Apprentissage par renforcement dans le cadre des processus décisionnels de Markov factorisés observables dans le désordre. Etude expérimentale du Q-Learning parallèle appliqué aux problèmes du labyrinthe et du New York Driving.

Guillaume J. Laurent, Emmanuel Piat

Apprentissage par renforcement dans le cadre des processus décisionnels de Markov factorisés observables dans le désordre

Étude expérimentale du Q-Learning parallèle appliqué aux problèmes du labyrinthe et du New York Driving

Guillaume J. Laurent — Emmanuel Piat

Laboratoire d'Automatique de Besançon - UMR CNRS 6596
24 rue Alain Savary, F-25000 Besançon
glaurent@ens2m.fr, epiat@ens2m.fr

RÉSUMÉ. *Cet article présente les résultats expérimentaux obtenus avec une architecture originale permettant un apprentissage générique dans le cadre de processus décisionnels de Markov factorisés observables dans le désordre (PDMFOD). L'article décrit tout d'abord le cadre formel des PDMFOD puis le fonctionnement de l'algorithme, notamment le principe de parallélisation et l'attribution dynamique des récompenses. L'architecture est ensuite appliquée à deux problèmes de navigation, l'un dans un labyrinthe et l'autre dans un trafic routier (New York Driving). Les tests montrent que l'architecture permet effectivement d'apprendre une politique de décisions performante et générique malgré le nombre élevé de dimensions des espaces d'états des deux systèmes.*

ABSTRACT. *This paper presents experimental results obtained with an original architecture that can do generic learning for randomly observable factored Markov decision process (ROFMDP). First, the paper describes the theoretical framework of ROFMDP and the working of this algorithm, in particular the parallelization principle and the dynamic reward allocation process. Then, the architecture is applied to two navigation problems (gridworld and New York Driving). The tests show that the architecture allows to learn a good and generic policy in spite of the large dimensions of the state spaces of both systems.*

MOTS-CLÉS : *apprentissage par renforcement, Q-Learning, W-Learning, DBN-MDP, PDM factorisé, PDMFOD.*

KEYWORDS: *reinforcement learning, Q-Learning, W-Learning, DBN-MDP, factored MDP, ROFMDP.*

1. Introduction

L'apprentissage par renforcement a pour objectif d'élaborer des politiques de contrôle de processus dont le modèle d'évolution est inconnu. L'apprentissage est guidé par une valeur scalaire (appelée renforcement ou récompense) qui indique la qualité de la dernière transition effectuée par le processus (triplet état précédent–action–nouvel état). Sous certaines conditions, l'apprentissage converge vers une politique de décision maximisant la somme des renforcements futurs.

L'apprentissage par renforcement est une méthode prometteuse pour le contrôle de robots mobiles intelligents en environnements inconnus ou incertains. Néanmoins, l'apprentissage du contrôle de systèmes réels se heurte à de nombreuses difficultés : capteurs bruités, capteurs insuffisamment précis, observation partielle de l'environnement, nécessité d'apprendre en temps réel et en temps limité (Mahadevan, 1996). S'y ajoute une problématique courante en robotique mobile : celle de la mise en correspondance des données capteurs entre deux instants de mesure.

En général, un robot mobile est pourvu de nombreux capteurs proprioceptifs (position, vitesse, charge des moteurs, *etc.*) et extéroceptifs (contact, télémètre, caméra vidéo, *etc.*). La plupart du temps, les données extéroceptives ne sont pas étiquetées. Par exemple, c'est le cas lorsqu'on utilise un système de vision qui extrait des points caractéristiques à partir d'une image perçue par une caméra. Les points caractéristiques sont alors fournis par le système de vision avec un ordre qui peut varier dans le temps. Pour être informatifs sur l'évolution de l'environnement, ces points caractéristiques doivent être mis en correspondance d'une image à l'autre.

De plus, l'apprentissage doit être guidé par une fonction de renforcement. Pour obtenir certains comportements, cette fonction peut être calculée à partir de l'état de quelques capteurs. Par exemple, si un robot mobile est bloqué par un obstacle, un capteur de surcharge moteur pourrait engendrer un renforcement négatif car cette situation n'est pas souhaitable. Comment alors faire le lien entre cette valeur négative et les informations issues du système de vision pour éviter cette situation à l'avenir ? Il y a là encore nécessité de mettre en correspondance renforcements et observations de l'environnement.

Enfin, si l'on envisage par exemple de contrôler un véhicule dans un trafic routier, la configuration des véhicules avoisinants est en perpétuelle évolution. Dans ces cas de figures, il n'est pas envisageable d'apprendre une réaction pour chacune des configurations des véhicules car le nombre de ces configurations croît trop rapidement avec le nombre de véhicules présents. Un mécanisme de généralisation est donc indispensable.

Dans cet article, nous avons cherché à introduire dans le formalisme général des processus décisionnels de Markov cette problématique de mise en correspondance des données capteurs et des renforcements. L'article traite donc de l'apprentissage par

renforcement dans le cas de processus à grands espaces d'états et dont les observations sont non étiquetées *a priori* mais supposées exhaustives[1].

Nous proposons tout d'abord un cadre théorique pour définir formellement cette double problématique. Pour représenter des espaces d'états de grandes dimensions, nous nous sommes appuyés sur les processus décisionnels de Markov factorisés. La section 2 présente un cadre formel original, baptisé *processus décisionnel de Markov factorisé observable dans le désordre*, suivi d'une étude bibliographique sur ce thème.

Dans la troisième section, nous proposons un algorithme qui répond à notre problématique dans le cas particulier où les composantes du vecteur d'état évoluent indépendamment les unes des autres. A l'origine, cette architecture a été développée pour apprendre à piloter un manipulateur robotique 2D. Il s'agissait d'apprendre à diriger un outil mobile sur plan. L'objectif était de collecter des objets particuliers en évitant d'autres objets jamais placés de la même manière. Les résultats obtenus avec le système robotique sont disponibles dans (Laurent *et al.*, 2002; Laurent, 2003). Cette application est similaire à la simulation du labyrinthe expérimentée dans la section 4. L'intérêt du labyrinthe est d'être suffisamment simple pour permettre l'analyse des mécanismes sous-jacents de l'algorithme.

Au-delà de l'application robotique et du labyrinthe, nous avons comparé l'architecture développée avec d'autres algorithmes sur un banc d'essais académique, le « New York Driving » de Mc Callum (McCallum, 1995). La section 5 présente les résultats obtenus avec cette simulation.

2. Cadre théorique

Le cadre général de cet article est celui des processus décisionnels de Markov (PDM) et plus particulièrement celui de l'apprentissage par renforcement dans le cadre des PDM factorisés observables dans le désordre proposé plus loin.

2.1. *Processus décisionnels de Markov (PDM)*

Définition 1 *Un processus décisionnel de Markov (Bellman, 1957; Sutton* et al.*, 1998) fini et stationnaire est défini par un axe temporel discret régulier*[2] *fini ou infini*[3] *et par un quadruplet* $\langle \mathcal{X}, \mathcal{U}, \mathsf{T}, \mathsf{R} \rangle$ *où :*

– $\mathcal{X}$ *est un ensemble fini d'états* x*,*

– $\mathcal{U}(x)$ *est un ensemble fini d'actions* u *possibles dans l'état* x*,*

1. Cette problématique de mise en correspondance étant d'elle-même déjà complexe, nous avons pour l'instant évacué les autres problématiques énoncées préalablement comme l'observation partielle de l'environnement.
2. Prise de décision à intervalles réguliers.
3. On parle alors d'horizon fini ou infini.

– $\mathsf{T} : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \rightarrow [0;1]$ *est une fonction de transition, telle que :*

$$\forall (x,u) \in \mathcal{X} \times \mathcal{U} \quad \sum_{x' \in \mathcal{X}} \mathsf{T}(x,u,x') = 1, \qquad [1]$$

– $\mathsf{R} : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \rightarrow \mathcal{F}(\mathbb{R})$ *est une fonction de renforcement qui associe à chaque transition une densité de probabilité (bornée dans* $\mathbb{R}$*).*

Soit x_k l'état du système à l'instant k (*i.e.* la réalisation de la variable aléatoire $\underline{x}_k$ à l'instant k). Si, à cet instant k, on applique une action u_k au système, ce dernier peut évoluer dans n'importe lequel des états possibles du système selon la distribution de probabilités donnée par la fonction T, soit :

$$\forall x' \in \mathcal{X} \quad P_r(\underline{x}_{k+1} = x' | \underline{x}_k = x_k, \underline{u}_k = u_k) = \mathsf{T}(x_k, u_k, x') \qquad [2]$$

Dans le cadre de l'apprentissage par renforcement, les probabilités T *de transition ainsi que la fonction* R *ne sont pas connues.*

Le critère à optimiser est défini par une fonction appelée fonction de valeur[4] $\mathsf{Q}^\pi(x,u)$ qui correspond à l'espérance (mathématique) de gain si l'on suit une politique d'actions π donnée dans le futur sachant que le système se trouve dans l'état x et que l'on a effectué l'action u. On appelle *gain* la somme (pondérée ou non) des renforcements futurs.

Dans cet article, nous utilisons le critère $gamma$-pondéré défini par la fonction de valeur Q suivante :

$$\mathsf{Q}^\pi(x,u) = E\left\{ \sum_{i=0}^{\infty} \gamma^i \underline{r}_{i+k+1} | \underline{x}_k = x, \underline{u}_k = u, \pi \right\} \quad 0 \leq \gamma < 1 \qquad [3]$$

γ est un coefficient dit d'actualisation.

Si l'on connaît la fonction de valeur optimale Q^*, alors une politique de contrôle optimale consiste à toujours choisir une action u pour laquelle $\mathsf{Q}^*(x,u)$ est maximale pour x fixé. Le but de l'agent est donc de trouver la fonction de valeur optimale Q^*.

Pour les PDM dont l'espace d'états est de grande dimension, un cadre spécifique a été proposé, celui des processus décisionnels de Markov factorisés.

2.2. *Processus décisionnels de Markov factorisés (PDMF)*

Définition 2 *Un processus décisionnel de Markov factorisé (Boutilier* et al.*, 1999; Guestrin* et al.*, 2003; Kearns* et al.*, 1999) fini et stationnaire est défini par un axe temporel discret régulier fini ou infini et par un n-uplet* $\langle \mathcal{X}, \mathcal{U}, \mathsf{T}, \mathsf{r}^1, \ldots, \mathsf{r}^m \rangle$ *où :*

4. La fonction de valeur est aussi appelée fonction de qualité.

– $\mathcal{X}$ est un espace fini d'états de la forme $\mathcal{X} = \mathcal{X}^1 \times \ldots \times \mathcal{X}^n$, un état $X \in \mathcal{X}$ s'écrit $X = (x^1, \ldots, x^n)$ avec $x^i \in \mathcal{X}^i$

– $\mathcal{U}$ est un ensemble fini d'actions,

– $\mathsf{T} : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \rightarrow [0; 1]$ est une fonction de transition habituellement définie par un réseau bayésien dynamique,

– $\mathsf{r}^j : \mathcal{C}^j \times \mathcal{U} \times \mathcal{C}^j \rightarrow \mathcal{F}(\mathbb{R})$ est un ensemble de m fonctions de renforcement élémentaires associées à m sous-espaces $\mathcal{C}^j$ de $\mathcal{X}$ ($\mathcal{C}^j \subset \mathcal{X}$), le signal de renforcement est un vecteur de la forme $R = (r^1, \ldots, r^m)$.

La fonction de renforcement globale associée à un PDMF est généralement définie par :

$$\mathsf{R}(X_k, u, X_{k+1}) = \sum_{j=1}^{m} r_{k+1}^j \qquad [4]$$

La dynamique d'évolution des PDMF est généralement représentée sous une forme compacte à l'aide de réseaux bayésiens dynamiques (DBN) et de tables de probabilités conditionnelles (CPT). A chaque action que peut effectuer l'agent correspond alors un réseau bayésien donné et on parle de DBN-MDP (Boutilier *et al.*, 1999).

