UNIVERSITÉ PARIS 13
SORBONNE PARIS CITÉ
LIPN — UMR CNRS 7030

T H È S E

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ PARIS 13

Discipline : Informatique

présentée et soutenue publiquement par

Christophe Rodrigues

le 21 Janvier 2013

Apprentissage incrémental de modèles d'action relationnels

Composition du jury

Céline Rouveirol, Professeur	Université Paris 13	*Directeur*
Pierre Gérard, Maître de conférences	Université Paris 13	*Co-directeur*
Christel Vrain, Professeur	Université d'Orléans	*Rapporteur*
Bruno Zanuttini, Maître de conférences HDR	Université de Caen Basse-Normandie	*Rapporteur*
Amal El Fallah Seghrouchni, Professeur	Université Pierre et Marie Curie	
Stuart Russell, Professeur	University of California, Berkeley	
Yann Chevaleyre, Professeur	Université Paris 13	

Remerciements

Mes premiers remerciements vont à ma directrice de thèse Céline Rouveirol. J'ai beaucoup appris à ses côtés grâce à sa rigueur scientifique et son encadrement sans faille. Merci également à Pierre Gérard de m'avoir mis le pied à l'étrier et de m'avoir fait confiance. Merci à vous deux d'avoir partagé tous ces moments enrichissants.

Merci aussi à mes rapporteurs, Christel Vrain et Bruno Zanuttini, de s'être intéressés à mes travaux et d'y avoir passé du temps. Merci pour les discussions et pour vos encouragements.

Merci également à Amal El Fallah Seghrouchni, Stuart Russell et Yann Chevaleyre d'avoir participé à mon jury. C'était pour moi un grand honneur.

Un remerciement particulier à Henry Soldano pour toutes les discussions que nous avons pu avoir et ses collaborations à mes travaux. Son enthousiasme et son aide ont été pour moi très stimulants.

Merci aussi à Aomar Osmani et Dominique Bouthinon pour leur esprit critique et leur sympathie. C'était un plaisir de partager du temps avec eux. Merci également à Yann Chevaleyre pour ses suggestions toujours pertinentes.

Merci aussi à Eric Alphonse qui savait trouver les mots pour m'encourager juste avant les deadlines.

Merci également à Aurélien, Thibault, Anna, Sarra, Pegah et Jorge d'avoir partagé à un moment ou à un autre leur bureau avec moi. Ce fut très agréable. Plus globalement, merci à tous les membres du LIPN qui m'ont permis de faire ma thèse dans d'excellentes conditions.

Enfin, merci à mon entourage, à Fanny ma compagne qui a supporté au quotidien d'être envahie par ma thèse. A mes parents, ma soeur Sandrine et ma nièce Andréa pour leur présence ainsi que leur aide matérielle et morale.

Sommaire

Chapitre 1

Introduction

Sommaire

Le présent travail se place dans le cadre de l'apprentissage automatique et porte plus précisément sur le problème de l'adaptation en ligne dans le but d'agir.

Un aspect fondamental de ce problème est que l'apprenant ne se limite pas à catégoriser ou classer des observations uniquement à des fins de prévision. Ici, l'apprentissage dépend de la propre expérience de l'apprenant que l'on désigne alors comme *agent*. L'agent obtient ses nouvelles observations par les actions qu'il entreprend sur l'environnement dans lequel il évolue, comme par exemple un robot explorant la planète Mars ou bien un personnage artificiel dans un jeu vidéo.

Pour parvenir à remplir un objectif, comme atteindre un cratère, la plupart des systèmes (de renforcement indirect ou de planification) supposent donnée la dynamique des actions. Sans cela, il serait nécessaire de fournir manuellement ces connaissances à l'agent. Procéder ainsi entre en contradiction avec un comportement autonome de l'agent.

En effet, cela signifie que chaque situation (ou la majorité de celles-ci) doit être entièrement prévue au préalable, par un expert du domaine. Cela implique concrètement que toutes les *conditions* possibles à la réalisation d'une *action* soient connues, ainsi que tous les *effets* de celle-ci sur l'environnement. Cela impose aussi de procéder de la sorte pour chaque environnement différent, ce qui est une hypothèse très restrictive. Notre but est donc d'étudier comment rendre un agent autonome en apprenant par

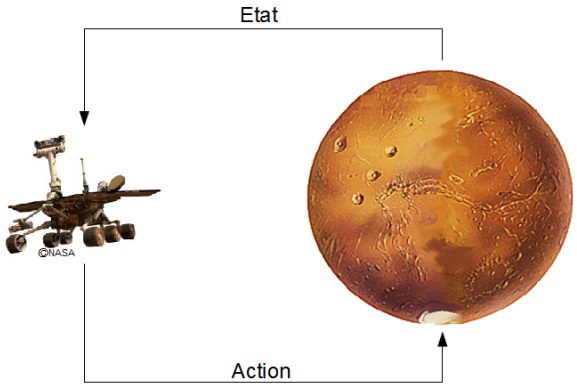


FIGURE 1.1.: Boucle sensori-motrice

son exploration du monde un *modèle d'action* qui permet de prévoir les effets de ses actions.

1.1. Agents autonomes adaptatifs

L'agent est en interaction avec son environnement et, par sa propre expérience, il apprend à y évoluer. La figure 1.1 illustre la boucle sensori-motrice dans laquelle l'agent évolue. Il se trouve dans un état s_t, effectue une action a_t et obtient en retour de la part de son environnement le nouvel état s_{t+1}. L'environnement est l'unique source d'exemples d'apprentissage et peut à certains égards être considéré comme un oracle fournissant des exemples sous la forme de triplets (s_t,a_t,s_{t+1}). Un exemple (ou *transition*) permet de décrire l'état de l'agent dans l'environnement avant et après exécution d'une action. Néanmoins, l'agent est contraint par l'état dans lequel il se trouve et ne peut obtenir de nouveaux exemples qu'en effectuant des actions à partir de cet état.

Dans la littérature, cette problématique est abordée sous différents angles : dans un cadre d'Apprentissage par Renforcement (AR) (Sutton et Barto, 1998) où à chaque état de l'agent est associé une récompense, le but de l'agent étant de maximiser le cumul des récompenses, mais également dans un cadre de planification (Russell et Norvig,

2003) où l'agent, à partir d'un état initial, doit pouvoir trouver un chemin, si possible le plus court, pour parvenir à un état but.

Dans ces deux cadres, la dynamique du monde n'est pas la principale finalité, celle-ci étant de gagner ou d'atteindre un but. Il est ainsi possible de contourner le problème : en AR, l'agent peut explorer tout l'espace sans jamais en représenter directement la dynamique ou *fonction de transition*. Cependant, cette connaissance permet d'anticiper en mémoire plusieurs actions à l'avance. De même, dans une optique de planification, l'ensemble des transitions possibles peut être représenté par un *modèle d'action*. Il devient ainisi indispensable à un planificateur n'interagissant pas directement avec un environnement. Le modèle d'action est alors souvent supposé donné par un expert du domaine ; l'agent ne peut pas être considéré comme complètement autonome.

L'apprentissage d'un modèle d'action peut s'inscrire dans le cadre suivant : l'agent n'a aucune connaissance *a priori* sur son environnement et récolte, au fur et à mesure de son expérience, des exemples d'actions qui ont réussi ou échoué. Il doit alors découvrir les *effets* des actions ainsi que les *conditions* dans lesquelles elles se produisent. Il s'agit donc d'un apprentissage incrémental qui procède par révision des modèles appris après chaque nouvelle interaction. L'apprentissage en ligne apporte des difficultés. Il est difficile d'envisager conserver tous les exemples d'apprentissage rencontrés par l'agent, car non seulement la mémoire de l'agent grandirait sans cesse à la rencontre d'un nouvel exemple, mais le processus de révision serait également de plus en plus long. Il devrait en effet traiter tous les exemples rencontrés, toujours plus nombreux. Il est donc nécessaire d'étudier comment obtenir un agent réactif et en mesure de se servir de ses connaissances acquises, mêmes partielles, à tout moment et sans pour autant qu'il n'y ait nécessité d'atteindre un certain niveau de performance avant de pouvoir les utiliser. Il n'est dès lors pas envisageable d'accumuler les expériences afin de se réduire à un apprentissage hors ligne.

La *perception* que l'agent a de son environnement peut aussi influencer l'apprentissage et entraîner des difficultés, notamment si la perception de l'environnement n'est pas parfaite et qu'il n'est pas possible de savoir dans quelle mesure il est possible de faire confiance à la perception de l'environnement.

1.2. Représentation

L'agent peut être confronté à des espaces d'états de très grande taille dont l'exploration exhaustive nécessiterait un temps considérable. Par exemple, dans un monde constitué uniquement de blocs sur une table, où un robot peut déplacer les blocs les uns sur les autres, il existe près de 60 millions d'états différents avec seulement 10 blocs. Cependant, dans la pratique, il n'est pas indispensable de parcourir tous les états pour obtenir un modèle d'action fiable. En effet, une représentation des états du monde capable de capturer la structure du problème rend inutile l'énumération de tout l'espace et permet ainsi d'obtenir une expression concise structurée de la stratégie apprise (Boutilier *et al.*, 2001).

Il est possible de représenter une telle structure à l'aide d'une représentation logique d'ordre un. Un état est alors représenté comme un ensemble d'objets et de relations entre eux. On peut donc décrire des états avec un tel langage de haut niveau. Par exemple, un bras robotisé peut identifier les objets qu'il peut saisir (p.ex. $bloc_1$, *table*) ainsi que leurs relations (p.ex. « le $bloc_1$ est sur la *table* »).

Une représentation relationnelle permet de généraliser les situations et de couvrir un grand nombre d'objets différents, en particulier par l'introduction de variables (l'exemple précédent peut alors s'écrire : « il existe un bloc B tel que $bloc_B$ est sur la *table* »). Une telle abstraction permet aussi d'envisager une réutilisation des connaissances acquises sur des environnements plus complexes permettant une nouvelle manière d'appréhender la taille de l'espace d'états.

Les travaux de Dzeroski *et al.* (2001) ont décrit l'usage d'une représentation relationnelle en AR pour l'apprentissage de stratégies d'action. Mais à notre connaissance, très peu de travaux se sont intéressés à l'apprentissage de la fonction de transition pour aider l'apprentissage de stratégies ou encore de modèle d'action relationnel pour la planification autonome.

1.3. Contributions

Une des principales contributions de nos travaux réside dans la conception et le développement d'une méthode pour l'apprentissage incrémental de modèles d'action dans des environnements relationnels.

L'approche originale IRALe (Incremental Relational Action Learning) (Rodrigues *et al.*, 2010a) permet à un agent d'apprendre uniquement par son interaction avec son environnement un modèle d'action relationnel directement utilisable par un planificateur. Nous proposons une approche autonome, ascendante, capable d'améliorer par généralisations successives (ou également par spécialisations) un modèle d'action initialement vide. Nous apportons également des garanties de convergence pour IRALe et effectuons différentes expérimentations afin d'évaluer les capacités de l'approche aussi bien en termes de qualité des prévisions que d'utilisation direct des modèles par un planificateur. Nous comparons aussi IRALe à d'autres approches existantes.

Nous développons également une méthode d'apprentissage dit actif (Rodrigues *et al.*, 2012). Ici, le but est de donner à l'agent la possibilité d'orienter « activement » l'exploration de son environnement en focalisant son apprentissage sur des actions susceptibles d'améliorer son modèle courant. Pour trouver ces actions, l'agent anticipe un pas de généralisation, à partir de son état courant sur des parties du modèle supposées insuffisamment générales. Nous montrons expérimentalement que ce comportement permet de réduire le nombre d'interactions nécessaires entre l'agent et son environnement afin d'obtenir un modèle correct et complet.

Enfin, nous avons généralisé l'approche IRALe afin de faire face à une incertitude sur la perception de l'environnement (Rodrigues *et al.*, 2010b). Nous supposons alors que l'agent ne possède pas à tout moment une perception parfaite de l'ensemble du monde dans lequel il évolue. Les généralisations et spécialisations du modèle d'action sont alors adaptées afin de traiter des exemples incertains. Nous présentons des résultats montrant la robustesse de l'approche proposée.

1.4. Structure du manuscrit

Avant de présenter nos contributions, nous commençons par introduire au chapitre 2 les notions nécessaires à la compréhension de notre approche d'une part sur la formalisation de l'action et d'autre part sur la représentation relationnelle ainsi que la généralisation dans ce contexte. Nous discutons ensuite au chapitre 3 des différentes méthodes existantes et proches de notre démarche. Ces deux chapitres permettent de préciser notre problématique de représentation de modèles d'action, ainsi que les différentes méthodes d'apprentissage utilisées dans des travaux connexes.

Au chapitre 4, nous présentons l'approche IRALe. Nous détaillons l'expressivité des modèles d'action appris et présentons l'approche en nous appuyant initialement sur une représentation propositionnelle afin d'en expliquer les principaux aspects.

Ensuite, au chapitre 5, nous étudions une méthode d'apprentissage actif permettant d'améliorer l'exploration de l'environnement par l'agent. Nous commençons par introduire les travaux proches, puis nous continuons en présentant, comme pour IRALe, l'approche à l'aide d'une représentation propositionnelle pour finalement développer la méthode d'exploration active relationnelle.

Enfin, au chapitre 6, nous présentons une généralisation d'IRALe à la présence de bruit dans la perception de l'environnement.

Chapitre 2

Pré-requis

Sommaire

Dans ce chapitre, nous abordons les deux notions fondamentales nécessaires au développement et la description de nos travaux. D'une part, nous introduisons ce qui relève de la dynamique de l'action. Nous nous plaçons alors dans le cadre bien défini des processus de décision Markovien(PDM) (Puterman, 1994). D'autre part, nous présentons les définitions et implications permettant l'usage d'une représentation relationnelle des environnements.

2.1. Processus de Décision Markovien

Un PDM permet de formaliser l'ensemble des éléments nécessaires à la description des interactions entre un agent et son environnement. Par la suite, nous utilisons les notations suivantes :

Un PDM est un ensemble $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ avec :
– $\mathcal{S}$ un espace d'états
– $\mathcal{A}$ un espace d'actions
– le temps est divisé en pas de temps discrets $t \in \mathbb{N}^+$

- $\mathcal{A}(s_t)$ est l'ensemble des actions disponibles à partir de s_t
- dans l'état $s_t \in \mathcal{S}$, l'agent émet l'action $a_t \in \mathcal{A}(s_t)$
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ une fonction de transition qui spécifie les conséquences des actions en termes de nouveaux états

$$\mathcal{T}(s, a, s') = \mathcal{T}_{sa}^{s'} = Pr\left\{s_{t+1} = s' \mid s_t = s, a_t = a\right\} \tag{2.1}$$

- Une politique $\pi_t(s, a) \in [0, 1]$ est la probabilité qu'à l'instant t, l'agent choisisse l'action a dans l'état s.
- l'action est choisie par l'agent selon sa politique courante π_t
- en retour, il reçoit de l'environnement une récompense immédiate $r_{t+1} \in \mathbb{R}$ et son nouvel état $s_{t+1} \in \mathcal{S}$
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ une fonction de récompense qui spécifie les conséquences des actions en termes de récompenses.

$$\mathcal{R}(s, a) = \mathcal{R}_{sa} = E\left\{r_{t+1} \mid s_t = s, a_t = a\right\}$$

Un environnement *déterministe* est un cas particulier de la fonction de transition où $\mathcal{T}_{sa}^{s'} = 1$. Dans ce cas, une action dans un état donné conduit toujours à un même état.

La vie de l'agent est décomposée en *épisodes*. Chaque épisode commence dans un état initial et se termine dans un état final ou après un nombre de pas de temps déterminé. Par exemple, dans un labyrinthe, la sortie est un état final et dès que l'agent l'atteint, un nouvel épisode commence avec un nouvel état initial.

En *apprentissage par renforcement*, le but du système est de maximiser le cumul des récompenses reçues à chaque pas de temps depuis l'instant t ; le retour attendu vaut :

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T$$

Cependant, les récompenses immédiates sont considérées comme plus importantes

qu'une récompense lointaine dans le futur. On ajoute donc un facteur de dépréciation γ :

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

où $\gamma \in [0, 1[$.

Le facteur γ permet de régler l'importance accordée aux récompenses futures vis-à-vis des récompenses immédiates :

- Si γ est proche de 0, les récompenses futures sont ignorées et seules les récompenses immédiates comptent.
- Si γ est proche de 1, les récompenses futures comptent presque autant que les récompenses immédiates.

L'objectif de l'agent est donc d'apprendre à agir de manière à maximiser R_t. Pour ce faire, on se dote de mécanismes d'évaluation des états, qui permettent de décider entre deux états lequel permettra de maximiser le retour R_t. Cette évaluation prend la forme d'une fonction valeur.

2.1.1. Fonction valeur

Comme les gains accumulés dépendent de la politique de l'agent, nous nous intéressons au lien qui existe entre ces deux notions. La fonction valeur renseigne sur l'intérêt d'être dans un état particulier. La valeur d'un état est relative à la proximité des récompenses. La fonction valeur est associée à une politique π et est notée $V^\pi(s)$ pour l'état s :

$$\begin{aligned} V^\pi(s) &= E_\pi \{R_t \mid s_t = s\} \\ &= E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right\} \end{aligned}$$

pour tout $s \in \mathcal{S}$ où $E_\pi \{\cdot \mid \cdot\}$ désigne une espérance en suivant π.

La valeur d'un état est donc le cumul des récompenses qu'on attend en moyenne à partir de cet état. Pour prendre en compte les actions, on définit la *fonction Qualité* ou Q-fonction qui renseigne sur la qualité d'une action a émise dans un état s donné. Pour tout $s \in \mathcal{S}$ et $a \in \mathcal{A}(s)$:

$$\begin{aligned} Q^\pi(s,a) &= E_\pi \{R_t \mid s_t = s, a_t = a\} \\ &= E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right\} \end{aligned}$$

L'équation de Bellman (1961) permet de définir récursivement la *fonction valeur* d'un état s par rapport à la valeur d'états postérieurs à s :

$$V^\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(s,a) \left[\mathcal{R}_{sa} + \gamma \sum_{s' \in \mathcal{S}^+} \mathcal{T}_{sa}^{s'} V^\pi(s') \right]$$

Une politique π est dite *meilleure* qu'une autre politique π' si les retours attendus en la suivant sont plus importants dans tous les états, autrement dit :

$$\pi \geq \pi' \; ssi \; \forall s \in S, V^\pi(s) \geq V^{\pi'}(s)$$

Il existe toujours une politique meilleure que toutes les autres. Une telle politique est dite *optimale* et on la note π^*.

On définit la fonction valeur optimale comme suit :

$$V^*(s) = max_\pi V^\pi(s)$$

On définit de même la fonction action-valeur (ou Qualité) optimale :

$$Q^*(s,a) = max_\pi Q^\pi(s,a)$$

$$Q^*(s,a) = E\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\}$$

Nous avons introduit les principales notions permettant de décrire la dynamique des interactions entre un agent et son environnement. A la section suivante, nous développons la représentation des états permettant de décrire l'environnement.

2.2. Représentation relationnelle

Dans cette section, nous introduisons les notations nécessaires à la description d'états et d'actions relationnelles ainsi que nos hypothèses de généralisation dans cette représentation.

2.2.1. Définitions

L'agent perçoit le monde en termes d'*objets* qui le composent et de *relations* entre ces derniers. On peut formaliser cette représentation à l'aide de *Datalog*, un sous-ensemble de la logique du premier ordre (Ceri *et al.*, 1990) où :

- Une *constante* représente un objet et est dénotée par un identificateur commençant par une minuscule (a,b,c...).
- Une *variable* prend ses valeurs sur un domaine d'objets et est dénotée par un nom commençant par une majuscule (X,Y...).
- Un *terme* est une *constante* ou une *variable*.
- Un *symbole de prédicat* représente une relation entre objets. Il possède une *arité* ou nombre d'arguments sur lequel il porte. On le note alors symbole de prédicat/arité (exemple : $on/2$).
- Un *atome* est un *symbole de prédicat* appliqué à des *termes* (i.e. respectivement $déplacer(a, X)$, $sur(b, Y)$).

– Un *fait* ou *atome clos* est un *atome* complètement instancié (sans *variable*).
– Un *littéral* est un *atome* ou la négation d'un *atome*.

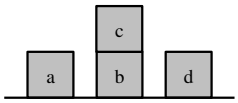


FIGURE 2.1.: Exemple d'état dans un monde de blocs

L'exemple d'état illustré figure 2.1 dans un monde de blocs peut s'écrire sous la forme de la conjonction d'atomes suivante :
$libre(a), libre(c), libre(d), sur(a,t), sur(b,t), sur(c,b), sur(d,t)$. Avec une représentation propositionnelle, le même exemple peut s'écrire :
$((0,0),(1,0),(1,1),(2,0))$ avec (x,y) une variable propositionnelle : x le numéro de pile et y la hauteur dans la pile.

La représentation propositionnelle ou vectorielle dépend du nombre de piles et de leur taille, car si un bloc est rajouté, il est nécessaire de définir une nouvelle variable propositionnelle. Avec une représentation relationnelle il n'est au contraire pas nécessaire de définir de nouveaux éléments dans la représentation. Pour rajouter un bloc à l'état par exemple, il suffit de rajouter ses relations par rapport à d'autres objets de l'état ainsi que ses propriétés s'il en possède (i.e. libre). Cela permet d'être indépendant du nombre d'objets dans l'état.

Une *expression* est soit un terme, soit un littéral, soit une conjonction de disjonctions de littéraux.

Une *clause* est une disjonction finie de littéraux. Une *clause de horn* possède au plus un littéral positif. Une *clause définie* possède exactement un littéral positif.

On dit que deux littéraux sont *compatibles* s'ils possèdent le même symbole de prédicat, la même arité et le même signe (positif ou négatif) (Plotkin, 1970).

Une *substitution* θ (Lloyd, 1987) est un ensemble fini de la forme $\{v_1/t_1, ..., v_n/t_n\}$, où chaque v_i est une variable, chaque t_i est un terme distinct de v_i, et où les variables $v_1, ..., v_n$ sont distinctes. Chaque élément v_i/t_i est un *appariement* pour v_i.

Soit $\theta = \{v_1/t_1, ..., v_n/t_n\}$ une substitution et E une expression. Alors $E\theta$, une instance de E par θ, est l'expression obtenue en remplaçant chaque occurrence de la variable v_i de E par le terme t_i.

Par exemple, soit une expression E = $libre(A), libre(B), sur(A, table)$ et une substitution $\theta = \{A/c, B/d\}$. Alors Eθ = $libre(c), libre(d), sur(c, table)$.

Une substitution θ est *injective* ssi, $\forall v_i$ et $v_j \in \theta$ avec $v_i \neq v_j$ et v_i/t_i, v_j/t_j alors $t_i \neq t_j$.

Une *substitution inverse* permet de substituer dans une expression des termes par des variables. Cependant, il n'est pas possible de définir directement une substitution inverse comme la fonction inverse d'une substitution car un terme pourrait être substitué par plusieurs variables, et la question de savoir quelle variable choisir se poserait. Une solution apportée par Nienhuys-Cheng et Flach (1991) propose de prendre en compte l'occurrence des termes qui, elle, est unique. La substitution inverse devient alors une fonction d'occurrences de termes vers des variables.

Cependant, pour des *substitutions injectives* la question ne se pose pas, puisque chaque variable distincte est substituée par un terme distinct. Si $\theta = \{v_1/t_1, ..., v_n/t_n\}$, on définit alors une substitution injective inverse $\theta^{-1} = \{t_1/v_1, ..., t_n/v_n\}$. Une substitution injective inverse permet de remplacer toutes les occurrences d'un terme par une variable. Ainsi, par l'introduction de variables, une substitution inverse permet de généraliser une expression.

Par exemple, soit une expression E = $libre(c), libre(d), sur(c, table)$ et une substitution inverse $\theta^{-1} = \{c/A, d/B\}$, alors E$\theta^{-1}$ = $libre(A), libre(B), sur(A, table)$

2.2.2. Relation de généralité

Avant tout, il est nécessaire de définir comment une règle peut expliquer un exemple ou le couvrir. En programmation logique inductive (PLI), une solution à ce problème

fondamental est la couverture. La θ-subsomption est une relation de couverture qui a été largement utilisée (Plotkin, 1970).

Définition 1 (θ-subsumption) *une clause c_1 θ-subsume une clause c_2 s'il existe une substitution θ telle que tous les littéraux de la clause $c_1\theta$ soient inclus dans c_2.*

Exemple avec des clauses Datalog (tous les objets sont de type bloc) :

c_1 : $dépilable(A, B) \leftarrow libre(A), sur(A, B)$

c_2 : $dépilable(b1, b2) \leftarrow libre(b1), sur(b1, b2), sur(b2, b3)$

c_1 θ-subsume c_2 avec la substitution $\theta_1 = \{A/b1, B/b2\}$

Afin de représenter les modèles d'action, il est nécessaire de prendre en compte l'identité de chaque variable ou objet à part entière. Ce problème trouve une solution avec la subsomption sous identité objet (OI) (Esposito *et al.*, 1996).

Définition 2 (OI-subsomption) *Une formule G OI-subsume une formule S ssi il existe une substitution θ telle que $G_{OI}\sigma \subseteq S_{OI}$ où θ est une substitution injective (deux variables distinctes du domaine de θ sont assignées à des termes distincts). On dit alors que G est une généralisation de S par OI-subsomption que l'on note $G \geq_{oi} S$.*

Définition 3 (substitution OI) *Soit une formule G et une formule S. S'il existe une substitution injective θ telle que $G\theta \subseteq S$ et que pour toute constante c avec $V/c \in \theta$, $c \in S$ et $c \notin G$ alors θ est une substitution OI.*

Définition 4 (constante OI) *Une constante c est OI pour une substitution θ s'il n'existe pas d'appariement V/c dans θ avec V une variable quelconque.*

Sous hypothèse OI, la clause suivante :

$déplaçable(A, B) \leftarrow libre(A), libre(B), sur(A, table)$

inclut implicitement les inégalités suivantes :

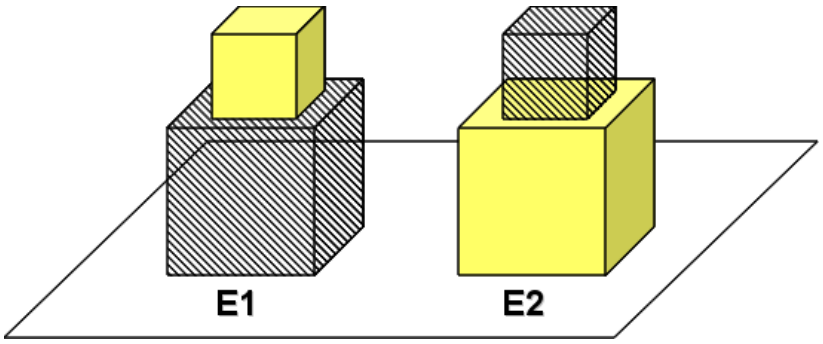


FIGURE 2.2.: Exemples $E1$ et $E2$

$déplaçable(A, B) \leftarrow libre(A), libre(B), sur(A, table), [A \neq B], [B \neq table], [A \neq table]$

La relation d'OI-subsomption est une relation injective : toute variable appartenant à une clause de départ est substituée par un seul terme exclusif de la clause d'arrivée.

2.2.3. Généralisation

La subsomption OI est une relation de généralité plus contrainte que la θ-subsomption (Plotkin, 1970). Il a été montré (Esposito *et al.*, 1996) que l'ensemble des généralisations sous OI d'une conjonction de littéraux est l'ensemble de ses sous-ensembles (à un renommage près) et l'opérateur de généralisation complet est la règle d'abandon de littéraux. Cependant, comme noté par exemple dans (Haussler, 1989) et (Kietz, 1993), la généralisation minimale de deux exemples n'est pas nécessairement unique et peut produire plusieurs moindres généralisés, chacun correspondant à un plus grand sous-ensemble commun entre les deux exemples d'entrée.

Soient les deux exemples $E1$ et $E2$ de la figure 2.2. D'après les travaux de Ferilli *et al.* (2002), cette figure décrit les exemples :

$$E1 : \quad pile(e1) \leftarrow appartient(e1, b1), appartient(e1, b2), sur(b1, b2), \\ cube(b1), cube(b2), petit(b1), grand(b2), jaune(b1), rayure(b2).$$

$$E2 : \quad pile(e2) \leftarrow appartient(e2, b3), appartient(e2, b4), sur(b3, b4), \\ cube(b3), cube(b4), petit(b3), grand(b4), jaune(b4), rayure(b3).$$

Si l'on généralise $E1$ et $E2$ sous OI, on obtient :

$$G1 : \quad pile(X) \leftarrow appartient(X, X1), appartient(X, X2), sur(X1, X2), \\ cube(X1), cube(X2), petit(X1), grand(X2).$$

et

$$G2 : \quad pile(X) \leftarrow appartient(X, X1), appartient(X, X2), cube(X1), \\ cube(X2), jaune(X1), rayure(X2).$$

Sous θ-subsomption, selon l'algorithme de calcul de moindre généralisé de Plotkin (1970), on obtient :

$$G : \quad pile(X) \leftarrow appartient(X, X1), appartient(X, X2), appartient(X, X3), \\ appartient(X, X4), sur(X1, X2), cube(X1), cube(X2), cube(X3), \\ cube(X4), petit(X1), grand(X2), jaune(X3), rayure(X4).$$

Sous OI-subsomption, le moindre généralisé $G1$ signifie que les exemples $E1$ et $E2$ sont composés d'un petit bloc sur un bloc plus grand, alors que le moindre généralisé $G2$ explique que chacun des deux exemples possède un cube jaune et un cube à rayure. $G1$ et $G2$ représentent bien chacun un plus grand sous-ensemble commun entre $E1$ et $E2$. Une généralisation met en avant la taille des blocs, et l'autre leur aspect. $G1$ et $G2$ sont incomparables dans cet exemple.

Définition 5 (incomparabilité) *Deux clauses c_1 et c_2 sont incomparables s'il n'existe pas de substitution θ telle que $c_1\theta \subseteq c_2$ ou $c_2\theta \subseteq c_1$.*

Définition 6 (équivalence) *Deux clauses c_1 et c_2 sont équivalentes sous OI-subsomption, si $c_1 \geq_{oi} c_2$ et $c_2 \geq_{oi} c_1$*

On note que sous θ-subsomption, la généralisation obtenue peut contenir plus de littéraux que chacun des deux exemples dont elle provient. La généralisation contient aussi plus de termes que les exemples de départ. Le moindre généralisé représente

une sous-structure commune aux deux exemples, il a une forme plus intuitive sous OI-subsomption où les moindres généralisés obtenus ne contiennent pas plus de littéraux et de termes différents que les exemples de départ.

La généralisation de deux clauses c_1 et c_2 constituées de n littéraux peut engendrer jusqu'à 2^n clauses, chacune sous-ensemble de c_1 et c_2. Comme noté par Esposito *et al.* (2004), pour chaque sous-ensemble, il faut de surcroît vérifier que la contrainte d'identité objet soit respectée et enfin comparer chaque généralisation trouvée aux autres pour en supprimer les généralisations *équivalentes* ou en relation de généralité afin d'obtenir un ensemble minimal de solutions.

Afin de réduire la complexité du calcul de toutes les généralisations possibles de deux clauses, Esposito *et al.* (2004) proposent d'introduire des biais de recherche en la taille des généralisations ainsi que le nombre total de généralisations. Afin de limiter encore l'espace de recherche, ils utilisent un graphe de dépendance sur les concepts, supposé donné. Chaque concept est représenté sous forme de clause(s) Datalog. Les exemples sont testés par les concepts ou clauses du graphe afin d'en éliminer les littéraux déductibles. Ce traitement permet notamment de réduire la longueur des clauses avant de calculer des moindres généralisés entre l'exemple réduit et la théorie courante.

L'algorithme 1 détaille le calcul de toutes les généralisations OI possibles entre deux clauses Datalog C_r et C_x. Nous utilisons cet algorithme dans la suite de nos travaux. On détaille l'approche en section 4.3.2.

Pour un sous-ensemble possible de littéraux compatibles appartenant à deux clauses C_r et C_x, pour chaque couple, on calcule une généralisation OI (cf. algorithme 2). Pour un couple de littéraux (l,l'), deux termes t_i et t'_i sont généralisables s'il existe un terme tg_i et deux substitutions θ_i et θ'_i telles que $tg_i\theta_i = t_i$ et $tg_i\theta_i{}' = t'_i$. Cependant, il est nécessaire de vérifier si la nouvelle généralisation tg_i ne contredit pas les substitutions OI déjà trouvées sur les autres termes déjà généralisés (appartenant aux couples de littéraux précédents et courants) et conservées dans θ et θ'. Si celles-ci sont OI alors il n'y a pas de contradiction, on continue le processus pour les autres termes du couple.

