Etude de différentes combinaisons de comportements adaptatives

Olivier Buffet[*,] , Alain Dutech[*] , François Charpillet[*]**

[*] *LORIA - INRIA-Lorraine / Campus Scientifique - B.P. 239*
54506 Vandœuvre-lès-Nancy / France
{dutech,charp}@loria.fr

[**] *National ICT Australia & The Autralian National University*
RSISE Building 115 - ANU / Canberra ACT 0200 / Australie
olivier.buffet@nicta.com.au

RÉSUMÉ. *Cet article s'intéresse à la synthèse automatique d'agents en environnement incertain, se plaçant dans le cadre de l'*apprentissage par renforcement*, et plus précisément des* processus de décision markoviens partiellement observables*. Les agents (dénués de modèle de leur environnement et de mémoire à court terme) sont confrontés à de multiples motivations/objectifs simultanés, problématique qui s'inscrit dans le domaine de la* sélection d'action*.*
Nous proposons et évaluons différentes architectures de sélection d'action. Elles ont en commun de combiner de manière adaptative *des comportements de base déjà connus, en apprenant les réglages de la combinaison afin de maximiser les gains de l'agent. La suite logique de ces travaux est d'automatiser la sélection et la conception des comportements de base eux-mêmes.*

ABSTRACT. *This article focusses on the automated synthesis of agents in an uncertain environment, working in the setting of* Reinforcement Learning*, and more precisely of* Partially Observable Markov Decision Processes*. The agents (with no model of their environment and no short-term memory) are facing multiple motivations/goals simultaneously, a problem related to the field of* Action Selection*.*
We propose and evaluate various Action Selection architectures. They all combine already known basic behaviors in an adaptive *manner, by learning the tuning of the combination, so as to maximize the agent's payoff. The logical continuation of this work is to automate the selection and design of the basic behaviors themselves.*

MOTS-CLÉS : *processus de décision markoviens partiellement observables, motivations multiples*

KEYWORDS: *partially observable markov decision processes, multiple motivations*

RSTI – RIA – 20/2006. PDM en IA, pages 311 à 343

1. Introduction

L'*apprentissage par renforcement* (A/R) vise à concevoir des agents en ne leur spécifiant que l'objectif qu'ils doivent atteindre, et éventuellement en leur fournissant un modèle de l'environnement au sein duquel ils se trouvent. Le formalisme des *processus de décision markoviens* (PDM) (Watkins, 1989, Kaelbling *et al.,* 1996, Sutton *et al.,* 1998a) est particulièrement adapté à de tels problèmes de prise de décision dans le cas où l'environnement est incertain. Considérant des problèmes où l'observabilité de l'agent n'est que partielle, nous faisons plus précisément usage de *processus de décision markoviens partiellement observables* (PDMPO) (Smallwood *et al.,* 1973, Monahan, 1982, Cassandra *et al.,* 1994). On se limite à l'utilisation de comportements réactifs *i.e.*, prenant une décision en fonction de l'observation actuelle.

Toutefois, les approches habituelles, qu'elles entrent dans le cadre de la programmation dynamique ou de la recherche directe de politique, souffrent souvent de ne pouvoir s'adapter à un nombre de motivations variables, l'ajout d'une motivation conduisant à ré-apprendre une politique complète. C'est à ce problème de "passage à l'échelle"[1] que nous nous attachons.

Nous nous intéressons précisément au cas où l'apparition d'objets dans l'environnements crée des nouvelles motivations pour l'agent. Les recherches s'attachant à un tel cadre proposent en général des architectures de sélection d'action (Brooks, 1989, Maes, 1991, Tyrrell, 1993), dans lesquelles de nombreux réglages sont nécessaires. Seul Humphrys (1997) propose d'utiliser l'apprentissage par renforcement pour automatiser la conception d'une telle architecture. Comme lui, nous considérons une architecture basée sur un certain nombre de comportements répondant chacun à un type de motivation différent. L'agent est soumis à diverses motivations, lesquelles déclenchent des comportements spécifiques, proposant chacun une réponse. L'agent doit alors décider des actions à exécuter en fonction de ces réponses. Ce procédé sera par la suite désigné comme étant l'opération de "combinaison des comportements".

Dans ce cadre, l'apprentissage par renforcement peut prendre deux formes :

– apprendre le procédé de combinaison ou régler ses paramètres, et

– apprendre les comportements associés aux diverses motivations.

Humphrys, dans ses travaux (Humphrys, 1997), aborde le second point en apprenant les comportements "en utilisation" *i.e.*, tous en parallèle pendant le fonctionnement de l'agent, donc pour un usage particulier. Nous préférons apprendre les comportements indépendamment les uns des autres, pour éviter de les spécialiser. L'apport principal de cet article est relatif au procédé de combinaison lui-même. S'il n'est pas appris, les paramètres permettant de le régler le sont, et différentes architectures sont proposées et analysées.

1. Nous parlerions de "scalability" en anglais.

Cet article présente d'abord en section 2 l'apprentissage par renforcement et la sélection d'action en insistant sur les aspects importants dans notre cadre de travail. De là, la section 3 décrit notre approche à travers un certain nombre de concepts qu'elle met en œuvre, puis la section 4 précise les différentes combinaisons de comportements envisagées. Les expérimentations menées viennent en section 5 avant une discussion et conclusion.

2. Etat de l'Art

2.1. *Utilisation de l'apprentissage par renforcement*

Le principe de l'apprentissage par renforcement (A/R) répond à notre souhait de concevoir un comportement de manière automatique. En théorie, en utilisant uniquement un signal de renforcement scalaire pour qualifier le résultat des actions de l'agent, les algorithmes d'apprentissage par renforcement permettent en effet de calculer des politiques de décision optimales pour contrôler un système quand on ne connaît pas la dynamique de ce dernier. Dans notre cadre, cela revient à définir le comportement de l'agent.

Du fait des contraintes considérées dans notre travail, et en particulier l'observabilité partielle du système par l'agent, l'utilisation de l'apprentissage par renforcement se place dans le formalisme des processus de décision markoviens partiellement observables (PDMPO) défini ci-dessous.

Définition 1 (Processus de décision markovien partiellement observable) *Un* processus de décision markovien partiellement observable *est défini par un tuple* $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O} \rangle$ *dans lequel :*

– $\mathcal{S} = \{s\}$ *est un ensemble fini d'*états.

– $\mathcal{A} = \{a\}$ *est un ensemble fini d'*actions.

– $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0;1]$ *est une* fonction de transition. *Elle donne la probabilité de passer entre les instants* t *et* $t+1$ *(le temps étant discret) de l'état* s *à l'état* s' *sachant l'action choisie* a *comme suit :*

$$\mathcal{T}(s, a, s') = P(s_{t+1} = s' \,|\, s_t = s, a_t = a).$$

– $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ *une* fonction de récompense[2]. *C'est par l'intermédiaire de cette fonction de récompense que l'on spécifie les buts, les motivations de l'agent.*

– Ω *est un ensemble fini d'*observations.

– $\mathcal{O} : \mathcal{S} \times \Omega \rightarrow [0;1]$ *est une* fonction d'observation. *Elle donne la probabilité de l'observation* o *sachant l'état* s *par :*

$$\mathcal{O}(s, o) = P(o_t = o \,|\, s_t = s).$$

2. On parlera aussi de renforcement ou de gain.

Dans ce cadre, notre problématique de conception d'agent revient à chercher un contrôle optimal du système ne dépendant que des observations car l'agent ne connaît pas l'état réel du système. Si le système que l'on doit apprendre à contrôler s'exprime avec un modèle markovien (la connaissance de l'état du système suffit à connaître sa dynamique), alors certains algorithmes d'apprentissage par renforcement permettent de déterminer un contrôle optimal de ce système au sens d'un critère fonction du signal de renforcement (cf. (Puterman, 1994)). C'est le cas du Q-learning (Watkins, 1989) sous certaines conditions d'utilisation. Dans le cadre des PDMPOs, on perd ce caractère markovien à moins de travailler sur des "états de croyances" (distribution de probabilité sur les états), ce qui revient à travailler sur un espace d'états infini, opération difficilement réalisable en pratique. C'est pourquoi nous avons fait le choix de ne rechercher que des politiques "réactives" ne dépendant que de l'observation courante, d'où le terme "agent sans mémoire".

Un avantage de ces agents réactifs avec des perceptions partielles est de diminuer la complexité apparente du problème en travaillant sur l'espace des observations (qui reste d'une taille raisonnable (Chadès, 2003)) comme s'il s'agissait de l'espace d'états. Cette approche facilite le passage à l'échelle puisque seule une partie limitée du système est perçue. Le problème, ainsi que l'expliquent (Singh *et al.*, 1994) ou (Dutech, 1999), est que le système considéré n'est plus markovien et qu'on ne peut assurer qu'un comportement optimal peut être appris. Un moindre mal est alors de *doter les agents de comportements stochastiques*, qui peuvent être plus performants que des comportements déterministes (cf. (Singh *et al.*, 1994)). Nous chercherons ainsi à doter l'agent d'une *politique* de contrôle de la forme $\pi : \mathcal{O} \rightarrow \Pi(\mathcal{A})$, où $\Pi(\mathcal{A})$ dénote une distribution de probabilité sur l'ensemble $\mathcal{A}$. La mesure de performance adoptée sera l'espérance de récompense en moyenne : $V(\pi) = lim_{T\rightarrow\infty}\mathbb{E}\left(\frac{1}{T}\sum_{t=1}^{T} r_t\right)$.

Même ainsi, le problème que nous voulons résoudre reste difficile avec des méthodes classiques car la complexité des algorithmes de renforcement est principalement liée à un facteur $|\mathcal{S}| * |\mathcal{A}|$. Or les caractéristiques de notre problème augmentent la taille de l'espace d'états du système de manière considérable : celui-ci croît de manière exponentielle avec le nombre d'objets présents. En outre, chaque modification de la tâche à apprendre, même minime, peut nécessiter un nouvel apprentissage qui peut se révéler aussi long, quand on utilise des méthodes d'apprentissage par renforcement, que si l'agent était reparti de zéro. Cela tient au fait que les techniques classiques passent par une évaluation d'une fonction qualité pour chaque couple (état, action) et que la propagation des changements de cette fonction est très lente.

Ainsi, les techniques classiques actuelles de l'apprentissage par renforcement sont limitées à des tâches peu complexes. Pour dépasser ces limitations, nous proposons de replacer l'apprentissage par renforcement dans le cadre de la sélection d'action où il ne s'agit plus d'appréhender une tâche dans son ensemble mais de la voir comme une combinaison de tâches plus simples. Dès lors, et les tâches simples, et le problème de combinaison peuvent être abordés par le biais de l'apprentissage par renforcement.