Lorsque les réseaux et les tables sont connus, de nombreux algorithmes de planification de stratégies d'actions optimales ou sous-optimales sont possibles (Boutilier *et al.*, 2000; Guestrin *et al.*, 2003). Les travaux effectués en apprentissage par renforcement dans le cadre de DBN-MDP supposent généralement que la structure des réseaux bayésiens est connue mais pas les tables de probabilités conditionnelles (Kearns *et al.*, 1998; Kearns *et al.*, 1999; Sallans, 2002).

Notre approche se situe dans un cadre théorique dérivé des PDMF mais dont les vecteurs d'états et les valeurs de renforcements ne sont pas directement observables.

2.3. *Processus décisionnels de Markov factorisés observables dans le désordre (PDMFOD)*

Définition 3 *Un processus décisionnel de Markov factorisé observable dans le désordre (PDMFOD) est défini par un axe temporel discret régulier fini ou infini et par un n-uplet $\langle \mathcal{X}, \mathcal{U}, \mathsf{T}, \mathsf{r}^1, \ldots, \mathsf{r}^m, \sigma_k, \rho_k \rangle$ où :*

– $\langle \mathcal{X}, \mathcal{U}, \mathsf{T}, \mathsf{r}^1, \ldots, \mathsf{r}^m \rangle$ est un PDM factorisé (que l'on qualifie de sous-jacent),

– $\sigma_k \in \mathcal{S}_n$ est une permutation appelée fonction d'observation de l'état, σ_k est non stationnaire ($\mathcal{S}_n$ est le groupe symétrique de degré n et n la dimension du vecteur d'état),

– $\rho_k \in \mathcal{S}_m$ est une permutation appelée fonction d'observation du renforcement, ρ_k est non stationnaire,

Si à un instant k, le système est dans l'état $X_k = (x_k^1, \ldots, x_k^n)$ et le vecteur de renforcement est $R_k = (r_k^1, \ldots, r_k^m)$, l'agent perçoit alors Y_k qui correspond au vecteur d'état dans le désordre, soit :

$$Y_k = (y_k^1, \ldots, y_k^n) = (x_k^{\sigma_k(1)}, \ldots, x_k^{\sigma_k(n)}) \quad [5]$$

et observe S_k qui correspond au vecteur de renforcement dans le désordre, tel que :

$$S_k = (s_k^1, \ldots, s_k^m) = (r_k^{\rho_k(1)}, \ldots, r_k^{\rho_k(m)}) \quad [6]$$

Les permutations σ_k et ρ_k ne sont évidemment pas connues et supposées non stationnaires.

Ainsi, l'observation y_k^i n'a *a priori* aucun rapport avec l'observation précédente y_{k-1}^i. Dans ces conditions, l'agent doit mettre en correspondance les observations de l'instant k avec celles de l'instant $k-1$.

2.4. *Étude bibliographique*

En regard de cette double problématique – grand espace d'états et observations dans le désordre – nous avons cherché dans la littérature les algorithmes d'apprentissage par renforcement susceptibles de fonctionner dans le cadre des PDMFOD.

Les algorithmes tabulaires (Q-Learning, TD(λ), Dyna-Q) sont clairement limités à des systèmes dont l'espace d'états est de faible dimension. Pour apprendre à contrôler un système ayant un grand espace d'états, de nombreuses alternatives ont été proposées, notamment des méthodes utilisant des fonctions d'approximation linéaires, des réseaux de neurones artificiels, des décompositions hiérarchiques du problème ou enfin une représentation factorisée de l'état. Le cas des PDM factorisés ayant été abordé dans la section 2.2, nous allons maintenant détailler brièvement les autres approches. Nous terminerons par une présentation de l'architecture du W-learning qui est à l'origine du Q-learning parallèle développé ici.

Les algorithmes utilisant des méthodes d'approximation linéaires permettent de réduire sensiblement le nombre de valeurs à optimiser lors d'un apprentissage (CMAC (Albus, 1975), RBF (Sutton *et al.*, 1998), *etc.*). La convergence de l'apprentissage est assurée, mais rien n'indique que la politique obtenue soit optimale (Bertsekas *et al.*, 1996). Ces méthodes d'approximation linéaires sont particulièrement bien adaptées au contrôle de processus continus dont l'espace d'états est de dimension faible (deux ou trois).

Pour les espaces de grande dimension, il est possible d'utiliser des structures type réseaux de neurones artificiels. Les réseaux de neurones ont prouvé que, dans certains cas, ils permettent d'obtenir un excellent apprentissage alors que l'espace d'états est de grande dimension (TD-Gammon (Tesauro, 1992), ascenseurs (Crites *et al.*, 1996), robots nageurs poly-articulés (Coulom, 2002)). Néanmoins, en apprentissage par renforcement, l'utilisation de réseaux de neurones constitue un apprentissage hors i.i.d.

(non indépendamment et identiquement distribué). Il n'existe pas de preuve de convergence des algorithmes vers une politique optimale, ni de méthode bien définie pour construire un réseau adapté à un système donné.

Les structures hiérarchiques offrent des alternatives intéressantes aux réseaux de neurones. Leur principe est de découper le problème global en sous-problèmes élémentaires plus simples à apprendre. Cette idée correspond bien à notre besoin de généralisation. Par exemple, le problème du labyrinthe comporte de nombreux éléments redondants (les murs sont identiques). Plutôt que d'apprendre à réagir face à une configuration spatiale particulière de plusieurs murs, une solution consiste à apprendre à réagir par rapport à un seul mur et à chercher un moyen pour en déduire une action pour une configuration de plusieurs murs.

Il existe de nombreuses structures hiérarchiques. Beaucoup d'entre elles sont des architectures compétitives à deux niveaux : un bas niveau correspondant aux comportements élémentaires comme aller vers un objet ou éviter un mur et un haut niveau qui a pour rôle de choisir le comportement de bas niveau à exécuter en fonction de la situation. Ce sélecteur de haut niveau est réalisé soit par apprentissage soit par une fonction dédiée. On peut notamment citer le Hierarchical Learning de Lin (Long-Ji, 1993; Faihe, 1999), le CQ-L (Singh, 1992; Tham *et al.*, 1994), la coordination par subsomption (Mahadevan *et al.*, 1992). Ces structures permettent d'apprendre plus vite qu'une architecture « plate ».

D'autres architectures à deux niveaux sont coopératives comme le W-Learning (Humphrys, 1995; Humphrys, 1996; Humphrys, 1997) ou la coordination hybride (Carreras, 2003), ce qui permet le mélange des comportements de bas niveaux. L'intérêt de ces architectures est de pouvoir générer des actions de compromis qui répondent aux objectifs d'une majorité de comportements de bas niveaux.

Des architectures à plus de deux niveaux utilisant des actions et des états abstraits de hauts niveaux ont été proposées (MAXQ (Dietterich, 2000), HDG Learning (Kaelbling, 1993), Feudal Reinforcement Learning (Dayan *et al.*, 1993), HQ-Learning (Wiering *et al.*, 1997)). Elles sont plutôt adaptées à un découpage temporel d'un problème en sous-objectifs à atteindre successivement dans le temps.

Parmi toutes ces architectures et en regard de notre problématique, le W-Learning a retenu notre attention. Cette architecture est constituée de plusieurs modules de bas niveau d'indice i percevant chacun à chaque instant k une partie x_k^i seulement de l'état X_k du système tel que $X_k = (x_k^1, x_k^2, \ldots, x_k^n)$ (avec par exemple x_k^i désignant la position d'un seul objet). Chaque module utilise l'algorithme du Q-Learning pour calculer en sortie une estimation de l'espérance de gain $\mathsf{Q}_i(x_k^i, u)$ pour chaque action possible u. Un module de coordination détermine l'action à effectuer en fonction des valeurs de Q_i. Chaque module perçoit alors une récompense liée à sa perception pour remettre à jour sa fonction de valeur. La figure 1 propose un exemple d'architecture W-Learning avec une version simplifiée d'un des systèmes testés par Humphrys : un

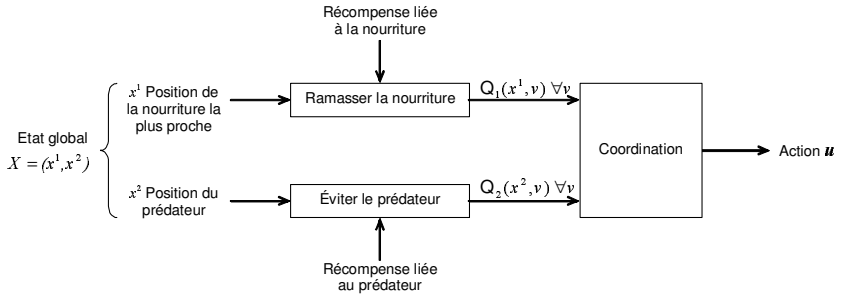


Figure 1. *Exemple d'architecture W-Learning dans le cas d'un robot dont le but est de ramasser de la nourriture en évitant un prédateur mobile*

agent doit collecter de la nourriture en évitant un prédateur mobile[5]. Cette architecture permet de réduire la complexité de l'apprentissage puisque chaque module effectue son apprentissage par rapport à une partie seulement de l'état du système, donc dans un sous-espace de l'espace d'états.

Le W-Learning nécessite de décrire le rôle et le nombre des modules de bas niveau et de définir les récompenses associées à chaque module. L'architecture du *Q-Learning parallèle*[6] que nous présentons ici est en quelque sorte une généralisation du W-Learning mais qui ne nécessite pas ces informations. Le Q-Learning parallèle ne s'applique *a priori* qu'à des problèmes de type PDM factorisés observables dans le désordre dont les sous-états x^i évoluent de façon indépendante. Il pourra néanmoins être utilisé dans certain cas sur des problèmes dans lesquels il existe des couplages faibles entre sous-états et non pas une stricte indépendance (comme dans l'exemple du labyrinthe présenté plus loin).

3. Q-Learning parallèle

3.1. *Hypothèses de travail*

Par rapport au cadre formel des PDMFOD, l'architecture parallèle que nous avons développée se place dans un cadre plus restreint. La plupart des travaux sur les PDMF supposent connue la structure des réseaux bayésiens dynamiques qui décrit la fonction de transition. Pour répondre à la problématique de l'observabilité dans le désordre, le Q-Learning parallèle suppose en plus que les fonctions de transition et de renforcement ont une forme particulière.

---

5. La version originale baptisée *Ant-World* simule la collecte de nourriture par une fourmi puis le transport vers son nid tout en évitant un prédateur mobile.
6. Nous avons choisi le terme « parallèle » non pas pour faire référence à un algorithme implantable sur un calculateur parallèle mais simplement parce que l'algorithme effectue *simultanément* plusieurs apprentissages de type Q-Learning.

3.1.1. *Notation*

Dans le cadre des PDMFOD, chaque x^i appartient a un espace particulier $\mathcal{X}^i$. Cette notation étant particulièrement lourde, nous allons dans le cadre de cet article supposer que tous les éléments x^i appartiennent à un même espace qu'on notera $\mathcal{X}$. Précisons néanmoins que si les x^i sont hétérogènes ($\mathcal{X}^i$ différent pour chacun), tous les résultats présentés dans cet article restent applicables en utilisant plusieurs $\mathcal{X}^i$ au lieu d'un seul $\mathcal{X}$.

3.1.2. *Hypothèse sur la fonction de transition*

On s'intéresse aux PDMFOD dont l'évolution de chacun des sous-états x^i du PDMF sous-jacent est indépendante. En d'autre termes, l'évolution d'un sous-état x^i_k à l'instant $k+1$ ne dépend que x^i_k et de u_k.

Hypothèse 1 *L'évolution d'un sous-état x^i_k du PDMF sous-jacent ne dépend que de x^i_k et de u_k, c'est-à-dire :*

$$\forall x' \in \mathcal{X} \quad P_r(\underline{x}^i_{k+1} = x' | x^i_k, u_k) = P_r(\underline{x}^i_{k+1} = x' | X_k, u_k) \qquad [7]$$

Comme les sous-états appartiennent au même[7] espace $\mathcal{X}$, cette hypothèse implique qu'il existe une fonction t décrivant les probabilités de transition entre sous-états telle que $\mathsf{t}(x, u, x')$ donne la probabilité d'obtenir le sous-état x' après avoir effectué l'action u et observé le sous-état x.