Autrement, s'il n'est pas possible pour au moins un couple de termes de l et l' de

Algorithme 1 $GEN_{OI}(C_r,C_x) : L_{GEN}$

Entrée: Les conjonctions C_r et C_x
Sortie: Des généralisations OI de C_r et C_x

$L_{GEN} \leftarrow \emptyset$
pour tout sous-ensemble co de littéraux compatibles (l_r, l_x) t.q $l_r \in C_r$, $l_x \in C_x$ **faire**
  $L_{co} \leftarrow \emptyset$, $\theta_r \leftarrow \emptyset$, $\theta_x \leftarrow \emptyset$
  tant que $co \neq \emptyset$ **faire**
    $(l_r, l_x) \in co$
    $L_{co} \leftarrow L_{co} \cup LIT\text{-}GEN_{OI}(l_r, l_x, \theta_r, \theta_x)$
    $co = co \backslash (l_r, l_x)$
  fin tant que
  $L_{GEN} \leftarrow L_{GEN} \cup (L_{co}, \theta_r, \theta_x)$
fin pour
retourne toutes les généralisations obtenues dans L_{GEN} ainsi que chacune des substitutions associées θ_r et θ_x

trouver un terme tg, alors on abandonne le littéral. On répète toute la procédure pour chaque sous-ensemble afin d'obtenir un ensemble de généralisations. Pour chaque généralisation, on conserve également les substitutions θ_r et θ_x associées. Elles peuvent servir ultérieurement à étendre les généralisations sur d'autres exemples en respectant les substitutions trouvées. Pour obtenir un ensemble minimal de moindres généralisés sous OI, il est encore nécessaire de comparer les généralisations afin de supprimer d'éventuels sous-ensembles ou doublons.

Algorithme 2 $LIT\text{-}GEN_{OI}(l,l',\theta,\theta') : g$

Entrée: Deux littéraux l et l', des substitutions OI θ et θ' à respecter en cours de construction

Sortie: La généralisation g de l et l' et les contraintes OI θ et θ' éventuellement mises à jour

1: Soit $l = p(t_1, ..., t_n), l' = p(t'_1, ..., t'_n)$
2: $i \leftarrow 1$, $stop \leftarrow faux$
3: **tant que** $i \leqslant n$ et $stop \neq vrai$ **faire**
4: **si** il existe un terme tg_i et deux substitutions OI θ_i et θ'_i telles que : $tg_i\theta_i = t_i$ et $tg_i\theta'_i = t'_i$ **alors**
5: $\theta_{tmp} \leftarrow \theta_{tmp}\theta_i$
6: $\theta'_{tmp} \leftarrow \theta'_{tmp}\theta'_i$
7: **sinon**
8: $stop \leftarrow vrai$
9: **fin si**
10: $i \leftarrow i + 1$
11: **fin tant que**
12: **si** $stop = faux$ **alors**
13: $\theta \leftarrow \theta_{tmp}$
14: $\theta' \leftarrow \theta'_{tmp}$
15: **retourne** $g = p(tg_1, ..., tg_n)$
16: **sinon**
17: **retourne** $\emptyset$
18: **fin si**

Chapitre 3

Etat de l'art

Sommaire

3.1. Représentation de modèles d'action relationnels

Par ses actions, l'agent observe la dynamique de l'environnement. La représentation de cette dynamique ou modèle d'action prend en compte certaines hypothèses qu'il est nécessaire de délimiter. Il est hors de portée de cette thèse de les étudier exhaustivement. Nous présentons les hypothèses les plus fréquemment utilisées.

On se place dans le "frame assumption" (McCarthy et Hayes, 1969) stipulant que lorsqu'un agent émet une action sur son environnement, tout ce qui n'est pas explicitement affecté par l'action demeure implicitement inchangé. Des travaux traitent du "frame problem" en représentant explicitement tout ce qui est affecté ou non directement par les actions comme (Otero, 2005) ou avec notamment le Calcul des situations comme (Boutilier *et al.*, 2001), (Sanner et Boutilier, 2009). Néanmoins ces langages plus expressifs posent des problèmes de complexité engendrés par la grande quantité d'information introduite pour décrire la moindre situation.

On se place également dans l'hypothèse du monde clos : tous les faits non représentés explicitement sont considérés comme faux. Ainsi, un état peut être représenté de façon compacte par une conjonction de faits.

Alors que les nombreux planificateurs existants possèdent la principale propriété d'être génériques ou indépendants des domaines (Hoffmann et Nebel, 2001), ils requièrent en entrée le modèle d'action des domaines à traiter. Ces modèles sont le plus souvent donnés par des experts des domaines et représentés par un ensemble de règles de type STRIPS (Fikes et Nilsson, 1971).

Nous nous focalisons sur l'aspect relationnel des objets pouvant entrer dans la description de l'action. Le langage STRIPS est adéquat pour la description de la dynamique d'un environnement, car il permet de représenter simplement quelles relations et objets ont été affectés par l'action.

On définit une règle r à la STRIPS composée de trois ensembles comme suit :

1. Les *pré-conditions* $r.p$ ou ce qui doit être vérifié avant l'action pour en permettre le déclenchement. Elles sont représentées par une conjonction d'atomes (pas de négation).
2. L'action $r.a$ à proprement parler ainsi que les objets sur lesquels elle opère. L'action est constituée d'un atome l'identifiant et des arguments sur lesquels elle porte.
3. Les *effets* $r.e$ ou conséquences de l'action décrivant ce qui devient vrai ou faux une fois l'action exécutée. La partie *effet* est composée de deux sous-ensembles d'atomes :

   a) $r.e.add$, un ensemble d'atomes devenant vrais après l'exécution de l'action,

   b) $r.e.del$, un ensemble d'atomes devenant faux après exécution de l'action.

   Une action $r.a$ ne possède pas d'autres effets que ceux décrits par $r.e$. Ainsi, tous les faits ne faisant pas partie des effets sont considérés comme inchangés par l'action.

Dans une règle STRIPS, les seules variables autorisées sont celles faisant partie des paramètres de l'action. Sous cette forme, une règle se note : $r.p/r.a/r.e$.

De nombreux raffinements du langage STRIPS ont été développés. Beaucoup sont normalisés au sein du langage PDDL (McDermott *et al.*, 1998) en constante évolution avec notamment :

- La possibilité d'exprimer pour une même action que des préconditions différentes peuvent aboutir à des effets différents. Cela permet de représenter des situations plus complexes.
- La représentation de *valeurs numériques* permettant entre autres de représenter des seuils à la vérification de faits.
- Des *actions continues* dont les effets peuvent durer un certain temps après l'exécution de l'action).
- La possibilité de représenter une *hiérarchie* entre les différents prédicats permettant de regrouper des actions primitives en action plus abstraites.
- Le traitement des actions aux *effets non-déterministes* en représentant plusieurs effets possibles pour une même action (chacun avec une probabilité de réalisation donnée).

A partir du langage de représentation d'action STRIPS et ses raffinements, dans la section suivante, nous nous intéressons aux nombreux travaux portant sur l'apprentissage de ces modèles d'action. Nous nous proposons de présenter ci-dessous les différentes approches proches de notre problématique.

3.2. Apprentissage de modèles d'action relationnels

Il est difficile de séparer les différents travaux en fonction des hypothèses qu'ils émettent, tellement celles-ci peuvent être variées. Parmi ces hypothèses on retrouve néanmoins principalement le fait que l'apprentissage soit incrémental ou hors-ligne, ainsi que la nature des environnements, déterministes, non-déterministes ou en présence de bruit de différentes formes.

L'origine des exemples d'apprentissage peut prendre plusieurs formes. Certains travaux émettent l'hypothèse de séparer l'exploration et l'apprentissage de l'environnement. Ils peuvent ainsi considérer que les actions sont des exemples d'apprentissage fournis directement en entrée d'un système d'apprentissage de modèle d'action. Parmi ceux-là,

certains supposent que les exemples sont fournis par des experts des domaines. Ainsi, seules des actions menant à des buts définis sont prises en compte. D'autres travaux supposent que la perception des états est correcte, mais incomplète. Dans ce cas, tous les faits décrivant un état du monde à un instant donné ne sont pas forcément connus de l'agent.

D'autres approches encore, plus générales, reposent sur des simulateurs fournissant des ensembles d'exemples d'apprentissage. Certains prennent en compte toutes les actions possibles dans l'environnement, y compris les actions ayant échoué.

Dans la section suivante, nous présentons différentes approches existantes ainsi que leurs hypothèses d'apprentissage.

3.2.1. Systèmes existants

Parmi les premiers, le système LIVE (Shen, 1993) est capable d'apprendre un modèle d'action déterministe en explorant en ligne son environnement, procédant par spécialisations successives de son modèle sans toutefois pouvoir revenir sur un mauvais choix de spécialisation, ce qui le rend inapplicable sur des environnements complexes.

Le système EXPO (Gil, 1994) est capable de raffiner par expérimentation des opérateurs de planification déterministe. Cependant, il ne peut apprendre un modèle d'action uniquement par l'expérimentation et doit partir d'un modèle sur-général (correct mais incomplet) pour le spécialiser. Si le modèle en entrée est incorrect, il ne sera pas en mesure de le corriger.

Le système OBSERVER (Wang, 1995) est plus expressif. Il apprend un modèle d'action permettant d'exprimer pour une même action des effets différents en fonction de préconditions différentes. Il n'est cependant pas autonome : son apprentissage s'effectue à partir de traces d'exécutions de plans fournies par un expert.

Le système TRAIL (Benson, 1995) peut apprendre un modèle d'action en présence de bruit. Il nécessite cependant l'intervention d'un oracle extérieur en cas d'erreur de

planification et n'est pas incrémental. Il utilise l'algorithme Golem (Muggleton et Feng, 1990) pour apprendre son modèle d'action.

Le système ARMS (Yang *et al.*, 2005, 2007) est capable d'apprendre à partir de traces incomplètes d'exécutions de plans fournis par un expert. Les actions issues de ces traces sont néanmoins toutes correctes et aboutissent à des effets. Le système LAMP (Zhuo *et al.*, 2010) étend les travaux de Yang à des extensions du langage STRIPS mais demeure hors-ligne.

De même, SLAF (Shahaf et Amir, 2006; Amir et Chang, 2008) est aussi capable d'apprendre à partir de traces incomplètes et en ligne. Cependant, plus l'agent rencontre d'exemples, et plus le temps nécessaire à l'apprentissage augmente de façon linéaire et ceux même si le taux de prévision du modèle a atteint 100%.

Les travaux d'Otero (2005) abordent l'apprentissage des modèles d'action relationnels riches en traitant le "frame problem" en représentant explicitement ce qui change et ce qui demeure identique après une action. Il utilise des algorithmes de généralisation hors ligne issus de l'ILP et ne traite pas le problème de l'exploration de l'environnement.

Les travaux de Pasula *et al.* (2007) se sont intéressés à l'apprentissage d'un modèle d'action aux effets stochastiques (l'exécution d'une action dans un état n'engendre pas toujours les mêmes effets). Le modèle appris est capable de représenter la distribution des différents effets les plus probables. Il étend un modèle STRIPS classique à l'aide de références déictiques permettant de faire référence dans les règles à des objets n'intervenant pas directement dans les paramètres de l'action (nous détaillons ce point section 4.1.2). Cependant, l'apprentissage de ce modèle n'est pas incrémental. Il est effectué à partir de traces générées aléatoirement par un agent évoluant dans un environnement en 3 dimensions qui simule une gravité réaliste permettant de reproduire des chutes de bloc mal empilés par des actions potentiellement approximatives d'un bras robotisé.

Les travaux de Mourão *et al.* (2010) se focalisent sur l'apprentissage de modèles d'action dans des environnements partiellement observables et en présence d'un léger taux de bruit (5%). Les observations de l'environnement sont traduites sous forme de vecteur

numérique. Le système d'apprentissage repose sur des noyaux et des perceptrons multi-classe (Freund et Schapire, 1999). La méthode n'est pas immédiatement exploitable car elle n'apprend pas un ensemble de règles directement utilisable par un planificateur. Néanmoins, Mourão *et al.* (2012) développe un processus postérieur capable de transcrire le modèle en règles mais au prix d'une perte d'expressivité : les règles ne peuvent exprimer des effets identiques pou des préconditions différentes alors que le modèle appris est capable de les prédire. La méthode apprend à partir d'actions ayant réussi ou échoué de façon incrémentale. La taille du modèle ne cesse cependant d'augmenter avec la rencontre d'exemple d'apprentissage. Le nombre de vecteurs de support croît avec le nombre de transitions rencontrées et l'interrogation du modèle est proportionnellement plus longue.

3.2.2. STRIPS et aspects théoriques

Différents travaux on traités de l'apprentissage de modèle d'action sous un angle expérimental, néanmoins à notre connaissance très peu se sont portés sur son aspect théorique.

Shahaf et Amir (2006) calculent des bornes en terme de temps de calcul et d'espace d'ordre polynomial pour leur méthode d'apprentissage de modèle d'action. Walsh et Littman (2008) sont cependant les seuls à notre connaissance, à donner des bornes en terme d'erreurs nécessaires pour apprendre un modèle d'action STRIPS à partir de l'exploration de l'environnement et non pas uniquement à partir de traces fournies par des experts.

L'étude porte sur des modèles STRIPS standards. Les pré-conditions sont uniquement des conjonctions de littéraux positifs. Les seules variables autorisées au sein des règles doivent faire partie des paramètres des actions. Il n'est donc pas possible de représenter des situations complexes où des préconditions différentes peuvent aboutir à des effets similaires.

On suppose avoir un planificateur complet à disposition (pour tout but, lorsqu'un plan existe, il le trouve), le but étant d'obtenir des plans corrects : leur exécution mène de

l'état initial au but.

Plus précisément, pour tout couple (état initial, but) donné, l'objectif est d'obtenir à chaque fois, soit un plan correct s'il existe, ou sinon un échec, à partir d'un planificateur complet et du modèle d'action appris. Tout autre comportement est considéré comme une erreur. Dans ce cadre d'apprentissage, c'est le nombre d'erreurs pour parvenir à ce modèle d'action qui est borné, ce qui représente une borne en terme d'exploration de l'environnement.

Le problème est divisé en deux sous-problèmes : le premier consiste à apprendre les effets en supposant que les pré-conditions des actions sont connues, le deuxième étant le complémentaire, il consiste à apprendre les pré-conditions en supposant que les effets sont connus.

Dans un premier temps, les effets sont supposés inconnus et les pré-conditions des règles sont supposées données. Ainsi, avec les préconditions connues, une action a à partir d'un état s appartenant à un plan ne peut échouer.

Initialement pour chaque action, tous les effets sont marqués comme inconnus. A chaque fois qu'une action est exécutée, les effets rencontrés sont mis à jour et marqués comme vrai. Le processus est répété à chaque action. Une erreur se produit lorsqu'au moins un atome des effets (*add* ou *del*) est mis à jour. Les effets sont initialement inconnus, et dans le pire des cas, une erreur ne porte que sur un atome. Pour connaître le nombre d'erreurs, il faut donc énumérer tous les atomes différents possibles, ce qui peut considérablement varier en fonction des environnements traités.

Un prédicat (utilisé dans les descriptions des états) a une arité maximum n et il y a au maximum m variables (arité des prédicats d'action, car dans STRIPS standard les seules variables autorisées aussi bien dans les pré-conditions que dans les effets doivent faire partie des arguments de l'action), il y a donc m^n atomes possibles.

Par exemple, avec *on*/2 et 3 variables A,B,C :
on(A,B),on(A,C),on(B,A),on(B,C),on(C,A),on(C,B),on(A,A),on(B,B),on(C,C), soit 3^2.

Comme il y a P prédicats et A actions(règles), le nombre d'erreurs est en $O(APm^n)$.

Dans un deuxième temps, on suppose cette fois les préconditions inconnues et les effets donnés. Dans ce cas, le problème est de nature différente, dans la mesure où, précédemment, une erreur commise sur les effets était identifiable en les comparant aux effets antérieurement observés. Ici, lors de l'exécution d'une action, soit celle-ci réussit, soit on est informé d'une erreur car des préconditions ne sont pas vérifiées. Cependant, dans ce cas, on ne sait pas quels atomes sont en cause. En l'absence d'information supplémentaire, il est nécessaire, afin de borner l'erreur, d'énumérer toutes les préconditions possibles. Néanmoins, en restreignant les préconditions à des conjonctions de k atomes, on obtient un nombre d'erreurs borné par $O(A(Pm^n)^k)$.

Ces deux sous-ensembles du problème initial sont exclusifs, si une action réussit les préconditions sont donc correctes. Il est alors possible de commettre au moins une erreur sur les effets en $O(APm^n)$. A l'inverse, si une action échoue (on n'observe aucun effet), l'erreur porte sur les préconditions en $O(A(Pm^n)^k)$. Finalement, pour apprendre à la fois les préconditions et les effets, le nombre d'erreurs est donc borné par $O(A(Pm^n)^{max(k,1)})$.

Les travaux de Walsh et Littman (2008) permettent de fournir des bornes à l'apprentissage de modèles d'action en terme d'exploration nécessaire de l'environnement. Cependant, ces travaux se limitent aux modèles d'action STRIPS standard, dont l'expressivité est limitée.

3.2.3. Travaux proches

A notre connaissance, les travaux de thèse de Croonenborghs (2009) font partie des seules études à apprendre un modèle d'action relationnel en explorant directement l'environnement et en calculant un modèle d'action relationnel de façon incrémentale (le modèle continue de s'enrichir en fonction des exemples rencontrés sans toutefois croître indéfiniment car la taille du modèle est bornée). Ces travaux ne s'inscrivent pas dans un cadre de planification mais dans un cadre d'apprentissage par renforcement relationnel ou RRL (Van Otterlo, 2008). On se propose de les étudier plus en détail

car ils s'intéressent à l'apprentissage d'un modèle d'action ainsi qu'à son utilisation en parallèle.

Nous commençons par introduire le cadre RRL à travers sa première implémentation pour exposer ensuite les travaux de Croonenborghs (2009) sur l'apprentissage d'un modèle d'action dans ce même cadre.

Apprentissage par Renforcement Relationnel

Le cadre du RRL a été introduit pour la première fois par Dzeroski *et al.* (1998). L'idée originale était de combiner l'apprentissage par renforcement et l'apprentissage dans des langages de la logique du premier ordre (Muggleton, 1991).

Cette première approche reprend l'algorithme de renforcement du Q-learning de Watkins et Dayan (1992), mais en exploitant des représentations relationnelles pour les états et les actions.

Algorithme 3 Q-learning avec mise à jour Bucket-Brigade

Initialiser $Q(s, a)$ arbitrairement (pour tout $s \in \mathcal{S}$ et $a \in \mathcal{A}(s)$)
répéter
  $/*$ *pour chaque épisode* $*/$
  générer un état initial s
  $i \leftarrow 0$
  répéter
    $/*$ *pour chaque pas de l'épisode* $*/$
    sélectionner une action a_i et l'exécuter
    réception d'une récompense immédiate r_i
    observation d'un nouvel état s_{t+1}
    $i \leftarrow i + 1$
  jusqu'à s_i soit terminal
  pour $j = i - 1$ à 0 **faire**
    $Q(s_j, a_j) \leftarrow r_j + \gamma max_{a'} Q(s_j + 1, a')$
  fin pour
jusqu'à plus d'épisodes

L'algorithme 3 illustre la boucle principale de Q-learning utilisée par Dzeroski *et al.* (1998). L'agent interagit avec son environnement en sélectionnant et exécutant une

action sur celui-ci ($l.8$). L'agent reçoit de l'environnement une récompense associée ($l.9$) et le nouvel état dans lequel il se trouve à présent ($l.10$). L'interaction est répétée jusqu'à ce que l'agent arrive à la fin d'un épisode. Soit parcequ'un état particulier est atteint, soit après un nombre défini d'interactions maximum.

La *Q-fonction* utilisée par le Q-learning ($l.14$) permet d'approximer la fonction *Qualité* optimale de façon incrémentale à l'aide des récompenses et des estimations courantes de la fonction *Qualité*. Dans cette version avec mise à jour dite « Bucket-Brigade », la mise à jour de la *Q-fonction* ne se fait pas après chaque action, mais à la fin de chaque épisode. Cela permet d'augmenter la vitesse de convergence de l'algorithme, car, par exemple, si un but est atteint (e.g. avec une forte récompense associée), sa valeur est rétro-propagée directement aux autres couples (état, action) appartenant à l'épisode.

Jusqu'ici, nous avons implicitement supposé qu'une valeur était stockée pour chaque couple (état, action). Avec une telle représentation non structurée ou naïve, un tableau suffit. Néanmoins, dans la pratique, avec des espaces d'états et/ou d'actions de grande taille, une représentation tabulaire pose le problème de la taille de mémoire nécessaire au stockage des valeurs de chaque couple (état, action) possible et aussi du temps nécessaire pour y accéder. L'absence de structure impose également de rencontrer chaque couple un grand nombre de fois avant de converger vers une fonction Qualité optimale (Jaakkola *et al.*, 1994).

La solution utilisée par la méthode RRL consiste à généraliser la fonction Qualité en essayant d'identifier la structure relationnelle des problèmes traités. Il est ainsi possible de prédire des valeurs approximant la fonction Qualité même pour des situations encore jamais rencontrées.

Pour généraliser la Q-fonction, l'algorithme RRL utilise l'arbre de décision relationnel, Tilde (Top-down Induction of Logic Decision Trees), (Blockeel *et al.*, 1998) dans sa variante pour la régression (Tilde-RT).

Tilde est une adaptation de l'arbre de décision propositionnel C4.5 (Quinlan, 1993) à une représentation relationnelle. Il est guidé par les données et supervisé.

A partir d'un ensemble d'exemples d'apprentissage, C4.5 utilise une approche « diviser pour mieux régner ». Dans le cadre d'une classification binaire, les exemples étant étiquetés positivement ou négativement, C4.5 essaye de séparer au mieux les exemples de ces deux classes en deux sous-ensembles distincts. Pour ce faire, l'algorithme cherche le meilleur attribut dont le test sur les exemples est positif sur le maximum d'exemples d'une classe (gain d'information).

Pour chaque sous-ensemble obtenu, le processus est répété récursivement jusqu'à ce que tous les exemples d'un sous-ensemble appartiennent à la même classe ou bien qu'une certaine profondeur dans l'arbre soit atteinte.

Dans un cadre de régression, comme c'est le cas pour l'approximation de la fonction Qualité, le comportement demeure fondamentalement le même. Le choix d'un attribut se fait sur sa capacité à minimiser la variance (F-test) de la variable cible (ici la Q fonction) sur les deux sous-ensembles obtenus. Autrement, si aucun attribut ne peut obtenir un F-test au dessus d'un certain seuil, alors la valeur de la fonction Qualité pour cet ensemble est égale à la moyenne des Qualités de tous les exemples de l'ensemble.

En ordre un, les tests pour raffiner l'arbre sont plus complexes. Ils ne portent pas seulement sur un attribut, mais peuvent porter sur des objets et des relations entre ceux-ci. Les tests sont des littéraux ou conjonction de littéraux et peuvent contenir des variables.

Le nombre de tests possibles ou espace des hypothèses est considérablement plus grand. Dans la pratique, afin de réduire la taille de cet espace, il est nécessaire d'écarter manuellement les hypothèses improbables à partir des connaissances du domaine.

Pour cela on utilise les types et modes. Les types permettent de restreindre les arguments possibles des prédicats. Par exemple :

$$type(clear(block))$$

Ce type interdit par exemple que la table qui n'est pas de type bloc soit libre.

Les modes permettent de restreindre l'espace de recherche en indiquant explicitement de quels littéraux peuvent être composés les tests et aussi comment les littéraux peuvent être utilisés au sein des tests. Par exemple :

$$rmode(on(+-X, floor))$$

Ce mode indique que l'on peut tester s'il existe un objet X sur la table. Devant une variable, les options (+) et (-) permettent d'imposer respectivement que la variable soit instanciée ou libre (non-instanciée). Dans cet exemple, les deux alternatives sont possibles.

C4.5 ou son adaptation à l'ordre un nécessitent d'avoir à disposition un ensemble d'exemples d'apprentissage afin de pouvoir inférer un arbre. Dans le cadre de l'approximation de la fonction Qualité, les exemples d'apprentissage (état, action, valeur) se présentent en séquence. L'utilisation et la construction de l'arbre sont alternées. Afin de pouvoir utiliser Tilde dans le cadre d'un apprentissage en ligne, les exemples d'apprentissage sont présentés à Tilde à chaque fin d'épisode. De plus, tous les couples (état, action) ainsi que leur valeur associée la plus récente sont conservés. Un nouvel arbre remplaçant le précédent est ainsi construit après chaque nouvel épisode en prenant en compte tous les précédents couples (état,action) rencontrés.

Pour éviter de conserver tous les exemples en mémoire et de réapprendre un nouvel arbre après chaque épisode, Driessens *et al.* (2001) ont proposé l'algorithme TG. TG est utilisé de la même façon que Tilde-RT dans une boucle de Q-learning. Il s'agit également d'un arbre de régression. Cependant, il ne nécessite pas de garder tous les exemples rencontrés et s'adapte progressivement après chaque exemple.

TG est une adaptation à l'ordre un de l'algorithme G (Chapman et Kaelbling, 1991). Tout comme Tilde-RT, il s'agit d'un arbre de régression guidé par les données et supervisé, mis à part qu'il est incrémental.

En outre, TG est similaire à Tilde-RT sur les tests utilisés au sein des noeuds de l'arbre et utilise les mêmes biais de langage. La différence fondamentale entre les

deux approches réside dans la gestion des exemples d'apprentissage. TG ne nécessite de garder aucun exemple en mémoire. Le processus de raffinement de l'arbre repose uniquement sur les statistiques de réussite des différents tests possibles sur les exemples présentés au fur et à mesure.

Ainsi, lorsqu'un exemple se présente, à partir du noeud racine, en fonction du résultat au test effectué, le processus se répète récursivement sur le sous-noeud gauche si le test a réussi ou sur le sous-noeud droit dans le cas contraire. Le processus s'arrête lorsqu'une feuille est rencontrée.

Au niveau des feuilles, pour chaque test encore possible, TG les évalue sur le nouvel exemple et maintient des statistiques associées à la réussite ou non des tests, ainsi que les statistiques associées à chaque exemple. A l'aide de ces données et d'un F-test, TG peut sélectionner un test qui minimisera la variance au sein des deux nouvelles feuilles obtenues par rapport à la variance de l'unique feuille antérieure. Si aucun test ne le permet, alors la structure de l'arbre demeure inchangée. En prédiction, chaque feuille contient la moyenne des valeurs des exemples qui vérifie les tests qui relient à la racine.

Avec une représentation attribut/valeur, il est possible de garder les statistiques des tests sur chacun des attributs de façon indépendante ce qui permet de restructurer l'arbre en fonction de nouveaux exemples rencontrés comme dans l'algorithme ITI (Utgoff *et al.*, 1996).

Cependant, en ordre un, les tests effectués de la racine jusqu'à une feuille ajoutent à chaque noeud des dépendances entre eux. Par exemple, figure 3.1, l'instanciation de la variable A au noeud racine doit être prise en compte dans tous les noeuds descendants. Cela rend difficile la restructuration de l'arbre. L'algorithme TG ne le permettant pas, le déroulement de l'apprentissage est alors fortement sensible aux choix des premiers tests. Pour réduire les conséquences de cette limitation, TG utilise un seuil minimum d'exemples rencontrés avant de sélectionner un test.

Grâce aux statistiques des tests accumulés par TG, il n'est pas nécessaire de conserver en mémoire tous les exemples d'apprentissage rencontrés. Cette caractéristique principale

permet un usage efficace en ligne du modèle notamment en temps d'apprentissage. Le choix d'un test plutôt q'un autre peut-être influencé par une séquence d'exemples plutôt qu'une autre. De plus, la méthode est exclusivement récursive : il n'est pas possible de revenir sur le choix d'un test. Pour résoudre ce problème, différentes approches ont été avancées, qui permettent une restructuration de l'arbre comme le système TGR (Ramon *et al.*, 2007) et Relational UTrees (Dabney et McGovern, 2007).

Nous avons introduit le cadre RRL qui permet l'apprentissage d'une fonction Qualité avec une représentation relationnelle de l'environnement. Nous présentons ci-après l'apprentissage d'un modèle d'action dans ce même cadre.

RRL et Apprentissage de modèle d'action

Dans les travaux présentés à la section précédente, l'effort se focalise sur l'apprentissage de la fonction Qualité. La fonction de transition est supposée connue par l'agent uniquement de façon locale. L'agent, dans son état courant, peut interroger l'environnement afin de savoir quelles actions sont alors possibles et en choisir une en fonction de sa politique. Cependant, la fonction de transition qui, pour chaque état et action associés renvoie le nouvel état résultant, n'est pas connue de l'agent (on parle alors de cadre Model free). Or, la connaissance de cette fonction (cadre Model Based) permet par exemple à l'agent de propager directement des récompenses reçues dans un état aux états environnants sans nécessiter leur visite. Cette connaissance permet de diminuer le nombre d'interaction nécessaires entre l'agent et son environnement et par conséquent d'apprendre des politiques plus rapidement. Nous nous intéressons donc non pas au cas où la fonction de transition(ou modèle d'action) est donnée, mais à l'apprentissage de celle-ci dans le cadre RRL.

Croonenborghs *et al.* (2007) développent le système *MARLIE* (Model-Assisted Reinforcement Learning in Expressive Languages) qui apprend un modèle d'action en ligne uniquement à partir de sa propre expérience. En parallèle de l'apprentissage d'un modèle d'action, *MARLIE* évolue avant tout dans un contexte d'apprentissage par renforcement et a donc pour but de maximiser le cumul des récompenses associé aux états[1]. Ce

1. ici, tous les états différents du but sont associés à des récompenses nulles.

système utilise son modèle d'action (même partiel) pour améliorer l'apprentissage d'une politique. Ces travaux débouchent sur plusieurs résultats significatifs. Ils montrent expérimentalement que l'apprentissage et l'exploitation en ligne d'un modèle d'action relationnel, même intermédiaire, est possible et peut considérablement améliorer la qualité de la stratégie de l'agent. Autrement dit, ils montrent qu'en exploitant le modèle d'action appris, leur méthode nécessite d'explorer moins d'exemples. Nous étudions plus en détails le compromis exploration/exploitation au Chapitre 5.

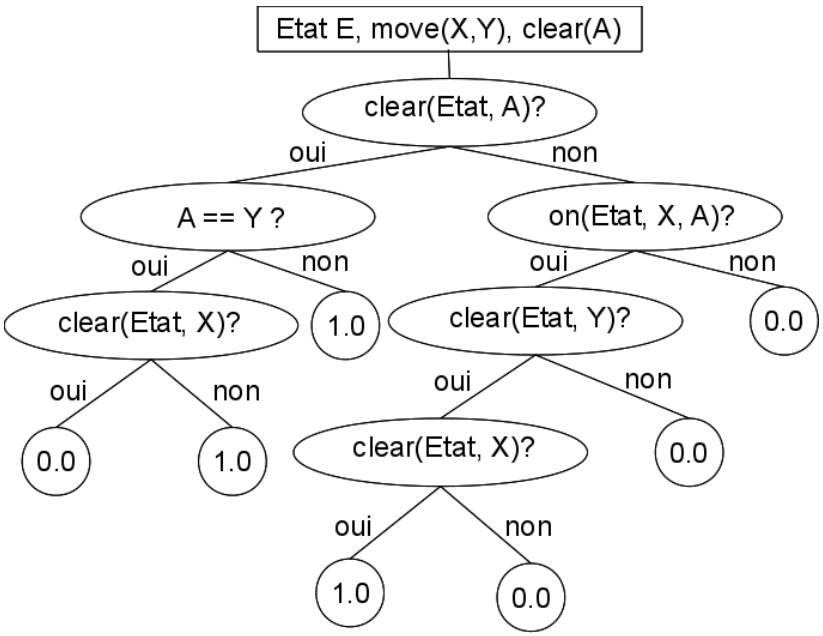


FIGURE 3.1.: Arbre de régression ($clear(X)$ est vrai s'il n'existe aucun bloc au dessus du bloc X).

Concrètement, le modèle se compose d'un ensemble d'arbres de régression inférés par l'algorithme TG décrit à la section précédente. Il est nécessaire d'apprendre un arbre pour chaque prédicat de l'environnement. Un tel arbre permet de répondre à la question suivante : étant donnés un état et une action, quelle est la probabilité qu'un atome clos donné soit vrai dans le nouvel état après exécution de l'action ? La figure 3.1 extraite de (Croonenborghs *et al.*, 2007) présente un tel arbre pour le prédicat $clear/1$ du monde de blocs. Chaque noeud représente une variable ou test binaire, les feuilles représentent la probabilité de présence de l'atome à l'état suivant en fonction des différents tests binaires. Ainsi, on peut tester cette partie du modèle sur la transition illustrée à la

Figure 4.1. On suppose être dans l'état s_t et choisir l'action a_t : $move(d, c)$. Détaillons par exemple l'opération pour l'objet d. On désire savoir si $clear(d)$ sera vrai après l'action. Les variables X,Y et A du modèle sont respectivement instanciées à d,c et d. On évalue le premier noeud à la racine de l'arbre, qui consiste à tester si $clear(d)$ est vrai dans s_t. C'est le cas, on continue donc sur la branche gauche du noeud. On teste alors si d est égale à c. C'est faux, on continue donc sur la branche droite du noeud. Il s'agit d'une feuille de valeur 1.0, ce qui signifie que l'atome $clear(d)$ est prévu comme vrai dans s_{t+1}.

