neg(c^a_1 \vee \cdots \vee c^a_{|a|})$ peut poser problème. Il est possible de le contourner en remarquant que ϕ peut être pré-calculée. Cependant, le langage des 2-CNF étant incomplet, il n'existe pas nécessairement de 2-CNF pouvant représenter ϕ. Malgré tout, le langage composé de disjonctions de 2-CNF est complet [Fargier et Marquis, 2008], il est alors possible de calculer l'ensemble des 2-CNF équivalent à ϕ et d'ajouter chacune d'elles dans la Q-valeur avec une valeur de $-\infty$. Pour cela, il faut recouvrir $\mathcal{M}(\phi)$ par un ensemble $\{M_1, \ldots, M_k\}$ tel que chaque M_i soit l'ensemble des modèles d'une formule 2-CNF, ce qui peut être déterminé en testant si M_i est clos par *majorité* [Schaefer, 1978]. Ensuite sur chaque M_i est appliqué un algorithme de *description* [Zanuttini et Hébrard, 2002] qui va calculer une formule 2-CNF ayant pour modèles M_i.

Les définitions suivantes caractérisent les bornes minimales et maximales : celles qui encadrent « au plus près » la formule initiale. C'est en effet ce type de bornes qui est le plus intéressant car elles permettent d'approcher au mieux le problème original.

Définition 30 (Borne inférieure 2-CNF maximale : GLB) *Soit ϕ une formule quelconque. La formule 2-CNF ϕ_{glb} est une borne inférieure maximale de ϕ si et seulement si $\mathcal{M}(\phi_{glb}) \subseteq \mathcal{M}(\phi)$ et il n'existe pas de formule 2-CNF ϕ' telle que $\mathcal{M}(\phi_{glb}) \subset \mathcal{M}(\phi') \subseteq \mathcal{M}(\phi)$.*

Définition 31 (Borne supérieure 2-CNF minimale : LUB) *Soit ϕ une formule quelconque. La formule 2-CNF ϕ_{lub} est une borne supérieure minimale de ϕ si et seulement si $\mathcal{M}(\phi) \subseteq \mathcal{M}(\phi_{lub})$ et il n'existe pas de formule 2-CNF ϕ' telle que $\mathcal{M}(\phi) \subseteq \mathcal{M}(\phi') \subset \mathcal{M}(\phi_{lub})$. Pour une formule ϕ donnée, il existe une unique borne supérieure minimale, possiblement équivalente à ϕ [Del Val, 1995].*

Exemple 32 *Soit la formule $\phi = (x \vee y \vee \neg z) \wedge (z \vee w)$. La formule $(x \vee y) \wedge (z \vee w)$ est une borne inférieure 2-CNF maximale de ϕ et $z \vee w$ en est une borne supérieure 2-CNF minimale. Ces deux formules vérifient bien $(x \vee y) \wedge (z \vee w) \models (x \vee y \vee \neg z) \wedge (z \vee w) \models z \vee w$*

Il existe des algorithmes, décrits dans [Selman et Kautz, 1996], pour calculer ces bornes à partir de formules représentées en CNF, mais leur importante complexité ne permet d'envisager l'approximation de formules du problème de décision que comme une étape de compilation *a priori*. La description des actions représente la dynamique de l'agent et de l'environnement dans lequel il évolue. Sa mission est représentée quant à elle par la fonction de récompense. Le coût de l'approximation des actions peut donc être amorti dans le cas où plusieurs missions sont à effectuer. Chaque nouvelle mission ne requiert alors que l'approximation de la fonction de valeur initiale associée à la récompense décrivant la mission. L'approximation des formules de récompense doit alors se faire dans la classe des termes de longueur 2 pour que V^0 soit en 2-CNF.

4.4 Utilisation « anytime » des bornes inférieures

On peut imaginer compenser la perte de qualité des fonctions de valeur obtenue par approximation inférieure en utilisant plusieurs approximations successivement, en choisissant pour chaque ensemble d'états la plus grande valeur donnée par ces approximations. Les théorèmes 27 et 26 garantissent que cette approche est correcte.

Intuitivement, combiner ainsi des approximations revient à couvrir au mieux les conditions d'action, tout en maintenant des « petites » formules, afin de prendre en compte le plus de spécificités possibles du problème. La propriété *anytime* s'obtient donc en utilisant de plus en plus de bornes inférieures sur la fonction de valeur, en fonction du temps disponible.

Le choix des approximations à utiliser revêt une importance toute particulière. Tout d'abord, pour des raisons d'efficacité de génération des approximations, nous proposons d'utiliser des bornes inférieures non nécessairement maximales. D'autre part, afin de ne pas générer des approximations trop similaires du problème, nous choisissons des approximations distantes les unes des autres.

Dans le cas de la 2-CNF, l'approximation inférieure d'une CNF quelconque s'obtient par *renforcement* des clauses : on retire tout simplement des littéraux de celles-ci pour que leur taille atteigne 2. Avec un tel procédé, on peut générer les approximations dans un ordre lexicographique, choisir les plus distantes selon cet ordre semble être un choix raisonnable pour représenter aux mieux les différents modèles de la formule originale car les formules obtenues auront peu de clauses en commun.

4.5 Résultats expérimentaux

Ne pouvant donner une borne théorique sur l'impact de ces approximations sur la fonction de valeur, nous avons mesuré l'écart de valeur sur des problèmes aléatoires. Nous discutons en premier lieu de la génération aléatoire de problèmes structurés dans notre représentation. Puis nous comparons les fonctions de valeur obtenues avec l'approximation à celles optimales.

4.5.1 Génération aléatoire de PDM structurés

Nous avons conçu un générateur de problèmes structurés aléatoires. Dans un premier temps la taille de l'espace d'états est définie par le nombre n de variables propositionnelles utilisées. Ensuite, des actions sont générées : une formule CNF satisfaisable est générée aléatoirement sur un ensemble de k variables tiré aléatoirement, les clauses sont de taille comprise entre 2 et $\frac{k}{2}$, pour un nombre de clauses égal à k afin d'avoir une quantité relativement importante d'états représentés par cette formule. Cette clause va servir de base aux conditions : elle représente tous les états dans lesquels l'action est exécutable. En fonction du nombre de conditions voulu, t autres variables sont choisies aléatoirement pour former 2^t termes qui, chacun en conjonction avec la clause de base, permettront de partitionner celle-ci.

Pour chacune de ces conditions, un ensemble de e effets munis chacun d'une probabilité est généré. Pour cela, $e-1$ variables sont tirées aléatoirement, dont la moitié proviennent des conditions d'actions générées précédemment. Ce choix de variables permet d'activer les conditions d'autres actions afin de diversifier les politiques et de complexifier le problème. Un premier effet (terme) de taille $e-1$ est généré, ainsi que sa probabilité ; pour les effets suivants, un littéral est retiré de l'effet précédent jusqu'à obtenir l'effet vide. Cette génération se base sur l'hypothèse que les différentes issues d'une action sont les mêmes à la différence près qu'elle peut dans certains cas ne pas fonctionner totalement, ou même n'avoir aucun effet du tout.

Les récompenses sont tout simplement des littéraux, munis d'une récompense aléatoire positive. Le nombre de formules de récompenses est paramétrable et permet de définir combien il y aura de récompenses différentes possibles.

Exemple 33 (génération aléatoire d'une action) *Prenons pour paramètres $n = 10$ variables, $e = 3$ effets, $c = 2$ conditions sur $k = 6$ variables par action.*

1. *Génération d'une CNF aléatoire sur k variables :*

$$\phi = (x_0 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee x_3 \vee \neg x_5)$$

2. *On choisit $\log c$ variables pour partitionner ϕ et obtenir les conditions : ici 1 variable, par exemple x_6. On obtient deux conditions mutuellement inconsistantes :*
   - $C_1 = (x_0 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2) \wedge (x_2 \vee x_4 \vee \neg x_5) \wedge x_6$
   - $C_2 = (x_0 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2) \wedge (x_2 \vee x_4 \vee \neg x_5) \wedge \neg x_6$
3. *Pour chacune des conditions on génère les effets et leurs probabilités :*

– *pour* C_1 *:* $(\{x_2, \neg x_4\},\ 0{,}6), (\{x_2\},\ 0{,}3), (\emptyset,\ 0{,}1)$,
– *pour* C_2 *:* $(\{\neg x_3, x_6\},\ 0{,}5), (\{x_6\},\ 0{,}2), (\emptyset,\ 0{,}1)$.

4.5.2 Résultats

Des PDM ont été générés de la façon présentée précédemment, avec leurs approximations pour obtenir un problème en 2-CNF. Pour chaque problème nous avons calculé la fonction de valeur et une politique optimales, ainsi que les politiques pour les problèmes approchés au même horizon que pour le problème initial. Ces deux politiques approchées ont ensuite été évaluées sur le problème original afin de déterminer leur fonction de valeur associée et de la comparer à l'optimale.

Nous avons réalisé un solveur de PDM factorisés faisant intervenir la bibliothèque CUDD [5] pour représenter les formules propositionnelles quelconques sous forme de diagrammes de décision binaires (BDD) ainsi qu'une implémentation des formules 2-CNF avec leurs opérations associées.

Les temps présentés pour la résolution des problèmes approchés sont obtenus en utilisant une représentation en BDD. En effet, il se trouve que la bibliothèque CUDD est plus efficace en pratique qu'une implémentation dédiée des classes de formules malgré la complexité théorique polynomiale des opérations. De plus, la description des états où une action n'est pas applicable n'est plus nécessaire puisque les BDD peuvent représenter des formules arbitraires.

Approximation en 2-CNF

Pour la résolution sur les problèmes aux conditions d'action approchées par des bornes maximales en 2-CNF nous avons résolu 10 problèmes aléatoires comportant chacun 16 variables (soit 65536 états) et 10 actions. Pour chaque problème présenté dans le tableau 4.1, les trois premières lignes donnent la taille de la fonction de valeur, le temps de résolution (calcul de l'approximation compris) et la qualité moyenne de la politique obtenue pour les problèmes exacts, approchés par une LUB en 2-CNF et approchés par une GLB en 2-CNF, respectivement.

5. `http://vlsi.colorado.edu/~fabio/CUDD/`

TABLE 4.1: Temps de calcul et qualité d'approximation par bornes inférieures et supérieures maximales en 2-CNF, sur des problèmes à 16 variables et 10 actions, à horizon 10. Légende : Approx. (Temps) : Type d'approximation et temps de calcul ; $|V|$: taille de la partition de valeurs finale en nombre de formules ; T : temps de résolution, (avec approximation), en secondes ; RM : récompense moyenne d'un état

Problème	Approx. (Temps)	$\|V\|$	T	(%)	RM	(%)
0	-	48527	333	(100)	250,92	(100)
	LUB 2-CNF (<1)	40968	306	(91,89)	87,86	(35,02)
	GLB 2-CNF (1)	43495	232	(69,67)	201,15	(80,16)
	GLB Horn (<1)	41131	221	(66,37)	207,01	(82,5)
1	-	31614	199	(100)	305,89	(100)
	LUB 2-CNF (<1)	20055	139	(69,85)	152,09	(49,72)
	GLB 2-CNF (1)	24999	124	(62,31)	248,3	(81,17)
	GLB Horn (<1)	30332	143	(71,86)	258,75	(84,59)
2	-	39735	258	(100)	288,35	(100)
	LUB 2-CNF (<1)	32711	268	(103,88)	122,35	(42,43)
	GLB 2-CNF (2)	35574	208	(80,62)	241,67	(83,81)
	GLB Horn (<1)	30969	146	(56,59)	232,76	(83,31)
3	-	49742	384	(100)	253,56	(100)
	LUB 2-CNF (<1)	47587	451	(117,45)	134,92	(53,21)
	GLB 2-CNF (2)	44273	276	(71,88)	215,94	(85,16)
	GLB Horn (<1)	43341	240	(62,5)	188,3	(80,19)
4	-	33581	222	(100)	288,34	(100)
	LUB 2-CNF (<1)	28673	201	(90,54)	166,31	(56)
	GLB 2-CNF (1)	32968	169	(76,13)	254,62	(85,74)
	GLB Horn (<1)	25466	130	(58,56)	238,13	(86,47)
5	-	26190	150	(100)	239,64	(100)
	LUB 2-CNF (<1)	21715	143	(95,33)	122,21	(51)
	GLB 2-CNF (1)	23133	117	(78)	199,43	(83,22)
	GLB Horn (<1)	21929	100	(66,67)	207,22	(79,04)
6	-	17749	86	(100)	264,36	(100)
	LUB 2-CNF (<1)	10753	63	(73,26)	134,66	(50,94)
	GLB 2-CNF (1)	11613	73	(84,88)	221,02	(83,61)

Autres résultats sur la page suivante

Problème	Approx. (Temps)	$\|V\|$	T	(%)	RM	(%)
	GLB Horn (<1)	12877	44	(51,16)	208,96	(79,04)
7	-	34559	224	(100)	260,44	(100)
	LUB 2-CNF (<1)	21761	148	(66,07)	148,45	(57)
	GLB 2-CNF (2)	42307	168	(75)	220,02	(85,55)
	GLB Horn (<1)	31814	171	(76,34)	232,04	(89,1)
8	-	25729	168	(100)	262,82	(100)
	LUB 2-CNF (<1)	13800	95	(56,55)	116,22	(44,22)
	GLB 2-CNF (2)	26288	128	(76,19)	207,81	(79,07)
	GLB Horn (<1)	30116	71	(86,9)	211,58	(80,5)
9	-	20862	109	(100)	274,35	(100)
	LUB 2-CNF (<1)	18890	121	(111,01)	231,13	(84,25)
	GLB 2-CNF (2)	20353	87	(79,82)	228,26	(83,2)
	GLB Horn (<1)	18937	71	(65,14)	234,32	(85,41)
Moyennes	-	32828,8	213,3	(100)	269,73	(100)
	LUB 2-CNF (<1)	25691,3	193,5	(87,58)	141,62	(52,38)
	GLB 2-CNF	30500,3	158,2	(75,45)	223,84	(82,97)
	GLB Horn (<1)	28691,2	141,2	(66,21)	221,91	(82,28)

Sans surprise, les valeurs obtenues en évaluant les politiques calculées grâce aux approximations supérieures minimales sont décevantes car certainement trop « optimistes ». De plus, le gain en temps de calcul est marginal.

Concernant les approximations inférieures maximales, les résultats sont plus réguliers : on observe sur tous les problèmes un gain en temps de calcul de l'ordre de 25 % pour une perte de qualité de l'ordre de 20 %. Si la perte de qualité reste acceptable, le gain en temps de calcul n'est pas significatif. Des expériences sur des problèmes plus grands ont montré une perte de qualité similaire pour un temps de résolution plus mauvais.

Notons que pour ces problèmes de taille raisonnable, le temps de calcul des approximations est négligeable vis-à-vis du temps de résolution.

Approximation en formules de Horn

Au vu des résultats décevants pour la 2-CNF, qui ne bénéficie pas expérimentalement de son efficacité théorique nous avons mené des expériences avec la classe des formules de Horn (CNF avec au plus 1 littéral positif par clause). La notion d'encadrement de la fonction de valeur présentée en section 4.2 étant valide pour toute classe de formules, cette technique reste en effet correcte.

Problème	Approx.	$\|V\|$	T	(%)	RM	(%)
0	-	72552	261	(100)	175,51	(100)
	GLB Horn	80928	226	(86,59)	160,83	(91,64)
1	-	197903	966	(100)	149,65	(100)
	GLB Horn	119649	357	(36,96)	120,78	(80,71)
2	-	78853	287	(100)	121,22	(100)
	GLB Horn	41593	83	(28,92)	100,58	(82,97)
3	-	91110	275	(100)	136,83	(100)
	GLB Horn	82316	168	(61,09)	118,78	(86,81)
4	-	69471	316	(100)	133,14	(100)
	GLB Horn	70253	258	(81,65)	122,25	(91,82)
5	-	141634	780	(100)	113,5	(100)
	GLB Horn	102656	359	(46,03)	104,82	(92,35)
6	-	75223	230	(100)	138,64	(100)
	GLB Horn	75563	152	(66,09)	113,84	(82,11)
7	-	205181	1133	(100)	260,44	(100)
	GLB Horn	101377	359	(31,69)	232,04	(89,1)
8	-	76542	259	(100)	262,82	(100)
	GLB Horn	70313	189	(72,97)	211,58	(80,5)
9	-	65497	194	(100)	274,35	(100)
	GLB Horn	49541	117	(60,31)	234,32	(85,41)
Moyennes	-	107396,6	470,1	(100)	176,61	(100)
	GLB Horn	79418,9	226,8	(57,23)	151,98	(86,34)

TABLE 4.2 – Temps de calcul et qualité d'approximation par bornes inférieures maximales en 2-CNF et Horn, sur des problèmes aléatoires à 20 variables et 12 actions à horizon 6

Nous avons conservé l'utilisation de BDD dans notre solveur, plutôt qu'une implémentation dédiée des opérations logiques sur les formules de Horn. Une telle implémentation s'est en effet révélée peu efficace pour la 2-CNF alors même que la complexité de l'oubli est meilleure que celle de Horn. En outre, au vu des résultats de la 2-CNF nous n'avons pas testé les bornes supérieures.

Le tableau 4.1 présente les résultats sur les mêmes problèmes que pour la 2-CNF, et les tableaux 4.2 et 4.3 comparent les temps de calcul avec le problème exact et approché par une borne inférieure maximale de Horn pour des problèmes plus importants, respectivement 20 variables, 12 actions, et 24 variables, 15 actions.

On constate sur ces approximations une perte de qualité de l'ordre de 15 à 20 % mais cette fois le gain de temps de calcul est significatif : entre 35 et 45 %. Il semblerait que cette approche soit d'autant plus efficace que le problème est grand, tendance qui reste à confirmer par d'autres

Problème	Approx.	$\|V\|$	T	(%)	RM	(%)
0	-	422089	2650	(100)	88,29	(100)
	GLB Horn	268890	1004	(37,89)	73,9	(83,7)
1	-	162033	706	(100)	133,24	(100)
	GLB Horn	137572	416	(58,92)	113,47	(85,16)
2	-	214643	710	(100)	96,32	(100)
	GLB Horn	186229	439	(61,83)	85,56	(88,83)
3	-	362938	1563	(100)	83,14	(100)
	GLB Horn	170185	871	(55,73)	66,79	(80,33)
4	-	227090	1166	(100)	111,94	(100)
	GLB Horn	186229	710	(60,89)	95,97	(85,73)
Moyennes	-	277758,6	1359	(100)	102,59	(100)
	GLB Horn	189821	688	(55,05)	87,14	(84,75)

TABLE 4.3 – Temps de calcul et qualité d'approximation par bornes inférieures maximales de Horn, sur des problèmes aléatoires à 24 variables et 15 actions à horizon 4

expériences. Notons que les résultats montrent une certaine robustesse, car ces gains se retrouvent sur toutes les instances testées.

Approximations successives

Nous avons également testé l'approche par approximations successives de la section 4.4, avec des 2-CNF. La figure 4.2 montre l'évolution de la récompense moyenne des politiques générées en fonction du temps, chaque palier représentant une nouvelle approximation. Le problème testé comporte 20 variables et 12 actions. On constate que la qualité augmente bien avec le temps, mais que pour un temps proche de celui de la résolution exacte la perte de qualité est trop importante. Ceci a été confirmé sur d'autres instances avec la 2-CNF et Horn.

L'explication semble être la suivante. Pour que le temps de calcul des approximations soit raisonnable, il faut utiliser des bornes inférieures non nécessairement maximales. En résulte en une perte de qualité trop importante, comme le montre le premier palier de la figure 4.2 : 50 % de qualité tandis que les bornes maximales montrent une qualité de l'ordre de 80 % (tableau 4.1).

4.6 Conclusion

Dans ces travaux, nous avons proposé une méthode de résolution de processus de décision markoviens représentés de manière compacte à l'aide de la logique propositionnelle. À partir de cette représentation nous avons étudié l'impact des classes de formules dites efficaces sur la complexité de la résolution. Par la suite, nous avons proposé une méthode d'approximation des

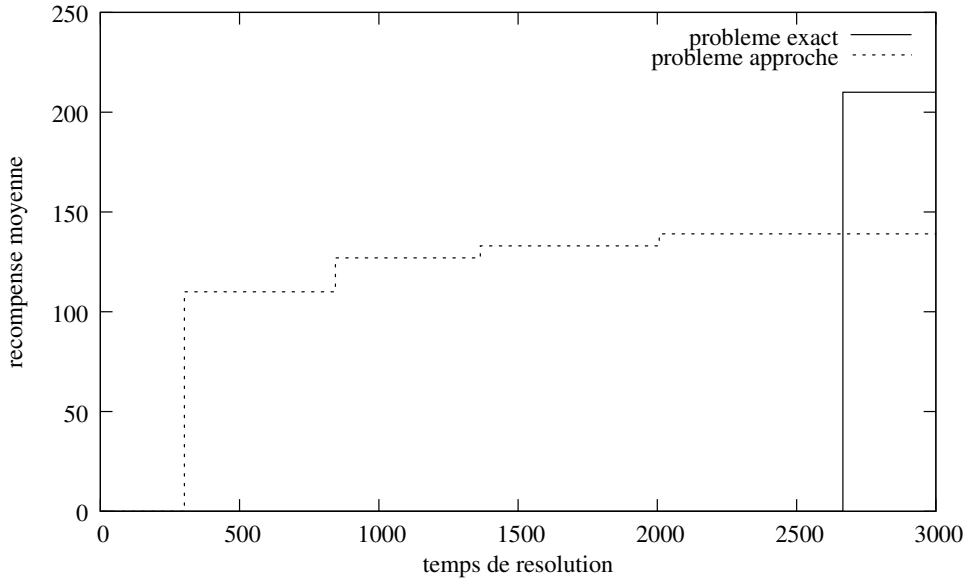


FIGURE 4.2 – Qualité des approximations successives et temps de résolution par bornes inférieures en 2-CNF

problèmes de décision basée sur l'encadrement de formules propositionnelles, une méthode issue de travaux sur l'approximation et la compilation de bases de connaissances.

Notre étude de la 2-CNF a montré que l'efficacité théorique de ses opérations ne s'est pas reflétée en pratique, car l'utilisation d'un solveur manipulant des formules sous la forme de BDD s'est révélée plus efficace qu'une implémentation spécifique pour cette classe de formules. De plus, le gain expérimental n'est pas significatif sur les problèmes aléatoires testés. De la même manière l'approche « anytime », pourtant séduisante en théorie, s'avère peu intéressante en pratique. Néanmoins, elle reste à expérimenter de manière parallèle en calculant différents minorants ou majorants de manière simultanée.

Ces résultats peu encourageants sont compensés par des résultats prometteurs avec des approximations dans la classe des formules de Horn. En effet, notre méthode permet un gain de temps de résolution de presque moitié pour une perte de qualité inférieure à 20 % sur les problèmes testés.

Bien que la qualité des approximations soit en deçà de nos espérances, celles-ci pourraient s'avérer utiles en tant qu'heuristiques pour des algorithmes tels que SLAO* et sRTDP [Feng et Hansen, 2002, Feng *et al.*, 2003]. Ceux-ci requièrent en effet une fonction de valeur initiale qui est un majorant de la fonction de valeur optimale.

Le gain en temps de calcul semble venir de la plus grande simplicité des formules et donc des BDD manipulés. Cependant, nous n'avons pas de résultat théorique appuyant cette intuition ; par exemple, dans le cas des 2-CNF, il existe des BDD de taille exponentielle bien qu'équivalents à une 2-CNF. Quoi qu'il en soit, les résultats expérimentaux restent similaires sur les diverses instances testées, ce qui montre une certaine robustesse de notre approche.

D'un point de vue théorique, notre approche d'approximation de problèmes fournit un cadre solide pour l'étude d'autres biais de représentation des PDM factorisés. Citons par exemple l'approximation sur un sous-ensemble de variables. De manière différente à d'autres approches de simplification des problèmes par élimination de variables, telles que [Dearden et Boutilier, 1997], notre cadre permet seulement d'obtenir un encadrement de la fonction de valeur réelle. Des expériences restent à mener en ce sens.

Une autre perspective consiste à tester notre méthode sur des problèmes réels. Typiquement, de tels problèmes exhibent plus de structure que les problèmes aléatoires et pourraient de fait bénéficier d'autant plus des approximations dans des classes de formules qui capturent ces structures.

Enfin, de manière générale, notre approche de résolution de PDM factorisés permet de traiter naturellement des actions dont les effets portent sur plusieurs variables. Ceci est une différence essentielle avec l'algorithme SPUDD [Hoey *et al.*, 1999], qui ne traite de tels problèmes qu'au prix d'une représentation moins naturelle (*via* les « arcs synchrones »).

Chapitre 5

Un nouvel algorithme de planification avec PPDDL

Sommaire

Ce chapitre est consacré à la présentation d'un algorithme de résolution de PDM décrits de manière compacte en PPDDL. Notre approche est motivée par le fait que les techniques courantes de résolution de tels problèmes se basent sur une traduction du PPDDL. En effet, les actions PPDDL sont traduites sous forme de Réseaux Bayésiens Dynamiques avec lesquels il est alors possible d'utiliser des solveurs de PDM factorisés tels que SPUDD [Hoey *et al.*, 1999], APRICODD [St-Aubin *et al.*, 2001] ou autres.

Cet algorithme utilise les diagrammes de décision algébriques pour représenter toutes les fonctions de valeurs générées au cours de la résolution. En sortie, il produit alors une représentation compacte de la fonction de valeur optimale et de la politique associée.

Une des propriétés importantes de cet algorithme est qu'il permet de calculer la fonction de valeur et la politique optimales en effectuant pour chaque itération un nombre d'opérations sur les ADD *linéaire* en la taille de la description PPDDL du PDM. L'efficacité de ces opérations reste bien entendu dépendante de l'existence d'une représentation de petite taille de ces fonctions.

L'élément central de cet algorithme est la notion de fonctions de valeur d'action dites « frameless » qui sont une représentation plus riche que les simples valeurs d'actions. Tout d'abord ces fonctions et leurs propriétés seront présentées ainsi que les *règles* permettant de les calculer directement à partir des effets PPDDL. Ensuite sera présentée une étude expérimentale de performances de cet algorithme selon différentes caractéristiques des problèmes à résoudre. Les travaux de ce chapitre sont décrits dans [Lesner et Zanuttini, 2011a].

5.1 Valeurs d'actions « frameless »

Le point clé d'un algorithme de calcul de politique optimale tel que l'itération de valeur est la *mise à jour* des valeurs d'actions. Étant donnée une fonction de valeur V, mettre à jour V pour une action a revient à calculer la valeur d'action $\mathfrak{B}^a V$. Quand a est décrite en PPDDL, c'est-à-dire $a = (\phi_a, e_a)$ cette valeur se calcule comme :

$$(\mathfrak{B}^a V)(s) = \begin{cases} \mathbf{E}_{\langle t,r\rangle \sim D(e_a,s)} \left[r + V(apply(t,s)) \right] & \text{si } s \models \phi_a \\ -\infty & \text{sinon} \end{cases} \tag{5.1}$$

Dans l'équation ci-dessus, on considère que lorsque l'état ne satisfait pas la pré-condition de l'action, celle-ci a une valeur de $-\infty$. Ceci permet de marquer cette action comme impossible.

Calculer $\mathfrak{B}^a V$ directement à partir de l'équation (5.1) nécessiterait de calculer la distribution $D(e_a,s)$. Une telle approche est inefficace pour deux raisons. Premièrement, cela nécessiterait d'énumérer les 2^n états du PDM et donc de ne pas avoir de représentation compacte des fonctions de valeur. Deuxièmement, calculer la distribution $D(e_a,s)$ peut prendre un temps et un espace exponentiel en la taille de l'effet e_a, comme le montre l'exemple 34. Notons que ce dernier point est représentatif de la grande expressivité de PPDDL.

Exemple 34 (distribution induite de taille exponentielle) *Soit l'effet* $e = (x_1 \triangleright (0{,}5 \neg x_1)) \wedge \cdots \wedge (x_n \triangleright (0{,}5 \neg x_n))$. *Pour un état* s *composé de* m *littéraux positifs,* $D(e,s)$ *est alors une distribution uniforme sur* 2^m *effets différents. Ainsi pour* $s = x_1 \dots x_n$ *tous les* 2^n *états du PDM sont atteignables avec une probabilité* 2^{-n} *; par contre, pour* $s = \bar{x}_1 \dots \bar{x}_n$ *aucun état autre que* s *lui-même n'est atteignable. Pour certains états, un effet de taille* $O(n)$ *peut donc induire une distribution de taille* $O(2^n)$.

Pour calculer efficacement et de manière compacte la fonction $\mathfrak{B}^a V$, nous introduisons un nouveau type de fonctions de valeur d'action *strictement* plus informatives, dont les propriétés permettent d'utiliser directement la structure des effets PPDDL. Ces fonctions sont très similaires aux valeurs d'actions à l'exception qu'elles ne supposent pas l'hypothèse du cadre, d'où leur nom de fonction de valeur d'action « frameless ».

Définition 35 (valeur d'action frameless) *Soit* $V : \mathcal{S} \to \mathbb{R}$ *une fonction de valeur et* e *un effet. La valeur d'action frameless (F-valeur) de* e *selon* V *est une fonction* $F_V^e : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ *qui*

pour $s,s' \in \mathcal{S}$ donne l'espérance de récompense obtenue en appliquant e dans s et en forçant les valeurs de s' aux variables non modifiées par e :

$$F_V^e(s,s') = \underset{\langle t,r\rangle \sim D(e,s)}{\mathbf{E}} \left[r + V(apply(t,s')) \right]$$

Intuitivement, les F-valeurs se comportent comme les valeurs d'action $\mathfrak{B}^a V$ à l'exception qu'elles ne requièrent pas que les variables non explicitement modifiées par l'action restent inchangées comme le suppose l'hypothèse du cadre. Elles permettent donc d'évaluer une action pour toutes les affectations possibles des variables non forcées par l'action. Si on suppose que ces variables ne changent pas, on obtient alors les valeurs d'actions :

$$(\mathfrak{B}^a V)(s) = F_V^{e_a}(s,s)$$

Exemple 36 *L'évaluation des F-valeurs permet de considérer tous les futurs possibles non générés par l'exécution de l'action. Prenons la F-valeur de l'effet* $\mathtt{x} \triangleright (0{,}6\ \mathtt{x} | 0{,}4\ \mathtt{y})$ *présentée sur la figure 5.1. Si on exécute cet effet dans un état où x est faux, et si l'on arrive dans un état où x est toujours faux, alors on espère une récompense de* 10*. Si par contre x est vrai dans l'état d'arrivée, l'espérance de récompense est de* 2 *si y est faux et de* 5 *sinon.*

5.2 Calcul des F-valeurs

5.2.1 Règles d'action

Les F-valeurs peuvent se calculer récursivement via des *règles d'action*. Étant donné une F-valeur V' et un effet PPDDL e, une règle $B_{V'}(e)$ permet de calculer $F^e_{V'}$ selon chaque type d'effet e. Étant donné que les F-valeurs sont des fonctions de $\mathcal{S} \times \mathcal{S}$, elles sont définies sur l'ensemble de variables $\mathcal{X} \cup \mathcal{X}'$ avec $\mathcal{X}' = \{x' \mid x \in \mathcal{X}\}$.

Quand V' est donnée sous la forme d'un ADD, la table 5.1 donne l'expression des F-valeurs sous forme d'opérations sur des ADD. La figure 5.1 donne un exemple complet de calcul des F-valeurs avec des ADD.

Théorème 37 *Pour un effet e et une fonction de valeur $V : \mathcal{S} \to \mathbb{R}$, soit $V' : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ définie par $V'(\cdot,s) = V(s)$. Alors :*

$$B_{V'}(e) = F_V^e$$

Démonstration. Nous montrons par récurrence sur la description de l'effet e que :

$$B_{V'}(e)(s,s') = \underset{\langle t,r\rangle \sim D(e,s)}{\mathbf{E}} \left[r + V'(s,apply(t,s')) \right] \qquad \text{(HR)}$$

Effet e	ADD $B_{V'}(e)$
$\top$	V'
x	$V'[x' = 1]$
$\neg x$	$V'[x' = 0]$
$\uparrow r$	$V' + r$
$\phi \triangleright e$	$\text{ITE}(\phi, B_{V'}(e), V')$
$p_1\ e_1 \mid \ldots \mid p_k\ e_k$	$p_1 \times B_{V'}(e_1) + \cdots + p_k \times B_{V'}(e_k) + (1 - \sum_{i=1}^{k} p_i) \times V'$
$e_1 \wedge \cdots \wedge e_k$	$B_{B_{V'}(e_1)}(e_2 \wedge \cdots \wedge e_k)$

TABLE 5.1 – Règles de calcul des F-valeurs sous forme d'ADD selon chaque type d'effet.

• Soit $e = x$:

$$\begin{aligned} B_{V'}(x)(s,s') &= V'[x' = 1](s,s') && \{ \text{ définition de } B_{V'}(x) \ \} \\ &= V'(s,apply(x,s')) && \{ \text{ définition de } V'[x = 1] \ \} \\ &= \mathop{\mathbf{E}}_{\langle t,r\rangle \sim D(x,s)} [r + V'(s, apply(t,s'))] && \{ \ D(x,s) = \{\langle x,0\rangle : 1\} \text{ (définition 17) } \} \end{aligned}$$

Le cas de $e = \neg x$ est similaire.

• Soit $e = \uparrow r$:

$$\begin{aligned} B_{V'}(\uparrow r)(s,s') &= (V' + r)(s,s') && \{ \text{ définition de } B_{V'}(\uparrow r) \ \} \\ &= r + V'(s,s') && \{ \text{ définition de } V' + r \ \} \\ &= \mathop{\mathbf{E}}_{\langle t,r\rangle \sim D(x,s)} [r + V'(s, apply(t,s'))] && \{ \ D(\uparrow r,s) = \{\langle \emptyset,r\rangle : 1\} \text{ (définition 17) } \} \end{aligned}$$

• Soit $e = \phi \triangleright e$:

$$\begin{aligned} B_{V'}(\phi \triangleright e')(s,s') &= \text{ITE}(\phi, B_{V'}(e'), V')(s,s') && \{ \text{ définition de } B_{V'}(\phi \triangleright e) \ \} \\ &= \begin{cases} B_{V'}(e')(s,s') & \text{si } s \models \phi \\ V'(s,s') & \text{sinon} \end{cases} && \{ \text{ définition de ITE } \} \\ &= \begin{cases} \mathbf{E}_{\langle t,r\rangle \sim D(e',s)} [r + V'(s, apply(t,s'))] & \text{si } s \models \phi \\ \mathbf{E}_{\langle t,r\rangle \sim D(e,s)} [r + V'(s, apply(t,s'))] & \text{sinon} \end{cases} && \begin{matrix} \{ \text{ (HR) } \} \\ \{ \ D(e,s) = \{\langle \emptyset,0\rangle : 1\} \ \} \end{matrix} \end{aligned}$$

• Soit $e = p_1\ e_1 \mid \ldots \mid p_k\ e_k$:

$$\begin{aligned}
B_{V'}(p_1\ e_1 \mid \ldots \mid p_k\ e_k)(s,s') &= \left(\sum_{i=1}^{k} p_i \times B_{V'}(e_i)\right)(s,s') \quad \{\text{ définition de } B_{V'}(p_1\ e_1 \mid \ldots \mid p_k\ e_k)\ \} \\
&= \sum_{i=1}^{k} p_i B_{V'}(e_i)(s,s') \\
&= \sum_{i=1}^{k} p_i \mathop{\mathbf{E}}_{\langle t,r\rangle \sim D(e_i,s)} \left[r + V'(s, apply(t,s'))\right] \qquad \{\text{ (HR) }\} \\
&= \mathop{\mathbf{E}}_{\langle t,r\rangle \sim D(p_1\ e_1 \mid \ldots \mid p_k\ e_k,s)} \left[r + V'(s, apply(t,s'))\right]
\end{aligned}$$

• Soit $e = e_1 \wedge e_2$:

$$\begin{aligned}
&B_{V'}(e_1 \wedge e_2)(s,s') = B_{B_{V'}(e_1)}(e_2)(s,s') \qquad \{\text{ définition de } B_{V'}(e_1 \wedge e_2)\ \} \\
&= \mathop{\mathbf{E}}_{\langle t_2,r_2\rangle \sim D(e_2)} \left[r_2 + B_{V'}(e_1)(s, apply(t_2,s'))\right] \qquad \{\text{ (HR) }\} \\
&= \mathop{\mathbf{E}}_{\langle t_2,r_2\rangle \sim D(e_2)} \left[r_2 + \mathop{\mathbf{E}}_{\langle t_1,r_1\rangle \sim D(e_1)} \left[r_1 + V'(s, apply(t_2, apply(t_1,s')))\right]\right] \qquad \{\text{ (HR) }\} \\
&= \mathop{\mathbf{E}}_{\substack{\langle t_1,r_1\rangle \sim D(e_1,s) \\ \langle t_2,r_2\rangle \sim D(e_2,s)}} \left[r_1 + r_2 + V'(s, apply(t_2, apply(t_1,s')))\right] \\
&= \mathop{\mathbf{E}}_{\substack{\langle t_1,r_1\rangle \sim D(e_1,s) \\ \langle t_2,r_2\rangle \sim D(e_2,s)}} \left[r_1 + r_2 + V'(s, apply(t_1 \cup t_2,s'))\right] \qquad \{\ t_1 \text{ et } t_2 \text{ consistants }\} \\
&= \mathop{\mathbf{E}}_{\langle t,r\rangle \sim D(e_1 \wedge e_2,s)} \left[r + V'(s, apply(t,s'))\right] \qquad \{\text{ définition de } D(e_1 \wedge e_2,s)\ \}
\end{aligned}$$

Le cas de la conjonction $e_1 \wedge \cdots \wedge e_k, k > 2$ se déduit de l'associativité de $\wedge$.

