# Synthèse de comportements par apprentissages par renforcement parallèles : application à la commande d'un micromanipulateur plan

Guillaume J. Laurent


## ▶ To cite this version:

Guillaume J. Laurent. Synthèse de comportements par apprentissages par renforcement parallèles : application à la commande d'un micromanipulateur plan. Automatique / Robotique. Université de Franche-Comté, 2002. Français. NNT : . tel-00008761

## HAL Id: tel-00008761
## https://theses.hal.science/tel-00008761

Submitted on 14 Mar 2005

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N°950

Année 2002

# THÈSE

*préparée au*
**Laboratoire d'Automatique de Besançon (UMR CNRS 6596)**

*présentée à*
**L'U.F.R. DES SCIENCES ET TECHNIQUES DE L'UNIVERSITÉ DE FRANCHE-COMTÉ**

*pour obtenir le*
**GRADE DE DOCTEUR D'UNIVERSITÉ**
*spécialité* **Automatique et Informatique**

## Synthèse de comportements par apprentissages par renforcement parallèles

*Application à la commande d'un micromanipulateur plan*

*par*
**Guillaume Laurent**
(Ingénieur ENSMM, DEA *spécialité* Informatique, Automatique et Productique)

**Soutenue le 18 décembre 2002 devant la commission d'examen :**

| | |
|---|---|
| Rapporteurs | **Raja Chatila** (Directeur de recherche au LAAS–CNRS, Toulouse III) |
| | **Philippe Gaussier** (Professeur à l'Université de Cergy-Pontoise) |
| Examinateurs | **François Charpillet** (Directeur de recherche au LORIA–INRIA, Nancy I) |
| (Président de jury) | **Jean-Arcady Meyer** (Directeur de recherche à l'AnimatLab–CNRS, Paris VI) |
| Directeurs de thèse | **Nadine Lefort-Piat** (Professeur à l'ENSMM, Besançon) |
| | **Emmanuel Piat** (Maître de conférences à l'ENSMM, Besançon) |

*The key observation is that the world is its own best model. It is always exactly up to date. It always contains every detail there is to be known. The trick is to sense it appropriately and often enough.*

Rodney Brooks (1990)

# Remerciements

En tout premier lieu, je tiens à exprimer ma profonde reconnaissance à Nadine Lefort-Piat, Professeur à l'ENSMM, et à Emmanuel Piat, Maître de Conférences à l'ENSMM, qui ont inspiré et dirigé ce travail. J'ai particulièrement apprécié tout au long de ces années les échanges autant scientifiques qu'amicaux qui ont caractérisé notre collaboration.

Je tiens à remercier ceux qui m'ont fait l'honneur d'accepter de juger ce travail. Je remercie les rapporteurs, Raja Chatila, Directeur de recherche au LAAS, et Philippe Gaussier, Professeur à l'Université de Cergy-Pontoise, pour l'intérêt qu'ils ont porté à mon travail ainsi que pour leurs critiques constructives. Je remercie les examinateurs, Jean-Arcady Meyer, Président du jury, Directeur de recherche à l'AnimatLab et François Charpillet, Directeur de recherche au LORIA. J'aurais beaucoup aimé répondre de vive voix aux questions de Philippe Gaussier et de François Charpillet mais le verglas et le sort en ont décidé autrement...

Je voudrais également remercier les personnes qui m'ont offert l'opportunité d'effectuer cette thèse dans les meilleures conditions : Alain Bourjault, Directeur du Laboratoire d'Automatique de Besançon, ainsi que l'ensemble des membres du laboratoire. Je voudrais dire combien la dynamique et la bonne ambiance régnant au sein du laboratoire ont été pour moi une formidable source de motivation.

Enfin, un grand merci à ma famille et à mes amis, avec une mention particulière pour Karine, mon épouse, qui m'ont soutenu et encouragé tout au long de cette petite aventure.

# Sommaire

**Glossaire des notations** ix

**1 Introduction** 1
1.1 Cadre applicatif . . . 2
1.1.1 Microrobotique et micromanipulation . . . 2
1.1.2 Micromanipulateur de cellule sans fil WIMS (WIreless Micromanipulation System) . . . 3
1.1.3 Objectifs expérimentaux et contraintes minimales à respecter . . . 3
1.2 Problématiques scientifiques abordées . . . 7
1.2.1 Cadre théorique . . . 7
1.2.2 Objectifs théoriques . . . 8
1.3 Plan . . . 9

**2 Apprentissage par renforcement** 11
2.1 Introduction . . . 11
2.2 Conventions . . . 12
2.3 Histoire et fondements théoriques . . . 12
2.3.1 Historique . . . 12
2.3.2 Boucle sensori-motrice . . . 14
2.3.3 Processus de décision markovien . . . 14
2.3.4 Objectifs . . . 15
2.3.5 Programmation dynamique . . . 17
2.3.6 Apprentissage par renforcement . . . 18
2.4 Méthodes directes . . . 18
2.4.1 Q-Learning . . . 19
2.4.2 Étude comportementale du Q-Learning . . . 22
2.4.3 TD($\lambda$) . . . 29
2.4.4 Autres méthodes directes . . . 33
2.4.5 Synthèse . . . 33
2.5 Méthodes indirectes . . . 33
2.5.1 Principe général . . . 33
2.5.2 Dyna-Q . . . 34
2.5.3 Autres méthodes indirectes . . . 36

2.5.4 Synthèse . . . . . 36
2.6 Méthodes directes et fonctions d'approximation . . . . . 38
2.6.1 Fonctions d'approximation différentiables . . . . . 38
2.6.2 Fonctions d'approximation linéaires . . . . . 39
2.6.3 Fonctions d'approximation non linéaires . . . . . 41
2.6.4 Apprentissage de fonctions d'approximation . . . . . 42
2.6.5 Synthèse . . . . . 43
2.7 Conclusion . . . . . 43

**3 Architectures comportementales et apprentissage par renforcement 45**
3.1 Introduction . . . . . 45
3.2 Architectures compétitives . . . . . 46
3.2.1 Introduction . . . . . 46
3.2.2 Architecture à subsomption . . . . . 47
3.2.3 Architecture à sélection d'actions de Pattie Maes . . . . . 48
3.2.4 Synthèse . . . . . 48
3.3 Architectures coopératives . . . . . 49
3.3.1 Introduction . . . . . 49
3.3.2 Architecture orientée schémas . . . . . 49
3.3.3 Architecture DAMN . . . . . 51
3.3.4 Synthèse . . . . . 52
3.4 Application à l'apprentissage par renforcement . . . . . 52
3.4.1 Subsomption et apprentissage par renforcement . . . . . 52
3.4.2 Sélection d'actions et apprentissage par renforcement . . . . . 53
3.4.3 Autres approches . . . . . 54
3.4.4 Une architecture originale : le W-Learning . . . . . 55
3.4.5 Synthèse . . . . . 57
3.5 Conclusion . . . . . 58

**4 Q-Learning parallèle 59**
4.1 Introduction . . . . . 59
4.2 Idée fondatrice . . . . . 59
4.3 Définitions et hypothèses fondamentales . . . . . 62
4.3.1 Situations et perceptions . . . . . 62
4.3.2 Commandes . . . . . 63
4.3.3 Récompenses . . . . . 63
4.3.4 Hypothèses fondamentales . . . . . 64
4.4 Fonctionnement d'un contrôleur parallèle . . . . . 66
4.4.1 Fonction de mémorisation . . . . . 67
4.4.2 Fonction de renforcement . . . . . 67
4.4.3 Fonction de fusion . . . . . 74
4.4.4 Fonction de décision . . . . . 76
4.5 Conclusion . . . . . 76

**5 Application du Q-Learning parallèle à l'exemple du labyrinthe 79**
5.1 Introduction . . . . . 79
5.2 Définition du cadre expérimental . . . . . 79
5.2.1 Labyrinthe de test . . . . . 79
5.2.2 Définition des outils de mesure . . . . . 80
5.3 Labyrinthe avec un seul type d'objet . . . . . 81
5.3.1 Stabilité . . . . . 81
5.3.2 Influence des paramètres . . . . . 83
5.3.3 Influence du nombre de fromages . . . . . 85
5.3.4 Synthèse . . . . . 86
5.4 Labyrinthe avec deux types d'objets . . . . . 86
5.4.1 Comparaison avec le cas précédent . . . . . 86
5.4.2 Influence des paramètres . . . . . 87
5.4.3 Apprentissage progressif . . . . . 87
5.4.4 Problème de fusion . . . . . 89
5.4.5 Synthèse . . . . . 91
5.5 Conclusion . . . . . 91

**6 Dyna-Q parallèle et application à la commande du macro-manipulateur WIMS 93**
6.1 Introduction . . . . . 93
6.2 Dyna-Q parallèle . . . . . 93
6.2.1 Introduction . . . . . 93
6.2.2 Fonctionnement . . . . . 94
6.2.3 Fonction mémorisation . . . . . 95
6.2.4 Fonction de modélisation parallèle . . . . . 95
6.2.5 Fonction d'optimisation asynchrone . . . . . 96
6.3 Application du Dyna-Q parallèle à l'exemple du labyrinthe . . . . . 97
6.3.1 Influence du coefficient de pondération $\gamma$ . . . . . 97
6.3.2 Influence de la dimension $h$ de l'historique . . . . . 97
6.3.3 Influence du nombre $N$ de mises à jour . . . . . 98
6.3.4 Comparaison du Q-Learning parallèle et du Dyna-Q parallèle . . . 98
6.3.5 Synthèse . . . . . 99
6.4 Application à la commande du macro-manipulateur WIMS . . . . . 99
6.4.1 Définition du cadre expérimental . . . . . 99
6.4.2 Apprentissage avec uniquement des cibles . . . . . 102
6.4.3 Apprentissage avec des cibles et des objets . . . . . 106
6.4.4 Test de robustesse . . . . . 106
6.4.5 Synthèse . . . . . 108
6.5 Conclusion . . . . . 108

**7 Dyna-Q parallèle et fusion de modèles** **111**
7.1 Introduction . . . 111
7.2 Idée fondatrice . . . 112
7.3 Fonctionnement dans le cadre général . . . 116
7.3.1 Définitions . . . 116
7.3.2 Algorithmes . . . 116
7.4 Application à l'exemple du labyrinthe . . . 122
7.4.1 Influence des paramètres . . . 122
7.4.2 Comparaison des fonctions de fusion . . . 123
7.5 Tests sur des labyrinthes avec impasses . . . 125
7.5.1 Impasse à quatre murs . . . 125
7.5.2 Conséquences des erreurs de la mise en correspondance temporelle 125
7.6 Conclusion . . . 127