La répétition de la procédure pour chaque objet du monde permettra de savoir lesquels auront la propriété *clear* dans l'état suivant. De façon générale, afin de construire une prévision complète sur l'état suivant, ce mécanisme est appliqué sur toutes les différentes instanciations des prédicats du monde. Il est à noter que dans le cas de prédicats d'arité supérieure à 1, il faut tester toutes les combinaisons possibles des arguments sur leur domaine. Cela revient à tester tous les atomes clos possibles. Par exemple, dans le monde de blocs, pour le prédicat $on/2$: $on(a, b), on(a, c), ..., on(b, a)...$ Le modèle d'action n'est pas explicitement représenté, son interrogation est donc un processus lourd empêchant l'utilisation directe du modèle par un planificateur.

La généralisation au sein du système *MARLIE* aussi bien pour la fonction de transition que pour la fonction Qualité repose sur l'arbre de régression incrémental relationnel TG (Driessens *et al.*, 2001) possédant certaines limitations. En effet, il s'agit d'une approche exclusivement descendante, qui spécialise son hypothèse au fur et à mesure des exemples sans permettre aucune restructuration de l'arbre induit. Le partitionnement de l'espace ainsi obtenu est, par conséquent, fortement influencé par les premiers exemples d'apprentissage rencontrés et peut engendrer un arbre sur-spécialisé, ce quie le rend difficilement utilisable sur des environnements complexes. Dans la pratique, pour palier ce problème, la profondeur de l'arbre est fixée *a priori*, ce qui permet de limiter la taille du modèle.

3.2.4. Révision de théorie

L'apprentissage de notre modèle d'action sous la forme d'un ensemble de règles s'apparente au domaine de la révision de théorie en ordre un dans la mesure où les règles induites par l'expérience devront être affinées lors de l'apprentissage. Néanmoins, la révision de théorie part classiquement d'un modèle incorrect et/ou incomplet et donné *a priori*, ce qui n'est pas le cas dans notre cadre d'apprentissage où le modèle est initialement vide. Ici, c'est uniquement l'expérience du système qui permettra de créer et d'améliorer le modèle. Il y a peu de systèmes de révision de théorie en programmation logique inductive (Richards et Mooney, 1995) et à notre connaissance pas dans le contexte de l'apprentissage d'un modèle d'action. Le système le plus proche du nôtre est INTHELEX de Esposito *et al.* (2000). INTHELEX opère aussi sous OI-subsomption et utilise également un opérateur de généralisation (par moindres généralisés) guidé par les données. La principale différence avec IRALe concerne l'opérateur de spécialisation : INTHELEX spécialise des clauses en leur ajoutant des littéraux pour rejeter des exemples négatifs, et doit faire face au choix difficile de sélectionner le meilleur littéral à ajouter lorsqu'il y a plusieurs littéraux candidats. De plus, il conserve tous les exemples rencontrés en mémoire.

Notre algorithme de spécialisation est plus simple dans la mesure où il ne spécialise pas directement, mais revient sur les généralisations antérieures jusqu'à en trouver une cohérente.

Chapitre 4

IRALe

Sommaire

Dans ce chapitre, nous abordons l'apprentissage d'un modèle d'action au sein d'environnements relationnels et déterministes. Nous commençons par décrire en détails les modalités d'interaction entre un agent apprenant et son environnement. Nous continuons en introduisant les différentes définitions permettant de tester les règles du modèle sur les exemples d'apprentissage. Ensuite, nous introduisons notre contribution sous la forme de l'approche IRALe (Rodrigues *et al.*, 2010a) (Incremental Relational Action Learning).

Afin d'argumenter les différents choix de notre approche, nous présentons dans un premier temps l'approche dans le cadre d'une représentation propositionnelle rendant plus aisée le développement des principaux mécanismes. Dans un second temps, nous abordons les problèmes supplémentaires induits par l'utilisation d'une représentation relationnelle ainsi que les solutions que nous proposons. Nous étudions enfin les propriétés de convergence de la méthode et clorons ce chapitre par une étude empirique d'IRALe, en détaillant les expériences réalisées ainsi que les résultats obtenus.

4.1. Interactions entre l'agent et son environnement

On place l'apprentissage de la fonction de transition (cf. equation 2.1, section 2.1) ou modèle d'action dans un cadre d'exploration. Seule l'action de l'agent sur son environnement est source d'exemples d'apprentissage. Dans cette section, nous commençons par introduire la forme et la nature des exemples d'apprentissage, puis nous définissons les propriétés d'un modèle d'action capable de les décrire.

4.1.1. Répresentation des exemples d'apprentissage

L'agent a une perception complète de l'état courant. L'environnement informe l'agent sur l'état dans lequel il se trouve après chaque action et ceci sans erreur ni omission. Une action est le fait de produire un effet sur quelque chose. Sa réalisation est dépendante d'un contexte ou de l'état à partir duquel elle a lieu. Un exemple d'action doit rendre compte des conditions et des effets de celle-ci sur son environnement.

Formellement, un exemple x se compose de trois ensembles pouvant se représenter de la façon suivante :

1. un ensemble $x.s$ décrivant l'*état* dans lequel se trouve le système (ce qui est vrai avant l'action)
2. un ensemble $x.a$ décrivant une *action* effectuée dans cet état

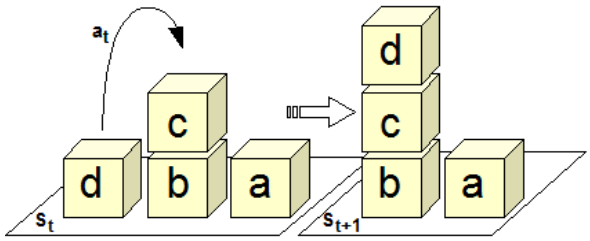


FIGURE 4.1.: Exemple de transition

3. un ensemble $x.e$ décrivant les *effets* de l'action sur l'état (ce qui change après l'action), il est composé de deux sous-ensembles :
   a) un ensemble *add* décrivant les atomes apparaissant après application de l'action (devenant vrais)
   b) un ensemble *del* décrivant les atomes disparaissant après application de l'action (devenant faux)

Les littéraux non affectés par l'action ne sont pas décrits dans la partie effet. Ainsi, lorsqu'un fait est vérifié dans l'état avant l'action, et aussi après celle-ci, alors il n'est pas représenté dans les effets de l'exemple. Seuls les changements sont observés et pris en compte. On note les exemples : $x.s/x.a/x.e.add, x.e.del$. Sous cette forme, l'exemple de la figure 4.1 se note alors :
$clear(d), on(d, floor), clear(c), on(c, b), on(b, floor), clear(a), on(a, floor)/$
$move(d, c)/$
$on(d, c), \neg clear(c), \neg on(d, floor).$

L'agent se trouve dans un état s_t et émet une action a_t, l'environnement l'informe du nouvel état s_{t+1} résultant. L'opération est répétée jusqu'à parvenir à un état particulier ou but.

Définition 7 (action légale) *L'action a_t est légale si elle a un effet, c'est-à-dire si $s_{t+1} \neq s_t$.*

Définition 8 (action illégale) *Si après une action a_t, $s_{t+1} = s_t$, l'action est dite illégale ou sans effet.*

La figure 4.1 montre un exemple d'action légale. A partir de l'état s_t, on effectue le déplacement du bloc d sur le bloc c. Les autres déplacements légaux à partir de s_t sont : $move(d, a), move(c, d), move(c, a), move(c, table), move(a, c), move(a, d)$. Par contre, par exemple l'action $move(b, c)$ n'est pas légale car c est sur b.

L'agent découvre les objets du monde en observant les états qu'il rencontre. Il démarre son apprentissage avec uniquement les différentes actions syntaxiquement possibles représentées par des symboles de prédicats avec leurs arités respectives ainsi que les différents types d'objets en paramètre. Choisir au hasard une action sans connaissance supplémentaire revient à choisir aléatoirement pour chaque paramètre de l'action, un objet parmi ceux des types possibles. Par exemple, pour le monde de blocs illustré figure 4.1 composé de 5 objets (4 blocs et une table), l'agent est capable de saisir des blocs et de les poser sur d'autres blocs ou sur la table. Dans cet état, l'action $move(type : bloc, type : objet)$ possède 20 instanciations différentes possibles correspondant à toutes les permutations possibles de blocs et d'objets (bloc et table). Il s'agit alors d'actions syntaxiquement correctes. Néanmoins, parmi ces actions, toutes ne sont pas permises par l'environnement. Ainsi sur l'exemple de la figure 4.1, parmi les 20 actions syntaxiquement correctes, seules 7 sont légales.

Ce point précis sur la sélection de l'action permet de souligner une distinction significative de notre travail avec la plupart des travaux proches. En effet, différents travaux comme ceux de Wang (1995), Amir et Chang (2008) ou encore de Yang *et al.* (2007) utilisent comme source d'apprentissage des traces fournies par des experts. Autrement dit, ils utilisent uniquement des exemples d'actions légales menant à des buts particuliers, ce qui simplifie le problème.

Par opposition, dans notre étude, nous nous plaçons dans le même cadre que l'algorithme MARLIE de Croonenborghs *et al.* (2007) ou encore celui de Walsh et Littman (2008), qui dans les deux cas permettent à l'agent d'exécuter des actions qui peuvent être sans effets. Cette approche est plus générale et permet d'éviter l'introduction de connaissances du domaine en posant le problème de la découverte des actions légales. Cependant, l'apprentissage des actions légales est subséquemment ralenti par l'exécution d'actions sans effets, ne serait-ce que par leur fréquence dans le cadre d'une exploration exclusivement aléatoire, comme effectué dans nos expériences (cf. section 4.5). De plus,

pour ces actions, l'agent doit également apprendre à prédire des effets vides. En résumé, les seules connaissances données à notre agent préalablement à toute expérience se résume aux prédicats d'actions et aux domaines d'objets sur lesquels elles peuvent porter. Aucune autre connaissance sur les états n'est donnée. Seule l'expérience permet à l'agent d'observer et de découvrir les relations et les objets du monde pour en apprendre les règles d'action, dont les conditions et les effets sont également à découvrir.

L'agent est ainsi immergé dans son environnement et doit en apprendre les lois afin de modéliser ce qu'il est permis de faire sous certaines conditions, ainsi que les conséquences de ses actions. Pour ce faire, l'agent ne peut interroger son environnement librement et demeure localement contraint. Pour l'agent, l'environnement est une boîte noire qui l'informe uniquement de l'état dans lequel il se trouve. L'agent est alors contraint d'agir à partir de cet état pour constituer des exemples d'apprentissage à partir de son expérience. Hormis dans le chapitre 5 sur l'apprentissage actif, on suppose dans ce qui suit, qu'IRALe suit une politique aléatoire.

Après avoir décrit les interactions entre l'agent et son environnement, nous définissons dans la section suivante l'expressivité du modèle d'action.

4.1.2. Expressivité du modèle d'action

Il est d'usage, avec un langage STRIPS (cf. section 3.1), notamment dans les domaines de la conférence internationale de planification (IPC), que toutes les variables d'une règle doivent appartenir aux arguments de l'action.

Avec cette contrainte, une fois l'action instanciée, tous les objets qui en sont affectés sont fixés, ce qui permet de simplifier le test de couverture d'un état par une règle (que l'on définit à la section 4.3.1). Néanmoins, cela oblige à introduire artificiellement dans le prédicat d'action des variables indirectement impactées par l'action, ce qui reporte la complexité d'instanciation sur le prédicat d'action.
Ainsi, sous cette notation, l'exemple 1 ci-après, décrit le déplacement d'un bloc X d'une pile au dessus d'un bloc Y (tous deux libres : cl). Comme d'autres variables W et Z interviennent dans l'action, celles-ci sont donc rajoutées au prédicat d'action.

Exemple 1 $r_1 : cl(X), cl(Y), on(X,Z), on(Z,T), on(Y,W)/move(X,Y,W,Z,T)/$
$on(X,Y), cl(Z), \neg on(X,Z), \neg cl(Y)$

Une des conséquences de ce choix est qu'une même action dans des conditions différentes nécessitant un nombre de variables distinct engendrera deux règles distinctes avec des actions d'arité différente. L'exemple 2 suivant, illustre la même action *move* dans une situation différente et avec une arité différente. Cet exemple décrit le déplacement d'un bloc X sur un bloc Y (tous deux libres : cl) alors que X, contrairement à l'exemple précédent, se trouve sur la table.

Exemple 2 $r_2 : cl(X), cl(Y), on(X,floor), on(Y,W)/move(X,Y,W)/$
$on(X,Y), \neg on(X,floor), \neg cl(Y)$

Une telle représentation ne fonctionne que lorsqu'on se place dans un cadre de planification, avec un modèle d'action supposé donné et directement utilisé par le planificateur. Mais qu'en est-il dans le cas d'une découverte automatique du modèle d'action ? L'apprentissage du modèle d'action se fait à partir des actions de l'agent sur son environnement, dans ce cas, il est difficile de supposer que l'arité des actions est préalablement définie par toutes les variables nécessaires à la description des règles que l'on cherche à apprendre. Il est alors plus naturel d'envisager que les actions sont décrites (appelées) uniquement sur les objets directement affectés par l'action et qu'il est nécessaire de chercher uniquement par l'expérience les objets reliés à l'action.

Ainsi, la règle de l'exemple 1 s'écrit alors :

Exemple 3 $r_1 : cl(X), cl(Y), on(X,Z), on(Z,T), on(Y,W)/move(X,Y)/$
$on(X,Y), cl(Z), \neg on(X,Z), \neg cl(Y)$

Les travaux de Pasula *et al.* (2007) avec les "Deictic reference", aussi utilisées par Mourão *et al.* (2012) permettent aussi d'introduire de nouvelles variables en dehors du prédicat d'action. Notre approche permet sur ce point la même expressivité.

Cependant Pasula *et al.* (2007) représente aussi une distribution partielle d'effets stochastiques, ce qui ne rentre pas dans le cadre de nos travaux.

Nous nommons notre formalisme *STRIPS déterministe étendu (*EDS pour Extended Deterministic STRIPS*)*. Il est plus expressif que le STRIPS déterministe utilisé par exemple par Walsh et Littman (2008). Dans *EDS*, l'agent peut apprendre plusieurs règles avec des préconditions différentes pour un même couple (action,effet) donné.

Ceci permet d'exprimer qu'une même action peut avoir des mêmes effets sous des préconditions différentes, ce qui représente alors une disjonction de préconditions. Chacune des préconditions est exprimée par une conjonction de littéraux.

De plus, *EDS* accepte les références « Deictic ». Notons qu'on peut toujours transformer un modèle d'action en *EDS* comme étant un modèle d'action dans STRIPS déterministe de la façon suivante :

- Pour chaque règle EDS r_i telle que $r_i = (r_i.p/r_i.a/r_i.e)$, on définit une nouvelle règle r'_i avec un symbole de prédicat d'action unique A_i telle que $A_i = r_i.a$. On obtient $r'_i = (r'_i.p = r_i.p/r'_i.a = A_i/r'_i.e = r_i.e)$
- Pour chaque règle r'_i, on ajoute aux attributs de l'action $r_i.a$ toutes les autres variables appartenant à $r'_i.p$ et $r'_i.e$.

A partir des règles obtenues, on peut ainsi utiliser un planificateur STRIPS standard.

Exemple 4 *Considérons le modèle d'action EDS suivant :* $M = \{$
$r_1 : r(X), r(Y), on(X,Z)/move(X,Y)/on(X,Y), \neg on(X,Z)$;
$r_2 : r(X), b(Y), on(X,Z)/move(X,Y)/r(Y), \neg b(Y)\}$
où l'effet de déplacer X *sur* Y *réussit si* X *et* Y *sont tous les deux rouges (*r_1*) et autrement change la couleur bleu de* Y *pour le rouge (*r_2*). La transformation résultante vers STRIPS est la suivante :* $M' = \{$
$r'_1 = r(X), r(Y), on(X,Z)/move_1(X,Y,Z)/on(X,Y), \neg on(X,Z)$;
$r'_2 = r(X), b(Y), on(X,Z)/move_2(X,Y,Z)/r(Y), \neg b(Y)\}$

Nous notons nos actions p-actions comme représentant les actions pragmatiques connues par l'agent avant apprentissage, et s-actions les actions STRIPS résultantes. Ce qui signifie qu'apprendre dans EDS consiste à apprendre un ensemble de triplets (*préconditions, p-action, effets*) qui pourraient être traduits en un modèle d'action STRIPS

déterministe. Une traduction préservant les p-actions comme des actions modélisées peut être obtenue en permettant des effets identiques avec des préconditions différentes dans STRIPS. Apprendre un modèle d'action EDS inclut l'apprentissage d'un "schéma" de s-actions STRIPS (avec le nombre de règles et les variables exactes prenant part dans l'action), ce qui représente une tâche difficile pour un expert.

On fixe aussi les propriétés suivantes, que chaque règle d'action du modèle doit vérifier.

Définition 9 (bien-formée) *Une règle r est* bien formée *si :*
i) $r.e.del \subseteq r.p$
ii) $r.e.add \cap r.p = \emptyset$
iii) et $r.p \cup r.a$ doivent être connectés (tout terme, constante ou variable présent dans $r.p$ doit être relié par un chemin de littéraux à un terme du littéral de l'action $r.a$)
iv) Tout terme de $r.e.add$ doit apparaître dans $r.p$ ou dans $r.a$.

i) Pour qu'un atome ne soit plus vérifié après l'action, il doit être vrai avant celle-ci.

ii) De même, pour qu'un atome devienne vrai après l'action, il ne doit pas faire partie des préconditions. Avec i) et ii) on s'assure des effets de l'action.

iii) On suppose que les objets permettant le déroulement d'une action possèdent un lien (direct ou indirect) avec les objets de l'action. Le lien est direct si les objets des préconditions appartiennent aussi à l'action, comme X et Y dans l'exemple 3. Le lien est indirect pour les objets des préconditions n'appartenant pas à l'action, comme Z, W et T de l'exemple 3, mais sont reliés à X et Y par un chemin de littéraux. Par exemple, la variable T est reliée à la variable X de l'action par les littéraux : $on(X, Z), on(Z, T)$. Ainsi, tout terme (variable ou action) des préconditions doit être connecté avec un terme de l'action.

iv) Concernant les effets, comme $r.e.del \subseteq r.p$ et que les préconditions sont connectées à l'action, alors $r.e.del$ est aussi connecté. Cependant, si les termes de $r.e.add$ n'appartiennent ni à $r.p$ ni à $r.a$, alors pour une instanciation donnée des règles dans un état et pour une action donnée, l'instanciation de $r.e.add$ peut ne pas être unique comme dans l'exemple suivant :

Exemple 5 $r_4 : p(X), p(Y)/a(X,Y)/p(Z), \neg p(Y)$

Pour la règle r_4, si l'on se trouve dans l'état $p(x), p(y)$ et qu'on exécute $a(x,y)$, il n'est pas possible d'instancier de façon unique $r_4.e.add = p(Z)$.

L'instanciation de la variable Z n'est pas unique et peut être effectuée par n'importe quel objet non encore apparié de l'état courant. Afin d'éviter tout choix, on impose que toutes les variables de $r.e.add$ doivent apparaître dans $r.a$ ou $r.p$. Elles seront ainsi instanciées de façon unique, une fois $r.a$ et $r.p$ instanciées.

Néanmoins, il est possible d'avoir plusieurs instanciations d'une même règle. Pour l'exemple 2, il y a autant d'instanciations possibles pour X qu'il y a de blocs libres sur la table. La règle de l'exemple 3 est une règle *bien formée*.

Après avoir défini un modèle d'action relationnel comme notre but à atteindre, dans un premier temps, nous nous s'intéressons dans la section suivante aux relations de couverture qu'il existe entre les exemples et un modèle d'action propositionnel ainsi que son apprentissage, pour dans un deuxième temps étendre l'approche au cas d'un modèle d'action relationnel.

4.2. Modèle d'action propositionnel

Afin d'expliciter les différents mécanismes dans l'acquisition d'un modèle d'action EDS et de faciliter leur présentation, nous commençons par définir les différentes notions de couverture entre les exemples et le modèle à l'aide d'une représentation propositionnelle. Ensuite, nous utilisons ces définitions pour l'apprentissage d'un modèle d'action propositionnel.

4.2.1. Définitions

Nous commençons par définir un modèle d'action propositionnel. Soit un ensemble P de variables propositionnelles, $P = \{p_1, ..., p_n\}$. Un état est une assignation de chaque variable p_i $(1 \leqslant i \leqslant n)$ à vrai ou faux. On représente un état comme une conjonction de variables propositionnelles positives. Suivant l'hypothèse du monde clos, toutes les propositions non représentées sont considérées comme fausses. Exemple d'état : $e_1 = p_1, p_3$.

On définit aussi un ensemble fini A d'actions $A = \{a_1, ..., a_m\}$. Une règle r est un triplet composé de pré-conditions, d'une action et d'effets respectivement sous la forme des trois ensembles : $P/A/E$. Soient s et s' deux états avant et après l'exécution d'une action a. L'ensemble E est composé de deux parties *add* et *del* pour les variables propositionnelles qui changent de valeur de vérité suite à l'application de l'action (ce qui devient faux est précédé du symbole ¬). Nous obtenons alors *add* : $s' \setminus s$ et *del* : $s \setminus s'$.

Couvertures entre règles et exemples

Nous définissons les différents tests élémentaires permettant de relier un modèle d'action composé de règles et les exemples qu'il rencontre. Sauf précision, nous illustrons chaque définition sur l'exemple d'apprentissage décrit dans le tableau 4.1.

Soit x un exemple noté $x.s/x.a/x.e$ et soit r une règle notée $r.p/r.a/r.e$.

Définition 10 (pré-couverture $\overset{sa}{\sim}$) *On dit qu'une règle r pré-couvre un exemple x si $r.p \subseteq x.s$ et $r.a = x.a$*

Si une règle pré-couvre un exemple, cela signifie que les conditions de la règle sont vérifiées dans la partie s de l'exemple. Il est alors possible d'utiliser cette règle afin de prévoir les effets de l'exemple qu'elle pré-couvre. ($r'_1 \overset{sa}{\sim} x_3$ page 53).

Définition 11 (post-couverture $\overset{ae}{\sim}$) *On dit qu'une règle r post-couvre un exemple x si $r.a = x.a$ et $r.e = x.e$*

Si une règle post-couvre un exemple, elle décrit les effets observés de l'action dans cet exemple. ($r_1 \overset{ae}{\sim} x_2$ page 53).

Définition 12 (couverture $\approx$) *On dit qu'une règle couvre un exemple si elle pré-couvre et post-couvre à la fois cet exemple.*

Si une règle pré-couvre et post-couvre un exemple, cela signifie qu'elle prévoit les effets de l'exemple et que la prévision est correcte. Soit un exemple $x_a = (p_1, p_3/a_1/\neg p_1, p_6)$, avec la règle r'_1 page 53, $r'_1 \approx x_a$.

Définition 13 (contradiction $\nsim$) *Si un exemple est pré-couvert par une règle mais non post-couvert par cette même règle, on dit que cet exemple contredit la règle.*

Lorsqu'un exemple x contredit une règle r, cela signifie que r prévoit pour e un effet différent de celui observé dans l'exemple. (page 53, $r'_1 \nsim x_3$).

L'agent rencontre également des actions particulières car elles le laissent dans le même état. Il s'agit d'exemples sans effets (ou illustrant des actions illégales). Le modèle n'a pas vocation à représenter des exemples sans effets puisqu'ils ne décrivent aucun changement (cf. section 4.5.2). Néanmoins, le modèle doit être capable de prévoir correctement l'absence d'effets si un tel exemple se présente. Ainsi, si pour une action et dans un état donnés, le modèle prévoit un effet alors qu'aucun effet n'est observé, on détecte une incohérence. Dans ce cas, cela signifie qu'une règle du modèle est contredite par un contre-exemple d'action illégale et qu'une révision s'impose. Pour que le modèle soit capable de couvrir un exemple aux effets vides, on définit un comportement par défaut : si pour un exemple donné, aucune règle du modèle ne peut prévoir d'effets, alors l'agent prévoit des effets vides.

Un modèle M est composé d'un ensemble de règles $M.R$ et d'un ensemble de contre-exemples $M.X$.

Les contre-exemples sont stockés dans $M.X$ tel que $M.X = M.X^+ \cup M.X^-$.

Définition 14 (contre-exemple négatif) *Si un exemple avec effets vides est pré-couvert par le modèle M (au moins une règle de $M.R$), alors on dit que cet exemple est un contre-exemple négatif pour M. Cet exemple est alors conservé dans $M.X^-$.*

Définition 15 (contre-exemple positif) *Si un exemple avec effets non vides n'est pas couvert par le modèle M, soit parce qu'aucune règle de $M.R$ ne le couvre, soit parce qu'une règle de $M.R$ le contredit, alors on dit que cet exemple est un contre-exemple positif pour M. Cet exemple est alors conservé dans l'ensemble $M.X^+$.*

Les exemples x_1, x_2 et x_3 du tableau 4.1 sont des contre-exemples positifs.

Définition 16 (complétude) *On dit que M est* complet *ssi $\forall x \in M.X^+$, il existe au moins une règle $r \in M.R$ telle que $r \approx x$. Cela signifie que l'ensemble de règles $M.R$ permet de prévoir correctement les effets pour chaque contre-exemple positif rencontré auparavant.*

Définition 17 (correction) *On dit que M est* correct *ssi i) $\forall x \in M.X^-$, il n'existe aucune règle $r_i \in M.R$ telle que $r_i \overset{sa}{\approx} x$ et que ii) $\forall x \in M.X^+$, il n'existe aucune règle $r_i \in M.R$ telle que $x \nsim r_i$. Cela signifie qu'aucune règle de $M.R$ n'est capable de prévoir d'effets pour chaque contre-exemple négatif rencontré auparavant, et qu'aucun contre-exemple positif n'est contredit.*

Définition 18 (cohérence) *On dit que M est* cohérent *si M est* complet *et* correct.

4.2.2. Apprentissage d'un modèle d'action propositionnel

Nous présentons l'agent IRALe (Rodrigues *et al.*, 2010a) (Incremental Relational Action Learning) capable d'apprendre un modèle d'action relationnel en ligne directement à partir d'exemples obtenus par ses interactions avec son environnement. Il est guidé par les données avec une approche principalement ascendante, mais il est aussi capable de revenir par spécialisation sur des généralisations antérieures. Nous commençons par introduire IRALe à l'aide d'une représentation propositionnelle permettant d'illustrer le comportement général de l'agent en vue d'introduire dans la section suivante les problèmes spécifiques de l'apprentissage d'un modèle d'action relationnel.

L'approche IRALe repose sur certaines hypothèses, mais le biais principal réside dans le guidage de l'apprentissage à partir des *effets des actions*. Concrètement, nous

considérons que le nombre d'objets distincts affectés par l'exécution d'une action est moindre relativement au nombre total d'objets d'un état. Ainsi, par exemple, l'action de saisir un bloc d'une pile affecte les relations existantes entre ce bloc et la pile sans toutefois affecter les autres objets du monde (la table et les blocs des autres piles). On suppose également que des actions ayant des effets similaires s'appliquent dans des conditions semblables. Il est alors possible, intuitivement, d'estimer les effets d'actions similaires comme des étiquettes de classe. Si l'on suppose que des exemples appartiennent à une certaine classe, apprendre les points communs des exemples appartenant à cette classe revient alors à chercher les conditions communes des actions appartenant à ces exemples et menant à ces mêmes effets.

Opérateurs

A partir des définitions exposées à la section 4.2.1, nous pouvons maintenant développer les principaux opérateurs de parcours de l'espace de recherche pour notre modèle.

Dans le cas où le modèle est capable de couvrir correctement l'exemple légal rencontré, il n'y a pas de révision de celui-ci. Il en va de même si l'exemple est sans effets et qu'aucune règle ne le pré-couvre.

Sinon, deux types de modifications peuvent avoir lieu :

- l'exemple légal n'est pas pré-couvert, il faut alors généraliser le modèle car aucune règle n'est capable d'en prévoir les effets
- les effets de l'exemple (vides ou non) sont incorrectement prédits, il faut alors spécialiser le modèle car il existe une règle (éventuellement plusieurs) qui pré-couvre(nt) l'exemple sans en prévoir correctement les effets.

L'exemple devient dans tous les cas un contre-exemple du modèle.

Si aucune règle de $M.R$ ne pré-couvre x, le modèle M ne peut prévoir d'effets pour x. Le modèle est donc incomplet et l'exemple x devient un contre-exemple pour M. Pour restaurer la complétude du modèle, il est possible de généraliser les pré-conditions d'une règle post-couvrant x ou de créer une nouvelle règle. Il faut aussi s'assurer que

la nouvelle généralisation obtenue préserve la cohérence du modèle. Si aucune règle du modèle ne post-couvre x, l'exemple est ajouté tel quel, comme nouvelle règle, à $M.R$.

Définition 19 (Généralisation) *Si r post-couvre x, la généralisation unique r' d'une règle r et d'un contre-exemple x est définie par $r' = Gen(r, x)$ et calculée comme : $r'.p = r.p \cap x.s$, $r'.a = r.a$ et $r'.e = r.e$.*

Pour une règle r construite par n généralisations successives, on a :

$$\begin{cases} r^0 & = x^0 \\ r^n & = Gen(r^{n-1}, x^{n-1}) \end{cases}$$

avec $Gen(r, x)$ la généralisation de la règle r avec le contre-exemple x et r^0 la création de la règle à partir de l'exemple x^0.

La règle r'_1 du tableau 4.1 peut s'écrire comme $r'_1 = Gen(r_1, x_2)$ et $r_1 = x_1$.

Si le modèle prévoit des effets erronés pour un exemple x et que l'exemple a des effets non vides, cela signifie que x contredit au moins une règle de $M.R$. Le modèle est donc incohérent et l'exemple x devient un contre-exemple. Pour rétablir la cohérence, il est nécessaire de spécialiser les règles contredites par x.

La spécialisation d'une règle est le retour en arrière sur des généralisations remises en cause par la rencontre de nouveaux contre-exemples. Pour permettre ce retour, il est nécessaire de conserver les généralisations successives d'une règle et la liste des contre-exemples responsables de ces différentes généralisations. Pour chaque règle, chacune de ces généralisations successives ainsi que les contre-exemples associés sont conservés.

Définition 20 (Spécialisation) *La spécialisation d'une règle r construite par n généralisations successives est définie par :*

$$\begin{cases} r = x^0 & si\ n{=}0 \\ r = r^{n-1} & avec\ r^n = Gen(r^{n-1}, x^{n-1}) \end{cases}$$

Le tableau 4.1 illustre une trace d'apprentissage à partir de la séquence des exemples x_1, x_2, x_3 ainsi que l'évolution du modèle d'action après l'observation de chaque exemple.

Modèle	*Règles (M.R)*	*Contre-exemple (M.X)*	*Etat/ Pré-conditions*	*Action*	$Effets$
M_0	$\emptyset$				
M_1	r_1	x_1	p_1, p_2, p_3	a_1	$\neg p_1, p_6$
		x_2	p_1, p_4, p_5	a_1	$\neg p_1, p_6$
M_2	r'_1		p_1	a_1	$\neg p_1, p_6$
		x_3	p_1, p_2, p_5	a_1	$\neg p_5, p_3$
M_3	r_1		p_1, p_2, p_3	a_1	$\neg p_1, p_6$
	r_2		p_1, p_4, p_5	a_1	$\neg p_1, p_6$
	r_3		p_1, p_2, p_5	a_1	$\neg p_5, p_3$

TABLEAU 4.1.: Trace d'apprentissage dans un langage propositionnel.

Initialement, le modèle M_0 est vide. Le modèle ne contenant aucune règle, il ne peut prévoir les effets de x_1 ; l'exemple est ajouté tel quel en tant que règle au modèle M_1. Ensuite, l'agent rencontre l'exemple x_2. Encore une fois, le modèle n'est pas capable de prévoir les effets du nouvel exemple x_2 car la seule règle du modèle r_1 ne le pré-couvre pas. Cependant r_1 et x_2 décrivent des couples (action, effets) identiques. Pour rétablir la complétude, on généralise les pré-conditions de r_1 avec l'état initial de x_2 en calculant le moindre généralisé de ces pré-conditions. On en déduit la règle r'_1 plus générale que r_1. Le modèle M_2 obtenu est capable de prévoir correctement les effets de x_1 et x_2.