Pour conclure, la fonction V' ne dépendant que de son second argument et d'après la définition de F_V^e (définition 35) nous obtenons :

$$\mathop{\mathbf{E}}_{\langle t,r\rangle \sim D(e,s)} \left[r + V'(s, apply(t,s'))\right] = \mathop{\mathbf{E}}_{\langle t,r\rangle \sim D(e,s)} \left[r + V(apply(t,s'))\right] = F_V^e(s,s')$$

□

5.2.2 Récupération des valeurs d'action

Une fois la F-valeur $F_V^{e_a}$ d'une action calculée sous forme d'ADD, il faut la transformer en valeur d'action $\mathfrak{B}^a V$. Il s'agit donc de restreindre l'ADD de $F_V^{e_a}$ de façon à ne garder que les chemins où les variables x_i et x_i' ont la même valeur. Dans le cas où x_i' est présente mais pas x_i, on remplace simplement x_i' par x_i. Une première approche pour décrire ce procédé en termes d'opérations sur les ADD est la suivante :

$$\mathfrak{B}^a V = \exists \mathcal{X}' \left[x_1 \leftrightarrow x_1' \times \cdots \times x_n \leftrightarrow x_n' \times F_V^{e_a}\right]$$

Bien que cette approche utilise des opérateurs classiques sur les ADD, elle souffre de nombreux défauts qui la rendent peu efficace :

- L'ADD $x_1 \leftrightarrow x'_1 \times \cdots \times x_n \leftrightarrow x'_n$ est très sensible à l'ordre imposé sur les variables. L'ordre optimal permet d'obtenir un diagramme avec $2n$ nœuds tandis que, dans le pire des cas, il peut avec 2^n nœuds comme le montre l'exemple 10. L'ordre optimal pour représenter cette fonction peut très bien être sous-optimal pour la fonction $F_V^{e_a}$.
- La multiplication entre $x_1 \leftrightarrow x'_1 \times \cdots \times x_n \leftrightarrow x'_n$ et $F_V^{e_a}$ peut mener à un diagramme intermédiaire de grande taille et elle requiert un parcours complet du diagramme $F_V^{e_a}$. De plus l'opération $\exists \mathcal{X}'$ nécessite un parcours complet de ce diagramme intermédiaire.

Pour éviter ces problèmes, il est possible de construire une procédure dédiée, qui calcule $\mathfrak{B}^a V$ en un seul parcours du diagramme de $F_V^{e_a}$. Cette procédure `Persist` est décrite par l'algorithme 9, page 76. Intuitivement, cette procédure parcourt toutes les branches du diagramme, en remplaçant les variables primes par leur équivalent non prime et force la consistance des branches. Un élément important de `Persist` est l'utilisation d'un cache qui permet de stocker les résultats intermédiaires déjà calculés. En effet, comme les nœuds peuvent avoir plusieurs parents, il existe plusieurs chemins pour atteindre un même nœud. Quand ces cas surviennent, le cache permet de n'exécuter qu'une seule fois le calcul de `Persist`.

5.3 L'algorithme RBAB

Ayant une procédure de calcul pour les valeurs d'actions à partir d'une représentation PPDDL, une simple adaptation de l'algorithme d'itération de valeur permet de construire un algorithme complet pour le calcul de politiques optimales. L'algorithme RBAB - pour Rule-Based Action Backup - (algorithme 10) ici présenté tient compte aussi du fait que l'on puisse définir un ensemble d'états buts en PPDDL sous la forme d'une formule Γ. Tout état s tel que $s \models \Gamma$ est alors un état but avec pour récompense R_Γ.

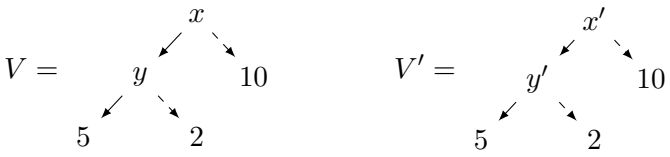
5.4 Évaluation

5.4.1 Méthodologie

Pour évaluer les performances de RBAB, il a été comparé à SPUDD [Hoey *et al.*, 1999] (algorithme 4). Les problèmes d'évaluation sont issus des compétitions internationales de planification probabiliste (IPPC) des années 2006 et 2008 [6]. Ces problèmes sont divisés en domaines, chacun représentant une situation particulière : problème de logistique, de feux d'intersections, etc. Pour chacun de ces domaines, 15 instances de difficulté et de taille croissantes sont disponibles. Les cinq premières instances étant des problèmes triviaux se résolvant en moins d'une

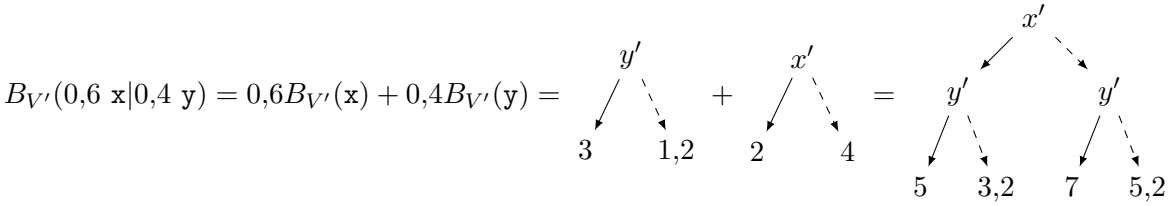
6. L'évaluation n'a pas été possible avec les domaines de l'IPPC 2011 car PPDDL n'est plus utilisé pour cette compétition.

- Fonction de valeur initiale V et de la F-valeur initiale $V'(\cdot,s) = V(s)$

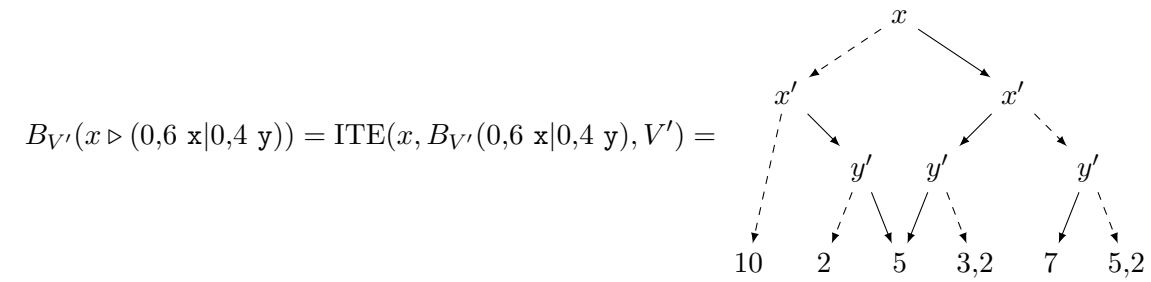


- Calcul des F-valeurs des effets `x` et `y`

- Calcul de la F-valeur de l'effet 0,6 `x`|0,4 `y`



- Calcul de la F-valeur de l'effet $x \triangleright$ (0,6 `x`|0,4 `y`)



- Application de `Persist` pour le calcul de la valeur d'action

$$\mathfrak{B}^a V = \exists x',y' \left[B_{V'}(x \triangleright (0{,}6\ \texttt{x}|0{,}4\ \texttt{y})) \times x \leftrightarrow x' \times y \leftrightarrow y' \right]$$

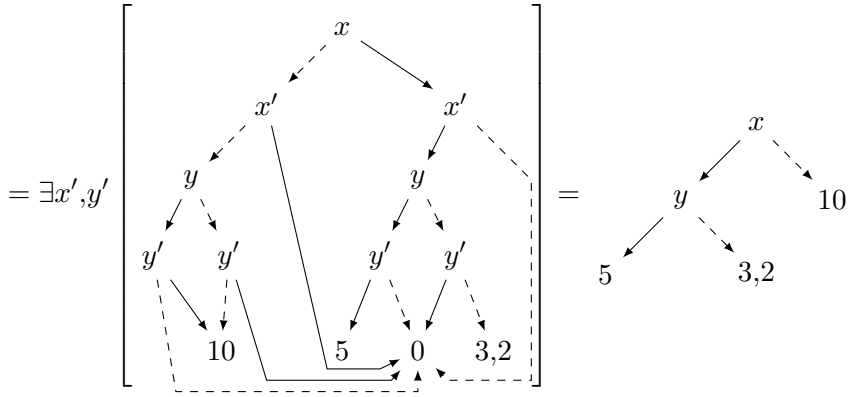


FIGURE 5.1 – Calcul de la F-valeur et de la Q-valeur pour une action $a = (\top, x \triangleright (0{,}6\ \texttt{x}|0{,}4\ \texttt{y}))$.

Algorithme 9 : Persist

Entrées : N, un ADD représentant une F-valeur
Entrées : A, un ensemble de littéraux à forcer (initialement $\emptyset$)
`/* Cas terminal */`
si N *est un ADD constant* **alors**
  return N
`/* Vérifie si le résultat a déjà été calculé */`
si $(N,A) \rightarrow R$ *est en cache* **alors**
  return R
`/* Récupère la variable de décision de` N `*/`
$v \leftarrow var(N)$
`/* Si` v `ou` $\bar{v}$ `est dans` A`, cela signifie que l'on a déjà donné une valeur à` v `précédemment, on retourne alors le sous diagramme correspondant */`
si $v \in A$ **alors**
  return `Persist`$(Then(N), A \setminus \{v\})$
sinon si $\bar{v} \in A$ **alors**
  return `Persist`$(Else(N), A \setminus \{\bar{v}\})$
`/* À ce stade, on rencontre` v `pour la première fois, si` v `n'est pas une variable prime, on force la valeur de` v' `dans les sous diagrammes, et inversement si` v `est prime. */`
$x \leftarrow$`echange(`v`)/* echange(`v'`)=`v`; echange(`v`)=`v' `*/`
`/* Appel récursif pour propager la valeur de` x `dans les sous-diagrammes */`
$T \leftarrow$ `Persist`$(Then(N), A \cup \{x\})$
$E \leftarrow$ `Persist`$(Else(N), A \cup \{\bar{x}\})$
`/* Crée un nœud avec la variable non prime */`
si v *est prime* **alors**
  $u \leftarrow x$
sinon
  $u \leftarrow v$
$R \leftarrow$ `ITE`(u, T, E)
`/* Mise-à-jour du cache. */`
insérer $(N,A) \rightarrow R$ dans le cache
retourner R

seconde n'ont pas été utilisés. Les caractéristiques des différents domaines sont résumées dans la table 5.2.

RBAB calculant des politiques ϵ-optimales, il a été comparé à SPUDD plutôt qu'aux planificateurs probabilistes des IPPC, qui exploitent la connaissance d'un état initial et qui fournissent parfois des résultats approchés. Deux variantes de SPUDD ont été utilisées : SPUDD « 1 by 1 » qui correspond exactement à l'algorithme 4 et SPUDD « matrix » qui calcule au préalable l'ADD représentant la matrice de transition de chaque action (le produit $\prod_{i=1}^{n} T_a(x'_i)$). Les comparai-

Algorithme 10 : Rule-Based Action Backup (RBAB)

Entrées : $\mathcal{A}$: L'ensemble d'actions au format PPDDL.
Entrées : Γ : Un ADD booléen représentant les états buts.
Entrées : R_Γ : Un ADD constant représentant la récompense des états buts
Sorties : π : ADD de politique, chaque état correspond à un indice d'action

```
/* Initialisation de la fonction de valeur avec la récompense des buts   */
```
$V \leftarrow \Gamma \times R_\Gamma$
répéter
  $W \leftarrow -\infty$
  $V' \leftarrow \gamma \times \mathit{prime\_variables}(V)$
  pour chaque *action* $a = (\phi_a, e_a)$ **faire**
    $F \leftarrow B_{V'}(e_a)$; `/* Calcul de la F-valeur de` a `*/`
    $Q \leftarrow \texttt{ITE}(p_a, \texttt{Persist}(F), -\infty)$
    $W \leftarrow \max(W, Q)$
  $W \leftarrow \texttt{ITE}(\Gamma, R_\Gamma, W)$; `/* Force une valeur de` R_Γ `pour les états buts */`
  converge $\leftarrow ||V - W||_\infty \leq \frac{\epsilon(1-\gamma)}{2\gamma}$
  $V \leftarrow W$
jusqu'à *converge* = VRAI
`/* Extraction de la politique` π `*/`
$W \leftarrow -\infty$
$V' \leftarrow \gamma \times \mathit{prime\_variables}(V)$
$\pi \leftarrow 0$
pour chaque *action* a_i, $i = 1, \ldots, |A|$ **faire**
  $Q \leftarrow \texttt{ITE}(p_a, \texttt{Persist}(B_{V'}(e_a)), -\infty)$
  $G \leftarrow Q > W$
  $\pi \leftarrow \max(\pi, G \times i)$
  $W \leftarrow \texttt{ITE}(G, Q, W)$; `/* Équivalent à` $W \leftarrow \max(Q,W)$ `mais plus efficace */`
retourner $\pi \times \neg\Gamma$; `/* Les états buts utilisent une action 0 factice */`

sons sont faites en termes de temps de calcul pour une politique 0,1-optimale à horizon infini pondéré avec $\gamma = 0{,}9$. Pour les deux variantes de SPUDD le temps de traduction du problème en DBN est pris en compte dans les résultats présentés. Pour toutes les instances d'évaluation les différents solveurs se sont vu allouer une heure de temps de calcul pour fournir une politique. Un exception a été faite sur le domaine « drive » où nous avons alloué six heures à tous les solveurs.

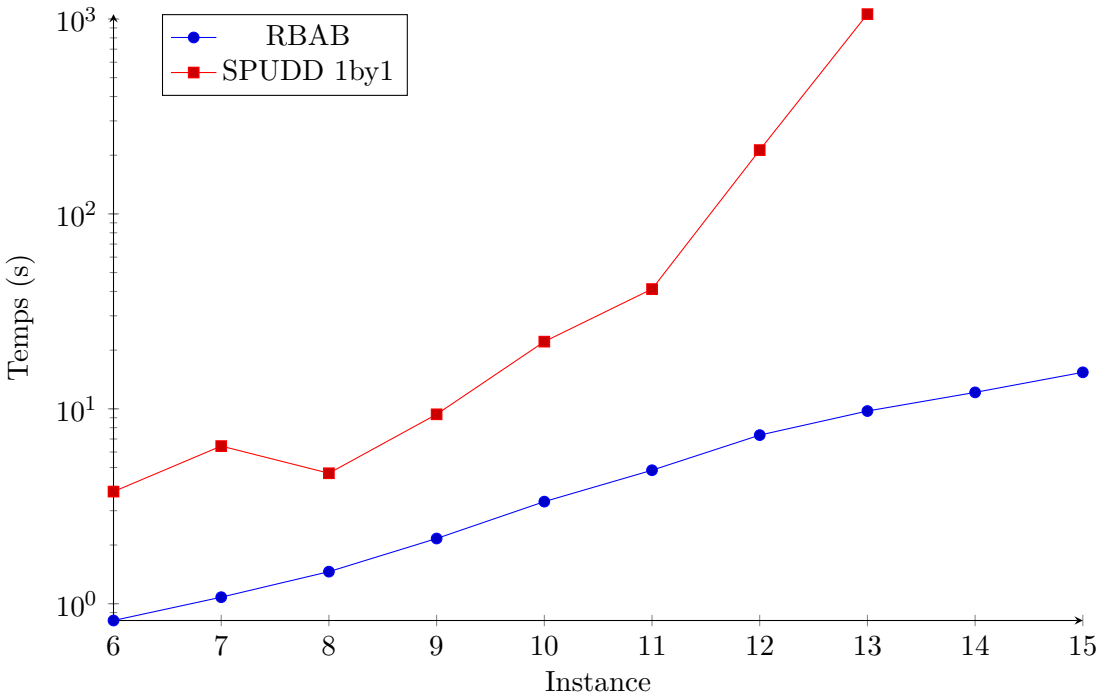


FIGURE 5.2 – Temps de résolution pour les instances du domaine `search-and-rescue`.

5.4.2 Résultats des domaines d'évaluation

Search and Rescue

Le domaine « search and rescue » est issu de la catégorie probabiliste de la 6e compétition internationale de planification qui s'en déroulée en 2008. Ce domaine met en scène un hélicoptère autonome dont le but est de secourir une personne. La zone de recherche est décomposée en zones qui doivent être explorées afin de déterminer si l'atterrissage est possible. Rien n'empêche a priori l'hélicoptère d'atterrir, mais il ne le détecte qu'avec une certaine probabilité. L'hélicoptère peut atterrir dans n'importe quelle zone où c'est possible pour secourir l'humain, cependant, en exécutant la plupart des actions, l'humain a une probabilité de mourir et donc de faire échouer la mission.

Les différentes instances de ce domaine se distinguent seulement par le nombre de zones qu'il comporte. Ce nombre varie de 4 zones pour l'instance 1 à 50 zones pour l'instance 15, avec un nombre de variables allant de 18 à 157, le nombre d'actions varie entre 20 et 208. Les temps de calcul d'une politique ϵ-optimale sont présentés figure 5.2.

Les performances de RBAB sur ce domaines sont très encourageantes, puisque l'on peut observer une diminution exponentielle du temps de calcul comparé à SPUDD.

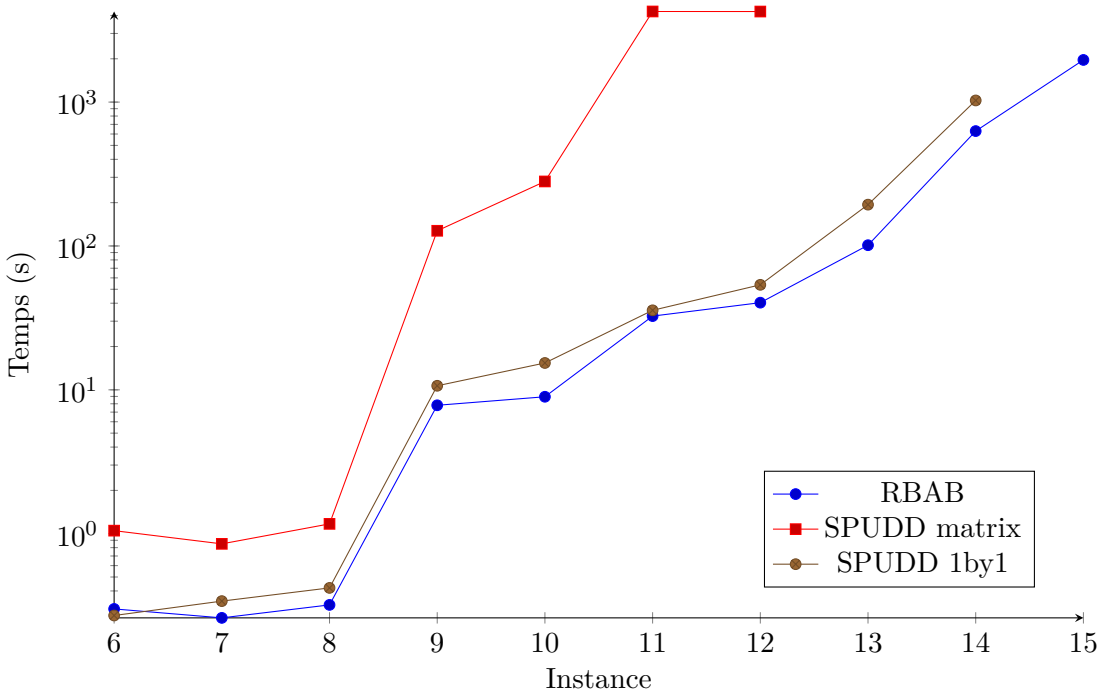


FIGURE 5.3 – Temps de résolution pour les instances du domaine `pitchcatch`.

Pitchcatch

Le domaine « pitchcatch » est inspiré du jeu de base ball, où il met en jeu un cycle de lancer, de réception et de passe de balles. Chaque instance propose différents types de balles dont la difficulté à réceptionner varie. Un cycle de jeu consiste à lancer une balle dont le type est choisi selon une certaine distribution de probabilités. La probabilité de rattraper une balle dépend de l'activation de « bits ». Plus il y a de bits activés plus la balle est facile à rattraper. Bien que chaque lancer active certains bits avec une petite probabilité, l'agent peut les activer explicitement mais au risque de mourir et de perdre le jeu. Une fois la balle récupérée, il faut la passer pour compléter le cycle, mais la passe peut être mortelle.

Les instances de ce domaine proposent de plus en plus de types de balles et de différents bits afin de compliquer les actions. Ce domaine a l'intérêt d'être très riche car les nuances entre les différentes actions sont subtiles. Les grands espaces d'états, d'actions et leurs nombreuses issues possibles ont mis à mal les planificateurs participant à la compétition de planification de 2006 [Bonet et Givan, 2006]. On observe sur la figure 5.3 que RBAB et SPUDD 1-by-1 se comportent de façon similaire, avec tout de même un léger avantage pour RBAB qui arrive notamment à résoudre la 15^{e} instance dans le temps imparti. Toutefois SPUDD matrix est nettement moins efficace avec des temps de calculs de 10 à 100 fois plus importants que les autres algorithmes.

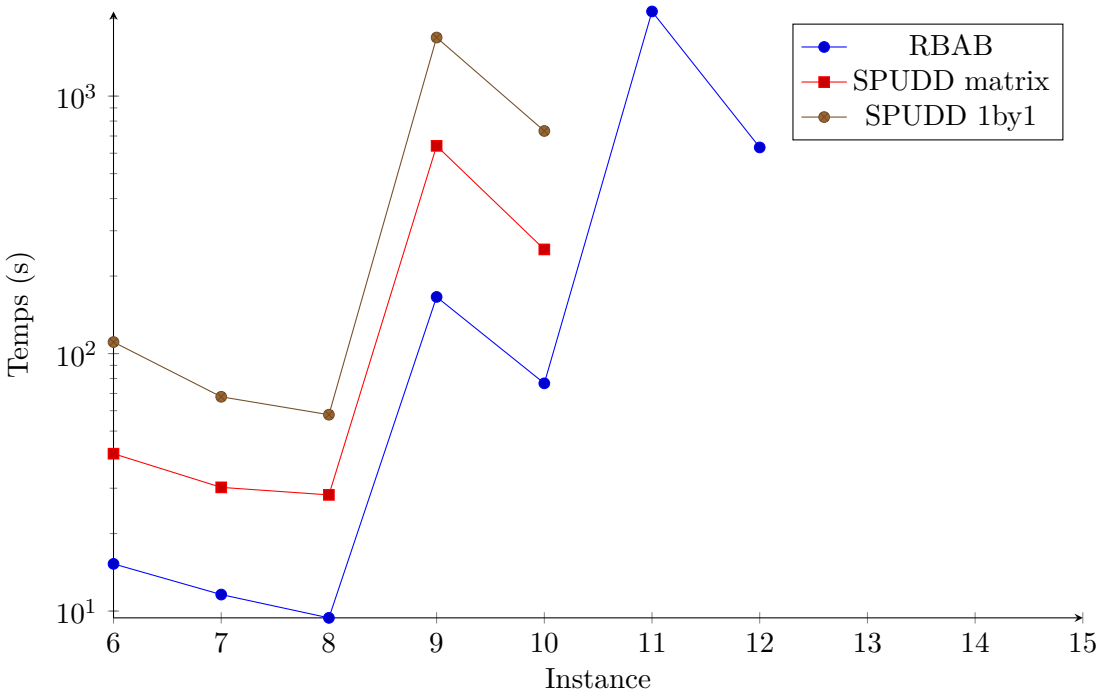


FIGURE 5.4 – Temps de résolution pour les instances du domaine `rectangle-tireworld`.

Rectangle Tireworld

Le domaine « rectangle-tireworld » met en scène une voiture qui peut se déplacer entre les différentes intersections d'une grille rectangulaire via des routes pour atteindre une destination but. À chaque fois qu'un déplacement est entrepris il y a une certaine probabilité de crever un pneu et donc d'échouer. Il existe des positions sûres et d'autres dangereuses ainsi que des positions « normales ». Un déplacement vertical ou horizontal depuis une position sûre réussira tout le temps tandis que si la position est normale ou dangereuse il y aura une probabilité de rester sur place. Les déplacements en diagonale sont plus rapides mais plus risqués. Un tel déplacement depuis une position dangereuse conduit à un échec certain tandis que dans les autres cas l'action réussit le plus souvent avec néanmoins une probabilité de crevaison et donc d'échec. Il s'agit donc de déterminer le chemin permettant d'atteindre le but avec la plus grande probabilité.

Les différentes instances de ce domaine proposent des environnements de plus en plus grands avec un nombre croissant de positions dangereuses. Une fois propositionalisé, ce domaine contient un nombre d'actions très important (cf. table 5.2) qui le rend très difficile à traiter. On observe sur la figure 5.4 que RBAB a un net avantage sur les autres algorithmes. Il est environ deux fois plus rapide que SPUDD matrix - qui auparavant était toujours le moins performant - et presque 10 fois plus rapide que SPUDD 1-by-1. En effet, sur ce domaine, RBAB a pu résoudre deux instances de plus que SPUDD.

Drive

Le domaine « drive » représente des problèmes de navigation dans une grille. Les intersections sont reliées par des segments de longueur variable, et sont munies chacune d'un feu de circulation. Lors des déplacements ou de l'attente aux feux, l'agent encourt constamment un risque de mourir. L'agent dispose de trois types d'actions : regarder la couleur des feux, attendre le passage au vert et avancer à la prochaine intersection. La couleur des feux est déterminée stochastiquement via l'action regarder. Si le feu est rouge, l'action attendre le fait passer au vert avec une certaine probabilité (il y a toujours un risque de mourir avec cette action). Une fois le feu vert il peut alors se déplacer avec aussi un risque de mourir.

Ce domaine est intéressant du fait que les effets PPDDL de l'action « regarder les feux » est décrite de manière peu compacte. L'effet de cette action contient notamment :

$$
\begin{aligned}
& 0{,}9((n \wedge l) \triangleright g)|0{,}1((n \wedge l) \triangleright r) \wedge 0{,}9((s \wedge l) \triangleright g)|0{,}1((s \wedge l) \triangleright r) \\
\wedge \quad & 0{,}1((e \wedge l) \triangleright g)|0{,}1((e \wedge l) \triangleright r) \wedge 0{,}1((w \wedge l) \triangleright g)|0{,}1((w \wedge l) \triangleright r) \\
\wedge \quad & 0{,}9((ns \wedge \neg l) \triangleright g)|0{,}1((ns \wedge \neg l) \triangleright r) \wedge 0{,}9((s \wedge \neg l) \triangleright g)|0{,}1((s \wedge \neg l) \triangleright r) \\
\wedge \quad & 0{,}1((e \wedge \neg l) \triangleright g)|0{,}1((e \wedge \neg l) \triangleright r) \wedge 0{,}1((w \wedge \neg l) \triangleright g)|0{,}1((w \wedge \neg l) \triangleright r)
\end{aligned}
$$

Cet effet est composé de 47 sous-effets redondants. Une formulation équivalente et plus compacte peut être obtenue en appliquant les règles d'équivalence de la table 2.4 :

$$
\begin{aligned}
& ((l \wedge (e \vee w))) \vee (\neg l \wedge (n \vee s))) \triangleright (0.1g|0{,}9r) \\
\wedge \quad & ((\neg l \wedge (e \vee w))) \vee (l \wedge (n \vee s))) \triangleright (0.9g|0{,}1r)
\end{aligned}
$$

Cette écriture du même effet comporte désormais 9 sous-effets. Elle correspond au domaine « drive-compact ». Les solveurs ont aussi été comparés avec cette représentation. Le temps de calcul de RBAB étant directement dépendant du nombre de règles d'action à appliquer, on peut voir sur la figure 5.5 que ce compactage des effets améliore grandement l'efficacité de RBAB.

Une autre façon équivalente d'exprimer ces problèmes est le « déroulage » des actions. Il s'agit de décomposer une action en plusieurs autres plus simples, chacune avec une pré-condition différente. De façon générale, une action a de la forme $(\phi, (\psi_1 \triangleright e_1) \wedge \cdots \wedge (\psi_k \triangleright e_k))$ telles que les formules ψ_i sont mutuellement inconsistantes peut se décomposer en k actions $a_i, i = 1, \ldots, k$ telles que $a_i = (\phi \wedge \psi_i, e_i)$. Bien que cette transformation augmente le nombre d'actions, elles sont plus simples et évitent le coût important du calcul des F-valeurs pour les effets conjonctifs. La figure 5.5 montre que les deux solveurs profitent du déroulage des actions, surtout RBAB qui devient alors nettement plus rapide que SPUDD (les courbes présentées pour SPUDD utilisent les temps de la meilleure variante pour l'instance considérée).

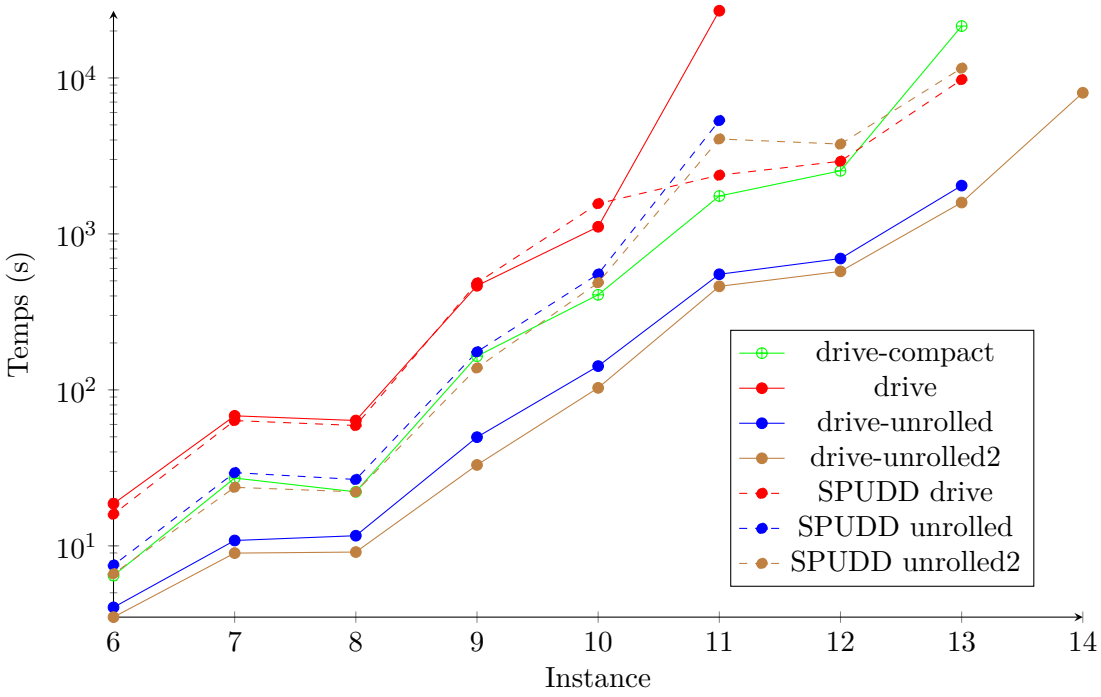


FIGURE 5.5 – Évolution du temps de calcul d'une politique optimale avec RBAB en fonction de la compacité de la description PPDDL.

instance		6	7	8	9	10	11	12	13	14	15
search-and-rescue	$\|\mathcal{X}\|$	42	48	54	66	81	96	111	126	141	156
	$\|\mathcal{A}\|$	52	60	68	84	104	124	144	164	184	204
pitchcatch	$\|\mathcal{X}\|$	20	20	20	27	27	30	30	33	36	39
	$\|\mathcal{A}\|$	16	16	16	25	25	34	34	45	58	73
rectangle-tireworld	$\|\mathcal{X}\|$	23	23	31	31	41	41	41	61	61	-
	$\|\mathcal{A}\|$	12940	12940	45724	45724	147364	147364	147364	763744	763744	-
drive	$\|\mathcal{X}\|$	24	33	33	44	44	57	57	72	89	108
	$\|\mathcal{A}\|$	176	236	236	366	432	602	602	800	1026	1280
drive-unrolled	$\|\mathcal{X}\|$	24	33	33	44	44	57	57	72	89	108
	$\|\mathcal{A}\|$	227	318	316	486	549	760	772	1013	1290	1607

TABLE 5.2 – Nombre de variables et d'actions pour les instances d'évaluation.

5.5 Conclusion

5.5.1 Bilan

Dans ce chapitre, nous avons présenté l'algorithme « Rule-Based Action Backup » ou RBAB pour la résolution factorisée de PDM décrits en PPDDL. Le point clé de cet algorithme est la notion de valeur d'action *frameless* qui permet de manipuler les effets conjonctifs. Les performances de RBAB sont dues à l'utilisation des Diagrammes de Décision Algébriques qui permettent à la fois de représenter de manière compacte et de manipuler efficacement les fonctions de valeur. Les évaluations sur les domaines de la compétition internationale de planification probabiliste sont encourageants. Il reste cependant des domaines inaccessibles à RBAB, tout comme à SPUDD. Ces derniers sont en effet trop complexes pour espérer pouvoir les résoudre sans recours à des méthodes approchées.

RBAB peut théoriquement profiter des méthodes d'approximation d'APPRICODD [St-Aubin *et al.*, 2001] une version de SPUDD pour la résolution approchée. Les méthodes d'approximation d'APPRICODD sont basées sur l'approximation des ADD des fonctions de valeur construites à chaque itération et sont donc indépendantes de la méthode employée pour générer ces diagrammes. Une autre piste serait l'utilisation d'ADD affines [Sanner et McAllester, 2005] (AADD) qui dans certains cas peuvent être exponentiellement plus compacts que les ADDs pour représenter certaines fonctions. De plus, ces derniers ont pour avantage de pouvoir multiplier en temps constant un AADD quelconque par une constante, ce qui peut se révéler avantageux pour le calcul des F-valeurs d'effets probabilistes. À ce jour, il n'existe cependant pas d'implémentation efficace de cette structure de données et les premières tentatives réalisées dans le cadre de cette thèse ont montré une grande instabilité numérique au delà de 20 variables.

5.5.2 À propos des domaines de l'IPPC et de PPDDL

Bien que tenant lieu de problèmes de référence pour l'évaluation de problèmes probabilistes, les jeux de tests des compétitions internationales de planification probabiliste sont très influencés par leurs origines dans la planification classique. Par conséquent, tous les domaines possèdent des états buts plutôt qu'une fonction de récompense. Seuls quelques domaines possèdent une fonction de coût sur les actions à optimiser. Certains domaines introduisent inutilement des prédicats pour représenter l'absence d'autres prédicats, de façon similaire à STRIPS qui ne peut pas exprimer de pré-conditions requérant *l'absence* de certains prédicats dans l'état courant. L'absence de variables multivaluées en PPDDL étant simulée par des prédicats, de nombreuses variables inutiles sont alors introduites lors de la propositionalisation des problèmes, puisque de nombreuses combinaisons de valeurs pour ces variables sont interdites.

De la même façon, de nombreuses variables non modifiables sont introduites pour encoder la topologie du domaine. Par exemple dans les instances de « drive », ce sont des prédicats binaires qui encodent la relation de voisinage entre les intersections, et des prédicats unaires sont utilisés pour indiquer la longueur des segments de route. Ainsi, si l'instance spécifie qu'une route est

courte, SPUDD et RBAB vont calculer une politique aussi pour les états où cette route est moyenne ou longue alors que de tels états ne seront jamais rencontrés à l'exécution.