**8 Conclusion et perspectives** **129**
8.1 Bilan des travaux . . . 129
8.2 Perspectives . . . 131
8.2.1 Évolutions de l'architecture parallèle . . . 131
8.2.2 Domaines d'application . . . 132
8.2.3 Étude du convoyage de cellules par poussée . . . 133
8.3 Conclusion . . . 134

**Annexes** **135**

**A Logiciel de simulation du labyrinthe** **135**
A.1 Application *système* . . . 135
A.2 Application *contrôleur* . . . 135
A.3 Application *pupitre de test* . . . 137
A.4 Description des exemples automatiques de démonstration du logiciel . . . 138

**B Plateforme de macro-manipulation WIMS** **141**
B.1 Macro-manipulateur . . . 141
B.1.1 Outil et objets . . . 141
B.1.2 Table de positionnement de l'aimant . . . 141
B.1.3 Électronique de puissance . . . 144
B.2 Système informatique . . . 144
B.2.1 Acquisition vidéo . . . 144
B.2.2 Calculateur . . . 145
B.2.3 Carte de commande . . . 145
B.3 Logiciel . . . 145

**C Description des enregistrements vidéo** **149**

**Bibliographie** **151**

# Table des figures


1.1 Fonctionnement du micromanipulateur de cellule sans fil WIMS. . . . . . 3
1.2 Différentes vues du micromanipulateur WIMS . . . . . . . . . . . . . . . . 4
1.3 Objectifs du contrôleur. . . . . . . . . . . . . . . . . . . . . . . . . . . 4
1.4 Illustration du phénomène d'hystérésis. . . . . . . . . . . . . . . . . . . 5

2.1 Boucle sensori-motrice. . . . . . . . . . . . . . . . . . . . . . . . . . . 14
2.2 Algorithme d'itérations sur les valeurs. . . . . . . . . . . . . . . . . . . 17
2.3 Schéma fonctionnel des méthodes directes. . . . . . . . . . . . . . . . . 18
2.4 Algorithme du Q-Learning. . . . . . . . . . . . . . . . . . . . . . . . . . 21
2.5 Le labyrinthe 10×10 en configuration initiale. . . . . . . . . . . . . . . . 22
2.6 Stratégies optimales pour différentes valeurs de $r_a$ et $\gamma$ . . . . . . . . . . . 23
2.7 Courbe d'apprentissage du Q-Learning sur l'exemple du labyrinthe 10×10 24
2.8 Stratégie optimale pour atteindre l'état terminal dans le cas d'un labyrinthe avec des murs. . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
2.9 Exemple de configuration où la valeur de $\gamma$ intervient sur le comportement de l'algorithme. . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
2.10 Algorithme du Sarsa($\lambda$). . . . . . . . . . . . . . . . . . . . . . . . . 30
2.11 Algorithme du Q($\lambda$) de Christopher Watkins. . . . . . . . . . . . . . . . 31
2.12 Courbes d'apprentissage du Sarsa($\lambda$) et du Q($\lambda$) par rapport au Q-Learning 31
2.13 Courbes du nombre de pas cumulés du Sarsa($\lambda$) et du Q($\lambda$) par rapport au Q-Learning . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
2.14 Schéma fonctionnel des méthodes indirectes. . . . . . . . . . . . . . . . 34
2.15 Algorithme du Dyna-Q. . . . . . . . . . . . . . . . . . . . . . . . . . . 35
2.16 Courbe d'apprentissage du Dyna-Q par rapport au Q-Learning . . . . . . 35
2.17 Comparaison de la solution trouvée des algorithmes Q-Learning, Q($\lambda$), Sarsa($\lambda$) et Dyna-Q en fonction de la taille du labyrinthe . . . . . . . . . . 37
2.18 Comparaison du temps d'apprentissage des algorithmes Q-Learning, Q($\lambda$), Sarsa($\lambda$) et Dyna-Q en fonction de la taille du labyrinthe . . . . . . . . . . 37
2.19 Exemple de partition d'un CMAC. . . . . . . . . . . . . . . . . . . . . . 40
2.20 Exemple de perceptron multicouche à une couche cachée. . . . . . . . . . . 41
2.21 Algorithme de choix de la structure en fonction de la dimension et de la nature de l'espace d'états du système. . . . . . . . . . . . . . . . . . . 44

3.1 Exemple d'architecture comportementale. . . . . . . . . . . . . . . . . . 46

3.2 Exemple d'architecture à subsomption. . . . . . . . . . . . . . . . . . . . . . 47
3.3 Exemple simplifié de l'architecture à sélection d'actions. . . . . . . . . . . 48
3.4 Exemple de trajectoire classique produite par une architecture compétitive. 49
3.5 Exemple d'architecture orientée schémas. . . . . . . . . . . . . . . . . . . . 50
3.6 Exemple de trajectoire classique produite par une architecture orientée schéma. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
3.7 Exemple d'arbitre DAMN. . . . . . . . . . . . . . . . . . . . . . . . . . . 51
3.8 Exemple d'architecture à sélecteur. . . . . . . . . . . . . . . . . . . . . . . 53
3.9 Exemple d'architecture W-Learning . . . . . . . . . . . . . . . . . . . . . 55

4.1 Exemple de labyrinthe. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 60
4.2 Principe de l'architecture parallèle. . . . . . . . . . . . . . . . . . . . . . 61
4.3 Boucle sensori-motrice. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62
4.4 Les interactions entre la souris et les murs nuisent à l'indépendance des perceptions. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65
4.5 Schéma fonctionnel de l'approche parallèle. . . . . . . . . . . . . . . . . . 66
4.6 Algorithme du Q-Learning parallèle. . . . . . . . . . . . . . . . . . . . . . 66
4.7 Nouvelles fonctionnalités de la fonction de renforcement présentée par la figure 4.5. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68
4.8 Exemples de mises en correspondance. . . . . . . . . . . . . . . . . . . . . 69
4.9 Algorithme de la fonction de mise en correspondance temporelle. . . . . . 70
4.10 Illustration du problème de l'attribution des récompenses . . . . . . . . . 72
4.11 Algorithme de la fonction d'attribution des récompenses. . . . . . . . . . . 73
4.12 Algorithme de la fonction de renforcement parallèle. . . . . . . . . . . . . 74
4.13 Comment estimer $Q_t$ ? . . . . . . . . . . . . . . . . . . . . . . . . . . . . 75
4.14 Algorithme de la fonction de fusion (opérateur somme). . . . . . . . . . . 75
4.15 Algorithme de la fonction de décision ($\epsilon$-gourmand). . . . . . . . . . . . 76

5.1 Évolution du gain moyen en fonction du nombre de périodes d'échantillonnage . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 80
5.2 Labyrinthes de test. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 81
5.3 Illustration du processus d'auto-vérification de l'attribution des récompenses. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 82
5.4 Influence du coefficient d'apprentissage $\alpha$ sur le gain limite et sur le temps de convergence . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 83
5.5 Influence du coefficient de pondération $\gamma$ sur la limite de la récompense moyenne et sur le temps de convergence . . . . . . . . . . . . . . . . . 84
5.6 Influence du coefficient de pondération $\gamma$ sur le comportement du contrôleur. 85
5.7 Influence du nombre de fromages sur le gain limite et sur le temps de convergence . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 86
5.8 Comparaison de l'évolution du gain moyen avec un labyrinthe contenant 10 murs et avec un labyrinthe sans mur . . . . . . . . . . . . . . . . . . 87
5.9 Influence du coefficient d'apprentissage $\alpha$ sur le gain limite et sur le temps de convergence . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 88

5.10 Influence du coefficient de pondération $\gamma$ sur la limite de la récompense moyenne et sur le temps de convergence . . . . . . . . . . . . . . . . . . . 88
5.11 Comparaison de l'apprentissage progressif et de l'apprentissage normal . . 89
5.12 Exemple de formation d'un cycle entre trois états . . . . . . . . . . . . . . 90
5.13 Évolution du gain dans un labyrinthe avec 5 fromages et une impasse de 4 murs au centre du labyrinthe . . . . . . . . . . . . . . . . . . . . . . . . . . 90

6.1 Schéma fonctionnel du Dyna-Q parallèle. . . . . . . . . . . . . . . . . . . . 94
6.2 Algorithme du Dyna-Q parallèle. . . . . . . . . . . . . . . . . . . . . . . . . 95
6.3 Algorithme de la fonction de modélisation parallèle. . . . . . . . . . . . . . 96
6.4 Algorithme de la fonction d'optimisation asynchrone. . . . . . . . . . . . . 97
6.5 Influence de la dimension $h$ de l'historique sur le gain limite et sur le temps de convergence . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 97
6.6 Influence du nombre de mises à jour par période d'échantillonnage sur le gain limite et sur le temps de convergence . . . . . . . . . . . . . . . . . . 98
6.7 Comparaison de l'évolution des gains moyens obtenus avec le Dyna-Q parallèle et avec le Q-Learning parallèle . . . . . . . . . . . . . . . . . . . 99
6.8 Boucle sensori-motrice de commande du macro-manipulateur WIMS. . . . 100
6.9 Exemple de situation de manipulation sur le WIMS. . . . . . . . . . . . . 101
6.10 Influence du nombre de mises à jour par période d'échantillonnage sur le gain limite et sur le temps de convergence . . . . . . . . . . . . . . . . . 102
6.11 Influence de la dimension de l'historique sur le gain limite et sur le temps de convergence . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 103
6.12 Évolution du gain moyen avec le système simulé avec 10 cibles . . . . . . . 104
6.13 Exemple de trajectoire de l'outil sur le système simulé après apprentissage 104
6.14 Évolution du gain moyen avec le système réel avec 10 cibles . . . . . . . . 105
6.15 Exemple de trajectoire de l'outil sur le système réel après apprentissage . 105
6.16 Évolution du gain moyen avec le système réel après l'ajout de 10 d'objets et après un apprentissage avec 10 cibles . . . . . . . . . . . . . . . . . . . 107
6.17 Exemple de trajectoire de l'outil sur le système réel après apprentissage . 107
6.18 Évolution du gain moyen avec le système réel après la rotation de 45° du manipulateur . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 108
6.19 Comparaison des trajectoires de l'outil générées par le contrôleur Dyna-Q parallèle et un opérateur humain entraîné. . . . . . . . . . . . . . . . . . . 109

7.1 Exemple de situation. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 112
7.2 Illustration du fonctionnement de la fonction de fusion de modèles pour la position (0,2) de la situation 7.1. . . . . . . . . . . . . . . . . . . . . . . 113
7.3 Illustration du fonctionnement de la fonction de fusion de modèles pour la position (-1,1) de la situation 7.1. . . . . . . . . . . . . . . . . . . . . . . 114
7.4 Représentation des projections et des translations de la fonction de fusion de modèles. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 117
7.5 Nouvelles fonctionnalités de la fonction de fusion présentée au chapitre 4 par la figure 4.5. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 117