2.2. *Sélection d'action adaptative et extensible*

A l'image de Humphrys (Humphrys, 1997), par "sélection d'action" nous n'entendons pas le problème de bas-niveau du choix d'une action dans la poursuite d'un unique but cohérent tel que trouver la sortie d'un labyrinthe. Nous entendons plutôt le problème de haut-niveau du choix d'une action dans le cadre de buts conflictuels et hétérogènes. Ces objectifs sont poursuivis en parallèle. Ils peuvent parfois se combiner pour accomplir des objectifs à plus grande échelle, mais généralement ils interfèrent simplement entre eux. Ils peuvent ne pas avoir de condition de terminaison.

Un principe commun à diverses méthodes de sélection d'action est d'associer *des comportements de base* à chacun des objectifs ou des motivations et de combiner ces comportements pour obtenir une décision "globale". Ceci peut être vu comme une décomposition du problème global en plusieurs sous-problèmes *a priori* moins complexes (au sens vu précédemment), ce qui répond en partie aux limitations de l'A/R que nous venons d'évoquer. En effet, plutôt que de traiter directement une tâche complexe, on peut traiter chaque comportement simple séparément et ensuite s'attaquer au problème de la combinaison de ces comportements. Si l'on retrouve diverses formes de décomposition dans des approches hiérarchiques (Matarić, 1997, Sutton *et al.,* 1998b, Dietterich, 2000, Hengst, 2002) ou usant de factorisation de l'espace d'états (Boutilier *et al.,* 2000, Guestrin *et al.,* 2003), qui correspond à une structuration du comportement tirée d'une structuration connue du système. Le terme de comportement que nous employons apparaît aussi dans (Matarić, 1997), correspondant dans (Sutton *et al.,* 1998b) par exemple à la notion d'option. la différence majeure avec notre travail est que nous cherchons à faire un compromis entre plusieurs comportements actifs s*imultanément*, et non à apprendre à en sélectionner un en particulier à un moment donné. Cette approche est d'ailleurs conseillée par (Tyrrell, 1993) qui la qualifie de *hiérarchie à flux libre* (ou *free-flow hierarchy*) par opposition aux structures de décision *winner takes all*. L'avantage principal d'une architecture à flux libre est que l'action finalement choisie peut ne pas être une des actions préconisées par les divers comportements mais bel et bien une action de compromis.

En outre, le fait de vouloir combiner plusieurs comportements simples répond aux exigences de généricité et de passage à l'échelle que nous nous sommes fixées. En effet, des comportements simples bien choisis pourraient être ré-utilisés pour résoudre d'autres tâches complexes. Si on sait par exemple se déplacer et poser/prendre des objets, on peut effectuer des tâches telles que distribuer du courrier, du café ou ramasser les poubelles. De même, une partie des problèmes de passage à l'échelle peut être résolue en combinant plusieurs fois le même comportement : pour peindre dix cubes, c'est gérer dix fois le comportement peindre *un* cube en l'appliquant à des cubes différents. Nous suggérons donc de doter les agents d'une bibliothèque de comportements simples, que nous appellerons *comportements de base*, ces comportements devront être aussi indépendants que possible des tâches à réaliser et devront s'appliquer à des *types de situations* plutôt qu'à des instances de situations.

2.3. *Quelques travaux proches*

Divers travaux ont déjà abordé l'apprentissage par renforcement sous l'angle de la sélection d'action, se basant sur un ensemble de comportements. Nombre d'approches, telles que dans (Mahadevan *et al.*, 1992, Lin, 1993), sont de type *winner takes all*, partageant ce caractère avec les approches hiérarchiques évoquées précédemment. D'autres ont le défaut de produire des comportements spécifiques à la tâche globale choisie, ce qui les rend peu réutilisables. C'est le cas du W-learning (Humphrys, 1997) ou du Q-learning parallèle (Laurent, 2002). Dans cette dernière approche, l'apprentissage est même continuel, afin que l'agent bénéficie d'une mémoire à court terme de ses échecs ou réussites. Pour finir, un défaut commun à la plupart des travaux rencontrés est de se baser sur des algorithmes adaptés à un système markovien, tels que le Q-learning, alors que cette hypothèse n'est clairement pas vérifiée.

2.4. *Bilan*

Notre approche fait l'hypothèse que l'agent à concevoir dispose de comportements de base simples. Ces comportements, adaptés à différentes motivations et sous-tâches, devront être génériques, réutilisables et stochastiques. Par un processus s'inscrivant dans le cadre de l'apprentissage par renforcement l'agent va apprendre à combiner ces comportements en utilisant une méthode de sélection d'action "flux libre" s'appuyant sur un nombre minimal de paramètres. Ce principe est formalisé et décrit en détail dans la section suivante.

3. Concepts mis en œuvre

3.1. *Environnement et perception*

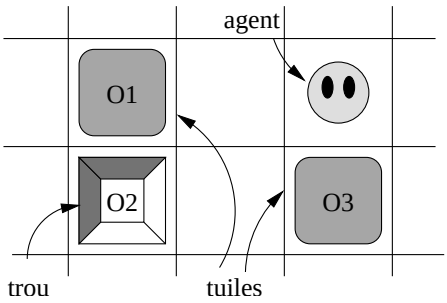
Nous considérons qu'un agent évolue dans un environnement composé d'objets de types définis et qu'il sait reconnaître. Il ne perçoit que partiellement le système à travers une liste de percepts liés à chaque objet.

Ainsi, un agent perçoit la scène courante (la partie du système qui lui est accessible) comme une **observation** qui est un ensemble d'objets **typés** définis par leurs **percepts**. La figure 1 explicite ces notions pour un agent placé dans un monde de type grille dans lequel il perçoit deux tuiles, dont une qui lui est adjacente, et un trou.

En notant $\mathfrak{T}$ l'ensemble des types d'objets présents dans le système, nous allons spécifier quelques concepts utilisés par la suite :

– objet perçu obj : défini par son type et la perception courante que l'agent en a : $obj = [t, per]$ où $t \in \mathfrak{T}$. Nous noterons $t(obj)$ et $\mathcal{O}(obj)$ le type et la perception courante de l'objet ;

– observation o : ensemble d'objets (typés) actuellement perçus par l'agent : $o = \{obj_i\}$;



L'**observation** de cette scène par l'agent est composée des **objets** suivants :

objet X :	(direction ?, proche ?)
tuile O_1 :	(`O`, `faux`)
trou O_2 :	(`SO`, `faux`)
tuile O_3 :	(`S`, `vrai`)

Figure 1. *Une scène avec quelques objets visibles et l'observation associée*

– configuration c : une configuration est un sous-ensemble ordonné d'objets de l'observation. L'ordre est important car deux objets de même type peuvent avoir des rôles différents ;

– type de configuration C^T : un type de configuration $C^T = \langle t_1, \ldots t_n \rangle$ est une liste de types d'objets (non nécessairement uniques). Cela nous permet de définir un opérateur C^T qui extrait d'une observation donnée o toutes les configurations *compatibles* avec le type de configuration : $\mathcal{C}^T(o) = \{c \subseteq o$ telles qu'il existe une application bijective $m : C^T \rightarrow c\}$;

– observation d'une configuration : liste des perceptions associées avec les objets d'une configuration c : $o(c) = (\mathcal{O}(obj), obj \in c)$.

3.2. *Comportements*

Les comportements permettent à l'agent de prendre une décision d'action en fonction de son observation et de ses motivations. Dans notre cadre de sélection d'action, nous allons formaliser deux types de comportements. Les comportements de base constituent la bibliothèque de comportements de "bas niveau" de l'agent. Par notre processus de combinaison de comportements, nous allons combiner les décisions données par ces comportements pour constituer le comportement "extensible" de l'agent.

Définition 2 (Comportement de base)
Un **comportement de base**[3] b *est défini par un tuple* $\langle \mathcal{C}_b^T, P_b, Q_b \rangle$*, où :*

1) C_b^T *est un type de configuration. L'opérateur associé permet de savoir avec quelles configurations de l'observation courante ce comportement de base est compatible et peut être apparié.*

3. On abrégera parfois "comportement de base" par *bb*, acronyme de "*b*asic *b*ehavior". La lettre 'b' de "behavior" désignera par la suite un comportement, de manière à réserver le 'c' aux configurations.

2) $P_b(a|o(c))$ est une politique de décision stochastique apprise par renforcement. Etant donnée une configuration c appropriée, cette politique associe à l'observation de c une distribution de probabilité sur les actions : $P_b : \mathcal{A} \times \mathcal{O} \times \mathcal{C}_b \rightarrow [0,1]$, où $\mathcal{C}_b$ est l'ensemble des configurations de type $\mathcal{C}_b^T$.

3) $Q_b(o(c), a)$ est une fonction que nous appelons Q-valeur et qui mesure la qualité de la politique. Elle donne, pour chaque action effectuée dans une configuration particulière, une mesure de l'utilité de cette action.

Ainsi, un comportement de base étant associé à un type de configuration est générique et peut être utilisé plusieurs fois par l'agent si ce dernier détecte plusieurs configurations compatibles avec le comportement. Cette définition est tout à fait adaptée à l'utilisation de méthodes classiques d'apprentissage par renforcement comme le Q-*learning* (Watkins, 1989) (situation totalement observable) ou la descente de gradient de Baxter et Bartlett (Baxter *et al.*, 2001a, Baxter *et al.*, 2001b) (dans notre situation partiellement observable). On fait ici le choix de définir Q_b comme l'espérance de récompense avec actualisation pour ce comportement de base.

Définition 3 (Comportement extensible)
Un **comportement extensible** eb *est défini par un algorithme de combinaison f_{combi} et un ensemble de comportements de base avec les paramètres de pondération associés, ce qui donne un ensemble de paires $\{(bb, \theta_{bb})\}$. C'est le comportement complexe d'agent que nous cherchons à obtenir, dans lequel les paramètres θ doivent être adaptés à la tâche globale.*

Un point remarquable est qu'un comportement de base, tel que nous l'avons défini, doit être utilisé avec un ensemble donné d'objets dans le voisinage, alors qu'un comportement extensible sera adapté à des nombres variables d'objets. Cette propriété permet de répondre aux exigences de passage à l'échelle que nous avons exprimées.

Par la suite, nous allons proposer des formes différentes pour la fonction de combinaison f_{combi} et montrer comment les paramètres de combinaison θ peuvent être adaptés automatiquement.

3.3. *Algorithmes : décomposition, combinaison*

L'algorithme principal suivi par notre agent s'inscrit dans le cadre général de l'apprentissage par renforcement. Etant données une bibliothèque de comportements de base et une fonction de récompense *globale* définissant la tâche que devra accomplir l'agent, cet algorithme permet de trouver automatiquement les paramètres définissant un comportement extensible performant.

L'algorithme 1 donne le détail des opérations effectuées. Comme dans de nombreux algorithmes d'apprentissage par renforcement, il s'agit essentiellement d'itérations pendant lesquelles l'agent perçoit le système, décide et effectue une action,

observe le résultat de son action et modifie son comportement en fonction de la récompense reçue.