Troisième partie

Apprentissage

Introduction de la partie

D'un point de vue général, l'apprentissage automatique consiste à concevoir un algorithme capable de faire des prédictions sur ce qui va se produire dans l'environnement et ce uniquement à partir d'observations collectées au fil du temps. Plus précisément, cette thèse considère que l'environnement se comporte comme un PDM mais dont la spécification des fonctions de transition et de récompense sont inconnues. Au fil du temps, l'agent doit essayer les différentes actions à sa disposition pour finalement être capable de prédire le comportement et la récompense obtenue par ces actions. À terme, l'agent en question doit être capable d'agir d'une manière qui soit efficace voir quasi optimale.

Ce type d'apprentissage se modélise comme un problème *d'apprentissage par renforcement* [Sutton et Barto, 1998] où l'agent apprend à agir par tentatives. L'agent est directement placé dans l'environnement dans lequel il doit agir, il ne possède aucune idée a priori sur ce qu'il doit faire. Dans chaque état qu'il rencontre, il choisit une action à exécuter, puis il observe l'état résultant et un signal de récompense. À force de tentatives, il arrive alors à déterminer les « bonnes » actions qui apportent le plus de récompense. La difficulté majeure de ce problème d'apprentissage est le dilemme de l'exploitation contre l'exploration. Plus l'agent fait d'essais d'actions pour apprendre leur comportement, plus il risque de ne pas prendre les bonnes décisions. Au contraire, agir uniquement selon un petit nombre de connaissances accumulées risque de mener à un comportement sous optimal faute d'idée précise de l'environnement.

Une partie des travaux présentés par la suite s'intéressent à des algorithmes d'apprentissage par renforcement capables de donner des garanties sur les temps d'exploration requis pour obtenir un comportement quasi-optimal. Tandis que d'autres sont heuristiques : on ne peut pas donner de garanties sur le comportement de l'agent. Nous verrons cependant que celles-ci donnent de très bons résultats sur certains domaines.

Cette partie est organisée de la manière suivante. Après quelques préliminaires sur le problème d'apprentissage par renforcement et un panorama des approches existantes, nous présentons deux techniques pour l'apprentissage du modèle de transitions d'un PDM représenté compactement en STRIPS probabiliste. Ces algorithmes exploitent l'existence d'une telle représentation compacte afin d'accélérer le processus d'apprentissage. Cette partie se termine par une évaluation empirique de nos algorithmes sur des domaines de la littérature.

Chapitre 6

L'apprentissage par renforcement

Sommaire

Ce chapitre présente le cadre de l'apprentissage par renforcement. Nous décrivons tout d'abord le problème général et les différentes familles de méthodes existant pour ce problème. Nous nous intéressons ensuite plus particulièrement aux approches capables de donner certaines garanties théoriques sur le comportement de l'agent. Nous présentons en effet plusieurs algorithmes de la littérature, que ce soit dans le cadre des PDM factorisés ou non. Finalement nous présentons le cadre d'apprentissage « *Knows What It Knows* » dans lequel se situent une grande partie de nos contributions.

6.1 L'apprentissage par renforcement

6.1.1 Le problème général

Dans le cadre de l'apprentissage par renforcement [Sutton et Barto, 1998] l'agent est mis directement en situation avec son environnement. Il ne possède pas d'information sur la dynamique (la fonction de transition) et les récompense du PDM qui décrit cet environnement.

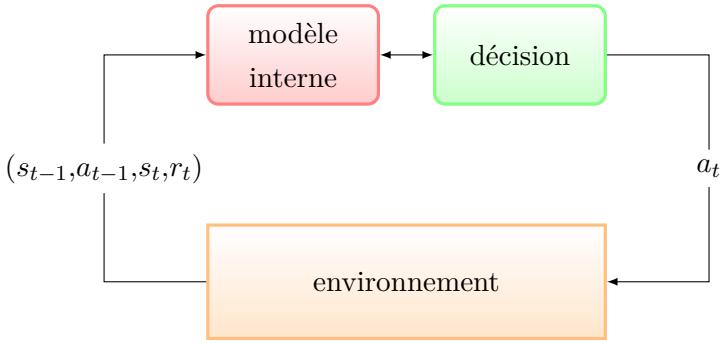


FIGURE 6.1 – Architecture d'un algorithme d'apprentissage par renforcement

On suppose que l'agent agit indéfiniment dans son environnement, et que le temps est découpé en instants discrets. À chaque pas de temps $t = 1,2,\ldots$, l'agent observe un quadruplet $(s_{t-1},a_{t-1},s_t,r_t)$ où s_{t-1} est l'état précédent de l'agent et s_t et r_t étant respectivement l'état et la récompense résultant de l'exécution de l'action a_{t-1} dans s_{t-1}. Les observations sont interprétées pour mettre à jour un modèle interne qui sera utilisé pour choisir l'action suivante à exécuter (figure 6.1).

L'agrégation de ces observations dans un modèle interne peut être faite de trois manières différentes qui définissent trois familles d'algorithmes d'apprentissage par renforcement. On distingue les approches :

model-based les observations sont utilisées pour estimer le modèle de transition $\hat{P}(s'|a,s)$ et de récompense $\hat{R}(s,a)$. La décision suivante de l'agent requiert alors de résoudre le PDM correspondant.

action-based à partir des observations, l'agent produit une estimation des valeurs d'action $\hat{Q}(s,a)$, et choisit l'action qui maximise ces fonctions.

policy-based les observations sont directement utilisées pour inférer une politique $\hat{\pi}$ sans représentation intermédiaire.

Différentes propriétés de ces approches sont résumées dans la figure 6.2. Les approches « model-based » ou *indirectes*, bien que coûteuses en espace car elles stockent une estimation des modèles de transitions et de récompense, permettent de dériver des algorithmes d'apprentissage avec des garanties fortes sur la qualité du comportement de l'agent avec un temps d'apprentissage *fini*. D'un autre côté, les approches *directes* « action-based » et « policy based » s'éloignent en quelque sorte des observations pour maintenir un modèle interne plus compact et plus adapté à la prise de décision. Cependant, avec ce type d'approches indirectes il est plus difficile d'apporter des garanties de qualité autrement qu'après un temps infini (convergence asymptotique).

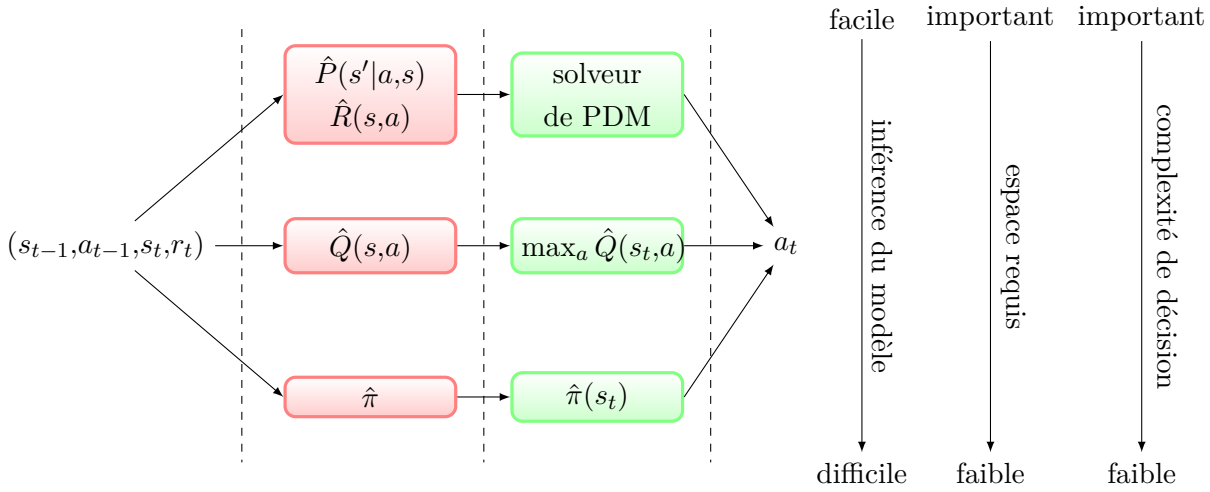


FIGURE 6.2 – Les trois familles d'algorithmes d'apprentissage par renforcement

6.1.2 Apprentissage PAC MDP

Le cadre PAC-MDP (Probably Approximately Correct Markov Decision Process) s'inspire du PAC-apprentissage de concepts booléens [Valiant, 1984]. Pouvoir apprendre dans ce cadre donne de nombreuses garanties sur ce qui est appris. En particulier, il est possible de garantir avec une grande confiance (« Probably ») que ce qui est appris est proche de la cible (« Approximately Correct ») seulement après un nombre fini d'exemples.

Dans le cadre PAC, les exemples d'apprentissage sont tirés selon une distribution de probabilité fixée mais inconnue de l'apprenant. L'apprentissage est considéré comme un succès si une bonne approximation du concept (une formule par exemple) cible est apprise. Le cadre PAC requiert qu'après un petit nombre (polynomial) d'exemples (des modèles de la formule par ex.), l'algorithme obtienne un succès avec une grande probabilité. Traditionnellement on donne à un algorithme PAC deux paramètres : $\epsilon > 0$ et $0 < \delta < 1$. ϵ est une borne sur la qualité de l'approximation, une mesure de qualité appropriée doit au préalable être définie pour le concept cible. δ est une borne supérieure de la probabilité d'échec de l'algorithme, on requiert donc que l'algorithme réussisse avec probabilité au moins $1 - \delta$. Le nombre d'exemples requis doit alors être au pire polynomial en $\frac{1}{\epsilon}$ et $\frac{1}{\delta}$.

La notion d'algorithme PAC pour l'apprentissage de concepts est aussi applicable pour l'apprentissage par renforcement dans les PDM. On parle alors du cadre PAC-MDP. Dans le cadre PAC-MDP la notion de complexité en exemples est remplacée par la *complexité d'exploration* qui correspond au nombre de pas de temps où l'agent n'a pas choisi une action quasi-optimale.

Définition 38 (complexité d'exploration [Kakade, 2003]) *Soit $\epsilon > 0$, la* complexité d'exploration *d'un algorithme d'apprentissage par renforcement dans un PDM est une borne sur le nombre de pas de temps t où les actions a_t exécutées par l'agent vérifient*

$$Q^*(s_t, a_t) < V^*(s_t) - \epsilon$$

Un algorithme PAC-MDP est formellement défini comme suit.

Définition 39 (PAC-MDP [Strehl *et al.*, 2009]) *Un algorithme est dit* efficient PAC-MDP *si pour tout* $\epsilon > 0$ *et* $0 < \delta < 1$*, les complexités en temps, en espace et d'exploration sont bornées par un polynôme* $p(|\mathcal{S}|, |\mathcal{A}|, \frac{1}{\epsilon}, \frac{1}{\delta}, \frac{1}{1-\gamma})$ *avec probabilité au moins* $1-\delta$*. Si l'on retire la contrainte sur la complexité en temps, on parle alors d'algorithme* PAC-MDP*. Si le PDM est factorisé et décrit par n variables propositionnelles, on requiert alors une borne en* $p(n, |\mathcal{A}|, \frac{1}{\epsilon}, \frac{1}{\delta}, \frac{1}{1-\gamma})$.

6.2 Quelques algorithmes PAC-MDP

Dans cette section nous présentons quatre algorithmes ayant des garanties PAC-MDP. D'abord dans le cadre des PDM énumérés nous présentons RMAX et E^3, deux algorithmes indirects et un algorithme direct : Delayed Q-Learning. Une étude détaillée des propriétés PAC-MDP de RMAX et DELAYED Q-LEARNING est donnée dans [Strehl *et al.*, 2009]. Ensuite nous présentons SLF-RMAX [Strehl *et al.*, 2007] et MET-RMAX [Diuk *et al.*, 2009] pour l'apprentissage indirect de PDM factorisés. Le tableau 6.1 récapitule les propriétés des différents algorithmes présentés.

6.2.1 Rmax

L'algorithme RMAX [Brafman et Tennenholtz, 2003] est un des premiers algorithmes d'apprentissage par renforcement indirect qui soit efficient PAC-MDP. RMAX estime les probabilités de transitions du PDM par la fréquence observée. Mais la principale innovation de cet algorithme est la façon dont il gère le compromis exploration/exploitation. Pour cela, il applique le principe de *l'optimisme face à l'incertitude*. On suppose la connaissance a priori d'une borne supérieure $U(s,a)$ sur les valeurs d'action $Q^*(s,a)$, cette hypothèse n'est cependant pas très restrictive car $U(s,a) = \frac{\max_{s,a} R(s,a)}{1-\gamma}$ (équation 1.1) est toujours valide, il suffit donc de connaître $\max_{s,a} R(s,a)$. Dans les zones de l'espace d'états où RMAX ne possède pas ou peu d'informations, il suppose qu'il pourra obtenir une récompense espérée de $U(s,a)$. De cette façon, l'agent est attiré *à la fois* vers les zones où il manque d'informations et où il pourra explorer efficacement et vers celles pour lesquelles il sait qu'il obtiendra de grandes récompenses et pourra exploiter.

RMAX estime la fonction de transition P à l'aide de compteurs $n(s,a)$ et $n(s,a,s')$ qui dénotent respectivement le nombre de fois où l'agent a exécuté l'action a dans l'état s et le nombre de fois où a dans s a mené à s'. De cette façon, l'estimation empirique de la fonction de transition est une fonction $\hat{P}$ définie comme :

$$\hat{P}(s'|s,a) = \frac{n(s,a,s')}{n(s,a)}$$

La récompense peut être observée directement dès qu'un couple état-action a été testé[7]. À chaque instant, l'agent choisit l'action qui maximise les valeurs d'actions $\hat{Q}(s,a)$ calculées à

7. Dans le cas d'une récompense stochastique, RMAX peut calculer son espérance de manière similaire à la fonction de transition.

partir de son modèle estimé :

$$\hat{Q}(s,a) = \begin{cases} \hat{R}(s,a) + \gamma \sum_{s'} \hat{P}(s'|s,a) \max_{a'} \hat{Q}(s',a') & \text{si } n(s,a) \geq m \\ U(s,a) & \text{sinon} \end{cases}$$

Le nombre $m > 0$ est le *seuil d'exploitation*, le nombre d'exemples à partir duquel on utilise les transitions estimées. Une valeur appropriée de m respectant les contraintes du cadre PAC-MDP peut être calculée pour garantir que l'estimation de la fonction de transition soit précise. Pour $m = \frac{|\mathcal{S}|^2|\mathcal{A}|}{\epsilon^2} \ln \frac{|\mathcal{S}||\mathcal{A}|}{\delta}$ RMAX est *efficient* PAC-MDP avec une complexité d'exploration de

$$O\left(\frac{|\mathcal{S}|^2|\mathcal{A}|}{\epsilon^3(1-\gamma)^6} \ln \frac{|\mathcal{S}||\mathcal{A}|}{\delta} \ln \frac{1}{\delta} \ln \frac{1}{\epsilon(1-\gamma)}\right)$$

et ce avec probabilité au moins $1 - \delta$.

6.2.2 Delayed Q-learning

L'algorithme DELAYED Q-LEARNING [Strehl *et al.*, 2006b] est le premier algorithme PAC-MDP direct. Il ressemble à l'algorithme Q-LEARNING à l'exception que la mise à jour des valeurs d'actions est différée. Comme RMAX, il prend en entrée un paramètre m sur le nombre d'observations de chaque couple état-action qu'il doit recevoir avant de donner une prédiction pour ce couple. En l'occurrence, pour chaque couple (s,a), la fonction $\hat{Q}(s,a)$ n'est mise à jour qu'après avoir reçu m observations pour (s,a). En initialisant $\hat{Q}(s,a)$ de manière optimiste et en ne faisant sa mise à jour que si le changement de valeur est significatif, DELAYED Q-LEARNING est *efficient* PAC-MDP avec une complexité d'exploration de

$$O\left(\frac{|\mathcal{S}||\mathcal{A}|}{\epsilon^4(1-\gamma)^7} \ln \frac{|\mathcal{S}||\mathcal{A}|}{\delta} \ln \frac{1}{\delta} \ln \frac{1}{\delta\epsilon(1-\gamma)}\right)$$

avec probabilité au moins $1 - \delta$.

Remarquons que la complexité d'exploration de DELAYED Q-LEARNING dépend linéairement de $|\mathcal{S}|$ en comparaison à RMAX qui en dépend quadratiquement. La dépendance en ϵ et $1-\gamma$ est cependant plus élevée, des bornes plus précises sont données dans [Strehl *et al.*, 2009]. Notons toutefois que cette dépendance linéaire en $|\mathcal{S}|$ correspond à celle d'une borne inférieure de

$$\Omega\left(\frac{|\mathcal{S}||\mathcal{A}|}{\epsilon^2(1-\gamma)^2} \ln \frac{|\mathcal{S}|}{\delta}\right)$$

applicable pour tout algorithme PAC-MDP [Strehl *et al.*, 2009].

6.2.3 Autres algorithmes PAC-MDP non factorisés

De nombreux autres algorithmes avec des garanties PAC-MDP existent pour les PDM énumérés. Un des précurseurs du domaine est l'algorithme E^3 [Kearns et Singh, 1998, Kearns et Singh, 2002] pour « Explicit Exploit or Explore » qui combine deux politiques, une pour l'exploration et l'autre pour l'exploitation. Ensuite, on trouve notamment MBIE [Strehl et Littman,

2005] et une version améliorée MBIE-EB [Strehl et Littman, 2008] qui se basent sur des intervalles de confiance sur les estimations de la fonction de transition, une technique inspirée des travaux sur les bandits à k bras. Des variantes incrémentales de MBIE et RMAX ont été proposées par [Strehl *et al.*, 2006a]. Ces deux nouveaux algorithmes : RTDP-IE et RTDP-RMAX, s'inspirent de la programmation dynamique en temps réel (RTDP [Barto *et al.*, 1995, Bonet et Geffner, 2003]) pour améliorer nettement la complexité en temps de calcul de RMAX et de MBIE tout en préservant les garanties PAC-MDP. Leur complexité d'exploration reste néanmoins la même que celle de RMAX.

6.2.4 Apprentissage dans les PDM factorisés

RMAX et E^3 ont été adaptés au cadre des PDM factorisés avec une représentation en DBN. Lorsque que le graphe de dépendance des DBN est connu, les algorithmes PAC-MDP FACTORED RMAX [Gestrin *et al.*, 2002] ou DBN-E^3 [Kearns et Koller, 1999] peuvent être utilisés. Par la suite, cette hypothèse de connaissance du graphe de dépendances du DBN a été levée avec l'algorithme SLF-RMAX [Strehl *et al.*, 2007] qui peut apprendre à la fois les tables de probabilité conditionnelles et le graphe de dépendances si le nombre maximum de parents de chaque variable est borné par une constante k. Cet algorithme a une complexité d'exploration en $O(n^{2k})$ au lieu de $O(2^n)$ pour RMAX. Par la suite, [Diuk *et al.*, 2009] ont proposé MET-RMAX, qui fait passer cette dépendance de $O(n^{2k})$ à $O(n^k)$.

Apprentissage des parents avec SLF-Rmax

Nous détaillons ici le principe de fonctionnement de la découverte des parents d'une variable avec SLF-RMAX. Nous réutiliserons ensuite une variante de cette technique pour l'apprentissage des conditions d'actions dans les opérateurs STRIPS probabilistes.

Pour apprendre les parents d'une variable x'_i, en connaissant une borne $k \geq |Parents(x'_i)|$, SLF-RMAX va énumérer les $\binom{n}{2k}$ ensembles possibles de $2k$ variables et collecter m exemples pour chacune des affectations possibles à ces variables. Ainsi, pour chacun des $2^{2k}\binom{n}{2k}$ termes t de taille $2k$, il maintient un compteur $c(i,t)$ correspondant au nombre de transitions ($s = x_1 \dots x_n$,$s' = x'_1 \dots x'_n$) telles que $t \subseteq s$ et $x'_i = 1$. À partir de ce compteur, il est possible d'estimer la probabilité de $x'_i = 1$ dans les états vérifiant t de la manière suivante :

$$\hat{P}(x'_i = 1|t) = \frac{c(i,t)}{m}$$

Le fait d'utiliser des observations conditionnées par $2k$ variables permet de découvrir $Parents(x'_i)$. En effet, quand la fonction de transition est représentée par un DBN, cette dernière ne dépend que des valeurs de $Parents(x'_i)$:

$$P(x'_i = 1|s,a) = P(x'_i = 1|s[Parents(x'_i)],a)$$

alors, pour tout ensemble de variables $X \subseteq \mathcal{X}$ nous avons :

$$P(x'_i = 1|s,a) = P(x'_i = 1|s[Parents(x'_i) \cup X], a) \tag{6.1}$$

Maintenant, étant donné un état s et $K \subseteq \mathcal{X}$ un ensemble de k variables, si pour toute paire d'ensembles $K \cup X$ et $K \cup Y$ de $2k$ variables la condition suivante est vérifiée :

$$\left|\hat{P}(x_i = 1|s[K \cup X]) - \hat{P}(x_i = 1|s[K \cup Y])\right| \leq \epsilon$$

Alors, tout ensemble $K \cup Z$ de $2k$ variables vérifie

$$\left|\hat{P}(x_i = 1|s[K \cup Z]) - P(x_i = 1|s,a)\right| \leq 2\epsilon$$

La condition ci-dessus se justifie de la manière suivante. Tout d'abord, quel que soit K il existe un ensemble K^* tel que $\mathit{Parents}(x_i) \subseteq K \cup K^*$. L'application de l'inégalité de Hoeffding [Hoeffding, 1963] permet de choisir une valeur de m telle que la prédiction $\hat{P}(x_i = 1|s[K \cup K^*],a)$ vérifie $\left|\hat{P}(x_i = 1|s[K \cup K^*],a) - P(x_i = 1|s,a)\right| \leq \frac{\epsilon}{2}$ avec une grande probabilité. Ensuite, la condition impliquant que $\left|\hat{P}(x_i = 1|s[K \cup K^*],a) - P(x_i = 1|s[K \cup Z],a)\right| \leq \epsilon$. L'inégalité triangulaire donne alors le résultat souhaité. Notons que cette condition est toujours vérifiée avec une grande probabilité lorsque $\mathit{Parents}(x_i) \subseteq K$. Donc une bonne estimation de $P(x_i = 1|s,a)$ est nécessairement trouvée.

Apprentissage des parents avec MET-Rmax

L'algorithme MET-Rmax demande un plus petit nombre d'exemples pour découvrir les parents d'une variable. Il requiert toujours une borne k sur le nombre de parents d'une variable, mais il ne nécessite de collecter que des exemples depuis des termes (affectations des parents) de taille k (et non $2k$ comme SLF-Rmax). Le principe est que MET-Rmax collecte suffisamment d'exemples afin de calculer les prédictions $\hat{P}(x_i = 1|t,a)$ pour chacun des $2^k\binom{n}{k}$ termes de taille k. Ensuite, l'algorithme passe dans une phase d'élimination des hypothèses. Pendant un certain nombre M de pas de temps, il mesure l'erreur de chaque prédiction par rapport aux observations reçues et élimine ensuite les hypothèses dont l'erreur de prédiction est trop élevée. Une analyse théorique permet de déterminer qu'une quantité M polynomiale en n^k suffit à éliminer les mauvaises hypothèses avec une grande probabilité. Une variante plus générale de cette technique est l'algorithme Noisy-Union [Li *et al.*, 2011] utilisé pour la sélection d'hypothèses.

Autres méthodes d'apprentissage des parents

Les algorithmes SLF-Rmax et MET-Rmax présentés ci-dessus ont l'avantage de donner des garanties PAC-MDP. Cependant, les techniques qu'ils utilisent restent assez lourdes pour une application pratique. En parallèle de ces algorithmes on trouve SPITI [Degris *et al.*, 2006] qui apprend des tables de probabilités conditionnelles sous la forme d'arbres de décision. Les parents d'une variable sont donc les variables apparaissant dans les nœuds internes de ces arbres. Ils sont découverts grâce à une mesure d'information de type Kolmogorov-Smirnoff ou χ^2. Informellement ces mesures sont utilisées pour quantifier la pertinence de l'ajout d'une variable dans l'arbre de décision (et donc dans l'ensemble de parents) pour classifier les exemples d'apprentissage. Même ci cette approche ne donne aucune garantie de convergence, les résultats empiriques en termes

Algorithme	Factorisé ?	Structure connue ?	Complexité[†] d'exploration	Référence
RMAX	•	-	$\tilde{O}\left(\frac{\lvert\mathcal{S}\rvert^2\lvert\mathcal{A}\rvert}{\epsilon^3(1-\gamma^6)}\right)$	[Brafman et Tennenholtz, 2003]
E^3	•	-	$\tilde{O}\left(\frac{\lvert\mathcal{S}\rvert^2\lvert\mathcal{A}\rvert}{\epsilon^4(1-\gamma)^8}\right)$	[Kearns et Singh, 1998]
DELAYED Q-LEARNING	•	-	$\tilde{O}\left(\frac{\lvert\mathcal{S}\rvert\lvert\mathcal{A}\rvert}{\epsilon^4(1-\gamma)^7}\right)$	[Strehl *et al.*, 2006b]
RTDP-IE	•	-	$\tilde{O}\left(\frac{\lvert\mathcal{S}\rvert^2\lvert\mathcal{A}\rvert}{\epsilon^3(1-\gamma)^6}\right)$	[Strehl *et al.*, 2006a]
RTDP-RMAX	•	-	$\tilde{O}\left(\frac{\lvert\mathcal{S}\rvert^2\lvert\mathcal{A}\rvert}{\epsilon^3(1-\gamma)^6}\right)$	[Strehl *et al.*, 2006a]
MBIE	•	-	$\tilde{O}\left(\frac{\lvert\mathcal{S}\rvert^2\lvert\mathcal{A}\rvert}{\epsilon^3(1-\gamma)^6}\right)$	[Strehl et Littman, 2005]
MBIE-EB	•	-	$\tilde{O}\left(\frac{\lvert\mathcal{S}\rvert^2\lvert\mathcal{A}\rvert}{\epsilon^3(1-\gamma)^6}\right)$	[Strehl et Littman, 2008]
DBN-E^3	✓	oui		[Kearns et Koller, 1999]
FACTORED-RMAX	✓	oui	$\tilde{O}\left(\frac{n^3 2^{k+1} k\lvert\mathcal{A}\rvert}{\epsilon^3(1-\gamma)^6}\right)$	[Gestrin *et al.*, 2002]
SLF-RMAX	✓	taille : k	$\tilde{O}\left(\frac{n^{3+2k} 2^{2k} k\lvert\mathcal{A}\rvert}{\epsilon^3(1-\gamma)^6}\right)$	[Strehl *et al.*, 2007]
MET-RMAX	✓	taille : k	$\tilde{O}\left(\frac{n^{3+k} 2^{k+1} k\lvert\mathcal{A}\rvert}{\epsilon^3(1-\gamma)^6}\right)$	[Diuk *et al.*, 2009]

[†]La notation $\tilde{O}(\cdot)$ ignore les facteurs logarithmiques.

TABLE 6.1 – Tableau récapitulatif des algorithmes d'apprentissage PAC-MDP.

de récompense accumulée et de compacité du modèle de transitions ont montré l'efficacité de cet algorithme.

6.3 Knows What It Knows

La cadre d'apprentissage KWIK (knows what it knows) [Li *et al.*, 2011] décrit les propriétés que doit avoir un algorithme d'apprentissage pour avoir une exploration efficace (polynomiale) en se basant sur une approche similaire à RMAX. Un algorithme KWIK estime une fonction cible h^* inconnue (ici des fonctions de transition de PDM). Étant donné un exemple x donné par l'environnement, il *doit* donner une estimation précise de $h^*(x)$, ou alors répondre « je ne sais pas » ($\perp$) et reçoit une *observation* de $h^*(x)$. Cependant il n'a le droit de répondre $\perp$ qu'un nombre polynomial de fois.

6.3.1 Protocole d'apprentissage

Les fonctions cibles d'un algorithme KWIK sont définies sur des ensembles quelconques $\mathcal{X}$ d'entrées et $\mathcal{Y}$ de sorties. La classe des *hypothèses* (les fonctions envisageables par l'apprenant) est notée $\mathcal{H} \subseteq (\mathcal{X} \to \mathcal{Y})$. La fonction cible est notée $h^* : \mathcal{X} \to \mathcal{Y}$ et génère les exemples fournis à l'apprenant. Un troisième ensemble, noté $\mathcal{Z}$, est l'ensemble des observations qui permet de modéliser le bruit de perception de l'apprenant.

L'apprenant connaît $\mathcal{H}$ (mais pas h^*), et les paramètres $\epsilon > 0$ et $\delta \in]0,1[$. L'environnement connaît aussi ces paramètres en plus de la fonction cible h^*. Ensuite, pour chaque instant $t = 1,2,\ldots$:

- L'environnement choisit un exemple $x_t \in \mathcal{X}$ de manière adversaire. La valeur de $y_y = h^*(x_t)$ est inconnue de l'apprenant.
- L'apprenant fait une prédiction $\hat{y}_t \in \mathcal{Y} \cup \{\perp\}$. Si $\hat{y}_t = \perp$, cela signifie que l'apprenant « ne sait pas », autrement on dit que $\hat{y}_t$ est valide.
- Si la prédiction $\hat{y}_t = \perp$, l'apprenant reçoit une observation $z_t \in \mathcal{Z}$ de y_t. Si la fonction cible est déterministe et s'il n'y a pas de bruit de perception ,on a alors $z_t = y_t$. Dans un cadre probabiliste, on peut avoir par exemple y_t pour la probabilité d'un événement aléatoire E et l'observation z_t est alors E et non pas sa probabilité.

6.3.2 KWIK-apprenabilité

Définition 40 (KWIK-apprenabilité [Li *et al.*, 2011]) *Soit $\mathcal{H} \subseteq (\mathcal{X} \to \mathcal{Y})$ une classe d'hypothèses. On dit qu'une classe $\mathcal{H}$ est KWIK-apprenable, s'il existe un algorithme* **A** *et pour tous paramètres $\epsilon > 0$ et $\delta \in]0,1[$, les deux propriétés suivantes sont satisfaites avec une probabilité* au moins $1 - \delta$ *lors d'une exécution complète du protocole d'apprentissage par* **A** *:*

1. **précision** *si $h^* \in \mathcal{H}$, alors toutes les prédictions valides (différentes de $\perp$) sont ϵ-précises ; c'est-à-dire que $d(\hat{y}_t, y_t) < \epsilon$, pour une mesure d'erreur ou de distance d spécifique à la fonction apprise. Par exemple si $\mathcal{Y} \subseteq \mathbb{R}$, on peut utiliser $d(\hat{y}_t, y_t) = |\hat{y}_t - y_t|$, ou si $\mathcal{Y} \subseteq \mathbb{R}^k$, d peut être une norme sur les vecteurs etc.*
2. **complexité d'échantillonnage** *le nombre de prédictions $\perp$ renvoyées tout au long de l'exécution de l'algorithme, noté $B(\epsilon,\delta, dim(\mathcal{H}))$ est borné par un polynôme de $\frac{1}{\epsilon}$, $\frac{1}{\delta}$ et $dim(\mathcal{H})$, où $dim(\mathcal{H})$ correspond à une mesure de taille ou de complexité de $\mathcal{H}$.*

Un algorithme **A** satisfaisant la définition 40 est appelé un *KWIK-algorithme* et $B(\epsilon,\delta, dim(\mathcal{H}))$ est la *borne KWIK* de **A**. Si **A** ne requiert qu'une complexité temporelle polynomiale en $\frac{1}{\epsilon}$, $\frac{1}{\delta}$ et $dim(\mathcal{H})$ à chaque pas de temps, $\mathcal{H}$ est dite *efficacement KWIK-apprenable.*

6.3.3 KWIK-Rmax

La définition de KWIK-apprenabilité partage de nombreuses similitudes avec celle de PAC-MDP, notamment sur les garanties de complexité exigées et l'interprétation des paramètres. En effet, un résultat important de [Li *et al.*, 2011] concerne la connexion entre KWIK et PAC-MDP. Ce résultat énonce que si la fonction de transition et la fonction de récompense d'une classe de PDM sont KWIK-apprenables, alors il existe un algorithme d'apprentissage par renforcement PAC-MDP pour cette classe de PDM.

Définition 41 *Soit un ensemble d'états $\mathcal{S}$ et d'actions $\mathcal{A}$ et un facteur de pondération γ.*

1. **transitions** *Soit $\mathcal{X} = \mathcal{S} \times \mathcal{A}$, $\mathcal{Y}_T = \mathcal{P}_\mathcal{S}$ et $\mathcal{Z}_T = \mathcal{S}$ (les états sont observés, pas leur probabilité). Soit $\mathcal{H}_T \subseteq (\mathcal{X} \to \mathcal{Y}_T)$ un ensemble de fonctions de transitions d'un PDM. $\mathcal{H}_T$ est (efficacement) KWIK-apprenable avec $d(\hat{P}(\cdot|s,a), P(\cdot|s,a)) = \left\|\hat{P}(\cdot|s,a) - P(\cdot|s,a)\right\|_1$ comme fonction d'erreur.*
2. **récompenses** *Soit $\mathcal{X} = \mathcal{S} \times \mathcal{A}$, $\mathcal{Y}_R = \mathcal{Z}_R = [0,1]$ (les récompenses sont directement observables, finies, positives et bornées par 1). Soit $\mathcal{H}_R \subseteq (\mathcal{X} \to \mathcal{Y}_R)$ un ensemble de fonctions de récompenses d'un PDM. $\mathcal{H}_R$ est (efficacement) KWIK-apprenable avec $d(\hat{R}(s,a), R(s,a)) = \left|\hat{R}(s,a) - R(s,a)\right|$ comme fonction d'erreur.*
3. **PDM** *La classe de PDM $\mathcal{M} = \{(\mathcal{S},\mathcal{A},P,R,\gamma) | T \in \mathcal{H}_T, R \in \mathcal{H}_R\}$ est (efficacement) KWIK-apprenable si $\mathcal{H}_T$ et $\mathcal{H}_R$ sont (efficacement) KWIK-apprenables.*

Dans le cadre des PDM, la notion de dimension d'une classe d'hypothèses $dim(\mathcal{H})$ a l'interprétation suivante. Si le PDM est énuméré alors on considère $dim(\mathcal{H}) = (|\mathcal{S}|, |\mathcal{A}|)$. Dans le cas factorisé sur n variables d'états, on considérera $dim(\mathcal{H}) = (n, |\mathcal{A}|)$.

Théorème 42 ([Li *et al.*, 2011]) *Soit $\mathcal{M}$ une classe de PDM, si elle est (efficacement) KWIK-apprenable avec les algorithmes $\mathbf{A}_T$ pour les transitions et $\mathbf{A}_R$ pour les récompenses avec pour bornes KWIK respectives $B_T(\epsilon_T,\delta_T)$ et $B_R(\epsilon_R,\delta_R)$, alors l'algorithme* KWIK-Rmax *(algorithme 11) est PAC-MDP avec les paramètres suivants :*

$$\epsilon_T = \frac{\epsilon(1-\gamma)^2}{16}, \qquad \epsilon_R = \frac{\epsilon(1-\gamma)}{16}, \qquad \epsilon_P = \frac{\epsilon(1-\gamma)}{24}, \qquad \delta_T = \delta_R = \frac{\delta}{4}$$

La complexité d'exploration de KWIK-Rmax *est alors de :*

$$O\left(\frac{B_T(\epsilon(1-\gamma)^2,\delta) + B_R(\epsilon(1-\gamma),\delta)}{\epsilon(1-\gamma)^2} \ln\frac{1}{\delta} \ln\frac{1}{\epsilon(1-\gamma)}\right)$$

Algorithme 11 : KWIK-RMAX

Input : $\mathcal{S}$,$\mathcal{A}$, $\mathbf{A}_T$ (avec les paramètres ϵ_T, δ_T), $\mathbf{A}_R$ (avec ϵ_R, δ_R), ϵ_P

pour chaque *pas de temps* $t = 1,2,3,\ldots$ **faire**

`// Mise-à-jour du PDM empirique` $\hat{M} = (\mathcal{S},\mathcal{A},\hat{P},\hat{R})$

pour chaque $(s,a) \in \mathcal{S} \times \mathcal{A}$ **faire**

si $\mathbf{A}_T(s,a) = \perp$ *ou* $\mathbf{A}_R(s,a) = \perp$ **alors**

$$\hat{P}(s'|s,a) \leftarrow \begin{cases} 1 & \text{si } s' = s \\ 0 & \text{sinon} \end{cases}$$

$\hat{R}(s,a) \leftarrow 1$

sinon

$\hat{P}(s'|s,a) \leftarrow \mathbf{A}_T(s,a)$

$\hat{R}(s,a) \leftarrow \mathbf{A}_R(s,a)$

Calculer une fonction de valeur d'action Q ϵ_P-optimale pour $\hat{M}$

Observer l'état courant s_t

Exécuter $a_t = \arg\max_{a \in \mathcal{A}} Q(s_t,a)$

Observer la récompense r_{t+1}

Observer le nouvel état s_{t+1}

si $\mathbf{A}_T(s_t, a_t) = \perp$ **alors**

Présenter l'exemple $(s_t, a_t) \mapsto s_{t+1}$ à $\mathbf{A}_T$

si $\mathbf{A}_R(s_t, a_t) = \perp$ **alors**

Présenter l'exemple $(s_t, a_t) \mapsto r_{t+1}$ à $\mathbf{A}_R$

Chapitre 7

Apprentissage heuristique d'actions STRIPS probabilistes

Sommaire

Ce chapitre traite de l'apprentissage par renforcement dans les PDM quand le modèle d'actions sous-jacent est représenté de manière compacte en opérateurs STRIPS probabilistes. Nous considérons que les seules observations accessibles à l'apprenant sont des *transitions entre états*, il n'observe pas directement les effets qui provoquent ces transitions. Nous identifions tout d'abord le phénomène d'ambiguïté des observations qui est un obstacle majeur à l'apprentissage des effets d'action. Ensuite nous présentons une première approche pour apprendre un modèle de transition avec un nombre d'exemples d'apprentissage ne dépendant pas du nombre d'états du PDM. Cet algorithme [Lesner et Zanuttini, 2011b] utilise une heuristique basée sur la programmation linéaire pour apprendre les effets d'actions et leurs probabilités. Une évaluation empirique est ensuite présentée dans le chapitre 9.