7.6 Algorithme de la fonction de fusion de modèles. . . . . . . . . . . . . . . . 119
7.7 Algorithme de la fonction de planification. . . . . . . . . . . . . . . . . . . 120
7.8 Influence du nombre $N_c$ d'optimisations de la carte sur le gain limite et sur le temps de convergence . . . . . . . . . . . . . . . . . . . . . . . . . . 123
7.9 Influence du coefficient de pondération $\gamma$ sur la limite de la récompense moyenne et sur le temps de convergence . . . . . . . . . . . . . . . . . . 124
7.10 Comparaison de l'évolution des gains moyens obtenus avec le Dyna-Q parallèle avec fusion de modèles et avec le Dyna-Q parallèle sans fusion de modèles . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 124
7.11 Comparaison des gains obtenus dans un labyrinthe avec 5 fromages et une impasse de quatre murs avec fusion de modèles et sans fusion de modèles . 125
7.12 Illustration de l'influence d'une erreur de mise en correspondance sur la fusion de modèles. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 126
7.13 Exemples de trajectoires obtenues sans réapprentissage avec un contrôleur à fusion de modèles entraîné sur un labyrinthe avec 5 fromages et 10 murs dispersés. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 127

8.1 Exemple de convoyage vers la droite de trois objets après apprentissage . . 134

A.1 Fenêtre de l'application système du labyrinthe. . . . . . . . . . . . . . . . 136
A.2 Fenêtre de l'application contrôleur. . . . . . . . . . . . . . . . . . . . . . 136
A.3 Fenêtre de l'application pupitre de test. . . . . . . . . . . . . . . . . . . . 138

B.1 Vue d'ensemble de la plateforme de macro-manipulation WIMS. . . . . . . 142
B.2 Macro-manipulateur WIMS. . . . . . . . . . . . . . . . . . . . . . . . . . 143
B.3 Table de positionnement deux axes. . . . . . . . . . . . . . . . . . . . . . 144
B.4 Étage de puissance pour l'alimentation des moteurs à courant continu. . . 145
B.5 Traitement de l'image vidéo. . . . . . . . . . . . . . . . . . . . . . . . . . 146
B.6 Fenêtre de l'application système du macro-manipulateur WIMS. . . . . . . 147

# Glossaire des notations

$\alpha$ coefficient d'apprentissage
$\epsilon$ taux d'exploration de l'$\epsilon$-favori
$\mathsf{e}$ fonction d'éligibilité d'un couple *état—commande*
$\gamma$ coefficient de pondération des récompenses futures
$\mathsf{G}$ gain (somme $\gamma$-pondérée des récompenses futures)
$\lambda$ coefficient de redistribution
$\mathsf{m}$ modèle de transition
$N$ nombre de mises à jour par période d'échantillonnage
$\pi$ fonction de stratégie de commande (contrôleur)
$\pi^*$ fonction de stratégie optimale de commande (contrôleur)
$\mathsf{p}$ fonction probabiliste de transition
$\mathsf{Q}$ fonction d'utilité par rapport à un couple *état—commande* (estimation de l'espérance de gain)
$\mathsf{Q}^*$ fonction d'utilité optimale
$\mathsf{Q}_i$ valeur initiale de la fonction d'utilité $\mathsf{Q}$
$r$ récompense scalaire
$\mathsf{r}$ fonction de récompense immédiate
$\mathbb{R}$ ensemble des nombres réels
$t$ temps ou période d'échantillonnage
$u$ commande
$\mathcal{U}$ ensemble des commandes (espace de commandes)

### Notations spécifiques au chapitre 2

$x$ état du système
$\mathcal{X}$ ensemble des états (espace d'états)

### Notations spécifiques aux chapitres 4 à 7

$\mathsf{a}$ fonction d'attribution des récompenses
$a$ triplet *perception précédente—perception suivante—récompense attribuée*

| | |
|---|---|
| $A$ | multi-ensemble[1] de triplets $a$ |
| $\mathsf{c}$ | fonction de mise en correspondance temporelle |
| $c$ | couple *perception précédente—perception suivante* |
| $C$ | multi-ensemble[1] de couples $c$ |
| $\mathcal{C}$ | carte de la méthode de fusion de modèles |
| $\mathsf{f}$ | fonction de fusion |
| $h$ | dimension de l'historique mémorisé par le modèle $\mathsf{m}$ pour chaque couple *perception—commande* |
| $\mathsf{M}$ | modèle global de transition entre positions dans la carte $\mathcal{C}$ |
| $N_c$ | nombre d'optimisations totales de la fonction d'utilité $\mathsf{Q_c}$ par période d'échantillonnage |
| $O$ | position du système dans la carte $\mathcal{C}$ (projeté de l'origine du repère défini sur l'ensemble $\mathcal{X}$ des perceptions) |
| $p$ | position imaginaire du système dans la carte $\mathcal{C}$ |
| $\Pi$ | opérateur de projection de $\mathcal{X}$ dans le plan de la carte $\mathcal{C}$ |
| $\mathsf{q}$ | fonction d'utilité d'un couple *perception—commande* (estimation de l'espérance de gain d'un couple *perception—commande*) |
| $\mathsf{Q_c}$ | fonction d'utilité d'un couple *position—commande* (dans la carte) |
| $R$ | récompense globale (multi-ensemble[1] de scalaires) |
| $X$ | situation (état global du système, multi-ensemble[1] *de* perceptions[2] de cardinal $n$) |
| $x$ | perception élémentaire (élément de $X$) |
| $\mathcal{X}$ | ensemble *des* perceptions[2] (espace de perceptions) |

1. Ensemble dans lequel un même élément peut apparaître plusieurs fois. Autrement dit, la multiplicité d'un élément d'un multi-ensemble peut être supérieure à 1.
2. On notera la distinction entre un ensemble de perceptions et l'ensemble des perceptions.

# Chapitre 1

# Introduction

Imaginez que vous vous installez dans une grande ville dont vous ne connaissez rien. Si vous devez vous rendre en voiture à la mairie, que vous n'avez ni plan, ni pancarte, ni personne pour vous renseigner, vous êtes contraint de la trouver par hasard. La première fois, si vous avez un peu de chance, vous trouverez la mairie au bout d'un moment. La deuxième fois, vous n'irez plus complètement au hasard et votre parcours sera probablement moins long mais il a de bonnes chances de ne pas être le plus court. Au bout de plusieurs essais, vous trouverez le chemin le plus rapide et vous serez même capable d'en trouver un autre s'il y a des travaux sur votre route qui vous empêchent de passer par votre chemin habituel. Vous aurez donc appris à aller rapidement et sûrement à la mairie.

Prenons un autre exemple. Dans un jeu vidéo, vous dirigez un personnage devant en affronter d'autres gérés par l'ordinateur. A la première partie, vous allez être sans aucun doute rapidement battu. Pourtant, au fil des parties, vous vous améliorerez, apprendrez à maîtriser votre personnage et surtout vous trouverez les failles des personnages informatiques, en un mot, vous aurez appris à vaincre l'ordinateur.

Dans ces deux exemples, la faculté principale qui vous a permis de parvenir à votre but est l'apprentissage. Cette faculté est un élément essentiel pour trouver une stratégie d'action quand le système est complètement ou partiellement inconnu (la ville dans le premier cas, le comportement de l'ordinateur dans le second).

Développée depuis les années 1980, la théorie de l'apprentissage par renforcement a pour objectif de reproduire cette faculté qui nous est naturelle. L'apprentissage par renforcement est une méthode de contrôle automatique qui ne nécessite pas de connaître le système mais simplement un critère de satisfaction, comme par exemple la satisfaction de trouver la mairie ou de gagner la partie contre l'ordinateur.

Les applications de cette approche sont nombreuses notamment en robotique mobile, domotique, robotique ludique et plus généralement dans les applications où le comportement du système est inconnu. Voici quelques exemples :

– un robot se déplace dans des ateliers pour aller chercher des pièces, il apprend à trouver son chemin même si la configuration spatiale de l'atelier change (nouvelles machines, ouvriers et machines en mouvement, *etc.*),

– un robot aspirateur apprend à parcourir entièrement le sol d'une habitation qu'il ne connaît pas au départ et qui peut changer (par exemple des meubles qui ne sont pas à la même place),
– dans un jeu vidéo, les personnages gérés par l'ordinateur progressent en même temps que le joueur,
– dans un jeu de réflexion (le backgammon par exemple), l'ordinateur apprend à jouer de mieux en mieux à chaque partie,

Parmi les autres applications possibles, la microrobotique et plus particulièrement la micromanipulation posent des problèmes liés à la modélisation des interactions dynamiques à l'échelle microscopique qui s'inscrivent dans le cadre de l'apprentissage. C'est en réalité par et pour cette application de micromanipulation que nous nous sommes intéressés en premier lieu à l'apprentissage.

## 1.1 Cadre applicatif

### 1.1.1 Microrobotique et micromanipulation

La microrobotique est une discipline relativement nouvelle dont l'objectif est de concevoir, réaliser et commander des systèmes de tailles millimétriques ou micrométriques capables d'effectuer avec une grande précision des fonctions données dans un milieu donné.

En terme d'applications, la microrobotique vise la manipulation d'éléments biologiques (cellules, micro-organismes, *etc.*) ou l'assemblage de micro-pièces (micro-engrenages, micro-miroirs, *etc.*). Elle concerne aussi l'exploration et l'inspection en milieu fortement confiné comme la visite et la réparation de canalisations de très faibles diamètres. Dans le domaine médical, la microrobotique est utilisée dans le diagnostic, la thérapie (administration des médicaments) ou les opérations de chirurgie micro-invasive (biopsie, traitement des anévrismes, *etc.*).

Parmi ces applications, l'aide à la manipulation de cellules in vitro fait partie des enjeux d'aujourd'hui. Plus particulièrement dans le domaine de la fécondation in vitro, les spécialistes aimeraient pouvoir estimer la qualité des ovocytes par quelques tests déterminants afin de choisir les meilleurs candidats à la fécondation. Ce type d'application nécessite notamment la capture, le convoyage et le maintien d'un ovocyte sur un site de test.

A ce titre, le Laboratoire d'Automatique de Besançon développe actuellement de nouveaux outils destinés à manipuler les cellules biologiques. Il a notamment démarré un programme de recherche visant à concevoir un micromanipulateur de cellule sans fil, baptisé WIMS (WIreless Micromanipulation System). Cette thèse s'inscrit dans ce cadre et traite de la commande de ce micromanipulateur.