L'agent rencontre ensuite l'exemple x_3. Il tente de prévoir les effets de x_3 avec son modèle courant M_2. Le modèle est composé d'une seule règle r'_1 pré-couvrant x_3 mais prévoyant des effets différents menant à une contradiction. Pour rétablir la cohérence du modèle, on spécialise les règles ayant mené aux erreurs de prévision en revenant sur les anciennes généralisations en cause. On revient sur les généralisations précédentes tant que ces généralisations contredisent x_3. Ce comportement est possible en gardant en mémoire tous les contre-exemples associés aux généralisations antérieures. La règle r'_1 est alors spécialisée en r_1 et r_2 et l'exemple x_3 est ajouté tel quel en r_3 car aucune règle ne peut décrire d'effets similaires.

on décide de ne pas modéliser directement de règles pour les contre-exemples issues

d'actions illégales. Ainsi, pour que le modèle soit néanmoins capable de couvrir un exemple aux effets vides, on définit un comportement par défaut : si pour un exemple donnée, aucune règle du modèle ne peut prédire d'effets, alors le système prédit des effets vides.

Algorithme 4 IRALe(M,x)

Entrée: Un exemple x, un modèle d'action M
Sortie: Un modèle M cohérent, couvrant x

$L_x \leftarrow \emptyset$
pour tout $r \in M.R$ telle que $x \nsim r$ **faire**
  $L_x \leftarrow L_x \cup \text{SPECIALISER}(M, x, r)$
fin pour
$M.X \leftarrow M.X \cup x$ **si** $L_x \neq \emptyset$
% A ce stade, x ne contredit plus $M.R$,
% mais la complétude du modèle n'est plus garantie
$\forall cx \in L_x, \text{GENERALISER}(M, cx)$
si $x.e \neq \emptyset$ et ($L_x \neq \emptyset$ ou $\nexists r \in M.R$ telle que $r \approx x$) **alors**
  $M.X^+ \leftarrow M.X^+ \cup x$
  GENERALISER(M, x)
fin si

L'algorithme 4 illustre la boucle principale d'IRALe. A chaque fois que l'agent rencontre un exemple, la correction et la complétude du modèle sont testées sur ce dernier. Si l'exemple a des effets non vides, qu'il ne contredit aucune règle et qu'il est couvert par une règle de $M.R$, alors il ne s'agit pas d'un contre-exemple. Il en va de même si le contre-exemple est sans effets et n'est pré-couvert par aucune règle. Le modèle est cohérent et n'est donc pas révisé.

Sinon, si l'exemple possède des effets non vides, qu'il contredit une règle ou encore qu'aucune règle ne le couvre, alors il s'agit d'un contre-exemple positif pour M et doit être pris en compte au sein du modèle.

Dans un premier temps, si un contre-exemple (positif ou négatif) contredit des règles de $M.R$, alors celles-ci doivent être spécialisées. Pour ce faire, l'algorithme 5 procède à un retour sur les généralisations antérieures des règles en cause tant que l'exemple courant est contredit. En spécialisant, les contre-exemples associés aux généralisations contradictoires sont stockés dans L_x.

A ce stade de l'algorithme 4, la correction du modèle est garantie car si l'exemple courant est sans effets, plus aucune règle ne le pré-couvre. S'il possède des effets, il ne peut désormais contredire aucune règle.

Dans le cas où le contre-exemple courant est sans effets, il doit être ajouté au modèle en tant que contre-exemple négatif dans $M.X$, mais il ne peut être généralisé et faire partie des règles. En effet, aucune règle ne doit pré-couvrir un contre-exemple négatif.

Algorithme 5 SPECIALISER(M,x,r) :L_x

Entrée: Un contre-exemple x en contradiction avec une règle $r \in M.R$
Sortie: Un modèle M spécialisé, correct, une liste L_x de contre-exemples responsables des généralisations contredites

1: $M.R \leftarrow M.R \setminus \{r\}$
2: $L_x \leftarrow \emptyset$
3: $r^n = r$ % *avec n le nombre de généralisations ayant conduit à la règle r*
4: **tant que** $x \nsim r^n$ et $n > 0$ **faire**
5: $\quad L_x \leftarrow L_x \cup x^n$
6: $\quad r^n \leftarrow r^{n-1}$
7: $\quad n \leftarrow n - 1$
8: **fin tant que**
9: **si** $n = 0$ **alors**
10: $\quad L_x \leftarrow L_x \cup x^0$
11: **sinon**
12: $\quad M.R \leftarrow M.R \cup \{r^n\}$
13: **fin si**

Les contre-exemples du modèle étaient tous correctement couverts avant la spécialisation de celui-ci et le demeurent, à l'exception des contre-exemples de L_x associés aux spécialisations. Ceux-ci ne contredisent aucune règle, mais la complétude du modèle n'est plus garantie vis-à-vis de ces contre-exemples.

Il est nécessaire de rétablir la complétude du modèle sur les contre-exemples de L_x sans toutefois perdre la correction et la complétude sur l'ensemble des contre-exemples du modèle.

Pour restaurer la complétude, tous les contre-exemples de L_x sont généralisés (cf. algorithme 6) afin que chacun soit couvert par une règle. Cependant, il est nécessaire de généraliser sous conditions pour ne pas introduire de nouvelles contradictions.

Algorithme 6 GENERALISER$_{PROP}(M,x)$

Entrée: Un contre-exemple x
Sortie: Un modèle M couvrant x

1: $mod \leftarrow faux$
2: **pour tout** $r \in M.R$ **faire**
3: **si** $r \overset{ae}{\sim} x$ **alors**
4: $r^n = r$ % *avec n le nombre de généralisations ayant conduit à la règle r*
5: $r^{n+1} \leftarrow Gen(r^n, x)$
6: **si** $\nexists cx \in M.X, cx \nsim r^{n+1}$ **alors**
7: $mod \leftarrow vrai$
8: $M.R \leftarrow M.R \backslash \{r^n\} \cup \{r^{n+1}\}$
9: **fin si**
10: **fin si**
11: **fin pour**
12: % *Si x n'est pas généralisé, il est ajouté tel quel à M.R*
13: **si** $mod = faux$ **alors**
14: $M.R \leftarrow M.R \cup \{x\}$
15: **fin si**

Puisque l'on garantit qu'aucun contre-exemple ne puisse être contredit par une nouvelle généralisation, il n'est pas possible d'introduire de nouvelles contradictions entre les contre-exemples $M.X$ et les règles de $M.R$.

De plus, comme chaque contre-exemple de L_x et le contre-exemple courant (s'il est positif) sont généralisés et donc couverts sans rajouter de nouvelles contradictions, alors tous les contre-exemples positifs sont couverts et aucun contre-exemple négatif n'est pré-couvert. Dans le pire des cas, il y a autant de règles que de contre-exemples positifs. Le modèle est alors la disjonction de tous les exemples légaux.

Comme tous les contre-exemples positifs sont couverts par les règles, alors on assure que le modèle est complet. Depuis la procédure de spécialisation, il n'y a plus de contradiction et les généralisations ne peuvent introduire de nouvelles contradictions. Le modèle est donc correct et cohérent par rapport à tous les contre-exemples rencontrés.

L'algorithme 6 détaille la procédure de généralisation propositionnelle. Chaque règle qui post-couvre le contre-exemple est remplacée par la généralisation obtenue entre la règle et le contre-exemple. Néanmoins, la généralisation obtenue peut être contredite par un

autre contre-exemple. Si aucune généralisation n'est possible, soit parce qu'il n'existe aucune règle post-couvrant l'exemple, soit parce qu'il n'existe pas de généralisation qui introduise de nouvelles contradictions, alors l'exemple en question est ajouté directement en tant que règle à $M.R$.

IRALe (cf. algorithme 4) peut alterner une phase de spécialisation et de généralisation. Cependant, il ne peut boucler entre deux modèles. Tous les contre-exemples rencontrés par IRALe sont conservés dans $M.X$ et constituent toutes les erreurs commises. A chaque généralisation, il est vérifié qu'aucun contre-exemple n'est contredit. Il n'est donc pas possible de refaire les mêmes erreurs.

4.3. Modèle d'action relationnel

Dans la section précédente, nous avons défini la boucle d'apprentissage d'un agent capable de généraliser les exemples qu'il rencontre en un modèle composé de règles permettant d'anticiper sur les conséquences de ses actions. Pour présenter l'algorithme, nous avons choisi dans un premier temps de nous abstraire d'une représentation relationnelle plus complexe. Ici, nous détaillons les conséquences de l'utilisation d'une représentation aussi expressive. Ainsi, nous commençons par adapter les précédentes définitions introduites section 4.2.1 au cadre relationnel afin de pouvoir développer la méthode IRALe pour l'apprentissage de modèle d'action EDS.

4.3.1. Définitions

Nous introduisons les définitions nécessaires à la description des relations de couverture entre les règles et les exemples dans un cadre relationnel. Toutes les définitions non modifiées dans cette section demeurent identiques à celles introduites section 4.2.1.

Les capacités de généralisation introduites par l'usage d'une représentation relationnelle permettent entre autres de décrire des modèles d'actions indépendants du nombre d'objets mis en jeu dans les environnements. Cependant, par rapport à une description

Règle $r = clear(A), clear(B), on(A, C)/$
$move(A, B)/$
$on(A, B), clear(C), \neg on(A, C), \neg clear(B)$
Règle $r_2 = clear(A), clear(c), on(A, c)$
$move(A, c)/$
$on(A, c), \neg on(A, f), \neg clear(c)$

$x_1 = clear(c), clear(b), on(c, a), on(a, f), on(b, f)/$
$move(c, b)/$
$on(c, b), clear(a), \neg on(c, a), \neg clear(b)$

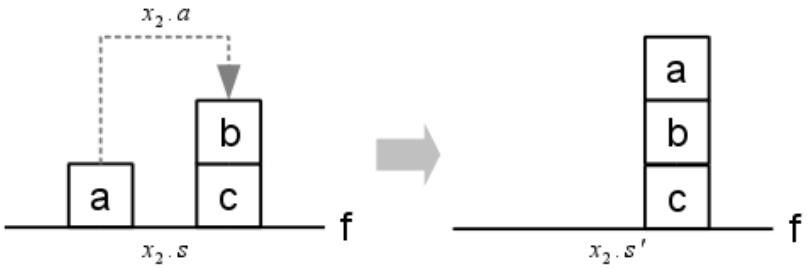
FIGURE 4.2.: Exemple x_1

propositionnelle comme celle utilisée précédemment, plusieurs nouvelles difficultés surviennent. En effet, l'introduction de variables dans les règles change considérablement la façon de généraliser les exemples rencontrés par l'agent. Ainsi, les pré-conditions d'une règle peuvent couvrir un grand nombre d'états. Un test d'inclusion seul comme établi à la définition 10 pour la pré-couverture ne suffit donc plus. Il en va de même pour la post-couverture des règles. On utilise alors l'OI subsomption (cf. définition 3 page 14). La définition 11 de la post-couverture ne suffit plus car les post-conditions des règles peuvent aussi être généralisées grâce à l'introduction de variables. Les tests de couverture deviennent plus complexes ; il est donc nécessaire de redéfinir ces tests pour tester la couverture entre une règle et un exemple.

Dans la suite, on considère un exemple $x = x.p/x.a/x.e$ et une règle $r = r.p/r.a/r.e$.

Définition 21 (pré-couverture $\overset{sa}{\sim}$) *Une règle r pré-couvre un exemple x ssi il existe une substitution OI θ_r telle que i) $(r.a)\theta_r = x.a$ ii) $(r.p)\theta_r \subseteq x.p$ iii) pour toute constante $c \in r.p$ ou $r.a$ alors $constante_{OI}(\theta_r, c)$ (cf. définition.4 page 14) est vrai.*

Par exemple, la figure 4.2, montre que la règle r *pré-couvre* l'exemple x_1 avec la



$x_2 = clear(a), clear(b), on(a, f), on(b, c), on(c, f)/$
$move(a, b)/$
$on(a, b), \neg on(a, f), \neg clear(b)$

FIGURE 4.3.: Exemple x_2

substitution $\theta = \{A/c, B/b, C/a\}$. Figure 4.3, la règle r *pré-couvre* l'exemple x_2 avec la substitution $\theta = \{A/a, B/b, C/f\}$.

Définition 22 (post-couverture $\overset{ae}{\sim}$) *Une règle r post-couvre un exemple x ssi il existe une substitution OI θ_r telle que i) $(r.a)\theta_r = x.a$ ii) $(r.e.add)\theta_r = x.e.add$ iii) $(r.e.del)\theta_r = x.e.del$ iv) pour toute constante $c \in r.a$ ou $r.e$ alors $constante_{OI}(\theta_r, c)$ est vrai.*

A la figure 4.2, la règle r *post-couvre* l'exemple x_1 avec la substitution $\theta = \{A/c, B/b, C/a\}$. La règle r_2 ne *post-couvre* pas x_1 car il n'existe pas de substitution θ telle que $(r_2.e.add)\theta = x_1.e.add$.

Contrairement à l'égalité stricte utilisée pour comparer les effets des règles et des exemples, comme réalisé précédemment, avec une représentation relationnelle et la présence de variables, notamment au sein de l'action et des effets des règles, la post-couverture ne suffit pas. Il faut en effet pouvoir généraliser aussi les effets en transformant les constantes en variables. C'est pourquoi on calcule une généralisation OI entre les actions et effets des règles et des exemples en imposant des restrictions. Lorsqu'on calcule une post-généralisation définie à l'algorithme 7, il est nécessaire d'interdire l'oubli de littéraux par $LIT\text{-}GEN_{OI}$ (page 19) car celle-ci doit permettre non pas de décrire un sous-ensemble de l'action et des effets d'un exemple, mais leur intégralité. Cependant, elle doit permettre l'introduction éventuelle de variables (ce qui réalise alors la généralisation). Afin de garantir ce comportement, il est nécessaire de vérifier qu'il existe des substitutions OI θ_r et θ_x pouvant s'appliquer sur l'action et les effets.

Algorithme 7 post-généralisation $\overset{AE}{\sim}$

Entrée: Une règle r, un exemple x
Sortie: En cas de succès, les substitutions OI θ_r et θ_x, $r \overset{AE}{\sim} x$

$\theta_r \leftarrow \emptyset$, $\theta_x \leftarrow \emptyset$
si $LIT\text{-}GEN_{OI}(r.a, x.a, \theta_r, \theta_x) \neq \emptyset$ **alors**
  si $\exists add_{gen}$, $add_{gen} = UNE\text{-}GEN_{OI}(r.e.add, x.e.add, \theta_r, \theta_x)$ et $(add_{gen})\theta_x = x.e.add$ et $\forall$ constante $c \in r.e.add$ ou $r.a$ t.q $constante_{OI}(\theta_r, c)$ **alors**
    si $\exists del_{gen}$, $del_{gen} = UNE\text{-}GEN_{OI}(r.e.del, x.e.del, \theta_r, \theta_x)$ et $(del_{gen})\theta_x = x.e.del$ et $\forall$ constante $c \in r.e.add, r.a$ ou $r.e.del$ t.q $constante_{OI}(\theta_r, c)$ **alors**
      retourne vrai % ainsi que θ_r et θ_x
    fin si
  fin si
fin si
retourne faux

Exemple 6 Définition 23 (post-généralisation $\overset{AE}{\sim}$) *Une règle r post-généralise un exemple x avec les substitutions θ_r et θ_x ssi toutes les conditions de l'algorithme 7 de post-généralisation sont vérifiées.*

il existe une substitution OI θ_r telle que i) $(r.a)\theta_r = x.a$ ii) $(r.e.add)\theta_r = x.e.add$ iii) $(r.e.del)\theta_r = x.e.del$ iv) pour toute constante $c \in r.a$ ou $r.e$ alors $constante_{OI}(\theta_r, c)$ est vrai.

La règle r_2 ne post-couvre pas l'exemple x_2 mais le post-généralise. Illustrons l'algorithme 7 sur cet exemple. Initialement, θ_r et θ_x sont vides.
$LIT\text{-}GEN_{OI}(move(A, c), move(a, b), \theta_r, \theta_x) = move(A, C)$ *avec* $\theta_r = \{C/c\}$ *et* $\theta_x = \{A/a, C/b\}$.

on continue avec les add :

$UNE\text{-}GEN_{OI}(on(A, c), on(a, b), \theta_r, \theta_x) = on(A, C)$
$on(A, C)\theta_x = on(a, b) = x.e.add$ *avec θ_r et θ_x inchangées*

on continue avec les del :

$UNE\text{-}GEN_{OI}(on(A, f), clear(c), on(a, f), clear(b), \theta_r, \theta_x) = on(A, f), clear(C)$
$on(A, f), clear(C)\theta_x = on(a, f), clear(b) = x.e.del$ *avec θ_r et θ_x identiques.*

On en conclut que $r \overset{AE}{\sim} x$ *avec* $\theta_r^{-1} = \{c/C\}$

Définition 24 (couverture $\approx$) *Soit une règle* r *et un exemple* x, $r \approx x$ *ssi* $r \overset{sa}{\sim} (x.s, x.a)$ *et* $r \overset{AE}{\sim} (x.a, x.e)$ *avec la même substitution OI* θ_r *et que* $\forall$ *constante* $c \in r.p$, $r.a$ *ou* $r.e$ *alors* $constante_{OI}(\theta_r, c)$ *est vrai.*

La règle r couvre l'exemple x_1 car elle *pré-couvre* et *post-couvre* x_1 avec la substitution $\sigma = \{A/c, B/bC/a\}$. La règle r ne couvre pas l'exemple x_2 car elle ne le *post-couvre* pas. Ce test permet de savoir si un exemple est correctement prévu par une règle du modèle.

Définition 25 (contradiction $\nsim$) *Un exemple* x contredit *une règle* r *(*$x \nsim r$*) si* r *pré-couvre* x *avec la substitution OI* θ *et* r *ne post-couvre pas* x *avec la même substitution.*

L'exemple x_2 contredit la règle r car r *pré-couvre* x_2 avec la substitution OI $\theta = \{A/a, B/b, C/f\}$, et ne *post-couvre* pas x_2. Ce test permet d'identifier les exemples pour lesquels une règle prévoit incorrectement les effets de l'action.

Les autres définitions introduites section 4.2.1 demeurent identiques avec les différentes représentations.

4.3.2. Apprentissage d'un modèle d'action relationnel

Nous portons l'approche IRALe au cadre relationnel en étayant les difficultés apportées par l'apprentissage dans ce cas. La difficulté principale réside dans la procédure de généralisation, les autres procédures demeurant inchangées. La généralisation des pré-conditions des règles sous forme de conjonctions d'atomes pose de nouveaux problèmes par rapport à la généralisation de conjonctions de propositions. Contrairement à cette dernière, le calcul d'un moindre généralisé sous identité objet n'est pas unique (*cf.* section 2.2.3) et leur nombre croît exponentiellement avec la longueur des conjonctions.

Nous développons une nouvelle approche pour gérer le problème de complexité lié aux multiples moindres généralisés. Au lieu de calculer tous les moindres généralisés possibles, on se propose de ne calculer qu'une seule généralisation.

Dans le cadre d'un apprentissage de modèle d'action, ce choix permet de généraliser le modèle rapidement en réduisant le temps nécessaire à l'agent entre deux interactions avec l'environnement. Le système repose alors sur les exemples futurs afin de remettre en cause si nécessaire un précédent choix inapproprié de moindre généralisé.

Comme souligné par Esposito *et al.* (2004), afin de garantir que l'ensemble des résultats trouvés représentent bien l'ensemble minimal de tous les moindres généralisés, il est nécessaire de tous les comparer entre eux. Autrement dit, si l'on se restreint à n'en calculer qu'un seul, il n'est pas possible avec cette procédure de garantir que la généralisation obtenue soit un moindre généralisé. Un contre-exemple futur pourra alors éventuellement remettre en cause ce choix.

Le nombre de moindres généralisés est limité grâce à l'étape de post-généralisation. Celle-ci permet de fixer les appariements de tous les objets directement impliqués dans l'action. Leur prise en compte élague considérablement les généralisations possibles.

Avec ces choix de conception, (on réduit le coût du calcul de la généralisation de façon drastique), on obtient un agent réactif capable de réviser son modèle rapidement.

L'algorithme 8 détaille la procédure de généralisation adaptée à une représentation relationnelle. Les autres procédures d'IRALe demeurent identiques.

Soit un contre-exemple x et une règle r. On commence par tester si r post-généralise x (*ligne* 3, *cf. algorithme7 page 60*). Si c'est le cas, il est possible de généraliser r avec x. On calcule alors (*ligne* 7) une généralisation p_{gen} entre les pré-conditions de r et de x en prenant en compte les substitutions obtenues par la post-généralisation à l'aide des substitutions inverses θ_r^{-1} et θ_x^{-1}.

Ensuite, on teste si la nouvelle règle généralisée est syntaxiquement *bien formée* et si aucun contre-exemple ne la contredit (comme pour l'algorithme 6 $GENERALISER_{PROP}$).

Algorithme 8 GENERALISER$_{REL}(M,x)$

Entrée: Un modèle M, un contre-exemple x
Sortie: Un modèle M couvrant x

$mod \leftarrow faux$
pour tout $r \in M.R$ **faire**
  si $r \overset{AE}{\sim} x$ (avec θ_r et θ_x) **alors**
    $r^n = r$ % avec n le nombre de généralisations ayant conduit à la règle de r
    $r^{n+1}.a = (r^n.a)\theta_r^{-1}$
    $r^{n+1}.e = (r^n.e)\theta_r^{-1}$
    $p_{gen} \leftarrow UNE\text{-}GEN_{OI}(r^n\theta_r^{-1},\ x,\ \theta_r,\ \theta_x)$
    $r^{n+1} = p_{gen}/r^{n+1}.a/r^{n+1}.e$
    si bien-formée(r^{n+1}) et $\nexists cx \in M.X, cx \nsim r^{n+1}$ **alors**
      $mod \leftarrow vrai$
      $M.R \leftarrow M.R \backslash \{r^n\} \cup \{r^{n+1}\}$
    fin si
  fin si
fin pour
% Si x n'est pas généralisé, il est ajouté tel quel à $M.R$
si $mod = faux$ **alors**
  $M.R \leftarrow M.R \cup \{x\}$
fin si

Si ces conditions sont vérifiées, la généralisation de r est rajoutée au modèle dans $M.R$ à la place de sa version antérieure.

Autrement, si le contre-exemple x ne peut généraliser aucune règle, il est alors ajouté directement en tant que règle de $M.R$ afin de maintenir la complétude.

Comme une des conditions indispensables à la généralisation d'une règle par un contre-exemple est que la règle doive post-généraliser celui-ci, le calcul de la généralisation des pré-conditions d'une règle prend en compte les substitutions (inverses) déjà calculées pour tester la post-généralisation. Par exemple, un calcul de post-généralisation peut renseigner que la variable X est substituée à la constante a. Cette substitution est vérifiée sur l'action ainsi que sur les effets, mais elle doit aussi être vérifiée sur les pré-conditions. Autrement, une règle ne pourrait pas couvrir un exemple (*cf. def.24*) puisque la substitution résultante ne serait pas injective ; cela reviendrait à violer l'hypothèse OI. C'est pourquoi la généralisation des pré-conditions doit aussi prendre

en compte les substitutions obtenues par la post-généralisation. Ces substitutions peuvent réduire la complexité de la généralisation en fonction des termes des pré-conditions déjà rencontrés dans l' action et les effets qui représentent alors autant de points de choix en moins dans le calcul de la généralisation.

Algorithme 9 $UNE\text{-}GEN_{OI}(r,x,\theta_r,\theta_x) : GEN$

Entrée: Une règle r et un contre-exemple x avec éventuellement des contraintes OI à respecter avec les substitutions OI θ_r et θ_x
Sortie: Une généralisation OI de r avec les substitutions OI θ_r et θ_x associées

$GEN \leftarrow \emptyset$, $L_{GEN} \leftarrow \emptyset$
$L_r \leftarrow r.p$
$L_x \leftarrow x.p$
tant que L_r et $L_x \neq \emptyset$ **faire**
  si $Sélection(L_r, L_x, r, x, \theta_r, \theta_x) : GEN, l_r, l_x$ **alors**
    $L_x \leftarrow L_x \backslash l_x$
  fin si
  $L_r \leftarrow L_r \backslash l_r$
fin tant que
retourne une généralisation OI : GEN

La procédure $UNE\text{-}GEN_{OI}$ (*cf.* Algorithme 9) retourne la première généralisation trouvée entre les pré-conditions de la règle et du contre-exemple. Cette généralisation est effectuée de la même manière que pour l'Algorithme 1 de GEN_{OI} (page 18) en utilisant la même procédure $LIT\text{-}GEN_{OI}$. cependant, dans notre cas, au lieu de calculer toutes les généralisations possibles, une seule est retournée. La généralisation obtenue dépend alors de l'ordre des littéraux choisis lors de la procédure de Sélection (cf. Algorithme 10). Afin de ne pas introduire de biais dus à un ordre particulier de littéraux, la combinaison des littéraux compatibles entre les pré-conditions de la règle et du contre-exemple s'effectue alors dans un ordre aléatoire.

Lorsque la généralisation p_{gen} est calculée, il n'est pas possible que de nouveaux appariements contredisent ceux établis sur l'action et les effets grâce aux substitutions θ_r et θ_x. Cela signifie que les substitutions trouvées avec p_{gen} ne peuvent contredire ni les variables présentes dans l'action ou les effets, ni les nouvelles variables introduites par le calcul de post-généralisation. Cependant, si p_{gen} introduit une nouvelle variable v qui n'apparaissait pas dans $(r.a)\theta_r^{-1}$ ou dans $(r.e)\theta_r^{-1}$, alors que v provient d'une constante c ayant des occurrences à la fois dans $r.p$ et dans $r.a$ ou $r.e$, cela signifie que

Algorithme 10 *Sélection*(L_r, L_x, r, x, θ_r, θ_x) : GEN, l_r, l_x

Entrée: Les conjonctions de littéraux L_r et L_x à généraliser depuis GEN et des substitutions OI θ_r et θ_x, la règle r et le contre-exemple x à l'origine de la généralisation

Sortie: En cas de succès, une généralisation GEN de $r.p$ étendue aux littéraux l_r et l_x aléatoirement choisis parmi les littéraux compatibles de L_r et L_x, ainsi qu'une mise à jour des substitutions OI θ_r et θ_x obtenues

1: $\theta_r^{tmp} \leftarrow \theta_r$
2: $\theta_x^{tmp} \leftarrow \theta_x$
3: **si** $\exists l_r \in L_r$ et $l_x \in L_x$ choisis aléatoirement t.q l_r et l_x compatibles et **si** $\exists L_{GEN}^{tmp} = LIT\text{-}GEN_{OI}(l_r, l_x, \theta_r^{tmp}, \theta_x^{tmp})$ t.q $(GEN \cup L_{GEN}^{tmp}/r.a/r.e) \approx x$ **alors**
4: $GEN \leftarrow GEN \cup L_{GEN}^{tmp}$
5: $\theta_r \leftarrow \theta_r^{tmp}$
6: $\theta_x \leftarrow \theta_x^{tmp}$
7: **retourne** vrai % ainsi que GEN, l_r et l_x
8: **sinon**
9: **retourne** faux
10: **fin si**

les occurrences de c dans $r.a$ et $r.e$ n'ont pas été généralisées, contrairement à celles de $r.p$. Dans ce cas, on obtient une contradiction : la généralisation p_{gen} d'une règle r et d'un contre-exemple x ne couvre pas x à cause de la contrainte OI.

Afin de supprimer cette contradiction, on s'assure lors de la sélection des littéraux candidats à la généralisation p_{gen} avec l'Algorithme 10 (*ligne 3*) que la généralisation obtenue couvre bien x.

On développe l'exemple suivant illustrant la généralisation d'une règle r avec un contre-exemple x (la dénomination des variables est arbitraire) :

r	$p(a,x), p(b,z), q(b,C)$	$m(a,b)$	$p(a,b)$, $\neg q(b,C)$
x	$p(a,i), p(b,j), q(b,c)$	$m(a,b)$	$p(a,b)$, $\neg q(b,c)$

Post-généralisation

On commence par tester la post-généralisation (cf. algorithme 7) de x par r :
$LIT\text{-}GEN_{OI}(r.a,\ x.a,\ \theta_r,\ \theta_x)$=$m(a,b)$ avec $\theta_r = \emptyset$ et $\theta_x = \emptyset$.
On continue le processus sur les effets. On généralise $r.e.add$ et $x.e.add$, on obtient $p(a,b)$ et enfin on calcule :
$UNE\text{-}GEN_{OI}(r.e.del,\ x.e.del,\ \theta_r,\ \theta_x)$=$q(b,C)$ avec $\theta_r = \emptyset$ et $\theta_x = \{C/c\}$.
c est $constante_{oi}$, les conditions de la post-généralisation sont vérifiées, alors $r \overset{AE}{\sim} x$ avec $\theta_r = \emptyset$ et $\theta_x = \{C/c\}$.

Généralisation des pré-conditions

On appelle l'algorithme 9, $UNE\text{-}GEN_{OI}$ afin de généraliser les pré-conditions de r avec x : deux littéraux compatibles l_r et l_x de $r.p$ et $x.p$ sont donc sélectionnés aléatoirement, on suppose ici qu'il s'agit respectivement de $q(b,C)$ et $q(b,c)$.
$LIT\text{-}GEN_{OI}$ vaut $q(b,C)$ avec θ_r et θ_x inchangées. Comme la généralisation obtenue permet de couvrir x, elle est donc conservée dans GEN.

On continue le processus sur les littéraux restants de L_r et L_x, à savoir respectivement : $p(a,x), p(b,z)$ et $p(a,i), p(b,j)$. On suppose que $p(b,z)$ et $p(a,i)$ sont compatibles et sélectionnés aléatoirement.
$LIT\text{-}GEN_{OI}$ vaut alors $p(B,Z)$, avec $\theta_r^{tmp} = \{B/b, Z/z\}$ et $\theta_x^{tmp} = \{C/c, B/a, Z/i\}$.
Dans ce cas, la généralisation courante : $(GEN \cup L_{GEN}^{tmp})$ vaut $p(B,Z), q(b,C)$.
On obtient la règle : $p(B,Z), q(b,C)/m(a,b)/p(a,b), \neg q(b,C)$ qui ne peut couvrir x. En effet, θ_r^{tmp} contient l'appariement B/b alors qu'il demeure des occurrences de la même constante b dans GEN, $r.a$ et $r.e$, ce qui contredit la contrainte OI. (cf. def. 3).

On teste alors s'il existe un autre couple de littéraux compatibles au sein de L_r et L_x (resp. $p(a,x), p(b,z)$ et $p(a,i), p(b,j)$). Il existe $p(b,z)$ et $p(b,j)$ ou encore $p(a,x)$ et $p(a,i)$.
Supposons que le couple $p(b,z)$ et $p(b,j)$ soit choisi :
$LIT\text{-}GEN_{OI}$ vaut $p(b,Z)$, on a $\theta_r = \{Z/z\}$ et $\theta_x = \{C/c, Z/i\}$.

La généralisation courante : $(GEN \cup L^{tmp}_{GEN})$ vaut $p(b, Z), q(b, C)$. On obtient la règle : $p(b, Z), q(b, C)/m(a, b)/p(a, b), \neg q(b, C)$ qui couvre x.

On réitère une dernière fois le processus, il ne reste que $p(a, x)$ et $p(a, i)$ dans L_r et L_x qui sont compatibles. $LIT\text{-}GEN_{OI}$ retourne $p(a, X)$ avec $\theta_r = \{X/x, Z/z\}$ et $\theta_x = \{X/i, C/c, Z/i\}$.

La généralisation courante : $(GEN \cup L^{tmp}_{GEN})$ vaut $p(a, X), p(b, Z), q(b, C)$. On obtient la règle : $p(a, X), p(b, Z), q(b, C)/m(a, b)/p(a, b), \neg q(b, C)$ qui couvre x.

Finalement, L_r et L_x sont vides, la généralisation courante retournée est une généralisation des pré-conditions de r avec x.

4.4. Convergence de l'approche

En gardant tous les contre-exemples en mémoire, il est possible d'apporter des garanties de convergence pour notre approche. Chaque contre-exemple sauvegardé dans $M.X$ représente une erreur de prévision du modèle, et les algorithmes présentés assurent que la même erreur ne soit pas commise à nouveau. Ainsi, les modèles successifs après révisions restent toujours corrects et complets relativement à toutes les erreurs passées.