La fonction globale de transition du PDMF sous-jacent est donc supposée de la forme :

$$\forall X' \in \mathcal{X} \quad \mathsf{T}(X, u, X') = \mathsf{t}(x^1, u, x'^1) \times \mathsf{t}(x^2, u, x'^2) \times \ldots \mathsf{t}(x^n, u, x'^n) \quad [8]$$

avec : $X = (x^1, x^2, \ldots, x^n)$, $X' = (x'^1, x'^2, \ldots, x'^n)$

Dans ce contexte, l'utilisation de réseaux bayésiens dynamiques qui permettent à l'inverse de modéliser les dépendances entre les variables n'est plus indispensable.

3.1.3. *Hypothèses sur la fonction de renforcement*

D'une part, on suppose que toutes les fonctions de renforcement élémentaires du PDMF sous-jacent r^j sont identiques et qu'elles associent une valeur réelle à un triplet sous-état—action—sous-état.

Hypothèse 2 *Toutes les fonctions de renforcement élémentaires du PDMF sous-jacent r^j sont identiques*[8] *et définies de $\mathcal{X} \times \mathcal{U} \times \mathcal{X}$ dans $\mathcal{F}(\mathbb{R})$, c'est-à-dire :*

$$\forall j \in \{1, \ldots, m\} \quad \mathsf{r}^j = \mathsf{r} \quad \textit{et} \quad \mathsf{r} : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \to \mathcal{F}(\mathbb{R}) \qquad [9]$$

7. Dans le cas où plusieurs espaces $\mathcal{X}^i$ seraient utilisés alors l'hypothèse implique qu'il existe autant de fonctions t^i qu'il y a d'espaces $\mathcal{X}^i$.

8. Dans le cas où plusieurs espaces $\mathcal{X}^i$ seraient utilisés alors on suppose qu'il existe autant de fonctions $\mathsf{r}^i : \mathcal{X}^i \times \mathcal{U} \times \mathcal{X}^i \to \mathcal{F}(\mathbb{R})$ qu'il y a d'espaces $\mathcal{X}^i$.

D'autre part, on suppose que le vecteur de renforcement R_k du PDMF sous-jacent a autant[9] de composantes que le vecteur d'état X_k.

Hypothèse 3 *Une fonction de renforcement élémentaire* r *est associée à chaque composante de vecteur d'état* X_k *du PDMF sous-jacent.*

Sous ces hypothèses, le vecteur de renforcement du PDMF sous-jacent s'écrit à chaque instant k :

$$R_k = \left(\mathsf{r}(x_k^1, u_k, x_{k+1}^1),\ \ldots\ , \mathsf{r}(x_k^n, u_k, x_{k+1}^n)\right) \qquad [10]$$

avec n la dimension du vecteur X_k.

3.1.4. *Remarque sur la non-stationnarité de la dimension du vecteur d'état*

L'architecture parallèle telle qu'elle est construite ne fait pas d'hypothèse sur la dimension n du vecteur d'état. Il est donc tout à fait possible que cette dimension soit non stationnaire au cours du temps (ce qui est d'ailleurs le cas pour les simulations de labyrinthe et du New York Driving).

3.2. *Illustration*

Afin d'illustrer notre propos, nous introduisons ici notre banc d'essais « labyrinthe » (*cf.* figure 2). Dans ce labyrinthe, une souris peut se déplacer case par case à l'aide de quatre actions discrètes ($\mathcal{U}$={nord, sud, est, ouest}). Des fromages et des murs sont disposés au hasard dans le labyrinthe. Le système est déterministe.

L'objectif est de diriger la souris vers les fromages en évitant les murs. Si la souris atteint une case pourvue d'un fromage, elle est récompensée. Si la souris tente de passer sur un mur, elle ne bouge pas et est punie.

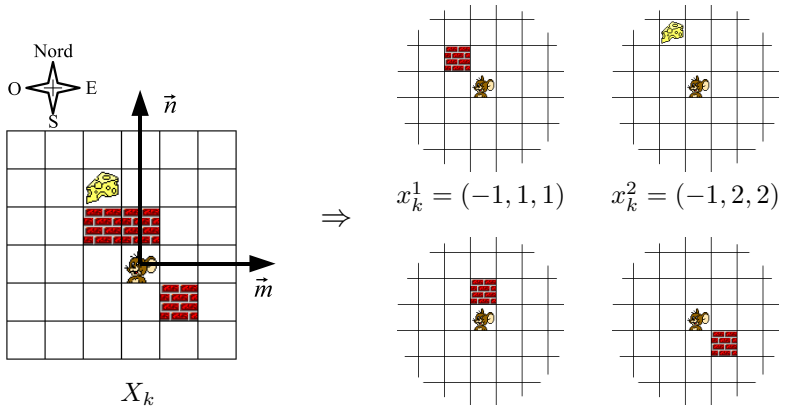
Pour ce système, un sous-état x_k est défini par l'abscisse m_k, l'ordonnée n_k et le type t d'un objet présent dans le labyrinthe (mur ou fromage) à l'instant k, soit :

$$x_k = (m_k, n_k, t) \qquad [11]$$

L'abscisse m_k et l'ordonnée n_k sont calculées *dans un repère lié à la souris* (la souris est donc immobile dans le repère considéré). Le type t prend la valeur 1 pour un mur et la valeur 2 pour un fromage (ce qui correspond à la « classe » de l'objet). Un sous-état représente donc un et un seul objet.

A titre d'exemple, dans la situation de la figure 2, le sous-état correspondant au fromage se note $x = (-1, 2, 2)$.

9. Néanmoins, l'approche reste valable si le vecteur de renforcement R_k a moins de composantes que le vecteur d'état X_k car il suffit dans ce cas d'ajouter des valeurs nulles pour vérifier l'hypothèse.



(a) État (b) Sous-états

Figure 2. *Structure du vecteur d'état*

L'état X_k du système regroupe l'ensemble des sous-états x_k^i à l'instant k. Dans notre exemple, l'état X_k est :

$$X_k = \big((-1, 1, 1), (-1, 2, 2), (0, 1, 1), (1, -1, 1)\big) \quad [12]$$

Le système est supposé totalement observable dans le désordre. Le vecteur observé contient systématiquement autant de sous-états qu'il y a d'objets. L'horizon de l'agent est infini : l'agent « voit » la totalité du labyrinthe et notamment « au travers » des murs qui l'entourent.

Dans l'exemple de la figure 2, si l'agent décide d'aller au nord, il percutera un mur, événement puni par un renforcement de -1. R_{k+1} contiendra donc une valeur négative et 3 récompenses nulles (R_{k+1}=(0,-1,0,0)) correspondant aux autres éléments perçus.

Dans un labyrinthe sans mur et parsemé de fromages, le déplacement relatif d'un fromage par rapport à la souris ne dépend que de son ancienne position et de la dernière action effectuée, quelles que soient les positions des autres fromages. Dans ce cas, les évolutions des sous-états sont complètement indépendantes.

En revanche, la présence de murs infranchissables couple légèrement le système : sans mur, si l'agent décide d'aller au nord alors il observera qu'un fromage donné se déplace d'une certaine manière par rapport à lui. Avec un mur au nord, la même action ne fera pas déplacer l'agent et l'évolution du même fromage sera différente de son point de vue. Ce faible couplage produit un effet qui s'apparente à un comportement légèrement stochastique des sous-états. Pour notre étude, cet effet ne pose pas de

problème car l'agent recevant un renforcement négatif s'il tente de franchir un mur, évitera de reproduire ce type d'action.

3.3. *Principe du Q-Learning parallèle*

3.3.1. *Exploitation de la structure supposée du PDMFOD*

Le principe de fonctionnement du Q-Learning parallèle est d'utiliser la forme particulière de l'espace d'état pour effectuer un apprentissage générique.

L'idée principale est d'utiliser une fonction de valeur définie sur l'espace $\mathcal{X}$ des sous-états x et non pas sur l'espace d'état global $\mathcal{X}$ des états X. Elle mémorise donc l'estimation $\mathsf{q}(x,u)$ pour tous les sous-états x de $\mathcal{X}$ et actions u de $\mathcal{U}$. A chaque évolution de X_k, la fonction de renforcement procède à un apprentissage de type Q-Learning pour chacun des n sous-états x_k^i de X_k en utilisant *toujours* la même fonction[10] de valeur q (*cf.* figure 3). La fonction de valeur que l'on cherche à optimiser est définie pour tout sous-état x^i et toute action u par :

$$\mathsf{q}^\pi(x^i,u) = E\left\{\sum_{t=0}^{\infty}\gamma^t \underline{r}_{t+k+1}^i | \underline{x}_k^i = x^i, \underline{u}_k = u, \pi\right\} \quad 0 \leq \gamma < 1 \qquad [13]$$

D'une part, cette méthode permet de réduire la portée de l'apprentissage puisque ce dernier s'effectue sur l'ensemble $\mathcal{X}$ des sous-états et non sur l'ensemble $\mathcal{X}$ des états du système. D'autre part, si l'état X comporte n sous-états, la fonction de valeur q sera mise à jour n fois à chaque instant k, ceci permet *a priori* de réduire le nombre d'itérations nécessaires à la convergence de q.

Néanmoins, l'agent apprend à optimiser l'espérance de gain par rapport aux sous-états et non par rapport à l'état global du système. Il est donc nécessaire de recourir à une étape de *fusion* pour décider de la meilleure action à prendre d'un point de vue global.

3.3.2. *Contraintes liées à l'observabilité du PDMFOD*

A chaque instant k, l'agent observe un vecteur $Y_k = (y_k^1, \ldots, y_k^n)$, Y_k étant une permutation non stationnaire du vecteur d'état X_k. Quel que soit i, le sous-état y_k^i n'est donc pas *a priori* le successeur de y_{k-1}^i.

Il est nécessaire d'ajouter un outil pour apparier les sous-états de l'observation Y_{k-1} avec ceux de l'observation Y_k. C'est le rôle de la fonction de *mise en correspondance temporelle*.

---

10. On peut noter que si on n'inclut pas la « classe » t du sous-état dans x_k^i (*cf.* section 3.1.1) alors il sera nécessaire d'utiliser autant de fonctions q qu'il y a de classes de sous-états (par exemple deux fonctions q dans le cas du labyrinthe).

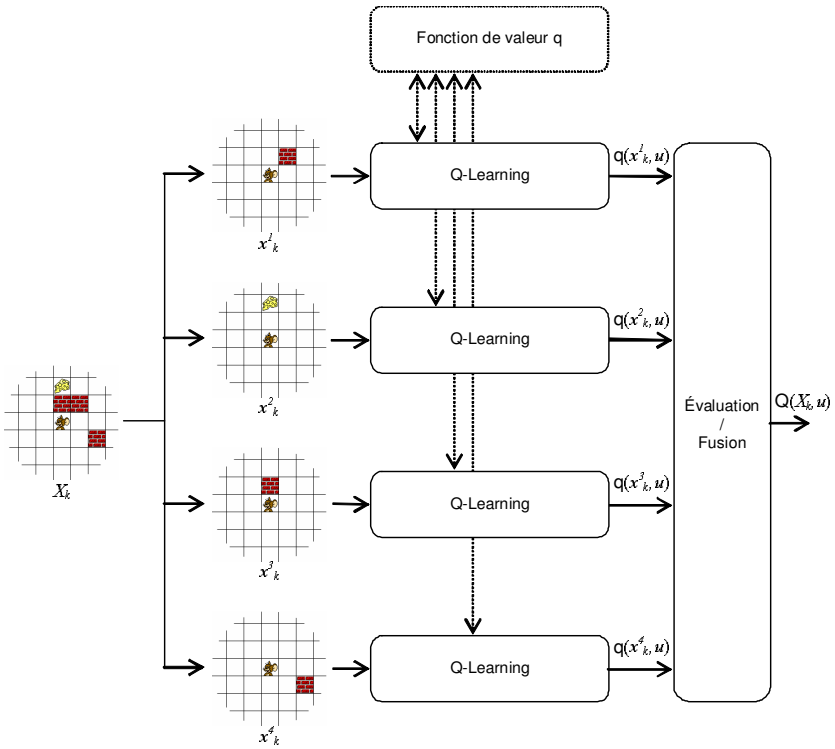


Figure 3. *Principe du Q-Learning parallèle*

Figure 4. *Illustration du problème de l'attribution des récompenses : à quel sous-état la récompense obtenue correspond-elle ?*

Par ailleurs, l'agent reçoit le vecteur de renforcement $S_k = (s_k^1, \ldots, s_k^n)$ permuté par rapport au vecteur R_k du PDMF sous-jacent. Chaque composante de R_k découle de la transition d'un seul sous-état. L'agent doit donc apprendre l'association des renforcements élémentaires s_k^i et des sous-états à l'aide des seules informations dont il dispose. C'est le rôle de la fonction d'*attribution des récompenses*.