Algorithme 1 Algorithme général

ALGORITHMEGÉNÉRAL
Entrées: $\mathcal{B}$ ensemble des comportements de base fournis
Sorties: Combine les comportements de base et améliore cette combinaison
boucle
$o \leftarrow$ observation courante de la scène
$\mathcal{BC}(o) =$DÉCOMPOSITION(o)
$\mathcal{P}(\cdot|o) =$ COMBINAISON($\mathcal{BC}(o)$)
$a \leftarrow$ CHOIXACTION($\mathcal{P}(\cdot|o)$)
$r \leftarrow$ récompense immédiate de l'agent
$o' \leftarrow$ nouvelle observation
$\theta \leftarrow$ APPRENTISSAGERENFORCEMENT(o, a, r, o', θ)
fin boucle

Quelle que soit la forme de la combinaison de comportements employée, la mesure de performance utilisée est le renforcement moyen gagné pendant un pas de temps. Ce critère est non seulement naturel, mais aussi simple à estimer expérimentalement.

Il reste à préciser dans cet algorithme le fonctionnement de trois étapes : la décomposition de l'observation, la combinaison des comportements, et le renforcement des paramètres de la combinaison. Les sections suivantes décrivent les deux premiers algorithmes, la question du renforcement étant traitée ultérieurement.

3.3.1. *Décomposition de l'observation*

L'objectif de la décomposition de l'observation courante est d'identifier les paires $(comportement\ de\ base,\ configuration)$ dans l'environnement accessible (paires désormais notées $(bb,\ cfg)$), c'est-à-dire mettre en œuvre l'opérateur $\mathcal{C}^T$. L'application qui à l'observation courante associe l'ensemble des paires $(bb,\ cfg)$ de la scène sera notée $\mathcal{BC}$. Ainsi, avec deux comportements de base : `pousser` une tuile dans un trou (b_p) et `éviter` un trou (b_e), l'analyse de la figure 1 par l'agent donne les trois paires $(bb,\ cfg)$ qui suivent :

$$\mathcal{BC}(o) \quad = \quad \{(b_e, \langle O_2 \rangle),\ (b_p, \langle O_1, O_2 \rangle),\ (b_p, \langle O_3, O_2 \rangle)\}. \qquad [1]$$

L'algorithme 2 décrit comment décomposer l'observation courante. La fonction TYPESUIVANT(t : type d'objet) renvoie le type suivant t dans le type de configuration C_b^T. Le parcours récursif garantit de trouver toutes les configurations, quitte à ne faire qu'échanger les rôles de deux objets : l'un peut servir d'outil pour manipuler l'autre, et inversement. On aurait aussi pu choisir d'interdire ces échanges, ce qui permet de réduire le nombre de paires $(bb,\ cfg)$, mais au prix d'une perte d'information.

Algorithme 2 Décomposition de l'observation

DÉCOMPOSITION(o : observation)
Entrées: Ensemble de comportements de base $\mathcal{B}$
Sorties: $\mathcal{BC}(o)$ ensemble de paires $(bb,\ cfg)$
$\mathcal{BC}(o) = \emptyset$
pour tout $b \in \mathcal{B}$ **faire**
$c = \emptyset$
$t =$PREMIERTYPE(C_b^T)
AJOUTER(c,t)
fin pour

AJOUTER(c : configuration, t : type d'objet)
pour tout $obj \in o$ tel que $\neg(obj \in c)$ et $t(obj) = t$ **faire**
$c' = c + obj$
si dernierType(t) **alors**
$\mathcal{BC}(o) \leftarrow \mathcal{BC}(o) \bigcup \{(b, c')\}$
sinon
AJOUTER(c', TYPESUIVANT(t))
fin si
fin pour

3.3.2. *Combinaison de comportements*

La combinaison de comportements de base peut être faite de diverses manières, lesquelles restent de simples heuristiques. Seules des situations particulières rendent possible une convergence prouvée vers une politique optimale, comme dans le cas étudié par (Singh *et al.*, 1998), où une situation markovienne permet de joindre de multiples PDM, analogues à nos comportements de base, en un PDM composite unique. Cette section va maintenant présenter la formulation générale de la fonction de combinaison que nous avons employée.

Etant donné un ensemble de paires $(bb,\ cfg)$, le choix d'une action peut être fait de nombreuses manières telles que vote, enchères ou choix aléatoire. La figure 2 illustre de manière très schématique cette deuxième phase, en cachant dans la boîte "Combinaison" le point critique qui nous intéresse maintenant.

Pour mettre en œuvre une hiérarchie à flux libre, nous avons décidé de calculer une politique $\mathcal{P}(a|o)$ donnant une distribution de probabilité sur les actions a pour chaque observation o. Connaissant l'ensemble des comportements de base à utiliser, la formulation générale que nous proposons est :

$$\mathcal{P}(a|o) \quad = \quad \frac{1}{K} \bigoplus_{(b,c)\in\mathcal{BC}(o)} w(b, o(c), a) \otimes P_b(a|o(c)) \qquad [2]$$

où :

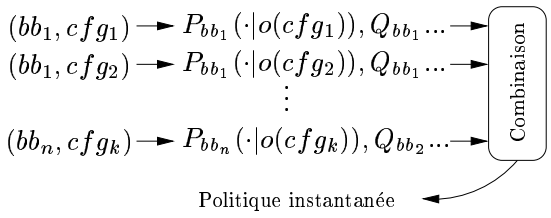


Figure 2. *Schéma de la phase de recombinaison : récupération des données liées à chaque paire* $(bb,\ cfg)$ *et production d'une politique immédiate*

– Les w sont des fonctions positives des Q-valeurs et des paramètres θ. Appelées *poids*, elles permettent de pondérer la proposition de décision qu'est $P_b(a|o(c))$ selon son importance estimée (nous reparlerons de ce poids en section 3.4).

– K est un facteur de normalisation (une correction pour avoir $\sum_a \mathcal{P}(a|o) = 1$).

De plus, les opérateurs $(\oplus, \otimes)$ peuvent être aussi bien $(+, *)$ ou $(*, x^y)$, ou encore $(\max, +)$. Une fois ces probabilités calculées pour une observation donnée, l'action est choisie selon la distribution $\mathcal{P}$.

Par la suite, nous ferons aussi référence à la fonction de combinaison en écrivant $\mathcal{P} = f_{combi}(\mathcal{B})$ où $\mathcal{B}$ est l'ensemble des comportements de base. Les différentes formes de f_{combi} seront présentées en section 4 dans un ordre correspondant à une évolution vers des traitements de plus en plus efficaces et adaptés.

3.4. *Rôle des poids*

Nous avons choisi de faire dépendre les poids w de f_{combi} de deux facteurs :

– des Q-valeurs liées à chaque comportement, et fixées *avant* d'entrer dans le processus de combinaison ; et

– des paramètres θ, dédiés chacun à un comportement de base et appris *pendant* le processus d'adaptation de la combinaison.

Toutefois, toutes les formes de f_{combi} n'utiliseront pas ces deux termes.

3.4.1. *Utilisation des Q-valeurs dans les poids*

Nous avons choisi d'employer la classique définition de Q-valeur actualisée ($0 < \gamma < 1$) :

$$Q^\pi(o, a) = E^\pi \left[\sum_{k=1}^{\infty} \gamma^k r_k | o_0 = o, a_0 = a \right],$$

même si elle n'est pas en relation directe avec la mesure de performance globale choisie : l'espérance de récompense moyenne. En effet cette définition de la fonction de

valeur permet de bien différencier l'intérêt relatif entre deux actions a_1 et a_2 possibles pour une même observation o, au contraire de critères comme la récompense moyenne ou la distance à la récompense moyenne (cf (Bertsekas, 1987) qui assignent des Q-valeurs identiques aux actions même quand les probabilités associées à ces actions sont différentes).

On notera toutefois que Q doit être calculée pour des observations partielles et non des états, ce qui empêche d'employer la formule d'estimation de Q-valeurs habituelle. Nous avons donc utilisé la formule suivante :

$$Q(o,a) \quad \leftarrow \quad (1-\alpha) * Q(o,a) + \alpha * \left(r + \gamma \sum_{a' \in \mathcal{A}} [\pi(o',a') * Q(o',a')] \right) \quad [3]$$

3.4.2. *Rôle des paramètres θ*

En fait, pour la combinaison de comportement, les Q-valeurs seules ne permettent pas une pondération des comportements de base suffisamment discriminante. Les Q-valeurs sont certes discriminantes au sein d'un même comportement de base, et cette propriété est conservée même si ce comportement est utilisé dans un autre contexte (toutes les Q-valeurs seraient alors modulées de la même manière sans que cela ne perturbe leurs valeurs *relatives*). Par contre, pour pouvoir comparer deux comportements appris séparément, il est nécessaire d'avoir recours à un facteur d'échelle indiquant les **priorités relatives entre ces comportements** : c'est là le rôle principal des paramètres θ.

Nous proposons ainsi d'introduire un simple paramètre θ pour chaque comportement de base, lequel servira à définir un facteur d'échelle $e^\theta > 0$ pour les Q-valeurs. L'utilisation de la forme e^θ se justifie pour des raisons pratiques : cela permet de faire une exploration simple sur l'ensemble des réels alors que les poids doivent être strictement positifs.

3.4.3. *Apprentissage*

La valeur de ces paramètres θ qui viennent d'être décrits dépendent de la tâche complexe considérée, contrairement aux Q-valeurs. En tant que tels, ils doivent être appris spécifiquement, ce qui résulte en une combinaison *adaptative* des comportements de base.

Etant dans un cadre partiellement observable pour apprendre les paramètres de la combinaison comme pour apprendre chaque comportement de base, c'est aussi vers la recherche de politiques stochastiques que nous devons nous tourner ici. Si la descente de gradient en ligne de Baxter et Bartlett (Baxter *et al.*, 2001b) est destinée précisément à régler les paramètres d'une politique stochastique, elle requiert malheureusement des propriétés mathématiques qui ne peuvent être ici assurées. N'ayant qu'un nombre limité de paramètres à considérer, nous avons opté pour un algorithme d'optimisation simple : le recuit simulé (Laarhoven, 1987).

Cet algorithme requiert une évaluation assez longue de chaque vecteur de paramètres essayé (N pas de temps). Pour cette raison, ce vecteur reste en fait inchangé pendant N répétitions de la boucle principale de l'algorithme général 1. A la fin d'un tel cycle, les récompenses accumulées donnent une mesure de la performance de la combinaison pour ce vecteur, mesure que l'algorithme de recuit simulé peut utiliser pour choisir le prochain vecteur de paramètres à employer.

4. Détails de la combinaison de comportements

Nous explicitons maintenant les différentes formes de f_{combi} que nous avons étudiées. A part la première formulation qui servira en fait de valeur de référence, toutes ces formulations suivent la formule générale présentée précédemment. Un tableau final récapitule les formulations présentées.