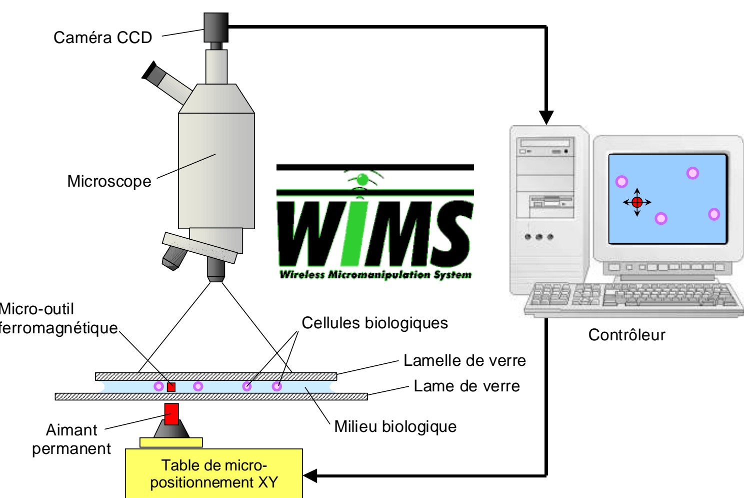



FIG. 1.1 – *Fonctionnement du micromanipulateur de cellule sans fil WIMS.*

### 1.1.2 Micromanipulateur de cellule sans fil WIMS (WIreless Micromanipulation System)

Le fonctionnement du micromanipulateur WIMS repose sur l'utilisation d'un champ magnétique pour déplacer à distance un micro-outil (*cf.* figures 1.1 et 1.2). La faible taille du micro-outil permet de l'insérer directement dans le milieu biologique avec les objets à manipuler. Ainsi, il est possible de manipuler les objets biologiques directement sous microscope entre lame et lamelle.

Le champ magnétique est généré par un aimant permanent que l'on peut déplacer à l'aide d'une table de micro-positionnement deux axes. Le manipulateur intègre donc deux degrés de liberté (les deux directions du plan des lames de verre). Un système de contrôle commande la table de micro-positionnement et reçoit les informations sur l'état du système par les images vidéo issues du microscope.

### 1.1.3 Objectifs expérimentaux et contraintes minimales à respecter

#### Objectifs expérimentaux

L'un des objectifs de ce projet est de réaliser avec cet outil des opérations automatiques de capture et de convoyage d'un ovocyte vers un site de test. On désire donc concevoir un contrôleur capable de déplacer l'outil entre les cellules vers une position cible (*cf.* figure 1.3a) ou capable de pousser une cellule vers un banc de test en évitant les autres cellules (*cf.* figure 1.3b).

Outre la génération de trajectoires évitant les cellules, il faut surmonter trois difficultés majeures pour établir une loi de commande.

Tout d'abord, le manipulateur WIMS présente un inconvénient inhérent à son mode d'actionnement : il existe une hystérésis dans la transmission du mouvement de l'aimant à l'outil. Quand l'aimant se déplace, l'outil le suit à une certaine distance. D'une certaine

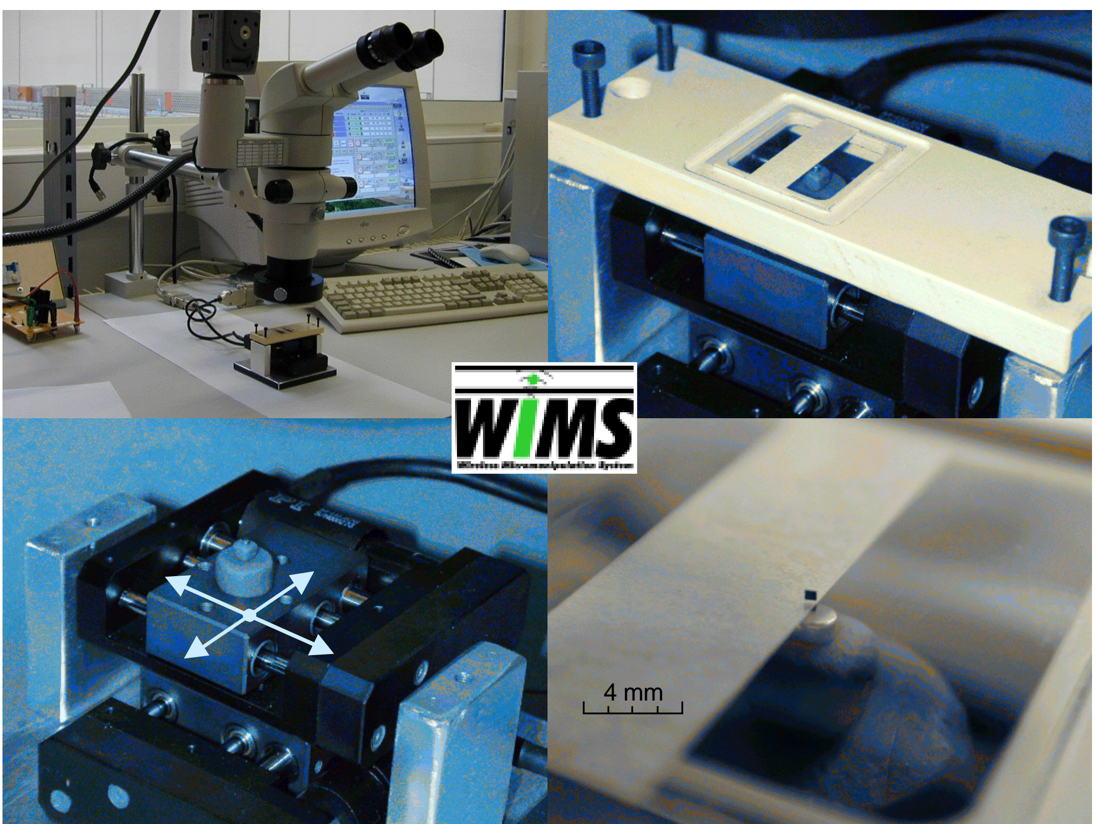
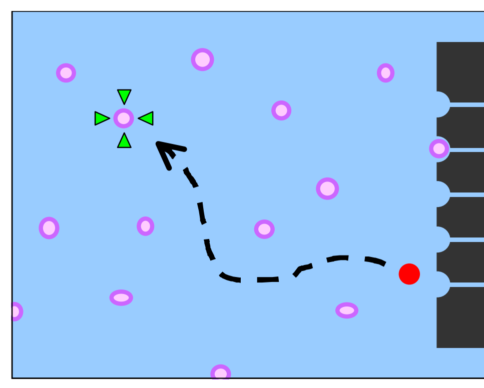
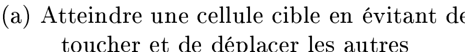
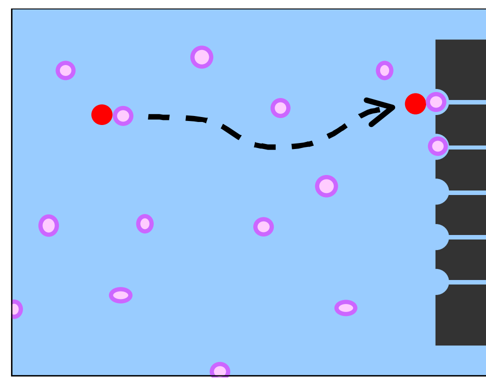



FIG. 1.2 – *Différentes vues du micromanipulateur WIMS actuellement en développement au Laboratoire d'Automatique de Besançon (en haut à gauche : vue d'ensemble, en haut à droite : le micromanipulateur, en bas à gauche : la table de micro-positionnement, en bas à droite : détail de l'aire de manipulation et de l'outil ferromagnétique).*





(a) Atteindre une cellule cible en évitant de toucher et de déplacer les autres



(b) Amener la cellule cible vers le banc de test en la poussant et en évitant les autres

FIG. 1.3 – *Objectifs du contrôleur.*

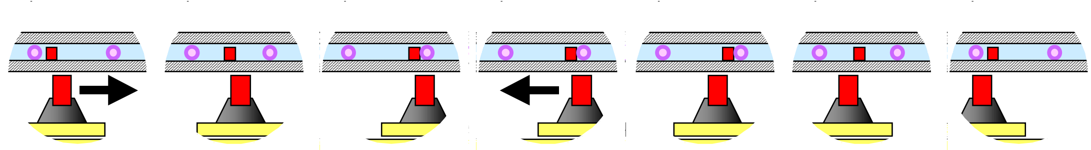


FIG. 1.4 – *Illustration du phénomène d'hystérésis.*

manière, l'aimant « traîne » l'outil. Lorsqu'il faut changer de direction, l'outil ne bouge pas tant que l'aimant n'est pas passé sous lui (*cf.* figure 1.4). Cette hystérésis induit en outre un couplage entre les deux axes de déplacement de l'outil qui rend délicat son positionnement dans le plan.

Ensuite, certains phénomènes physiques dus aux frottements ne sont pas aisés à modéliser. Notamment, quand l'aimant se déplace, l'outil le suit en effectuant de petits bonds. Son mouvement est saccadé car il se bloque puis avance d'un coup sec. Ce phénomène est appelé stick-slip. Il est dû à l'état de la surface de l'outil en contact avec la lame de verre. Ce phénomène est imprévisible et rend donc la commande très difficile.

Enfin, lors d'une tâche de poussée d'un ovocyte par l'outil, l'interaction mécanique entre l'outil et l'ovocyte et notamment les conditions d'adhésion sont difficiles à modéliser. En effet, les caractéristiques des ovocytes (forme, taille, souplesse, état de sa surface, *etc.*) sont variables. Il est difficile dans ces conditions de prévoir une commande qui fonctionne avec tous les ovocytes.

Pour ces raisons, nous avons choisi de ne pas chercher un modèle précis de la dynamique du micromanipulateur et des interactions avec les cellules. Nous nous sommes au contraire orientés vers une approche par apprentissage par renforcement qui ne nécessite pas de modélisation *a priori*.

Cette approche par apprentissage par renforcement est fondée sur la théorie de la commande optimale et consiste à rechercher par essais et erreurs une stratégie de commande qui maximise un critère de satisfaction associé à des transitions ou à des états donnés. En d'autre termes, le contrôleur va apprendre à effectuer sa tâche sans connaissances initiales sur le système, mais en recevant un signal caractérisant une satisfaction vis-à-vis de la tâche à effectuer. Il devra aussi être capable de s'adapter à n'importe quels changements des caractéristiques du système.