Par exemple, si la souris trouve un fromage et observe le vecteur de renforcement $S_k = (1, 0, 0, 0)$, le sous-état (fromage) qui a provoqué la récompense positive peut ne pas se trouver en première position dans le vecteur Y_{k-1}. La récompense positive est due au fromage qui est actuellement à la même position que la souris mais cette information est issue de *notre* connaissance du système. À son niveau, la souris doit apprendre la cause de cette récompense (*cf.* figure 4). Autre exemple, si la souris est sur une case encadrée de deux murs et si elle en percute un, dans notre labyrinthe discret ces deux murs sont l'un et l'autre à la même distance de la souris (qui n'a pas bougé à cause de la collision). La proximité n'est donc pas discriminante pour l'attribution de la récompense négative obtenue.

3.4. *Fonctionnement du Q-Learning parallèle*

Le Q-Learning parallèle est fondé sur le schéma classique de l'apprentissage par renforcement : à chaque instant k, l'algorithme utilise Y_k, S_k, Y_{k-1} et u_{k-1} pour mettre à jour la fonction de valeur. L'algorithme sélectionne ensuite l'action u_k a appliquer en fonction de critères d'exploitation et d'exploration et de l'estimation courante de l'espérance de gain globale, que l'on note $\mathsf{Q}(X_k, v)$.

Dans le cas du Q-Learning parallèle, il est donc nécessaire de calculer une estimation $\widehat{\mathsf{Q}}(X_k, v)$ de l'espérance de gain global $\mathsf{Q}(X_k, v)$ pour l'état global X_k et les actions $v \in \mathcal{U}$ à partir des seules informations disponibles, c'est-à-dire des valeurs $\mathsf{q}(x_k^i, v)$. Ainsi, une fonction dite de *fusion* va utiliser la connaissance de q pour estimer une espérance de gain adaptée à l'état global du système.

Pour répondre à la spécificité de l'observation des états et des renforcements dans le désordre, nous avons créé les fonctions déjà mentionnées de mise en correspondance temporelle et d'attribution des récompenses.

Le fonctionnement général de cette architecture est synthétisé par l'algorithme de la figure 5.

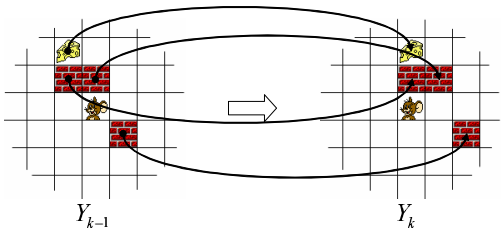
3.4.1. *Mémorisation de la fonction de valeur*

La mémoire stocke une estimation de la valeur de chaque paire *sous-état–action*. Elle associe à chaque couple *sous-état–action* une valeur réelle, soit : $\mathsf{q} : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$.

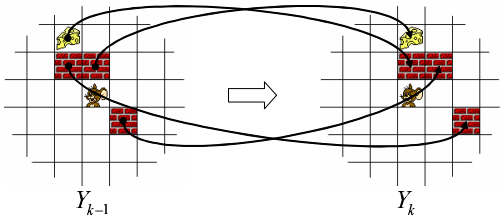
La fonction q est implantée par un tableau.

Entrée : Y_{k-1}, u_{k-1}, Y_k, S_k
Sortie : u_k
Début
  C_k ←MiseEnCorrespondance(Y_{k-1}, Y_k)
  A_k ←Attribution(C_k, S_k, u_{k-1}, q)
  MiseAJour(A_k, u_{k-1}, q)
  $\widehat{\mathsf{Q}}_k$ ←Fusion(Y_k, q)
  u_k ←Décision($\widehat{\mathsf{Q}}_k$) *(ε-glouton)*
Fin

Figure 5. *Algorithme d'une itération de Q-Learning parallèle*



(a) Mise en correspondance exacte



(b) Mise en correspondance erronée

Figure 6. *Exemples de mises en correspondance*

3.4.2. *Fonction de mise en correspondance temporelle*

Le rôle de la mise en correspondance est d'apparier les sous-états de l'observation Y_{k-1} avec ceux de l'observation Y_k quand l'ordre des sous-états n'est pas le même entre deux instants. Pratiquement, la mise en correspondance doit reconstituer l'évolution de chaque sous-état. Par exemple, dans le labyrinthe, elle doit reconstituer le déplacement relatif de chaque objet par rapport à la souris entre deux instants k (*cf.* figure 6). La mise en correspondance est dite *exacte* si elle correspond à la réalité de l'évolution des sous-états.

Entrée : Y_{k-1}, Y_k
Sortie : C_k
Début
  $C_k \leftarrow \emptyset$
  Si $\operatorname{card} Y_{k-1} < \operatorname{card} Y_k$ **alors**
    Tant que $Y_{k-1} \neq \emptyset$ **faire**
      Retirer au hasard un sous-état y de Y_{k-1}
      $y' \leftarrow \underset{z \in Y_k}{\arg\min} \|y - z\|$ *(les ex æquo sont départagés par tirage au sort)*
      Retirer y' de Y_k
      $C_k \leftarrow C_k + \{(y, y')\}$ *(concaténation)*
    Fin tant que
  Sinon
    Tant que $Y_k \neq \emptyset$ **faire**
      Retirer au hasard un sous-état y' de Y_k
      $y \leftarrow \underset{z \in Y_{k-1}}{\arg\min} \|y' - z\|$ *(les ex æquo sont départagés par tirage au sort)*
      Retirer y de Y_{k-1}
      $C_k \leftarrow C_k + \{(y, y')\}$ *(concaténation)*
    Fin tant que
  Fin si
Fin

Figure 7. *Algorithme de la fonction de mise en correspondance temporelle*

La sortie de cette fonction est un ensemble C_k de couples c_k^i. Chaque couple c_k^i est constitué d'un sous-état y_{k-1}^j de Y_{k-1} et de son successeur supposé y_k^l appartenant à Y_k tel que $c_k^i = (y_{k-1}^j, y_k^l)$.

Afin d'obtenir une solution en un faible temps de calcul, nous avons opté pour une méthode heuristique très simple : il s'agit de prendre au hasard un sous-état de l'observation Y_k et de l'associer au sous-état le plus proche (en terme de distance) appartenant à Y_{k-1}, puis de recommencer jusqu'à avoir apparié tous les sous-états[11]. La figure 7 donne le détail de cette méthode.

Cette heuristique implique la définition d'une métrique sur $\mathcal{X}$. Cette définition doit être cohérente avec le fonctionnement du système. Par exemple, la distance entre un sous-état « fromage » et un sous-état « mur » doit être beaucoup plus grande que la distance entre deux sous-états de même type dans la même configuration spatiale. Dans le cas contraire, le changement de type serait aussi facile que le déplacement d'un objet, ce qui n'est pas réaliste.

Dans l'exemple du labyrinthe, si m^i, n^i et t^i sont les coordonnées et le type d'un sous-état y^i et m^j, n^j et t^j ceux d'un sous-état y^j, nous utilisons une distance de la forme :

$$\|y^i - y^j\| = |m^i - m^j| + |n^i - n^j| + K|t^i - t^j| \qquad [14]$$

11. Dans l'exemple du labyrinthe, si un nouvel objet apparaît ou disparaît (prise d'un fromage par exemple), Y_{k-1} et Y_k n'auront pas le même cardinal ($n_{k-1} \neq n_k$). Dans ce cas, les sous-états qui n'ont pas été associés sont ignorés par la mise en correspondance pendant une itération.

Entrée : C_k, S_k, u_{k-1}
Entrée-sortie : q
Sortie : A_k
Début
  $A_k \leftarrow \emptyset$
  Tant que $S_k \neq \emptyset$ **faire**
    $r \leftarrow \min S_k$
    retirer r de S_k
    $(y, y') \leftarrow \underset{(y,y') \in C_k}{\arg\min} \left(\mathsf{q}(y, u_{k-1}) - \gamma \max_{v \in \mathcal{U}} \mathsf{q}(y', v) \right)$ *(les ex æquo sont départagés par tirage au sort)*
    retirer (y, y') de C_k
    $A_k \leftarrow A_k + \{(y, y', r)\}$ *(concaténation)*
  Fin tant que
Fin

Figure 8. *Algorithme de la fonction d'attribution des récompenses*

avec K un réel positif suffisamment grand pour rendre difficile le changement de type.

Cette méthode heuristique n'est pas exacte et commet parfois quelques erreurs d'appariement, notamment quand les objets sont regroupés. Pour d'autres systèmes dont la dynamique serait plus complexe, il est évidemment possible d'améliorer cette fonction en utilisant une méthode plus performante ou un critère plus pertinent[12]. L'influence des erreurs de mise en correspondance sur la convergence est discutée plus loin.

3.4.3. *Fonction d'attribution des récompenses*

Le rôle de la fonction d'attribution des récompenses est d'apparier chaque couple c_k^j de C_k avec une valeur scalaire s_k^f de S_k. Cette fonction doit en effet retrouver les sous-états qui sont en rapport avec l'obtention de chaque valeur de renforcement.

Le résultat de cette fonction est un ensemble A_k de triplets a_k^i. Ces triplets a_k^i associent les deux membres d'un couple de sous-états $c_k^j = (y_{k-1}^l, y_k^h)$ à la récompense s_k^f attribuée à ce couple, tels que $a_k^i = (y_{k-1}^l, y_k^h, s_k^f)$.

Soit a la fonction d'attribution, on a :

$$A_k = \mathsf{a}(C_k, S_k) \qquad [15]$$

12. La littérature est très riche dans le domaine de la mise en correspondance.

La méthode d'attribution que nous avons développée utilise la seule information disponible : la fonction de valeur q. D'après les équations de Bellman (Sutton *et al.*, 1998), on a :

$$\forall (x,u) \in \mathcal{X} \times \mathcal{U} \quad \mathsf{q}^*(x,u) = \sum_{x' \in \mathcal{X}} \mathsf{t}(x,u,x')[\mathsf{r}(x,u,x') + \gamma \max_{v \in \mathcal{U}} \mathsf{q}^*(x',v)] \qquad [16]$$

Si le système est déterministe et si l'on pose $x_{k-1} = x$, $x_k = x'$ et $r_k = \mathsf{r}(x,u,x')$, alors pour l'action u effectuée par l'agent, on a :

$$P_r(x'|x,u) = 1 \quad \text{et} \quad P_r(s|x,u) = 0 \quad \text{si } s \neq x' \qquad [17]$$

Et donc :

$$\mathsf{q}^*(x,u) = \mathsf{r}(x,u,x') + \gamma \max_{v \in \mathcal{U}} \mathsf{q}^*(x',v) \qquad [18]$$

On obtient finalement l'expression suivante qui donne la récompense immédiate associée à la dernière transition effectuée en fonction de q^* :

$$\mathsf{r}(x,u,x') = \mathsf{q}^*(x,u) - \gamma \max_{v \in \mathcal{U}} \mathsf{q}^*(x',v) \qquad [19]$$

Or, pendant la phase d'apprentissage, l'algorithme optimise la fonction de valeur courante q. Notre idée est d'utiliser les estimations $\mathsf{q}(x,u)$ et $\mathsf{q}(x',v)$ pour calculer une estimation $\widehat{\mathsf{r}}$ de $\mathsf{r}(x,u,x')$:

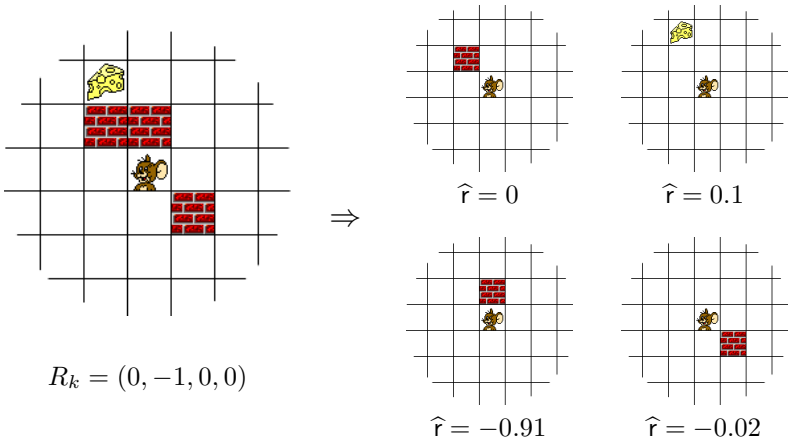
$$\widehat{\mathsf{r}}(x,u,x') = \mathsf{q}(x,u) - \gamma \max_{v \in \mathcal{U}} \mathsf{q}(x',v) \qquad [20]$$

D'une manière générale, que le système soit déterministe ou pas, plus la fonction de valeur q sera proche de l'optimum, meilleure sera cette estimation.