7.1 Cadre formel

Dans cette section, nous présentons le modèle d'apprentissage étant données les observations reçues par l'agent. Bien que nous nous intéressions à l'apprentissage par renforcement où l'agent a un certain contrôle sur les états dans lesquels il reçoit des observations, nous ne faisons *pas* cette

hypothèse dans ce qui suit. Ceci nous permet de considérer le cadre adversaire où l'environnement présente des transitions à partir d'états minimalement informatifs. L'avantage de cette approche est que tous les résultats peuvent s'appliquer directement dans l'apprentissage par renforcement où l'utilisation de certaines stratégies d'exploration permet de guider la collecte d'exemples vers des états informatifs.

Les actions que nous cherchons à apprendre sont représentées sous la forme d'opérateurs STRIPS probabilistes. Il s'agit d'apprendre le modèle caché d'une action $a = \{(c_i,D_i)|i = 1,\ldots,k\}$, où, pour tout $i = 1,\ldots,k$, c_i est une condition et D_i une distribution de probabilités sur un ensemble d'effets. L'apprenant ne connaît ni les conditions, ni les distributions, ni les effets sur lesquels portent ces distributions. Dans un premier temps, nous présentons le cadre formel pour apprendre une action cachée comportant une seule condition, qui est représentée sous la forme d'une distribution d'effets $D = \{(e_i,p_i)|i = 1,\ldots,\ell\}$. Nous présenterons ensuite comment étendre ces résultats pour découvrir à la fois les conditions et les distributions associées.

7.1.1 Ambiguïté des effets et représentation compacte

La difficulté majeure de l'apprentissage d'un modèle d'action PSO est *l'ambiguïté des effets observés*. En effet, nous supposons que seule l'observation des transitions entre états - pas des effets les ayant provoqués - est possible, il existe donc *plusieurs* effets pouvant provoquer cette transition, comme le montre le lemme suivant.

Lemme 43 (ambiguïté des effets observés) *Étant donnés deux états s et s', les effets e tels que $s' = apply(e,s)$ sont exactement ceux tels que*

$$s' \setminus s \subseteq e \subseteq s'$$

Exemple 44 *Pour une transition observée entre l'état $s = x_1x_2x_3$ et $s' = x_1\bar{x}_2x_3$, les effets qui engendrent cette transition sont : $\bar{x}_2$,$x_1\bar{x}_2$, $\bar{x}_2x_3$ et $x_1\bar{x}_2x_3$.*

Une conséquence positive de ce lemme est que l'ensemble des effets pouvant provoquer une transition d'un état s à un état s' peut être représenté de manière compacte sous forme *d'intervalle d'effets*.

Définition 45 (intervalle d'effets) *Étant donnés deux termes l,$u \in 3^{\mathcal{X}}$, l'intervalle d'effets noté $[l,u]$ contient tous les effets e tels que $l \subseteq e$ et $e \subseteq u$. Un intervalle est vide si et seulement si $l \not\subseteq u$. Les intervalles d'effets non vides obéissent aux règles suivantes :*

- $[l,u] \cap [l',u'] = [l \cup l', u \cap u']$
- $[l,u] \cap [l',u'] \neq \emptyset \iff l \subseteq u' \wedge l' \subseteq u$
- $[l,u] \subseteq [l',u'] \iff l' \subseteq l \wedge u \subseteq u'$

Par convention, l'intervalle singleton contenant un seul effet e est noté $[e]$.

Ainsi, l'ensemble des effets provoquant une transition de s à s' est exactement l'intervalle $[s' \setminus s, s']$. L'effet $s' \setminus s$ représente l'effet minimal : au moins $s' \setminus s$ doit changer mais il se peut que tous

les littéraux de s' aient changé. Quand un effet e a provoqué une transition de s à $s' = apply(e,s)$ nous avons l'identité suivante :

$$[apply(e,s) \setminus s, apply(e,s)] = [e \setminus s, apply(e,s)]$$

Exemple 46 (suite) *Pour les états s et s' de l'exemple 44, l'intervalle correspondant est $[\bar{x}_2, x_1\bar{x}_2x_3]$.*

La définition suivante nous sera utile par la suite.

Définition 47 (termes consistants pour un état) *Soit s un état (ou une partie d'un état) et k un entier strictement positif. L'ensemble des termes de taille k consistants avec s est défini comme :*

$$T_s^k = \{t \mid t \subseteq s,\ |t| = \min\{k, |s|\}\}$$

7.1.2 Exemples et observations

Définition 48 (séquence etats-effets) *Une* séquence d'états-effets *est n'importe quelle séquence de couples état-effet $\mathcal{T} = ((s_i, e_i))_{i=1,\ldots,m}$ où pour $i = 1, \ldots, m$, s_i est un état et e_i un effet.*

Définition 49 (distribution induite) *Soit une séquence d'état-efffets $\mathcal{T} = ((s_i, e_i))_{i=1,\ldots,m}$. La* distribution induite *par $\mathcal{T}$, notée $D_\mathcal{T}$, est une distribution de probabilités sur les effets de $\mathcal{T}$, où la probabilité $D_\mathcal{T}(e)$ de chaque effet e est la proportion d'indices $i = 1, \ldots, m$ tels que $e_i = e$.*

Exemple 50 *Pour la séquence d'états-effets $\mathcal{T} = ((x_1x_2x_3, \bar{x}_1), (x_1x_2x_3, \bar{x}_1), (x_1x_2x_3, x_2))$ la distribution induite $D_\mathcal{T}$ est telle que $D_\mathcal{T}(\bar{x}_1) = \frac{2}{3}$ et $D_\mathcal{T}(x_2) = \frac{1}{3}$.*

Définition 51 (précision) *Soit D une distribution d'effets et $\mathcal{T}$ une séquence d'états-effets. Alors $\mathcal{T}$ est dit ϵ-précise pour D si $\|D_\mathcal{T} - D\|_1 \leq \epsilon$.*

Exemple 52 (suite) *Soit une distribution d'effets D telle que $D(\bar{x}_1) = 0{,}6$ et $D(x_2) = 0{,}4$, et $\mathcal{T}$ la séquence d'états-effets de l'exemple 50 alors $\mathcal{T}$ est $\frac{2}{15}$-précise pour D.*

$$\|D_\mathcal{T} - D\|_1 = |D_\mathcal{T}(\bar{x}_1) - D(\bar{x}_1)| + |D_\mathcal{T}(x_2) - D(x_2)| = \left|\frac{2}{3} - 0{,}6\right| + \left|\frac{1}{3} - 0{,}4\right| = \frac{1}{15} + \frac{1}{15} = \frac{2}{15}.$$

Si les m effets de $\mathcal{T}$ sont tirés indépendamment selon la même distribution D, alors pour $\epsilon > 0$, [Weissman *et al.*, 2003] ont montré l'inégalité suivante, variante de la borne de Hoeffding [Hoeffding, 1963] pour les variables aléatoires multivaluées :

$$\Pr(\|D_\mathcal{T} - D\|_1 \geq \epsilon) \leq (2^E - 2)e^{-m\epsilon^2/2} \tag{7.1}$$

Avec E le nombre d'effets avec probabilité non nulle dans D. En substituant la partie gauche de cette inégalité par δ, on en déduit que pour $\epsilon > 0$, $\mathcal{T}$ est ϵ-précise pour D avec probabilité $1-\delta$ quand

$$m \geq \frac{2}{\epsilon^2} \ln \frac{2^E - 2}{\delta}$$

L'utilisation de la norme L_1 pour mesurer la précision est justifiée par le fait qu'une approximation de la fonction de transition d'un PDM donne des garanties sur la qualité de la fonction de valeur calculée avec cette approximation. Le lemme suivant, dont des variantes sont décrites dans [Kearns et Singh, 1998, simulation lemma] et [Li *et al.*, 2011, lemme 9] borne l'erreur introduite par de telles approximations.

Lemme 53 *Soient P et $\hat{P}$ deux fonctions de transitions telles que pour tout état s et action a, $\left\|P(\cdot|s,a) - \hat{P}(\cdot|s,a)\right\|_1 \leq \alpha$. Alors les fonctions de valeur optimales V^* et $\hat{V}^*$ respectives des PDM $M = \langle \mathcal{S},\mathcal{A},P,R\rangle$ et $\hat{M} = \langle \mathcal{S},\mathcal{A},\hat{P},R\rangle$ vérifient pour tout état $s \in \mathcal{S}$:*

$$\left|V^*(s) - \hat{V}^*(s)\right| \leq \frac{\alpha\gamma R_{\max}}{(1-\gamma)^2}$$

avec $R_{\max} = \max_{s,a} R(s,a)$.

Démonstration. Voir annexe A. □

On peut alors déduire le lemme suivant :

Lemme 54 *Soit $\epsilon \geq 0$, les PDM $M = \langle \mathcal{S},\mathcal{A},P,R\rangle$ et $\hat{M} = \langle \mathcal{S},\mathcal{A},\hat{P},R\rangle$ tels que pour tout état s et action a,*

$$\left\|P(\cdot|s,a) - \hat{P}(\cdot|s,a)\right\|_1 \leq \epsilon \frac{(1-\gamma)^2}{\gamma R_{\max}}$$

ont leurs fonctions de valeur respectives V^* et $\hat{V}^*$ qui vérifient pour tout état s :

$$\left|V^*(s) - \hat{V}^*(s)\right| \leq \epsilon$$

Dans le cadre proposé, l'agent n'observe pas directement les effets mais les états résultant de l'application de ces effets dans un état donné. Ainsi ce qu'observe l'agent pour un état s et un effet e est l'intervalle $[(\mathit{apply}(e,s)) \setminus s, \mathit{apply}(e,s)]$. Ceci revient à observer la transition de s à $s' = \mathit{apply}(e,s)$ (cf. lemme 43).

Définition 55 (observations induites) *Soit une séquence d'états-effets $\mathcal{T} = ((s_i, e_i))_{i=1,\ldots,m}$.* L'observation induite *d'un couple (s_i, e_i) est l'intervalle d'effets $[s'_i \setminus s_i, s'_i]$ avec $s'_i = \mathit{apply}(e_i,s_i)$. La famille d'observations induites par $\mathcal{T}$, notée $\mathcal{O}_\mathcal{T}$ est définie par $\mathcal{O}_\mathcal{T} = \{[s'_i \setminus s_i, s'_i] | (s_i, e_i) \in \mathcal{T}\}$.*

Quand bien même l'agent ne peut observer directement les effets, nous utiliserons parfois la formulation suivante des observations induites lors de leur étude formelle :

$$\mathcal{O}_\mathcal{T} = \{[e_i \setminus s_i, s'_i] | (s_i, e_i) \in \mathcal{T}\}$$

Notons aussi qu'étant donné une famille d'observations $\mathcal{O}$, la séquence d'états-effets

$$\mathcal{T}_{\mathcal{O}} = \{(s_i, e_i) | [e_i \setminus s_i, s'_i] \in \mathcal{O}\}$$

induit $\mathcal{O}$, c'est-à-dire $\mathcal{O}_{\mathcal{T}_{\mathcal{O}}} = \mathcal{O}$. L'ambiguïté des observations a pour conséquence que toute séquence d'états-effets de la forme :

$$\mathcal{T}' = \{(s_i, e) | [e_i \setminus s_i, s'_i] \in \mathcal{O} \text{ et } e \text{ un effet quelconque de } [e_i \setminus s_i, s'_i]\}$$

induit $\mathcal{O}$. Alors même si $D_{\mathcal{T}}$ est ϵ-précise pour une distribution D les distributions induites par ces séquences d'états-effets $\mathcal{T}'$ ne sont pas nécessairement ϵ-précises pour D comme le montre l'exemple suivant.

Exemple 56 *Soit* $D = \{x_1 : 0{,}5, \emptyset : 0{,}5\}$ *et soit* $\mathcal{T} = ((x_1,x_1), (x_1, \emptyset))$ *les observations induites par* $\mathcal{T}$ *sont* $\mathcal{O}_{\mathcal{T}} = \{[\emptyset, x_1], [\emptyset, x_1]\}$. *La distribution induite par* $\mathcal{T}$ *:* $D_{\mathcal{T}} = \{x_1 : 0{,}5, \emptyset : 0{,}5\}$ *est* 0*-précise pour* D. *Cependant, la séquence d'états-effets* $\mathcal{T}' = ((x_1,x_1), (x_1,x_1))$ *induit aussi* $\mathcal{O}_{\mathcal{T}}$. *Pourtant* $D_{\mathcal{T}'} = \{x_1 : 1\}$ *est seulement* 1*-précise pour* D.

Définition 57 (fréquence) *La* fréquence *d'une observation* $o = [s' \setminus s, s'] \in \mathcal{O}_{\mathcal{T}}$, *notée* $f_{o,\mathcal{O}_{\mathcal{T}}}$ *(ou* f_o *quand* $\mathcal{O}_{\mathcal{T}}$ *est clairement défini dans le contexte) correspond à la proportion d'indices* i *tels que* $[s'_i \setminus s_i, s'_i] = o$ *dans* $\mathcal{O}_{\mathcal{T}}$ *:*

$$f_{o,\mathcal{O}_{\mathcal{T}}} = \frac{|\{o_i = o | o_i \in \mathcal{O}_{\mathcal{T}}\}|}{m}$$

Une conséquence de l'observation d'intervalles d'effets - et non pas des effets eux-mêmes - est que pour une famille d'observations générée par l'environnement, *plusieurs* distributions peuvent en être induites. L'ensemble de ces distributions possibles est capturé par la définition suivante.

Définition 58 (ensemble de distributions observées) *Soit une famille d'observations* $\mathcal{O} = ([s'_i \setminus s_i, s'_i] | i = 1, \ldots, m)$. L'ensemble de distributions observées *de* $\mathcal{O}$, *noté* $[\mathcal{O}]$ *est défini comme l'ensemble de toutes les distributions* D *telles qu'il existe une séquence d'états-effets* $\mathcal{T}$ *vérifiant :*

- $\mathcal{O}$ *est induite par* $\mathcal{T}$ *:* $\mathcal{O} = \mathcal{O}_{\mathcal{T}}$ *et ;*
- D *est induite par* $\mathcal{T}$ *:* $D = D_{\mathcal{T}}$.

Cependant, cette définition nous permet de garantir que pour une séquence d'états-effets $\mathcal{T}$, la distribution induite $D_{\mathcal{T}}$ fait toujours partie des distributions observées.

Proposition 59 *Soit* $\mathcal{T}$ *une séquence d'états-effets. Alors* $D_{\mathcal{T}} \in [\mathcal{O}_{\mathcal{T}}]$.

7.2 Distributions plausibles

Dans cette section, nous présentons la *déviation*, une mesure de la plausibilité qu'une distribution donnée explique une famille d'observation données. Bien que la distribution qui a généré les observations soit toujours parmi les plus plausibles, nous verrons cependant que l'ambiguïté des effets implique qu'il peut exister d'autres distributions parfaitement plausibles, mais parfois arbitrairement éloignées de la distribution cible.

7.2.1 Calcul de la plausibilité

Nous donnons tout d'abord la définition formelle de la *déviation* d'une distribution à une famille d'observations. Plus cette valeur de déviation est faible, plus il est *plausible* que cette distribution soit en effet celle qui a généré ces observations.

Définition 60 (déviation) *Soit D une distribution d'effets et $\mathcal{O}$ une famille d'intervalles observés. La* déviation *de O par rapport à D est définie comme :*

$$\Delta(\mathcal{O}|D) = \min_{D' \in [\mathcal{O}]} \|D' - D\|_1$$

Cette mesure est optimiste, puisqu'elle considère le meilleur des cas possibles parmi les distributions consistantes avec les observations. L'avantage de cette mesure est qu'elle est non biaisée et par conséquent la distribution cible qui génère les observations est toujours plausible comme le montre la proposition suivante.

Proposition 61 *Soit $\mathcal{T}$ une séquence d'états-effets générée par une distribution D. Alors*

$$\Delta(\mathcal{O}_\mathcal{T}|D) \leq \|D_\mathcal{T} - D\|_1$$

Démonstration. D'après la proposition 59 nous avons $D_\mathcal{T} \in [\mathcal{O}_\mathcal{T}]$, par conséquent nous avons d'après la définition de la déviation :

$$\begin{aligned}\Delta(\mathcal{O}_\mathcal{T}|D) &= \min_{D' \in [\mathcal{O}_\mathcal{T}]} \|D' - D\|_1 \\ &\leq \|D_\mathcal{T} - D\|_1\end{aligned}$$

□

De cette proposition, nous pouvons déduire que pour une séquence $\mathcal{T}$ générée par une distribution D, l'estimation $D_\mathcal{T}$ - que nous aurions pu obtenir si les effets étaient observables - a une déviation nulle pour $\mathcal{O}_\mathcal{T}$ car $\|D_\mathcal{T} - D_\mathcal{T}\|_1 = 0$. Par contre, comme le montre l'exemple suivant, il existe des distributions de déviation nulle qui sont arbitrairement éloignées de la distribution cible.

Exemple 62 (déviation) *Soit $D = \{x_1 : \frac{2}{3}, \bar{x}_2 : \frac{1}{3}\}$ une distribution d'effets, et*

$$\mathcal{T} = ((\bar{x}_1 x_2, x_1), (x_1 x_2, x_1), (x_1 x_2, \bar{x}_2))$$

une séquence d'états-effets induite par D. La famille d'observations correspondante est

$$\mathcal{O}_\mathcal{T} = \{[x_1, x_1 x_2], [\emptyset, x_1 x_2], [\bar{x}_2, x_1 \bar{x}_2]\}$$

La déviation de D est $\Delta(\mathcal{O}_\mathcal{T}, D) = 0$ car $D \in [\mathcal{O}_\mathcal{T}]$. Or pour la distribution $P = \{x_1 x_2 : \frac{2}{3}, x_1 \bar{x}_2 : \frac{1}{3}\}$ sa déviation est aussi $\Delta(\mathcal{O}_\mathcal{T}, P) = 0$ car $P \in [\mathcal{O}_\mathcal{T}]$. P est donc plausible malgré qu'elle soit maximalement[8] *éloignée de D : $\|D - P\|_1 = 2$.*

8. La plus grande distance L_1 entre deux distributions de probabilité est 2.

`Variables :`	$c_{e,o}$	$\forall o \in \mathcal{O}, e \in o \cap E$
`Minimiser :`	$\sum_{e \in D} \lvert D(e) - \sum_{o \in \mathcal{O}} c_{e,o} \rvert + \sum_{o \in \mathcal{O}, o \cap E = \emptyset} f_{o,\mathcal{O}}$	
`Contraintes :`	$c_{e,o} \geq 0$	$\forall o \in \mathcal{O}, e \in o \cap E$
	$\sum_{e \in o \cap E} c_{e,o} = f_{e,\mathcal{O}}$	$\forall o \in \mathcal{O}, o \cap E \neq \emptyset$

FIGURE 7.1 – Programme linéaire pour calculer la déviation $\Delta(\mathcal{O}|D)$ d'une famille d'observations $\mathcal{O}$ à une distribution d'effets D.

7.2.2 Calcul par programme linéaire

La notion de déviation est utilisée comme mesure de la plausibilité qu'une distribution d'effets ait induit une famille d'observations données. En pratique, elle peut être calculée par un programme linéaire dont l'objectif est de déterminer la distance minimale entre une distribution D sur un ensemble E d'effets et une distribution $D' \in [\mathcal{O}]$. Il faut donc décomposer la contrainte $D' \in [\mathcal{O}]$ sous la forme d'un ensemble de contraintes linéaires.

Pour chaque effet $e \in E$ et tous les intervalles $o \in \mathcal{O}$ qui contiennent e, on ajoute une variable de *contribution* $c_{e,o}$ de la fréquence de o à la probabilité de e/ Ces variables représentent dans quelle proportion un effet e a généré l'intervalle o. Pour chaque intervalle o, ces variables sont sujettes aux contraintes :

- $\sum_{e \in o \cap E} c_{e,o} = f_{o,\mathcal{O}}$: on répartit la fréquence de o entre tous les effets e qui sont dans E et dans o ;
- $c_{e,o} \geq 0$: on n'autorise pas de contribution négative.

À l'aide de ces variables de contribution, la probabilité d'un effet e selon D', est donc la somme des contributions de tous les intervalles à e :

$$D'(e) = \sum_{o \in \mathcal{O}} c_{e,o}$$

S'il existe des intervalles o tels qu'aucun effet $e \in o$ n'est pas dans E alors la fréquence $f_{o,\mathcal{O}}$ de ces intervalles est directement ajoutée à la déviation car pour ces effets $D(e) = 0$ et donc $\sum_{e \in o} |D(e) - c_{e,o}| = \sum_{e \in o} c_{e,o} = f_{o,\mathcal{O}}$. Le programme linéaire résultant est résumé sur la figure 7.1.

Toutefois, comme la fonction objectif de ce programme contient des valeurs absolues, ce n'est pas un objectif linéaire. Il est cependant possible d'obtenir un programme linéaire équivalent en introduisant pour chaque valeur absolue $|x - y|$ apparaissant dans l'objectif une variable auxiliaire z et les contraintes $z \geq x - y$ et $z \geq y - x$. Comme l'objectif est à minimiser, cette reformulation est correcte (ce n'est pas le cas pour une maximisation). Après cette reformulation, on obtient le programme linéaire de la figure 7.2.

`Variables :`	$c_{e,o}$	$\forall o \in \mathcal{O}, e \in o \cap E$
	δ_e	$\forall e \in E$
`Minimiser :`	$\sum_{e \in E} \delta_e + \sum_{o \in \mathcal{O}, o \cap E = \emptyset} f_{o,\mathcal{O}}$	
`Contraintes :`	$c_{e,o} \geq 0$	$\forall o \in \mathcal{O}, e \in o \cap E$
	$\sum_{e \in o \cap E} c_{e,o} = f_{e,\mathcal{O}}$	$\forall o \in \mathcal{O}, o \cap E \neq \emptyset$
	$\delta_e \geq D(e) - \sum_{o \in \mathcal{O}} c_{e,o}$	$\forall e \in E$
	$\delta_e \geq \sum_{o \in \mathcal{O}} c_{e,o} - D(e)$	$\forall e \in E$

FIGURE 7.2 – Programme linéaire reformulé pour calculer la déviation $\Delta(\mathcal{O}|D)$ d'une famille d'observations $\mathcal{O}$ à une distribution d'effets D.

7.3 Variance d'un ensemble d'observations

Dans cette section, nous dérivons la notion de déviation d'une distribution à des d'observations en une notion plus générale de *variance* d'un *ensemble* d'observations. L'objectif de cette notion de variance est de déterminer la plausibilité qu'un ensemble d'observations ait été généré par la même distribution. La variance est formellement définie comme suit.

Définition 63 (variance) *Soit $\mathcal{O}_1, \ldots, \mathcal{O}_q$ un ensemble de familles d'observations et $\epsilon \in [0,2]$. Alors $\mathcal{O}_1, \ldots, \mathcal{O}_q$ dont dits* ϵ-variants *s'il existe une distribution d'effets D telle que pour tout $i = 1, \ldots, q$, $\Delta(\mathcal{O}_i|D) \leq \epsilon$.*

La variance s'inspire de la notion de *centre de Chebyshev* qui est le centre de la boule de plus petit rayon qui contient un ensemble de points. Dans notre cadre, les points en question sont des distributions de probabilités sur des effets. Le centre de Chebyshev en norme L_1 de $D_1, \ldots, D_k$ est défini comme

$$D = \arg\min_{D'} \max_{D_i} \left\| D' - D_i \right\|_1$$

Cependant, la variance est différente dans le sens où nous cherchons une distribution qui serait le centre d'un ensemble d'observations. L'ambiguïté a pour conséquence qu'une observation $\mathcal{O}_i$ définit plusieurs distributions : $[\mathcal{O}_i]$; par conséquent, pour les observations $\mathcal{O}_1, \ldots, \mathcal{O}_q$ on recherche un centre D tel que

$$\begin{aligned} D &= \arg\min_{D'} \max_{\mathcal{O}_i} \min_{D_i \in [\mathcal{O}_i]} \left\| D' - D_i \right\|_1 \\ &= \arg\min_{D'} \max_{\mathcal{O}_i} \Delta(\mathcal{O}_i|D') \end{aligned}$$

Ainsi D peut être vu comme le centre de Chebyshev en norme Δ de $\mathcal{O}_1, \ldots, \mathcal{O}_q$. En concaténant une instance du programme linéaire de la déviation pour chaque $\mathcal{O}_i$, la variance peut se calculer par un programme linéaire (figure 7.3). Le max présent dans la fonction objectif de ce programme est exprimé sous forme de contraintes afin de préserver la linéarité. Ainsi, pour minimiser la fonction $\max\{x_1, \ldots, x_k\}$, on introduit une variable supplémentaire x avec les contraintes $x \geq x_i, i = 1, \ldots, k$ et l'objectif devient alors la minimisation de x.

`Variables :`	$c^i_{e,o}$	$\forall i = 1,\dots,k, o \in \mathcal{O}_i, e \in o \cap E$
	$D(e)$	$\forall e \in E$
`Minimiser :`	$\max_{i=1,\dots,k} \sum_{e\in E} \left\lvert D(e) - \sum_{o\in\mathcal{O}_i} c^i_{e,o} \right\rvert$	
`Contraintes :`	$c^i_{e,o} \geq 0$	$\forall i = 1,\dots,k, o \in \mathcal{O}_i, e \in o \cap E$
	$\sum_{e\in o\cap E} c^i_{e,o} = f_{e,\mathcal{O}_i}$	$\forall i = 1,\dots,k, o \in \mathcal{O}_i$
	$\sum_{e\in E} D(e) = 1$	

FIGURE 7.3 – Programme linéaire pour calculer la variance d'un ensemble de familles d'observations.

Cette fois, nous ne connaissons pas à l'avance l'ensemble d'effets E sur lequel est défini la distribution D. Par conséquent on peut supposer que E est l'ensemble des 3^n termes possibles sur $\mathcal{X}$. Le nombre de variables à optimiser est alors exponentiel. Cependant, en choisissant le plus petit ensemble E tel que pour tout intervalle $o \in \mathcal{O}_1,\dots,\mathcal{O}_s$ il existe un effet $e \in E$ tel que $e \in o$, le nombre de variables à traiter est grandement réduit. Bien évidemment, dans le pire des cas E peut toujours être de taille exponentielle, mais il reste très petit en pratique. Un tel ensemble d'effets peut être calculé par l'algorithme REDUIRE (algorithme 12).

Algorithme 12 : Ensemble suffisant d'effets pour la variance

REDUIRE2 $(\mathcal{O}_1, \mathcal{O}_2)$
début
  $\mathcal{O} \leftarrow \emptyset$
  reste $\leftarrow \mathcal{O}_1 \cup \mathcal{O}_2$
  pour chaque $o_1 \in \mathcal{O}_1$ **faire**
    pour chaque $o_2 \in \mathcal{O}_2$ **faire**
      si $o_1 \cap o_2 \neq \emptyset$ **alors**
        $\mathcal{O} \leftarrow \mathcal{O} \cup \{o_1 \cap o_2\}$
        reste $\leftarrow$ reste $\setminus \{o_1, o_2\}$
  $\mathcal{O} \leftarrow \mathcal{O} \cup$ reste
  retourner $\{o \in \mathcal{O} |\ \nexists\ :\ o' \in \mathcal{O} o' \subset o\}$
fin
REDUIRE $(\mathcal{O}_1, \dots, \mathcal{O}_q)$
début
  $\mathcal{O} \leftarrow$ REDUIRE2 $(\mathcal{O}_1, \mathcal{O}_2)$
  pour $i = 3,\dots,q$ **faire**
    $\mathcal{O} \leftarrow$ REDUIRE2 $(\mathcal{O}, \mathcal{O}_i)$
  retourner $\{e \in o$ choisi arbitrairement$|o \in \mathcal{O}\}$
fin

7.3.1 Utilisation de la variance pour l'apprentissage de conditions d'actions

Dans le cadre plus général où l'on souhaite apprendre des actions contenant des conditions, c'est-à-dire dans toute l'expressivité du STRIPS probabiliste, nous pouvons utiliser la variance pour :

– identifier les conditions des actions et

– pour chacune de ces conditions, déterminer une distribution d'effets associée.

L'identification des conditions est basée sur le principe de SLF-Rmax dont de plus amples détails sont donnés dans le chapitre précédent. La principale différence est qu'au lieu de manipuler des probabilités binomiales (la probabilité de mettre une variable à 0 ou 1) nous manipulons directement des distributions de probabilités sur un ensemble d'effets. De façon similaire à SLF-RMAX qui nécessite une borne sur le nombre de parents de chaque variable, notre algorithme requiert en entrée une borne k sur le nombre de variables apparaîssant dans les conditions de l'action. De cette façon, il existe une représentation équivalente de cette action qui possède 2^k conditions, une pour chaque affectation à ces variables. L'algorithme considère donc tous les ensembles possibles de k variables comme candidats pour définir les conditions.

Ainsi pour un état s et chaque ensemble candidat K de k variables, nous calculons la variance de l'ensemble d'observations $O_K = \{\mathcal{O}^{s[K \cup X]} | X \subseteq \mathcal{X} \setminus K, |X| = k\}$. Ainsi si O_K a une faible variance, il est plausible de considérer que ces observations ont été générées par la même distribution et donc que K est bien l'ensemble des variables apparaissant dans les conditions de l'action. En effet, si K n'est pas le bon ensemble de variables, les observations O_K sont générées par différentes distributions ce qui *devrait* aboutir à une grande variance. Néanmoins, cette approche reste heuristique car l'ambiguïté des observations peut toutefois amener à trouver un mauvais ensemble de variables qui ait une faible variance et qui soit injustement considérée comme une hypothèse plausible. La seule garantie que fournit cette heuristique est que le véritable ensemble de variables qui conditionne l'action aura bien une faible variance avec grande probabilité. L'algorithme LP-CONDITIONS (algorithme 13) permet de prédire la fonction de transition d'une action étant donnée la connaissance d'une borne k sur le nombre de variables apparaissant dans les conditions. Cet algorithme ne donne aucune garantie sur la qualité des prédictions faites bien qu'il ait le même principe de fonctionnement qu'un algorithme KWIK. L'algorithme prédit $\perp$ (je ne sais pas) jusqu'à avoir collecté m observations pour chaque terme de taille $2k$ consistant avec l'état courant. De cette façon, nous pouvons intégrer cet algorithme directement dans KWIK-RMAX (algorithme 11) pour obtenir l'algorithme PSO-LP pour l'apprentissage par renforcement.

Définition 64 (algorithme PSO-LP) *L'algorithme d'apprentissage par renforcement PSO-LP est défini comme l'intégration dans* KWIK-RMAX *(algorithme 11) d'une instance de l'algorithme* LP-CONDITIONS *(algorithme 13) pour chaque action du PDM. La fonction de récompense est supposée connue à l'avance ainsi qu'une borne k sur le nombre de variables apparaissant dans les conditions de la représentation STRIPS probabiliste de l'action. L'algorithme prend en paramètre le seuil d'exploitation m qui est transmis à chaque instance de* LP-CONDITIONS.

Algorithme 13 : LP-CONDITIONS

Entrées : m, k

début

$\mathcal{O} \leftarrow \emptyset$

pour chaque *pas de temps* $t = 1,2,\ldots$ **faire**

Observer s_t

si $\exists x \in T_{s_t}^{2k} \ : \ |\mathcal{O}^x| < m$ **alors**

Prédire $\perp$

Observer s'_t

$\mathcal{O} \leftarrow \mathcal{O} \cup \{[s'_t \setminus s_t, s'_t]\}$

sinon

pour chaque *condition candidate* $c \subseteq s, |c| = k$ **faire**

$O \leftarrow \{\mathcal{O}^{c \cup x} | x \in T_{s_t \setminus c}^{k}\}$

Calculer la variance σ_c des observations O et leur centre D_c avec le programme linéaire de la figure 7.3 ;

$c^* \leftarrow \arg\min_c \sigma_c$

Prédire D_{c^*}

fin

Chapitre 8

KWIK-apprentissage d'actions STRIPS probabilistes

Sommaire

Dans ce chapitre nous présentons une méthode KWIK pour l'apprentissage de la fonction de transition d'un PDM avec pour seules observations des transitions de la forme (s, s'). Les algorithmes que nous présentons exploitent la compacité d'une représentation en STRIPS probabiliste de l'action pour généraliser leur expérience et pouvoir prédire plus rapidement. Comme dans le chapitre précédent, nous considérons dans un premier temps que l'action à apprendre n'a pas de conditions et se réduit donc à une simple distribution d'effets.

Nos contributions se découpent en plusieurs parties, l'apprentissage des effets, puis de leurs probabilités et enfin des conditions d'action.

8.1 Identification des effets

Nous présentons tout d'abord un algorithme KWIK pour apprendre l'ensemble d'effets d'une distribution. Notre approche consiste à laisser la possibilité à l'apprenant de forcer certaines propriétés sur les états à partir desquels il reçoit les observations.

Cette section introduit la notion de *k-clôture* d'un ensemble d'effets. Cette notion peut être vue comme une mesure de complexité pour l'apprentissage d'un ensemble d'effets. Nous introduisons ensuite un algorithme KWIK capable d'apprendre un ensemble d'effets k-clos dont la complexité en exemples a seulement une dépendance exponentielle en k et non pas en n, le nombre de variables. Ainsi, les ensembles d'effets k-clos pour un k borné sont apprenables en temps polynomial.

8.1.1 Stratégies d'exploration

Notre approche pour influencer les états à partir desquels les observations sont reçues est de faire en sorte d'en collecter suffisamment depuis des états $s \supseteq t$ *pour tout terme* t *de taille fixée.* L'idée de cette approche vient de l'observation que certains états produisent plus d'ambiguïtés que d'autres. On cherche donc à varier le plus possible les états à partir desquels sont générées les transitions observées.

Ainsi, l'apprenant demandera des observations pour chaque terme t possible. Si l'apprenant obtient suffisamment d'observations pour chacun de ces termes, alors, sous certaines conditions, il pourra apprendre une bonne approximation de l'ensemble d'effets. Pour ce faire, nous utilisons les définitions suivantes.

Définition 65 (projection) *Soit* $\mathcal{O}$ *une séquence d'observations et* t *un terme. La* projection *de* $\mathcal{O}$ *sur* t*, notée* $\mathcal{O}^t$ *est l'ensemble des observations de* $\mathcal{O}$ *induites par des états* s *tels que* $t \subseteq s$*.*

$$\mathcal{O}^t = \{[s' \setminus s, s'] \in \mathcal{O} | t \subseteq s\}$$

Définition 66 (effets plausibles) *Soit* $\mathcal{O}$ *une séquence d'observations et* $\mathcal{T}$ *un ensemble de termes. Les effets plausibles de* $\mathcal{O}$ *selon* $\mathcal{T}$ *sont définis comme tous ceux apparaissant dans au moins un intervalle de* $\mathcal{O}^t$ *pour tout* $t \in \mathcal{T}$ *:*

$$\hat{E}(\mathcal{O},\mathcal{T}) = \bigcap_{t \in \mathcal{T}} \left\{ e | e \in o, o \in \mathcal{O}^t \right\}$$

L'intuition derrière la notion d'effet plausible est que tous les effets sont tirés depuis une même distribution D, indépendamment de l'état. Par conséquent, chaque $\mathcal{O}^t$ contient des intervalles qui à leur tour contiendront ces effets. Il en découle que ces effets apparaîtront dans $\hat{E}(\mathcal{O},\mathcal{T})$.