Soient $\mathcal{H}$ l'espace des ensembles de règles d'action, et $\mathcal{E}$ l'espace des exemples. Dans notre approche, indépendamment de la représentation utilisée, un exemple est composé d'un état, d'une action et d'un ensemble d'effets. Les effets de l'action définissent la cible d'apprentissage. Dans l'hypothèse d'un monde déterministe, dans un état et pour une action donnés, il ne peut y avoir qu'un seul ensemble d'effets possible. Pour tout exemple appartenant à $\mathcal{E}$, à la présentation de l'état et de l'action de l'exemple, le modèle doit retourner l'ensemble des effets de l'exemple. On suppose que ce problème d'apprentissage est réalisable et qu'un tel modèle M existe et appartient à $\mathcal{H}$. Les ensembles $\mathcal{H}$ et $\mathcal{E}$ sont supposés finis.

Proposition 1 *Il est possible de trouver une solution exacte M en au plus $|\mathcal{H}|$ ou $|\mathcal{E}|$ révisions.*

Preuve 1 *A chaque erreur de couverture, par défaut de complétude ou de correction, le modèle courant est révisé afin que l'exemple à l'origine de l'erreur soit prévu correctement. Par la suite, le procédé est répété pour chaque erreur rencontrée. Chaque erreur est conservée sous la forme de contre-exemple dans $M.X$. Le modèle courant est donc toujours correct et complet par rapport à l'ensemble des contre-exemples rencontrés. Dans le pire des cas, chaque fois qu'un nouvel exemple (jamais rencontré auparavant) se présente, le modèle courant commet une erreur de prévision. Comme le modèle ne peut pas commettre d'erreurs sur des contre-exemples déjà rencontrés et qu'il ne peut y avoir plus de contre-exemples que d'exemples différents, IRALe fait au plus $\mathcal{E}$ erreurs ou révisions du modèle.*

Toujours dans ce même pire des cas, étant donné que chaque nouvelle erreur mène aussi à une nouvelle hypothèse, il ne peut y avoir plus d'erreurs qu'il n'existe de modèles différents dans $\mathcal{H}$. Il y a donc au maximum $|\mathcal{H}|$ révisions.

Dans notre proposition, les bornes apportées ne dépendent pas du nombre d'actions effectuées par l'agent. Elles s'expriment uniquement en fonction de ses erreurs. En cela, notre proposition se place dans le cadre du *Mistake Bound* introduit par Littlestone (1988).

Les bornes apportées ne permettent pas de connaître le nombre d'interactions nécessaires entre l'agent et son environnement. Ainsi, il est possible qu'un exemple correctement couvert et présenté plusieurs fois à l'agent (mais non stocké) devienne incorrectement prévu après une mise à jour du modèle. Néanmoins, après la mise à jour, l'exemple (devenu contre-exemple) sera nécessairement correctement prévu.

Le nombre de contre-exemples sauvegardés dans $M.X$ est un des paramètres de complexité d'IRALe car à chaque nouvelle mise à jour, il est nécessaire de vérifier qu'elle n'introduit pas de nouvelles contradictions par rapport à l'ensemble des contre-exemples. En pratique, comme discuté sur différents problèmes à la section 4.5, IRALe

sauvegarde peu de contre-exemples relativement à la taille de l'espace des exemples rencontrés.

Autoriser conjointement la généralisation et la spécialisation des règles pourrait introduire des boucles alternant les deux opérateurs si aucune précaution n'était prise. Dans IRALe, il est possible qu'une règle soit spécialisée, puis généralisée, puis encore spécialisée, etc. Cependant, chaque spécialisation aura pour origine un nouveau contre-exemple, et chaque généralisation suivante prendra en compte le nouveau contre-exemple ; il n'y a donc pas de boucle car le modèle prend en compte après chaque révision de nouvelles informations. Ainsi, à chaque révision, on s'assure de couvrir un exemple supplémentaire ou de rejeter un exemple de plus.

4.5. Etude empirique

Afin d'évaluer la pertinence de notre approche, nous développons un protocole expérimental dans le but de répondre aux questions suivantes :
– Quelle est la qualité des modèles générés ?
– Peut-on comparer notre modèle d'action à d'autres méthodes ?
– Les modèles générés sont-ils utilisables et efficaces ?
Pour y parvenir, nous commençons par introduire les différents domaines sur lesquels nous allons tester IRALe. Ensuite, nous expliquons le protocole d'apprentissage commun aux différentes expériences. Enfin, nous détaillons chacune des expériences menées ainsi que leurs interprétations respectives, permettant de répondre aux questions principales annoncées plus haut, en commentant à chaque fois les résultats obtenus.

4.5.1. Domaines

Pour évaluer notre système, il est nécessaire de le confronter à un environnement. Dans cette optique, nous utilisons dans un premier temps le monde de blocs déjà employé plus haut comme exemple illustratif. Il s'agit, avec certaines de ses variantes, de l'environnement relationnel le plus utilisé dans la littérature aussi bien pour la

planification (Slaney et Thiébaux, 2001), (Pasula *et al.*, 2007) qu'en *RRL* (Kersting et De Raedt, 2004), (Driessens, 2004), (Mellor, 2009). Il constitue donc un banc d'essai privilégié.

Le monde de blocs est notamment utilisé pour tester le système MARLIE (Croonenborghs *et al.*, 2007) ainsi qu'un autre domaine traitant d'un problème de logistique. Le système MARLIE n'étant pas disponible, nous utiliserons dans un premier temps ces deux domaines pour une comparaison indirecte avec notre méthode. Plus loin, nous comparerons directement notre approche en utilisant des arbres de régression avec le même mode opératoire que le système MARLIE.

Nous décrivons en détails les différents domaines utilisés ci-après.

Monde de blocs

Dans le *monde de blocs*, les états sont composés de différents blocs (a, b, ...) ainsi que d'une table t sur laquelle ils se trouvent. Un état est défini par des empilements de blocs donnés. La description des différents états du monde nécessite l'utilisation des prédicats suivants :

- $on(X, Y)$ lorsque X se trouve sur Y
- $clear(X)$ si aucun autre bloc ne se trouve sur X
- $bloc(X)$ si X est un bloc.

Les actions sont représentées à l'aide de l'unique prédicat d'action $move(X, Y)$. Un déplacement légal consiste à déplacer un bloc libre vers un autre bloc libre ou vers la table. Trois règles sont ici nécessaires pour représenter le modèle d'action (cf. Annexe). Nous évaluons le système dans un monde composé de 7 blocs dans les mêmes conditions que (Croonenborghs *et al.*, 2007) ce qui représente près de 38000 états différents possibles. Les différentes règles du monde de blocs sont données en Annexe.

Pour un monde composé de n blocs, le nombre d'états possibles est :

$$\sum_{i=0}^{n} \binom{n}{i} \frac{(n-1)!}{(i-1)!}$$

Nombre de blocs	Nombre d'états
5	501
6	4 051
7	37 633
8	394 353
9	4 596 553
10	58 941 091

TABLEAU 4.2.: Nombre d'états dans des mondes de blocs de différentes tailles.

Monde de blocs colorés

Pour rendre le problème du choix de généralisation plus complexe, nous ajoutons des attributs de couleurs au monde de blocs décrit ci-dessus. Chaque bloc peut être soit noir soit blanc, respectivement à l'aide des prédicats $b(X)$ et $w(X)$. Ce choix a comme première influence d'agrandir l'espace d'états. Avec 2 couleurs, dans un monde de 7 blocs, on passe alors à près de 5 millions d'états possibles. Les couleurs sont uniformément réparties. Au-delà du nombre d'états, on complexifie surtout le modèle d'action. Lorsque $move(X, Y)$ est choisi, le bloc X est déplacé sur le bloc Y uniquement si les deux blocs sont de la même couleur. Dans le cas contraire, le bloc X n'est pas déplacé, et il prend la couleur du bloc Y. Pour représenter le modèle d'action, sept règles sont ici nécessaires (*cf.* Annexe). Dans ce domaine, les conditions pour déplacer un bloc sont plus complexes car elles dépendent aussi de leur couleur. Plusieurs règles peuvent avoir les mêmes effets sans pour autant avoir les mêmes conditions. Comme la couleur n'intervient pas explicitement dans les effets de certaines règles, il est possible qu'elle soit oubliée, menant ainsi à des règles trop générales et par conséquent à des erreurs de prévision.

Logistique

Le domaine *logistique* est aussi utilisé pour l'évaluation du système MARLIE. Il s'agit d'un problème de transport décrivant l'acheminement de paquets vers des villes destinations à l'aide de camions. Un problème $logistique(p, v, c)$ se compose de p paquets, de v villes et de c camions. Un état représente la localisation des différents paquets et camions. On représente chaque état à l'aide des prédicats suivants :

- $dans\_camion(P, C)$ si le paquet P est dans le camion C
- $dans\_ville(P, V)$ si le paquet P est dans la ville V
- $camion\_dans(C, V)$ si le camion C est dans la ville V.

Le déplacement des différents objets se réalise à l'aide des prédicats suivants :

- $charger(P, C)$ pour charger le paquet P dans le camion C
- $décharger(P, C)$ pour décharger le paquet P du camion C
- $déplacer(C, V)$ pour déplacer le camion C vers la ville V.

nous procédons à l'évaluation dans le domaine $logistique(5, 5, 5)$ dans les mêmes conditions que (Croonenborghs *et al.*, 2007) mais aussi dans le domaine $logistique(10, 10, 10)$ avec un plus grand espace d'états.

Pour un monde composé de c camions, b boîtes et v villes, le nombre d'états est :

$$c \times v \times b \times (c + v)$$

Les problèmes logistique(5,5,5) et logistique(10,10,10) possèdent donc respectivement 1250 et 20000 états.

Par rapport au monde de blocs, le problème logistique présente plusieurs différences. Contrairement au monde de blocs, dans un état donné, plusieurs types d'actions différentes sont applicables. Le domaine logistique comporte notamment plusieurs objets aux attributs différents et en égale proportion alors que dans le monde de blocs, seul l'objet table diffère des autres objets.

Pour ces trois environnements, nous donnons en Annexe leur modèle d'action sous forme de règle EDS.

4.5.2. Protocole expérimental d'apprentissage

En début d'apprentissage, le modèle d'action de l'agent est vide. Il n'a aucune connaissance *a priori* sur son environnement. L'agent sait uniquement quelles sont les actions qu'il est en mesure de réaliser. Cependant, ni les conditions de leurs applications ni leurs effets ne sont connus. Seule la confrontation avec l'environnement peut fournir des exemples d'apprentissage afin d'en apprendre un modèle d'action. Dans cette étude, IRALe est uniquement guidé par une marche aléatoire. Ceci permet de ne pas introduire de biais liés à une stratégie particulière. On peut alors réduire l'apprentissage de l'agent à des enchaînements d'actions choisies aléatoirement et qui permettent une exploration du monde.

Cependant, à chaque instant, les exemples susceptibles d'être découverts dépendent de l'état courant. L'environnement fournit donc successivement des exemples qui ne sont pas uniformément répartis dans l'espace des états et des actions.

L'apprentissage se déroule en enchaînements d'épisodes. Chaque épisode est une séquence d'actions commençant à partir d'un état initial tiré aléatoirement et se finit après 20 actions ou si un but donné est atteint.

Les exemples d'apprentissage –les triplets $(état_t, action, état_{t+1})$– proviennent des actions exécutées par l'agent et des retours fournis par son environnement. L'agent n'a aucune connaissance a priori du modèle d'action de son environnement et donc aucune idée de ce qu'il est possible ou pas de réaliser. N'étant a priori pas plus doté d'une politique ou stratégie particulière, le choix de l'agent d'émettre une action plutôt qu'une autre est aléatoire. Par conséquent, il a la possibilité d'émettre des actions non prévues par son environnement, actions considérées sans aucun effet mais syntaxiquement correctes. De telles actions qualifiées d'illégales laisseront ainsi l'agent dans le même état. Par exemple, dans le monde de blocs, empiler la table sur un bloc, ou déplacer un bloc non libre sont des actions sans effets car illégales.

L'agent doit apprendre les actions légales parmi les actions syntaxiquement correctes. Cela complexifie considérablement l'apprentissage dans la mesure où les actions syntaxiquement correctes sont plus nombreuses que les actions légales dans les environnements

étudiés. Par exemple, à la figure 4.1 de la page 41, à partir de l'état s_t, on effectue le déplacement légal du bloc d sur le bloc c. Les autres déplacements légaux à partir de s_t sont : $move(d,a), move(c,d), move(c,a), move(c,table), move(a,c), move(a,d)$. Puisque le monde est composé de 5 objets, l'action move(X,Y) possède 25 instanciations différentes possibles. Il y a seulement 7 actions légales sur 25 actions syntaxiquement possibles. Il existe une probabilité plus grande de choisir une action illégale qu'une action légale.

Afin de rendre compte de l'évolution du modèle d'action au cours du temps, chacune des expériences qui suivent est fonction du nombre d'actions exécutées par l'agent.

4.5.3. Qualité des modèles

Protocole expérimental

Afin d'évaluer la qualité des modèles appris, nous testons le modèle sur des ensembles d'exemples tirés aléatoirement et uniformément au fil de l'apprentissage. Comme pour les exemples d'apprentissage, les exemples d'évaluation peuvent représenter des actions légales ou sans effets.

Pour chaque exemple x aléatoire tel que $x = s/a/e$, l'évaluation porte sur la capacité d'IRALe à prévoir correctement les conséquences de l'action a à partir de l'état s et donc d'en déduire l'état résultant s' tel que $s' = x.s \cup x.e.add \setminus x.e.del$). Pour déduire un état $\hat{s}'$ à partir d'IRALE, nous utilisons l'Algorithme 11 de Prévision. Pour une action a dans un état donné s, s'il existe une règle r du modèle capable de pré-couvrir x avec θ_r, alors on en déduit les $effets_{prévus}$ de a à partir de s en appliquant θ_r aux effets de r. Dans le cas où plusieurs règles pré-couvrent x, une règle est choisie aléatoirement. Autrement, dans le cas où aucune règle ne pré-couvre x, le modèle prévoit une action sans effets.

De manière à pouvoir comparer IRALe à MARLIE, nous utilisons dans un premier temps les mêmes mesures d'évaluation que Croonenborghs *et al.* (2007). On évalue

Algorithme 11 PREVISION(M,x) : $\hat{s}'$

Entrée: Un état s et une action a, un modèle d'action M
Sortie: Une prévision $\hat{s}'$ de l'exécution de a à partir de s

1: $x = s/a/effets_{prevus}$
2: **si** $\nexists r \in M.R, r \overset{sa}{\sim} x$ **alors**
3: $\quad effets_{prevus}.add = \emptyset$
4: $\quad effets_{prevus}.del = \emptyset$
5: **sinon**
6: $\quad$ sélectionner une règle $r \in M.R$ t.q $r \overset{sa}{\sim} x$ avec θ_r
7: $\quad$ % si plusieurs règles, en choisir une au hasard
8: $\quad effets_{prevus}.add = (r.e.add)\theta_r$
9: $\quad effets_{prevus}.del = (r.e.del)\theta_r$
10: **fin si**
11: $\hat{s}' = x.s \cup effets_{prevus}.add \setminus effets_{prevus}.del$

l'erreur du modèle en fonction du nombre d'atomes incorrectement prévus dans l'état résultant observé. On distingue des erreurs de deux sortes :

– en termes d'atomes observés dans l'environnement ($\in s'$) mais pas par le modèle ($\notin \hat{s}'$) ou faux négatifs (FN)
– en termes d'atomes prévus par le modèle ($\in \hat{s}'$) mais non observés dans l'environnement ($\notin s'$) ou faux positifs (FP).

On obtient un taux de FN et FP en normalisant relativement au nombre total d'atomes de s'. Pour chaque expérience, on calcule la moyenne des taux de FN et de FP sur 100 exemples.

Pour chaque expérience, après chaque pas d'apprentissage (ou action), on teste le modèle sur un échantillon de 100 exemples tirés aléatoirement. On calcule alors la moyenne des taux de FN et de FP sur ces exemples. On parle alors d'erreur de classification. Chaque courbe représente la moyenne de 100 expériences.

Dans un deuxième temps, on utilise une évaluation plus simple permettant de montrer uniquement si les prévisions sont correctes ($\hat{s}' = s'$) ou non. On procède de la même manière que précédemment en calculant la moyenne des prévisions incorrectes sur 100 exemples après chaque pas d'apprentissage. On parle alors d'erreur de prévision.

De plus, pour évaluer le modèle, on évalue aussi le nombre de contre-exemples en

mémoire. En effet, ce nombre représente également le nombre d'erreurs de prévision en phase d'apprentissage par IRALe. Il permet d'illustrer la complexité des domaines rencontrés.

Résultats

En premier lieu, nous comparons IRALe aux travaux de Croonenborghs *et al.* (2007), les plus proches des nôtres. MARLIE est en effet capable d'apprendre incrémentalement une fonction de transition relationnelle. Nous nous plaçons exactement dans le même cadre d'évaluation que Croonenborghs *et al.* (2007) dont les courbes sont reproduites figure 4.4.

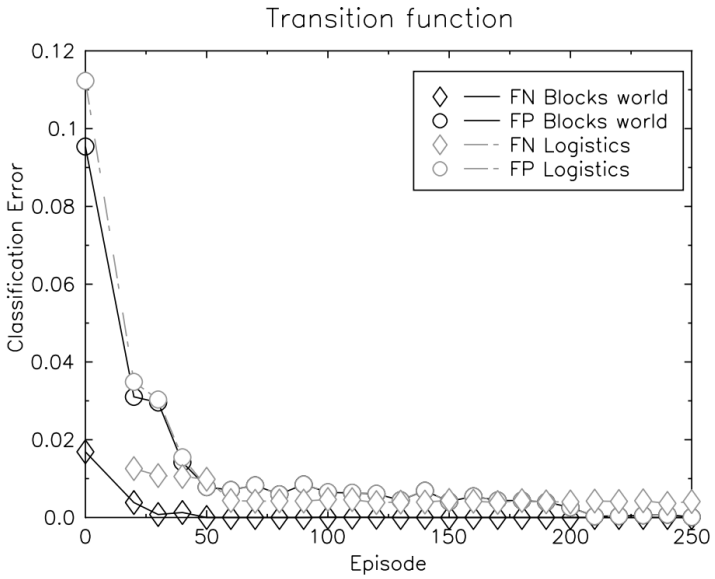


FIGURE 4.4.: Courbes extraites de Croonenborghs *et al.* (2007). Erreur de classification (*FP*, *FN*) dans un monde de 7 blocs et dans le domaine Logistique (5-5-5).

Avec IRALe, nous obtenons les résultats de la figure 4.5. Nous évaluons le nombre d'erreurs du système au fur et à mesure des épisodes d'apprentissage. Chaque épisode est composé de 20 actions.

Dans le domaine du *monde de 7 blocs* et *logistique(5,5,5)*, un modèle d'action est

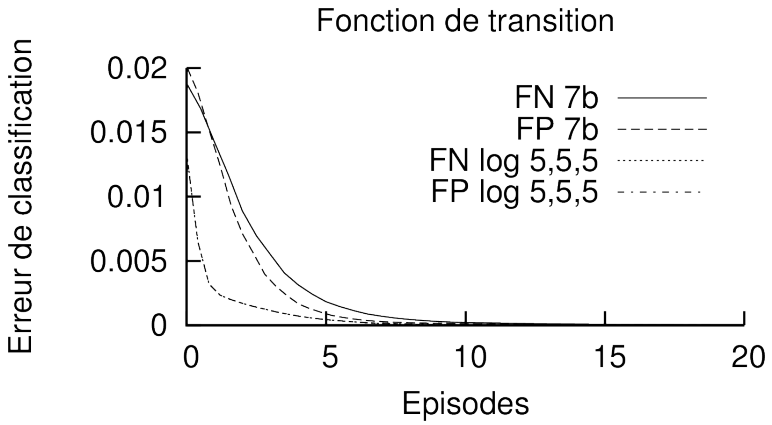


FIGURE 4.5.: Erreur de classification (*FP*, *FN*) dans un monde de 7 blocs et dans le domaine Logistique (5-5-5).

parfaitement appris après 20 épisodes. Dans IRALe, les erreurs FP se produisent sur la partie *add* de l'ensemble des effets et les erreurs FN sur la partie *del*. Contrairement à MARLIE, les courbes FN des faux négatifs et FP des faux positifs sont superposées pour le domaine logistique et assez proches sur le monde de blocs, ce qui peut s'expliquer pour IRALe car les parties *add* et *del* sont aussi liées lors de l'apprentissage. MARLIE est moins efficace sur les erreurs FP que sur les FN ce qui peut traduire un modèle trop général du fait de son approche descendante.

Pour les deux problèmes traités, IRALe apprend un modèle cohérent sur les exemples d'évaluation en moins de 20 épisodes, alors que MARLIE ne converge pas complètement même après 250 épisodes sur le domaine logistique.

Comme l'évaluation en terme d'erreurs de classification est normalisée au nombre total d'atomes, plus la représentation d'un état dans un domaine nécessite d'atomes, et plus l'erreur semble faible. Pour éviter ce problème, on utilise alors une évaluation en terme d'erreurs de prévision.

Les figures 4.6 , 4.7 et 4.8 illustrent les résultats obtenus en terme d'erreurs de prédiction sur les domaines des mondes des blocs, logistique et aussi un monde des blocs coloré en fonction de l'expérience de l'agent (ou nombre d'actions effectuées).

Chacun des environnements est aussi testé sur des espaces d'états plus vastes qu'en

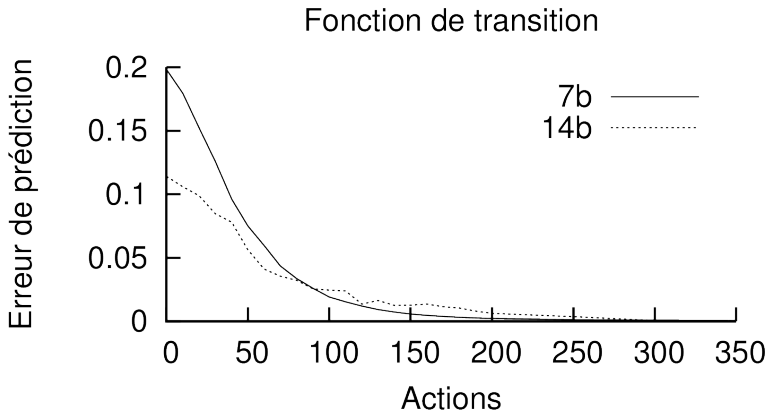


FIGURE 4.6.: Erreur de prévision dans un monde de 7 et 14 blocs.

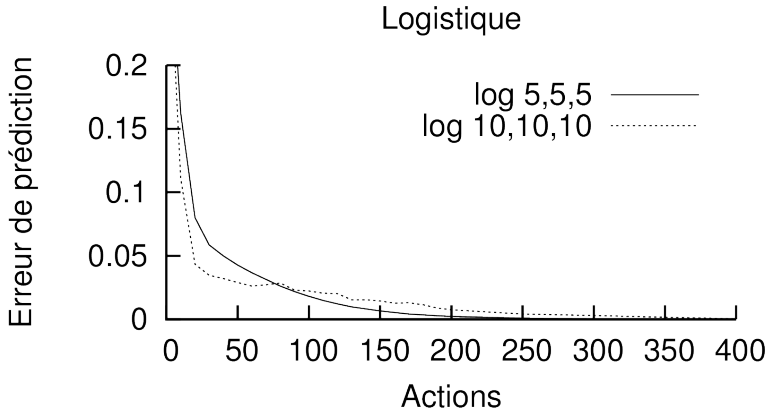


FIGURE 4.7.: Erreur de prévision pour logistique(5,5,5) et (10,10,10).

figure 4.5. IRALe converge relativement rapidement. On note que les courbes se croisent pour des domaines de taille différente. Une hypothèse est que la méthode généralise mieux sur des états plus grands représentés avec plus d'atomes, car les généralisations identifient alors mieux les points communs des actions impliquées dans le déroulement des actions par rapport aux atomes plus nombreux n'intervenant pas. Plus les états possèdent d'atomes et plus les atomes ne participant pas à l'action (par exemple des blocs d'une autre pile) sont nombreux et différents. Pour confirmer cette hypothèse, on évalue aussi IRALe en terme d'erreurs en apprentissage égale au nombre de contre-exemples rencontrés.

On note figure 4.9 que le monde des blocs colorés est plus complexe que les autres

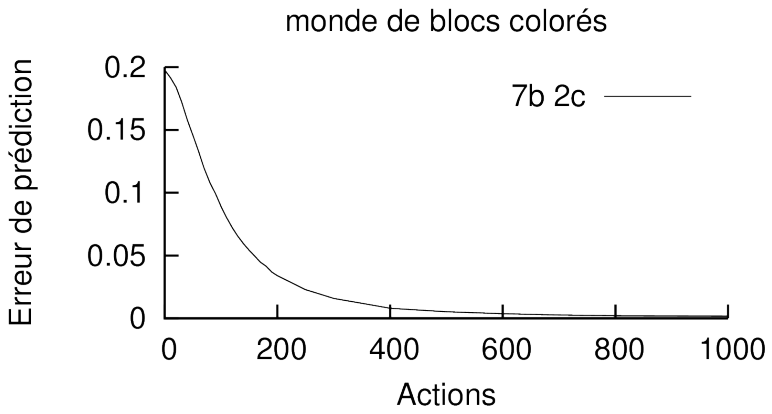


FIGURE 4.8.: Erreur de prévision pour un monde de 7 blocs et 2 couleurs.

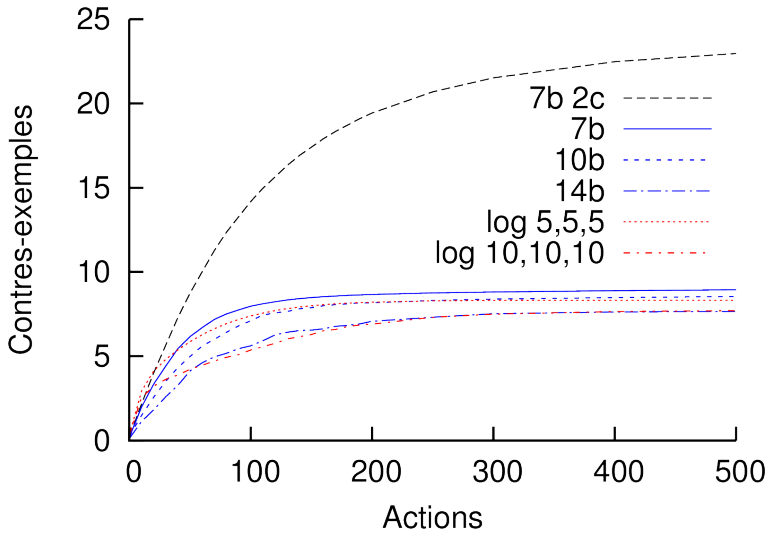


FIGURE 4.9.: Nombre de contre-exemples rencontrés dans les domaines étudiés.

domaines avec plus de 20 contre-exemples en mémoire, contre moins de 10 pour les autres domaines. On note également que le nombre de contre-exemples est assez proche pour le monde des blocs ou logistique. Cependant, sur ces deux domaines, on observe qu'IRALe rencontre moins de contre-exemples lorsque l'espace d'états est plus grand, ce qui permet de confirmer l'hypothèse émise sur les courbes précédentes. La généralisation est meilleure puisqu'elle nécessite moins de contre-exemples. Néanmoins, IRALe converge moins rapidement sur les environnements plus grands. En effet, même si les résultats sont meilleurs en début d'apprentissage (figure 4.6 et 4.7), les espaces

d'états étant plus grands, il est par la suite plus difficile de rencontrer les contre-exemples capables d'améliorer le modèle d'action. Nous traitons ce problème d'exploration au Chapitre 5.

Nous avons montré qu'IRALe était capable d'apprendre de façon efficace des modèles d'action. Nous allons nous comparer directement à une approche descendante.

4.5.4. Comparaison

Comme évoqué chapitre 3, IRALe n'est pas le premier système à apprendre un modèle d'action relationnel. Différentes approches expérimentales ont été développées, toutes ne partageant pas les mêmes hypothèses d'apprentissage comme l'incrémentalité, l'exploitation d'un modèle incomplet mais correct en début d'apprentissage, des actions aux effets stochastiques... La méthode la plus proche d'IRALe demeure le système MARLIE. En effet, comme IRALe, MARLIE explore un environnement déterministe de façon complètement autonome, sans aucune connaissance *a priori* sur la légalité ou les conséquences du choix de ses actions. De même, cette exploration n'est pas guidée par un oracle extérieur, mais suit une marche aléatoire. Toutefois, pour MARLIE, la sélection de l'action initialement focalisée sur l'exploration est au fil du temps réduite au profit de l'exploitation de sa fonction de récompense.

Protocole expérimental

Malheureusement, le système MARLIE n'étant pas disponible, nous avons développé une méthode similaire reposant sur des arbres de régression exactement de la même manière, afin de procéder à des tests comparatifs. Néanmoins, au lieu d'utiliser le système TG (section 3.2.3) comme arbre de régression relationnel en introduisant par la même occasion des paramètres supplémentaires, nous simplifions notre étude en utilisant l'algorithme Tilde (Blockeel *et al.*, 1998). L'intérêt d'utiliser Tilde est multiple car orthogonal à IRALe en différents points. Il permet de nous comparer à une méthode à la fois descendante et aussi hors-ligne. Tilde est disponible à partir de la plateforme de datamining ACE de l'équipe Machine Learning de l'université catholique de Leuven.

En effet, contrairement à l'algorithme TG, Tilde n'apprend pas un modèle incrémentalement, il faut donc le relancer à chaque pas d'apprentissage. Cette façon de procéder a déjà été utilisée par Dzeroski *et al.* (2001) dans son introduction du cadre RRL où Tilde est relancé à la fin de chaque épisode comme décrit section 3.2.3). Dans nos expérimentations, nous utilisons exactement les mêmes biais de recherches (*rmode*) que ceux définis par Croonenborghs (2009).

Afin de comparer notre approche à Tilde, il n'est pas possible de procéder comme le fait MARLIE avec TG. Tilde n'est pas incrémental ; si pour chaque nouvel exemple rencontré, il est relancé de zéro sur tous les exemples précédents, le temps d'apprentissage va constamment s'accroître et devenir rédhibitoire. Pour éviter cet écueil, nous nous proposons de procéder comme pour IRALe en réduisant la phase d'apprentissage uniquement aux exemples mal prévus par le modèle. Pour le reste, nous utilisons la même procédure utilisée par MARLIE comme détaillé section 3.2.3.

Les courbes sont issues du même protocole d'évaluation que pour l'erreur de prévision introduite section 4.5.3.

Résultats

Sur les différents environnements testés figure 4.10, 4.11 et 4.12, les courbes montrent qu'IRALe converge plus rapidement que Tilde incrémental. On note également à la figure 4.12 que sur le domaine logistique, Tilde ne converge pas. On retrouve ainsi le même phénomène que dans les expériences de Croonenborghs *et al.* (2007).

Les figures 4.13, 4.14 et 4.15 apportent plus de renseignements sur le fonctionnement des deux approches. Ainsi, pour chacun des domaines, IRALe nécessite toujours moins de contre-exemples que Tilde incrémental. De plus, sur le domaine logistique (figure 4.15), le nombre de contre-exemples croît constamment, ce qui montre les difficultés de méthode sur ce problème.

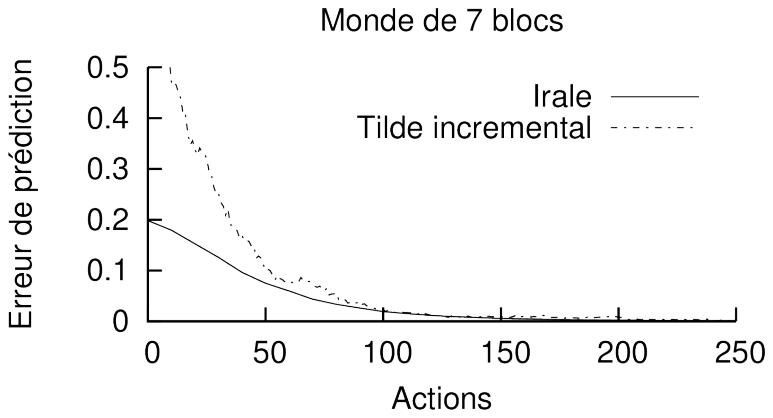


FIGURE 4.10.: Comparaison entre Tilde incrémental et IRALe sur un monde de 7 blocs.

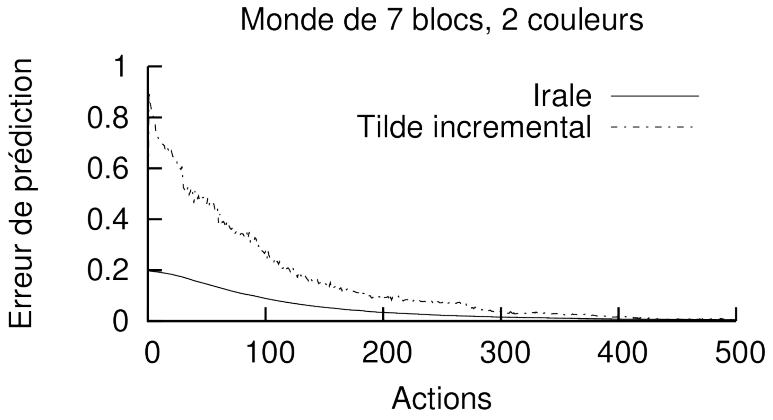


FIGURE 4.11.: Comparaison entre Tilde incrémental et IRALe sur un monde 7 blocs et 2 couleurs.