A partir de cette constatation, nous avons développé un algorithme d'attribution des récompenses fondé sur la fonction de valeur. Il consiste à trouver une association entre récompenses et couples de sous-états qui minimise la somme des écarts entre les valeurs des récompenses obtenues et les estimations de récompenses des couples. On cherche donc une fonction a d'attribution des récompenses telle que, pour toute autre fonction d'attribution h, on ait :

$$\sum_{\forall (y,y',r) \in \mathsf{a}(C_k,S_k)} |\widehat{\mathsf{r}}(y,u_{k-1},y') - r| \leq \sum_{\forall (y,y',r) \in \mathsf{h}(C_k,S_k)} |\widehat{\mathsf{r}}(y,u_{k-1},y') - r| \qquad [21]$$

Par hypothèse, les valeurs de renforcement obtenues S_k sont aussi nombreuses que les couples de sous-états C_k. Pour trouver une fonction d'attribution optimale,



(a) Récompenses observées

(b) Estimations des récompenses

Figure 9. *Illustration du calcul des estimations de récompense après l'action « aller au nord »*

il suffit d'associer la plus petite des récompenses avec le couple de sous-états dont l'estimation de la récompense est la plus faible, d'enlever chacun de ces deux éléments des ensembles S_k et C_k correspondants, puis de recommencer le procédé tant que les ensembles ne sont pas vides. Il est trivial de démontrer que cette attribution minimise bien la somme des écarts (si on effectue une permutation, la nouvelle somme des écarts est forcément supérieure ou égale à l'ancienne). L'algorithme de cette fonction est donné par la figure 8.

Dans l'exemple du labyrinthe, si l'agent décide de monter dans la situation de la figure 9, il ne va pas bouger et le système va rester dans le même état. L'ensemble des récompenses observé est alors $(0, -1, 0, 0)$. L'agent calcule les récompenses estimées des couples de sous-états et cherche l'attribution des récompenses qui minimise la somme des écarts. Ainsi, le couple dont la récompense estimée est la plus faible reçoit la récompense -1, les autres couples recevant 0. De même, si la souris atteint un fromage et observe $R = (0, 0, 1, 0)$, la récompense 1 est attribuée au couple dont la récompense estimée est la plus grande, les autres couples recevant 0.

3.4.4. *Fonction de mise à jour*

Une fois la mise en correspondance et l'attribution des récompenses effectuées, la fonction de mise à jour peut utiliser les informations contenues dans l'ensemble de

Entrée : A_k, u_{k-1}
Entrée-sortie : q
Début
  Pour tout (y, y', r) de A_k **faire**
    $\mathsf{q}(y, u_{k-1}) \leftarrow \mathsf{q}(y, u_{k-1}) + \alpha(r + \gamma \max_{v \in \mathcal{U}} \mathsf{q}(y', v) - \mathsf{q}(y, u_{k-1}))$
  Fin pour
Fin

Figure 10. *Algorithme de la fonction de mise à jour*

triplets A_k pour améliorer la fonction de valeur courante q. Pour cela, elle utilise la formule de mise à jour du Q-Learning avec chaque triplet (y, y', r) de A_k, soit :

$$\mathsf{q}(y, u_{k-1}) \leftarrow \mathsf{q}(y, u_{k-1}) + \alpha \left(r + \gamma \max_{v \in \mathcal{U}} \mathsf{q}(y', v) - \mathsf{q}(y, u_{k-1}) \right) \qquad [22]$$

avec α un coefficient dit d'apprentissage.

La figure 10 présente l'algorithme de cette fonction.

3.4.5. *Fonction de fusion*

Le rôle de la fonction de fusion est de calculer une estimation $\widehat{\mathsf{Q}}(X, u)$ de la fonction de valeur *globale* $\mathsf{Q}(X, u)$ de chaque action u et état X du système que nous avons choisi de définir par :

$$\mathsf{Q}^{\pi}(X, u) = E \left\{ \sum_{i=0}^{\infty} \gamma^i \sum_{j=1}^{n} \underline{r}_{i+k+1}^{j} \mid \underline{X}_k = X, \underline{u}_k = u, \pi \right\} \quad 0 \leq \gamma < 1 \qquad [23]$$

Ce choix rejoint la définition conventionnelle du gain adopté dans le cadre des PDMF (*cf.* équation 4).

Comme la mémoire stocke l'espérance de gain des actions par rapport aux sous-états, l'agent dispose de l'estimation de l'espérance de gain $\mathsf{q}(y_k^i, u)$ de chaque action u quand il observe le sous-état y_k^i. A partir de ces valeurs, il faut trouver un moyen de calculer une estimation de l'espérance de gain global $\mathsf{Q}(X_k, u)$.

Nous avons choisi de présenter dans cet article les résultats obtenus avec une méthode de fusion simple. Cette méthode correspond au principe de maximisation de l'espérance collective du W-Learning (Humphrys, 1995; Humphrys, 1996; Humphrys, 1997) et calcule une estimation $\widehat{\mathsf{Q}}(X_k, u)$ de l'espérance de gain global, telle que :

$$\widehat{\mathsf{Q}}_k(X_k, u) = \sum_{j=1}^{n} \mathsf{q}(x_k^j, u) \qquad [24]$$

Comme l'ordre des sous-états n'a pas d'importance dans la sommation, on a aussi :

$$\widehat{\mathsf{Q}}_k(X_k, u) = \widehat{\mathsf{Q}}_k(Y_k, u) = \sum_{j=1}^{n} \mathsf{q}(y_k^j, u) \qquad [25]$$

Ce choix est discuté dans la dernière section.

3.4.6. *Fonction de décision*

La fonction de décision se fonde sur les valeurs globales $\widehat{\mathsf{Q}}(Y_k, u)$ de chaque action u de $\mathcal{U}$ afin de déterminer l'action à effectuer.

Les stratégies de décision classiques s'adaptent parfaitement à l'approche parallèle. Nous utilisons dans la suite l'ϵ-glouton[13] car il permet d'évaluer facilement l'impact de l'exploration sur les performances. Toute autre stratégie pourrait aussi bien être utilisée. ϵ est fixé à $0,1$.

3.4.7. *Convergence de la fonction de valeur* q

L'exploration de l'espace des sous-états est garantie par la stratégie de décision de l'ϵ-glouton. Si l'évolution des sous-états est réellement indépendante et si les étapes de mise en correspondance et d'attribution ne font pas d'erreur, alors le problème revient à une optimisation classique de q pour un PDM simple qui serait défini par t et r sur $\mathcal{X}$. Sous ces conditions, la fonction de valeur q convergera donc vers la fonction de valeur optimale.

Malheureusement, ces hypothèses ne sont pas toujours vérifiées. La fonction de mise en correspondance commet parfois des erreurs. Si une erreur survient, un sous-état sera momentanément permuté par erreur avec un autre. Du point de vue de l'agent, l'évolution de ce sous-état sera donc anormale. Néanmoins, ces erreurs ne dépendent que de la configuration spatiale des sous-états et d'un tirage aléatoire (*cf.* algorithme de la figure 7). Si la fréquence des erreurs de mise en correspondance est faible, ces erreurs se traduiront au point de vue de l'agent par un comportement légèrement stochastique des sous-états, ce qui ne nuit pas à la convergence de la fonction de valeur. Dans le cas des bancs d'essais présentés dans l'article, les erreurs de mise en correspondance sont rares. Dans d'autres systèmes, il pourrait s'avérer nécessaire d'utiliser une fonction de mise en correspondance plus élaborée.

En revanche, la convergence dépend fortement de l'attribution correcte des récompenses. La fonction d'attribution des récompenses dépend de l'estimation q qui elle-même varie dans le temps, la politique d'attribution des récompenses est non stationnaire. Il n'est donc pas établi que cette méthode associée à l'apprentissage de la fonction de valeur q engendre une attribution sans erreur et permette ainsi à la fonction

13. Avec une probabilité ϵ, l'action est choisie au hasard indépendamment des valeurs de $\widehat{\mathsf{Q}}(Y_k, u)$, sinon, l'action sélectionnée est celle pour laquelle $\widehat{\mathsf{Q}}(Y_k, u)$ est maximale pour Y_k fixé.

de valeur q de converger. La stabilité de ce principe d'attribution des récompenses est discutée du point de vue expérimental dans la section suivante.

4. Tests expérimentaux sur le labyrinthe

4.1. *Labyrinthe de test*

Nous avons choisi un labyrinthe de 400 cases (20×20) sur lequel sont disposés au hasard des fromages et des murs dispersés (*cf.* figure 11). Afin de limiter les effets de bord, les objets sont toujours disposés à plus de 5 cases du bord du labyrinthe.

Si à l'instant k, la souris atteint une case pourvue d'un fromage, une récompense égale à 1 est ajoutée à l'ensemble S_k de récompenses. A l'instant $k+1$, le fromage a disparu et est replacé à l'instant $k+2$ au hasard sur une case libre du labyrinthe.

Si à l'instant k, la souris tente de passer sur un mur, elle reste sur sa case et une récompense égale à -1 est ajoutée à l'ensemble S_k de récompenses. Les murs sont fixes.

Au début de chaque apprentissage, la fonction $\mathsf{q}(x,u)$ est initialisée à 0 pour tout sous-état x et toute action u.

4.2. *Évolution du gain moyen*

Les courbes de la figure 11 sont les moyennes sur 100 simulations des évolutions du gain moyen en fonction du nombre d'itérations. Le gain moyen $\mathsf{G}_m(t)$ est défini par :

$$\mathsf{G}_m(t) = \frac{1}{T}\sum_{k=0}^{T-1} \mathsf{G}(t+k) \qquad [26]$$

avec T un petit nombre d'itérations (typiquement $T = 200$) et G le gain défini par l'équation :

$$\mathsf{G}(t) = \sum_{i=0}^{\infty} \gamma^i \sum_{j=1}^{n} r_{i+t+1}^j \qquad [27]$$

Par rapport à l'équation 23 qui correspond à l'espérance mathématique de la somme pondérée des renforcements futurs pour tout un *ensemble* de trajectoires possibles, G correspond à la somme pondérée des renforcements de *la* trajectoire suivie par le système pendant la simulation.

L'utilisation du gain moyen permet de lisser le tracé de la courbe.