8.1.2 Propriétés des ensembles d'effets

Une première notion concernant les ensembles d'effets est celle *d'ensemble équivalent minimal* pour un état donné. Pour un ensemble d'effets E et un état s, l'ensemble équivalent minimal est

le plus petit ensemble d'effets E' qui contient les effets les plus courts qui soient équivalents à ceux de E dans l'état s.

Définition 67 (ensemble équivalent minimal) *Soit E un ensemble d'effets et s un état.* L'ensemble équivalent minimal *de E pour s est défini comme :*

$$[E]_s = \{e \setminus s | e \in E\}$$

Cette notion a pour but d'assouplir l'égalité de deux ensembles d'effets. Comme nous le verrons par la suite, lors de l'apprentissage d'un ensemble d'effets E pour un état s, E ne sera pas nécessairement apprenable exactement. Cependant, si l'on obtient un ensemble $E' \neq E$ mais tel que $[E]_s = [E']_s$ alors E' sera suffisant pour déterminer les états successeurs de s.

Exemple 68 (ensemble équivalent minimal) *Pour l'ensemble d'effets $E = \{x_1\bar{x}_2x_3, x_2x_3\}$ et l'état $s = \bar{x}_1x_2x_3$, l'ensemble équivalent minimal de E pour s est $[E]_s = \{x_1\bar{x}_2, \top\}$. Pour l'ensemble $E' = \{x_1\bar{x}_2x_3, x_2x_3, \bar{x}_2\}$ nous avons $[E']_s = [E]_s$ alors que $E' \neq E$.*

Nous introduisons maintenant formellement la notion de k-clôture d'un ensemble d'effets. Cette clôture se base sur la notion de compatibilité entre effets :

Définition 69 (compatibilité) *Soit deux effets e et e' et un terme t. Alors e et e' sont dits* compatibles *sur t si $e \cap t = e' \cap t$ et e et e' sont consistants.*

Informellement, le fait que deux effets e et e' soient compatibles sur un terme t indique dans une certaine mesure la possibilité pour l'environnement de « tromper » l'apprenant pour qu'il *confonde* e et e' sur des observations générées par certains états de la forme $s \supseteq \bar{t}$. Comme $e' \sim_t e$ implique que e et e' sont consistants alors $e \cup e'$ est un terme consistant lui aussi. On peut donc construire un état $s \supseteq (e \setminus t) \cup (e' \setminus t) \cup \bar{t}$. Or, pour un tel état s, nous avons $e \setminus s \subseteq e'$ et $e' \subseteq apply(e,s)$ (voir le lemme 96 en annexe). De fait, l'intervalle $o = [e \setminus s, apply(e,s)]$ contient *à la fois* e et e'.

Exemple 70 (compatibilité) *Prenons les effets $e_1 = x_1x_2$, $e_2 = x_1\bar{x}_2$ et $e_3 = x_1\bar{x}_3$. Alors e_1 et e_2 sont incompatibles car ils sont inconsistants : ils diffèrent sur la polarité de x_2. Par contre, e_1 et e_3 sont compatibles pour $t = x_1$:*

$$e_1 \cap t = x_1x_2 \cap x_1 = x_1 = x_1\bar{x}_3 \cap x_1 = e_3 \cap t$$

ils sont aussi compatibles pour $t' = \bar{x}_1\bar{x}_2x_3$ car ni e_1, ni e_3 ne contient de littéral de t' et donc :

$$e_1 \cap t' = e_3 \cap t' = \emptyset$$

Si l'on prend $t = x_1$, et l'état $s_1 = \bar{x}_1x_2\bar{x}_3x_4 \supseteq \bar{t}$ on obtient l'intervalle $o_1 = [e_1 \setminus s_1, apply(e_1,s_1)] = [x_1, x_1x_2\bar{x}_3x_4]$. Alors on a $e_1 \in o_1$ et $e_2 \in o_1$. De même, pour l'état $s_2 = \bar{x}_1x_2\bar{x}_3\bar{x}_4 \supseteq \bar{t}$ on obtient $o_2 = [e_2 \setminus s_2, apply(e_2,s_2)] = [x_1, x_1x_2\bar{x}_3\bar{x}_4]$ pour lequel $e_1 \in o_2$ et $e_2 \in o_2$. Les deux effets sont présents dans les deux intervalles.

Définition 71 (clôture) *Soit s un état, E un ensemble d'effets et $k \leq n$ un entier naturel. Un effet $e \in 3^{\mathcal{X}}$ appartient à la k-*clôture *de E pour s, notée $Cl_k(E,s)$ si, pour tous les termes t de taille k tels que $t \subseteq s$, il existe un effet $e' \in E$ tel que $e' \sim_{\bar{t}} e$.*

$$Cl_k(E,s) = \left\{ e \in 3^{\mathcal{X}} \middle| \forall t \in T_s^k,\ \exists e' \in E\ :\ e \sim_{\bar{t}} e' \right\}$$

Définition 72 (ensemble d'effets k-clos) *Un ensemble d'effets est k-*clos *si pour tout état s*

$$[Cl_k(E,s)]_s = [E]_s$$

Exemple 73 *Soit $\mathcal{X} = \{x_1,x_2\}$ et l'ensemble d'effets $E = \{\top, x_1, x_1x_2\}$ les clôtures de E sont*

$$\begin{aligned}
Cl_1(E, \bar{x}_1\bar{x}_2) &= \{\top, x_1, x_2, \bar{x}_1, \bar{x}_2, \bar{x}_1\bar{x}_2, x_1\bar{x}_2, x_1x_2\}\\
Cl_1(E, x_1\bar{x}_2) &= \{\top, x_1, x_2, \bar{x}_2, x_1\bar{x}_2, x_1x_2\}\\
Cl_1(E, \bar{x}_1x_2) &= \{\top, x_1, \bar{x}_1, x_2, \bar{x}_1x_2, x_1x_2\}\\
Cl_1(E, x_1x_2) &= \{\top, x_1, x_2, x_1x_2\}\\
Cl_2(E, \bar{x}_1\bar{x}_2) &= \{\top, x_1, \bar{x}_1, \bar{x}_2, \bar{x}_1\bar{x}_2, x_1\bar{x}_2, x_1x_2\}\\
Cl_2(E, x_1\bar{x}_2) &= \{\top, x_1, x_2, \bar{x}_2, x_1\bar{x}_2, x_1x_2\}\\
Cl_2(E, \bar{x}_1x_2) &= \{\top, x_1, \bar{x}_1, x_2, \bar{x}_1x_2, x_1x_2\}\\
Cl_2(E, x_1x_2) &= \{\top, x_1, x_2, x_1x_2\}
\end{aligned}$$

Les ensembles équivalents minimaux de ces clôtures sont :

$$\begin{aligned}
[Cl_1(E, \bar{x}_1\bar{x}_2)]_{\bar{x}_1\bar{x}_2} &= \{\top, x_1, x_2, x_1x_2\}\\
[Cl_1(E, x_1\bar{x}_2)]_{x_1\bar{x}_2} &= \{\top, x_2\}\\
[Cl_1(E, \bar{x}_1x_2)]_{\bar{x}_1x_2} &= \{\top, x_1\}\\
[Cl_1(E, x_1x_2)]_{x_1x_2} &= \{\top\}\\
[Cl_2(E, \bar{x}_1\bar{x}_2)]_{\bar{x}_1\bar{x}_2} &= \{\top, x_1, x_1x_2\}\\
[Cl_2(E, x_1\bar{x}_2)]_{x_1\bar{x}_2} &= \{\top, x_2\}\\
[Cl_2(E, \bar{x}_1x_2)]_{\bar{x}_1x_2} &= \{\top, x_1\}\\
[Cl_2(E, x_1x_2)]_{x_1x_2} &= \{\top\}
\end{aligned}$$

L'ensemble d'effets E est donc 2-clos.

Quelques cas particuliers de clôtures :

Exemple 74 *Soit $\mathcal{L} = \{\ell_1, \ldots, \ell_m\}$ un ensemble de littéraux consistants.*

- *L'ensemble d'effets $E = \{\top, \ell_1, \ell_1\ell_2, \ldots, \ell_1\ell_2 \cdots \ell_m\}$ est 2-clos.*
- *L'ensemble d'effets $E = \mathcal{P}(\mathcal{L})$ est 1-clos.*
- *L'ensemble d'effets $E = \{\ell_1, \ell_2, \ldots, \ell_m\}$ est au moins m-clos.*
- *L'ensemble d'effets $E = \{\top, \ell_1, \ell_2, \ldots, \ell_m\}$ est 2-clos.*

8.1.3 Clôture et effets plausibles

Nous établissons à présent le lien entre la clôture d'un ensemble d'effets et les effets plausibles déduits des observations.

Proposition 75 *Soit s un état, $\mathcal{O}$ une séquence d'observations induite par une distribution D sur un ensemble d'effets E et soit $k \in \mathbb{N}$. Alors :*

$$\hat{E}(\mathcal{O}, T_s^k) \subseteq Cl_k(E,s)$$

Démonstration. Par construction, pour chaque effet $e \in \hat{E}(\mathcal{O},T_s^k)$ il existe pour tout terme t un intervalle $o^t \in \mathcal{O}^t$ de la forme $[e^t \setminus s^t, apply(e^t,s^t)]$ tel que $e \in o^t$. Par définition de l'appartenance d'un effet à un intervalle, nous avons : $e^t \setminus s^t \subseteq e$ et donc $(e^t \setminus s^t) \cap \bar{t} \subseteq e \cap \bar{t}$. Or $t \subseteq s^t$, donc $(e^t \setminus s^t) \cap \bar{t} = e^t \cap \bar{t}$, ce qui implique finalement

$$e^t \cap \bar{t} \subseteq e \cap \bar{t} \tag{i}$$

De plus, toujours par définition de l'inclusion dans un intervalle, nous avons : $e \subseteq apply(e^t,s^t)$ ce qui implique

$$e \text{ et } e^t \text{ sont mutuellement consistants} \tag{ii}$$

et aussi $e \cap \bar{t} \subseteq apply(e^t,s^t) \cap \bar{t}$. Sachant que $s^t \cap \bar{t} = \emptyset$, nous obtenons alors

$$e \cap \bar{t} \subseteq e^t \cap \bar{t} \tag{iii}$$

Pour conclure (i) et (iii) impliquent $e \cap \bar{t} = e^t \cap \bar{t}$. Ce qui en conjonction avec (ii) implique que e et e^t sont compatibles sur le terme $\bar{t}$ ($e \sim_{\bar{t}} e^t$), et donc $e \in Cl_k(E,s)$. □

La proposition ci-dessus montre que l'on ne peut découvrir que des effets de la k-clôture d'un ensemble d'effets cible. Nous faisons maintenant l'hypothèse suivante, que nous relaxons par la suite.

Hypothèse 76 *Pour toute séquence d'observations $\mathcal{O}$ induite par une distribution de probabilités sur un ensemble d'effets E, on suppose que pour tout terme t considéré, tout effet $e \in E$ a été échantillonné au moins une fois depuis un état $s \supseteq t$.*

Intuitivement, cette hypothèse permet de s'assurer que même les effets de très faible probabilité auront induit au moins un intervalle, ce qui les fera apparaître parmi les effets plausibles. Cette hypothèse sera relaxée en bornant la somme des probabilités de ces effets « manqués ».

Proposition 77 *Soit $\mathcal{O}$ une séquence d'observations induite par une distribution D sur un ensemble d'effets E telle que $\mathcal{O}$ vérifie l'hypothèse 76, soit $k \in \mathbb{N}$ et soit $\mathcal{T}$ un ensemble non vide de termes de taille k. Alors*

$$E \subseteq \hat{E}(\mathcal{O},\mathcal{T})$$

Démonstration. Soit $e \in E$, l'hypothèse 76 garantit qu'il existe un intervalle induit par e (et donc qui contient e) dans chaque $\mathcal{O}^t$ pour tout terme $t \in \mathcal{T}$, par conséquent $e \in \hat{E}(\mathcal{O},\mathcal{T})$. □

En conséquence des propositions 77 et 75 et toujours dans le cadre de l'hypothèse 76, on déduit que l'ensemble d'effets plausibles déterminé via les observations sur les termes de taille k vérifie pour tout état s :

$$E \subseteq \hat{E}(\mathcal{O}, T_s^k) \subseteq Cl_k(E,s) \tag{8.1}$$

8.1.4 KWIK-apprentissage des effets

Nous présentons maintenant un algorithme KWIK pour l'apprentissage d'un ensemble d'effets.

Rappelons tout d'abord quelques caractéristiques d'un apprenant KWIK. Un apprenant KWIK pour le problème d'apprentissage d'un ensemble d'effets caché E doit, quand l'environnement lui présente un état s, renvoyer une « bonne estimation » $\hat{E}$ de E qui correspond aux effets possibles pour s. S'il ne peut faire une telle prédiction il renvoie $\perp$. On pourrait *a priori* demander que si l'apprenant renvoie $\hat{E} \neq \perp$, alors $\hat{E} = E$. Cependant, une telle exigence est trop restrictive car il se peut que pour deux effets $\hat{e} \in \hat{E}$ et $e \in E$ on ait $\hat{e} \neq e$ mais que pour autant ils soient équivalents pour s : $apply(\hat{e},s) = apply(e,s)$. C'est pourquoi nous considérons qu'un tel $\hat{e}$ est correct même s'il n'appartient pas à E. Ainsi une prédiction $\hat{E}$ est définie comme exacte si $[\hat{E}]_s = [E]_s$.

De plus, les effets E sont échantillonnés selon une distribution D, il se peut donc que les effets e dont la probabilité $D(e)$ est très faible ne soient jamais tirés par l'environnement. Par conséquent, l'apprenant ne pourrait pas les mentionner dans ses prédictions. C'est pourquoi nous définissons la mesure d'erreur d'apprentissage comme la masse de probabilité totale des effets non prédits, à équivalence près pour l'état considéré.

Définition 78 (fonction d'erreur) *Soit D une distribution de probabilités sur un ensemble d'effets E, s un état et $\hat{E}$ un ensemble d'effets. Si $[\hat{E}]_s \subseteq [E]_s$, la mesure d'erreur associée à s entre E et $\hat{E}$ est définie comme la somme des probabilités des effets $e \in E$ qui n'ont pas d'équivalent pour s dans $\hat{E}$:*

$$d_s(\hat{E}, E) = \sum_{\substack{e \in E \\ e \setminus s \notin [\hat{E}]_s}} D(e)$$

sinon si $[\hat{E}]_s \not\subseteq [E]_s$, on définit $d_s(\hat{E}, E) = \infty$.

Cette définition de la mesure d'erreur considère que si un effet « de trop » est découvert alors l'apprentissage est un échec. C'est pourquoi nous devons garantir que notre apprenant ne prédit que des $\hat{E}$ tels que $[\hat{E}]_s \subseteq [E]_s$. Cependant, avec une séquence d'observations $\mathcal{O}$ et pour un entier k, $1 \leq k \leq n$ l'équation 8.1 nous indique que la prédiction $\hat{E} = \hat{E}(\mathcal{O},T_s^k)$ pour un état s est telle que $E \subseteq \hat{E} \subseteq Cl_k(E,s)$ et donc $[E]_s \subseteq [\hat{E}]_s \subseteq [Cl_k(E,s)]_s$. Dans ce cas pour un k fixé,

les seuls ensembles d'effets apprenables avec une erreur finie sont ceux tels que pour tout état s

$$[E]_s = [Cl_k(E,s)]_s$$

C'est-à-dire les ensembles d'effets k-clos. Par conséquent, pour un tel ensemble d'effets et dans le cas où $\mathcal{O}$ vérifie l'hypothèse 76, l'apprentissage est exact : $[\hat{E}]_s = [E]_s$ et donc $d_s(\hat{E}, E) = 0$. Au contraire, si l'hypothèse 76 n'est pas satisfaite alors $[\hat{E}]_s \subset [E]_s$ et nous avons $0 < d_s(\hat{E}, E) \leq 1$.

L'étape suivante consiste donc à lever l'hypothèse 76 en bornant le nombre d'observations qu'il faut recueillir pour garantir $d_s(\hat{E}, E) \leq \epsilon$ pour un $\epsilon > 0$ donné. Pour cela, rappelons que pour un terme t, il y a une séquence d'états-effets générés par l'environnement $\mathcal{T}_t$ qui a induit $\mathcal{O}^t$, par conséquent tout effet $e \in E$ de probabilité non-nulle selon la distribution induite $D_{\mathcal{T}_t}$ appartient à $\hat{E}(\mathcal{O}, \{t\})$.

Lemme 79 *Soit E un ensemble d'effets et $E' \subseteq E$, soient D et D' des distributions de probabilités sur E et E' respectivement, et $\alpha \geq 0$. Alors :*

$$\left\|D' - D\right\|_1 \leq \alpha \implies \sum_{e \in E \setminus E'} D(e) \leq \frac{\alpha}{2}$$

Démonstration.

$$\begin{aligned}
\left\|D' - D\right\|_1 \leq \alpha &\iff \sum_{e, D'(e)>0} \left|D'(e) - D(e)\right| + \sum_{e, D'(e)=0} D(e) \leq \alpha \\
&\implies \sum_{e, D'(e)>0} D'(e) - D(e) + \sum_{e, D'(e)=0} D(e) \leq \alpha \\
&\iff 1 - \sum_{e, D'(e)>0} D(e) + \sum_{e, D'(e)=0} D(e) \leq \alpha \\
&\iff 2 \sum_{e, D'(e)=0} D(e) \leq \alpha \\
&\iff \sum_{e, D'(e)=0} D(e) \leq \frac{\alpha}{2}
\end{aligned}$$

On conclut en observant que les effets $e \in E \setminus E'$ sont exactement ceux pour lesquels $D'(e) = 0$. □

En conséquence de ce lemme, si pour un état s, et chaque terme $t \in T_s^k$, la distribution $D_{\mathcal{T}_t}$ vérifie

$$\left\|D - D_{\mathcal{T}_t}\right\|_1 \leq \alpha$$

alors, sachant que $|T_s^k| = \binom{n}{k}$, on obtient :

$$d_s(\hat{E}, E) \leq \frac{\binom{n}{k}\alpha}{2}$$

Ainsi si l'on désire $d_s(\hat{E}, E) \leq \epsilon$ pour $\epsilon > 0$, il suffit que $D_{\mathcal{T}_t}$ vérifie :

$$\|D - D_{\mathcal{T}_t}\|_1 \leq \frac{2\epsilon}{\binom{n}{k}} \tag{8.2}$$

À l'aide de l'inégalité de [Weissman *et al.*, 2003] ceci est vérifié avec confiance $1 - \delta_0$ après que pour tout $t \in T_s^k$, $\mathcal{O}^t$ contient au moins

$$m_0 = \frac{\binom{n}{k}^2}{2\epsilon^2} \ln \frac{2^{|E|} - 2}{\delta_0}$$

intervalles observés. Comme il existe au total $2^k\binom{n}{k}$ termes de taille k, l'inégalité de Boole garantit qu'avec confiance $1-\delta$, l'équation 8.2 est vraie pour tout état s avec $\delta_0 = \frac{\delta}{2^k\binom{n}{k}}$ après que chaque $\mathcal{O}^t$ contienne au moins

$$m = \frac{\binom{n}{k}^2}{2\epsilon^2} \ln \frac{2^k\binom{n}{k}(2^{|E|} - 2)}{\delta} = O\left(\frac{n^{2k}}{\epsilon^2} \ln \frac{n^k 2^{|E|+k}}{\delta}\right) \tag{8.3}$$

intervalles observés.

Nous pouvons désormais conclure par le théorème suivant.

Théorème 80 *Soit D une distribution de probabilités sur un ensemble d'effets E tel que pour tout état s et un entier k fixé, $[E]_s = [Cl_k(E,s)]_s$. Alors pour chaque état s, l'algorithme 14 peut apprendre $[E]_s$ avec la borne KWIK*

$$B_E(\epsilon,\delta) = O\left(\frac{2^k n^{3k}}{\epsilon^2} \ln \frac{n^k 2^{|E|+k}}{\delta}\right)$$

Démonstration. On requiert (équation 8.3) $m = O\left(\frac{n^{2k}}{\epsilon^2} \ln \frac{n^k 2^{|E|+k}}{\delta}\right)$ exemples pour chaque famille d'observations $\mathcal{O}^t$, pour tout terme t de taille k. Sachant qu'il existe $2^k\binom{n}{k} = O(n^k 2^k)$ tels termes, on en déduit la borne du théorème. □

Algorithme 14 : KWIK-Effects

Entrées : m

$\mathcal{O} \leftarrow \emptyset$

pour chaque *pas de temps* $\tau = 1,2,\ldots$ **faire**

  Observer l'état s_τ

  si $\exists t \in T_{s_\tau}^k : |\mathcal{O}^t| < m$ **alors**

    Prédire $\perp$

    Observer s'_τ

    $\mathcal{O} \leftarrow \mathcal{O} \cup \{[s'_\tau \setminus s_\tau, s'_\tau]\}$

  sinon

    Prédire $[\hat{E}(\mathcal{O}, T_{s_\tau}^k)]_{s_\tau}$

8.2 Identification des probabilités d'effets

Dans cette section, nous nous intéressons au problème de l'apprentissage des probabilités des effets. De façon similaire à la k-clôture, nous introduisons la notion de k-séparabilité d'un ensemble d'effets. À partir de cette notion, nous verrons que la complexité en exemples des algorithmes présentés ne dépend exponentiellement que de k et non pas de n. Nous présentons un premier algorithme pour apprendre les probabilités d'un ensemble d'effets k-séparable connu à l'avance ou observable. Puis un second, qui combine à la fois l'apprentissage de l'ensemble d'effets et de leur probabilités afin d'apprendre une distribution.

8.2.1 KWIK-apprentissage des probabilités d'effets

Nous décrivons un algorithme KWIK pour l'apprentissage des probabilités des effets lorsque ces derniers sont connus à l'avance ou observables. Pour cela nous introduisons tout d'abord quelques définitions et propriétés caractérisant les ensembles d'effets.

Définition 81 (distribution réduite) *Soit D une distribution de probabilités sur un ensemble d'effets E et s un état. La distribution* réduite *à s, notée D^s est définie sur les effets $e \in [E]_s$ par :*

$$D^s(e) = \sum_{\substack{e' \in E \\ e = e' \setminus s}} D(e')$$

La notion de distribution réduite est utile dans le sens où elle préserve la fonction de transition. Ainsi pour une distribution d'effets D et un état s nous avons

$$T(s'|s,a) = \sum \{D(e) | apply(e,s) = s'\} = D^s(e)$$

où l'effet e de $D^s(e)$ est l'unique effet de $[E]_s$ tel que $s' = apply(e,s)$.

Définition 82 (terme séparateur) *Soit E un ensemble d'effets et s un état. Pour un effet $e \in E$, un terme $t \subseteq s$ est un* séparateur *de e pour E dans s si pour tout état $s' \supseteq t$ et tout effet $e' \in E$ tel que $e' \setminus s \neq e \setminus s$:*

$$[e \setminus s, apply(e,s)] \cap [e' \setminus s', apply(e',s')] = \emptyset \quad \text{(i)}$$

$$\wedge\ [e \setminus s', apply(e,s')] \cap [e' \setminus s, apply(e',s)] = \emptyset \quad \text{(ii)}$$

Intuitivement, la condition (i) de la séparabilité signifie « aucun effet équivalent à e pour s n'apparaît dans une observation d'un autre effet e' » et la condition (ii) se lit « aucune observation de e ne contient d'effet e' non équivalent pour s. ».

Exemple 83 (terme séparateur) *Soit l'ensemble d'effets $E = \{x_1x_2, \bar{x}_2\}$ et l'état $s = x_1\bar{x}_2$. Pour l'effet $e = x_1x_2$, le terme $t = x_1$* n'est pas *un séparateur de e pour E dans s. En effet, pour l'état $s' = x_1x_2 \supseteq t$ et l'effet $e' = \bar{x}_2$, la condition (ii) n'est pas respectée :*

$$[e \setminus s', apply(e,s')] \cap [e' \setminus s, apply(e',s)] = [\emptyset, x_1x_2] \cap [\emptyset, x_1\bar{x}_2] = [\emptyset, x_1] \neq \emptyset$$

Par contre, le terme $t' = \bar{x}_2$ est un séparateur de e pour E dans s. Considérons les différents états $s' \supseteq t$:

$$\begin{aligned}[e \setminus s, apply(e,s)] \cap [e' \setminus s', apply(e',s')] =& [x_2, x_1x_2] \cap [\emptyset, x_1\bar{x}_2] = \emptyset && \text{avec } s' = x_1\bar{x}_2 \\ & [x_2, x_1x_2] \cap [\emptyset, \bar{x}_1\bar{x}_2] = \emptyset && \text{avec } s' = \bar{x}_1\bar{x}_2\end{aligned}$$

La condition (i) est bien vérifiée. Maintenant, nous avons

$$\begin{aligned}[e \setminus s', apply(e,s')] \cap [e' \setminus s, apply(e',s)] =& [x_2, x_1x_2] \cap [\emptyset, x_1\bar{x}_2] = \emptyset && \text{avec } s' = x_1\bar{x}_2 \\ & [x_1x_2, x_1x_2] \cap [\emptyset, x_1\bar{x}_2] = \emptyset && \text{avec } s' = \bar{x}_1\bar{x}_2\end{aligned}$$

ce qui vérifie la condition (ii).

Définition 84 (séparabilité) *Soit E un ensemble d'effets et k un entier naturel tel que $1 \leq k \leq n$. Alors E est dit* k-séparable *si pour tout état s, tout effet $e \in E$, il existe un terme t de taille k, qui soit séparateur de e pour E et s.*

Exemple 85 (séparabilité) *Dans le chapitre d'évaluation, section 9.1 page 133, sont présentées les valeurs de k-séparabilité pour les effets de différentes actions. Pour ces problèmes utilisant de 6 à 11 variables, les effets sont toujours 1 ou 2-séparables.*

La proposition suivante décrit l'apprenabilité des probabilités d'un ensemble d'effets k-séparable donné. La notion de k-séparabilité garantit que pour tout effet e il existe un terme séparateur t_e de taille k, tel que les observations de $\mathcal{O}^{t_e}$ qui contiennent e ne contiennent pas d'autre effet. Ainsi la fréquence de ces observations correspond à la fréquence de l'effet e. Cette proposition traite directement le cas d'un ensemble incomplet d'effets, tel qu'il aurait pu être appris par l'algorithme KWIK-Effects.

Proposition 86 *Soit D une distribution de probabilités sur un ensemble E^* d'effets k-séparable et s un état. Soit $E \subseteq E^*$ un ensemble d'effets tel que*

$$d_s(E,E^*) = \sum_{e \in E^* \setminus E} D(e) \leq \epsilon_0$$

Enfin, soit $\mathcal{O}$ une séquence d'observations. Si pour tout terme $t \in T_s^k$, la séquence d'états-effets $\mathcal{T}_t$ qui a induit $\mathcal{O}^t$ vérifie

$$\left\| D^s_{\mathcal{T}_t} - D^s \right\|_1 \leq \epsilon_1$$

alors, la pseudo-distribution[9] *$\hat{D}$ définie sur $[E]_s$ telle que la probabilité $\hat{D}(e)$ d'un effet e correspond à la somme des fréquences des intervalles contenant un effet équivalent à e pour s dans*

9. $\hat{D}$ n'est pas une distribution de probabilités au sens strict du terme car les probabilités ne somment pas à 1. Cependant la démonstration du théorème 87 montre qu'une normalisation de $\hat{D}$ préserve sa précision.

les observations $\mathcal{O}^{t_e}$ où t_e est un [10] *terme séparateur de e pour E^* dans s :*

$$\hat{D}(e) = \sum_{\substack{o \in \mathcal{O}^{t_e} \\ o \cap [e \setminus s, apply(e,s)] \neq \emptyset}} f_{o,\mathcal{O}^{t_e}}$$

vérifie

$$\left\| \hat{D} - D^s \right\|_1 \leq \epsilon_1 \min \left\{ |E^*|, \binom{n}{k} \right\} + \epsilon_0$$

Démonstration. Soit $e \in E^*$ un effet, par définition de la k-séparabilité il existe donc un terme t_e qui soit séparateur de e pour E^* dans s. Notons $\mathcal{O}_e$ pour l'ensemble des intervalles de $\mathcal{O}^{t_e}$ qui contiennent un effet équivalent à e pour s :

$$\mathcal{O}_e = \left\{ o \in \mathcal{O}^{t_e} \middle| o \cap [e \setminus s, apply(e,s)] \neq \emptyset \right\}$$

Par définition d'un séparateur, nous savons que tous les intervalles $o \in \mathcal{O}_e$ ne contiennent aucun effet $e' \in E^*$ qui ne soit pas équivalent à e pour s. La somme des fréquences de ces intervalles, notée $\hat{P}(e \setminus s)$, est donc égale à $D^s_{\mathcal{T}_{t_e}}(e \setminus s)$:

$$\hat{P}(e \setminus s) = \sum_{o \in \mathcal{O}_e} f_{o,\mathcal{O}^{t_e}} = D^s_{\mathcal{T}_{t_e}}(e \setminus s)$$

Nous définissons maintenant la couverture des effets selon leur séparateur :

$$\left\{ E^t = \{e \in E^* | t \text{ est séparateur de } e\} \middle| t \in T^k_s \right\}$$

ceci est bien une couverture de l'ensemble des effets : E^* étant k-séparable, chaque effet de E^* appartient donc à au moins un E^t (notamment quand $t = t_e$). Maintenant sachant que $E^t \subseteq E^*$ on a :

$$\left\| D^s_{\mathcal{T}_t} - D^s \right\|_1 \leq \epsilon_1 \implies \sum_{e \in E^t} |D^s_{\mathcal{T}_t}(e \setminus s) - D^s(e \setminus s)| \leq \epsilon_1$$

et qu'il y a au plus $\binom{n}{k}$ termes séparateurs possibles, on obtient

$$\left\| \hat{P} - D^s \right\|_1 \leq \epsilon_1 \binom{n}{k}$$

Cependant, il y a au plus $|E^*|$ effets, par conséquent on obtient aussi

$$\left\| \hat{P} - D^s \right\|_1 \leq \epsilon_1 |E^*|$$

Par conséquent

$$\left\| \hat{P} - D^s \right\|_1 \leq \epsilon_1 \min \left\{ |E^*|, \binom{n}{k} \right\}$$

10. il peut existe plusieurs termes séparateurs pour un effet donné, ici on en choisit un arbitrairement.

Finalement, en développant $\left\|\hat{D} - D^s\right\|_1$ on obtient

$$\begin{aligned}\left\|\hat{D} - D^s\right\|_1 &= \sum_{e\in[E]_s} \left|\hat{D}(e) - D^s(e)\right| + \sum_{e\in[E^*\setminus E]_s} D^s(e) \\ &\leq \sum_{e\in[E^*]_s} \left|\hat{P}(e) - D^s(e)\right| + \epsilon_0 \\ &= \left\|\hat{P} - D^s\right\|_1 + \epsilon_0 \\ &\leq \epsilon_1 \min\left\{|E^*|, \binom{n}{k}\right\} + \epsilon_0\end{aligned}$$

□

Avec les outils décrits précédemment, nous pouvons en déduire l'algorithme KWIK-Probas (algorithme 15) pour l'apprentissage des probabilités d'un ensemble d'effets k-séparable E. Pour des raisons de simplicité, nous donnons à cet algorithme la capacité de pouvoir observer pour chaque pas de temps t l'ensemble d'effets $[E]_{s_t}$. Toutefois, si E est connu à l'avance, l'algorithme peut toujours fonctionner en calculant $[E]_{s_t}$.

Théorème 87 *Soit D une distribution de probabilités sur un ensemble d'effets E^* k-séparable. Alors pour chaque pas de temps t, et si un ensemble d'effets $[E]_{s_t}$ est observable et tel que $d_{s_t}(E, E^*) \leq \epsilon_0$, alors l'algorithme* KWIK-Probas *produit avec confiance au moins $1-\delta$ des prédictions $\hat{D}_t$ de D^{s_t} qui sont $(\epsilon + 2\epsilon_0)$-précises avec la borne KWIK*

$$B_P(\epsilon,\delta) = O\left(\frac{2^k n^k \min\left\{|E|^2, n^{2k}\right\}}{\epsilon^2} \ln \frac{n^k 2^{|E|+k}}{\delta}\right)$$

Démonstration. En choisissant

$$\epsilon_1 = \frac{\epsilon}{2\min\left\{|E|,\binom{n}{k}\right\}}$$

la proposition 86 implique alors

$$\left\|\hat{D} - D^s\right\|_1 \leq \frac{\epsilon}{2} + \epsilon_0$$

L'application de l'inégalité de [Weissman *et al.*, 2003] et de l'inégalité de Boole sur les $2^k\binom{n}{k} = O(2^k n^k)$ termes de taille k possibles garantissent que cette précision est atteinte avec confiance $1-\delta$ après que chaque $\mathcal{O}^t$ contienne au moins

$$m = \frac{4\min\left\{|E|^2, \binom{n}{k}^2\right\}}{\epsilon^2} \ln \frac{2^k\binom{n}{k}(2^{|E|}-2)}{\delta} = O\left(\frac{\min\left\{|E|^2, n^{2k}\right\}}{\epsilon^2} \ln \frac{n^k 2^{|E|+k}}{\delta}\right)$$

intervalles observés. Demander m observations pour chaque terme de taille k conduit à la borne du théorème.

Néanmoins, la proposition 86 ne garantit pas que les probabilités de $\hat{D}$ somment à 1. Toutefois, la distribution normalisée $\bar{D} = \frac{\hat{D}}{\left\|\hat{D}\right\|_1}$ renvoyée par l'algorithme est $(\epsilon + 2\epsilon_0)$-optimale :

$$\begin{aligned}
\left\|\hat{D} - D^s\right\|_1 \leq \frac{\epsilon}{2} + \epsilon_0 &\iff \sum_e \left|\hat{D}(e) - D^s(e)\right| \leq \frac{\epsilon}{2} + \epsilon_0 \\
&\implies \left|\sum_e \hat{D}(e) - D^s(e)\right| \leq \frac{\epsilon}{2} + \epsilon_0 \\
&\iff \left|\sum_e \hat{D}(e) - \sum_e D^s(e)\right| \leq \frac{\epsilon}{2} + \epsilon_0 \\
&\iff \left|\left\|\hat{D}\right\|_1 - 1\right| \leq \frac{\epsilon}{2} + \epsilon_0 \\
&\iff \left\|\hat{D} - \bar{D}\right\|_1 \leq \frac{\epsilon}{2} + \epsilon_0 \qquad \{ \text{ lemme 94 en annexe } \}
\end{aligned}$$

par application de l'inégalité triangulaire on obtient

$$\left\|\bar{D} - D^s\right\|_1 \leq \left\|\bar{D} - \hat{D}\right\|_1 + \left\|\hat{D} - D^s\right\|_1 \leq \frac{\epsilon}{2} + \epsilon_0 + \frac{\epsilon}{2} + \epsilon_0 = \epsilon + 2\epsilon_0$$

□

Algorithme 15 : KWIK-PROBAS

Entrées : m, k
$\mathcal{O} \leftarrow \emptyset$
pour chaque *pas de temps* $\tau = 1,2,\ldots$ **faire**
  Observer l'état s_τ et l'ensemble d'effets $[E]_{s_\tau}$
  si $\exists t \in T^k_{s_\tau} : |\mathcal{O}^t| < m$ **alors**
    Prédire $\perp$
    Observer s'_τ
    $\mathcal{O} \leftarrow \mathcal{O} \cup \{[s'_\tau \setminus s_\tau, s'_\tau]\}$
  sinon
    pour chaque $e \in [E]_{s_\tau}$ **faire**
      $I \leftarrow [e, apply(e,s_\tau)]$
⋆   choisir $t \in T^k_{s_\tau}$ tel que pour tout $e' \in [E]_{s_\tau} e \neq e'$, il n'existe pas d'intervalle $o \in \mathcal{O}^t$ tel que $I \cap o \neq \emptyset \wedge [e', apply(e',s_\tau)] \cap o \neq \emptyset$
      $\hat{D}(e) \leftarrow \sum_{\substack{o \in \mathcal{O}^t \\ o \cap I \neq \emptyset}} f_{o,\mathcal{O}^t}$
    Prédire $\hat{D} = \frac{\hat{D}}{\left\|\hat{D}\right\|_1}$

Retrait de la connaissance de k

L'algorithme KWIK-PROBAS requiert en entrée une borne k telle que l'ensemble d'effets à apprendre soit k-séparable. Il est toutefois possible de se passer de cette connaissance a priori.