4.1. *Combinaison directe de Q-valeurs*

La première forme est tout à fait indépendante de la formulation générale présentée auparavant. C'est une formulation "naïve" qui nous servira à montrer qu'on ne peut simplement pas additionner les Q-valeurs de différents comportements pour espérer obtenir un bon comportement extensible. Cette combinaison se fait en deux étapes :

1) On calcule la somme des Q-valeurs :

$$\mathcal{Q}_\pi(o,a) \quad = \quad \sum_{(b,c)\in\mathcal{BC}(o)} Q_b(o(c),a). \qquad [4]$$

2) Ensuite, une distribution de probabilité sur les actions est calculée à travers une formule de type équation de Boltzmann :

$$\boxed{\mathcal{P}(a|o) = \frac{e^{\frac{\mathcal{Q}_\pi(o,a)}{T}}}{\sum_{b\in\mathcal{A}} e^{\frac{\mathcal{Q}_\pi(o,b)}{T}}}}, \qquad [5]$$

où T est une température fixée.

On retrouve ici le principe d'additionner simplement des Q-valeurs déjà présent dans la variante "maximiser la joie collective" du W-learning de Humphrys (Humphrys, 1997). Mais le lien intuitif entre Q-valeurs et probabilités d'action peut être dangereux, comme le montre un contre-exemple développé en annexe 7.

4.2. *Un poids par configuration ($+$)*

En considérant la formule générique de f_{combi} avec les opérateurs $+$ et $*$ (on parlera d'une combinaison *additive*, notée par le signe ($+$) dans le titre de la sec-

tion), une première idée est d'employer l'utilité de l'observation d'une configuration $V_b(o(c)) = \max_a Q_b(o(c), a)$ comme poids.

Cette formule intuitivement séduisante est dangereuse quand on mélange des récompenses négatives et positives. Par exemple, si un comportement de base permet d'éviter un obstacle en attribuant une récompense négative à cet obstacle et une récompense nulle partout ailleurs, alors, pour ce comportement, $V_b(o(c)) = 0$ partout. Les probabilités liées à cet important comportement auraient alors un poids de 0 dans la combinaison finale.

Comme nos comportements de base ne concerneront que des signaux de renforcement la plupart du temps nuls, nous proposons de remplacer $V_b(o(c)) = \max_a Q_b(o(c), a)$ par $W_b(o(c)) = \max_a |Q_b(o(c), a)|$. De cette façon, on donne de l'importance aussi bien à des décisions permettant un gain (valeurs positives) qu'à celles évitant un coût (valeurs négatives), les deux étant symétriques. Dans cette première formulation, on utilise $w(b, o(c), a) = e^{\theta_b}.W_b(o(c))$, ce qui donne :

$$\boxed{\mathcal{P}(a|o) = \frac{1}{K} \sum_{(b,c)\in\mathcal{BC}(o)} e^{\theta_b}. \max_{a'} |Q_b(o(c), a')| * P_b(a|o(c))} \quad [6]$$

4.3. *Un poids par configuration et par action (+)*

Utiliser un poids unique pour toutes les actions est assez restrictif. Une modification possible de la forme précédente consiste à employer $w(b, o(c), a) = e^{\theta_b}.|Q_b(o(c), a)|$.

L'intérêt de faire dépendre les poids de l'action peut aussi être illustré par le même cas d'un agent côtoyant un trou (figure 3a) :

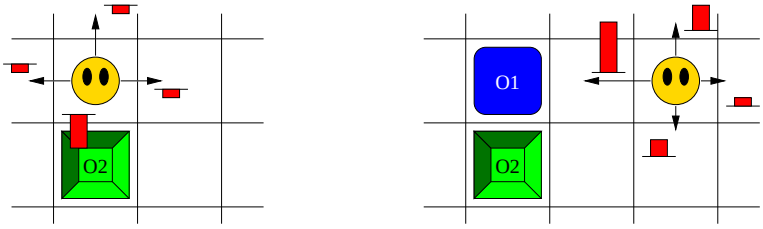
– avec des poids différenciés par action, c'est la probabilité d'action liée à la plus grande perte ou au plus grand gain possible qui prend le plus d'importance. Ici, la probabilité nulle d'aller vers le trou va peser fortement dans la prise de décision car cette probabilité est associée à une forte valeur négative.

– si d'autres comportements sont de bons conseils pour choisir parmi les autres directions, leurs voix seront prééminentes car, pour ces actions, le comportement d'évitement ne pèsera pas beaucoup dans la décision (poids quasiment nul).

La figure 3 illustre sur deux comportements de base la pertinence de la valeur absolue des Q-valeurs pour qualifier l'importance d'une probabilité d'action.

Ainsi, on obtient la seconde formulation suivante :

$$\boxed{\mathcal{P}(a|o) = \frac{1}{K} \sum_{(b,c)\in\mathcal{BC}(o)} e^{\theta_b}.|Q_b(o(c), a)| * P_b(a|o(c))} \quad [7]$$



a- Eviter un trou b- Pousser une tuile dans un trou

Figure 3. *Probabilités d'action (flèches) pour deux exemples de comportements de base, et les Q-valeurs associées (colonnes, à valeurs négatives dans le cas* `a`*)*

4.4. *Un poids par configuration et par action ($+$) $\rightarrow$ correction*

La précédente formule, qui vise à différencier les informations données par les comportement action par action peut être améliorée en évitant de ne faire qu'une seule normalisation qui, finalement, mélange toutes les informations. Il est plus pertinent d'utiliser d'abord les informations action par action et seulement ensuite la distribution de probabilité sera normalisée par rapport aux actions. Nous suggérons alors d'appliquer plutôt un processus en deux étapes :

– d'abord, pour chaque action a, les comportements de base (vus ici comme des experts) donnent leurs avis de manière à obtenir une nouvelle probabilité moyenne $\mathcal{R}(o,a)$ d'effectuer l'action donnée a. On calcule ainsi la somme pondérée des probabilités pour l'action a indépendamment des autres actions ;

$$\mathcal{R}(o,a) = \frac{1}{k(o,a)} \sum_{(b,c)\in\mathcal{BC}(o)} e^{\theta_b}.|Q_b(o(c),a)| * P_b(a|o(c)) \quad [8]$$

$$\text{où } k(o,a) = \sum_{(b,c)\in\mathcal{BC}(o)} e^{\theta_b}.|Q_b(o(c),a)| \quad [9]$$

– ensuite seulement ces résultats sont normalisés pour assurer qu'on obtient bien une distribution de probabilités.

$$\mathcal{P}(a|o) = \frac{1}{K}\mathcal{R}(o,a), \quad \text{où} \quad K = \sum_{a'\in\mathcal{A}} \mathcal{R}(o,a'). \quad [10]$$

On peut écrire cela à travers une unique formule plus synthétique où les e^{θ_b} sont mis en commun, ce qui peut aussi être fait sur d'autres formules :

$$\mathcal{P}(a|o) = \frac{1}{K}\frac{1}{k(o,a)}\sum_{b\in\mathcal{B}} \underbrace{e^{\theta_b}}_{\text{à apprendre}} \left[\underbrace{\sum_{c\in\mathcal{C}(o,b)} |Q_b(o(c),a)| * P_b(a|o(c))}_{\text{déjà connus}} \right] \quad [11]$$

Ce calcul ne respecte en fait pas strictement la forme générique que nous avons donnée : ici w est non seulement fonction de b,$o(c)$ et a, mais aussi des autres paires comportement-configuration dans la scène (à cause du terme $\frac{1}{k(o,a)}$).

4.5. *Un poids par configuration ($*$)*

Il n'y a pas de raison particulière d'utiliser des combinaisons arithmétiques (additives) plutôt que géométriques (multiplicatives) dans les heuristiques que nous développons.[4] C'est pourquoi nous avons aussi essayé les opérateurs $(*, x^y)$, notés par le signe $(*)$ dans le titre de section.

La première formule multiplicative proposée n'utilise que les paramètres e^θ dans les poids (des expériences avec la fonction w définie dans la section 4.3 précédente n'ayant pas apporté de meilleurs résultats), d'où l'expression :

$$\mathcal{P}(a|o) = \frac{1}{K}\prod_{(b,c)\in\mathcal{BC}(o)} P_b(a|o(c))^{e^{\theta_b}} \quad [12]$$

Stabilité

Comme nous travaillons sur des combinaisons multiplicatives, un problème de stabilité apparaît. En effet, si pour toute action $a \in \mathcal{A}$ il y a un facteur nul dans le produit $\prod_{(b,c)\in\mathcal{BC}(o)} P_b(a|o(c))^{e^{\theta_b}}$, on va tomber sur un cas problématique : K étant lui-même nul, on obtient une division par zéro et $\mathcal{P}$ n'est plus une distribution de probabilité.

Il faut en fait que la combinaison de comportements respecte certaines propriétés :

– *stabilité* : qu'elle fournisse une distribution de probabilités, et

– *commutativité* et *associativité* : que l'ordre dans lequel sont traités les comportements soit indifférent.

Dans les différentes formes de combinaison présentées, commutativité et associativité ne posent pas de problème. Par contre, le problème de division par zéro que l'on

4. Tyrrell (1993) fait un commentaire comparable sur un problème de combinaison de stimuli.

vient de voir nuit à la stabilité. Pour qu'elle soit assurée, il faut "interdire" les probabilités nulles dans les politiques des comportements de base. En pratique, on a remplacé toute probabilité nulle par $\epsilon > 0$, puis renormalisé le tout.

En d'autres termes, nous avons restreint l'espace $\Pi(\mathcal{A})$ des distributions de probabilités sur les actions pour interdire les probabilités nulles. Le sous-espace résultant, noté $\Pi^*(\mathcal{A})$, est stable pour notre fonction de combinaison.

4.6. *Un poids par configuration et par action ($*$) $\rightarrow$ correction*

Ce dernier processus de combinaison est la contrepartie multiplicative de la combinaison présentée en section 4.4. On obtient ainsi :

$$\mathcal{P}(a|o) = \frac{1}{K} \left(\prod_{(b,c)\in\mathcal{BC}(o)} [P_b(a|o(c))]^{e^{\theta_b}\cdot|Q_b(o(c),a)|} \right)^{\frac{1}{k(o,a)}} \qquad [13]$$

Résumé

Le tableau 1 fait un rapide rappel de la liste des méthodes de combinaisons exposées, de façon à les avoir toutes en tête.[5]

5. Application/Expériences

Trois séries d'expériences sont présentées ici, elles concernent :

1) l'évaluation des diverses formules de combinaison et l'analyse des résultats obtenus,

2) l'utilité de l'ajout d'un comportement de base "bruit" qui puisse rendre le comportement extensible moins déterministe, et

3) l'adaptabilité et la réutilisabilité des poids (paramètres θ) appris.

5.1. *Problème applicatif*

L'exemple choisi est le problème du *monde des tuiles* (Joslin *et al.*, 1993, Wooldridge *et al.*, 1995), lequel va être utilisé dans les sections suivantes pour décrire l'approche que nous proposons et évaluer les résultats obtenus. Les idées qui sont introduites à travers cet article n'en restent pas moins génériques et donc exploitables dans d'autres situations.