4.5.5. Planification

Jusqu'à présent, nous avons uniquement montré que notre méthode était bien capable d'apprendre un modèle d'action relationnel incrémentalement. Mais n'est-il pas plus pertinent de savoir si ce modèle est utilisable ? Dans l'affirmative, à partir de combien d'actions l'agent a-t-il acquis assez de connaissances sur son environnement pour l'utiliser ? Afin de répondre à ces questions, nous proposons d'utiliser le modèle d'action avec un planificateur et d'observer son comportement. En effet, le fait de représenter notre modèle à l'aide d'un formalisme reposant sur STRIPS permet de planifier directement à partir de celui-ci et par la même occasion d'être compatible avec la plupart des

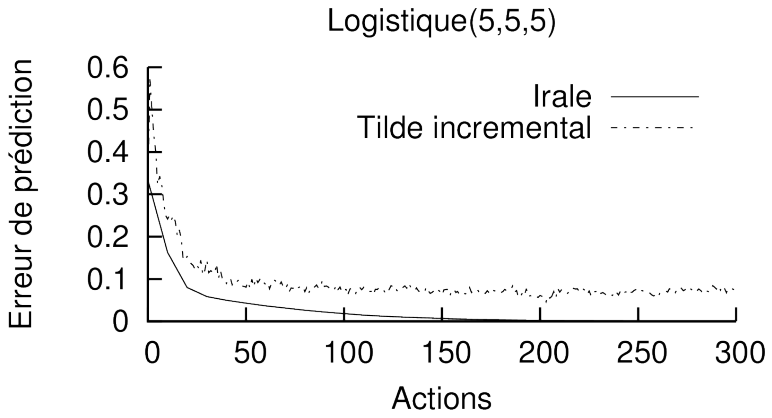


FIGURE 4.12.: Comparaison entre Tilde incrémental et IRALe sur logistique(5,5,5).

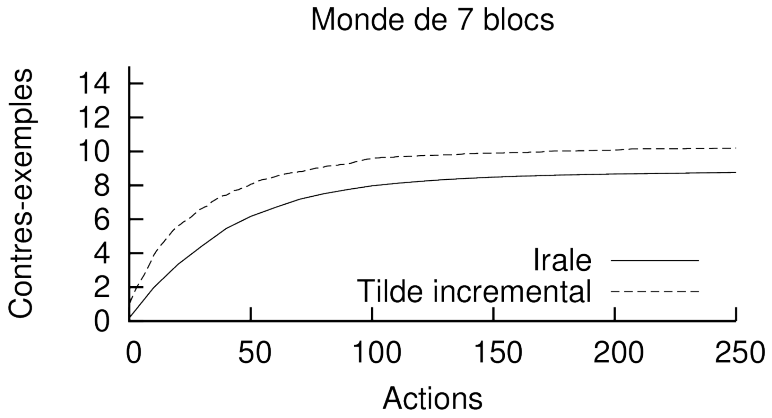


FIGURE 4.13.: Nombre de contre-exemples pour Tilde incrémental et IRALe sur un monde de 7 blocs.

planificateurs.

Protocole expérimental

Plus précisément, pour standardiser la description des domaines et des problèmes en planification, la communauté utilise notamment le langage PDDL (McDermott *et al.*, 1998) (Planning Domain Definition Language) comme lors de l'international planning conference 2011. Ce langage évolue avec la compétition et inclut notamment une description STRIPS qui représente uniquement une variation syntaxique du modèle

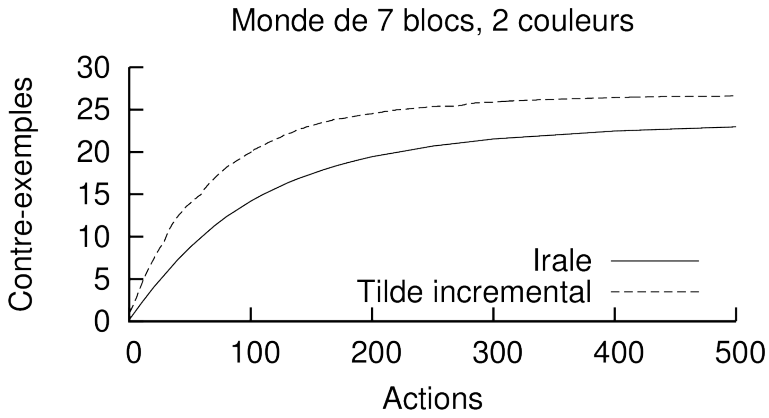


FIGURE 4.14.: Nombre de contre-exemples pour Tilde incrémental et IRALe sur un monde de 7 blocs et 2 couleurs.

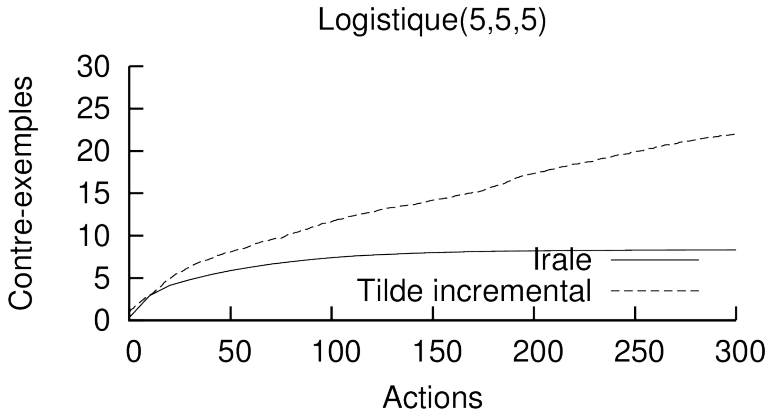


FIGURE 4.15.: Nombre de contre-exemples pour Tilde incrémental et IRALe sur logistique(5,5,5).

induit par IRALe.

D'autre part, le planificateur FF (Hoffmann et Nebel, 2001) est l'un des plus utilisés par la communauté et permet d'effectuer des comparaisons de différents systèmes dans le domaine de la planification.

Pour évaluer la qualité du modèle en utilisation, nous nous proposons de le donner en entrée au planificateur FF et d'analyser la qualité des plans obtenus.

Concrètement, nous tentons de comparer les plans trouvés par FF à l'aide du modèle

obtenu par IRALe aux plans trouvés à l'aide d'un modèle idéal écrit à la main (comme fourni en Annexe). De plus, cette comparaison est effectuée au fur et à mesure de l'apprentissage de l'agent afin d'illustrer l'évolution de l'efficacité du modèle. Ici, on ne cherche pas à évaluer les performances du planificateur, mais du modèle qui lui est fourni. Pour chaque comparaison, un état initial et un état final sont tirés aléatoirement, le planificateur FF doit tenter de trouver un plan dans un temps limité (10 secondes) pour le modèle d'IRALe et le modèle écrit à la main. Un plan est considéré comme réussi si le plan est correct et si l'exécution du plan dans l'environnement permet d'atteindre également le même état final. En effet, il est possible que le plan trouvé puisse être irréalisable ou avec des conséquences différentes dans l'environnement car trouvé à partir d'un modèle d'action incorrect. Il est donc nécessaire de vérifier la vraisemblance du plan trouvé en le confrontant à l'environnement.

Enfin, on définit cette comparaison par la distance de variation dv suivante :

$$vs = \begin{cases} \frac{\text{nb plans réussis avec le modèle courant}}{\text{nb plans réussis avec le modèle idéal}} & \text{si nb. plans avec modèle idéal} > 0 \\ 1 & \text{sinon} \end{cases}$$

$$dv = 1 - vs$$

Ainsi, si pour un modèle appris donné, il y a autant de plans réussis trouvés par FF qu'à partir d'un modèle idéal, alors $dv = 0$. Inversement, si tous les plans trouvés à partir d'un modèle idéal réussissent alors qu'aucun n'aboutit avec le modèle appris, alors $dv = 1$.

La figure 4.16 illustre l'évolution de la distance de variation, toutes les 10 actions d'apprentissage, on compare le modèle au modèle idéal sur 10 situations (état initial et final) aléatoires. Les courbes représentent la moyenne de 100 expériences.

Résultats

La figure 4.16 illustre le comportement de FF à partir des modèles d'actions en cours d'apprentissage sur les différents domaines. Les courbes montrent que le modèle d'action

est utilisable par le planificateur même si celui-ci est encore imparfait. Les courbes sont superposables aux courbes obtenues sur l'évaluation en terme d'erreur de prévision. Pour tous les domaines, après 200 actions le planificateur arrive à trouver des plans dans 90% des cas. Les courbes prouvent qu'IRALe est incrémental en apprentissage mais aussi à l'utilisation. Notre approche ascendante permet d'obtenir rapidement un modèle suffisamment correct pour être utilisé par un planificateur. De plus, comme l'apprentissage se fait directement sur les règles, le modèle est directement utilisable, contrairement aux travaux de Croonenborghs *et al.* (2007) ou de Mourão *et al.* (2012).

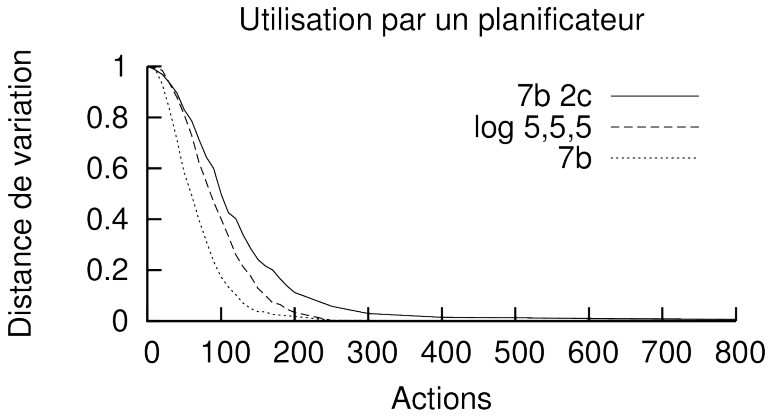


FIGURE 4.16.: Utilisation des modèles directement par le planificateur FF.

4.6. Conclusion

Dans ce chapitre, nous avons présenté l'approche originale IRALe capable d'apprendre incrémentalement des modèles d'action dans des environnements relationnels déterministes.

Les modèles d'action EDS sont plus expressifs que STRIPS car ils permettent de représenter des objets n'appartenant pas directement aux paramètres de l'action (Deictic references) et aussi de représenter pour un même symbole d'action des pré-conditions et des effets différents.

Nous avons apporté des garanties de convergence pour l'approche reposant sur les erreurs du modèle en conservant les contre-exemples associés.

Nous avons montré expérimentalement qu'IRALe peut apprendre efficacement un modèle d'action en réalisant très peu d'erreurs comparativement à MARLIE (Croonenborghs *et al.*, 2007) ou à l'adaptation Tilde incrémental. Nous avons aussi montré que les modèles appris sont directement utilisables par un planificateur même si IRALe n'a pas encore totalement convergé.

Les courbes d'évaluation ont permis de mettre en relief le problème de l'exploration de l'environnement. Ainsi, même si IRALe effectue moins d'erreurs sur un domaine plus vaste en généralisant mieux, il ne converge pas pour autant plus vite en terme de nombre d'actions. Pour limiter cet effet lié à la taille de l'espace d'états et à son exploration aléatoire, on se propose dans le chapitre suivant d'étudier une exploration active de l'environnement.

Chapitre 5

Apprentissage actif

Sommaire

5.1. Pourquoi faire de l'apprentissage actif ?

Jusqu'ici, notre approche s'est focalisée sur la réalisation d'un agent capable d'apprendre un modèle d'action en agissant sur son environnement. L'apprentissage se déroule au fur et à mesure des exemples rencontrés par l'agent. Afin de collecter les informations nécessaires à la génération d'un modèle, l'agent suit une politique aléatoire (cf. section 4.5.2). Avec une telle stratégie, l'agent n'utilise pas les connaissances acquises (son modèle d'action) qui lui permettraient par exemple de ne pas repasser plusieurs fois par des chemins déjà rencontrés. Ainsi, l'agent pourrait se focaliser sur des zones inexplorées et potentiellement riches en nouvelles informations.

5.1.1. Apprendre à agir et agir pour apprendre

Le problème d'exploration de l'environnement est un problème classique en apprentissage par renforcement (Sutton et Barto, 1998), plus connu sous le nom de dilemme *exploration contre exploitation*. Une des difficultés de l'apprentissage en ligne par un agent réside dans l'entremêlement des phases d'apprentissage et d'évaluation. L'agent ne peut pas seulement induire un modèle à partir des exemples d'apprentissage communiqués par son environnement, mais doit aussi agir pour les acquérir. Afin de parvenir à un but, un agent peut soit choisir d'exploiter ses connaissances acquises et prendre le risque qu'elles ne soient pas suffisamment complètes, soit explorer de nouvelles pistes en espérant améliorer ses connaissances. Il existe alors un compromis entre l'exploration de l'environnement pour l'acquisition de nouvelles connaissances et l'exploitation des connaissances acquises.

De nombreuses méthodes d'exploration existantes sont abordées par Wiering (1999). Nous introduirons les méthodes de sélection de l'action les plus utilisées, puis nous présenterons des méthodes actives plus informées utilisant par exemple le nombre de passages dans un état. Enfin, nous présenterons les différentes méthodes et adaptations dans des environnements relationnels.

5.1.2. Modèle et aléatoire

Une des méthodes les plus simples pour traiter le compromis exploration/exploitation est la méthode de sélection de l'action ϵ-*gloutonne*. Cette méthode permet de séparer explicitement d'une part une exploration complètement aléatoire, et d'autre part une exploitation de la meilleure politique connue.

Lors du choix d'une action, il y a une probabilité ϵ fixée de choisir une action aléatoire (*explorer*) et $1 - \epsilon$ de choisir la meilleure action (*exploiter*) selon la politique courante (cf. section 2.1).

$$\pi(s,a) \begin{cases} \text{arg } max_{a \in A(s)} Q(s,a) & \text{avec une probabilité 1-}\epsilon \\ \text{action } a \in A(s) \text{ aléatoire} & \text{avec une probabilité } \epsilon \end{cases}$$

En ordre un, les travaux de Dabney et McGovern (2007), par exemple, approximent une fonction valeur et utilisent une sélection ϵ-*gloutonne* de l'action. Cependant, dans cette méthode, le compromis exploration/exploitation est fixe et n'évolue pas en cours d'apprentissage.

Une autre méthode souvent employée, notamment en apprentissage par renforcement relationnel, est la sélection $Softmax$, ou plus précisément selon une distribution de $Boltzmann$. Elle est utilisée par exemple par Gartner *et al.* (2003) et Croonenborghs *et al.* (2007) et permet d'affiner empiriquement le compromis exploration/exploitation lors de l'apprentissage.

Avec une sélection ϵ-*gloutonne*, lorsque l'agent explore, l'action est choisie aléatoirement. Ce comportement peut engendrer la sélection d'actions dont la récompense est faible et déjà connue. Une sélection $Softmax$-$Boltzmann$ permet d'associer le choix aléatoire d'une action à la valeur estimée de celle-ci. Ainsi, les actions liées à des états à faible récompense estimée ont moins de chances d'être sélectionnées :

$$\pi(a,s) \begin{cases} \dfrac{e^{\frac{Q(s,a)}{\tau}}}{\sum_{a \in A(s)} e^{\frac{Q(s,a)}{\tau}}} \end{cases}$$

Le paramètre τ ou température permet de fixer la proportion d'exploration et d'exploitation. Ainsi, lorsque la température tend vers 0, la sélection de l'action devient gloutonne, alors qu'avec une température élevée, chaque action possède la même probabilité d'être sélectionnée. En pratique, la température évolue au cours de l'apprentissage. Contrairement à une sélection ϵ-*gloutonne*, cette méthode nécessite plusieurs paramètres difficiles à fixer empiriquement comme la valeur de départ de la température ainsi que son évolution à l'aide d'un facteur de décroissance.

5.1.3. Méthodes Actives

Les méthodes présentées ci-dessus reposent uniquement sur le hasard afin d'explorer l'environnement. Elles n'utilisent pas le modèle en cours d'acquisition pour chercher activement à l'améliorer. L'algorithme E^3 (Kearns, 1998) pour "Explicit, Explore and Exploit" est capable d'apprendre simultanément un modèle d'action et un modèle des récompenses. E^3 conserve des statistiques sur le nombre de visites des différents états et les probabilités de transitions entre états lui permettant de gérer explicitement le compromis exploitation/exploration. Ainsi, à partir d'un certain seuil de visites, un état est considéré comme connu. Dans un état inconnu, E^3 choisit arbitrairement une action à exécuter, mais lorsqu'il atteint un état déjà rencontré, E^3 choisit l'action la moins sélectionnée dans cet état.

Grâce aux statistiques accumulées, l'algorithme est alors face à deux alternatives : soit il estime que l'état est suffisamment connu et qu'il est alors possible d'exploiter une politique reposant sur tous les états connus (suffisamment visités). Soit il est nécessaire d'explorer davantage en tentant d'exécuter en priorité les actions menant à des états moins visités.

L'algorithme $Rmax$ (Brafman et Tennenholtz, 2003) utilise une approche similaire, mais utilise une démarche optimiste : il fixe initialement la récompense associée à chaque état au maximum possible ($Rmax$). Ainsi, il n'est plus nécessaire de gérer explicitement le compromis exploration/exploitation comme dans E^3. Le choix d'une action reposant sur l'espérance d'une forte récompense, l'exploration et l'exploitation sont traitées directement de la même manière. En effet, si l'état est connu et la récompense reçue est élevée, il s'agit dans ce cas d'une phase d'exploitation. Si la récompense espérée est élevée mais la récompense reçue est faible, il s'agit alors d'une phase d'exploration.

5.1.4. Actif et Relationnel

En apprentissage par renforcement, le problème de l'exploration est majeur, compte tenu des espaces rencontrés pouvant atteindre des millions d'états. Dans un cadre

relationnel, même s'il est possible de couvrir un grand nombre d'états avec une représentation compacte, la rencontre des exemples permettant de généraliser efficacement se pose aussi. Très peu de travaux se sont intéressés dans ce contexte au compromis exploration/exploitation. Certaines approches ont abordé la question de façon indirecte en tentant de réduire la part d'exploration dans l'expérience de l'agent, comme Driessens et Dzeroski (2004) qui ont exprimé le problème par la nécessité d'un « besoin de guidage ». L'idée est ici de fournir directement à l'agent quelques exemples issus de politiques d'experts. Plus récemment, Natarajan *et al.* (2011) vont plus loin en reposant l'apprentissage exclusivement à partir de trajectoires (séquences d'actions) fournies par un expert (*Imitation Learning*). Sans surprise, ces méthodes facilitent l'apprentissage par rapport à une exploration aléatoire puisqu'elles renoncent à la résolution automatique du problème.

D'autres travaux, comme Croonenborghs *et al.* (2006), ont étudié dans un cadre RRL l'utilité de l'apprentissage d'un modèle d'action (*cf section* 3.2.3). Par rapport à des méthodes d'apprentissage direct d'une fonction Qualité, ces travaux ont montré qu'au lieu d'exploiter la fonction Qualité apprise uniquement sur les actions possibles dans l'état courant (sélection ϵ-*gloutonne*), l'utilisation du modèle d'action appris permet d'anticiper (ou de planifier) (*lookahead*) sur plusieurs pas d'actions futures et de mieux estimer les récompenses, augmentant les chances de sélectionner les meilleures actions. Cependant, en pratique, la méthode est limitée à anticiper au maximum deux pas à l'avance, car l'utilisation de la fonction de transition apprise est dans ce cas un processus très lourd (il faut tester tous les atomes possibles de l'environnement afin de déduire les conséquences d'une seule action).

A notre connaissance, dans un cadre relationnel, la méthode REX (Lang, 2011) est la seule à directement orienter l'expérience de l'agent par une exploration active. REX repose sur l'algorithme de Pasula *et al.* (2007) pour l'apprentissage d'un modèle d'action non déterministe composé d'un ensemble de règles (cf section 3.2.1). REX adapte l'approche E^3 décrite plus haut à un modèle relationnel composé de règles.

L'algorithme original E^3 a une complexité polynomiale en le nombre d'états du monde et il garde trace des états déjà rencontrés. Cependant, E^3 est une approche tabulaire (il n'y a pas de généralisation du modèle d'action), ce qui le rend inapplicable dans

les grands espaces d'état. Cependant, dans un environnement relationnel, il n'est pas envisageable de pouvoir les traiter sans généralisation. Au lieu de conserver le nombre de visites des états, la méthode est transposée directement sur les règles du modèle d'action. Un poids est associé à chaque règle. Lorsqu'une transition est rencontrée, elle permet d'augmenter le poids de l'unique règle couvrant la transition. Ainsi, lors de la sélection d'une action, privilégier l'exploration consiste alors à utiliser de préférence une action couverte par une règle de poids minimal.

A chaque nouvel exemple (ou transition), REX reconstruit entièrement un modèle d'action à partir de tous les précédents exemples rencontrés. L'apprentissage du modèle d'action développé par Pasula *et al.* (2007) n'est en effet pas adapté pour un usage incrémental. Réeffectuer l'apprentissage sur un nombre croissant d'exemples ainsi que garder tous les exemples rencontrés sont des obstacles importants à l'application de cette méthode.

Contrairement aux travaux de Lang (2011) qui consistent à appliquer en priorité les règles les moins utilisées, notre approche essaye directement de généraliser les règles qui ne se déclenchent pas.

Le caractère ascendant de notre approche repose en premier lieu sur des généralisations plutôt que sur des spécialisations ; les problèmes rencontrés sont donc presque tous des problèmes de couverture du modèle et presque jamais des problèmes de cohérence. L'originalité de notre méthode d'apprentissage conduit donc à devoir reconsidérer des méthodes (Lang *et al.*, 2010) d'apprentissage actif, pourtant réputées générales à ce contexte.

5.2. Proposition d'apprentissage actif

Afin d'améliorer la convergence en termes d'actions nécessaires à exécuter pour apprendre un modèle correct et complet, nous nous proposons de développer une méthode d'exploration dite *active*. Ainsi, plutôt que de suivre une marche aléatoire, notre agent IRALe décrit au chapitre précédent doit être capable, lorsqu'il choisit une action, de

prendre en compte l'expérience accumulée dans le but d'améliorer l'apprentissage de son modèle d'action.

Plus concrètement, le modèle d'action induit par IRALe est composé d'un ensemble de règles. Lorsque l'agent se trouve dans un état donné, il est possible que certaines des règles de son modèle ne puissent se déclencher alors qu'elles pourraient expliquer la transition observée dans l'exemple. Lorsqu'une règle ne se déclenche pas, il peut y avoir deux raisons différentes. Les préconditions de la règle ne couvrent pas l'état soit parce que :

- les conditions de la règle ne s'appliquent pas dans cet état, qui décrit une situation différente. Par exemple, une règle permettant de décharger des paquets d'un camion alors que dans l'état courant le camion se trouve vide,
- la règle est trop spécifique et l'agent n'a pas encore rencontré les expériences qui lui permettraient de généraliser la règle. Une fois correctement généralisée, la nouvelle règle obtenue peut couvrir l'état afin de prévoir les effets de certaines actions dans celui-ci. Par exemple, une règle permettant de placer un certain bloc libre sur la table peut, après généralisation, s'appliquer cette fois à des blocs présents dans l'état courant.

Ce deuxième cas est la motivation de notre méthode d'exploration active. En effet, IRALe est une méthode incrémentale et principalement ascendante (cf. section 4.3.2). En cours d'apprentissage, les règles sont donc généralisées pas à pas et peuvent par conséquent demeurer trop spécifiques avant de converger totalement vers un modèle parfait. Avec l'expérience, le modèle de l'agent s'améliore en couvrant un nombre croissant d'exemples rendant plus rare la rencontre de contre-exemple menant directement à une mise à jour du modèle, surtout dans le cas d'environnements avec un très grand nombre d'états.

Par conséquent, il est utile de focaliser l'effort d'apprentissage sur les règles qui pourraient expliquer les effets observés. Une règle ne peut se déclencher à partir d'un état si les préconditions de la règle ne couvrent pas l'état.

La question que l'on se pose alors est : comment orienter l'expérience de l'agent afin de

généraliser activement le modèle d'action ?

Dans IRALe, pour généraliser une règle à l'aide d'un exemple, il est nécessaire qu'ils décrivent la même action (symbole de prédicat et arité identique) et que la règle puisse décrire les effets de l'exemple : la règle doit *post-généraliser* l'exemple (cf. algorithme 7, page 60).

On rappelle que l'apprentissage se fait à partir d'exemples sous forme de triplets (état, action, effets). Supposons un état donné et une règle ne pouvant pas prévoir d'effets à partir de cet état puisque les pré-conditions ne couvrent pas l'état. Pour tenter de généraliser activement cette règle, il faut choisir une action capable de mener à des effets qui seront post-généralisé par la règle, car la post-généralisation est un préalable à toute généralisation. En résumé, ce processus revient à construire un exemple complet à partir de l'état et du modèle courant dans le but d'obtenir un contre-exemple menant à une révision du modèle.

Pour ce faire, il est nécessaire de s'assurer que la règle puisse décrire des effets à partir de l'état courant. Le cas échéant, cela signifie qu'à partir de l'état courant, il n'est pas possible de tenter une exploration active de cette règle. Les effets des règles et des exemples suivent la même syntaxe STRIPS [1], ils sont composés de deux ensembles Add et Del, pour les atomes devenant respectivement vrai et faux après l'action, comme défini section 3.1. En conséquence, en suivant l'hypothèse du « *frame assumption* », ce qui d'après la règle devient vrai après l'action ne peut pas l'être avant l'action. Symétriquement, ce qui d'après la règle devient faux après l'action doit être vérifié dans l'état avant l'exécution de l'action. A l'aide de cette hypothèse, on fait le lien entre les effets de la règle et l'état courant en respectant le fait que la règle demeure *bien formée*. Ainsi, si l'ensemble *del* de la règle ne peut s'instancier dans l'état (on détaillera plus loin les différents tests appliqués), alors il n'est pas pertinent d'essayer de généraliser la règle et l'état.

Avant de rentrer plus en détails dans les différents aspects de notre méthode, nous commençons comme au chapitre 4 par en illustrer les mécanismes généraux à l'aide d'une

1. Syntaxiquement, seule la présence de variables distingue une règle d'un exemple

représentation propositionnelle permettant dans un premier temps de s'abstraire des problèmes de substitutions et de généralisation liés à une représentation relationnelle.

Règle	*Pré-conditions*	*Action*	$Effets$
r_1	p_1, p_2, p_3	a_1	$\neg p_2, p_4$
Exemple	*Etat*	*Action*	$Effets$
x_1	p_1, p_2	a_1?	???

TABLEAU 5.1.: Exemple d'exploration active dans un langage propositionnel.

Le tableau 5.1 illustre un exemple minimal d'exploration active avec une représentation propositionnelle. On suppose que le modèle de l'agent est constitué de la règle r_1 et que l'agent se trouve dans l'état $e_1 : p_1, p_2$. La règle r_1 n'est pas applicable car la proposition p_3 n'est pas vérifiée dans l'état e_1. Si celle-ci s'appliquait, elle permettrait de prévoir les effets de l'application de l'action a_1 à partir de cet état. Si cette supposition s'avère correcte, alors choisir d'exécuter l'action a_1 dans l'état e_1 mène aux mêmes effets que r_1 (on obtient l'exemple complet $x_1 = p_1, p_2; a_1; \neg p_2, p_4$). Pour vérifier cette hypothèse, l'agent soumet l'action a_1 à son environnement qui lui retourne les effets observés. Deux cas sont alors possibles : soit l'action a_1 est légale et permet à l'agent de changer d'état, soit elle n'a pas d'effet et dans ce cas l'exploration active n'est pas directement concluante mais peut permettre en fonction du modèle d' action courant, d'engendrer un contre-exemple, même sans effets.

Cependant, en cas de réussite, deux issues sont possibles relativement à l'adéquation ou non des effets observés et ceux anticipés par la règle. En effet, si les effets observés sont ceux anticipés : $\neg p_2, p_4$, alors r_1 ne pré-couvre pas l'exemple obtenu mais le post-généralise rendant possible la généralisation des pré-conditions de r_1 en abandonnant la proposition p_3.

L'autre possibilité est d'observer des effets non vides mais différents de ceux de r_1 pouvant entraîner éventuellement une mise à jour du modèle. Néanmoins, l'exploration active permet au moins dans ce cas un changement d'état de l'agent à partir duquel il sera possible de tenter une nouvelle exploration active. Cet exemple montre que la méthode d'exploration active que nous développons tente d'anticiper des généralisations du modèle à partir de celui-ci et de l'état courant. L'algorithme 12 illustre les principaux mécanismes de l'exploration active.

Algorithme 12 Sélection-Active(M, s) : L_a

Entrée: Un modèle d'action M et l'état courant s de l'agent
Sortie: Une liste d'actions potentiellement capables de généraliser des règles de M

```
Dans un état s donné :
pour tout règle ne se déclenchant pas faire
    si la partie del de la règle couvre l'état s alors
        calculer une généralisation des préconditions de la règle et l'état s
        si la partie add de la règle ne couvre pas l'état s alors
            déduire de la généralisation une action a
            rajouter a à la liste des actions possibles
        fin si
    fin si
fin pour
```

L'exploration active tire profit de la syntaxe des règles. On adapte la définition 9 d'une règle *bien formée* (cf. section 3.1) au cadre propositionnel. Une règle est alors *bien formée* ssi :

i) $r.e.del \subseteq r.p$

ii) $r.e.add \cap r.p = \emptyset$

L'exploration active tente de produire des généralisations, mais chaque généralisation doit rester *bien formée*. C'est pourquoi, afin d'en augmenter le succès parmi le nombre d'actions possibles à explorer, on vérifie pour un état s et une règle r que les conditions *i)* et *ii)* soient vérifiées. Dans l'exemple du tableau 5.1, avec e_1 et r_1, *i)* et *ii)* sont vraies.

Avec une représentation relationnelle, la condition *i)* réduite plus haut à un test d'inclusion devient plus complexe. La règle composée de relations et d'objets peut contenir des variables ; on calcule donc toutes les généralisations possibles entre $r.e.del$ et s. Comme discuté section 2.2.3, cette opération peut avoir un coût exponentiel, mais dans la pratique $r.e.del$ est très petit (deux atomes au maximum dans les domaines traités). Pour chaque généralisation Gen_{del_j} et substitution θ_{r_j} associée, si $(Gen_{del_j})\theta_{r_j} = r.e.del$, il est possible que la condition *i)* soit vérifiée, on peut alors tenter de généraliser les pré-conditions de la règle r, on obtient gen_{pre_j}.

Si $(r.e.add)\theta_{r_j} \not\subseteq s$, alors la condition *ii)* est vérifiée. Enfin, si la subsitution obtenue θ_{r_j} permet d'instancier $r.a$ afin d'en déduire une action à exécuter, alors celle-ci est ajoutée

à L_a. Le processus est répété pour chaque règle du modèle qui n'est pas plus générale que l'état courant. Malgré les tests nécessaires pour générer activement des actions exploratoires, ceux-ci ne sont pas suffisants pour garantir une révision du modèle ou même le succès de chaque action.

Algorithme 13 Sélection-Active(M, s) :L_a

Entrée: Un modèle d'action M, et un état s
Sortie: Une liste L_a d'actions susceptibles de généraliser des règles de M

$L_a \leftarrow \emptyset$
pour tout $r \in M.R$ t.q $\neg(r.p \geq_{OI} s)$ **faire**
  pour tout θ_{r_j} t.q $Gen_{del_j} \in GEN_{OI}(r.e.del, s)$ avec $(Gen_{del_j})\theta_{r_j} = r.e.del$ **faire**
    $gen_{pre_j} = UNE\text{-}GEN_{OI}(r.p, s, \theta_{r_j}, \emptyset)$
    si $(r.e.add)\theta_{r_j} \not\subseteq s$ et $(r.a)\theta_{r_j}$ est complètement instanciée **alors**
      $L_a \leftarrow L_a \cup ((r.a)\theta_{r_j}, longueur(gen_{pre_j}))$
    fin si
  fin pour
fin pour
si $L_a = \emptyset$ **alors**
  Sélectionner aléatoirement une action a à appliquer dans l'état s
sinon
  Sélectionner a telle que $(a, longueur) \in L_a$ et $longueur$ soit maximale dans L_a
fin si

De plus, la méthode peut indiquer plusieurs actions à explorer et il est nécessaire d'en sélectionner une. On décide de privilégier les actions associées aux généralisations ayant le plus grand nombre de littéraux, en supposant qu'elles sont plus prudentes. Une autre façon de faire, plus précise mais plus complexe, serait de calculer pour chaque généralisation *gen* un score en fonction du nombre de variables introduites et de littéraux oubliés. La liste L_a des actions à explorer n'est pas supprimée une fois qu'une action sélectionnée est exécutée par l'agent. Si l'action s'avère sans effets, l'agent demeure dans le même état. Si on rappelle le processus d'exploration active, il n'est pas nécessaire de refaire les mêmes calculs ; il suffit de sélectionner une action parmi celles restantes dans L_a.