En effet la ligne marquée du symbole $\star$ dans l'algorithme recherche un terme séparateur pour l'effet e. Si E n'est pas k-séparable pour la valeur de k fournie à l'algorithme, cette recherche peut tout de même aboutir en pratique si le choix des états par l'environnement est favorable.

Pour éliminer la connaissance de k, l'algorithme est exécuté avec $k = 1$. Si la ligne $\star$ échoue, il prédit $\perp$ et incrémente k de 1. L'algorithme recommence alors avec la nouvelle valeur de k en conservant les observations précédentes. Dans le pire des cas, il sera capable de prédire lorsque k sera tel que E est k-séparable.

8.2.2 KWIK-apprentissage d'une distribution d'effets

A présent, nous décrivons l'algorithme KWIK-Distribution (algorithme 16) qui combine les algorithmes KWIK-Effets et KWIK-Probas pour le KWIK-apprentissage d'une distribution d'effets. Cet algorithme requiert la connaissance a priori de la k_{cl}-clôture et de la k_{sep}-séparabilité de l'ensemble d'effets caché.

Algorithme 16 : KWIK-Distribution

Entrées : m_E, m_P, k_{sep}, k_{cl}
début
  Instancier un algorithme KWIK-Effects $\mathbf{A}_E$ avec les paramètres m_E et k_{cl}
  Instancier un algorithme KWIK-Proba $\mathbf{A}_P$ avec les paramètres m_P et k_{sep}
  pour chaque *pas de temps* $t = 1,2,\ldots$ **faire**
    Observer l'état s_t
    Exécuter $\mathbf{A}_E$ pour obtenir sa prédiction $\hat{E}_t$
    si $\hat{E}_t = \perp$ **alors**
      Prédire $\perp$; Observer s'_t
      Présenter s'_t à $\mathbf{A}_E$ et à $\mathbf{A}_P$
    sinon
      Exécuter $\mathbf{A}_P$ avec $\hat{E}_t$ pour obtenir sa prédiction $\hat{D}_t$
      si $\hat{D}_t = \perp$ **alors**
        Prédire $\perp$; Observer s'_t
        Présenter s'_t à $\mathbf{A}_P$
      sinon
        Prédire $\hat{D}_t$
fin

Théorème 88 *L'algorithme* KWIK-Distribution *peut prédire de manière précise une distribution D sur un ensemble d'effets E k_{cl}-clos et k_{sep}-séparable avec une borne KWIK*

$$B_D(\epsilon,\delta) = \max\left\{B_E\left(\frac{\epsilon}{3},\frac{\delta}{2}\right), B_P\left(\frac{\epsilon}{3},\frac{\delta}{2}\right)\right\}$$
$$= O\left(\frac{2^K n^{3K}}{\epsilon^2}\ln\frac{2^{|E|+K}n^K}{\delta}\right)$$

avec $K = \max\{k_{cl}, k_{sep}\}$.

Démonstration. L'algorithme KWIK-DISTRIBUTION est l'exécution en parallèle de KWIK-EFFECTS et de KWIK-PROBA. En exécutant le premier avec $m_E = B_E(\epsilon_0, \delta_0)$, pour chaque pas de temps t, ses prédictions $\hat{E}_t$ vérifient $d_{s_t}(E, \hat{E}_t) \leq \epsilon_0$ avec confiance au moins $1 - \delta_0$. Pour le second algorithme, instancié avec $m_P = B_E(\epsilon_1, \delta_1)$ on obtient d'après le théorème 87 que les prédictions $\hat{D}_t$ vérifient $\left\| D - \hat{D}_t \right\|_1 \leq \epsilon_1 + 2\epsilon_0$ avec confiance au moins $1 - \delta_1$.

Notons que même quand $\hat{E}_t = \perp$, les observations de s'_t sont fournies à $\mathbf{A}_P$ qui peut les utiliser pour apprendre même s'il doit attendre $\hat{E}_t \neq \perp$ pour faire ses prédictions. Ainsi la borne KWIK finale est le maximum des bornes de chaque algorithme, plutôt que leur somme.

Pour conclure en choisissant $\epsilon_0 = \epsilon_1 = \frac{\epsilon}{3}$ on obtient

$$\left\| D - \hat{D}_t \right\|_1 \leq \frac{\epsilon}{3} + 2\frac{\epsilon}{3} = \epsilon$$

En appliquant l'inégalité de Boole pour $\delta_0 = \delta_1 = \frac{\delta}{2}$ on obtient ce résultat avec confiance au moins $1 - \delta$. □

8.3 PSO-Rmax

Nous proposons désormais d'utiliser l'algorithme KWIK-DISTRIBUTION dans un algorithme capable d'apprendre par renforcement un PDM décrit avec des opérateurs STRIPS probabilistes. Pour cela, un tel algorithme doit être capable de découvrir, en plus des effets et de leurs probabilités, les conditions des actions. Nous faisons l'hypothèse que l'apprenant a connaissance d'une borne supérieure k sur le nombre de variables apparaissant dans les conditions d'actions, ainsi que des bornes k_{sep} et k_{cl} dérivées respectivement de la séparabilité et de la clôture définies précédemment. Nous verrons alors que le complexité de l'exploration de cet algorithme est polynomiale lorsque ces bornes sont des constantes fixées.

Cette section est dédiée à la description d'un algorithme KWIK capable de prédire la fonction de transition d'une action, pour chaque état. Contrairement aux algorithmes précédents, il y a plusieurs distributions différentes qui ont généré les exemples, chacune correspondant à une condition cachée de l'action. Un algorithme d'apprentissage par renforcement PAC-MDP est finalement obtenu en utilisant cet algorithme dans KWIK-RMAX (algorithme 11).

8.3.1 KWIK-apprentissage de distributions conditionnelles

Pour apprendre les conditions d'actions, nous supposons que l'apprenant connaît une borne k sur le nombre de variables apparaissant dans ces conditions. Il ne peut donc y avoir plus de 2^k conditions dans une action, chacune étant un terme de taille k. Les k variables n'étant pas connues il y a $\binom{n}{k} = O(n^k)$ ensembles de k variables pouvant constituer les conditions. Le principe de l'algorithme est le suivant. Il considère tous les ensembles de k variables comme hypothèses, pour chacune de ces hypothèses il recueille suffisamment d'observations provenant

d'états satisfaisant les 2^k termes pouvant être formés sur ces variables, chacun étant une condition. Pour toutes ces hypothèses candidates, un test de sélection de la meilleure hypothèse inspiré de SLF-RMAX [Strehl *et al.*, 2007] est effectué afin d'obtenir une bonne estimation de la fonction de transition.

Algorithme 17 : KWIK-PSO

Entrées : m, k, k_{sep}, k_{cl}

début

  pour chaque *terme c de* $2k$ *littéraux* **faire**
    Instancier un algorithme KWIK-DISTRIBUTION $\mathbf{A}_c$ avec les paramètres m et k_{sep}

  pour chaque *pas de temps* $t = 1,2,\ldots$ **faire**
    Observer l'état s_t
    `/* Ensemble des conditions candidates pour` s_t `*/`
    pour chaque $c \in T_{s_t}^{2k}$ **faire**
7   Exécuter $\mathbf{A}_c$ pour obtenir sa prédiction $\hat{D}_c$
    si $\exists c \in C \ : \ \hat{D}_c = \perp$ **alors**
      Prédire $\perp$ et observer s'_t
      Présenter s'_t à tous les $\mathbf{A}_c$ tels que $\hat{D}_c = \perp$
    sinon
      `/* Choix de la condition la plus plausible */`
      pour chaque $c \in T_{s_t}^{k}$ **faire**
13       $\rho_c = \max_{x,y \in T^k_{s_t \setminus c}} \left\| \hat{D}_{c \cup x} - \hat{D}_{c \cup y} \right\|_1$
14       $\hat{P}_c \leftarrow \hat{D}_{c \cup x}$ (avec $x \subseteq s_t \setminus c$, $|x| = k$ choisi arbitrairement)
      $\hat{c} = \arg\min_{c \in C} \rho_c$
      Prédire $\hat{P}_{\hat{c}}$

fin

Remarque 89 (Utilisation de l'algorithme Noisy-Union) *Notons que la sélection d'une hypothèse parmi un ensemble de candidats est traitée dans [Li* et al.*, 2011] avec l'algorithme* NOISY-UNION *qui permet de n'instancier des algorithmes* KWIK-DISTRIBUTION *que pour des conditions de taille k (au lieu de* $2k$*). Cependant, leur algorithme suppose que les sous-algorithmes prédisant les différentes hypothèses retournent la probabilité d'un événement binaire (la probabilité de changer une variable) et non pas* $|E|$*-aire (une distribution d'effets). L'idée de leur algorithme est d'éliminer les mauvaises hypothèses en mesurant leur erreur de prédiction les unes par rapport aux autres. Le calcul de cette mesure d'erreur nécessite d'observer directement l'effet produit, ce qui, pour des raisons d'ambiguïté des observations, est difficilement réalisable dans notre cadre.*

Proposition 90 *Soit a une action décrite en STRIPS probabiliste. Pour tout état* s_t*, on note* c_t *la condition de a correspondant à* s_t *et* D_t *la distribution d'effets associée à* c_t*. Avec le*

paramètre $m = B_D\left(\epsilon_0, \frac{\delta_0}{\binom{n-k}{k}2^{2k}}\right)$, toutes *les distributions* $\hat{D}_c$ *calculées par l'algorithme KWIK-PSO (ligne 7) vérifient avec confiance au moins* $1-\delta_0$ *:*

$$\left\|\hat{D}_c - D_t\right\|_1 \leq \rho_c + \epsilon_0$$

Démonstration. Soit c une condition candidate de taille k. Pour toutes les conditions $c \cup x$ de taille $2k$, l'algorithme KWIK-DISTRIBUTION calcule une prédiction $\hat{D}_{c\cup x}$. Or, il existe au moins un terme x^* de taille k tel que $c_t \subseteq c \cup x^*$. En appliquant l'inégalité de Boole sur les $\binom{n-k}{k}2^{2k}$ termes de taille $2k$ mentionnant un terme condition correct [11], toutes les prédictions $\hat{D}_{c\cup x^*}$ vérifient avec probabilité au moins $1-\delta_0$

$$\left\|\hat{D}_{c\cup x^*} - D_t\right\|_1 \leq \epsilon_0$$

De plus (ligne 13) nous avons

$$\rho_c = \max_{x,y\in T^k_{s_t\setminus c}} \left\|\hat{D}_{c\cup x} - \hat{D}_{c\cup y}\right\|_1$$

Et donc pour tout $x \in T^k_{s_t\setminus c}$ nous avons

$$\left\|\hat{D}_{c\cup x} - D_t\right\|_1 \leq \left\|\hat{D}_{c\cup x} - \hat{D}_{c\cup x^*}\right\|_1 + \left\|\hat{D}_{c\cup x^*} - D_t\right\|_1 \leq \rho_c + \epsilon_0$$

□

De cette proposition nous déduisons que la prédiction $\hat{D}_c$ qui minimise ρ_c est alors une estimation précise de la distribution cible D_t. Nous montrons maintenant qu'avec grande confiance, la condition cible c_t produit toujours de bonnes prédictions.

Proposition 91 *Soit a une action décrite en opérateurs STRIPS probabilistes. Pour tout état* s_t*, on note* c_t *la condition de a correspondant* s_t *et* D_t *la distribution d'effets associée à* c_t*. Avec le paramètre* $m = B_D\left(\epsilon_0, \frac{\delta_0}{\binom{n-k}{k}2^{2k}}\right)$*, la distribution* $\hat{P}_{c_t}$ *calculée par l'algorithme KWIK-PSO (ligne 14) vérifie avec confiance au moins* $1-\delta_0$ *:*

$$\left\|\hat{D}_{c_t} - D_t\right\|_1 \leq 3\epsilon_0$$

Démonstration. Pour la condition c_t et en appliquant l'inégalité de Boole de manière similaire à la proposition précédente, *toutes* les prédictions $\hat{D}_{c_t\cup x}$ vérifient avec probabilité au moins $1-\delta_0$

$$\left\|\hat{D}_{c_t\cup x} - D_t\right\|_1 \leq \epsilon_0$$

Par conséquent on en déduit :

$$\rho_{c_t} = \max_{x,y\in T^k_{s_t\setminus c_t}} \left\|\hat{D}_{c_t\cup x} - \hat{D}_{c_t\cup y}\right\|_1 \leq \max_{x,y\in T^k_{s_t\setminus c_t}} \left\|\hat{D}_{c_t\cup x} - D_t\right\|_1 + \left\|\hat{D}_{c_t\cup y} - D_t\right\|_1 \leq 2\epsilon_0$$

En utilisant cette inégalité avec la proposition 90 on obtient le résultat souhaité. □

Nous pouvons donc désormais conclure sur la borne KWIK de l'algorithme KWIK-PSO

11. Il y a 2^k conditions correctes de taille k, à chacune desquelles on peut rajouter $\binom{n-k}{k}2^k$ termes pour obtenir une condition de taille $2k$ qui contient une vraie condition.

Théorème 92 *Soit a une action décrite en opérateurs STRIPS probabilistes telle que*

- *les ensembles d'effets de a soient tous au plus k_{cl}-clos et k_{sep}-séparables, et ne contiennent qu'au plus E effets,*
- *l'ensemble de conditions des a ne mentionne au plus que k variables.*

Alors l'algorithme KWIK-PSO *avec* $m = B_D\left(\frac{\epsilon}{3}, \frac{\delta}{\binom{n}{2k}2^{2k}}\right)$, *prédit la fonction de transition de a avec pour borne KWIK*

$$B(\epsilon,\delta) = O\left(\frac{n^{2k+3K}2^{2k+K}}{\epsilon^2}\ln\frac{2^{E+2k+K}n^{k+K}}{\delta}\right)$$

avec $K = \max\{k_{cl}, k_{sep}\}$.

Démonstration. KWIK-PSO utilise $\binom{n}{2k}2^{2k}$ instances de l'algorithme KWIK-Distribution. Afin que *toutes* les prédictions $\hat{D}_c$, $c_t \subseteq c$ renvoyées par ces algorithmes vérifient avec confiance au moins $1-\delta$:

$$\left\|\hat{D}_c - D_t\right\|_1 \leq \epsilon$$

on pose $\epsilon_0 = \frac{\epsilon}{3}$ puis on applique l'inégalité de Boole pour obtenir une précision de ϵ_0 sur les $\binom{n-k}{k}2^{2k} = O(n^k 2^{2k})$ prédictions mentionnant une condition correcte pour finalement obtenir

$$\begin{aligned}
B(\epsilon,\delta) &= \binom{n}{2k}2^{2k}B_D\left(\frac{\epsilon}{3}, \frac{\delta}{\binom{n-k}{k}2^{2k}}\right) \\
&= O\left(2^{2k}n^{2k}\frac{2^{\max\{k_{sep},k_{cl}\}}n^{3\max\{k_{sep},k_{cl}\}}}{\epsilon^2}\ln\frac{2^{|E|+2k+\max\{k_{sep},k_{cl}\}}n^{k+\max\{k_{sep},k_{cl}\}}}{\delta}\right) \\
&= O\left(\frac{n^{2k+3K}+2^{2k+K}}{\epsilon^2}\ln\frac{2^{E+2k+K}n^{k+K}}{\delta}\right)
\end{aligned}$$

□

8.3.2 Apprentissage PAC-MDP d'opérateurs STRIPS probabilistes

En intégrant l'algorithme KWIK-PSO dans KWIK-Rmax nous obtenons un algorithme d'apprentissage par renforcement dans un PDM décrit en STRIPS probabiliste.

Définition 93 (PSO-Rmax) *L'algorithme d'apprentissage par renforcement* PSO-Rmax *est défini comme l'intégration dans* KWIK-Rmax *(algorithme 11) d'une instance l'algorithme* KWIK-PSO *(algorithme 17) pour chaque action du PDM. La fonction de récompense est supposée connue à l'avance ainsi qu'une borne k sur le nombre de variables apparaissant dans les conditions de la représentation STRIPS probabiliste de l'action et des bornes k_{cl} et k_{sep} respectivement sur la clôture et la séparabilité des distributions d'effets. L'algorithme prend en paramètre le seuil d'exploitation m qui est transmis à chaque instance de* KWIK-PSO.

Ainsi, avec $K = \max\{k_{cl},k_{sep}\}$ et $m = B_D\left(\frac{\epsilon}{3}, \frac{\delta}{\binom{n-k}{k}2^{2k}}\right)$, le théorème 42 nous assure que pour $\epsilon > 0$, $\delta \in]0,1]$ PSO-Rmax exécutera au plus

$$O\left(\frac{n^{2k+3K}2^{2k+K}}{\epsilon^3(1-\gamma)^6}\ln\frac{2^{E+2k+K}n^{k+K}}{\delta}\ln\frac{1}{\delta}\ln\frac{1}{\epsilon(1-\gamma)}\right)$$

actions a_t telles que $Q^*(s_t, a_t) < V^*(s_t) - \epsilon$ et ce avec confiance au moins $1 - \delta$.

8.3.3 Discussion sur les bornes de complexité d'exploration

Notons toutefois que cette borne théorique est une surestimation très grossière de la complexité d'exploration, dont le but principal est de montrer qu'elle est polynomiale lorsque k, k_{sep} et k_{cl} sont des constantes fixées. En effet, plusieurs approximations sont introduites :

- L'inégalité de [Weissman *et al.*, 2003] introduit déjà une approximation qui ne peut être réduite qu'avec certaines connaissances sur la distribution cible.
- L'inégalité de Boole utilisée très souvent ajoute encore de l'imprécision.
- L'approximation $\binom{n}{k} = O(n^k)$ est elle aussi très lâche, notamment quand k est proche de n (nous faisons cependant la supposition que k est très petit devant n).
- Enfin, pour simplifier les preuves, nous supposons qu'un intervalle observé ne contribue qu'à une seule famille d'observations $\mathcal{O}^t$ alors qu'il peut contribuer à jusqu'à $\binom{n}{|t|}$ telles familles.

Comme nous le verrons dans le chapitre suivant, en pratique de bons résultats sont obtenus en fixant le seuil d'exploration à des valeurs de l'ordre de quelques dizaines sans tenir compte de ces bornes. Nous remarquons finalement que la connaissance du nombre d'effets E n'intervient que dans ces bornes théoriques et n'est pas requise dans les algorithmes.

Chapitre 9

Évaluation de l'apprentissage par renforcement

Sommaire

Nous présentons désormais les résultats empiriques sur le comportement d'un apprenant par renforcement avec les algorithmes PSO-Rmax et PSO-LP. Dans toutes les évaluations présentées, nous considérons que la fonction de récompense est connue à l'avance. Pour cette évaluation, nous utilisons trois domaines de la littérature, pour chacun desquels une table résume les différents paramètres k, k_{sep} et k_{cl} de chaque action. Nous donnons aussi la plus grande récompense possible pour un état afin de pouvoir interpréter les erreurs moyennes de chaque algorithme.

9.1 Présentation des domaines d'évaluation

Coffee-64 Il s'agit du domaine présenté figure 13 page 26, il comporte 6 variables soit 64 états et 4 actions. La plus grande récompense d'un état est de 1.

Action	k	k_{sep}	k_{cl}
Move	3	1	1
Get-Umbrella	1	1	1
Buy-Coffee	1	1	1
Deliver-Coffee	1	2	2

Coffee-2048 Ce problème est une variante étendue du précédent, il comporte 11 variables soit 2048 états et 8 actions. Il a été introduit dans [Dearden et Boutilier, 1997]. La plus grande récompense d'un état est de 2,1.

Action	k	k_{sep}	k_{cl}
Move-Left	4	1	2
Move-Right	4	1	2
Deliver	4	1	2
Buy-Coffee	3	1	2
Buy-Bun	3	1	2
Get-Mail	3	1	2
Deliver-Mail	3	1	1
Get-Umbrella	2	1	1

Builder-512 Ce problème met en scène un robot qui doit préparer (polir, peindre, nettoyer, percer, ...) deux objets puis les attacher. Il comporte 9 variables soit 512 états et 10 actions. Il a été introduit dans [Dearden et Boutilier, 1997]. La plus grande récompense d'un état est de 1.

Action	k	k_{sep}	k_{cl}
Paint-A	1	1	2
Paint-B	1	1	2
Shape-A	1	2	2
Shape-B	1	2	2
Wash-A	0	1	1
Wash-B	0	1	1
Drill-A	1	1	1
Drill-B	1	1	1
Bolt	4	1	1
Glue	2	2	2

9.2 Méthodologie

Nous avons mené une comparaison entre PSO-Rmax, PSO-LP et l'algorithme non factorisé Rmax. Chaque évaluation est faite sur un ensemble d'épisodes de 200 pas de temps. Au début

de chaque épisode l'agent est replacé dans un état choisi aléatoirement. Cette méthode a été préférée à un seul long épisode car les problèmes considérés possèdent tous un état de récompense maximale pour lesquels il existe toujours une action qui permet d'y rester. Effectuer plusieurs épisodes depuis des états aléatoires permet d'explorer une plus grande partie de l'espace d'états et donc de vérifier le comportement de l'agent dans une grande variété de situations.

9.2.1 Métriques de performance

Erreur accumulée

Pour chacun de ces algorithmes nous mesurons leur performance en terme d'erreur accumulée par rapport à la politique optimale. Les graphiques de performance ont en abscisse le numéro d'épisode x et en ordonnée la fonction d'erreur accumulée

$$Err(x) = \sum_{t=1}^{x\times 200} V^*(s_t) - Q^*(s_t, a_t)$$

Une fois dessinée sur un graphique, cette métrique de performance a pour intérêt de décrire une courbe qui converge vers une droite une fois que l'algorithme d'apprentissage a cessé d'explorer. Étant donné qu'il est impossible de garantir une performance optimale, nous considérons que si cette courbe d'erreur croît légèrement à partir d'un certain point c'est que le comportement de l'agent est quasi-optimal (il n'accumule que peu d'erreur). À l'inverse, si la courbe continue de croître à un rythme important sur un grand nombre d'épisodes, c'est que l'agent n'a pas réussi à apprendre un bon comportement. La figure 9.1 schématise l'allure des courbes de performance d'un agent qui converge vers un comportement optimal 9.1(a), vers un comportement quasi-optimal 9.1(b) ou d'un échec d'apprentissage 9.1(c).

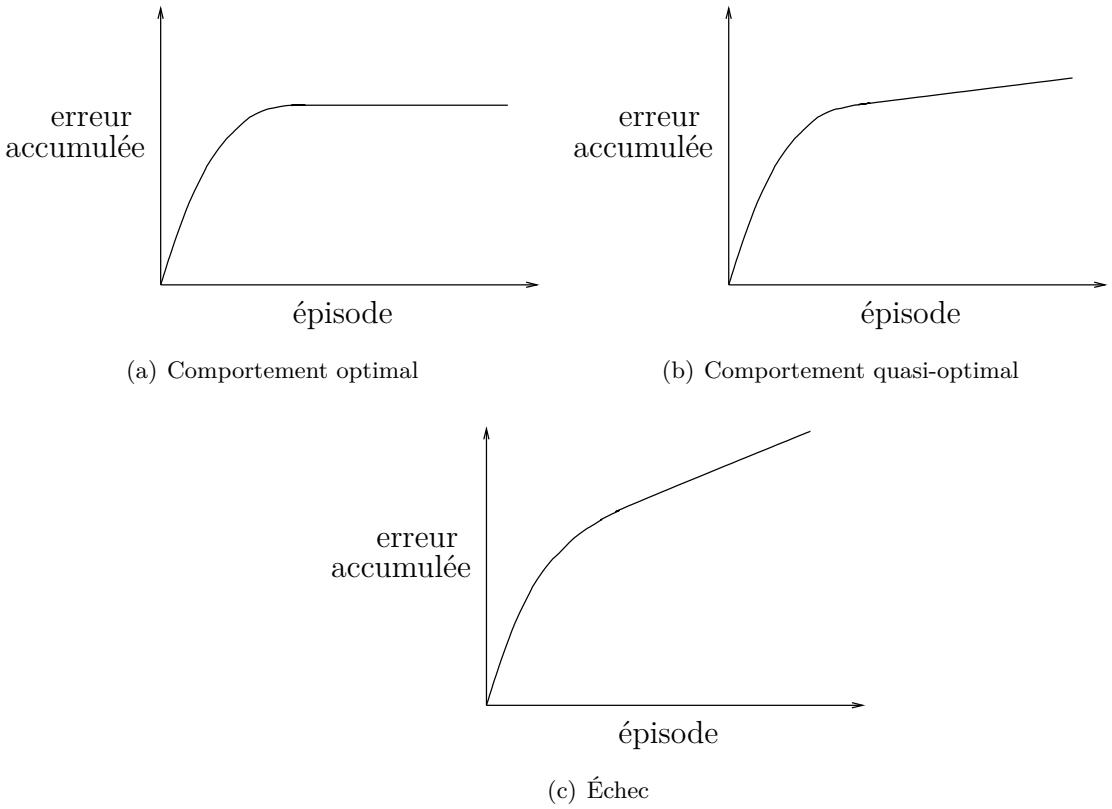
Erreur moyenne par pas de temps

À partir des données d'erreur accumulée en moyenne, nous construisons une seconde mesure de performance correspondant à l'erreur moyenne $\bar{\epsilon}$ commise par l'apprenant une fois la convergence atteinte. En effet, on observe qu'à partir d'un certain pas de temps t_0 les courbes d'erreur accumulée se stabilisent pour obtenir une droite. Cette droite signifie que l'erreur de chaque épisode est sensiblement constante. Ainsi la pente de ces droites correspond à l'erreur moyenne *d'un épisode*. Pour obtenir la valeur moyenne par pas de temps on divise cette valeur par 200 (la durée d'un épisode). Au final quand une évaluation se fait sur $t_{\max}$ épisodes on calcule :

$$\bar{\epsilon} = \frac{Err(t_{\max}) - Err(t_0)}{200(t_{\max} - t_0)}$$

Ainsi, pour tout pas de temps $t \geq t_0$ on a

$$Q^*(s_t,a_t) \approx V^*(s_t) - \bar{\epsilon}$$



(a) Comportement optimal

(b) Comportement quasi-optimal

(c) Échec

FIGURE 9.1 – Interprétation des allures des courbes d'erreur accumulée.

9.2.2 Paramétrage des algorithmes

Nos deux algorithmes prennent en entrée un paramètre m qui indique le nombre d'intervalles que doivent contenir les projections d'observations $\mathcal{O}^t$ pour chaque terme t. Ce paramètre influence directement la précision des distributions estimées pour les états satisfaisant ces termes t. De manière similaire, RMAX demande lui aussi un paramètre m correspondant aux nombres d'observations qu'il doit recevoir pour chaque état. Nous avons pour chacun d'entre eux testé différentes valeurs entre 20 et 50. Comme il est d'usage dans la littérature [Strehl *et al.*, 2006a, Strehl *et al.*, 2007, Diuk *et al.*, 2009] nous n'avons pas utilisé l'analyse théorique pour choisir cette valeur car de petites valeurs de m telles que nous les avons choisies suffisent à donner de bons résultats en pratique. Les valeurs données par l'analyse sont bien plus élevées (de plusieurs ordres de grandeur) et rendent les temps d'exécution bien trop longs alors que les performances en termes de mesure d'erreur n'accroissent que très peu en comparaison. Les paramètres suivants ont été utilisés pour l'étape de planification : $\gamma = 0{,}95$ et $\epsilon = 0{,}01$ pour tous les algorithmes et problèmes.

9.2.3 Évaluation des performances

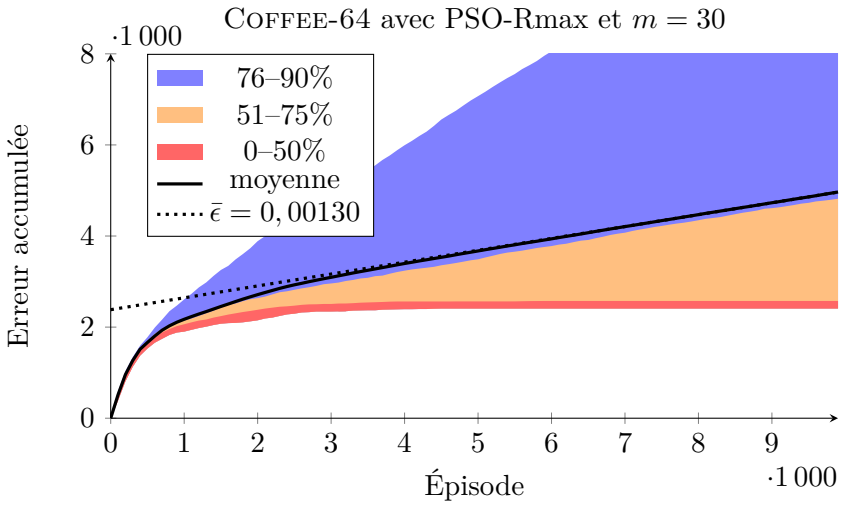
Pour chaque domaine, plusieurs graphiques de performance sont affichés. Pour chacun des algorithmes et valeurs de m, un graphique présente l'erreur accumulée de 100 exécutions de l'algorithme. Pour faciliter la lisibilité, les courbes ont été triées par erreurs croissantes puis regroupées en trois catégories, chacune représentée par une surface de couleur. Les surfaces en rouge recouvrent les 50% de courbes d'erreur ayant la plus petite erreur (de valeur inférieure à la médiane), ensuite les surfaces orange recouvrent 25% de courbes suivantes, puis les surfaces en bleu recouvrent les 15% de courbes suivantes. La moyenne des 100 exécutions est représentée par une courbe noire. La droite pointillée représente la courbe de pente $\bar{\epsilon}$. Ensuite, pour chaque domaine, un graphique récapitulatif regroupe la moyenne d'erreur des différents algorithmes pour chacune des valeurs de m utilisées.

9.3 Résultats

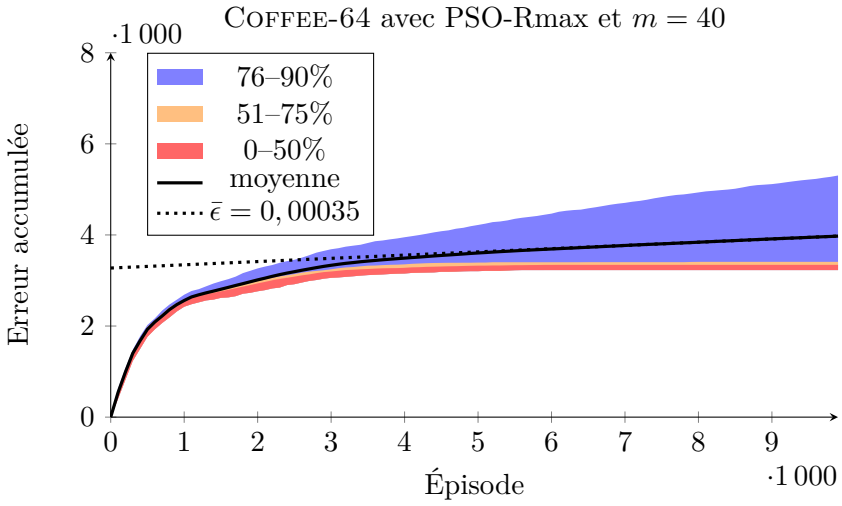
9.3.1 Coffee-64

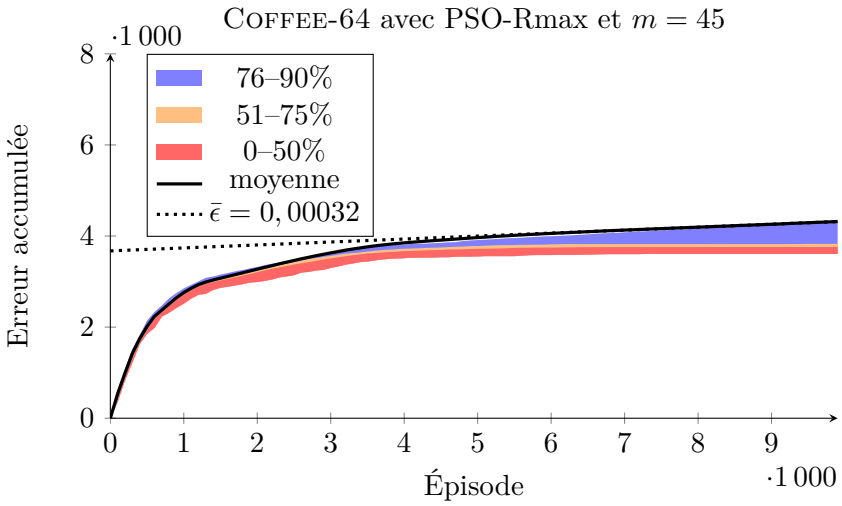
PSO-Rmax

Tout d'abord nous présentons les graphiques montrant l'impact de la valeur de m pour PSO-RMAX. Pour $m = 30$ on observe que le comportement en moyenne est quasi-optimal, comme espéré. Cependant, 25% des exécutions sont au-dessus de la moyenne et parfois très éloignées de l'optimal. En effet, un si faible nombre d'exemples rend la probabilité d'obtenir de bonnes estimations de la fonction de transition très faible.



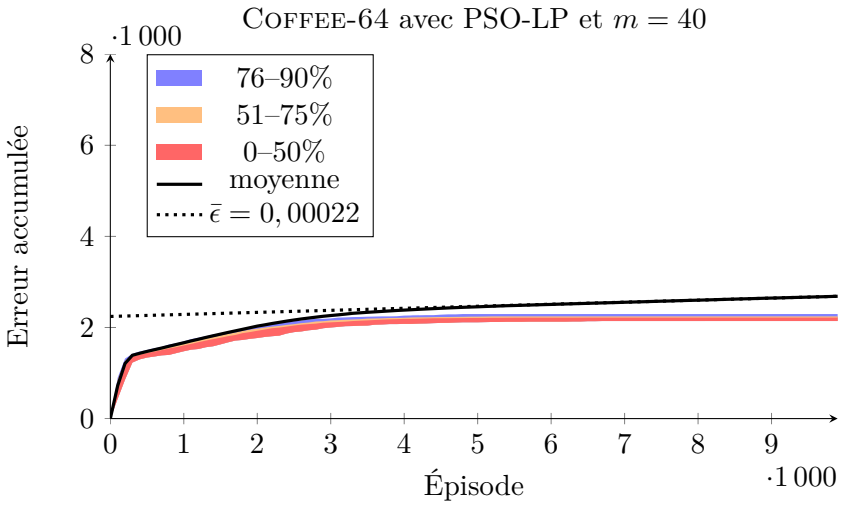
Par la suite pour les paramètres $m = 40$ et $m = 45$ l'erreur des différentes exécutions est plus concentrée. On remarque dans les deux cas que 75% des exécutions ont un comportement très proche de l'optimal et le comportement moyen est très satisfaisant. Pour le cas $m = 45$, 90% des exécutions se comportent mieux que la moyenne, ce qui suggère que l'apprentissage peut échouer avec une grande erreur, mais cependant avec une faible probabilité.