5. Leurs numéros seront désormais utilisés pour les identifier.

1) Combinaison directe de Q-valeurs

$$\mathcal{P}(a|o) = \frac{e^{\frac{\mathcal{Q}_\pi(o,a)}{T}}}{\sum_{b\in\mathcal{A}} e^{\frac{\mathcal{Q}_\pi(o,b)}{T}}}$$

2) $(+)$ Un poids par configuration

$$\mathcal{P}(a|o) = \frac{1}{K} \sum_{(b,c)\in\mathcal{BC}(o)} e^{\theta_b}.\max_{a'} |Q_b(o(c),a')| * P_b(a|o(c))$$

3) $(+)$ Un poids par configuration et par action

$$\mathcal{P}(a|o) = \frac{1}{K} \sum_{(b,c)\in\mathcal{BC}(o)} e^{\theta_b}.|Q_b(o(c),a)| * P_b(a|o(c))$$

4) $(+)$ Un poids par configuration et par action $\rightarrow$ Correction

$$\mathcal{P}(a|o) = \frac{1}{K}\frac{1}{k(o,a)} \sum_{b\in\mathcal{B}} e^{\theta_b} \left[\sum_{c\in\mathcal{C}(o,b)} |Q_b(o(c),a)| * P_b(a|o(c)) \right]$$

5) $(*)$ Un poids par configuration

$$\mathcal{P}(a|o) = \frac{1}{K} \prod_{(b,c)\in\mathcal{BC}(o)} P_b(a|o(c))^{e^{\theta_b}}$$

6) $(*)$ Un poids par configuration et par action $\rightarrow$ Correction

$$\mathcal{P}(a|o) = \frac{1}{K} \left(\prod_{(b,c)\in\mathcal{BC}(o)} [P_b(a|o(c))]^{e^{\theta_b}.|Q_b(o(c),a)|} \right)^{\frac{1}{k(o,a)}}$$

Tableau 1. *Récapitulation des méthodes de combinaison proposées*

Problème

Le monde des tuiles est un environnement quadrillé dans lequel une case peut contenir un *trou*, une *tuile* ou un *agent*. Dans le problème complet, l'agent (on en considère ici un seul) doit pousser des tuiles dans des trous aussi souvent que possible, tout en évitant de passer lui-même dans l'un de ces trous.

Quelques détails sur la simulation ont leur importance. D'abord, l'agent a la possibilité de pousser un groupe de plusieurs tuiles. Il peut aller librement dans un trou (et aussi en sortir), mais recevra alors une récompense négative. De plus, quand une

tuile est poussée dans un trou, tuile *et* trou disparaissent et réapparaissent au hasard quelque part sur la grille. Finalement, pour éviter quelques situations de blocage, tuiles et trous ne peuvent être sur des cases en bord du quadrillage : les tuiles ne pourraient être "décollées" du bord.

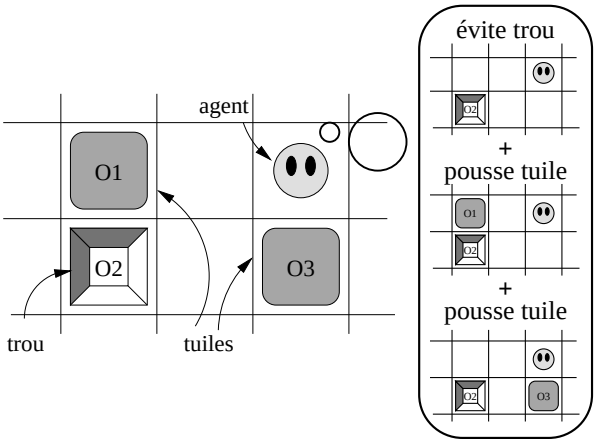


Figure 4. *Une scène avec quelques objets : la tâche globale est une combinaison de sous-tâches*

Architecture de l'agent

En s'attachant à re-situer le problème du monde des tuiles dans le cadre de l'apprentissage par renforcement de départ, l'architecture de l'agent est décrite par les éléments qui suivent.

perceptions : dans nos travaux, l'agent a toujours tous les autres objets de l'environnement en vue. Le principe de localité est toutefois présent dans le fait que les perceptions sont imprécises. Pour tout objet O de la scène, la *perception* que l'agent a de O donne :

– `dir(`O`)` : sa direction `(N-NO-O-SO-S-SE-E-NE)`, et

– `proche(`O`)` : la réponse à la question : "Est-il dans le carré de 9 cases centré sur l'agent `(vrai|faux)` ?"

Ainsi, pour chaque objet nous avons $2 * 8 = 16$ perceptions possibles, d'où un espace de taille au pire 16^n si n objets sont présents. Toutefois, des symétries et situations impossibles permettent de réduire cette dimension.

actions : les seules *actions* disponibles pour l'agent sont de se déplacer d'une case vers le `nord`, le `sud`, l'`est` ou l'`ouest`. Avec ce choix, l'agent ne peut tenter de rester sur sa case actuelle.

récompenses : il reste à spécifier l'objectif de l'agent, à savoir mettre le plus possible de tuiles dans des trous, tout en évitant de tomber dans ces derniers. Dans ce but, on définit la récompense reçue à chaque instant comme étant de :

$(+1)$ quand une tuile tombe dans un trou,

(-3) quand l'agent passe dans un trou, et

(0) le reste du temps.

Dans ces valeurs, le seul choix lié à notre approche est la valeur nulle quand aucun événement particulier n'arrive : le signe des Q-valeurs doit refléter un gain ou un danger proche. Sinon, les valeurs $(+1)$ et (-3) ne dépendent que de l'importance que l'on accorde aux différents événements.

Les comportements de base

Dans ce problème complexe complet, de nombreuses tuiles et trous doivent être gérés. Comme on peut le voir sur la figure 4, une décomposition simple du problème en deux comportements de base peut être faite intuitivement :

– `[évite trou]`, qui met en jeu un trou, ce que l'on note $(b_e, \langle trou \rangle)$ où $\langle trou \rangle$, sur l'exemple, peut être instancié avec O_2 et

– `[pousse tuile dans trou]`, qui met en jeu à la fois une tuile et un trou, ce que l'on note $(b_p, \langle tuile,\ trou \rangle)$. Ce comportement peut être instancié deux fois sur l'exemple, avec O_1 ou O_3 comme tuile, et O_2 comme trou.

Ces deux comportements, ou plus précisément les politiques associées, s'apprennent très bien séparément l'un de l'autre. Dans ce but, on place l'agent dans un environnement de taille 6×6. Pour apprendre `[évite trou]` (resp. `[pousse tuile dans trou]`), on fait interagir l'agent avec un trou (resp. une tuile et un trou) avec une récompense de -3 en cas de chute (resp. +1 si la tuile est dans le trou). Dans chacun de ces deux cas, on laisse l'apprentissage se faire jusqu'à stabilisation des comportements, l'algorithme utilisé étant dans notre cas la version en ligne de la descente de gradient de Baxter (Baxter *et al.*, 2001a, Baxter *et al.*, 2001b).

5.2. *Les diverses combinaisons*

Dans un premier temps, on va chercher simplement à évaluer l'efficacité des diverses combinaisons expérimentées et à identifier les difficultés ou comportements inattendus qui peuvent apparaître.

5.2.1. *Expériences effectuées*

Pour évaluer l'efficacité de notre algorithme avec les différentes formes de combinaisons présentées en section 4, il a été testé dans différents environnements : la taille de la grille a été fixée à 6×6, mais le nombre d'objets (tuiles et trous) change.

Dans l'idéal, la meilleure politique stochastique devrait être utilisée comme référence dans chacun de ces cas. La seule façon de l'obtenir est de passer par un apprentissage par renforcement. A part avec une tuile et un trou, une descente de gradient ne réussit pas ici à apprendre d'autres tâches que celle consistant à éviter les trous ce qui

correspond à un minimum local dans le processus d'optimisation. La meilleure référence disponible a été obtenue avec un Q-learning boltzmannien adapté,[6] lequel doit souvent être loin de l'optimum, mais montre les résultats qu'une méthode classique peut amener.

Le tableau 2 présente dans sa première colonne les situations dans lesquelles des tests ont été conduits. Les résultats sont montrés dans la deuxième colonne[7], où les barres indiquent la récompense moyenne obtenue après une phase d'apprentissage des deux paramètres, un par comportement, et les barres d'erreur donnent l'écart-type observé. Dans chaque cas, 10 simulations ont été effectuées avec :

– une phase d'apprentissage de $70 * 100\,000$ pas de temps utilisant un recuit simulé, et

– une phase d'évaluation de $10 * 100\,000$ pas de temps.

Cette "unité de base" de $100\,000$ pas de temps est justifiée pour évaluer de manière précise la qualité d'un ensemble de paramètres. Finalement, une ligne horizontale est dessinée dans chaque figure au niveau atteint avec le Q-learning boltzmannien adapté. La température $T_\infty = 0,08$ utilisée a été choisie à travers diverses expérimentations sur différents problèmes-jeu.

5.2.2. *Analyse des résultats*

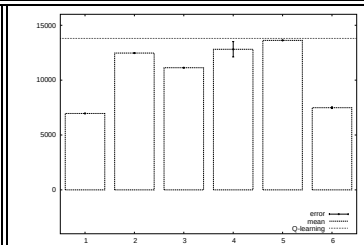
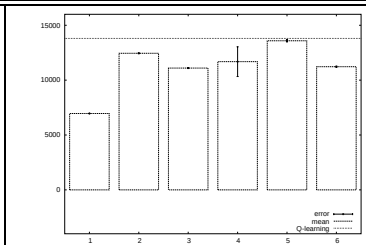
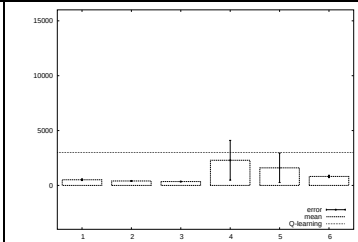
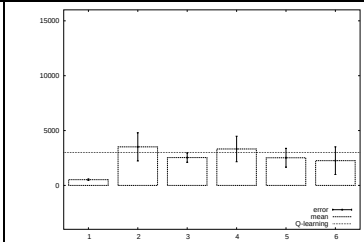
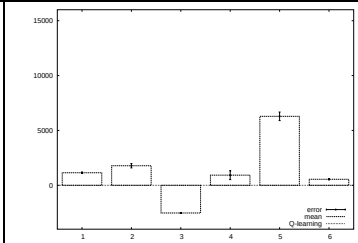
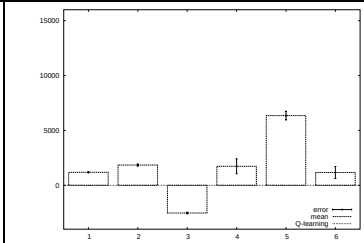
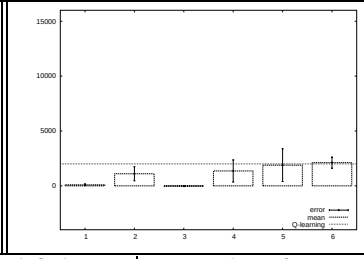
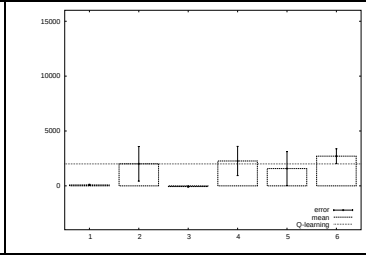
Première impression. Une première remarque est que, globalement, les combinaisons semblent être toutes très efficaces avec une tuile et un trou, et subir une dégradation notable dans les autres cas. Toutefois, le Q-learning suit une évolution similaire.