Après cette étape, si la post-généralisation est *a priori*[2] possible entre la règle et l'état, on anticipe l'existence d'une généralisation entre ces deux derniers. A chaque étape,

2. La post-généralisation est une condition nécessaire à la généralisation mais non suffisante pour en garantir le succès.

des objets (variables ou constantes) de la règle sont appariés avec les constantes de l'état courant. De ces appariements, on déduit des instanciations de l'action de la règle. Finalement, on obtient des actions que l'agent pourra exécuter sur son environnement.

Exemple 7 *On considère le problème logistique. On suppose que le monde est composé de trois camions, trois colis et trois villes et un modèle courant M. Soit $r \in M.R$ la règle suivante :*

$$r : \; boxOnTruck(b_2, c_a), boxInCity(b_1, c_a), truckInCity(T_b, c_a) \; / \\ load(b_1, T_b) \; / \; boxOnTruck(b_1, T_b), \neg boxInCity(b_1, c_a)$$

On suppose que l'agent se trouve dans l'état s suivant :

$$s : \; boxInCity(b_1, c_b), truckInCity(t_b, c_b), boxInCity(b_2, c_a)$$

La règle r ne s'applique pas car il n'existe pas de littéral $boxOnTruck(b_2, c_a)$ dans l'état courant s (la condition ligne 2 de l'algorithme 13 est vérifiée).

Il y a deux généralisations possibles de s et r.e.del : $Gen_{del_1} = boxInCity(b_1, Y)$ et $Gen_{del_2} = boxInCity(X, c_a)$ avec respectivement $\theta_{r_1} = \{Y/c_a\}$ et $\theta_{r_2} = \{X/b_1\}$. On calcule à partir de chaque substitution une généralisation des pré-conditions de r. On obtient $gen_{pre_1} = TruckInCity(T_b, Y)$ avec $\theta_{r_1} = \{Y/c_a, T_b/t_b\}$ et $gen_{pre_2} = boxInCity(X, c_a)$ avec θ_{r_2} inchangée. Dans les deux cas $(r.e.add)\theta_r \not\subseteq s$, on obtient alors $(r.a)\theta_{r_1} = load(b_1, t_b)$ et $(r.a)\theta_{r_2} = load(b_2, T_b)$ qui n'est pas complètement instanciée. L'agent applique l'action $load(b_1, t_b)$ et l'état résultant $s' =$ $boxOnTruck(b_1, t_b), truckInCity(t_b, c_b), boxInCity(b_2, c_a)$ crée un exemple qui, comme attendu, n'est pas couvert par le modèle d'action courant. La révision consiste alors à généraliser la règle r : le littéral $boxOnTruck(b_2, c_a)$ est supprimé des pré-conditions de r.

En résumé, cette méthode d'exploration active est une procédure de généralisation du modèle à l'aide d'un exemple (S/A/E) incomplet. Seul l'état S est connu et non couvert par la règle. Les effets E sont alors construits pour autoriser la généralisation et en déduire une action. Ainsi, l'agent pourra exécuter une action choisie en fonction

de son modèle courant qui permettra éventuellement de l'améliorer. L'algorithme 12 donne une vue d'ensemble de la méthode d'exploration active.

Si la méthode permet de proposer des actions candidates à partir d'un état et du modèle courant de l'agent, ce dernier se retrouve alors confronté à un choix : quelle action exécuter parmi celles générées sachant que tant qu'une action n'est pas exécutée, nous n'avons pas de garantie sur son succès ?

Nous avons testé différentes heuristiques, comme ne calculer qu'une seule généralisation aléatoire entre l'état courant et les *del* des règles ou encore choisir l'action à explorer au hasard parmi celles déduites. Nous avons aussi testé de privilégier les actions déduites des règles les moins utilisées comme dans les travaux de Lang (2011). L'heuristique retenue dans le système prend en compte la longueur du moindre généralisé obtenu entre les préconditions de la règle et l'état. Il permet de formaliser l'intuition qu'une action a d'autant plus de chances de réussir que les points communs entre la règle et l'état qui l'ont engendrée sont nombreux. On privilégie encore la notion de « moindre »généralisation. Il s'agit d'un indicateur très simple mais peu fin, car il ne prend pas en compte l'introduction de nouvelles variables. Néanmoins, dans la pratique, cette approximation s'avère suffisante et rapide à calculer. Elle se résume alors à compter les littéraux du moindre généralisé obtenu.

5.3. Etude expérimentale

Afin d'évaluer empiriquement l'intérêt de notre approche d'exploration active, nous nous plaçons dans le cadre expérimental décrit section 4.5, où nous mesurons l'efficacité des modèles en fonction du nombre d'actions rencontrées.

On évalue alors les modèles appris en les utilisant directement avec le planificateur FF comme décrit section 4.5.5. On compare alors la qualité du modèle obtenu par IRALe en suivant une marche aléatoire ou une marche active. On décide d'utiliser une politique ϵ-*gloutonne* pour introduire un compromis entre l'exploration aléatoire et l'exploration active. On présente les résultats avec différents seuils d'exploration

active, à partir d'un taux ϵ_a de 0 pour une marche exclusivement aléatoire jusqu'à un taux de 0.5 où une action est choisie aléatoirement ou activement dans les mêmes proportions.

5.3.1. Evaluation

Sur les différents environnements testés aux figures 5.1, 5.2 et 5.3, l'utilisation d'une part d'exploration active permet pour un même nombre d'actions effectuées d'obtenir des modèles de meilleure qualité, ce qui permet de réduire le nombre d'interactions nécessaires avec l'environnement. Cependant, on note aussi que l'utilisation d'une forte proportion d'actif (50%) ou une proportion beaucoup plus faible (25%) engendre des modèles de qualité proche. Cela signifie que choisir un nombre d'actions d'exploration active au-delà d'un certain seuil n'est pas une meilleure stratégie que de choisir des actions aléatoires, puisque malgré leur quantité doublée (de $\epsilon_a = 0.25$ à 0.5) la qualité des plans reste similaire. On note également que l'écart entre l'exploration complètement aléatoire et partiellement active se réduit au fur et à mesure que la qualité du modèle augmente, car les opportunités de nouvelles généralisations diminuent alors.

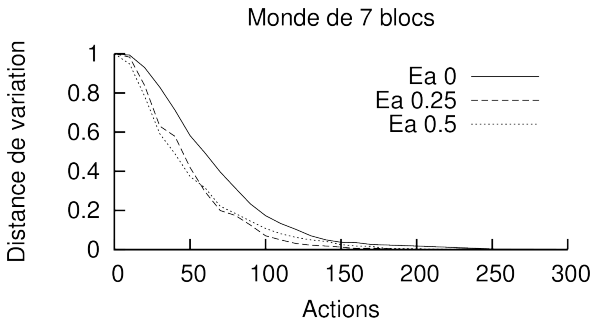


FIGURE 5.1.: Evaluation en utilisation par FF sur un monde de 7 blocs avec différents taux ϵ_a d'exploration active.

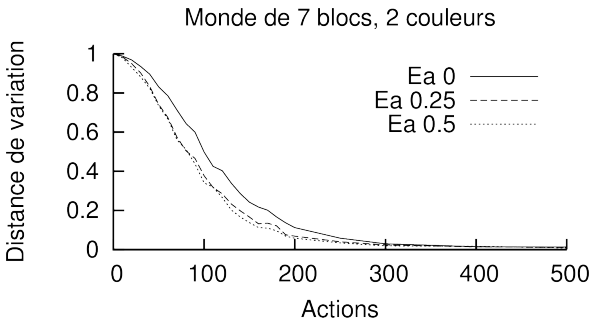


FIGURE 5.2.: Evaluation en utilisation par FF sur un monde de 7 blocs et 2 couleurs avec différents taux ϵ_a d'exploration active.

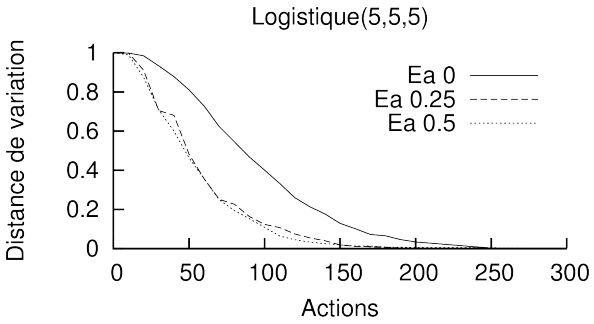


FIGURE 5.3.: Evaluation en utilisation par FF sur le domaine logistique(5,5,5) avec différents taux ϵ_a d'exploration active.

Les figures 5.4, 5.5 et 5.6 illustrent le nombre de contre-exemples moyens rencontrés au fur et à mesure de l'expérience acquise. Elles permettent de montrer clairement et directement l'impact de l'exploration active sur les modèles d'action induits. Pour chaque environnement testé et pour un nombre d'actions donné, on note qu'avec l'exploration active, le système rencontre toujours plus de contre-exemples. L'exploration active conduit l'agent à commettre plus d'erreurs, cela signifie que la méthode influence positivement l'exploration de l'espace en focalisant le choix de l'action vers des zones où le modèle est supposé être incomplet.

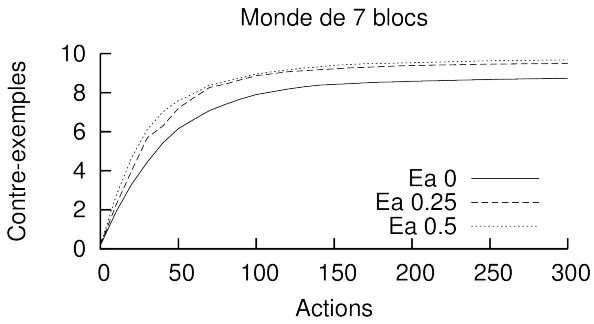


FIGURE 5.4.: Evolution du nombre de contre-exemples sur un monde de 7 blocs avec différents taux ϵ_a d'exploration active.

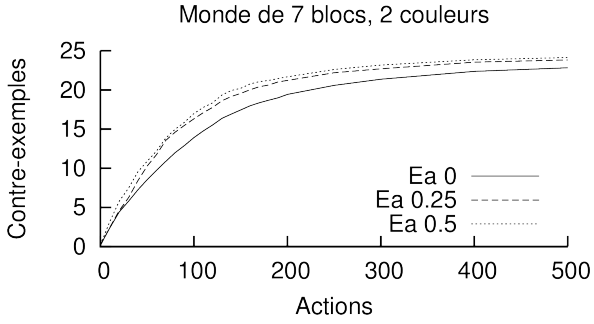


FIGURE 5.5.: Evolution du nombre de contre-exemples sur un monde de 7 blocs et 2 couleurs avec différents taux ϵ_a d'exploration active.

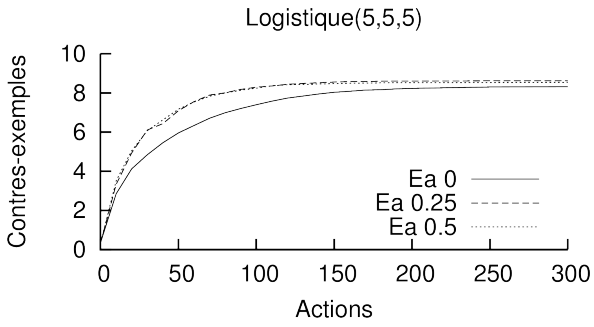


FIGURE 5.6.: Evolution du nombre de contre-exemples sur le domaine logistique(5,5,5) avec différents taux ϵ_a d'exploration active.

5.4. Conclusions

Dans ce chapitre, nous avons introduit une méthode d'apprentissage actif capable d'améliorer considérablement l'apprentissage en ligne de l'agent. Cette méthode est adaptée à l'agent IRALe et exploite le caractère ascendant de la généralisation ainsi que les caractéristiques du modèle d'action. La proposition n'est pas directement adaptable à tout autre agent ; elle repose cependant sur un principe général qui consiste à anticiper à partir d'informations partielles (état courant) les évolutions possibles du modèle courant. Par rapport à l'existant, nous avons développé une approche fonctionnant en ligne sans réapprentissage du modèle contrairement aux travaux de Lang *et al.* (2010). La méthode que nous avons développée reste simple est pourrait être plus sophistiquée. Par exemple, la méthode n'anticipe qu'une action et uniquement à partir de l'état courant. Cependant, il serait envisageable d'exploiter le modèle d'action (comme le fait un planificateur) pour se diriger vers certaines zones de l'environnement et ensuite tenter d'y explorer activement.

Chapitre 6

Apprentissage d'un modèle d'action en présence de bruit de perception

Sommaire

Dans ce chapitre, nous présentons une adaptation de notre approche IRALe à des problèmes plus réalistes. La supposition que l'agent perçoit toujours parfaitement tout son environnement à chaque instant est en effet difficilement envisageable dans un problème réel. Pour s'approcher de situations plus concrètes, on se propose d'introduire du bruit dans la perception que l'agent se fait du monde. Nous présentons notre démarche (Rodrigues *et al.*, 2010b) fondée sur des généralisations et des spécialisations retardées ainsi que les résultats empiriques obtenus.

6.1. Travaux connexes

L'apprentissage d'un modèle d'action relationnel dans un environnement bruité a déjà été abordé pour la planification, notamment avec les travaux de Pasula *et al.* (2007). Le bruit traité dans ces travaux représente un indéterminisme sur les effets des actions.

Cette méthode est illustrée sur un monde de blocs simulé en 3D, implémenté à l'aide d'un moteur physique réaliste, et simulant un bras robotisé. A cause de l'imprécision des mouvements du bras, le fait de déposer un bloc sur une pile peut alors échouer, le bloc pouvant tomber à terre ou encore faire tomber toute la pile. Le modèle appris représente alors les différents effets des actions avec leurs probabilités respectives.

D'autres travaux ont traité du bruit, comme le système ARMS (pour Action-Relation Modelling System) de Yang *et al.* (2007). Le bruit étudié est lié à la perception des états. Les états perçus par le système sont corrects, mais peuvent être incomplets. Ces deux systèmes n'apprennent pas au fur et à mesure d'interactions avec un environnement, mais hors ligne, à partir de traces d'expériences fournies par lot. Les travaux de Mourão *et al.* (2012) développent l'apprentissage en ligne d'un modèle d'action en présence de bruit. Cependant, le bruit est appliqué de façon ad hoc à la représentation de l'agent et n'est pas issu de la perception de l'environnement de celui-ci. De plus, la méthode n'est testée que sur des taux de bruits très faibles (1 et 5%). En regard du système ARMS et des travaux de Mourão *et al.* (2012), nous proposons de considérer une forme de bruit plus générale. En effet, en plus de traiter une perception incomplète de l'environnement par l'agent, nous ajoutons la possibilité de présenter une perception incorrecte. Ainsi, nous nous proposons de simuler des capteurs de perception imparfaits. L'agent doit alors apprendre à partir d'observations éventuellement incomplètes et/ou incorrectes.

6.2. Exemple de bruit de perception

Supposons que nous observons un monde de blocs (cf. sec.4.5.1 et annexe A) composé de 3 blocs et d'une table. Après avoir rencontré plusieurs exemples, supposons que l'agent a induit les deux règles (correctes) suivantes :

$$r_a^n : \begin{array}{l} cl(X), cl(Y), on(X,Z), on(Y,floor), bl(X), bl(Y), bl(Z) \ / \\ move(X,Y) \ / \ on(X,Y), cl(Z), \neg cl(Y), \neg on(X,Z) \end{array}$$

pour poser un bloc libre X au dessus d'un bloc Y libre. Supposons que l'exemple bruité

suivant se présente :

$$x_1 : \begin{array}{l} cl(a), on(a,b), on(b,c), on(c,floor), bl(a), bl(b), bl(c) \ / \\ move(a,floor) \ / \ on(a,floor), cl(b), \neg\mathbf{cl}(\mathbf{floor}), \neg on(a,b) \end{array}$$

Si le prédicat **cl**(**floor**) fait partie de $x_n.e.del$, c'est que $cl(floor)$ a été observé dans l'état précédant l'action. Cependant, celui-ci n'a pas d'existence réelle. x_1 n'est pas *pré-couvert* par r_a, mais cependant r_a *post-couvre* x_1. Il est alors possible de généraliser r_a^n à partir de x_1, nous obtenons la généralisation suivante :

$$r_a^{n+1} : \begin{array}{l} cl(X), on(X,Z), bl(X), bl(Z) \ / \\ move(X,Y) \ / \ on(X,Y), cl(Z), \neg cl(Y), \neg on(X,Z) \end{array}$$

Supposons maintenant que l'exemple suivant soit rencontré :

$$x_2 : \begin{array}{l} cl(a), on(a,b), on(b,c), on(c,floor), bl(a), bl(b), bl(c) \ / \\ move(a,c) \ / \end{array}$$

x_2 est un exemple parfaitement perçu par l'agent mais sans effets (il n'est pas possible de déplacer a sur c car ce dernier n'est pas libre). r_a^{n+1} *pré-couvre* x_2 et peut alors lui prédire des effets, ce qui engendre une contradiction. Nous notons toutefois qu'avant la rencontre de l'exemple bruité x_1, la généralisation antérieure r_a^n ne pré-couvre pas x_2. La mise à jour du modèle à partir d'exemples bruités peut être source d'erreurs.

6.3. Généralisations et spécialisations retardées

Contrairement au cas déterministe, les exemples peuvent être contradictoires. Par exemple, deux exemples possédant exactement les mêmes préconditions mais des effets différents peuvent se présenter car du bruit s'est manifesté sur les effets de l'un d'entre eux. Ainsi, les règles ne peuvent être spécialisées immédiatement à chaque fois qu'elles sont contredites par un exemple éventuellement bruité. Les exemples bruités peuvent aussi conduire à des sur-généralisations. Ainsi, les opérations de spécialisation et de généralisation doivent être effectuées avec "précaution", un exemple bruité pouvant remettre en cause une règle correcte. C'est pourquoi nous proposons un algorithme

capable de retarder les opérations de généralisation et de spécialisation afin de rendre l'apprentissage plus robuste. Pour ce faire, nous adjoignons des estimateurs à chaque règle. Les estimateurs sont ensuite combinés afin d'évaluer l'intérêt d'une mise à jour du modèle. A chaque règle r dans M, on associe trois estimateurs :

- $r.n_{sa}$ le nombre d'exemples *pré-couverts* par la règle depuis sa création
- $r.n_{ae}$ le nombre d'exemples *post-généralisés* par la règle depuis sa création
- $r.n_{sae}$ le nombre d'exemples couverts par la règle depuis sa création

Ces estimateurs sont mis à jour pour chaque règle chaque fois qu'un exemple d'apprentissage est présenté. La valeur initiale pour $r.n_{sa}$, $r.n_{ae}$ et $r.n_{sae}$ est 1.

$(r.n_{ae} - r.n_{sae})$ représente le nombre de fois qu'une règle a *post-généralisé* un exemple sans le couvrir. Si $r.n_{ae}$ est grand relativement au nombre d'exemples couverts, alors on peut supposer que la règle doit être généralisée. Normalisé par le nombre de *post-généralisations* de la règle, on obtient l'estimateur de généralisation suivant : $r.gen = \frac{r.n_{ae} - r.n_{sae}}{r.n_{ae}}$ avec $r.gen \in [0, 1]$

De même, une règle doit être spécialisée si ses prévisions sont souvent incorrectes. C'est le cas si une règle a prévu correctement peu de fois par rapport au nombre total de fois où elle a été déclenchée. On obtient l'estimateur de spécialisation suivant : $r.acc = \frac{r.n_{sae}}{r.n_{sa}}$ avec $r.acc \in]0, 1]$

Enfin, pour décider de la mise à jour d'une règle, on définit les paramètres θ_{gen} et θ_{spc} déterminant les seuils à partir desquels les estimateurs déclencheront respectivement des généralisations et des spécialisations. Néanmoins, ces estimateurs ne peuvent être pertinents que si les règles auxquelles ils sont associés ont été suffisamment évaluées. On introduit donc un troisième paramètre au système : θ_{evl} (précisant le nombre d'évaluations nécessaires avant de prendre une décision). Ainsi, pour une règle donnée, on décide de :

- généraliser si $r.gen > \theta_{gen}$ et $r.n_{ae} > \theta_{evl}$ (souvent *post-généralisée*)
- spécialiser si $r.acc < \theta_{spc}$ et $r.n_{sa} > \theta_{evl}$ (souvent *pré-couverte*)

Cette première approche demeure globalement la même que dans l'agent IRALe (*cf. Chap.4*). Lorsqu'un nouvel exemple n'est *pré-couvert* par aucune règle, le système

cherche parmi l'ensemble des règles une règle *post-généralisant* l'exemple afin de la généraliser. Si une telle règle n'existe pas, l'exemple est ajouté tel quel. Ainsi, le nouveau modèle couvre l'exemple. Néanmoins, en présence de bruit, ces opérations sont modérées par les estimateurs afin de réduire la capacité d'un seul exemple à remettre une ou plusieurs règles en cause. De plus, ce mécanisme en présence de bruit dans la perception des exemples va produire des règles erronées qu'il sera difficile de généraliser, et que le système doit être capable d'identifier et de supprimer. Pour limiter la complexité du modèle appris, on borne le nombre maximal de règles dans $M.R$ à l'aide d'un paramètre N. Si la borne maximum(N) de l'ensemble de règles est atteinte, l'ajout d'une nouvelle règle sera tout de même possible, en écartant du modèle la règle dont la confiance est la plus faible. A chaque règle, on associe donc une confiance $r.cnf$. Cette confiance est élevée si la règle a été suffisamment utilisée avec succès (la règle est précise) :

$$r.cnf = \begin{cases} r.acc & \text{si} \quad r.n_{sa} \geqslant \theta_{evl} \\ 0 & \text{sinon} \end{cases}$$

Les mécanismes décrits plus haut pour résister au bruit ont aussi pour effet d'admettre des règles contradictoires au sein du modèle. Ainsi, si l'on présente un couple ($état$,$action$) au modèle et que l'on interroge celui-ci sur le nouvel état résultant, plusieurs règles contradictoires sont susceptibles de se déclencher et de fournir chacune une réponse. Dans ce cas, le système sélectionne la règle possédant la confiance la plus élevée.

6.4. Algorithme détaillé

IRALe demeure dans les grandes lignes la même approche que celle décrite à l'Algorithme 4 page 54. Cependant, comme les exemples peuvent être contradictoires, il n'est plus possible d'assurer la cohérence du modèle par rapport à ceux-ci. Ainsi, pour que la taille du modèle n'explose pas, on ne garde plus en mémoire tous les contre-exemples rencontrés depuis le début de l'apprentissage. Pour ce faire, on limite le nombre de règles possibles dans $M.R$. S'il est nécessaire d'ajouter une nouvelle règle et qu'on a atteint le nombre de règles maximum, les règles considérées comme moins bonnes

($r.conf$) sont alors oubliées, ainsi que les contre-exemples associés à la généralisation de ces règles.

Algorithme 14 APPRENTISSAGE(M,x)

Entrée: Un exemple x, un modèle d'action M
Sortie: Un modèle M éventuellement révisé

$L_x \leftarrow \emptyset$
$\forall r \in M$, MAJ-ESTIMATEURS(r, x)
si $\neg$ PREVISION(M, x) **alors**
  pour tout $r \in M.R$ telle que $r \nsim x$ **faire**
    $L_x \leftarrow L_x \cup$ SPECIALISER(M, x, r)
  fin pour
  $L_x \leftarrow L_x \cup \{(x, \emptyset)\}$ **si** $x.e \neq \emptyset$
  $\forall (x_{tab}, r_{tab}) \in L_x$, $GENERALISATION_{REL}(M, x_{tab}, r_{tab})$
fin si

En outre, avant de spécialiser ou de généraliser, on évalue les différentes statistiques obtenues sur les règles afin de décider s'il est nécessaire de modifier les règles.

Dans la boucle principale de l'Algorithme 14 d'apprentissage, l'exemple est testé afin de mettre à jour les estimateurs par l'Algorithme 15. On teste la couverture, la pré-couverture et la post-généralisation sur lesquelles reposent les estimateurs. On procède aussi de même sur les généralisations antérieures des règles afin de ne pas avoir des statisques si celles-ci sont spécialisées.

Afin de prévoir les effets d'un exemple, il faut sélectionner une règle capable de le pré-couvrir. S'il y en a plusieurs, on utilise celle dont la confiance est la plus élevée. S'il n'y en a aucune, on prévoit des effets vides. Si la prévision est correcte, il n'y a pas de révision du modèle, de même que pour l'approche dans un environnement déterministe, on teste si aucune règle n'est contredite par l'exemple courant afin de spécialiser (si les estimateurs le permettent) les règles en cause. Cependant, on garde dans L_x les contre-exemples ainsi que les règles associées à ces généralisations contredites.

Ensuite, on essaie de généraliser tous les contre-exemples de L_x. Pour éviter de refaire exactement les mêmes erreurs de généralisation, on empêche un contre-exemple de L_x d'être à nouveau généralisé avec la règle r_{tab} venant d'être spécialisée et à laquelle il était associé.

Algorithme 15 MAJ-ESTIMATEURS(r,x)

Entrée: Une règle r et un exemple x
Sortie: La règle r avec ses estimateurs mis à jour

si $r \approx x$ **alors**
  $r.n_{sae} \leftarrow r.n_{sae} + 1$
sinon si $r \overset{sa}{\sim} x$ **alors**
  $r.n_{sa} \leftarrow r.n_{sa} + 1$
sinon si $r \overset{AE}{\sim} x$ **alors**
  $r.n_{ae} \leftarrow r.n_{ae} + 1$
fin si
$r.gen = (r.n_{ae} - r.n_{sae})/r.n_{ae}$
$r.acc = r.n_{sae}/r.n_{sa}$
si $n_{sa} < \theta_{evl}$ **alors**
  $r.conf \leftarrow 0$
sinon
  $r.conf \leftarrow r.acc$
fin si
$r^n = r$ (avec n le nombre de généralisations ayant conduit à la règle r)
si $n > 0$ **alors**
  MAJ-ESTIMATEURS(r^{n-1}, x)
fin si

Algorithme 16 PREVISION(M,x) : $prev$

Entrée: Un exemple x, un modèle d'action M
Sortie: Un booléen $prev$ indiquant si M couvre x

si $\nexists r \in M.R, r \overset{sa}{\sim} x$ **alors**
  $effets_{prevus} = \emptyset$
sinon
  sélectionner une règle $r \in M.R$ t.q $r \overset{sa}{\sim} x$ avec une substitution OI θ_r, $r.conf \geq r_i.cnf$, $\forall r_i \in M.R$ t.q $r_i \overset{sa}{\sim} x$
  $effets_{prevus}.add = (r.e.add)\theta_r$
  $effets_{prevus}.del = (r.e.del)\theta_r$
fin si
si $x.e = effets_{prevus}$ **alors**
  $prev \leftarrow vrai$
sinon
  $prev \leftarrow faux$
fin si

Algorithme 17 SPECIALISER(M,x,r) :L_x

Entrée: Un contre-exemple x en contradiction avec une règle $r \in M.R$
Sortie: Un modèle M spécialisé mis à jour, une liste L_x des contre-exemples associés aux règles spécialisées

si $r.acc < \theta_{spc}$ et $r.n_{sa} > \theta_{evl}$ **alors**
  $M.R \leftarrow M.R \setminus \{r\}$
  $L_x \leftarrow \emptyset$
  $r^n = r$ % avec n le nombre de généralisations ayant conduit à la règle r
  tant que $x \nsim r^n$, $r^n.acc < \theta_{spc}$, $r^n.n_{sa} > \theta_{evl}$ et $n > 0$ **faire**
    $L_x \leftarrow L_x \cup (x^n, r)$
    $r^n \leftarrow r^{n-1}$
    $n \leftarrow n - 1$
  fin tant que
  si $n = 0$ **alors**
    $L_x \leftarrow L_x \cup (x^0, \emptyset)$
  sinon
    $M.R \leftarrow M.R \cup \{r\}$
  fin si
fin si

La généralisation d'un contre-exemple n'est pas possible dans les cas suivants : si aucune règle (différente de r_tab) ne post-généralise le contre-exemple, si les estimateurs ne sont pas supérieurs aux seuils ou encore si les généralisations obtenues ne sont pas *bien-formées*. Dans ce cas, le contre-exemple est ajouté directement à $M.R$ en tant que règle par l'algorithme 19. Si le nombre maximum de règle est atteint, le contre-exemple prend la place de la règle à la confiance la plus faible.

6.5. Etude empirique

Afin d'évaluer notre méthode, nous utilisons les mêmes environnements que ceux utilisés Section 4.5.1. Un bruit de perception est ajouté à la perception des états. Les états observés par l'agent peuvent être aléatoirement altérés. Avec une probabilité ϵ, des littéraux tirés aléatoirement sont ajoutés ou enlevés de l'état réel. Le nombre d'ajouts/suppressions est choisi aléatoirement entre 1 et n_ϵ. Comme les états sont présentés en séquences, un état bruité s_t influence deux exemples : x_t et x_{t+1}. Cette

Algorithme 18 GENERALISATION$_{REL}(M, x, r_{tab})$

Entrée: Un exemple x non couvert par M, une règle $r \in M.R$
Sortie: un booléen *mod* indiquant si M est modifié par une généralisation r^{n+1} de r couvrant x

$mod \leftarrow$ faux
pour tout $r \in M.R$ **faire**
  si $r \approx x$ **alors**
    $mod \leftarrow$ vrai
  sinon
    si $r \neq r_{tab}$, $r \overset{AE}{\sim} x$ (avec θ_r et θ_x), $r.gen > \theta_{gen}$ et $r.n_{ae} > \theta_{evl}$ **alors**
      $r^n = r$ % avec n le nombre de généralisations ayant conduit à la règle r
      $r^{n+1}.a = (r^n.a)\theta_r^{-1}$
      $r^{n+1}.e = (r^n.e)\theta_r^{-1}$
      $p_{gen} \leftarrow UNE\text{-}GEN_{OI}((r^n)\theta_r^{-1}, x, \theta_r, \theta_x)$
      $r^{n+1} = p_{gen}/r^{n+1}.a/r^{n+1}.e$
      si bien-formée(r^{n+1}) **alors**
        $mod \leftarrow vrai$
        $M.R \leftarrow M.R \backslash \{r^n\} \cup \{r^{n+1}\}$
      sinon
        $mod \leftarrow$ faux
      fin si
    fin si
  fin si
fin pour
si $mod = faux$ **alors**
  $r^0 \leftarrow x$
  $AJOUT(M, r^0)$
fin si

Algorithme 19 AJOUT(M,r)

Entrée: Une règle r à ajouter au modèle M
Sortie: M est modifié : r est ajoutée et éventuellement une autre règle est supprimée

si $|M| > N$ **alors**
  sélectionner une règle $r_{suppr} \in M.R$ t.q $r_{suppr}.cnf \leq r_i.cnf$, $\forall r_i \in M.R$
  $M \leftarrow M \setminus \{r_{suppr}\}$
fin si
$M \leftarrow M \cup \{r\}$

sorte de bruit permet une grande variété de fausses observations possibles.

L'évaluation suit le même protocole que l'évaluation en erreur de prévision décrit Section 4.5.3. Les exemples sont générés en séquence : chaque épisode est borné à 20 pas d'apprentissage. Les actions séquentielles sont choisies aléatoirement. Tous les 100 exemples d'apprentissage, 1000 exemples aléatoires sont générés et utilisés pour estimer l'efficacité du modèle. Les résultats présentés sont des moyennes sur 10 expériences.

Le taux de bruit est fixé à $\epsilon = 10\%$, c'est-à-dire que lorsque l'agent observe un état, il y a 10% de chance que l'état soit mal perçu.