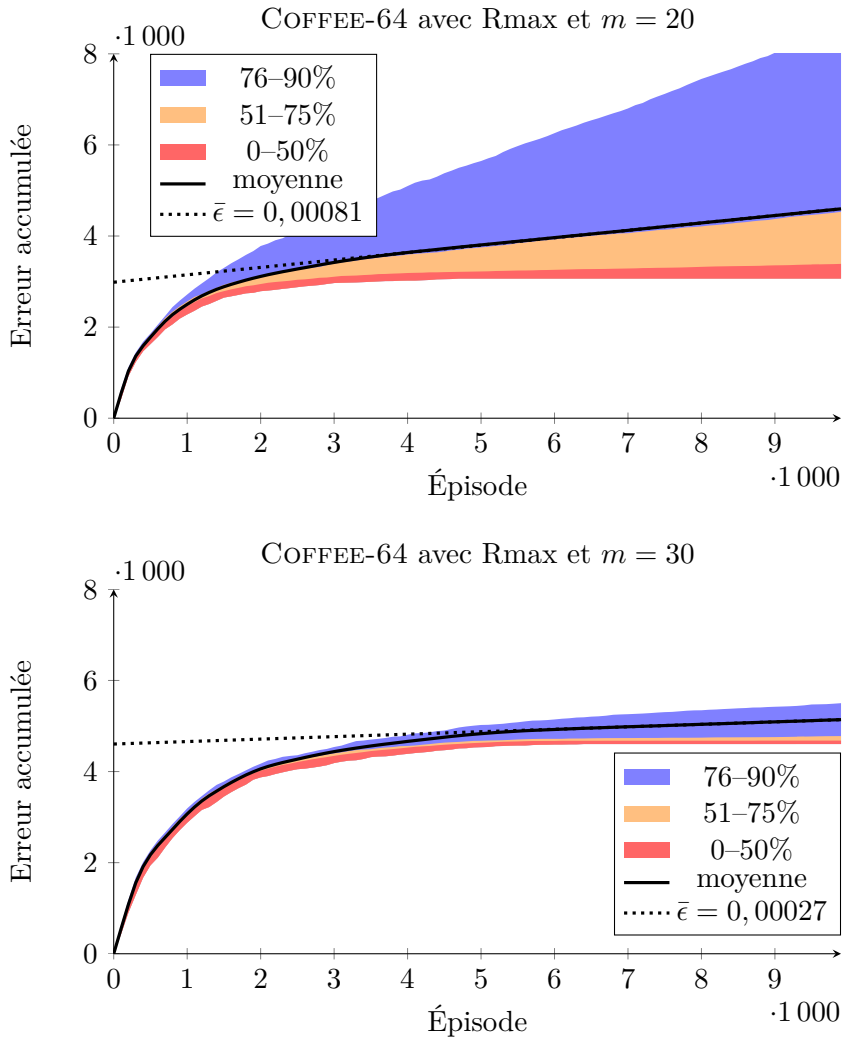
PSO-LP

Le cas de l'algorithme PSO-LP montre un très bon comportement avec $m = 40$. On observe que 90% des exécutions convergent vers un comportement optimal après environ 4000 épisodes. Ce temps de convergence est similaire à celui de PSO-Rmax. Cependant, l'erreur accumulée est sensiblement moins élevée (2000 contre 3000 environ). Notons toutefois que la moyenne d'erreur est nettement plus élevée que celle de 90% des exécutions, ce qui montre qu'un échec d'apprentissage de PSO-LP peut mener à un comportement radicalement mauvais (la pire exécution a accumulé une erreur de plus de 17000 au 10000^{e} pas de temps).



Rmax

Pour Rmax avec $m = 30$, la convergence est atteinte aux environs de l'épisode 5500 avec une erreur d'environ 4000. On observe que 75% des exécutions convergent vers un comportement optimal et que l'erreur reste groupée autour de la valeur moyenne.



Comparaison

La figure 9.2 compare les moyennes d'erreur accumulées des différents algorithmes et leurs paramétrages de m qui donnent un comportement similaire après convergence. Bien que ne donnant aucune garantie théorique sur son comportement, PSO-LP est l'algorithme qui converge le plus vite et ce en accumulant significativement moins d'erreur que les autres algorithmes. Comme espéré, ce sont bien les approches exploitant la structure STRIPS Probabiliste du domaine qui se comportent le mieux le plus rapidement. Le relativement faible écart entre Rmax et PSO-Rmax peut s'expliquer par le petit nombre d'états du problème qui ne permet pas d'exploiter toute la structure du problème.

9.3.2 Coffee-2048

Pour le domaine Coffee-2048, nous ne présentons que les résultats pour PSO-Rmax et Rmax. L'imposante quantité de programmes linéaires à résoudre par PSO-LP rend impraticable son évaluation. En effet, une exécution sur 35000 pas de temps nécessite plus d'une semaine

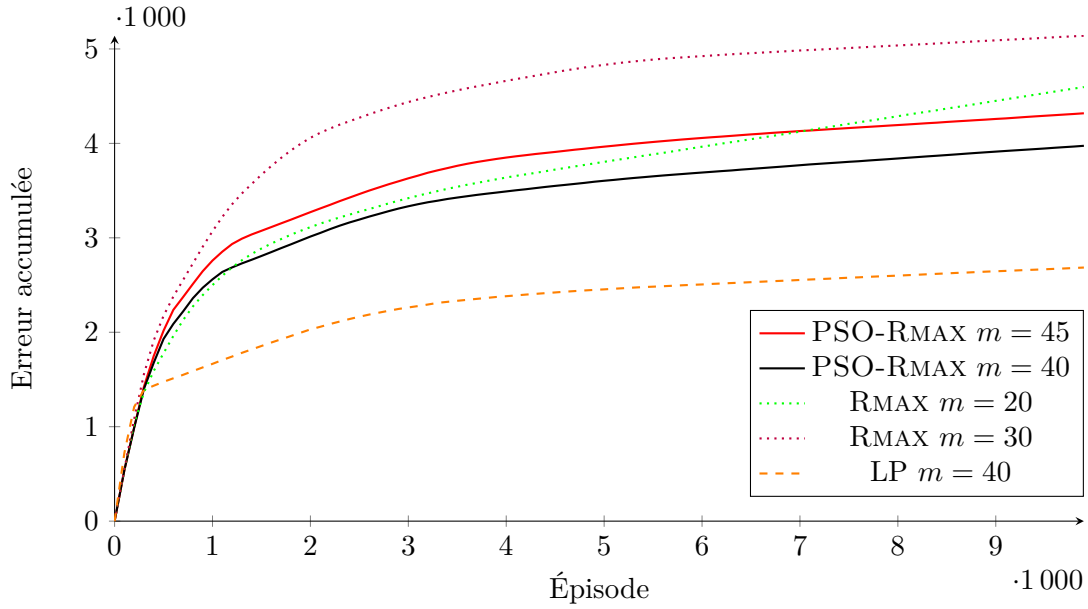
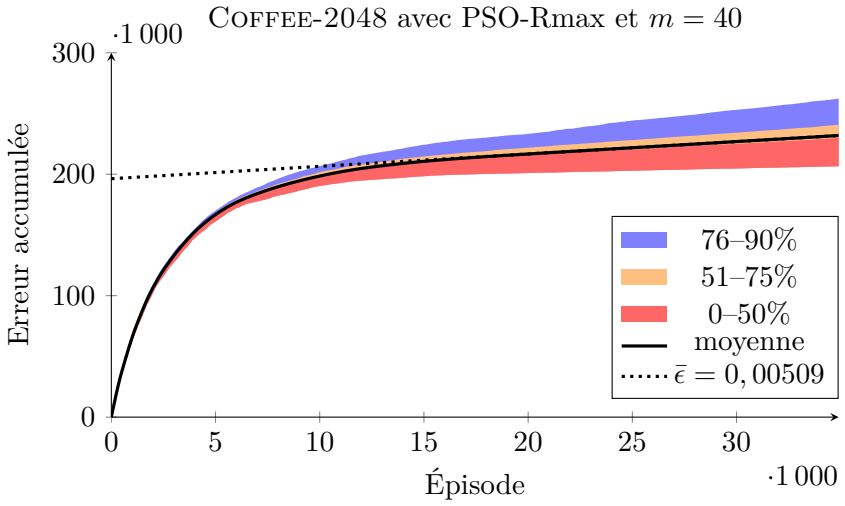


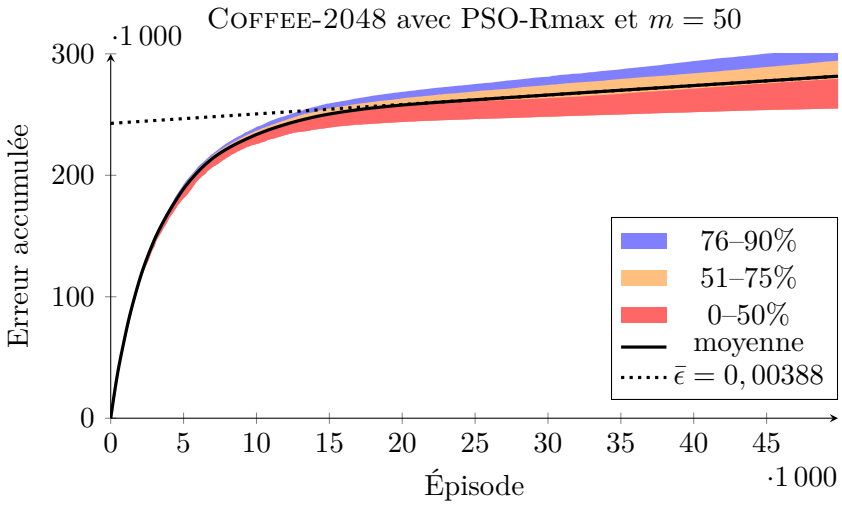
Figure 9.2 – Erreur moyenne accumulée sur le domaine Coffee-64.

de calcul. Obtenir un nombre suffisant d'exécutions pour obtenir des résultats en moyenne significatifs est de fait impossible avec les moyens à notre disposition. A titre de comparaison, une exécution de PSO-Rmax ou Rmax sur un même horizon se fait en quelques heures.

PSO-Rmax

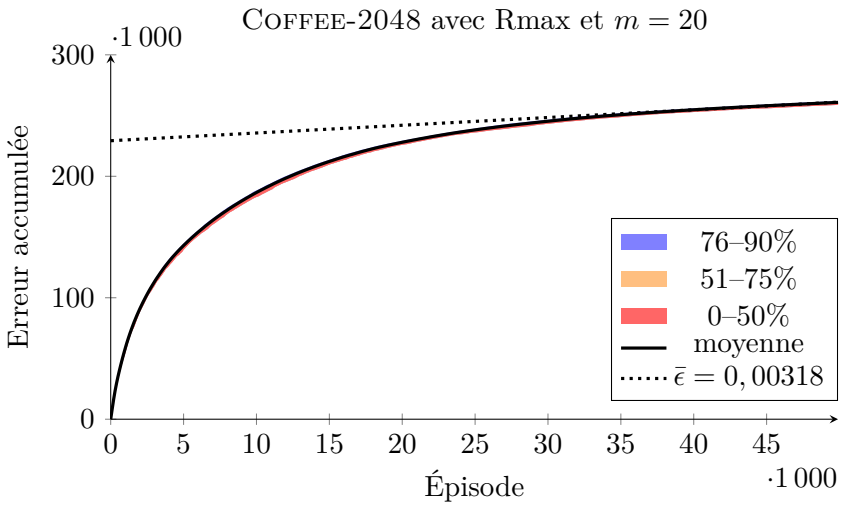
Pour PSO-Rmax, les résultats expérimentaux sont très concluants. Pour les deux valeurs $m = 40$ et $m = 50$ testées, on observe que les résultats sont très groupés autour de la moyenne qui converge à $\bar{\epsilon} = 0{,}005$ après environ 15000 épisodes pour $m = 40$ et $\bar{\epsilon} = 0{,}0038$ après 18000 épisodes pour $m = 50$. Dans le pire des cas, il n'y a pas d'erreur radicalement importante ce qui montre que la confiance observée est nettement plus élevée que la confiance théorique pour une telle valeur de m.





Rmax

Avec Rmax le comportement des 100 exécutions est complètement identique. Toutes les exécutions convergent à $\bar{\epsilon} = 0{,}00318$ après environ 32000 épisodes pour $m = 20$.



Comparaison

Entre les deux algorithmes, l'erreur accumulée à la convergence est similaire, voire légèrement meilleure pour Rmax en comparaison à PSO-Rmax avec $m = 50$. Cependant on observe que PSO-Rmax est capable d'exploiter la structure du problème et de généraliser son expérience pour atteindre une convergence quasi-optimale ($\bar{\epsilon} = 0{,}0038$) après seulement 18000 épisodes alors que Rmax obtient une qualité similaire ($\bar{\epsilon} = 0{,}00318$) après 32000 épisodes.

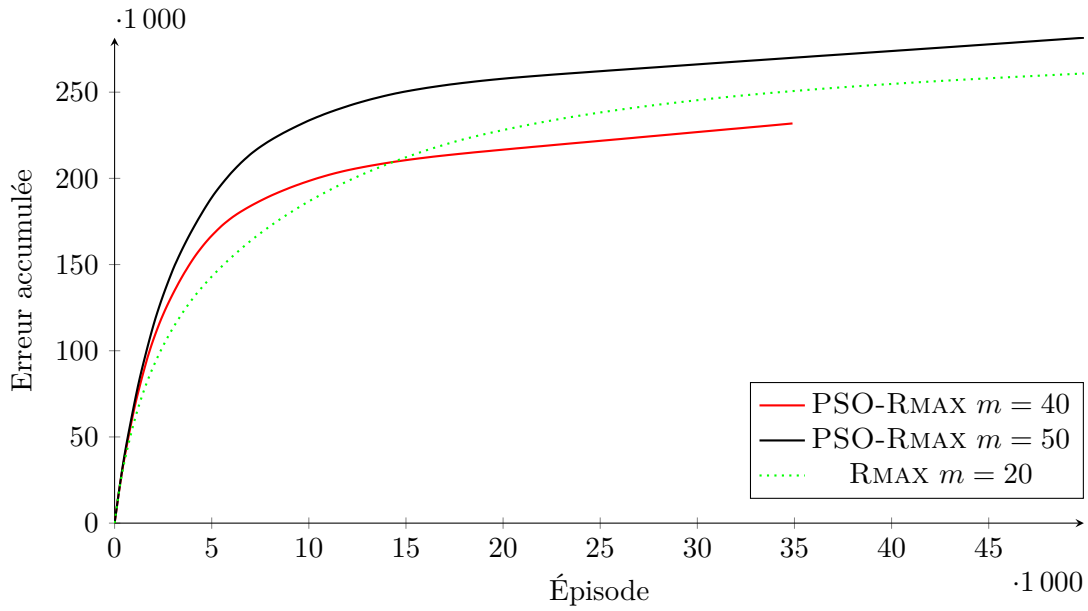
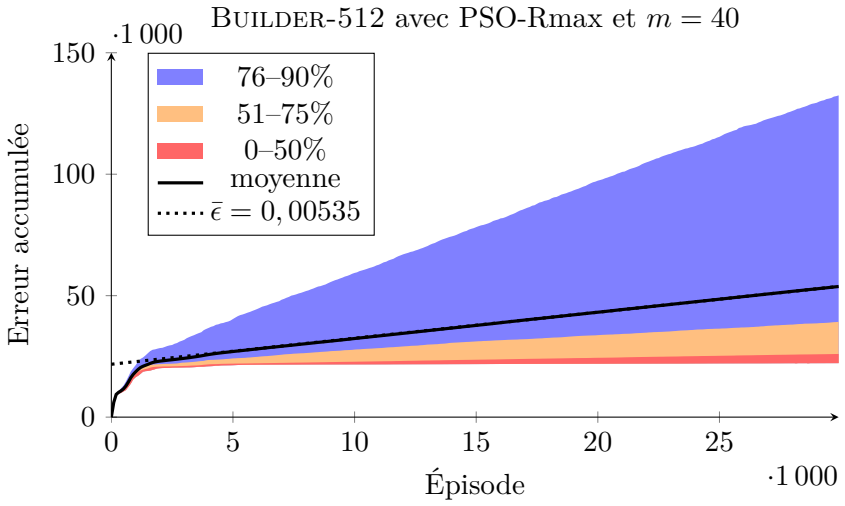


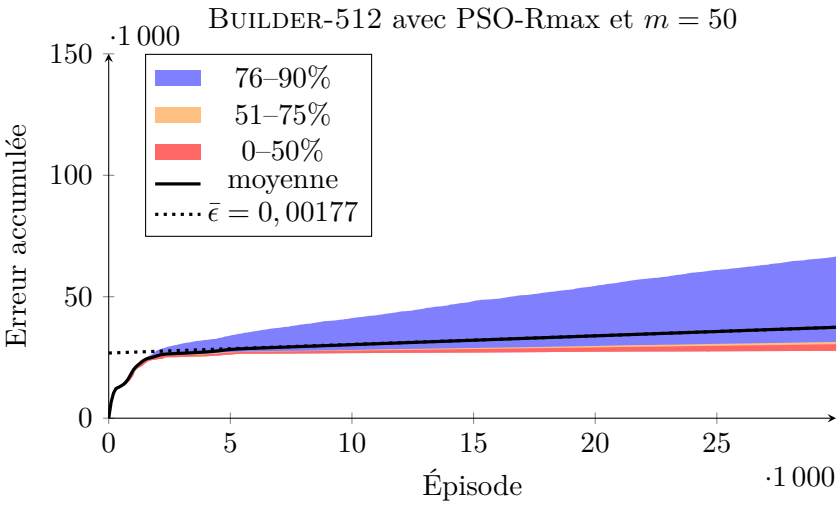
Figure 9.3 – Erreur moyenne accumulée sur le domaine Coffee-2048.

9.3.3 Builder-512

PSO-Rmax

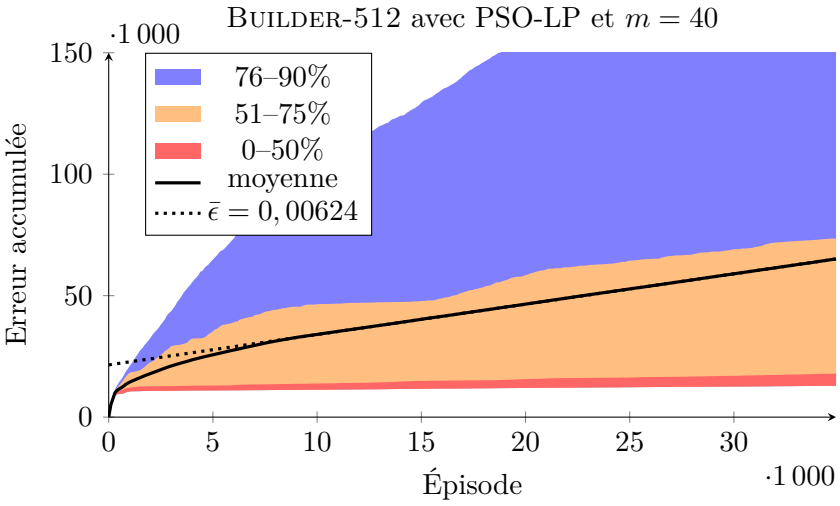
Pour l'algorithme PSO-Rmax sur le domaine Builder-512 on observe de bonnes performances en moyenne avec $\bar{\epsilon} = 0{,}00535$ pour $m = 40$ et $\bar{\epsilon} = 0{,}00177$ pour $m = 50$. Pour les deux valeurs de m utilisées, plus de 75% des exécutions ont une erreur inférieure à la moyenne. Cependant, quelques exécutions très mauvaises ont eu lieu et pénalisent fortement la moyenne. Le fait d'augmenter m améliore légèrement la qualité d'une grande majorité des cas, sans pour autant améliorer les pires cas. PSO-Rmax se comporte néanmoins de façon très satisfaisante sur ce domaine en convergeant vers un très bon comportement aux alentours de l'épisode 4000 pur $m = 40$ et 5000 pour $m = 50$.

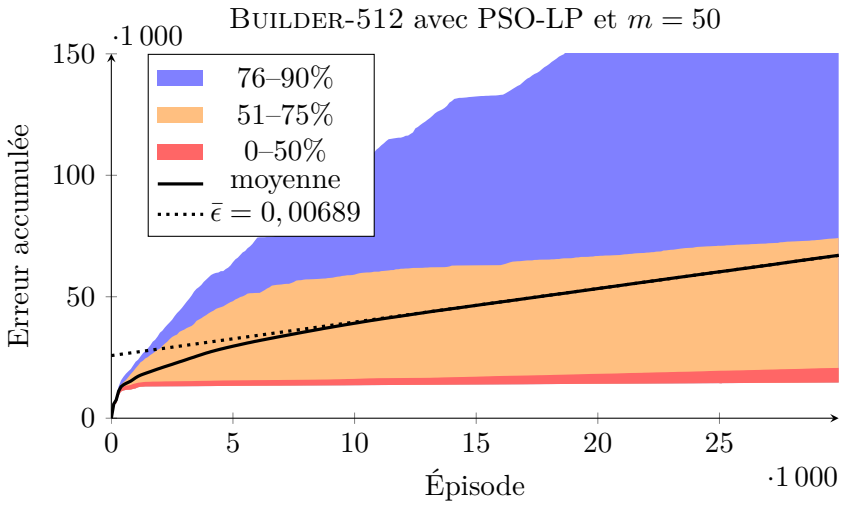




PSO-LP

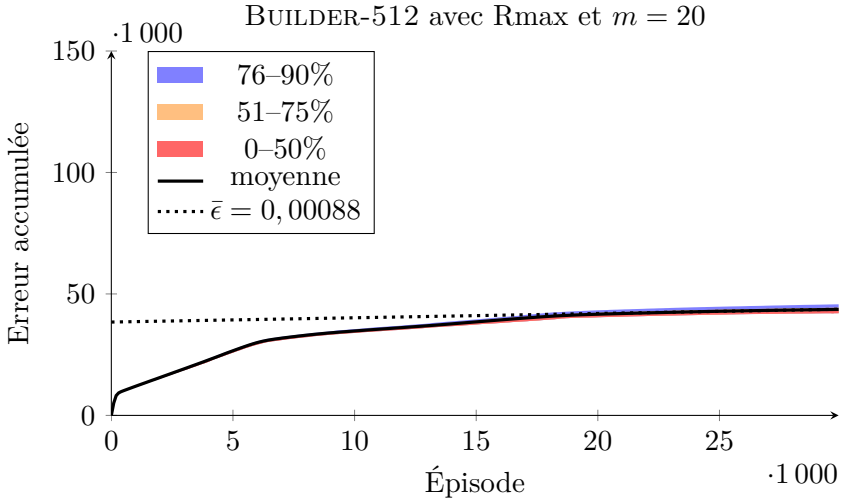
Les résultats de PSO-LP sur ce domaine sont nettement moins bons que ceux de PSO-Rmax quand bien même il était nettement plus performant sur Coffee-64. Pour $m = 40$ la convergence à $\bar{\epsilon} = 0{,}00624$ est atteinte après environ 8000 épisodes et environ 10000 pour $m = 50$. Pour les deux valeurs de m testées il y a tout de même plus de la moitié des exécutions qui ont un comportement quasi-optimal. Cependant on observe une grande dispersion entre les différentes exécutions. Même en passant m de 40 à 50 le phénomène ne semble pas s'atténuer, ce qui montre les limites de l'approche heuristique sur ce domaine.





Rmax

Pour Rmax les résultats sont très peu dispersés avec la quasi-totalité des exécutions qui s'éloignent très peu de la moyenne qui converge à $\bar{\epsilon} = 0{,}00088$ après environ 18000 épisodes pour $m = 20$.



Comparaison

Pour le domaine Builder-512 on observe sur la figure 9.4 que cette fois PSO-LP n'est pas le meilleur algorithme. Ceci est très certainement dû à sa nature heuristique ainsi qu'à une certaine difficulté du domaine. En effet, pour les deux approches factorisées, les comportements les plus mauvais, bien que peu fréquents, ont une très grande erreur accumulée ce qui pénalise sensiblement leur moyenne. Cependant, ces deux algorithmes arrivent tout de même à bien exploiter la structure du domaine pour converger 2 à 4 fois plus vite que Rmax.

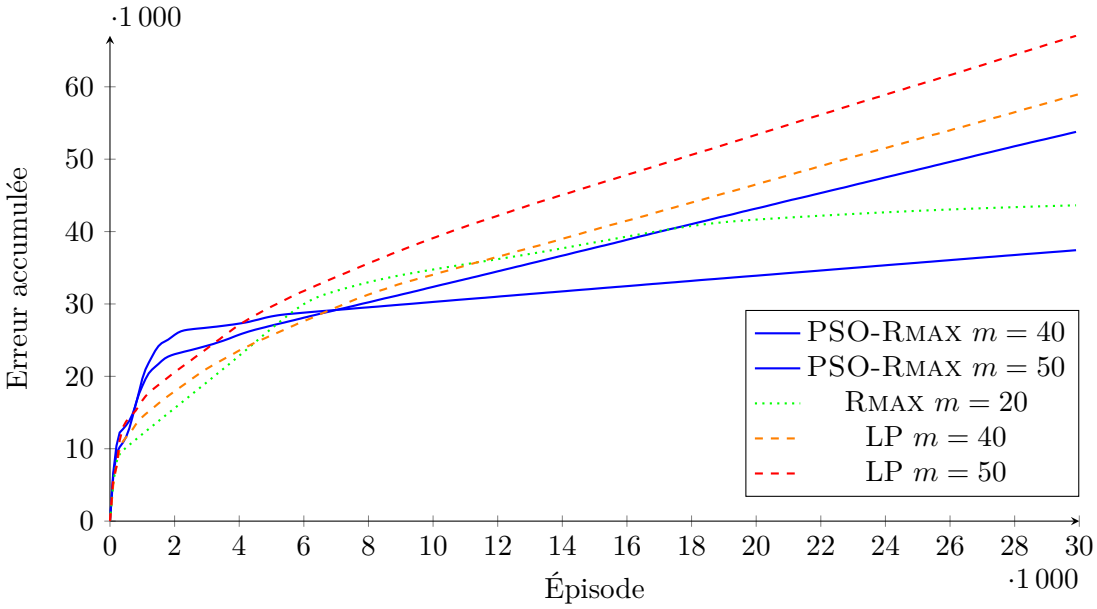


FIGURE 9.4 – Erreur moyenne accumulée sur le domaine BUILDER-512.

9.4 Discussion

Dans tous les cas testés, PSO-RMAX converge vers un bon comportement plus rapidement que RMAX. Cependant, le facteur d'accélération observé est de l'ordre de 1,5, ce résultat s'explique par le fait que les valeurs de k, k_{sep} et k_{cl} sont assez élevées pour certaines actions. De fait, utiliser des projections d'observations sur les termes de taille $K = 2k + \max\{k_{sep}, k_{cl}\}$ demande beaucoup d'exploration car K est finalement assez proche du nombre de variables n ce qui ne permet pas d'exploiter au maximum la structure des problèmes. Pour mettre plus en avant la manière dont PSO-RMAX exploite la compacité d'une représentation STRIPS probabiliste, nous avons modifié le problème COFFEE-64 en lui ajoutant trois variables ainsi que des actions pour changer leur polarité ce qui porte le domaine à 512 états. Cette modification ne modifie pas les paramètres k, k_{sep} et k_{cl} mais les rend plus petits vis-à-vis de n. Ainsi sur la figure 9.5 on observe que le temps de convergence pour PSO-Rmax a très peu augmenté sur ce domaine en comparaison au problème original. Par contre, le temps de convergence de RMAX est fortement pénalisé.

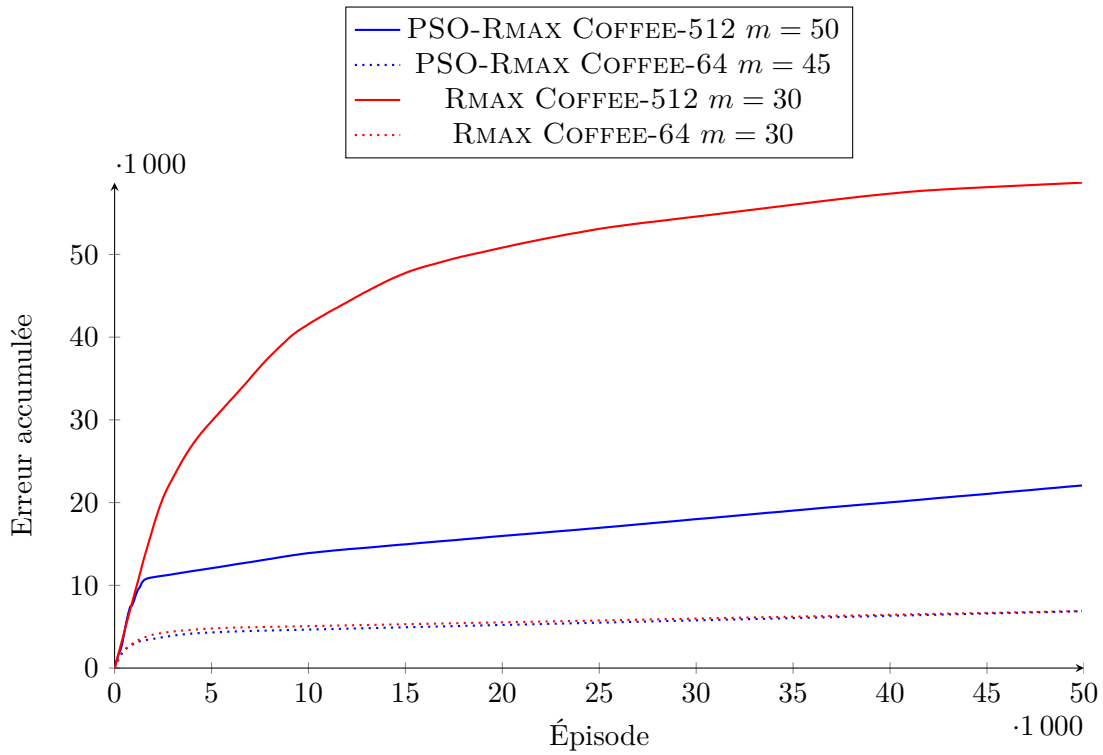


Figure 9.5 – Erreur moyenne accumulée sur les domaines Coffee-64 et Coffee-512.

Conclusion de la partie

Dans ce chapitre nous avons abordé l'apprentissage par renforcement dans les PDM lorsque le modèle d'action sous-jacent peut être décrit de façon compacte en opérateurs STRIPS probabilisteq. Les approches présentées apprennent un modèle de la dynamique de l'environnement, puis l'utilisent pour planifier les actions à prendre à chaque pas de temps. Ces approches basées sur la compacité du modèle d'action ont l'intérêt d'apporter de bonnes réponses à la gestion du compromis exploration/exploitation qui est au centre de la problématique de l'apprentissage par renforcement. Ce compromis est géré grâce à l'approche de RMAX qui attire l'agent vers les parties de l'espace d'états inconnues avec une forte récompense. De plus, si l'agent est capable d'apprendre une bonne approximation du modèle de transitions, alors ce type d'exploration permet de donner des garanties théoriques sur le comportement de l'agent à l'exécution.

La difficulté majeure d'apprendre un modèle STRIPS probabiliste est l'ambiguïté des effets qui survient lorsque les observations données à l'apprenant ne sont que des couples d'états correspondant à avant et après l'exécution d'une action. Pour faire face à ce problème, nous avons introduit une représentation compacte des effets plausibles pour une transition observée (les intervalles) ainsi que les éléments fondamentaux (les observations et leurs projections) pour nos contributions.

Notre première approche basée, sur une notion de justesse et de variance des observations recueillies par l'agent, a permis de définir des heuristiques pour l'identification des effets et de leurs probabilités. Ces heuristiques calculées par programmation linéaire donnent de bons résultats en pratique malgré l'absence de garanties théoriques.

Notre seconde approche se place dans le cadre KWIK. Afin de concevoir des algorithmes KWIK pour l'identification des effets et de leur probabilités, nous avons défini deux propriétés structurelles des ensembles d'effets : la clôture et la séparabilité. Ces deux notions permettent de caractériser la dimension de la classe d'hypothèses à apprendre. De façon générale, partant de la k-séparabilité ou de la k-clôture des ensembles d'effets, nous en déduisons que la dimension de l'espace d'hypothèses à apprendre pour les distributions d'effets est n^k. Cette dépendance exponentielle en k implique alors que les algorithmes que nous proposons ne sont KWIK que si k *est une constante connue*. Cependant, c'est le cas dans de nombreuses approches d'apprentissage de modèles structurés que ce soit en apprentissage par renforcement avec SLF-RMAX [Strehl *et al.*, 2007] et MET-RMAX [Diuk *et al.*, 2009] ou en apprentissage de concepts avec par exemple

les k-CNF dans le cadre PAC [Valiant, 1984] ou encore pour les modèles graphiques [Abbeel *et al.*, 2006].

L'algorithme PSO-RMAX que nous proposons exploite la structure du problème pour converger vers un bon comportement plus rapidement que les approches non factorisées. Notons toutefois que cette approche se veut complémentaire à des algorithmes du type MET-RMAX qui eux exploitent la structure en Réseaux Bayésiens Dynamiques du problème. Le choix de ce type d'algorithme ou de PSO-RMAX reste cependant délicat, puisqu'il est difficile de savoir à l'avance quelle représentation sera la plus compacte. Une heuristique raisonnable pour le choix d'un algorithme pourrait être la suivante. Si l'on suppose l'indépendance des variables, alors il est légitime d'utiliser SLF-RMAX ou MET-RMAX. À l'inverse, si on pense que les effets d'actions sont corrélés il est plus judicieux d'utiliser PSO-RMAX.

Les évolutions possibles de ces travaux seraient d'améliorer les bornes KWIK de l'identification des effets et du test de conditions. En effet, la borne en n^{3k} sur le seuil d'exploitation m pour l'identification des effets est prohibitive, alors qu'en pratique de petites (de l'ordre de quelques dizaines) valeurs suffisent. L'amélioration du test de sélection des conditions (avec sa borne en n^{2k}) est une piste à plus court terme, car elle pourrait s'inspirer de l'algorithme `noisy-union` de [Li *et al.*, 2011] qui permettrait une borne en n^k. Bien entendu, l'identification de propriétés moins restrictives que la clôture et la séparabilité pourrait aussi mener à des approches différentes avec de meilleures bornes KWIK.

L'algorithme KWIK-EFFECTS proposé requiert la connaissance de la valeur k tel que l'ensemble d'effets à apprendre est k-clos. Bien que cette connaissance a priori puisse sembler restrictive, il n'existe pas à ce jour d'algorithme capable d'apprendre les effets d'action. En effet, de nombreux travaux similaires tels que ceux de [Pasula *et al.*, 2004, Pasula *et al.*, 2007] pour l'apprentissage de règles d'actions ou ceux mentionnés dans [Walsh *et al.*, 2009] se basent sur de heuristiques qui ne fournissent aucune garantie sur les effets découverts.

Le principal obstacle à toutes les approches d'apprentissage par renforcement indirect est qu'elles ont toutes recours à un planificateur non factorisé comme l'itération de valeur pour choisir l'action à exécuter à chaque pas de temps. Ceci est dû au fait que le modèle d'action compact n'est pas appris tel quel. Ce qui est appris permet de prédire la fonction de transition pour des états individuels. L'hypothèse de l'existence d'une représentation compacte a pour seule conséquence d'accélérer l'apprentissage du modèle de transitions.

L'utilisation d'autres méthodes de planification est à considérer, comme la planification approchée de type RTDP [Bonet et Geffner, 2003], ou le « Sparse Sampling » [Kearns *et al.*, 1999] dont la complexité est indépendante de la taille de l'espace d'états mais exponentielle en l'horizon. Un compromis serait d'utiliser le fait qu'il faut choisir une action pour l'état courant seulement, plutôt que calculer une politique complète. Bien que l'apport *théorique* en terme de complexité soit nul, ce type de solution a un intérêt évident en pratique.

Conclusion générale et perspectives

Les travaux décrits dans cette thèse s'articulent autour des PDM avec représentations compactes d'actions. Plus précisément nous avons utilisé les langages de descriptions d'actions STRIPS probabiliste et PPDDL. Nos travaux visaient à proposer des algorithmes exploitant la compacité de ces représentations pour d'une part le calcul de politiques et d'autre part l'apprentissage par renforcement.

1 Bilan sur la planification

1.1 Planification approchée dans les biais logiques

En ce qui concerne la planification, nous avons tout d'abord étudié les opérateurs STRIPS probabilistes et conçu un algorithme se basant sur la logique propositionnelle pour manipuler les descriptions des actions et de la fonction de récompense. Cet algorithme n'a pas été conçu dans le but de proposer des méthodes efficaces *en pratique* pour la résolution de PDM. Il est avant tout un premier pas pour l'intégration de techniques issues de la compilation de connaissances dans la planification probabiliste. Nous avons donc souhaité étudier l'impact de l'approximation de la structure d'un problème sur les politiques résultantes. Les résultats présentés ont montré que l'approximation des conditions d'actions dans la classe des CNF de Horn (et 2-CNF dans une moindre mesure) permettaient de réduire les temps de calcul d'une politique de 30 à 50% pour un problème donné, le tout pour une perte de qualité moyenne de 20% environ.

Perspective : intégration de bases de connaissances Du fait de la représentation interne des ensembles d'états sous la forme de formule propositionnelles, nous pourrions utiliser cet algorithme comme point de départ pour exploiter l'ajout de connaissances logiques dans le processus de décision. En effet, en supposant que nous ayons à notre disposition un modèle STRIPS très général pour un problème ainsi que sa politique (optimale ou non), dans quelle mesure peut-on utiliser une base de connaissances logique qui décrit une variante de ce problème pour obtenir une politique prenant en compte ces connaissances ?