L'objet de cette thèse est donc de concevoir un contrôleur par apprentissage par renforcement du manipulateur WIMS. Dans cette thèse, nous nous limiterons à l'étude du premier objectif de manipulation : atteindre une cellule cible en évitant les autres cellules quelle que soit leur configuration spatiale.

Le micromanipulateur WIMS est actuellement en cours de développement (Michaël Gauthier et Emmanuel Piat 2002). Pour pouvoir tester nos algorithmes, nous utilisons un manipulateur aux dimensions macroscopiques dont la réalisation technique ne pose aucun problème. Ce macro-manipulateur est identique sur le principe au micromanipulateur, mais ses dimensions ont été multipliées par 20 par rapport au micromanipulateur. A cette échelle, les ovocytes sont évidemment fictifs et sont représentés par des billes de

plastique d'environ 3 mm de diamètre (un ovocyte humain mesure environ 150 $\mu$m). L'annexe B fournit une description détaillée de ce macro-manipulateur WIMS.

Il est bien entendu que le macro-manipulateur avec des ovocytes fictifs a un comportement différent de la future application, mais notre but n'est pas de développer un contrôleur dédié à ce macro-manipulateur, mais bien un contrôleur capable de s'adapter à n'importe quel manipulateur permettant de déplacer un micro-outil au milieu d'objets disposés sur un plan.

### Espace d'états du système

Les informations disponibles sur l'état du système sont les images vidéo. Grâce à des opérations de traitement d'images, il est admis que l'on peut calculer la position des différents objets et de l'outil.

Pour pouvoir adapter la trajectoire de l'outil à la configuration spatiale des objets, il faut tenir compte de leur position respective. Si $n$ objets sont disposés sur l'aire de manipulation et que chaque objet peut être dans $m$ positions possibles, l'espace d'états d'un tel système est de cardinal $m^n$.

Même avec peu d'objets, cet espace d'états demeure très grand. Il faudra donc trouver ou concevoir un algorithme capable de fonctionner avec des espaces d'états de grandes dimensions.

### Espace de commandes

Pour l'espace de commande, nous nous sommes limités dans cette thèse au strict minimum. Les commandes sont discrètes et réduites à l'essentiel. Elles permettent de déplacer l'outil dans les quatre directions du plan de la lame de verre (haut, bas, gauche, droite) selon deux avances différentes (déplacements fins ou déplacements grossiers).

### Puissance de calcul

Le contrôleur est implanté sur un micro-ordinateur de type PC (*cf.* annexe B). La seule exigence en terme de temps de calcul est que le contrôleur soit assez rapide pour commander le système en temps réel. La possibilité de réaliser des calculs en temps masqué par rapport à l'exécution des commandes sur le manipulateur n'est pas exclue.

Par ailleurs, l'algorithme utilisé doit permettre un apprentissage rapide pour obtenir une stratégie de commande du système en un temps raisonnable (quelques dizaines de minutes au maximum).

## 1.2 Problématiques scientifiques abordées

### 1.2.1 Cadre théorique

#### Boucle sensori-motrice

Notre approche se situe dans le cadre classique d'une boucle sensori-motrice. A chaque période d'échantillonnage, le contrôleur reçoit des informations sur l'état du système puis décide d'une commande qui agit sur le système.

Nous nous attachons à la commande de systèmes dont le contrôleur perçoit des informations sous la forme d'un ensemble de variables indépendantes et markoviennes appelées *perceptions*. Cet ensemble de perceptions, appelé *situation du système*, n'est pas ordonné et peut être de cardinal variable au cours du temps. Toutes les perceptions sont issues d'un même espace (ou d'un même ensemble si ce n'est pas un espace vectoriel).

En résumé, l'état des systèmes étudiés est décomposable en une situation $X_t$ définie par le produit cartésien de plusieurs perceptions $x_t^i$ appartenant à un unique ensemble $\mathcal{X}$, soit :

$$X_t = \{x_t^i \,|\, 1 \leq i \leq n_t, x_t^i \in \mathcal{X}\} \tag{1.1}$$

avec :

- $X_t$ la situation du système à l'instant $t$,
- $n_t$ le cardinal de la situation $X_t$,
- $x_t^i$ une perception,
- $\mathcal{X}$ l'espace des perceptions.

Les espaces d'états des systèmes considérés sont donc de la forme $\mathcal{X}^{n_t}$. Beaucoup de systèmes peuvent être appréhendés sous cette forme. C'est le cas de l'application de manipulation WIMS : chaque objet constitue une perception élémentaire, l'état du système est défini par l'ensemble des objets visibles par la caméra.

#### Apprentissage par renforcement

Le formalisme utilisé est celui des processus décisionnels de Markov et plus particulièrement de la commande optimale de Richard Bellman. L'objectif de ces approches consiste à rechercher une commande qui conduise le système dans un état final donné tout en maximisant (ou minimisant) un gain (ou un coût) associé à la trajectoire suivie par le système.

Au sein de ce formalisme, la classe des méthodes dites d'apprentissage par renforcement permet d'établir une stratégie de commande donnant une trajectoire de gain maximal (ou de coût minimal) sans connaître de modèle du système. Dans ce but, le contrôleur reçoit à chacune de ses interactions avec le système, un signal de récompense qui qualifie le nouvel état du système ou son changement d'état. Ainsi, l'objectif du contrôleur est de trouver les commandes qui permettent de maximiser le gain défini par la somme de récompenses futures (la somme est pondérée si l'horizon est infini).

Dans le cadre classique, le signal de récompense est un scalaire. Par rapport à la complexité des systèmes considérés, nous pensons que cette définition est réductrice car elle ne permet pas de qualifier des situations où une commande aurait provoqué à la fois une conséquence positive et une conséquence négative. Pour cette raison, nous avons choisi une définition plus large du signal de récompense : à chaque période d'échantillonnage, le contrôleur reçoit un multi-ensemble [1] $R_t$ de scalaires supposés non ordonnés. Le cardinal de $R_t$ est égal à celui de la situation $X_t$.

Le cas non ordonné est un cas d'étude général qui suppose que le contrôleur ne connaît pas d'emblée les perceptions auxquelles les récompenses sont associées. On peut alors parler de récompenses non étiquetées *a priori*. Ce cas de figure correspond par exemple à un enfant qui serait puni sans la moindre explication par rapport à sa situation courante. C'est alors à l'enfant de comprendre à quel élément de la situation se rapporte la punition pour éviter à l'avenir qu'elle ne se reproduise. Le cas ordonné où les récompenses sont classées dans le même ordre que leur perception correspondante dans la situation est donc un cas particulier du cas envisagé et serait beaucoup plus informatif pour le contrôleur.

### 1.2.2 Objectifs théoriques

Notre objectif premier consiste à réaliser la synthèse d'un contrôleur par le biais de l'apprentissage. Dans cette optique, on suppose que le modèle du système est complètement inconnu. L'objectif du contrôleur étant d'apprendre à commander le système pour obtenir un gain maximum, les objectifs concrets de contrôle (comme par exemple atteindre une consigne) seront traduits par une politique de récompense adéquate que l'on suppose connue. Notons que nous nous imposons des durées d'apprentissage très courtes, de l'ordre de quelques dizaines de minutes si on vise une application réelle de manipulation de cellules.

Par rapport à la complexité des systèmes étudiés, les algorithmes classiques d'apprentissage par renforcement ne sont pas utilisables car ils sont limités par leur vitesse d'apprentissage à des systèmes dont l'espace d'états est de faible dimension. Nous pensons qu'il est nécessaire de diviser le problème de commande globale en de multiples sous-problèmes simples dont l'apprentissage par une méthode classique ne pose pas de problème.

Notre approche s'inspire donc à la fois des architectures multi-comportements (ou comportementales) et de l'apprentissage par renforcement. Schématiquement, l'avantage des architectures comportementales est de permettre de réaliser la commande de systèmes complexes à l'aide de modules élémentaires plus simples à concevoir. L'inconvénient est que les comportements sont généralement spécifiés de manière *ad hoc* et leur coordination (ou fusion) conduit souvent à des cycles oscillants dus à la présence de maxima locaux. Les problématiques théoriques que nous abordons dans cette thèse sont donc les suivantes :

- créer une architecture de commande constituée de multiples comportements simples à apprendre dans le but de réduire la complexité du système et ainsi d'accélérer l'apprentissage,

1. Ensemble dans lequel un même élément peut apparaître plusieurs fois. Autrement dit, la multiplicité d'un élément d'un multi-ensemble peut être supérieure à 1.

- créer une architecture auto-constructive qui évite de devoir spécifier précisément les comportements au sein de l'architecture,
- créer une architecture dont la fonction de coordination (ou fusion) ne génère pas de maximum local ni de cycle oscillant.

## 1.3 Plan

Ce mémoire se compose de huit chapitres. La premier et le dernier sont destinés respectivement à introduire et à conclure notre réflexion, tandis que les chapitres 2 à 7 décrivent le contenu scientifique de la thèse.

Le chapitre 2 traite des différentes méthodes d'apprentissage par renforcement. Il commence par en décrire la théorie, puis énumère les algorithmes qui nous semblent les plus représentatifs et les plus justes par rapport à nos problématiques.

Dans le troisième chapitre, nous nous intéressons aux architectures dites comportementales qui proposent des solutions pour les systèmes de très grandes dimensions comme notre application. Nous présentons notamment les approches qui allient architectures comportementales et apprentissage par renforcement. A l'issue de ce chapitre, nous définissons l'orientation de notre axe de recherche.

Le chapitre 4 expose l'approche que nous avons développée pour l'application WIMS. Cette approche, appelée Q-Learning parallèle, consiste en une parallélisation de l'algorithme classique du Q-Learning. Son objectif est de réduire la complexité de l'état du système et d'augmenter la vitesse d'apprentissage. Son fonctionnement est décrit en détail en s'appuyant sur un exemple simple de labyrinthe.

Le cinquième chapitre est consacré à l'étude de l'algorithme parallèle sur l'exemple du labyrinthe. Il a pour but de valider le principe parallèle et de déterminer les caractéristiques, l'influence des paramètres et les performances de l'algorithme.

Le chapitre 6 présente une évolution de notre algorithme afin de l'adapter à la commande de systèmes réels. Ce nouvel algorithme est une version à apprentissage plus rapide que le précédent. Il est fondé sur l'utilisation d'un algorithme de type Dyna-Q. Après avoir décrit son fonctionnement, nous étudions son application à la commande du système réel de manipulation WIMS.

Le septième chapitre propose une évolution de l'architecture pour répondre aux problèmes de maxima locaux qui surviennent lorsqu'on fusionne des comportements. La méthode est décrite puis testée sur des labyrinthes complexes.