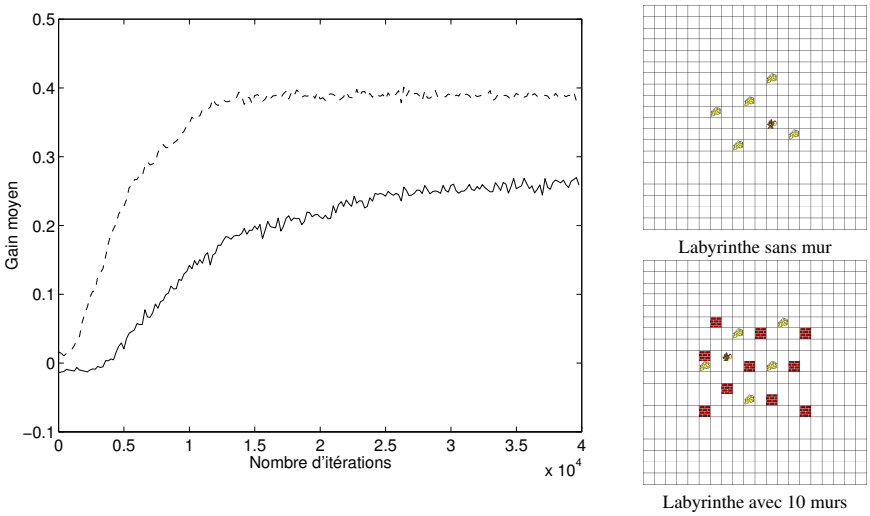


Figure 11. *Comparaison de l'évolution du gain moyen avec un labyrinthe contenant 10 murs dispersés (trait continu) et avec un labyrinthe sans mur (trait pointillé) (moyenne sur 100 simulations avec 5 fromages,* $\alpha = 1$, $\gamma = 0,5$ *et* $\epsilon = 0,1$*).*

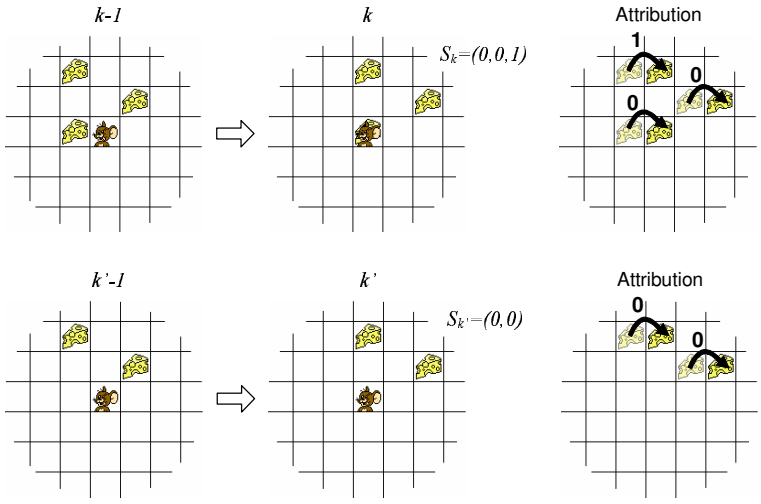
4.3. *Analyse des résultats*

Dans le cas du labyrinthe sans mur, les observations directes du mouvement de la souris montrent qu'elle ne fait pas de détour entre les fromages, que les trajectoires obtenues et donc la politique sont de bonne qualité. Étant donné la grande dimension de l'espace d'état, il est difficile de calculer une politique optimale. Il est donc difficile de dire si la politique obtenue est vraiment optimale.

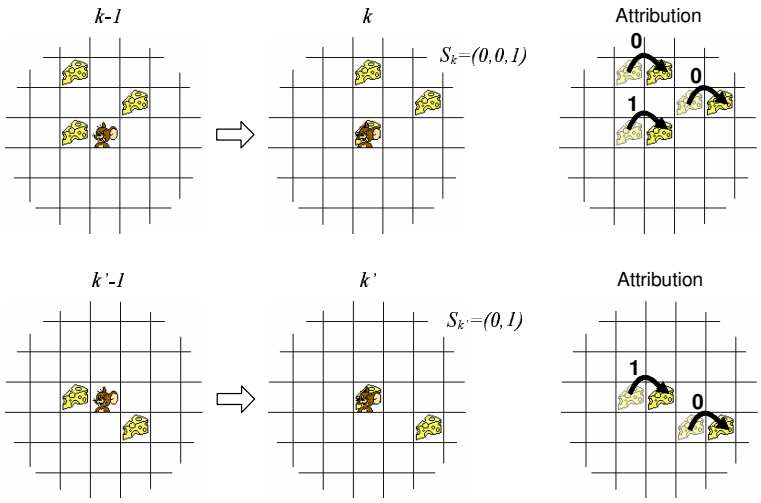
Dans le cas du labyrinthe avec murs, le gain limite obtenu est sensiblement inférieur. Cette diminution ne s'explique pas seulement par le contournement des murs qui rallonge les distances entre les fromages. En effet, nous verrons plus loin que l'on peut obtenir de meilleurs gains. La politique obtenue avec des murs est donc sous-optimale.

Dans les deux situations, les courbes montrent que l'algorithme converge malgré les erreurs de mises en correspondance, d'attributions et malgré le léger couplage entre les sous-états. Le principe de l'attribution des récompenses fonctionne. Le succès de cette méthode s'explique par un processus d'*auto-vérification*, expliqué ci-dessous, lié au principe même de l'algorithme.

Au début de l'apprentissage, la fonction de valeur contient des valeurs identiques pour tous les couples *sous-état—action* (x, u) ($\mathsf{q}_0 = 0$). Aussi, lors de l'attribution, les récompenses sont distribuées au hasard aux couples de sous-états. Si la souris rencontre par hasard un fromage, l'ensemble des récompenses contiendra une récompense positive qui sera attribuée au hasard à un couple de sous-états. Cette attribution



(a) Premier scénario : à l'instant k, la souris trouve un fromage et attribue la récompense positive au sous-état de coordonnées $(-1, 2)$ (dans le repère lié à la souris). A l'instant $k' > k$, la souris se retrouve dans la même situation vis-à-vis d'un sous-état qui doit donner une récompense positive, mais les récompenses obtenues sont nulles. L'attribution précédente était donc erronée et la nouvelle attribution corrige cette erreur.



(b) Second scénario : à l'instant k, la souris trouve un fromage et attribue la récompense positive au sous-état de coordonnées $(-1, 0)$. A l'instant $k' > k$, la souris se retrouve dans la même situation vis-à-vis d'un sous-état qui doit donner une récompense positive. l'agent obtient alors une récompense positive. L'attribution précédente était donc correcte et l'agent attribue de nouveau la récompense positive au sous-état de coordonnées $(-1, 0)$.

Figure 12. *Illustration du processus d'auto-vérification de l'attribution des récompenses*

a donc peu de chances d'être correcte (*cf.* figure 12a). Comme la récompense est attractive, l'agent va essayer d'observer à nouveau le sous-état qui, selon lui, a provoqué une récompense positive. Après un certain nombre d'itérations, le système va donc se retrouver dans une situation présentant le sous-état qui, d'après l'agent, fournira une récompense positive en effectuant la même action que précédemment. Si l'attribution était erronée, alors les récompenses obtenues seront nulles, et le sous-état en question se verra attribuer une récompense nulle. En revanche, si l'attribution était correcte (*cf.* figure 12b), l'agent reçoit bien la récompense positive attendue et le processus attribuera à nouveau la récompense positive au même sous-état. Ce processus d'*auto-vérification* assure donc une certaine stabilité de la fonction d'attribution dans le cas des récompenses positives.

Dans le cas de l'attribution d'une récompense négative, ce mécanisme ne fonctionne pas. Si l'attribution d'une récompense négative est erronée, l'agent ne va pas tenter d'observer à nouveau le sous-état qui, selon lui, a provoqué une récompense négative puisqu'il cherche à maximiser son gain. Seule une action d'exploration permettra de corriger cette erreur d'attribution, ce qui prend beaucoup plus de temps. Ce problème de non vérification de l'attribution des récompenses négatives explique la diminution du gain lors de la présence de murs.

Pour pallier ce problème de diminution du gain et vérifier notre analyse, nous avons procédé avec « pédagogie » : au lieu d'exiger de l'algorithme d'apprendre tout en même temps, il est préférable de présenter d'abord un labyrinthe contenant uniquement des fromages, puis un labyrinthe contenant aussi des murs (*cf.* figure 13). L'apprentissage est plus rapide et la politique obtenue est de meilleure qualité. Avec cette méthode progressive, les récompenses positives sont d'abord attribuées correctement grâce au mécanisme d'auto-vérification, puis les récompenses négatives sont attribuées éventuellement avec des erreurs. Ces erreurs ont moins d'impact que précédemment car l'agent sait déjà comment obtenir les récompenses positives.

4.4. *Influence du nombre de fromages*

La figure 14 montre le gain limite et le temps d'apprentissage en fonction du nombre de fromages disposés dans le labyrinthe. La première constatation concerne le temps d'apprentissage : il décroît de façon importante lorsque le nombre de fromages augmente. En fait, la présence de plusieurs fromages permet d'expérimenter un plus grand nombre de sous-états à chaque itération, ainsi il met à jour la fonction de valeur plusieurs fois et pour des sous-états différents. *Le grand nombre d'objets devient donc un avantage pour l'agent.*

L'autre observation concerne le gain limite qui augmente avec le nombre de fromages, ce qui était prévisible. En effet, quand la quantité de fromages est importante, la souris a statistiquement moins de chemin à parcourir entre chaque fromage, ce qui augmente le gain.

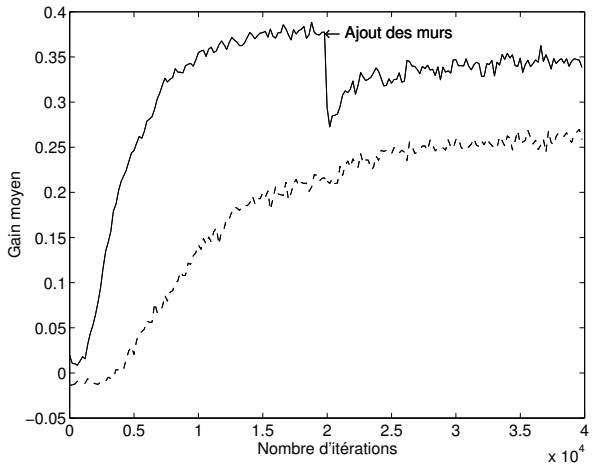


Figure 13. *Comparaison de l'apprentissage progressif (trait continu) et de l'apprentissage normal (trait pointillé) (moyenne sur 100 simulations avec 5 fromages, 10 murs,* $\alpha = 1$, $\gamma = 0,5$ *et* $\epsilon = 0,1$*).*

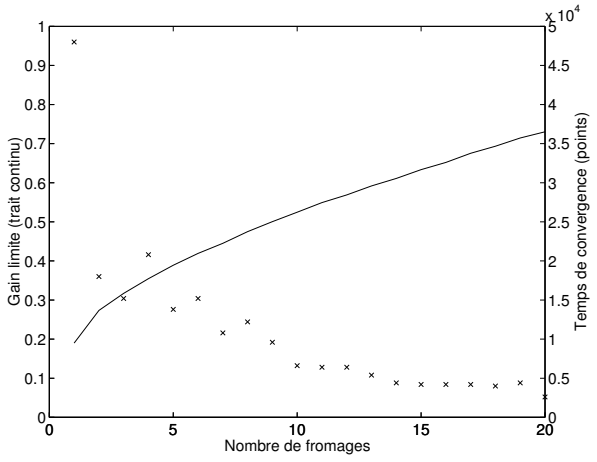
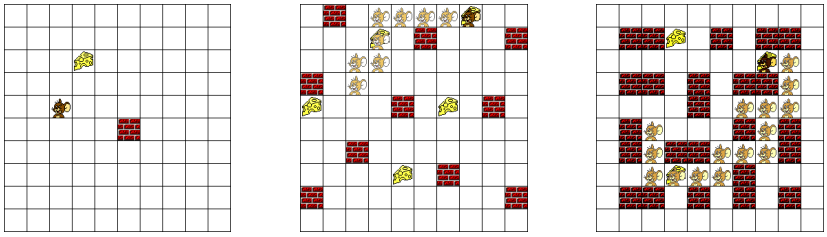


Figure 14. *Influence du nombre de fromages sur le gain limite (trait continu) et sur le temps de convergence (points). Le temps de convergence est exprimé en nombre d'itérations (moyenne sur 100 simulations avec* $\alpha = 1$, $\gamma = 0,5$ *et* $\epsilon = 0,1$*).*



(a) Apprentissage dans un premier labyrinthe

(b) Exemple de trajectoire obtenue dans un deuxième labyrinthe sans ré-apprentissage

(c) Exemple de trajectoire obtenue dans un troisième labyrinthe sans ré-apprentissage

Figure 15. *Illustration de la généricité de l'apprentissage*

4.5. *Illustration de la généricité de l'apprentissage*

Par construction, l'apprentissage ne dépend pas du labyrinthe choisi et s'avère générique. Pour illustrer cette généricité, nous avons changé plusieurs fois la souris de labyrinthe après un apprentissage dans un labyrinthe ayant un seul fromage et un seul mur. La figure 15 montre des exemples de trajectoires obtenues dans ces différents labyrinthes sans ré-apprentissage.