Le paramètre d'environnement n_ϵ est fixé à 2 dans les expériences suivantes, ce qui signifie qu'avec une probabilité de 10%, un ou deux littéraux peuvent être ajoutés ou supprimés à chaque état. Ainsi, ce choix de n_ϵ permet de simuler des remplacements de littéraux si un littéral est supprimé et un autre ajouté. La suppression d'un prédicat se fait aléatoirement parmi les prédicats de l'état. L'ajout d'un prédicat se fait par la sélection aléatoire d'un prédicat de l'environnement et les objets du prédicat choisi sont aussi sélectionnés parmi les objets possibles de l'environnement. On fixe la taille du modèle à 20 règles ce qui est trois fois le nombre de règles nécessaires pour le monde des blocs colorés (cf. Annexe). Fixer une limite assez haute du nombre de règles permet de ne pas introduire de biais ad hoc par rapport au nombre de règles supposées des environnements tout en limitant la taille du modèle.

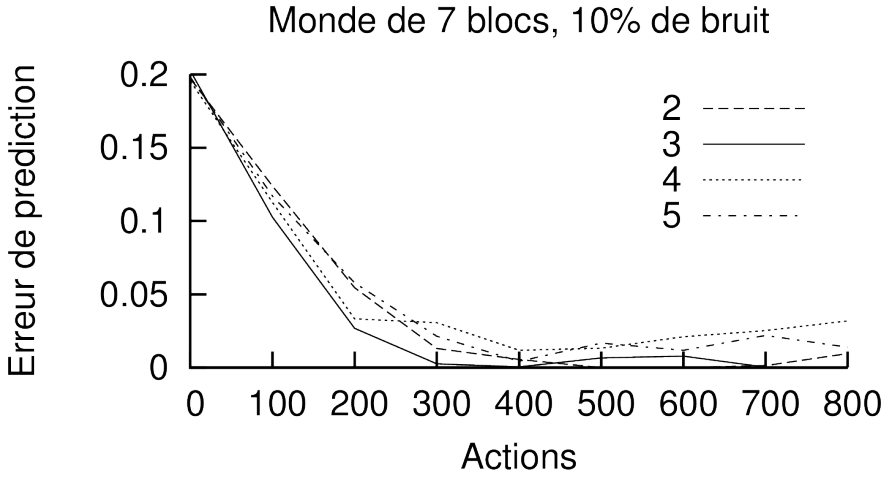


FIGURE 6.1.: Evaluation sur un monde de 7 blocs avec différentes valeurs de seuil pour θ_{evl}.

Dans un premier temps, on fait varier de manière indépendante les seuils d'évaluation d'une part, et ceux de spécialisation et de généralisation d'autre part. Pour simplifier l'étude et limiter l'importance des paramètres, on fixe θ_{gen} et θ_{spe} aux mêmes valeurs.

La Figure 6.1 montre l'évolution de l'erreur de prévision sur un monde de 7 blocs en présence de 10% de bruit. Les différentes courbes illustrent différentes valeurs de seuils θ_{evl}. On note que le nombre d'exemples nécessaires pour converger est plus que le double de celui des expériences sans bruit de perception. Les seuils de spécialisation et de généralisation θ_{spe} et θ_{gen} sont fixés à la même valeur 0.7.

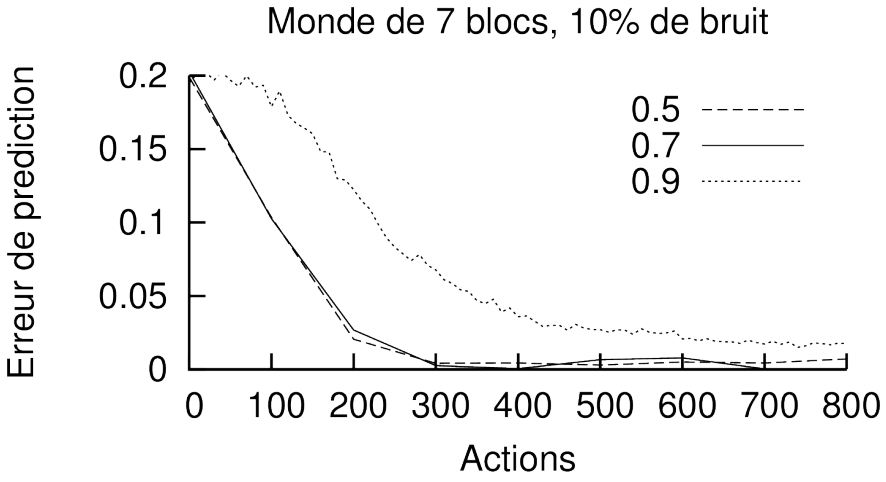


FIGURE 6.2.: Evaluation sur un monde de 7 blocs avec différentes valeurs de seuil de spécialisation et de généralisation.

De façon complémentaire, à la Figure 6.2, toujours avec un monde de 7 blocs et 10% de bruit, on fixe cette fois le seuil θ_{gen} à 3, et on observe l'effet de la variation de θ_{gen} et de θ_{spe}. L'agent atteint une erreur de prévision plus faible avec $\theta_{gen} = \theta_{spe} = 0.7$.

On note que même si les différentes valeurs envisagées des paramètres impactent la qualité de l'apprentissage, IRALe résiste à la présence de bruit.

A la Figure 6.3, on évalue les différents domaines décrits Section 4.5.1. Sur cette Figure θ_{evl}=3 et θ_{gen}=θ_{spe}=0.7. Même si IRALe ne converge pas complètement après 800 actions sur le domaine logistique et sur le monde de blocs colorés, il atteint cependant

moins de 5% d'erreurs de prévision, ce qui montre que l'approche offre une bonne résistance au bruit.

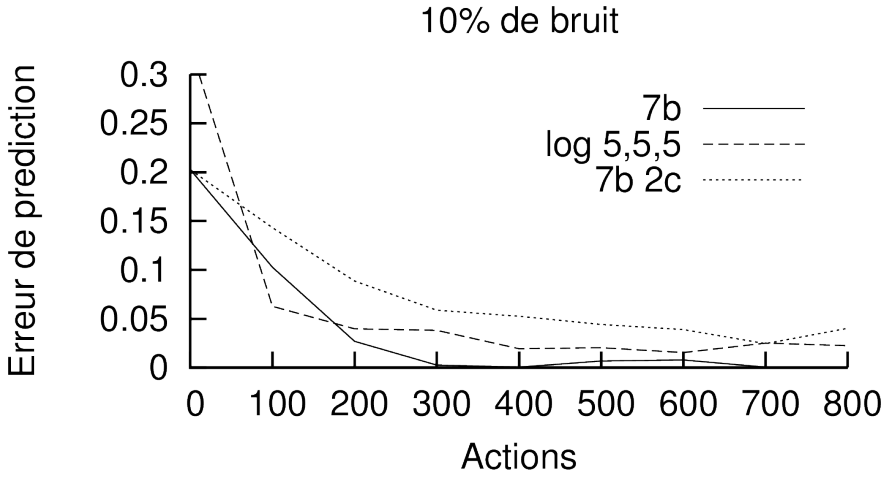


FIGURE 6.3.: Evaluation des différents domaines avec 10% de bruit de perception.

6.6. Conclusion

Dans ce chapitre, nous avons adapté l'approche IRALe afin de traiter des problèmes plus proches de conditions réelles. Notre méthode a montré sa capacité à apprendre un modèle d'action incrémentalement et malgré la présence de bruit.

Ce comportement est rendu possible par la mise à jour de statistiques sur les exemples rencontrés permettant de renseigner des estimateurs. Ces derniers permettent de décider de l'intérêt de généraliser ou de spécialiser une règle face à un exemple potentiellement faux.

Chapitre 7

Conclusions et perspectives

Sommaire

7.1. Conclusions

Dans cette thèse, nous avons développé une méthode permettant à un agent autonome d'apprendre un modèle d'action relationnel uniquement à partir de sa propre expérimentation face à son environnement.

Dans un premier temps, au chapitre 4, nous avons présenté la méthode IRALe (Rodrigues *et al.*, 2010a) guidée par les données, ascendante, capable de généraliser ses observations incrémentalement au sein d'une représentation relationnelle riche. Nous avons prouvé que le nombre d'erreurs était théoriquement borné. Expérimentalement, cette approche a montré ses capacités sur quelques domaines avec un nombre d'erreurs bien plus limité que la borne théorique. Nous avons évalué notre approche dans le même contexte et les mêmes domaines que le système MARLIE (Croonenborghs *et al.*, 2007) et aussi comparé IRALe à une approche descendante hors-ligne utilisée de façon similaire à MARLIE. Dans les deux cas, nous avons obtenu de meilleurs résultats en terme d'exemple nécessaires à l'apprentissage de modèles d'action efficaces. De plus, nous avons montré que notre modèle composé d'un ensemble de règles était directement utilisable par un planificateur et cela même en cours d'apprentissage.

Dans un deuxième temps, au chapitre 5, nous avons proposé une nouvelle méthode d'exploration active (Rodrigues *et al.*, 2012) adaptée à l'approche ascendante d'IRALe, capable d'orienter le choix des actions de l'agent vers des exemples supposés non couverts par le modèle et pouvant donc accélérer les mises à jour du modèle. Sur tous les domaines testés, les expériences ont montré qu'à une même qualité de modèle donnée, nous obtenons un gain en termes de nombre de pas d'exploration (actions) nécessaires.

Enfin, au chapitre 6, nous avons présenté une adaptation d'IRALe capable d'évoluer toujours dans des environnements déterministes, mais cette fois avec une perception imparfaite du monde. Expérimentalement, les résultats ont montré la robustesse de l'approche à l'aide de généralisations et de spécialisations retardées, empêchant une unique observation potentiellement bruitée de remettre en cause directement le modèle.

7.2. Perspectives

Nos travaux peuvent avoir de nombreuses perspectives. Nous développons ici les principales voies possibles.

7.2.1. Environnements non-déterministes

Afin de se rapprocher le plus possible de conditions réelles, le cadre expérimental utilisé par Pasula *et al.* (2007) peut être suivi et adapté à notre méthode. En effet, l'utilisation d'un simulateur physique permet de se rapprocher de conditions plus complexes et réalistes et de soumettre l'agent à des perturbations de différentes sortes. De même, comme pour les travaux de Pasula *et al.* (2007), une extension possible d'IRALe peut être la prise en compte d'environnements stochastiques. Par rapport à un modèle d'action déterministe, il s'agit d'être confronté non pas à un, mais à plusieurs effets pour des pré-conditions et une action données. Une façon de représenter un modèle d'action dans ce cas (Younes, 2003) est de se borner aux effets les plus probables et d'en approximer la distribution dans le cas de plusieurs effets. A notre connaissance, il

n'existe aucun système capable d'apprendre un modèle aussi expressif que Pasula *et al.* (2007) de façon incrémentale.

7.2.2. Environnement multi-agent

Dans cette thèse, nous avons considéré que l'agent était seul face à l'environnement. Cependant, il est intéressant d'étudier aussi le cas où plusieurs agents agissent au sein de l'environnement (Bourgne *et al.*, 2010). Nous avons abordé la question avec Henry Soldano(MdC-HDR) dans le cadre du stage de Toru Mizuno étudiant au LIPN et développé ainsi un cadre d'apprentissage relationnel multi-agent. Chaque agent est un agent IRALe devant apprendre un modèle d'action relationnel mais ayant en plus la particularité de pouvoir communiquer son modèle avec les autres agents dans le but d'obtenir un avis extérieur ainsi que d'autres contre-exemples. Des expérimentations prometteuses ont pu être menées et doivent être approfondies.

7.2.3. Apprentissage d'une fonction qualité ou d'une politique

Depuis les travaux de Dzeroski *et al.* (2001) introduisant le cadre RRL, de nombreux travaux s'inscrivant dans ce cadre se sont focalisés sur l'apprentissage d'une fonction qualité (Van Otterlo, 2008).

Cependant, parmi tous ces travaux, à notre connaissance, aucun n'a réussi expérimentalement à apprendre ou représenter efficacement une stratégie sur des domaines complexes. D'autres travaux, ont utilisé des approches différentes en apprenant directement une politique.

Le système FOALP de Sanner et Boutilier (2009) dérive une politique à partir d'un but et d'un modèle d'action donnés. Il se place dans des environnements relationnels et non-déterministes. Toutefois, le traitement du non-déterminisme est différent de Pasula *et al.* (2007) puisqu'il est considéré que seules les actions ayant parfaitement abouti ont un effet, les autres (comme par exemple la chute d'un bloc après un déplacement)

sont considérées comme sans effets et ne sont donc pas représentées. A partir du but, une régression est effectuée sur le modèle d'action afin d'obtenir une politique. La particularité de ce genre de méthode est de ne pas raisonner à partir de l'espace des états, mais directement au niveau du modèle d'action qui le représente de façon compacte, ce qui permet d'obtenir une politique générale et de s'abstraire de la taille de l'espace des états. De plus, pour réduire la taille de la politique dérivée, celle-ci est transformée en arbre de décision algébrique(ADD) (Bahar *et al.*, 1993). Cependant, comme montré par Kersting *et al.* (2004) avec le système REBEL, même dans certains cas très simples, il n'est pas possible de représenter une telle politique de façon compacte même en utilisant des ADD. Par exemple, dans un monde de blocs, où le but est qu'un bloc donné b soit libre, on peut obtenir une politique composée d'une chaîne infinie d'actions si l'on raisonne dans un niveau abstrait (celui du modèle d'action). Pour que le bloc b soit libre, il est nécessaire de dépiler tous les blocs se trouvant sur celui-ci. Cependant, comme la régression se fait au niveau du modèle d'action et non pas de l'espace des états, il n'y a pas de conditions d'arrêt. Toutefois, Sanner et Boutilier (2009) arrivent à limiter la taille de la politique en utilisant notamment une approximation linéaire de la fonction valeur.

D'autres travaux utilisent des listes de décisions composées de règles afin de représenter la politique, comme Fern *et al.* (2006) permettant de représenter des clôtures transitives (ou chaîne de littéraux entre objets) ou encore Khardon (1996) générant toutes les règles syntaxiquement possibles afin d'en sélectionner ensuite les meilleures.

Afin de parvenir à représenter des politiques compactes, les travaux de Gretton et Thiébaux (2004) mettent l'accent sur la nécessité de définir des biais de langage. Ils utilisent aussi la régression à partir du modèle d'action mais cette fois pour obtenir des règles candidates représentant des biais de langage afin d'utiliser un algorithme d'apprentissage inductif Alkemy (Lloyd, 2003).

A moyen terme, nous pensons que cette voie est prometteuse et qu'il serait intéressant d'une part d'adapter nos méthodes utilisées pour l'apprentissage d'un modèle d'action à l'apprentissage de politique sous forme de règles et d'autre part, de combiner l'apprentissage du modèle d'action et d'une politique avec le même genre de techniques. Il serait alors envisageable de tendre vers un agent à la fois efficace et encore plus

autonome.

Annexe A

Modèles d'action EDS

monde de blocs

$bl(A), bl(B), cl(A),$ $on(A,B), on(B,C)$	$move(A, floor)$	$on(A, floor), cl(B),$ $\neg on(A,B)$
$bl(A), bl(B), cl(A), cl(B),$ $on(A, floor), on(B,C)$	$move(A,B)$	$on(A,B),$ $\neg cl(B), \neg on(A, floor)$
$bl(A), bl(B), bl(C), cl(A), cl(B)$ $on(A,C), on(B,D)$	$move(A,B)$	$on(A,B), cl(C),$ $\neg on(A,C), \neg cl(B)$

monde de blocs coloré

$bl(A), bl(B), cl(A),$ $on(A,B), on(B,C)$	$move(A, floor)$	$on(A, floor), cl(B),$ $\neg on(A,B)$
$bl(A), bl(B), cl(A), cl(B)$ $blanc(A), blanc(B), on(A, floor), on(B,C)$	$move(A,B)$	$on(A,B),$ $\neg cl(B), \neg on(A, floor)$
$bl(A), bl(B), cl(A), cl(B),$ $noir(A), noir(B), on(A, floor), on(B,C)$	$move(A,B)$	$on(A,B),$ $\neg cl(B), \neg on(A, floor)$
$bl(A), bl(B), bl(C), cl(A), cl(B)$ $blanc(A), blanc(B), on(A,C), on(B,D)$	$move(A,B)$	$on(A,B), cl(C),$ $\neg on(A,C), \neg cl(B)$
$bl(A), bl(B), bl(C), cl(A), cl(B)$ $noir(A), noir(B), on(A,C), on(B,D)$	$move(A,B)$	$on(A,B), cl(C),$ $\neg on(A,C), \neg cl(B)$
$bl(A), bl(B), bl(C), cl(A), cl(B)$ $blanc(A), noir(B), on(A,C), on(B,D)$	$move(A,B)$	$noir(A),$ $\neg blanc(A)$
$bl(A), bl(B), bl(C), cl(A), cl(B)$ $noir(A), blanc(B), on(A,C), on(B,D)$	$move(A,B)$	$bleu(A),$ $\neg noir(A)$

domaine logistique

$boxOnTruck(B,T)$, $truckInCity(T,C)$	$unload(B,T)$	$boxInCity(B,C)$, $\neg boxOnTruck(B,T)$
$truckInCity(B,C)$	$move(B,T)$	$truckInCity(B,T)$, $\neg truckInCity(B,C)$
$truckInCity(T,C)$, $boxInCity(B,C)$	$load(B,T)$	$boxOnTruck(B,T)$, $\neg boxInCity(B,C)$

Bibliographie

Eyal Amir et Allen Chang : Learning partially observable deterministic action models. *J. Artif. Int. Res.*, 33(1):349–402, novembre 2008. ISSN 1076-9757. URL http://dl.acm.org/citation.cfm?id=1622698.1622708.

R. Iris Bahar, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abelardo Pardo et Fabio Somenzi : Algebraic decision diagrams and their applications. *Dans Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*, ICCAD '93, pages 188–191, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press. ISBN 0-8186-4490-7.

R. Bellman : Adaptive control processes : a guided tour. Rapport technique, Princeton University, 1961.

S. Benson : Inductive learning of reactive action models. *Dans ICML 1995*, pages 47–54, 1995.

H. Blockeel, L. De Raedt et J. Ramon : Top-down induction of clustering trees. *Dans ICML*, pages 55–63, 1998.

Gauvain Bourgne, Henry Soldano et Amal El Fallah Seghrouchni : Learning better together. *Dans Proceedings of the 2010 conference on ECAI 2010 : 19th European Conference on Artificial Intelligence*, pages 85–90, Amsterdam, The Netherlands, The Netherlands, 2010. IOS Press. ISBN 978-1-60750-605-8. URL http://dl.acm.org/citation.cfm?id=1860967.1860985.

C. Boutilier, R. Reiter et B. Price : Symbolic dynamic programming for first-order mdps. *Dans IJCAI*, pages 690–700. Morgan Kaufmann, 2001.

Ronen I. Brafman et Moshe Tennenholtz : R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *J. Mach. Learn. Res.*, 3:213–231, mars 2003. ISSN 1532-4435.

S. Ceri, G. Gottlob et L. Tanca : *Logic programming and databases*. Springer-Verlag New York, Inc., New York, NY, USA, 1990. ISBN 0-387-51728-6.

David Chapman et Leslie Pack Kaelbling : Input generalization in delayed reinforcement learning : An algorithm and performance comparisons. pages 726–731. Morgan Kaufmann, 1991.

T. Croonenborghs : *Model-Assisted Approaches for Relational Reinforcement Learning*. Thèse de doctorat, K.U.Leuven, Septembre 2009. Bruynooghe, Maurice and Blockeel, Hendrik (supervisors).

T. Croonenborghs, J. Ramon, H. Blockeel et M. Bruynooghe : Online learning and exploiting relational models in reinforcement learning. *Dans IJCAI*, pages 726–731, 2007.

Tom Croonenborghs, Jan Ramon, Hendrik Blockeel et Maurice Bruynooghe : Model-assisted approaches for relational reinforcement learning : some challenges for the srl community. *Dans Proceedings of the ICML-2006 Workshop on Open Problems in Statistical Relational Learning*, pages 1–8, 2006.

W. Dabney et A. McGovern : Utile distinctions for relational reinforcement learning. *Dans IJCAI*, pages 738–743, 2007.

K. Driessens : *Relational reinforcement learning*. Thèse de doctorat, Department of Computer Science, K.U.Leuven, Leuven, Belgium, Mai 2004.

K. Driessens, J. Ramon et H. Blockeel : Speeding up relational reinforcement learning through the use of an incremental first order decision tree algorithm. *Dans ECML, LNAI 2167*, pages 97–108, 2001.

Kurt Driessens et Saso Dzeroski : Integrating guidance into relational reinforcement learning. *Machine Learning*, 57(3):271–304, 2004.

S. Dzeroski, L. De Raedt et K. Driessens : Relational reinforcement learning. *Machine Learning*, 43:7–52, 2001.

Saso Dzeroski, Luc De Raedt et Hendrik Blockeel : Relational reinforcement learning. *Dans ICML*, pages 136–143, 1998.

F. Esposito, S. Ferilli, N. Fanizzi, T. M. A. Basile et N. Di Mauro : Incremental learning and concept drift in inthelex. *Intell. Data Anal.*, 8(3):213–237, août 2004. ISSN 1088-467X. URL http://dl.acm.org/citation.cfm?id=1293831.1293833.

F. Esposito, A. Laterza, D. Malerba et G. Semeraro : Refinement of Datalog programs. *Dans MLnet Familiarization Workshop on ILP (KDD)*, pages 73–94, 1996.

F. Esposito, G. Semeraro, N. Fanizzi et S. Ferilli : Multistrategy theory revision : Induction and abduction in INTHELEX. *Machine Learning*, 38 (1-2):133–156, 2000.

S. Ferilli, N. Fanizzi, N. Di Mauro et T. M. A. Basile : Efficient theta-subsumption under object identity. *Dans In Atti del Workshop AI*IA su Apprendimento Automatico*, 2002.

A. Fern, S. Yoon et R. Givan : Approximate policy iteration with a policy language bias : Solving relational markov decision processes. *Journal of Artificial Intelligence Research (JAIR)*, 25:85–118, 2006.

R. E. Fikes et N. J. Nilsson : Strips : a new approach to the application of theorem proving to problem solving. *Dans IJCAI'71 : Proceedings of the 2nd international joint conference on Artificial intelligence*, pages 608–620, San Francisco, CA, USA, 1971. Morgan Kaufmann Publishers Inc.

Yoav Freund et Robert E. Schapire : Large margin classification using the perceptron algorithm. *Mach. Learn.*, 37(3):277–296, décembre 1999. ISSN 0885-6125.

T. Gartner, K. Driessens et J. Ramon : Graph kernels and gaussian processes for relational reinforcement learning. *Dans ILP, LNCS 2835*, pages 146–163, 2003.

Y. Gil : Learning by experimentation : Incremental refinement of incomplete planning domains. *Dans ICML*, pages 87–95, 1994.

Charles Gretton et Sylvie Thiébaux : Exploiting first-order regression in inductive policy selection. *Dans Proceedings of the 20th conference on Uncertainty in artificial intelligence*, UAI '04, pages 217–225, Arlington, Virginia, United States, 2004. AUAI Press. ISBN 0-9749039-0-6.

D. Haussler : Learning conjunctive concepts in structural domains. *Machine Learning*, 4(1):7–40, 1989.

Jörg Hoffmann et Bernhard Nebel : The FF planning system : Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

Tommi Jaakkola, Michael I. Jordan et Satinder P. Singh : Convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6:1185–1201, 1994.

Michael Kearns : Near-optimal reinforcement learning in polynomial time. *Dans Machine Learning*, pages 260–268. Morgan Kaufmann, 1998.

K. Kersting et L. De Raedt : Logical markov decision programs and the convergence of logical td(λ). *Dans Proc. of ILP'04*, pages 180–197. SpringerVerlag, 2004.

K. Kersting, M. Otterlo et L. De Raedt : Bellman goes relational. *Dans ICML '04 : Proceedings of the twenty-first international conference on Machine learning*, page 59, New York, NY, USA, 2004. ACM.

Roni Khardon : Learning to take actions. *Dans Machine Learning*, pages 787–792. AAAI Press, 1996.

J.-U. Kietz : Some computational lower bounds for the computational complexity of inductive logic programmming. *Dans ECML*, 1993.

T. Lang, M. Toussaint et K. Kersting : Exploration in relational worlds. *Dans Proc. of the European Conf. on Machine Learning (ECML)*, 2010.

Tobias Lang : *Planning and Exploration in Stochastic Relational Worlds*. Thèse de doctorat, Fachbereich Mathematik und Informatik, Freie Universität Berlin, 2011.

N. Littlestone : Learning quickly when irrelevant attributes abound : A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1988.

J. W. Lloyd : *Foundations of logic programming ; (2nd extended ed.)*. Springer-Verlag New York, Inc., New York, NY, USA, 1987. ISBN 3-540-18199-7.

J.W. Lloyd : Learning comprehensible theories from structured data. *Dans* Shahar Mendelson et Alexander Smola, éditeurs : *Advanced Lectures on Machine Learning*, volume 2600 de *Lecture Notes in Computer Science*, pages 203–225. Springer Berlin / Heidelberg, 2003. ISBN 978-3-540-00529-2.

John McCarthy et Patrick J. Hayes : Some philosophical problems from the standpoint of artificial intelligence. *Dans* B. Meltzer et D. Michie, éditeurs : *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969. reprinted in McC90.

Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld et David Wilkins : PDDL – The Planning Domain Definition Language – Version 1.2. Rapport technique, Yale Center for Computational Vision and Control, 1998.

D. Mellor : *Reinforcement Learning, Logic and Evolutionary Computation : A Learning Classifier System Approach to Relational Reinforcement Learning*. LAP Lambert Academic Publishing, Germany, 2009.

Kira Mourão, Ronald P. A. Petrick et Mark Steedman : Learning action effects in partially observable domains. *Dans ECAI*, pages 973–974, 2010.

Kira Mourão, Luke Zettlemoyer, Ronald P. A. Petrick et Mark Steedman : Learning STRIPS operators from noisy and incomplete observations. *Dans Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI 2012)*, pages 614–623, 2012.

S. H. Muggleton : Inductive logic programming. *New Generation Computing*, 8 (4):295–318, 1991.

Stephen Muggleton et Cao Feng : Efficient induction of logic programs. *Dans ALT*, pages 368–381, 1990.

Sriraam Natarajan, Saket Joshi, Prasad Tadepalli, Kristian Kersting et Jude W. Shavlik : Imitation learning in relational domains : A functional-gradient boosting approach. *Dans IJCAI*, pages 1414–1420, 2011.

ShanHwei Nienhuys-Cheng et PeterA. Flach : Consistent term mappings, term partitions, and inverse resolution. *Dans* Yves Kodratoff, éditeur : *Machine Learning EWSL-91*, volume 482 de *Lecture Notes in Computer Science*, pages 361–374. Springer Berlin Heidelberg, 1991. ISBN 978-3-540-53816-5. URL http://dx.doi.org/10.1007/BFb0017030.

Ramón P. Otero : Induction of the indirect effects of actions by monotonic methods. *Dans ILP*, pages 279–294, 2005.

H. M. Pasula, L. S. Zettlemoyer et Kaelbling L. : Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research (JAIR)*, 29:309–352, 2007.

G. Plotkin : A note on inductive generalization. *Dans Machine Intelligence*, volume 5. Edinburgh University Press, 1970.

Martin L. Puterman : *Markov Decision Processes : Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st édition, 1994. ISBN 0471619779.

J. Ross Quinlan : *C4.5 : programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1-55860-238-0.

J. Ramon, K. Driessens et T. Croonenborghs : Transfer learning in reinforcement learning problems through partial policy recycling. *Dans Proc. of The 18th European Conf. on Machine Learning*. Springer-Verlag, 2007.

B. L. Richards et R. J. Mooney : Automated refinement of first-order horn-clause domain theories. *Machine Learning*, 19:95–131, 1995.

C. Rodrigues, P. Gerard, C. Rouveirol et H. Soldano : Incremental learning of relational action rules. *Dans ICMLA*, pages 451–458, 2010a.

C. Rodrigues, P. Gérard et C. Rouveirol : Incremental learning of relational action models in noisy environments. *Dans ILP*, pages 206–213, 2010b.

C. Rodrigues, P. Gérard, C. Rouveirol et H. Soldano : Active learning of relational action models. *Dans ILP*, pages 302–316, 2012.

S. J. Russell et P. Norvig : *Artificial Intelligence : A Modern Approach*. Pearson Education, 2003.

Scott Sanner et Craig Boutilier : Practical solution techniques for first-order mdps. *Artif. Intell.*, 173(5-6):748–788, avril 2009. ISSN 0004-3702.

Dafna Shahaf et Eyal Amir : Learning partially observable action schemas. *Dans AAAI*, pages 913–919, 2006.

W. M. Shen : Discovery as autonomous learning from the environment. *Machine Learning*, 12(1-3):143–165, 1993.

J. Slaney et S. Thiébaux : Blocks world revisited. *Artif. Intell.*, 125(1-2):119–153, 2001.

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

Paul E. Utgoff, Neil C. Berkman, Jeffery A. Clouse et Doug Fisher : *Decision Tree Induction Based on Efficient Tree Restructuring*. Thèse de doctorat, 1996.

M. Van Otterlo : *The logic of adaptive behavior*. Thèse de doctorat, University of Twente, Enschede, 2008.

T. J. Walsh et M. L. Littman : Efficient learning of action schemas and web-service descriptions. *Dans AAAI*, pages 714–719, 2008.

X. Wang : Learning by observation and practice : An incremental approach for planning operator acquisition. *Dans ICML*, pages 549–557, 1995.

C. J. C. H. Watkins et P. Dayan : Q-learning. *Machine Learning*, 8:279–292, 1992.

M. Wiering : *Explorations in Efficient Reinforcement Learning*. Thèse de doctorat, University of Amsterdam, 1999.

Q. YANG, K. WU et Y JIANG : Learning action models from plan examples using weighted max-sat. *Artificial Intelligence*, 171(2-3):107 – 143, 2007.

Qiang YANG, Kangheng WU et Yunfei JIANG : Learning actions models from plan examples with incomplete knowledge. *Dans ICAPS*, pages 241–250, 2005.

Håkan L. S. YOUNES : Extending pddl to model stochastic decision processes, 2003.

Hankz Hankui ZHUO, Qiang YANG, Derek Hao HU et Lei LI : Learning complex action models with quantifiers and logical implications. *Artif. Intell.*, 174(18):1540–1569, 2010.

Table des matières

Table des figures

Liste des Algorithmes

Liste des tableaux

Liste des définitions

Abstract

In this thesis, we study machine learning for action. Our work both covers reinforcement learning (RL) and inductive logic programming (ILP).
We focus on learning action models. An action model describes the preconditions and effects of possible actions in an environment. It enables anticipating the consequences of the agent's actions and may also be used by a planner.
We specifically work on a relational representation of environments. They allow to describe states and actions by the means of objects and relations between the various objects that compose them.

We present the IRALe method, which learns incrementally relational action models. First, we presume that states are fully observable and the consequences of actions are deterministic. We provide a proof of convergence for this method.
Then, we develop an active exploration approach which allows focusing the agent's experience on actions that are supposedly non-covered by the model.
Finally, we generalize the approach by introducing a noisy perception of the environment in order to make our learning framework more realistic.

We empirically illustrate each approach's importance on various planification problems. The results obtained show that the number of interactions necessary with the environments is very weak compared to the size of the considered states spaces. Moreover, active learning allows to improve significantly these results.

Résumé

Dans cette thèse, nous nous intéressons à l'apprentissage artificiel pour l'action. Nous nous situons à l'intersection de l'apprentissage par renforcement (AR) et de la programmation logique inductive (PLI).
Nous étudions plus précisément l'apprentissage de modèles d'actions. Un modèle d'action décrit les conditions et effets des actions possibles dans un environnement. Il permet d'anticiper les conséquences des actions d'un agent et peut aussi être utilisé par un planificateur.
Nous nous intéressons en particulier à une représentation relationnelle des environnements. Nous décrivons alors les états et les actions à l'aide d'objets et de relations entre les différents objets qui les composent.

Nous présentons la méthode IRALe apprenant de façon incrémentale des modèles d'action relationnels. Nous commençons par supposer que les états sont entièrement observables et que les conséquences des actions sont déterministes. Nous apportons une preuve de convergence pour cette méthode.
Ensuite, nous développons une approche d'exploration active qui essaye de focaliser l'expérience de l'agent sur des actions supposées non couvertes par le modèle. Enfin, nous généralisons l'approche en introduisant une perception de l'environnement bruitée afin de rendre plus réaliste notre cadre d'apprentissage.

Pour chaque approche, nous illustrons empiriquement son intérêt sur plusieurs problèmes de planification. Les résultats obtenus montrent que le nombre d'interactions nécessaires avec les environnements est très faible comparé à la taille des espaces d'états considérés. De plus, l'apprentissage actif permet d'améliorer significativement ces résultats.

Mots-clés : Apprentissage artificiel, Programmation logique inductive, Apprentissage par renforcement, Modèles d'action

DISCIPLINE : INFORMATIQUE

Laboratoire d'Informatique de Paris-Nord — UMR CNRS 7030
99 avenue Jean-Baptiste Clément, 93430 Villetaneuse, France