La chapitre 8 présente nos conclusions sur les travaux réalisés puis développe nos perspectives de recherches sur les évolutions et les applications possibles de l'architecture parallèle.

# Chapitre 2

# Apprentissage par renforcement

*Ce chapitre présente les origines historiques et théoriques de l'apprentissage par renforcement. Il décrit dans le détail les algorithmes élémentaires (Q-Learning, Q($\lambda$), Sarsa($\lambda$) et Dyna-Q) classés en deux groupes : les méthodes directes, qui n'utilisent aucun modèle du système à contrôler, et les méthodes indirectes, qui auto-construisent un modèle. Ce chapitre aborde ensuite les approches qui utilisent des fonctions d'approximation pour commander des systèmes à plus grands espaces d'états. Pour terminer, nous concluons vis-à-vis des différents algorithmes et précisons le choix de notre axe de recherche.*

## 2.1 Introduction

L'apprentissage par renforcement recouvre une classe de méthodes d'apprentissage automatique à mi-chemin entre l'apprentissage supervisé et l'apprentissage non supervisé. Dans l'apprentissage supervisé, le système apprenant propose une réponse à chaque situation qu'on lui présente. Un superviseur lui montre alors la bonne réponse et le système peut ainsi corriger sa réponse. Ce principe nécessite de connaître au moins une partie des réponses à apporter aux différentes situations. A l'inverse, dans l'apprentissage non supervisé, le système ne perçoit que les situations et doit apprendre des « régularités » dans ces situations afin d'en effectuer une classification.

L'apprentissage par renforcement se situe entre ces deux méthodes. A chaque changement de situation, le système reçoit une note ou renforcement qui qualifie ce changement. Par exemple, si vous cherchez la mairie dans une ville inconnue, vous serez satisfait de la trouver et insatisfait tant que vous ne l'aurez pas trouvée. Le renforcement n'indique pas la manière de trouver la mairie, mais simplement si vous l'avez trouvée ou non.

Ce chapitre se compose de quatre parties. La première introduit l'apprentissage par renforcement à partir de ses origines historiques et théoriques. La deuxième est consacrée à l'étude des algorithmes par renforcement dits directs car ils apprennent directement à partir de leurs expériences. Vient ensuite l'étude des méthodes dites indirectes qui construisent un modèle partiel du système à contrôler. La quatrième partie présente des

approches utilisant des fonctions d'approximation pour permettre l'apprentissage avec des systèmes plus complexes ou continus.

## 2.2 Conventions

Par souci de lisibilité des notations et des équations mathématiques, nous définissons les conventions suivantes :
- les caractères italiques minuscules désignent des variables élémentaires scalaires ou vecteurs (par exemple $x$, $r$, $u$),
- les caractères italiques majuscules désignent des multi-ensembles[1] de variables élémentaires (par exemple $X$, $R$ introduits au chapitre 4). Pour simplifier, nous utilisons le terme ensemble pour désigner un multi-ensemble,
- les caractères italiques cursifs désignent soit les ensembles des valeurs que peuvent prendre les variables élémentaires, soit des espaces vectoriels (par exemple $\mathcal{X}$, $\mathcal{U}$),
- les caractères sans sérif minuscules et majuscules désignent des fonctions (par exemple $\mathsf{q}$, $\mathsf{Q}$, $\mathsf{p}$, $\mathsf{a}$, $\mathsf{c}$, $\mathsf{f}$),
- les lettres grecques désignent des paramètres réels (par exemple $\alpha$, $\gamma$, $\epsilon$),
- l'opérateur $|\mathcal{X}|$ renvoie le cardinal de l'ensemble $\mathcal{X}$.

Un glossaire des notations page ix rappelle l'ensemble des variables utilisées dans le mémoire.

## 2.3 Histoire et fondements théoriques[2]

### 2.3.1 Historique

Le concept moderne d'apprentissage par renforcement est né à la fin des années 1980 de la rencontre entre deux domaines de recherche. L'un s'occupait de l'étude de la psychologie animale, l'autre des problèmes de commande optimale.

Le terme « commande optimale » apparaît à la fin des années 1950 pour décrire le problème de la conception de contrôleurs capables de minimiser un critère donné. Une approche de ce problème a été développée par Richard Bellman et ses collègues en prolongeant la théorie d'Hamilton et Jacobi datant du 19ème siècle. Cette approche utilise le concept d'état du système dynamique et de fonction de valeur pour définir une équation fonctionnelle à résoudre, appelée *équation de Bellman*. En 1957, Richard Bellman (1957b) introduit la version stochastique discrète du problème appelée *Processus de Décision Markovien*[3] (PDM) qui est le cadre habituel des algorithmes d'apprentissage par renforcement. Il propose une méthode de résolution par *itérations sur les valeurs*[4] (Ri-

1. Ensemble dans lequel un même élément peut apparaître plusieurs fois. Autrement dit, la multiplicité d'un élément d'un multi-ensemble peut être supérieure à 1.

2. Cette section s'inspire largement des ouvrages de référence de Richard Sutton et Andrew Barto (1998) et de L. Kaelbling, M. Littman et A. Moore (1996) ainsi que du cours de Patrick Fabiani, Jean-Loup Farges et Frédérick Garcia (2001).

3. Termes anglais : Markovian Decision Process (MDP).

4. Termes anglais : value iteration method.

chard Bellman 1957a). La classe des méthodes de résolution des problèmes de contrôles optimaux est alors nommée par Richard Bellman *programmation dynamique* [5]. En 1960, Ronald Howard publie une méthode de résolution par *itérations de stratégies* [6]. Ces méthodes sont à la base des algorithmes d'apprentissage par renforcement.

Les autres sources d'inspiration furent les théories empiriques sur l'apprentissage animal. Le premier à exprimer l'idée d'une sélection du comportement d'un animal en le récompensant ou non fut sans doute Edward Thorndike (1911). Il appelait cette méthode la *loi de l'effet* [7] qui peut être résumée ainsi : « tout comportement renforcé positivement a tendance à se reproduire dans la même situation ». Bien que parfois controversée, cette méthode est largement considérée comme une idée fondatrice de l'apprentissage par essais et erreurs. La loi de l'effet sous-entend deux principes de l'apprentissage par essais et erreurs : la sélectivité et l'associativité. La sélectivité permet la recherche d'alternatives et de solutions nouvelles, tandis que l'associativité permet d'associer chaque solution trouvée à une situation particulière. En d'autres termes, la loi de l'effet, comme l'apprentissage par essais et erreurs implique recherche et mémorisation.

L'idée de programmer un ordinateur pour apprendre une tâche par essais et erreurs date des premières spéculations sur l'intelligence artificielle (Allan Turing 1950). Dans les années 60, de nombreux chercheurs travaillent sur l'apprentissage par essais et erreurs appliqué à des problèmes de contrôle. Les travaux les plus influents sont ceux de Donald Michie (1961) qui décrit un système capable d'apprendre à jouer au morpion. Puis avec R. Chambers, ils proposent un autre système capable d'apprendre à maintenir un pendule inversé (D. Michie et R. Chambers 1968), exemple désormais classique de l'apprentissage par renforcement. Durant ces années, le terme d'apprentissage par renforcement apparaît dans le domaine des sciences de l'ingénieur.

Parallèlement à ces recherches, Arthur Samuel (1959) propose et implémente une méthode d'apprentissage appliquée à un jeu de dames utilisant une notion de différence temporelle. Cette notion, utilisée par les algorithmes modernes, introduit l'idée d'apprendre progressivement en utilisant la différence entre les estimations successives d'une même quantité (comme par exemple la probabilité d'obtenir une récompense).

Dans les années 70, Harry Klopf (1972, 1975) rapproche les idées d'apprentissage par essais et erreurs et de différence temporelle. Sur ces bases, Andrew Barto, Richard Sutton et Charles Anderson (1983) poursuivent les travaux de Harry Klops et développent la première méthode d'apprentissage par renforcement connue sous le nom d'*AHC-learning* [8] qu'ils appliquent au problème du pendule inversé de Michie et Chambers. Cette méthode qui utilise la différence temporelle et l'apprentissage par essais et erreurs est en fait une version sans modèle de l'algorithme par itérations de stratégies. Elle a été largement étudiée par Richard Sutton (1984) durant son doctorat.

Enfin, une étape importante est franchie quand Richard Sutton (1988) et Christopher Watkins (1989) publient à un an d'intervalle les algorithmes du *TD($\lambda$)* et du *Q-Learning*. Ces algorithmes rassemblent la différence temporelle et l'apprentissage par essais et er-

5. Termes anglais : dynamic programming.
6. Termes anglais : policy iteration method.
7. Termes anglais : law of effect.
8. Termes anglais : Adaptive Heuristic Critic learning ou actor-critic.

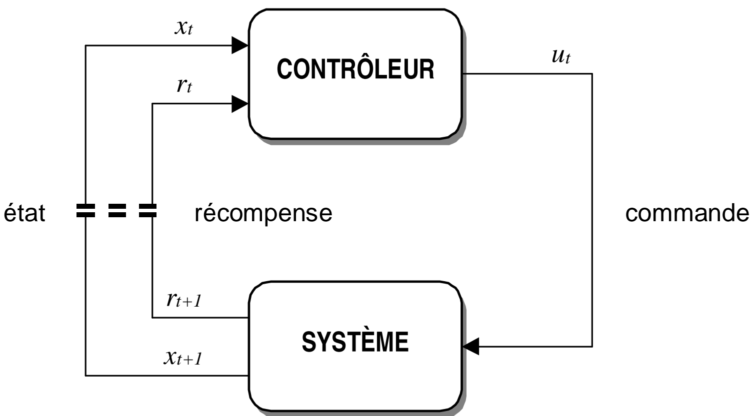



FIG. 2.1 – *Boucle sensori-motrice.*

reurs et utilisent la théorie de la commande optimale par *itérations sur les valeurs.* Leur convergence vers la stratégie de commande optimale est établie (Richard Sutton 1988, Peter Dayan 1992, Christopher Watkins et Peter Dayan 1992, Tommi Jaakkola, Michael Jordan et Satinder Singh 1994).

### 2.3.2 Boucle sensori-motrice

Les algorithmes d'apprentissage par renforcement se situent dans le cadre classique d'une boucle sensori-motrice échantillonnée (*cf.* figure 2.1).

Deux entités interagissent entre elles. Le *contrôleur* [9] perçoit des informations sur le *système* [10] ainsi qu'un *signal de récompense.*

L'*état* résume la situation du système à chaque instant. Le signal de récompense est un nombre réel.

A chaque instant, le contrôleur génère une *commande* [11] qui agit sur le système.