4.6. *Conclusion*

Ces résultats expérimentaux sur le labyrinthe montrent que le Q-Learning parallèle est stable. D'une part, le mécanisme d'attribution des récompenses fonctionne (particulièrement avec les récompenses positives). D'autre part, un grand nombre d'objets n'est pas un problème mais permet au contraire d'augmenter le nombre des expériences informatives pour l'algorithme, diminuant ainsi le nombre d'itérations nécessaires à l'apprentissage. L'apprentissage est bien indépendant de la configuration du système (position des fromages et des murs). Ainsi, face à un état inconnu, l'agent utilise ce qu'il a appris pour réagir immédiatement de manière adéquate.

5. Tests expérimentaux sur le New York Driving (NYD)

5.1. *Principe*

Le banc d'essai du New York Driving (NYD) a été initialement proposé par Mc Callum en 1995 (McCallum, 1995). Son principe de fonctionnement est décrit par la figure 16.

Cette simulation est destinée originellement au test d'algorithmes d'apprentissage dans le cadre des PDM partiellement observables et où certaines actions permettent de

choisir la partie observée de l'état du système (perception active). L'espace d'états de ce système est en effet trop grand pour être perçu entièrement par la plupart des algorithmes. Chaque véhicule peut prendre 17×4 positions différentes sur la route, deux vitesses différentes et 6 couleurs différentes. Il y a couramment plus d'une douzaine de véhicules sur la route simultanément.

Dans le test original, des actions particulières permettent de sélectionner la direction de la « vue » de l'agent. La perception du système décrit alors les caractéristiques du véhicule le plus proche. Ceci permet d'obtenir des observations locales de faibles dimensions mais avec des problèmes d'apprentissage dus à l'observation partielle du système.

Le Q-Learning parallèle permet au contraire de percevoir la totalité du système. Nous avons donc testé notre architecture dans un cas particulier que nous appelons New York Driving totalement observable dans le désordre. Les caractéristiques et le fonctionnement du simulateur sont parfaitement identiques au cas classique. En revanche, l'algorithme observe dans le désordre la totalité de l'état du système au lieu d'une observation partielle.

5.2. *Résultats expérimentaux*

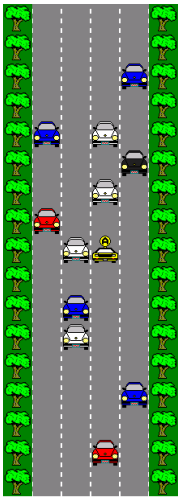
L'algorithme utilisé pour le test sur le NYD est exactement le même que celui précédemment utilisé pour le labyrinthe. Seule la définition des sous-états change. Un sous-état x du NYD correspond à un véhicule. On pose :

$$x = (i, j, v, c, d) \qquad [28]$$

avec : i la position verticale du véhicule (17 possibilités), j la position horizontale du véhicule (4 possibilités), v la vitesse du véhicule (2 possibilités), c la couleur du véhicule (6 possibilités), d la position horizontale de la voiture pilotée (4 possibilités). Il y a donc 3264 ($17 \times 4 \times 2 \times 6 \times 4$) sous-états possibles *par* véhicule. L'état X du NYD regroupe autant de sous-états qu'il y a de véhicules (nombre variable dans le temps, couramment avoisinant la douzaine).

La figure 17 présente les résultats obtenus avec le Q-Learning parallèle sur le NYD totalement observable dans le désordre. On observe une convergence rapide et un score final moyen de 0,007 par itération.

Une comparaison (figure 18) avec les autres algorithmes testés sur le NYD partiellement observable montre que ce score est particulièrement bon. Il ne faut pas conclure que l'approche parallèle est supérieure aux autres car l'information reçue par le Q-Learning parallèle est beaucoup plus riche. Il est donc normal de pouvoir obtenir une meilleure politique de décision. L'intérêt de l'approche parallèle est de pouvoir trouver une bonne politique malgré la taille extrêmement importante de l'espace d'états du système et dans le cas où l'état est observable dans le désordre.



Le test du New York Driving consiste à diriger une voiture (marquée d'un A ci-contre) entre des voitures plus rapides et plus lentes sur une route à quatre voies. La vitesse de la voiture pilotée est constante. Trois actions sont possibles : rester sur la même voie, aller sur la voie de droite ou aller sur la voie de gauche. Les autres voitures ne changent pas de voie et vont à une vitesse constante (soit à une vitesse rapide fixée soit à une vitesse lente aussi fixée). Tous les véhicules avancent de manière discrète (case par case).
A chaque pas, si la voiture pilotée gêne une voiture plus rapide, la récompense obtenue est -1. Si elle entre en collision avec une autre voiture, la récompense obtenue est -10 (mais la simulation continue). Enfin, dans les autres cas, la récompense obtenue est de $+0,1$.
Pour plus d'informations, le New York Driving est décrit dans la thèse de Mc Callum (McCallum, 1995) et une implantation en langage C est disponible sur sa page web.

Figure 16. *Principe de fonctionnement du New York Driving*

5.3. *Conclusion*

Ces tests sur le NYD montrent qu'il est possible de percevoir la totalité de l'état malgré sa grande dimension (en supposant que ces informations sont disponibles dans le désordre) et de trouver une meilleure politique de décision que dans le cas partiellement observable.

6. Discussions

6.1. *Discussion sur la mise en correspondance temporelle*

Nous avons utilisé dans cet article une méthode de mise en correspondance heuristique simple et efficace sur les deux problèmes expérimentés. Néanmoins, cette méthode a ses limites notamment pour des environnements compacts (sous-états très rapprochés) ou évoluant rapidement de manière continue.

L'intérêt du cadre théorique des PDMFOD et de l'architecture parallèle est de pouvoir changer la méthode de mise en correspondance indépendamment du reste de l'algorithme. Cette modularité permet d'incorporer aisément des méthodes plus avancées de mise en correspondance issues d'autres domaines scientifiques comme la vision artificielle ou la robotique mobile.

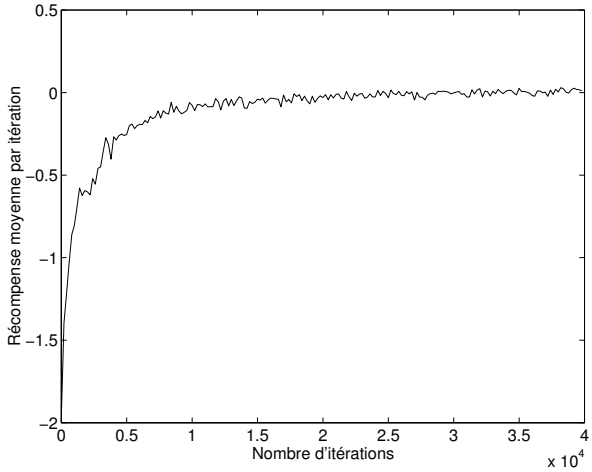


Figure 17. *Évolution de la récompense moyenne obtenue à chaque itération sur le NYD totalement observable (moyenne sur 20 simulations avec* $\alpha = 1$, $\gamma = 0,9$ *et* $\epsilon = 0$*)*

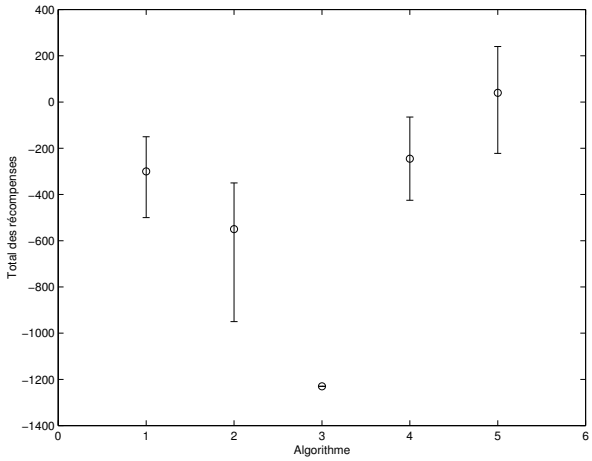


Figure 18. *Scores minimaux, maximaux et médians obtenus après apprentissage pour différents algorithmes sur le NYD (partiellement observable) pour 5000 itérations et sur un échantillon de 20 simulations par algorithme, (1) Q-Learning simple (Sallans, 2002), (2) Perceptron multi-couche local (Sallans, 2002), (3) U-Tree (McCallum, 1995), (4) Algorithme génétique stochastique (Glickman01 et al., 2001), (5) Q-Learning parallèle (sur le NYD totalement observable dans le désordre)*

6.2. *Discussion sur l'attribution des récompenses*

D'un point de vue expérimental, le mécanisme d'attribution des récompenses fonctionne bien notamment avec les récompenses positives. Malgré les erreurs que cette fonction peut commettre en début d'apprentissage, le Q-Learning parallèle est stable dans les deux simulations présentées.

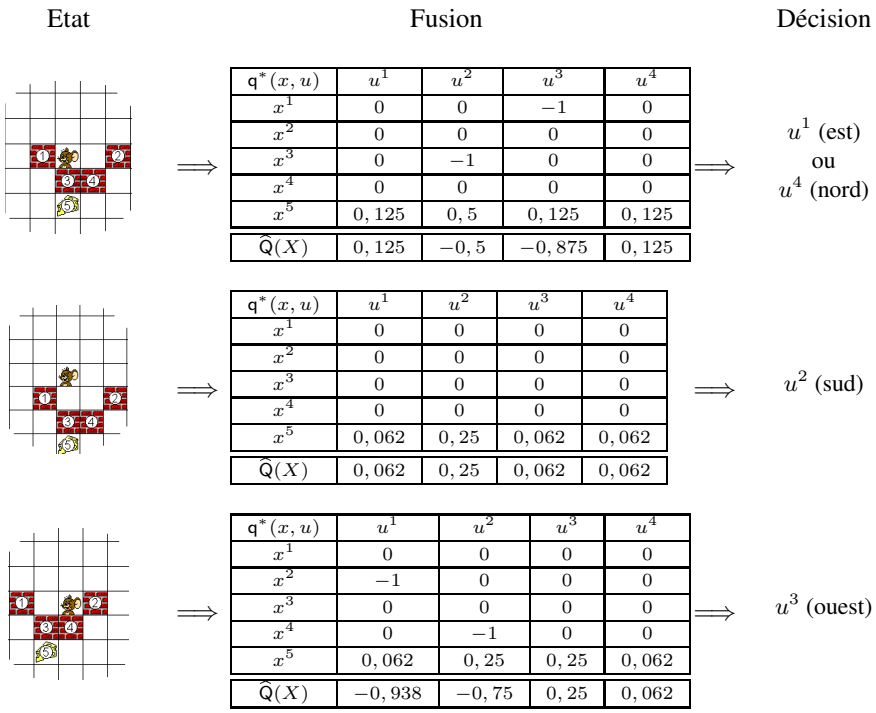
Ce résultat encourageant doit être nuancé. Tout d'abord, les deux simulations ne génèrent simultanément que rarement des valeurs de renforcements non nulles. Il serait donc intéressant de tester le mécanisme sur des systèmes générant simultanément de nombreuses récompenses non nulles. Ensuite, il faudrait tester l'attribution dans le cadre de systèmes non déterministes car il n'est pas certain que le mécanisme d'autovérification fonctionne toujours dans ce cas. Enfin, l'attribution des récompenses négatives n'est pas très performante et très lente car fondée sur le pur hasard, un mécanisme plus intelligent permettrait d'augmenter sensiblement la qualité des politiques obtenues et de se passer de l'apprentissage progressif.