Des combinaisons peu efficaces. L'idée qu'"estimer une politique d'après les Q-valeurs n'est pas une bonne intuition" est confirmée par les mauvais résultats de la combinaison 1 (combinaison de Q-valeurs). Il est aussi confirmé que la combinaison 3 ($(+)$ un poids par configuration et action, *non* corrigé) ne devrait pas être utilisée, puisqu'elle amène même à produire un comportement extensible avec une récompense moyenne négative dans le cas 1-tuile/2-trous.

Des résultats instables. Quand il y a deux tuiles dans l'environnement, la plupart des approches efficaces semblent donner des résultats très instables. Pendant l'exploration, le système est enclin à tomber régulièrement dans des optima locaux. Cela devrait être amélioré par des réglages du recuit simulé, mais aux dépens du temps de calcul déjà important.

Des poids dépendants de l'action ? La combinaison 4 ($(+)$ un poids par configuration et action, corrigé) peut être vue comme une évolution de la combinaison 2 ($(+)$ un poids par configuration) vers des pondérations dépendantes de l'action. Néanmoins, la comparaison des deux algorithmes additifs ne prouve pas que des poids dépendants de

6. Q-learning fournissant une politique stochastique grâce à une fonction de boltzmann (Dutech *et al.*, 2003).
7. La troisième colonne sera utilisée plus tard.

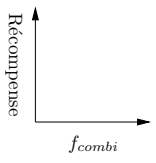
$\#_{tuiles}$ $+\#_{trous}$	combinaisons normales	...avec un comportement aléatoire
$1 + 1$		
$2 + 1$		
$1 + 2$		
$2 + 2$		

1. Combinaison directe de Q-valeurs
2. $(+)$ Un poids par confi guration
3. $(+)$ Un poids par confi guration et action
4. $(+)$ Un poids par confi guration et action $\longrightarrow$ correction
5. $(*)$ Un poids par confi guration
6. $(*)$ Un poids par confi guration et action $\longrightarrow$ correction

Tableau 2. *Efficacité des différentes combinaisons*

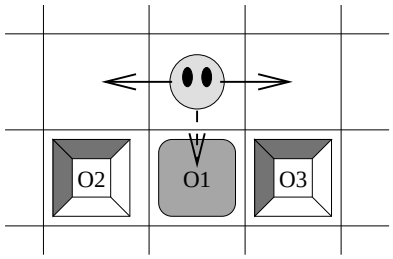
Sur les figures ci-dessus, l'axe des abscisses indique les combinaisons utilisées f_{combi}, et l'axe des ordonnées est consacré à la mesure de récompense.

Dans chaque situation, on donne la récompense moyenne atteinte dans nos expérimentations après apprentissage, ainsi que l'écart-type. Ces deux grandeurs sont calculées sur $100\,000$ pas de temps. Les figures dans la dernière colonne montrent les mêmes tests avec l'utilisation d'un comportement *aléatoire* additionnel (voir section 5.3).



l'action constituent une amélioration. En comparant les combinaisons 5 et 6 dans le cas 1-tuile/2-trous, des poids ne dépendant que du comportement peuvent apparaître comme la meilleure solution. Sans explication claire de ce phénomène, il serait toutefois hasardeux de tirer des conclusions sur ce point sur la base des présents résultats.

Situations bloquantes. L'observation directe des comportements extensibles de l'agent en activité amène des explications sur les problèmes rencontrés avec plus d'une tuile et d'un trou. En effet, deux situations de blocage typiques apparaissent, montrées sur les figures 5 et 6, dans lesquelles l'agent passe la plupart de son temps.



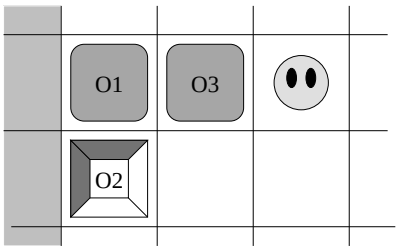
Ici, deux contrôleurs sont prépondérants (momentanément) pour l'agent :

– `pousser` O_1 dans O_2 qui incite à aller vers l'est, et

– `pousser` O_1 dans O_3 qui incite à aller vers l'ouest.

Or le choix le plus judicieux serait plutôt le sud, pour sortir la tuile d'entre les deux trous.

Figure 5. *Blocage avec* 1 *tuile et* 2 *trous*



Que ce soit pour `pousser` O_1 dans O_2 ou `pousser` O_3 dans O_2, les deux contrôleurs prépondérants incitent à aller vers l'ouest, ce qui ne permet pas au système d'évoluer.

Mieux vaudrait aller vers le nord pour aborder l'une des tuiles par un autre côté.

Figure 6. *Blocage avec* 2 *tuiles,* 1 *trou, et des cases interdites à l'ouest*

Dans les deux cas, l'action appropriée n'est suggérée par aucune des paires (bb, cfg) impliquées. Dans la figure 5, l'agent hésite entre aller vers l'est et vers l'ouest, d'où un mouvement oscillant. Dans la figure 6, les deux instances du comportement `pousser` veulent que l'agent aille vers l'ouest alors que cette direction est bloquée par un mur. De tels phénomènes amènent des limitations fortes au principe de combiner des comportements de base, d'autant qu'ils sont difficiles à corriger. En effet, si les décisions prises dans ces situations sont inappropriées, les décisions qui mènent à ces situations peuvent aussi être fautives, même prises à plusieurs pas de temps dans le passé.

5.3. *Ajout d'un comportement aléatoire*

Le fait de produire un comportement stochastique permet déjà d'éviter un grand nombre de blocages en ne se bornant pas à essayer une seule action. Pour continuer dans ce sens, une idée courante pour sortir un agent de situations bloquantes est d'ajouter du bruit à son système de prise de décision, donc ici à sa politique. Jusqu'à un certain point, un tel bruit est bénéfique. Au delà de ce point, il sera la cause d'une dégradation progressive de l'efficacité de l'agent. Dans notre approche, ce bruit va aussi avoir l'avantage important de s'assurer que chaque action est toujours accessible. C'est l'intérêt global d'un tel bruit supplémentaire que nous avons voulu mesurer ici.

5.3.1. *Expériences effectuées*

Pour mettre ce principe en application et aider à résoudre les deux blocages présentés dans les figures 5 et 6, un nouveau comportement de base artificiel a été créé. Ce comportement "aléatoire" est défini comme suit :

1) $\mathcal{C}_{b_e}^T$ est vide (il n'y a pas d'objets dont il faille se préoccuper).

2) $P_{b_e}(a|o(c))$ est équiprobable (toutes les actions ont toujours des chances égales d'être choisies).

3) $Q_{b_e}(o(c), a)$ est constante et non nulle (faute de quoi son poids serait nul dans toutes les combinaisons, et ce comportement ne servirait à rien).

La phase d'adaptation des paramètres servira à régler le volume du bruit nécessaire. Revenant au tableau 2, les quatre figures dans la troisième colonne correspondent aux mêmes expérimentations que leurs voisines, mais en faisant usage de ce comportement de base aléatoire additionnel.

5.3.2. *Analyse des résultats*

En regardant le cas "non pathologique" à 1 tuile et 1 trou, le comportement aléatoire ne semble pas améliorer les résultats obtenus. Au contraire, une légère baisse peut être observée. Ce point est à relier au plus grand nombre de paramètres à ajuster, lequel agrandit l'espace à explorer.

Dans les trois autres cas de cette étude, la plupart des combinaisons bénéficient de ce nouveau comportement de base. C'est évident dans les situations impliquant 2 tuiles et au moins 1 trou, probablement du fait d'une plus grande efficacité pour le problème décrit par la figure 6. Si cela ne résout pas complètement les blocages, cela peut aider dans le cas de combinaisons "trop déterministes".

5.4. *Réutilisabilité et "extensibilité"*

L'intérêt d'une méthode de sélection d'action ne réside pas uniquement dans sa capacité à résoudre un problème donné. On va donc voir ici si notre approche per-

met de réutiliser les paramètres appris, c'est-à-dire voir dans quelle mesure elle est "extensible".

5.4.1. *Expériences effectuées*

Les tests de réutilisabilité des paramètres s'organisent en deux parties :

1) Dans le tableau 3 est comparé l'apprentissage de paramètres dans le cas 2-tuiles/2-trous (désormais noté $(2+2)$) quand ils sont initialisés soit avec des valeurs nulles, soit avec la valeur moyenne des meilleurs paramètres appris dans les cas $(1+2)$ et $(2+1)$.

2) Puis, le tableau 4 montre l'efficacité des différentes combinaisons quand on réutilise les meilleurs paramètres pour $(2+2)$ dans des situations mettant en jeu un plus grand nombre d'objets.

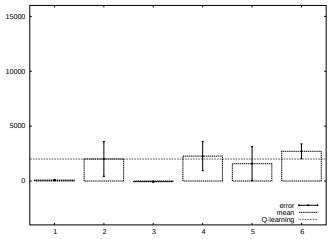
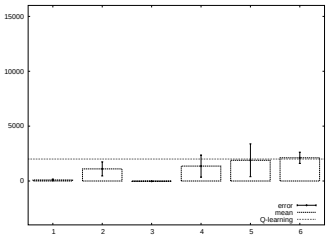
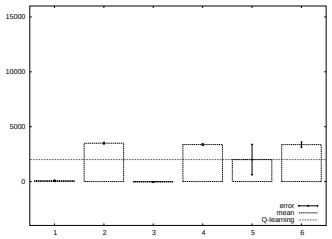
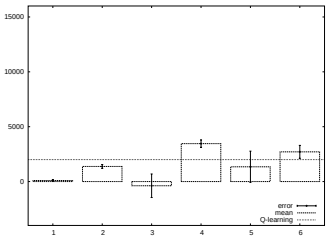
Initialisation	combinaisons normales	...avec comportement aléatoire
en partant de zéro		
en réutilisant des paramètres		

Tableau 3. *L'apprentissage est-il meilleur quand on réutilise des paramètres ?*

5.4.2. *Analyse des résultats*

Le premier tableau de figures (tableau 3) montre clairement l'intérêt de ne pas commencer l'adaptation des paramètres en partant de zéro. Les paramètres de départ calculés donnent de bonnes initialisations : les niveaux moyens d'efficacité obtenus sont notablement meilleurs et plus stables. Même s'il peut être nécessaire d'apprendre avec précaution les paramètres dans les deux cas simples $(1+2)$ et $(2+1)$, c'est un travail effectué pour le long terme.