### 2.3.3 Processus de décision markovien

Comme nous l'avons vu dans l'historique, la plupart des algorithmes d'apprentissage par renforcement se placent dans le cadre des processus de décision markoviens discrets.

Un processus de décision markovien discret est défini par :

- un axe temporel discret fini ou infini (on parle alors d'horizon fini ou infini),
- un ensemble fini d'états $\mathcal{X}$,
- un ensemble fini de commandes $\mathcal{U}$,
- les probabilités $\mathsf{p}$ de transition entre états,
- une fonction $\mathsf{r}$ associant une récompense immédiate à chaque transition.

9. Le terme *agent* est aussi employé.
10. Ou *environnement.*
11. Le terme *action* est souvent utilisé pour les algorithmes d'apprentissage par renforcement.

Dans le cas général, l'ensemble des états peut dépendre de l'instant $t$. De même, l'ensemble des commandes peut dépendre du temps et de l'état courant. Nous nous limiterons dans cette étude au cas classique où $\mathcal{X}$ et $\mathcal{U}$ sont constants.

Les probabilités $\mathsf{p}$ de transition décrivent la dynamique du système. $\mathsf{p}_t(x' \mid x, u)$ est la probabilité de passer dans l'état $x'$ après avoir effectué la commande $u$ dans l'état $x$ à l'instant $t$. On impose couramment :

$$\forall t \, \forall x \, \forall u, \sum_{\forall x' \in \mathcal{X}} \mathsf{p}_t(x' \mid x, u) = 1 \tag{2.1}$$

Les probabilités $\mathsf{p}_t$ de transition vérifient la propriété fondamentale de Markov. La probabilité d'atteindre un nouvel état $x_{t+1}$ après avoir effectué une commande $u_t$ ne dépend que de l'état précédent $x_t$ et de la commande $u_t$, et ne dépend pas des états et commandes précédents. En d'autres termes, si on note $h_t$ l'historique à l'instant $t$ du processus, soit $h_t = (x_0, u_0, \ldots, x_{t-1}, u_{t-1}, x_t, u_t)$, $\mathsf{p}_t$ vérifie la propriété suivante :

$$\forall t \, \forall h_t \, \forall x_{t+1}, \; \mathsf{p}_t(x_{t+1} \mid h_t) = \mathsf{p}_t(x_{t+1} \mid x_t, u_t) \tag{2.2}$$

Le contrôleur reçoit à chaque instant $t$ un signal $\mathsf{r}_t(x,u,x') \in \mathbb{R}$ de récompense immédiate qui qualifie la dernière transition définie par l'état précédent, la commande effectuée et le nouvel état $x'$. La fonction de récompense utilise parfois uniquement l'état précédent et la commande. Si ce signal est positif, on parle de *gain* ou simplement de récompense, s'il est négatif on parle de *coût* ou de *punition*.

Enfin, un processus de décision markovien est dit *stationnaire* si les probabilités de transition et la fonction de récompense ne dépendent pas du temps. Dans le cas contraire, on suppose généralement que la dérive de stationnarité est très lente (beaucoup plus lente que la vitesse d'apprentissage).

**Discussion :** Ces critères peuvent être facilement réunis dans le cas d'une application par simulation. En revanche, dans le cas réel, il est difficile d'obtenir une perception complète de l'état, ainsi, la propriété de Markov n'est plus forcément vérifiée.

Enfin, limiter le signal de récompense à un simple réel est réducteur. Dans un système complexe, il peut être difficile d'attribuer une valeur à une transition provoquant simultanément un succès et un échec. Ce point sera discuté plus largement dans la section 4.3.3.

### 2.3.4 Objectifs

Le but de la commande optimale et des algorithmes d'apprentissage par renforcement est de trouver une *stratégie de commande* [12] qui maximise le *gain* ou *somme des récompenses futures*. Une stratégie de commande est une fonction notée $\pi$ qui associe une commande à chaque état $x$, telle que :

$$\pi \, : \, x \in \mathcal{X} \longrightarrow \pi(x) \in \mathcal{U} \tag{2.3}$$

12. Une stratégie de commande est aussi appelée *contrôleur*, mais cette terminologie reste peu employée en apprentissage par renforcement.

La somme des récompenses futures peut être pondérée ou non. Le cas classique utilise une pondération par un *coefficient d'atténuation*[13] $\gamma$ qui réduit l'influence du futur lointain par rapport au futur proche. Dans ce cas, on parle de *critère $\gamma$-pondéré* et le gain s'écrit :

$$\mathsf{G}(t) = \sum_{k=0}^{\infty} \gamma^k \mathsf{r}_{t+k}(x_{t+k}, u_{t+k}, x_{t+k+1}) \tag{2.4}$$

avec $0 \leq \gamma < 1$.

Notons qu'il existe d'autres critères que nous n'utilisons pas dans cette thèse, notamment le *critère fini*, le *critère total* et le *critère moyen*.

Pour trouver une stratégie de commande qui maximise le gain, on utilise une *fonction de valeur de l'action*[14] ou *fonction d'utilité*[15] notée $\mathsf{Q}^\pi(x,u)$ qui correspond à l'espérance de gain d'effectuer la commande $u$ dans l'état $x$ en suivant la stratégie $\pi$, soit :

$$\mathsf{Q}^\pi(x,u) = E_\pi\{\mathsf{G}(t) \,|\, x_t = x, u_t = u\} \tag{2.5}$$

où $\mathsf{G}(t)$ est défini par (2.4).

Les fonctions d'utilité définissent un ordre partiel sur les stratégies. Une stratégie de commande $\pi$ est supérieure à une autre stratégie $\pi'$ si $\mathsf{Q}^\pi(x,u) \geq \mathsf{Q}^{\pi'}(x,u)$ pour tous les couples *état—commande* de $\mathcal{X} \times \mathcal{U}$. La stratégie de commande optimale $\pi^*$ vérifie donc :

$$\forall \pi \quad \forall (x,u) \in \mathcal{X} \times \mathcal{U} \quad \mathsf{Q}^{\pi^*}(x,u) \geq \mathsf{Q}^\pi(x,u) \tag{2.6}$$

On définit alors la fonction d'utilité optimale $\mathsf{Q}^*$ telle que :

$$\forall (x,u) \in \mathcal{X} \times \mathcal{U} \quad \mathsf{Q}^*(x,u) = \max_\pi \mathsf{Q}^\pi(x,u) \tag{2.7}$$

Si $\mathcal{X}$ et $\mathcal{U}$ sont finis et $\gamma < 1$, Richard Bellman a montré que la fonction d'utilité optimale $\mathsf{Q}^*$ est l'unique solution de l'équation suivante :

$$\mathsf{Q}^*(x,u) = \sum_{\forall x' \in \mathcal{X}} \mathsf{p}(x' \,|\, x, u)\big[\mathsf{r}(x,u,x') + \gamma \max_{v \in \mathcal{U}} \mathsf{Q}^*(x',v)\big] \tag{2.8}$$

Cette équation est connue sous le nom d'*équation d'optimalité de Bellman*. Elle traduit l'idée que quel que soit l'état initial $x$, la première commande d'une stratégie optimale mène à un nouvel état $x'$ à partir duquel l'évolution ultérieure est optimale.

Quand on connaît la fonction d'utilité optimale $\mathsf{Q}^*$, une stratégie optimale qui fournit la commande optimale $\pi^*(x)$ pour tout état $x$ est alors définie par :

$$\pi^*(x) = \arg\max_{v \in \mathcal{U}} \mathsf{Q}^*(x,v) \tag{2.9}$$

Trouver une stratégie de commande optimale $\pi^*$ revient donc à déterminer la fonction d'utilité optimale $\mathsf{Q}^*$.

13. Termes anglais : discount-rate parameter.

14. Nous avons délibérément choisi d'utiliser une fonction de valeur par rapport à un couple *état—commande* plutôt que la fonction de valeur traditionnelle par rapport à un état seul (souvent notée $V(x)$). Ce choix est motivé par un souci de clarté et de cohérence avec la présentation du Q-Learning qui suit et qui utilise cette même fonction.

15. Ou encore fonction de qualité, termes anglais : action-value function.

**Début**
  Initialiser $\mathsf{Q}(x,u)$ arbitrairement pour tout $(x,u)$ de $\mathcal{X} \times \mathcal{U}$
  **Répéter**
    $\delta \leftarrow 0$
    **Pour tout** $(x,u)$ de $\mathcal{X} \times \mathcal{U}$ **faire**
      $\xi \leftarrow \mathsf{Q}(x,u)$
      $\mathsf{Q}(x,u) \leftarrow \sum_{\forall x' \in \mathcal{X}} \mathsf{p}(x' \mid x,u)\left[\mathsf{r}(x,u,x') + \gamma \max_{v \in \mathcal{U}} \mathsf{Q}(x',v)\right]$
      $\delta \leftarrow \max\{\delta, |\xi - \mathsf{Q}(x,u)|\}$
    **Fin pour**
  **Jusqu'à** $\delta < \beta$
**Fin**

FIG. 2.2 – *Algorithme d'itérations sur les valeurs.*

### 2.3.5 Programmation dynamique

Dans le cadre de la programmation dynamique, on suppose connu le modèle du système décrit par les probabilités $\mathsf{p}$ de transition et par la fonction de récompense $\mathsf{r}$. Ces hypothèses ne sont pas vérifiées dans le cas de l'apprentissage par renforcement, mais l'étude de l'algorithme par itérations sur les valeurs aide à comprendre les algorithmes d'apprentissage par renforcement.

Si on connaît le modèle, l'algorithme par itérations sur les valeurs décrit par la figure 2.2 permet de calculer $\mathsf{Q}^*$. Il est fondé sur le *théorème du point fixe de Banach* qui assure la convergence d'une suite vers l'unique fonction solution de l'équation de Bellman. L'algorithme réalise donc des approximations successives jusqu'à obtenir une bonne stratégie de commande à l'aide de l'équation de Bellman, soit :

$$\mathsf{Q}_t(x,u) = \sum_{\forall x' \in \mathcal{X}} \mathsf{p}(x' \mid x,u)\left[\mathsf{r}(x,u,x') + \gamma \max_{v \in \mathcal{U}} \mathsf{Q}_{t-1}(x',v)\right] \tag{2.10}$$

Plusieurs critères d'arrêt peuvent être envisagés. Le critère d'arrêt le plus classique est défini par un seuil positif $\beta$ : les itérations sont stoppées si la plus grande différence entre deux approximations successives est inférieure à ce seuil. La convergence de l'algorithme vers la fonction d'utilité optimale est garantie en un nombre polynômial d'itérations fonction de $|\mathcal{X}|$, $|\mathcal{U}|$, $1/(1-\gamma)log(1/(1-\gamma))$ et du nombre de bits qui code les valeurs $\mathsf{p}$ et $\mathsf{r}$, chaque itération étant de complexité $O(|\mathcal{U}|\,|\mathcal{X}|^2)$ (M. Littman, T. Dean et L. Kaebling 1995).