6.3. *Discussion sur la méthode de fusion*

Dans les deux simulations, la fonction de fusion donne des résultats satisfaisants. En revanche, si l'on considère un labyrinthe plus complexe où plusieurs murs forment une impasse, la souris a tendance à y être piégée. Dans cette configuration, la fonction de fusion génère un maximum local pour la fonction de valeur globale, ce qui entraîne la formation d'un cycle entre quelques états (*cf.* figure 19).

Néanmoins, par rapport à des approches sans apprentissage, le Q-Learning parallèle est plus robuste car après un certain nombre d'essais infructueux, la souris finit par sortir de l'impasse. Les actions menant aux états du cycle deviennent en effet de moins en moins attractives puisque l'agent ne reçoit plus de récompense. Il finit donc par choisir d'autres actions plus prometteuses. La courbe de la figure 20 illustre ce phénomène. Elle montre clairement deux modes de fonctionnement : les instants où la souris se bloque dans l'impasse obtenant un gain négatif et les instants où la souris n'est pas bloquée et obtient un gain positif.

En réalité, la méthode de fusion (somme des $\mathsf{q}(y^i, u)$) ne permet pas d'obtenir l'espérance de gain globale exacte dans tous les cas. D'une manière générale, espérer trouver un opérateur qui fusionne à chaque instant k chaque $\mathsf{q}(y_k^i, u)$ pour estimer $\mathsf{Q}(Y_k, u)$ est plutôt illusoire car l'information contenue dans les $\mathsf{q}(y_k^i, u)$ n'est valable que pour chaque sous-état pris individuellement. Elle n'est donc que peu corrélée à l'état global du système. Pour résoudre ce problème, il est nécessaire de définir différemment l'espérance de gain global que l'on va chercher à maximiser et de développer, en accord avec cette définition, une approche qui permet de converger vers Q^*. Dans ce sens, nous avons développé une approche qui nécessite d'utiliser un algorithme indirect (Dyna-Q parallèle) suivi d'une étape de planification. Cette méthode sera l'objet d'un prochain article.



$\mathsf{q}^*(x,u)$	u^1	u^2	u^3	u^4
x^1	0	0	-1	0
x^2	0	0	0	0
x^3	0	-1	0	0
x^4	0	0	0	0
x^5	$0,125$	$0,5$	$0,125$	$0,125$
$\widetilde{\mathsf{Q}}(X)$	$0,125$	$-0,5$	$-0,875$	$0,125$

$\mathsf{q}^*(x,u)$	u^1	u^2	u^3	u^4
x^1	0	0	0	0
x^2	0	0	0	0
x^3	0	0	0	0
x^4	0	0	0	0
x^5	$0,062$	$0,25$	$0,062$	$0,062$
$\widetilde{\mathsf{Q}}(X)$	$0,062$	$0,25$	$0,062$	$0,062$

$\mathsf{q}^*(x,u)$	u^1	u^2	u^3	u^4
x^1	0	0	0	0
x^2	-1	0	0	0
x^3	0	0	0	0
x^4	0	-1	0	0
x^5	$0,062$	$0,25$	$0,25$	$0,062$
$\widetilde{\mathsf{Q}}(X)$	$-0,938$	$-0,75$	$0,25$	$0,062$

Figure 19. *Exemple de formation d'un cycle entre trois états (estimations des valeurs de* q *réalisées avec* $\gamma = 0,5$*)*

7. Conclusions et perspectives

Nous avons proposé dans cet article un cadre formel original, les PDMFOD, permettant d'intégrer aux PDM factorisés les problématiques de mise en correspondance des données capteurs et des renforcements, problématiques classiques en robotique mobile.

Nous avons ensuite décrit une nouvelle architecture, le Q-Learning parallèle, capable d'effectuer un apprentissage par renforcement dans ce cadre théorique particulier. Cette architecture suppose une évolution indépendante des sous-états du PDMF sous-jacents au PDMFOD. L'indépendance des sous-états est un présupposé très fort mais il permet de montrer, à travers le Q-Learning parallèle, que l'apprentissage par renforcement est possible dans le cadre des PDMFOD.

Au point vue expérimental, le Q-Learning parallèle a été appliqué à deux bancs d'essais déterministes mais dont les espaces d'états sont de très grandes dimensions. Ces tests montrent que l'algorithme converge malgré l'observabilité dans le désordre du PDMF sous-jacent. L'apprentissage est même plutôt rapide car l'architecture tire

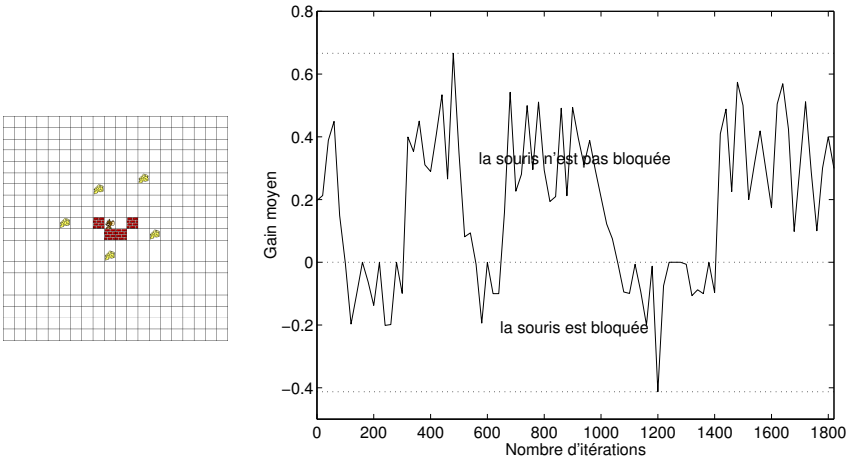


Figure 20. *Évolution du gain dans un labyrinthe avec 5 fromages et une impasse de 4 murs au centre du labyrinthe après apprentissage sur un labyrinthe aux murs dispersés (avec* $\alpha = 1$, $\gamma = 0,5$ *et* $\epsilon = 0,1$*)*

parti de la structure du vecteur d'état pour effectuer plusieurs mises à jour de la fonction de valeur à chaque itération. De plus, l'apprentissage réalisé est générique puisqu'il ne dépend pas d'une configuration spatiale particulière des sous-états et s'adapte à n'importe quelle nouvelle situation. Enfin, les politiques d'actions obtenues sont de très bonne qualité sur le New York Driving mais peuvent être largement sous-optimales dans le cadre de labyrinthes trop complexes en raison de la méthode de fusion employée.

Au point de vue applicatif, les PDMFOD définissent un cadre pour le contrôle par apprentissage d'un robot mobile confronté à un environnement à multiples objets et multiples objectifs. Nous pensons notamment au contrôle de robots évoluant dans un milieu dynamique : robots footballeurs, évolution d'un robot au milieu d'une foule, conduite d'un véhicule dans le trafic, robot domestique (Humphrys, 1997), *etc.* Dans ces exemples, il est courant que deux objets interagissent entre eux (deux robots footballeurs s'évitent, la balle rebondit sur un mur, *etc.*). Une prochaine étape pourrait donc être la détection automatique des couplages binaires entre sous-états tout en conservant la contrainte d'observabilité dans le désordre. Cette détection suffirait déjà à appliquer l'architecture à des environnements beaucoup plus complexes.

8. Bibliographie

Albus J. S., « A New Approach to Manipulator Control : The Cerebellar Model Articulator Controller (CMAC) », *Trans. of the ASME, J. Dynamic Systems, Measurement, and Control*, vol. 97, n° 3, p. 220-227, 1975.

Bellman R., *Dynamic Programming*, Princeton University Press, Princeton, NJ, 1957.

Bertsekas D. P., Tsitsiklis J. N., *Neural Dynamic Programming*, Athena Scientific, Belmont, MA, 1996.

Boutilier C., Dean T., Hanks S., « Decision-Theoretic Planning : Structural Assumptions and Computational Leverage », *Journal of Artificial Intelligence Research*, vol. 11, p. 1-94, 1999.

Boutilier C., Dearden R., Goldszmidt M., « Stochastic dynamic programming with factored representations », *Artificial Intelligence*, vol. 121, n° 1–2, p. 49-107, 2000.

Carreras M., A Proposal of a Behavior-based Control Architecture with Reinforcement Learning for an Autonomous Underwater Robot, PhD thesis, University of Girona, Spain, 2003.

Coulom R., Apprentissage par renforcement utilisant des réseaux de neurones, avec des applications au contrôle moteur, Thèse de doctorat, Institut National Polytechnique de Grenoble, 2002.

Crites R. H., Barto A. G., « Improving elevator performance using reinforcement learning », *Proc. of Advances in Neural Information Processing Systems*, The MIT Press, Cambridge, MA, 1996.

Dayan P., Hinton G. E., « Feudal reinforcement learning », *in* S. J. Hanson, J. D. Cowan, C. L. Giles (eds), *Proc. of Advances in Neural Information Processing Systems*, Morgan Kaufmann, San Mateo, CA, 1993.

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

Faihe Y., Hierarchical Problem Solving using Reinforcement Learning : Methodology and Methods, PhD thesis, University of Neuchâtel, Departement of Computer Science, Switzerland, 1999.

Glickman M., Sycara K., « Evolutionary search, stochastic policies with memory, and reinforcement learning with hidden state », *Proc. of the International Conference on Machine Learning*, 2001.

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

Humphrys M., W–learning : Competition among selfish Q–learners, Technical Report n° 362, University of Cambridge, 1995.

Humphrys M., « Action Selection methods using Reinforcement Learning », *Proc. of the International Conference on the Simulation of Adaptive Behavior : From Animals to Animats*, Massachusetts, USA, p. 135-44, 1996.

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

Kaelbling L. P., « Hierarchical learning in stochastic domains : Preliminary results », *Proc. of the International Conference on Machine Learning*, Morgan Kaufmann, Amherst, MA, 1993.

Kearns M., Koller D., « Efficient Reinforcement Learning in Factored MDPs », *Proc. of the International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, p. 740-747, August, 1999.

Kearns M., Singh S., « Near-Optimal Reinforcement Learning in Polynomial Time », *Proc. of the International Conference on Machine Learning*, p. 260-268, 1998.

Laurent G., Synthèse de comportements par apprentissages par renforcement parallèles : application à la commande d'un micromanipulateur plan, Thèse de doctorat, Université de Franche-Comté, Besançon, France, 2003.

Laurent G. J., Piat E., « Learning Mixed Behaviours with Parallel Q-learning », *Proc. of the IEEE International Conference on Intelligent Robots and Systems*, vol. 1, Lausanne, Switzerland, p. 1002-1007, September 30 – October 4, 2002.

Long-Ji L., « Hierarchical learning of robot skills by reinforcement », *Proc. of the International Conference on Neural Networks*, 1993.

Mahadevan S., « Machine learning for robots : A comparison of different paradigms », *Workshop on Towards Real Autonomy, IEEE/RSJ International Conference on Intelligent Robots and Systems*, Osaka, Japan, 1996.

Mahadevan S., Connell J., « Automatic programming of behavior-based robots using reinforcement learning », *Artificial Intelligence*, vol. 55, n° 2–3, p. 311-365, 1992.

McCallum A. K., Reinforcement Learning with Selective Perception and Hidden State, PhD thesis, Department of Computer Science, University of Rochester, New York, December, 1995.

Sallans B., Reinforcement Learning for Factored Markov Decision Processes, PhD thesis, Department of Computer Science, University of Toronto, 2002.

Singh S. P., « Transfer of learning by composing solutions of elemental sequential tasks », *Machine Learning*, vol. 8, n° 3, p. 323-340, 1992.

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

Tesauro G., « Practical issues in temporal difference learning », *Machine Learning*, vol. 8, p. 257-277, 1992.

Tham C. K., Prager R. W., « A modular Q-learning architecture for manipulator task decomposition », *Proc. of the International Conference on Machine Learning*, Morgan Kaufmann, San Francisco, CA, 1994.

Wiering M., Schmidhuber J., « HQ–Learning », *Adaptive Behavior*, 1997.