Comme l'illustre le dernier tableau (tableau 4), quelques combinaisons semblent mieux "passer à l'échelle", puisqu'une simple réutilisation de bons paramètres du cas

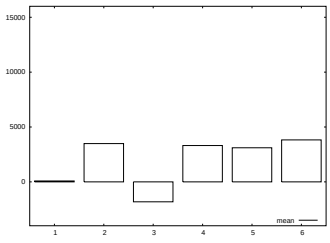
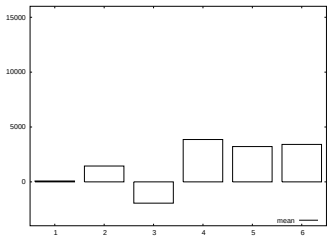
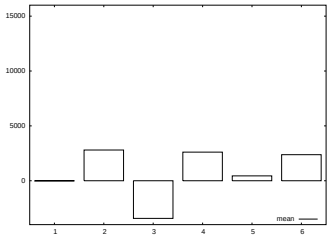
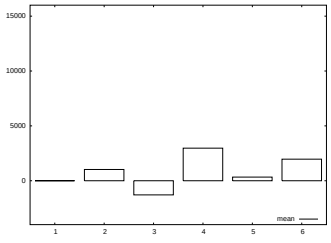
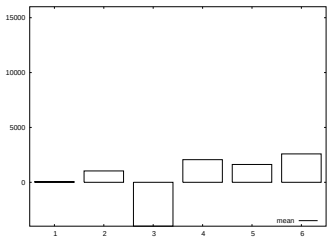
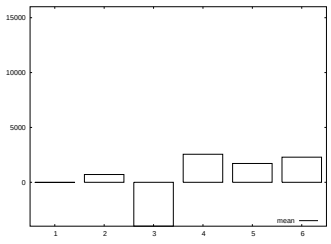
$\#_{tuiles}$ $+\#_{trous}$	combinaisons normales	...avec comportement aléatoire
$2 + 2$		
$3 + 2$		
$2 + 3$		

Tableau 4. *A quel point peut-on réutiliser des paramètres ?*

$(2 + 2)$ amène à des résultats très stables et très satisfaisants. C'est particulièrement remarquable dans les approches 4 et 6, ce qui fournit un argument en faveur de ces deux méthodes dans lesquelles les poids dépendent aussi de l'action. Ces algorithmes computationnellement plus complexes peuvent s'avérer intéressants avec un nombre croissant de comportements à combiner. D'autres exemples que le monde des tuiles devront toutefois être expérimentés pour renforcer cette hypothèse.

6. Discussion, perspectives

6.1. *Complétude et optimalité*

La principale idée sous-jacente à notre cadre de conception automatique est de décomposer la tâche des agents en sous-tâches, chacune étant plus simple. Une utilisation classique de l'apprentissage par renforcement revient à apprendre une politique

de la forme $\pi = F(\Delta)$ où Δ est un vecteur de $|\mathcal{O}| * |\mathcal{A}|$ paramètres à définir. Dans notre cas, nous partons d'un ensemble déjà défini de comportements de base $\mathcal{B}$ pour chercher une politique de la forme $\pi = F(\mathcal{B}, \Theta_{\mathcal{B}})$, avec seulement un paramètre θ_b par comportement de base. L'idée est bien évidemment de rendre l'apprentissage plus rapide et plus facile, en limitant le nombre d'optima locaux.

La contrepartie de cette approche est qu'il n'y a aucune garantie sur l'optimalité de la solution obtenue. Cela vient de ce que nous nous limitons à chercher des politiques dans un sous-ensemble de l'espace des solutions classiques. Si la politique optimale ne s'exprime pas sous la forme d'une combinaison des comportements de base, nous ne pourrons pas la trouver. Une question importante, à laquelle nous n'avons pas de réponse, est de savoir dans quelles conditions la famille de solutions que nous explorons est complète au sens de *capable de générer un comportement arbitraire donné*.

Pour augmenter la complétude de notre algorithme, il est possible d'utiliser un plus grand nombre de comportements de base, ce qui rend néanmoins l'apprentissage plus difficile car il y aurait plus de paramètres à adapter par apprentissage. Une meilleure solution pourrait alors être d'utiliser de meilleurs comportements de base, ou d'en ajouter d'autres quand un manque se fait clairement sentir. Il n'est pas non plus impossible que l'ajout d'un comportement de base "bruit-blanc" comme réalisé en section 5.3 participe en fait à augmenter la complétude de notre méthode. Enfin, le processus de combinaison lui-même pourrait être amélioré, sujet que la section 6.2 aborde plus en détail.

La deuxième raison qui explique le caractère sous-optimal de nos comportements extensibles tient au fait que nous utilisons des agents réactifs dans un cadre partiellement observé. Comme nous l'avons déjà évoqué, la recherche d'un comportement stochastique vise à répondre à cette difficulté. Une solution serait d'utiliser des comportements plus cognitifs, prenant en compte une partie du passé de l'agent. Idéalement, cette complexité accrue pourrait être confinée dans les comportements de base, ne changeant rien à la combinaison elle-même, laquelle garderait le nombre réduit de paramètres à adapter.

6.2. *Combinaison*

Un élément crucial de notre approche est le processus de combinaison adopté. Chacune des versions évaluées met en œuvre une règle simple supposée efficace pour tenir compte des différents comportements de base impliqués. Les règles utilisées ici traitent de manière aveugle les données qui leur sont fournies, alors qu'elles pourraient bénéficier d'informations de plus haut niveau, telles que savoir si les récompenses liées à différentes motivations peuvent être cumulées ou pas.

Un exemple simple est celui d'un agent-loup ayant une très bonne raison de **ne pas** aller à gauche : un buisson épineux, mais à qui 27 autres moins bonnes raisons disent d'aller vers la gauche, promettant des récompenses positives : 27 moutons. Il faut alors distinguer deux cas :

– soit ces récompenses peuvent toutes s'additionner : il y a 27 sources de nourriture dont on pourra profiter si celles-ci ne peuvent fuir,

– soit seule une des récompenses sera accessible si les autres s'échappent.

Dans le premier cas, il est vraisemblable que traverser le buisson est la meilleure option vu le gain cumulé possible. Mais dans le second cas, une seule des 27 paires (bb, cfg) incitant à aller vers la gauche devrait être considérée. Une règle adaptée est alors de choisir la paire la plus prometteuse.

Cet exemple montre que les liens entre deux paires (bb, cfg) peuvent être de forme *additive* ou *concurrentielle*, et que le traitement effectué devrait y être adapté. Mais la relation entre les comportements de base `éviter le buisson` et `chasser une proie` est d'un autre type : le premier est une contrainte pour le second. De même, un comportement lié à une ressource aura une plus grande importance si cette ressource vient à manquer, ce qui peut être exprimé en reliant le poids de tels comportements à des niveaux de ressource.

Idéalement, la manière de combiner les comportements de base, et donc les relations entre eux, ne devrait pas être fixée a priori mais être elle-même sujette à apprentissage.

6.3. *Extensibilité, passage à l'échelle*

Le fait de n'adapter qu'un paramètre par comportement de base, même quand ce comportement peut être impliqué plusieurs fois dans l'étape de combinaison, favorise le côté extensible de notre approche. Les paramètres s'avèrent d'ailleurs réutilisables quand le nombre d'objets change (voir les expériences de la section 5.4). Mais comme nous venons de le voir, cette capacité est limitée, la méthode de combinaison n'étant qu'une heuristique.

Il reste néanmoins un problème latent à notre approche, déjà évoqué en section 3.3.1. Il s'agit de la reconnaissance des différents comportements de base potentiellement applicables à une configuration observée. La recherche que nous faisons actuellement est exhaustive, de complexité exponentielle, et donc limitée à des agents ne percevant qu'un nombre réduit de percepts. L'extensibilité de notre approche dépend donc essentiellement de cette phase qui est à cheval sur la perception et la combinaison. Pour l'instant, nous n'avons pas exploré plus avant ce point délicat, mais deux axes de recherche paraissent prometteurs :

– utiliser des algorithmes plus efficaces pour rechercher les configurations compatibles. Une idée serait de ne faire les mises à jour des configurations que quand un objet apparaît ou disparaît ;

– ne pas faire une recherche exhaustive si l'on considère qu'un comportement de base b_1 en subsume un autre b_2 : les objets liés à une configuration de type $C^T_{b_1}$ ne doivent pas être considérés pour b_2. C'est une question très ouverte, qui demande notamment de se pencher sur les rôles et caractéristiques des comportements de base.

Par contre, la taille de l'environnement ne semble pas être un facteur limitant de notre approche car nos agents n'ont qu'une vision partielle et limitée de celui-ci. Cela reste bien sûr à confirmer par l'expérimentation sur d'autres problèmes.

Le passage à l'échelle de notre méthode semble ainsi plus lié au nombre moyen de percepts vus par un agent plutôt qu'au nombre total d'objets du système ou à la taille de ce dernier.

6.4. *Transfert, réutilisation, généricité*

Ainsi que nous l'avons expliqué dans la partie qui détaille la combinaison de comportements (section 3), un aspect crucial de la généricité de notre approche s'appuie sur le fait que nous utilisons des *types de configuration* liés aux comportements et des *paramètres d'équilibrage (θ)* lors de la combinaison. Comme l'ont montré les expériences, ce type de combinaison donne de bons résultats, et devrait bien s'adapter à des tâches nouvelles : le paramètre θ est le seul paramètre à adapter et il n'y a pas besoin de modifier les comportements de base appris quand on s'intéresse à une autre tâche globale.

C'est une des caractéristiques importantes de notre méthode. En effet, aussi bien dans le domaine de l'apprentissage par renforcement que dans le domaine de la sélection d'action, les travaux précédents requièrent une adaptation des comportements de base à la tâche globale, comme nous l'avons déjà mentionné en section 2.3.

Une idée pour améliorer la généricité de notre approche est de rendre l'agent capable de définir des *types d'objets abstraits*. Actuellement, la connaissance des types d'objets présents dans l'environnement est donnée à l'agent : on suppose fourni un module de perception identifiant les types utiles à distinguer. Mais nous pensons que l'on pourrait se servir des données de l'apprentissage pour catégoriser automatiquement les objets en classes comme par exemple "obstacles", "but", "à bouger", etc. Il s'agirait là d'un pas supplémentaire vers un véritable méta-apprentissage.