1.2 Planification exacte avec PPDDL

Nous avons décrit RBAB, un algorithme de planification exacte dans les PDM décrits en PPDDL. Le pivot de cet algorithme est la notion de valeur d'action « frameless » qui permet de construire des règles récursives sur la description des effets d'action pour calculer les fonctions de valeur. Les règles de RBAB exploitent directement les représentations compactes de fonctions sous forme d'arbres de décision algébriques pour rendre les calculs les plus efficaces possibles. Les comparaisons effectuées avec l'algorithme SPUDD (qui utilise aussi les ADD) ont montré que RBAB pouvait exploiter la compacité du PPDDL pour être le plus performant.

Perspective : passage au premier ordre Au delà de l'utilisation de variantes « classiques » des ADD telles que les ADD affines, il serait intéressant de modifier RBAB pour utiliser des structures du premier ordre. Faire évoluer RBAB au premier ordre permettrait de se passer de l'étape de propositionalisation des actions PPDDL et ainsi obtenir des politiques indépendantes du nombre d'objets présents dans le domaine de planification. Une telle évolution n'est pas triviale. Cependant ce type de techniques fait l'objet d'une attention grandissante de la part de la communauté scientifique et un certain nombre d'outils tels que les ADD du premier ordre pourraient permettre cette évolution de RBAB.

Perspective : observabilité partielle Une seconde perspective pour l'évolution de RBAB serait d'étendre PPDDL avec une notion d'observabilité partielle. Un tel langage permettrait de décrire des PDM partiellement observables de manière compacte. Il s'agirait donc ensuite d'étendre RBAB afin d'exploiter cette structure pour la résolution efficace de ces problèmes.

2 Bilan sur l'apprentissage

2.1 Apprentissage heuristique

L'algorithme d'apprentissage par renforcement PSO-LP et plus précisément l'algorithme LP-Conditions pour l'apprentissage de la fonction de transition d'une action ont donné de très bons résultats empiriques sur le domaine Coffee-64. L'importante quantité de programmes linéaires à résoudre ($\binom{n}{k}$ conditions à tester pour chaque état) est cependant un frein à son passage à l'échelle, comme par exemple sur le problème Coffee-2048 où le temps d'exécution se compte en semaines.

Perspective : passage à l'échelle L'amélioration du temps d'exécution de l'algorithme LP-Conditions peut s'aborder sur deux fronts. Tout d'abord, d'un point de vue de l'implémentation, par l'utilisation d'un solveur de programmes linéaires plus performant comme CPLEX au lieu du solveur libre GLPK, simple d'utilisation mais moyennement performant. D'un côté algorithmique cette fois il serait intéressant de disposer d'une heuristique pour l'élimination des conditions candidates qui ont donné de très grandes variances afin que celles-ci ne soient plus

testées pour d'autres états. Il faut toutefois rester prudent avec cette technique car une grande variance peut être simplement due à un échantillonnage défavorable des effets et non pas à une mauvaise condition candidate.

2.2 Apprentissage PAC-MDP

Notre contribution la plus importante dans le domaine de l'apprentissage est sans aucun doute la famille d'algorithmes KWIK pour l'apprentissage par renforcement avec des garanties PAC-MDP. En particulier l'algorithme KWIK-Effects est à notre connaissance le premier algorithme capable d'apprendre les effets d'une action avec des garanties KWIK, en comparaisons aux approches heuristiques de la littérature [Pasula *et al.*, 2007, Walsh *et al.*, 2009]. La notion de séparabilité, introduite pour l'apprentissage des probabilités d'un ensemble d'effets connu avec KWIK-Probas, est quant à elle une approche différente de celle présentée dans [Walsh *et al.*, 2009] qui utilisent un algorithme basé sur la régression linéaire pour résoudre ce problème.

Perspective : découvrir à l'exécution la k-séparabilité et la k-clôture Les algorithmes KWIK-Effects et KWIK-Probas requièrent a priori la connaissance d'une borne k sur la séparabilité et la clôture des ensembles d'effets. Il serait bénéfique de pouvoir se passer de cette connaissance préalable et de découvrir ces valeurs lors de l'exécution des algorithmes. Nous avons déjà évoqué comment se passer de la connaissance de la k-séparabilité. À l'inverse, pour la k-clôture, le problème reste ouvert.

Perspective : apprendre les conditions avec Noisy-Union Pour l'apprentissage des conditions, l'algorithme KWIK-PSO requiert la connaissance d'une borne k sur le nombre de variables apparaissant dans celles-ci. Bien qu'il semble improbable de pour se passer de cette information. Notre approche dérivée de SLF-Rmax [Strehl *et al.*, 2007] introduit une complexité en exemples en $\tilde{O}(n^{2k})$. Comme nous l'avons remarqué, il serait intéressant d'adapter l'algorithme Noisy-Union [Li *et al.*, 2011], utilisé par exemple dans MET-Rmax [Diuk *et al.*, 2009] qui permettrait une complexité en $\tilde{O}(n^k)$.

Annexes

Annexe A

Outils Théoriques

Dans cet annexe nous présentons une série de lemmes utiles, certains proviennent de la littérature tandis que d'autres ont été introduits pour cette thèse [12].

Lemme 94 *Soient D une pseudo-distribution de probabilités sur un ensemble fini d'événements. Alors la distance L_1 entre D et $\frac{D}{\|D\|_1}$ (D renormalisée) vérifie*

$$\left\| D - \frac{D}{\|D\|_1} \right\|_1 = |1 - \|D\|_1|$$

Démonstration.

$$\begin{aligned}
\left\| D - \frac{D}{\|D\|_1} \right\|_1 &= \left\| D \left(1 - \frac{1}{\|D\|_1} \right) \right\|_1 \\
&= \left\| D \left| 1 - \frac{1}{\|D\|_1} \right| \right\|_1 \\
&= \left\| D \left| \frac{\|D\|_1 - 1}{\|D\|_1} \right| \right\|_1 \\
&= \|D\|_1 \left| \frac{\|D\|_1 - 1}{\|D\|_1} \right| \\
&= |1 - \|D\|_1|
\end{aligned}$$

□

Lemme 95 (inégalité de Boole) *Pour une famille dénombrable d'événements $A_1, A_2, \ldots$ la probabilité* qu'au moins *un ces événements se produise est inférieure ou égale à la somme de probabilité des événements individuels*

$$\Pr\left(\bigcup_i A_i \right) \leq \sum_i \Pr(A_i)$$

12. Il est cependant très probable (certain ?) que ces résultats soient connus mais ils sont présentés ici faute de référence appropriée.

De manière équivalente, la probabilité que tous ces événements se produisent vérifie alors

$$\Pr\left(\bigcap_i A_i\right) = 1 - \Pr\left(\bigcup_i \bar{A}_i\right) \geq 1 - \sum_i \Pr(\bar{A}_i)$$

Nous utilisons cette inégalité plutôt sous sa seconde forme quand nous avons n événements équiprobables de probabilité $1-\delta$. Ainsi

$$\begin{aligned}\Pr\left(\bigcap_i^n A_i\right) &\geq 1 - \sum_i^n \delta \\ &= 1 - n\delta\end{aligned}$$

Lemme 96 *Soient e et e' deux effets et un terme t tel que $e \sim_t e'$. Alors pour tous les états $s \supseteq \bar{t} \cup (e \setminus t) \cup (e' \setminus t)$ nous avons*

$$e' \in [e \setminus s, \mathit{apply}(e,s)]$$

Démonstration. Soit ℓ un ensemble de littéraux tel que $s = \bar{t} \cup (e \setminus t) \cup (e' \setminus t) \cup \ell$. On développe $e \setminus s$:

$$\begin{aligned}
e \setminus s &= e \setminus (\bar{t} \cup (e \setminus t) \cup (e' \setminus t) \cup \ell) \\
&= (e \setminus (e \setminus t)) \setminus (\bar{t} \cup (e' \setminus t) \cup \ell) && \{\ A \setminus (B \cup C) = (A \setminus B) \setminus C\ \} \\
&= (e \cap t) \setminus (\bar{t} \cup (e' \setminus t) \cup \ell) && \{\ A \setminus (A \setminus B) = A \cap B\ \} \\
&= (e \cap t) \setminus ((e' \setminus t) \cup \ell) && \{\ t \setminus \bar{t} = t\ \} \\
&= (e \cap t) \setminus \ell && \{\ t \cap (e' \setminus t) = \emptyset\ \} \\
&= e \cap t && \{\ \bar{t} \subseteq s \implies t \cap s = \emptyset \implies t \cap \ell = \emptyset\ \} \\
&= e' \cap t && \{\ e \sim_t e' \implies e \cap t = e' \cap t\ \} \\
&= e' \setminus s && \{\ \text{même raisonnement que depuis le début en remplaçant } e \text{ par } e'\ \}
\end{aligned}$$

De $e \setminus s = e' \setminus s$ on en déduit $e \setminus s \subseteq e'$. On développe à présent $\mathit{apply}(e,s)$:

$$\begin{aligned}
\mathit{apply}(e,s) &= e \cup (s \setminus \bar{e}) \\
&= e \cup ((\bar{t} \cup (e' \setminus t) \cup (e \setminus t) \cup \ell) \setminus \bar{e}) \\
&= e \cup ((\bar{t} \setminus \bar{e}) \cup ((e' \setminus t) \setminus \bar{e}) \cup ((e \setminus t) \setminus \bar{e}) \cup (\ell \setminus \bar{e}) && \{\ \text{distributivité de } \setminus \text{ sur } \cup\ \} \\
&\supseteq e \cup ((e' \setminus t) \setminus \bar{e}) \\
&= e \cup (e' \setminus t) && \{\ e \sim_t e' \implies e \text{ et } e' \text{ consistants} \implies e' \cap \bar{e} = \emptyset\ \} \\
&= e \cup e' && \{\ e \sim_t e' \implies e \cap t = e' \cap t \implies e' \cap t \subseteq e\ \} \\
&\supseteq e'
\end{aligned}$$

Nous venons de montrer $e' \subseteq \mathit{apply}(e,s)$, en conjonction avec $e \setminus s \subseteq e'$ on obtient : $e' \in [e \setminus s, \mathit{apply}(e,s)]$. □

Lemme 53 (page 104) *Soient P et $\hat{P}$ deux fonctions de transitions telles que pour tout état s et action a, $\left\|P(\cdot|s,a) - \hat{P}(\cdot|s,a)\right\|_1 \leq \alpha$. Alors les fonctions de valeur optimales V^* et $\hat{V}^*$ respectives des PDM $M = \langle \mathcal{S},\mathcal{A},P,R\rangle$ et $\hat{M} = \langle \mathcal{S},\mathcal{A},\hat{P},R\rangle$ vérifient pour tout état $s \in \mathcal{S}$:*

$$\left|V^*(s) - \hat{V}^*(s)\right| \leq \frac{\alpha\gamma R_{\max}}{(1-\gamma)^2}$$

avec $R_{\max} = \max_{s,a} R(s,a)$.

Démonstration. Soient s un état et a une action. On suppose par hypothèse de récurrence sur l'horizon h, que pour tout s, $\left|V^h(s) - \hat{V}^h(s)\right| \leq \epsilon$.

• Cas initial $h = 0$

Avec $V^0 = \hat{V}^0 = 0$ on a $\left|V^0(s) - V^0(s)\right| = 0 \leq \epsilon$

• Récurrence

En développant $\left|(\mathfrak{B}^a V^h)(s) - (\hat{\mathfrak{B}}^a \hat{V}^h)(s)\right|$ on obtient :

$$\begin{aligned}
&\left|(\mathfrak{B}^a V^h)(s) - (\hat{\mathfrak{B}}^a \hat{V}^h)(s)\right| \\
&= \left|R(s,a) + \gamma \sum P(s'|s,a)V^h(s') - R(s,a) - \gamma \sum \hat{P}(s'|s,a)\hat{V}^h(s')\right| \\
&= \gamma\left|\sum P(s'|s,a)V^h(s') - \sum \left(\hat{P}(s'|s,a) + P(s'|s,a) - P(s'|s,a)\right)\hat{V}^h(s')\right| \\
&= \gamma\left|\sum P(s'|s,a)V^h(s') - \sum P(s'|s,a)\hat{V}^h(s') + \sum \left(P(s'|s,a) - \hat{P}(s'|s,a)\right)\hat{V}^h(s')\right| \\
&= \gamma\left|\sum P(s'|s,a)\left(V^h(s') - \hat{V}^h(s')\right) + \sum \left(P(s'|s,a) - \hat{P}(s'|s,a)\right)\hat{V}^h(s')\right| \\
&\leq \gamma\left|\sum P(s'|s,a)\left(V^h(s') - \hat{V}^h(s')\right)\right| + \gamma\left|\sum \left(P(s'|s,a) - \hat{P}(s'|s,a)\right)\hat{V}^h(s')\right| \\
&\leq \gamma \sum P(s'|s,a)\left|V^h(s') - \hat{V}^h(s')\right| + \gamma\left|\sum \left(P(s'|s,a) - \hat{P}(s'|s,a)\right)\hat{V}^h(s')\right| \\
&\leq \gamma \sum P(s'|s,a)\epsilon + \gamma\left|\sum \left(P(s'|s,a) - \hat{P}(s'|s,a)\right)\hat{V}^h(s')\right| \\
&= \gamma\epsilon + \gamma\left|\sum \left(P(s'|s,a) - \hat{P}(s'|s,a)\right)\hat{V}^h(s')\right| \\
&\leq \gamma\epsilon + \gamma \sum \left|P(s'|s,a) - \hat{P}(s'|s,a)\right|\hat{V}^h(s') \\
&\leq \gamma\epsilon + \gamma \sum \left|P(s'|s,a) - \hat{P}(s'|s,a)\right|\frac{R_{\max}}{1-\gamma} \\
&\leq \gamma\epsilon + \gamma\alpha\frac{R_{\max}}{1-\gamma}
\end{aligned}$$

Par conséquent, pour les fonctions de valeurs V^{h+1} et $\hat{V}^{h+1}$ on en déduit :

$$\begin{aligned}
\left|V^{h+1}(s) - \hat{V}^{h+1}(s)\right| &= \left|\max_a (\mathfrak{B}^a V^h)(s) - \max_a (\hat{\mathfrak{B}}^a \hat{V}^h)(s)\right| \\
&\leq \max_a \left|(\mathfrak{B}^a V^h)(s) - (\hat{\mathfrak{B}}^a \hat{V}^h)(s)\right| \\
&\leq \gamma\epsilon + \gamma\alpha\frac{R_{\max}}{1-\gamma}
\end{aligned}$$

Sachant que $V^* = \lim_{h\to\infty} V^h$ et $\hat{V}^* = \lim_{h\to\infty} \hat{V}^h$ on obtient donc

$$\left|V^*(s) - \hat{V}^*(s)\right| \leq \gamma\epsilon + \gamma\alpha\frac{R_{\max}}{1-\gamma}$$

On cherche donc ϵ tel que :

$$\epsilon = \gamma\epsilon + \gamma\alpha\frac{R_{\max}}{1-\gamma}$$

Alors

$$\left|V^*(s) - \hat{V}^*(s)\right| \leq \gamma\alpha\frac{R_{\max}}{(1-\gamma)^2}$$

□

Annexe B

Notations

Notation	Définition
M	Un processus de décision markovien (PDM)
$\mathcal{S}$	Espace d'états d'un PDM
$\mathcal{A}$	Espace d'actions d'un PDM
P	Fonction de transition d'un PDM
R	Fonction de récompense d'un PDM
a	Une action d'un PDM ($a \in \mathcal{A}$)
s	Un état d'un PDM ($s \in \mathcal{S}$)
$\mathfrak{B}^a$	Opérateur de Bellman pour une action a
$\mathfrak{B}$	Opérateur de Bellman pour toutes les actions
V	Fonction de valeur d'un PDM
V^*	Fonction de valeur optimale d'un PDM
Q	Fonction de valeur d'action d'un PDM
$R_{\max}$	Valeur espérée maximale pour un PDM
γ	Facteur de pondération sur l'horizon d'un PDM
e,e',...	Effets
$apply(e,s)$	Fonction d'application d'un effet e à un état s
$\mathcal{X}$	Ensemble des variables propositionnelles
x,y,x_1,x_2,...	Variables propositionnelles
$s[K]$	Partie d'un terme s restreint aux variables de K
ℓ	Littéral
n	Nombre de variables propositionnelles : $n = \lvert\mathcal{X}\rvert$
$Parents_a(x'_i)$	Ensemble des parents de la variable x'_i dans le DBN d'une action a
$CPT_a(x'_i)$	Table de probabilités conditionnelles d'une variable x'_i pour une action a
ϕ,ψ,φ,...	Formules propositionnelles
$\mathcal{M}(\phi)$	Ensemble des modèles d'une formule ϕ
$Var(\phi)$	Ensemble des variables d'une formule ϕ

Notation	Définition
$\mathcal{T}$	Séquence d'états-effets
$D_{\mathcal{T}}$	Distribution induite par $\mathcal{T}$
$[e,e']$	Intervalle d'effets
$\mathcal{O}$	Famille d'intervalles observés
$\mathcal{O}_{\mathcal{T}}$	Observations induites par $\mathcal{T}$
$\mathcal{O}^t$	Famille d'intervalles observés depuis des états $s \subseteq t$
$f_{o,\mathcal{O}}$	Fréquence d'un intervalle o dans $\mathcal{O}$
D	Distribution de probabilités sur un ensemble d'effets
$\Delta(\mathcal{O}, D)$	Déviation d'une distribution D à $\mathcal{O}$
E, E^*	Ensembles d'effets
$[E]_s$	Ensembles d'effet minimal équivalent pour l'état s
$\hat{E}(\mathcal{O},T)$	Ensembles d'effets plausibles
T_s^k	Ensemble des termes de taille k contenus dans un état s
$Cl_k(E,s)$	k-clôture d'un ensemble d'effets E pour un état s
ϵ,α	Bornes d'erreurs
δ	Probabilités d'erreur, la confiance est $1-\delta$
$\hat{\star}$	Approximation d'un objet $\star$
$\tilde{O}(\cdot)$	Ordre de grandeur qui ignore les facteurs logarithmiques

Bibliographie

[Abbeel *et al.*, 2006] Abbeel, P., Koller, D. et Ng, A. Y. (2006). Learning Factor Graphs in Polynomial Time and Sample Complexity. *Journal of Machine Learning Research*, 7:1743–1788.

[Adelman, 2004] Adelman, D. (2004). A Price-Directed Approach to Stochastic Inventory/Routing. *Operations Research*, 52(4):499–514.

[Aspvall *et al.*, 1979] Aspvall, B., Plass, M. et Tarjan, R. E. (1979). A linear-time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters*, 8:121–123.

[Bahar *et al.*, 1997] Bahar, R., Frohm, E., Gaona, C., Hachtel, G., Macii, E., Pardo, A. et Somenzi, F. (1997). Algebraic decision diagrams and their applications. *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*, 206:188–191.

[Barto *et al.*, 1995] Barto, A., Bradtke, S. et Singh, S. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1-2):81–138.

[Bellman, 1957] Bellman, R. (1957). A markovian decision process. *Journal of Mathematics and Mechanics*, 6(4):679–684.

[Bonet et Geffner, 2003] Bonet, B. et Geffner, H. (2003). Labeled RTDP : Improving the Convergence of Real-Time Dynamic Programming. *In Proceedings of ICAPS*, numéro Section 2, pages 12–31.

[Bonet et Givan, 2006] Bonet, B. et Givan, B. (2006). Results of Probabilistic Track in the 5th International Planning Competition.

[Boutilier et Dearden, 1994] Boutilier, C. et Dearden, R. (1994). Using Abstractions for Decision-Theoretic Planning with Time Constraints. *In Proceedings of the National Conference on Artificial Intelligence*, pages 1016–1022.

[Boutilier *et al.*, 1995] Boutilier, C., Dearden, R. et Goldszmidt, M. (1995). Exploiting structure in policy construction. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1104–1111.

[Boutilier *et al.*, 2000] Boutilier, C., Dearden, R. et Goldszmidt, M. (2000). Stochastic Dynamic Programming with Factored Representations. *Artificial Intelligence*, 121(1-2):49–107.

[Boutilier *et al.*, 2001] Boutilier, C., Reiter, R. et Price, B. (2001). Symbolic Dynamic Programming for First-Order MDPs. *In Proceedings of the International Joint Conference on Artificial Intelligence*, pages 690–697.

[Brace *et al.*, 1991] Brace, K., Rudell, R. et Bryant, R. (1991). Efficient implementation of a BDD package. *In Proceedings of the 27th ACM/IEEE design automation conference*, pages 40–45. ACM.

[Brafman et Tennenholtz, 2003] Brafman, R. et Tennenholtz, M. (2003). R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*, 3:213–231.

[Bryant, 1986] Bryant, R. (1986). Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691.

[Darwiche et Marquis, 2002] Darwiche, A. et Marquis, P. (2002). A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264.

[de Farias, 2002] de Farias, D. P. (2002). *The linear programming approach to approximate dynamic programming : theory and application*. Thèse de doctorat.

[de Farias et Van Roy, 2003] de Farias, D. P. et Van Roy, B. (2003). The linear programming approach to approximate dynamic programming. *Operations Research*, pages 850–865.

[de Farias et Van Roy, 2004] de Farias, D. P. et Van Roy, B. (2004). On Constraint Sampling in the Linear Programming Approach to Approximate Dynamic Programming. *Mathematics of operations research*, 29(3):462–478.

[de Farias et Van Roy, 2006] de Farias, D. P. et Van Roy, B. (2006). A Cost-Shaping Linear Program for Average-Cost Approximate Dynamic Programming with Performance Guarantees. *Mathematics of Operations Research*, 31(3):597–620.

[Dean et Givan, 1997] Dean, T. et Givan, R. (1997). Model minimization in Markov decision processes. *Proceedings of the National Conference on Artificial Intelligence*, pages 106–111.

[Dean *et al.*, 1997] Dean, T., Givan, R. et Leach, S. (1997). Model Reduction Techniques for Computing Approximately Optimal Solutions for Markov Decision Processes. *In Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence*, pages 124–131. Providence, RI.

[Dean et Kanazawa, 1989] Dean, T. et Kanazawa, K. (1989). A model for reasoning about persistence and causation. *Computational intelligence*, 5:142–150.

[Dearden et Boutilier, 1997] Dearden, R. et Boutilier, C. (1997). Abstraction and approximate decision-theoretic planning. *Artificial Intelligence*, 89(1-2):219–283.

[Degris *et al.*, 2006] Degris, T., Sigaud, O. et Wuillemin, P.-H. (2006). Learning the structure of Factored Markov Decision Processes in reinforcement learning problems. *Proceedings of the 23rd international conference on Machine learning - ICML '06*, pages 257–264.

[Del Val, 1995] Del Val, A. (1995). An Analysis of Approximate Knowledge Compilation. *In Proceedings of the International Joint Conference on Artificial Intelligence*, volume 14, pages 830–836.

[Diuk *et al.*, 2009] Diuk, C., Li, L. et Leffler, B. R. (2009). The Adaptive k-Meteorologists Problem and Its Application to Structure Learning and Feature Selection in Reinforcement Learning. *In Proceedings of the 26th Annual International Conference on Machine Learning*, pages 249–256.

[Dowling, W.F. and Gallier, 1984] Dowling, W.F. and Gallier, J. (1984). Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *The Journal of Logic Programming*, 1(3):267–284.

[Fargier et Marquis, 2008] Fargier, H. et Marquis, P. (2008). Extending the Knowledge Compilation Map : Krom, Horn, Affine and Beyond. *Proceedings of the National Conference on Artificial Intelligence*, 8:442–447.

[Feng et Hansen, 2002] Feng, Z. et Hansen, E. (2002). Symbolic heuristic search for factored Markov decision processes. *Proceedings of the National Conference on Artificial Intelligence*, pages 455–460.

[Feng *et al.*, 2003] Feng, Z., Hansen, E. A. et Zilberstein, S. (2003). Symbolic generalization for on-line planning. *In Proceedings of Uncertainty in Artificial Intelligence*, volume 19, pages 209–216.

[Fikes et Nilsson, 1971] Fikes, R. et Nilsson, N. J. (1971). STRIPS : A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2:189–208.

[Gestrin *et al.*, 2002] Gestrin, C., Patrascu, R. et Shuurmans, D. (2002). Algorithm-directed exploration for model-based reinforcement learning in factored MDPs. *In Proceedings of the International Conference on Machine Learning*, pages 235–242.

[Givan *et al.*, 2003] Givan, R., Dean, T. et Greig, M. (2003). Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 147:163–223.

[Guestrin *et al.*, 2003] Guestrin, C., Koller, D., Parr, R. et Venkataraman, S. (2003). Effcient Solution Algorithms for Factored MDPs. *Journal of Artificial Intelligence Research*, 19:399–468.

[Hoeffding, 1963] Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30.

[Hoey *et al.*, 1999] Hoey, J., St-Aubin, R., Hu, A. et Boutilier, C. (1999). SPUDD : Stochastic planning using decision diagrams. *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 279–288.

[Howard, 1960] Howard, R. A. (1960). *Dynamic programming and Markov processes*. MIT Press Cambridge, MA, USA.

[Kakade, 2003] Kakade, S. (2003). *On the sample complexity of reinforcement learning*. Thèse de doctorat.

[Kautz *et al.*, 1995] Kautz, H., Kearns, M. et Selman, B. (1995). Horn approximations of empirical data. *Artificial Intelligence*, 74:129–145.

[Kearns et Koller, 1999] Kearns, M. et Koller, D. (1999). Efficient reinforcement learning in factored MDPs. *International Joint Conference on Artificial Intelligence*, 16:740–747.

[Kearns *et al.*, 1999] Kearns, M., Mansour, Y. et Ng, A. (1999). A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *In Proceedings of the 16th international joint conference on Artificial intelligence-Volume 2*, pages 1324–1331. Morgan Kaufmann Publishers Inc.

[Kearns et Singh, 1998] Kearns, M. et Singh, S. (1998). Near-optimal reinforcement learning in polynomial time. *In International Conference on Machine Learning*, volume 2.

[Kearns et Singh, 2002] Kearns, M. et Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2):209–232.

[Kersting *et al.*, 2004] Kersting, K., Otterlo, M. V. et De Raedt, L. (2004). Bellman goes relational. *Twenty-first international conference on Machine learning - ICML '04*, page 59.

[Koller et Parr, 1999] Koller, D. et Parr, R. (1999). Computing factored value functions for policies in structured MDPs. *In International Joint Conference on Artificial Intelligence*, pages 1332–1339.

[Kushmerick *et al.*, 1995] Kushmerick, N., Hanks, S. et Weld, D. (1995). An algorithm for probabilistic planning. *Artificial Intelligence*, 76(1-2):239–286.

[Lang *et al.*, 2003] Lang, J., Liberatore, P. et Marquis, P. (2003). Propositional Independence : Formula-Variable Independence and Forgetting. *Journal of Artificial Intelligence Research*, 18:391–443.

[Lesner, 2009] Lesner, B. (2009). Résolution exacte et approchée de problèmes de décision sous incertitude formulés en logique propositionnelle. *In Actes des 9es Rencontres des Jeunes Chercheurs en Intelligence Artificielle (RJCIA 2009)*, pages 175–189.

[Lesner et Zanuttini, 2010] Lesner, B. et Zanuttini, B. (2010). Résolution exacte et approchée de processus de décision markoviens représentés en logique propositionnelle. *Revue d'intelligence artificielle*, 24(2):131–158.

[Lesner et Zanuttini, 2011a] Lesner, B. et Zanuttini, B. (2011a). Efficient Policy Construction for MDPs Represented in Probabilistic PDDL. *In Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS 2011). AAAI Press*, pages 146–153.

[Lesner et Zanuttini, 2011b] Lesner, B. et Zanuttini, B. (2011b). Handling Ambiguous E ffects in Action Learning. *In Proceedings of the nineth European Workshop on Reinforcement Learning (EWRL 2011).*

[Li *et al.*, 2011] Li, L., Littman, M. L., Walsh, T. J. et Strehl, A. L. (2011). Knows what it knows : a framework for self-aware learning. *Machine Learning*, pages 399–443.

[Li *et al.*, 2006] Li, L., Walsh, T. et Littman, M. (2006). Towards a unified theory of state abstraction for MDPs. *In Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, pages 531–539.

[Liberatore, 2002] Liberatore, P. (2002). The size of MDP factored policies. *Proceedings of the National Conference on Artificial Intelligence*, pages 267–272.

[Littman *et al.*, 1995] Littman, M., Dean, T. et Kaelbling, L. (1995). On the complexity of solving Markov decision problems. *In Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 394–402.

[Pasula *et al.*, 2004] Pasula, H., Zettlemoyer, L. et Kaelbling, L. (2004). Learning probabilistic planning rules. *In 14Th International Conference on Automated Planning & Scheduling (ICAPS'04)*.

[Pasula *et al.*, 2007] Pasula, H., Zettlemoyer, L. et Kaelbling, L. (2007). Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, 29(1):309–352.

[Puterman, 1994] Puterman, M. L. (1994). *Markov decision processes : Discrete stochastic dynamic programming*. John Wiley & Sons, Inc. New York, NY, USA.

[Rintanen, 2003] Rintanen, J. (2003). Expressive equivalence of formalisms for planning with sensing. *In Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling*, pages 185–194.

[Rudell, 1993] Rudell, R. (1993). Dynamic variable ordering for ordered binary decision diagrams. *Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*, pages 42–47.

[Sanner et Boutilier, 2005] Sanner, S. et Boutilier, C. (2005). Approximate Linear Programming for First-order MDPs. *In Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 509–517.

[Sanner et Boutilier, 2006] Sanner, S. et Boutilier, C. (2006). Practical linear value-approximation techniques for first-order MDPs. *In Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

[Sanner et Boutilier, 2007] Sanner, S. et Boutilier, C. (2007). Approximate solution techniques for factored first-order MDPs. *In 17th International Conference on Automated Planning and Scheduling (ICAPS-07)*, numéro 1, pages 288–295.

[Sanner et McAllester, 2005] Sanner, S. et McAllester, D. (2005). Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference. *In International Joint Conference on Artificial Intelligence*, volume 19, page 1384.

[Schaefer, 1978] Schaefer, T. J. (1978). The complexity of satisfiability problems. *In Proceedings of the tenth ACM symposium on Theory of computing*, pages 216–226.

[Schuurmans et Patrascu, 2001] Schuurmans, D. et Patrascu, R. (2001). Direct value-approximation for factored mdps. *Neural Information Processing Systems*, 14:1579–1586.

[Selman et Kautz, 1996] Selman, B. et Kautz, H. (1996). Knowledge Compilation and Theory Approximation. *Journal of the Association for Computing Machinery*, 43:193–224.

[Singh, 1994] Singh, S. (1994). An upper bound on the loss from approximate optimal-value functions. *Machine Learning*, 16(3):227–233.

[Skvortsova et Hölldobler, 2004] Skvortsova, O. et Hölldobler, S. (2004). A Logic-based Approach to Dynamic Programming. *In AAAI Workshop on Learning and Planning in Markov Processes - Advances and Challenges*, pages 31–36.

[St-Aubin *et al.*, 2001] St-Aubin, R., Hoey, J. et Boutilier, C. (2001). APRICODD : Approximate policy construction using decision diagrams. *Advances in Neural Information Processing Systems*, pages 1089–1096.

[Strehl *et al.*, 2006a] Strehl, A., Li, L. et Littman, M. (2006a). Incremental model-based learners with formal learning-time guarantees. *In Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence (UAI 2006)*.

[Strehl *et al.*, 2009] Strehl, A., Li, L. et Littman, M. (2009). Reinforcement Learning in Finite MDPs : PAC Analysis. *Journal of Machine Learning Research*, 10:2413–2444.

[Strehl *et al.*, 2006b] Strehl, A., Li, L., Wiewiora, E., Langford, J. et Littman, M. L. (2006b). PAC model-free reinforcement learning. *Proceedings of the 23rd international conference on Machine learning*, page 888.

[Strehl et Littman, 2005] Strehl, A. et Littman, M. (2005). A theoretical analysis of model-based interval estimation. pages 856–863.

[Strehl et Littman, 2008] Strehl, a. et Littman, M. (2008). An analysis of model-based Interval Estimation for Markov Decision Processes. *Journal of Computer and System Sciences*, 74(8):1309–1331.

[Strehl *et al.*, 2007] Strehl, A. L., Diuk, C. et Littman, M. L. (2007). Effcient Structure Learning in Factored-state MDPs. *In Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 645.

[Sutton et Barto, 1998] Sutton, R. S. et Barto, A. G. (1998). *Reinforcement Learning : An Introduction*. MIT Press, Cambridge, MA, USA.

[Valiant, 1984] Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142.

[Walsh *et al.*, 2009] Walsh, T., Szita, I., Diuk, C. et Littman, M. (2009). Exploring compact reinforcement-learning representations with linear regression. *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI-09)*, pages 591–598.

[Wang et Joshi, 2008] Wang, C. et Joshi, S. (2008). First order decision diagrams for relational MDPs. *Journal of Artificial Intelligence Research*, 31:431–472.

[Weissman *et al.*, 2003] Weissman, T., Ordentlich, E., Seroussi, G., Verdu, S. et MJ (2003). Inequalities for the l1 deviation of the empirical distribution. *HP Labs, Palo Alto*.

[Williams et Baird, 1994] Williams, R. et Baird, L. (1994). Tight performance bounds on greedy policies based on imperfect value functions. *In Proceedings of the tenth yale workshop on adaptive and learning systems*.

[Younes et Littman, 2004] Younes, H. et Littman, M. (2004). PPDDL1. 0 : An extension to PDDL for expressing planning domains with probabilistic effects. *In In Proceedings of the 14th International Conference on Automated Planning and Scheduling*.

[Zanuttini et Hébrard, 2002] Zanuttini, B. et Hébrard, J. J. (2002). A unified framework for structure identification. *Information Processing Letters*, 81:335–339.

Planification et apprentissage par renforcement avec modèles d'actions compacts

Nous étudions les Processus de Décision Markoviens représentés de manière compacte via des langages de définition d'actions basés sur le langage STRIPS Probabiliste.

Une première partie de ce travail traite de la résolution de ces processus de manière compacte. Pour cela nous proposons deux algorithmes. Un premier, basé sur la manipulation de formules propositionnelles, permet de résoudre de manière approchée les problèmes dans des fragments propositionnels traitables du type Horn ou 2-CNF. Le second algorithme résout, quant à lui, efficacement et de manière exacte les problèmes représentés en PDDL probabiliste via l'introduction d'une notion de fonction de valeur d'action étendue.

La seconde partie concerne l'apprentissage de ces modèles d'actions. Nous proposons différentes méthodes pour résoudre le problème de l'ambiguïté des observations qui à lieu lors de l'apprentissage. Une première méthode heuristique basée sur la programmation linéaire donne de bons résultats en pratique, mais sans garantie théorique. Par la suite nous décrivons une méthode d'apprentissage dans le cadre « Knows What It Knows ». Cette approche donne quant à elle des garanties théoriques sur la qualité des modèles d'actions appris ainsi que sur le nombre d'exemples requis pour obtenir un modèle d'actions correct. Ces deux approches sont ensuite incorporées dans un cadre d'apprentissage par renforcement pour une évaluation en pratique de leurs performances.

Planning and Reinforcement Learning With Compact Actions

We study Markovian Decision Processes represented with Probabilistic STRIPS action models.

A first part of our works is about solving those processes in a compact way. To that end we propose two algorithms. A first one based on propositional formula manipulation allows to obtain approximate solutions in tractable propositional fragments such as Horn and 2-CNF. The second algorithm solves exactly and efficiently problems represented in PPDDL using a new notion of extended value functions.

The second part is about learning such action models. We propose different approaches to solve the problem of ambiguous observations occurring while learning. Firstly, a heuristic method based on Linear Programming gives good results in practice yet without theoretical guarantees. We next describe a learning algorithm in the "Knows What It Knows" framework. This approach gives strong theoretical guarantees on the quality of the learned models as well on the sample complexity. These two approaches are then put into a Reinforcement Learning setting to allow an empirical evaluation of their respective performances.

Mots-clés :

- **indexation Rameau : Intelligence artificielle ; Markov, Processus de ; Apprentissage automatique ; Planification ; Calcul des propositions**
- **indexation libre : Processus de décision markovien factorisé, Apprentissage par renforcement, Complexité d'exploration**

Discipline : Informatique et applications

Groupe de Recherche en Informatique, Image, Automatique et Instrumentation de Caen,
CNRS UMR 6072,Université de Caen Basse-Normandie BP 5186,14032 Caen Cedex, FRANCE