7. Conclusion

Après avoir discuté du problème de l'apprentissage par renforcement dans un cas où plusieurs motivations sont concurrentes et après avoir introduit le domaine de la sélection d'action, le présent article a proposé une *architecture de sélection d'action* se basant sur des comportements de base appris par renforcement, et répondant aux souhaits auxquels l'analyse du problème nous a conduit :

– que ce soit une *architecture de type flux libre* (de façon à trouver un compromis parmi les comportements concurrents),

– que le *comportement produit soit stochastique* (ce qui évite dans une certaine mesure de provoquer des blocages), et enfin

– que les *paramètres de la combinaison soient, dans la mesure du possible, réglés automatiquement* (en faisant appel à des méthodes d'apprentissage par renforcement).

L'article a présenté différents opérateurs de combinaison de comportements de base et les principaux enseignements qui en ont été tirés.

Les travaux effectués ont dans l'ensemble répondu à ces attentes, même s'il ne s'agit que d'heuristiques, et que des formes de combinaison plus perfectionnées existent probablement. Ils ont aussi soulevé quelques difficultés persistantes ou aspects intéressants, parmi lesquels on peut citer :

– la réutilisabilité des paramètres θ appris : ils peuvent efficacement servir de base dans des situations plus complexes que celles auxquelles ils étaient dédiés ;

– l'intérêt, même s'il reste limité, d'ajouter un comportement de base "aléatoire" qui permet d'ajouter du bruit, est de sortir de blocages persistants ; et

– la grande perte de temps et donc d'efficacité liée à ces mêmes blocages, sur lesquels il faudrait donc en priorité concentrer l'amélioration de la prise de décision.

Sur ce dernier point, l'observation d'agents en action montre que l'on est très généralement proche d'un bon comportement. De là, des travaux complémentaires ont conduit à une méthode pour déterminer quels comportements de base utiliser (Buffet, 2003, Dutech *et al.*, 2004).

L'ensemble de la démarche a donc permis la conception d'un agent "extensible", dans le sens qu'il est capable de gérer avec succès un certain nombre de motivations simultanées. Les perspectives évoquées dans la discussion en section 6 sont nombreuses, mais il semble prioritaire de mettre en application notre approche sur divers problèmes afin d'approfondir son analyse.

8. Bibliographie

Baxter J., Bartlett P., « Infi nite-Horizon Policy-Gradient Estimation », *Journal of Artificial Intelligence Research*, vol. 15, p. 319-350, 2001a.

Baxter J., Bartlett P., Weaver L., « Experiments with Infi nite-Horizon, Policy-Gradient Estimation », *Journal of Artificial Intelligence Research*, vol. 15, p. 351-381, 2001b.

Bertsekas D., *Dynamic Programming : Deterministic and Stochastic Models*, Englewood Cliffs, NJ : Prentice-Hall, 1987.

Boutilier C., Dearden R., Goldszmidt M., « Stochastic Dynamic Programming with Factored Representations », *Artificial Intelligence*, vol. 121, n° 1-2, p. 49-107, August, 2000.

Brooks R., « A Robot that Walks ; Emergent Behavior from a Carefully Evolved Network », *Neural Computation*, vol. 1, n° 2, p. 253-262, 1989.

Buffet O., Une double approche modulaire de l'apprentissage par renforcement pour des agents intelligents adaptatifs, PhD thesis, Université Henri Poincaré, Nancy 1, septembre, 2003. Laboratoire Lorrain de recherche en informatique et ses applications (LORIA).

Cassandra A. R., Kaelbling L. P., Littman M. L., « Acting Optimally in Partially Observable Stochastic Domains », *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, Seattle, Washington, USA, p. 1023-1028, 1994.

Chadès I., Planification Distribuée dans les Systèmes Multi-agents à l'aide de Processus Décisionnels de Markov, PhD thesis, Université Henri Poincaré, Nancy 1, 2003.

Dietterich T., « Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition », *Journal of Artificial Intelligence Research*, vol. 13, p. 227-303, 2000.

Dutech A., Apprentissage d'environnement : approches cognitives et comportementales, PhD thesis, ENSAE, Toulouse, 1999.

Dutech A., Buffet O., Charpillet F., « Développement autonome des comportements de base d'un agent », *Actes de la Conférence d'Apprentissage (CAp'04), Montpellier, France*, 2004.

Dutech A., Samuelides M., « Un algorithme d'apprentissage par renforcement pour les processus décisionnels de Markov partiellement observés : apprendre une extension sélective du passé », *Revue d'Intelligence Artificielle*, vol. 17, n° 4, p. 559-589, 2003.

Guestrin C., Koller D., Parr R., Venkataraman S., « Efficient Solution Algorithms for Factored MDPs », *Journal of Artificial Intelligence Research (JAIR)*, vol. 19, p. 399-468, 2003.

Hengst B., « Discovering Hierarchy in Reinforcement Learning with HEXQ », *Proceedings of the Nineteenth International Conference on Machine Learning (ICML'02)*, p. 243-250, 2002.

Humphrys M., Action Selection methods using Reinforcement Learning, PhD thesis, University of Cambridge, 1997.

Joslin D., Nunes A., Pollack M. E., TileWorld Users' Manual, Technical Report n° TR 93-12, August, 1993.

Kaelbling L., Littman M., Moore A., « Reinforcement Learning : A Survey », *Journal of Artificial Intelligence Research*, vol. 4, p. 237-285, 1996.

Laarhoven P. V., *Simulated annealing : theory and applications*, Kluwer academic publishers, Dordrecht, Boston, London, 1987.

Laurent G., Synthèse de comportements par apprentissages par renforcement parallèles, PhD thesis, Université de Franche-Comté, 2002.

Lin L.-J., « Hierarchical Learning of Robot Skills », *Proceedings of the IEEE International Conference on Neural Networks (ICNN'93)*, 1993.

Maes P., « A Bottom-Up Mechanism for Behaviour Selection in an Artificial Creature », *From Animals to Animats 1 : Proceedings of the First International Conference on Simulation of Adaptive Behavior (SAB'91)*, 1991.

Mahadevan S., Connell J., « Automatic Programming of Behavior-based Robots using Reinforcement Learning », *Artificial Intelligence*, vol. 55, n° 2-3, p. 311-365, June, 1992.

Matarić M., « Reinforcement Learning in the Multi-Robot Domain », *Autonomous Robots*, vol. 4(1), p. 73-83, 1997.

Monahan G., « A survey of partially observable Markov decision processes », *Management Science*, vol. 28, p. 1-16, 1982.

Puterman M. L., *Markov Decision Processes– Discrete Stochastic Dynamic Programming*, John Wiley and Sons, Inc., New York, USA, 1994.

Singh S., Cohn D., « How to Dynamically Merge Markov Decision Processes », *Advances in Neural Information Processing Systems 10 (NIPS'98)*, 1998.

Singh S., Jaakkola T., Jordan M., « Learning without state estimation in Partially Observable Markovian Decision Processes », *Proceedings of the Eleventh International Conference on Machine Learning (ICML'94)*, 1994.

Smallwood R., Sondik E., « The Optimal Control of Partially Observable Markov Decision Processes over a Finite Horizon », *Operation Research*, vol. 21, p. 1071-1088, 1973.

Sutton R., Barto G., *Reinforcement Learning : an introduction*, Bradford Book, MIT Press, Cambridge, MA, 1998a.

Sutton R., Precup D., Singh S., Between MDPs and Semi-MDPs : Learning, planning, and representing knowledge at multiple temporal scales, Technical report, University of Massachusetts, Dept. of Computer and Information Sciences, Amherst, MA, 1998b.

Tyrrell T., Computational Mechanisms for Action Selection, PhD thesis, University of Edinburgh, 1993.

Watkins C., Learning from delayed rewards, PhD thesis, King's College of Cambridge, UK., 1989.

Wooldridge M., Müller J.-P., Tambe M., « Agent theories, architectures and languages : A bibliography », *Intelligent Agents II, IJCAI'95 Workshop*, p. 408-431, 1995.

Annexe Contre-exemple : pourquoi les Q-valeurs ne donnent pas toujours une bonne indication de la politique stochastique à suivre dans un PDMPO

En considérant les manières possibles d'obtenir une politique stochastique à partir de comportements de base, nous avons considéré l'idée d'utiliser les Q-valeurs pour en déduire les probabilités des décisions possibles. Nous allons toutefois voir ici que ce choix intuitif peut être mauvais.

Pour montrer que les Q-valeurs ne sont pas directement liées à la politique, considérons le PDMPO représenté sur la figure 7, sans faire l'hypothèse qu'un algorithme d'apprentissage par renforcement particulier est utilisé. Une unique observation o cache à tout moment l'un des deux états s_1 ou s_2. Les transitions dépendent des deux actions disponibles a et b, et sont toutes déterministes, excepté quand on choisit l'action b dans l'état s_1 : probabilité p de rester en s_1 et $(1-p)$ d'aller en s_2. La seule récompense est gagnée en cas de transition $(s_1, b) \rightarrow s_1$.

Supposons que p soit proche de 1. La politique optimale est alors de choisir l'action b la plupart du temps (a est requise de temps en temps, de manière à sortir de l'état s_2), l'état courant étant généralement s_1 :

$$P(o,a) \simeq 0 \quad \& \quad P(o,b) \simeq 1, \qquad [14]$$

Ainsi $Q^*(o,b) \simeq Q^*(s_1,b)$ est pratiquement la valeur optimale du PDMPO : V^*. Avec un facteur d'actualisation γ égal à 0.9, on a :

$$\begin{aligned} Q^*(o,a) &= 0 + \gamma * (P(s=s_1) * V^*(s_1) + P(s=s_2) * V^*(s_2)) \qquad [15] \\ &= \gamma * V^*(s_1) \simeq 0.9 * V^* \qquad [16] \end{aligned}$$

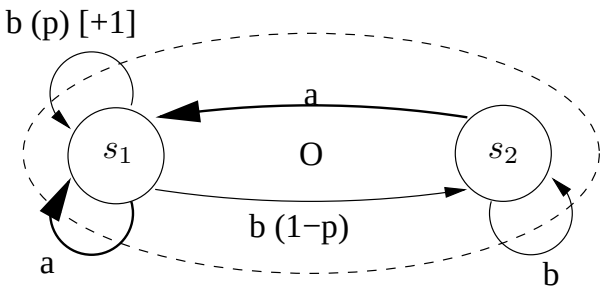


Figure 7. *Un exemple de PDMPO dont la politique optimale donne des Q-valeurs assez inattendues*

Les deux Q-valeurs ont des valeurs très proches, alors que les probabilités associées aux deux actions sont complètement opposées. Cela montre qu'il n'y a pas de lien fort entre les Q-valeurs et les probabilités d'action.

Si un Q-learning boltzmannien adapté était utilisé sur cet exemple simple, la température pourrait être réglée automatiquement pour atteindre une politique optimale. Mais dans des situations plus complexes, il y a bien plus que cet unique degré de liberté à gérer. On a toutefois gardé la combinaison de Q-valeurs comme un algorithme de référence dans nos expérimentations.