Il existe de nombreux autres algorithmes de programmation dynamique qui améliorent notamment la vitesse de convergence. Nous ne développerons pas ici ces algorithmes.

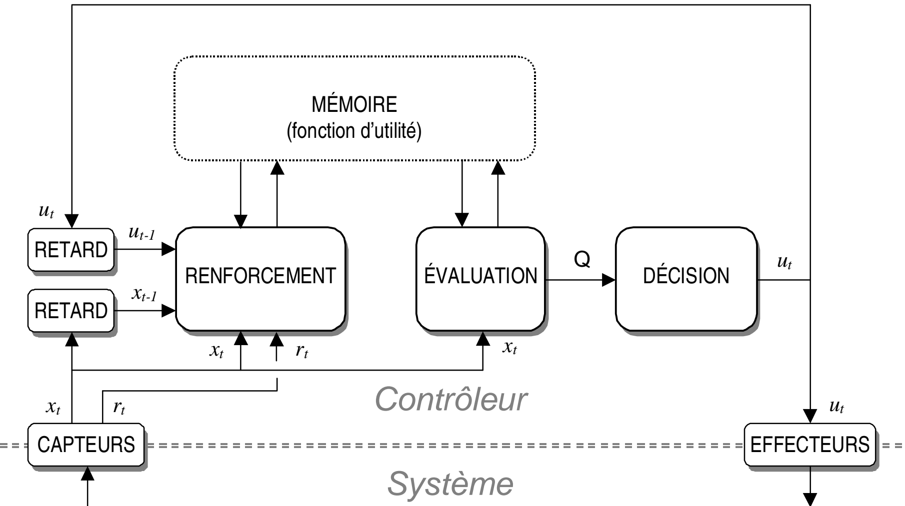



FIG. 2.3 – *Schéma fonctionnel des méthodes directes.*

### 2.3.6 Apprentissage par renforcement

La différence fondamentale entre la programmation dynamique et l'apprentissage par renforcement est la connaissance ou non du modèle du système. Dans le cadre de l'apprentissage par renforcement, on suppose les probabilités p de transitions et la fonction r de récompense inconnues.

L'idée de base des algorithmes d'apprentissage est d'utiliser l'expérience ou, autrement dit, les parcours effectués dans l'espace d'états pour améliorer la stratégie de commande.

Richard Sutton et Andrew Barto (1998, p. 254) distinguent deux classes de méthodes d'apprentissage. Les méthodes *directes* [16] optimisent directement la fonction d'utilité en expérimentant les états du système. Les méthodes directes les plus utilisées sont le *Q-Learning* et le *TD(λ)*. Les méthodes *indirectes* [17] construisent et mémorisent un modèle au fur et à mesure des états visités puis optimisent à l'aide de ce modèle partiel la fonction d'utilité. Les méthodes indirectes les plus connues sont le *Dyna-Q* et le *priority-sweeping*.

## 2.4 Méthodes directes

Contrairement à la programmation dynamique, les algorithmes d'apprentissage par renforcement sont exécutés en temps réel au sein de la boucle sensori-motrice.

Le fonctionnement des algorithmes directs peut être synthétisé par le schéma fonctionnel présenté en figure 2.3. Le contrôleur est constitué de quatre fonctions principales. Une *mémoire* stocke les informations nécessaires, comme par exemple la fonction d'utilité dans le cas du Q-Learning. La fonction de *renforcement* met à jour le contenu de la

16. Ou méthodes sans modèle (termes anglais : model-free).
17. Ou méthodes utilisant un modèle (termes anglais : model-based).

mémoire avec les informations disponibles à l'instant $t$ . La fonction d'*évaluation* extrait de la mémoire une estimation de l'utilité de chaque commande dans l'état actuel. Enfin, la fonction de décision choisit la commande à effectuer en fonction de ces estimations.

A chaque instant $t$, le contrôleur dispose de l'état précédent $x_{t-1}$, de la commande précédente $u_{t-1}$, du nouvel état $x_t$ et de la récompense $r_t$. Le principe des méthodes directes est d'utiliser ces informations pour mettre à jour la fonction d'utilité courante. Progressivement, si le système est conduit à visiter tous ses états, alors la fonction d'utilité sera optimisée dans tout l'espace d'états.

Il existe plusieurs méthodes directes. Nous allons étudier en détail la plus connue, le Q-Learning, puis deux autres algorithmes appartenant à la classe des méthodes appelée TD($\lambda$).

### 2.4.1 Q-Learning

L'algorithme du Q-Learning a été introduit par Christopher Watkins (1989). Il est sans doute l'algorithme d'apprentissage le plus utilisé. Ce succès s'explique par plusieurs atouts. D'une part, cet algorithme est très simple et très ouvert. D'autre part, sa convergence a été prouvée dans le cas des processus de décision markoviens.

#### Fonction mémoire

Le Q-Learning ne stocke que la fonction d'utilité courante $\mathsf{Q}$. A l'origine, la mémoire utilisait implicitement un tableau (statique ou dynamique) de dimension $|\mathcal{X}| \times |\mathcal{U}|$ pour stocker la fonction d'utilité.

L'emploi de cette structure est requis pour prouver la convergence du Q-Learning. Malgré tout, il est possible d'utiliser à la place d'un tableau une fonction d'approximation. Ce point sera exposé dans la section 2.6.

#### Fonction de renforcement

La fonction de renforcement du Q-Learning est fondée sur l'algorithme d'itérations sur les valeurs. Comme nous l'avons vu précédemment, l'objectif de cet algorithme est d'optimiser la fonction d'utilité pour chaque couple *état—commande* $(x,u)$ afin d'en déduire une stratégie de commande optimale.

Dans ce but, le Q-Learning utilise le principe de la *différence temporelle* pour remettre la fonction d'utilité à jour. La différence temporelle correspond à la différence entre deux estimations successives de l'espérance de gain d'un couple *état—commande*.

A chaque période d'échantillonnage, on dispose des informations courantes $x_{t-1}$, $u_{t-1}$, $x_t$ et $r_t = \mathsf{r}(x_{t-1},u_{t-1},x_t)$. Le Q-Learning met à jour la fonction d'utilité en utilisant ces informations selon l'équation suivante :

$$\mathsf{Q}_t(x_{t-1},u_{t-1}) = \mathsf{Q}_{t-1}(x_{t-1},u_{t-1}) + \alpha \left[ r_t + \gamma \max_{v \in \mathcal{U}} \mathsf{Q}_{t-1}(x_t,v) - \mathsf{Q}_{t-1}(x_{t-1},u_{t-1}) \right] \tag{2.11}$$

avec $0 < \alpha \leq 1$. $\alpha$ est appelé le *taux d'apprentissage* [18]. En théorie, $\alpha$ doit décroître à chaque nouveau passage par l'état $x = x_{t-1}$ et la commande $u = u_{t-1}$. En pratique, pour éviter de stocker des informations supplémentaires ou pour que l'apprentissage soit permanent (par exemple pour s'adapter à un changement de dynamique du système), on utilise un coefficient fixe. La valeur la plus courante est 0,1.

Dans cette équation, le terme $r_t + \gamma \max_{v \in \mathcal{U}} \mathsf{Q}_{t-1}(x_t,v) - \mathsf{Q}_{t-1}(x_{t-1},u_{t-1})$ correspond à la différence entre la nouvelle et l'ancienne estimation de $\mathsf{Q}(x_{t-1},u_{t-1})$. Cette différence est appelée *différence temporelle*.

### Fonction d'évaluation

Le but de la fonction d'évaluation est de faire le lien entre la mémoire et la fonction de décision. Dans le cas du Q-Learning, il s'agit simplement de fournir à la fonction de décision l'estimation courante de l'espérance de gain de chaque commande $v$ de $\mathcal{U}$ pour l'état actuel du système $x_t$, soit $\mathsf{Q}(x_t,v)$.

Nous avons introduit cette fonction uniquement pour faire le lien entre le Q-Learning classique et notre approche parallèle présentée au chapitre 4 et dont l'étape d'évaluation n'est plus du tout triviale.

### Fonction de décision : dilemme exploration—exploitation

A chaque instant $t$, le contrôleur doit choisir la commande à effectuer. Si cette commande est choisie au hasard uniformément dans $\mathcal{U}$, le contrôleur va faire visiter tous les états au système et on parle alors d'*exploration*. Cette exploration de l'espace d'états satisfait le critère de convergence de Q-Learning (voir paragraphe suivant), l'apprentissage se fait sûrement mais très lentement. La fonction d'utilité va être remise à jour aussi souvent qu'il s'agisse ou non d'états intéressants au point de vue des récompenses. Enfin, la somme des récompenses accumulées au cours de l'apprentissage est faible car la stratégie suivie est stochastique.

Inversement, le choix à chaque itération de la commande $u_t$ correspondant à la stratégie optimale courante définie par $\pi_t(x_t) = \arg\max_{v \in \mathcal{U}} \mathsf{Q}(x_t,v)$ est appelé *exploitation* car cette décision cherche à obtenir le maximum de récompenses avec les connaissances actuelles. Cette commande est appelée *commande gourmande* [19, 20]. L'exploitation n'est pas non plus satisfaisante car elle conduit généralement soit à une stratégie sous-optimale (maximum local), soit à la divergence de l'algorithme.

Il est donc nécessaire de réaliser un compromis entre exploitation et exploration. Généralement, les implémentations du Q-Learning retiennent le principe qui consiste à choisir la commande gourmande la plupart du temps, tout en choisissant plus ou moins régulièrement une commande aléatoire. De nombreux heuristiques existent. On les classe en deux catégories : les méthodes dirigées et les méthodes non dirigées. Les méthodes

18. Termes anglais : Learning rate parameter.
19. Le terme *glouton* est aussi employé, terme anglais : greedy-action.
20. En cas d'égalité de la valeur d'utilité $\mathsf{Q}(x_t,v)$ des commandes $v$ qui maximisent $\mathsf{Q}(x_t,v)$, la commande gourmande est choisie au hasard parmi les ex æquo.

**Début**
Initialiser $\mathsf{Q}(x,u)$ arbitrairement pour tout $(x,u)$ de $\mathcal{X} \times \mathcal{U}$
Initialiser l'état initial $x$
**Tant que** $x$ n'est pas un état absorbant **faire**
$u \leftarrow$Décision$(\mathsf{Q},x)$
Exécuter $u$ et observer le nouvel état $x'$ et la récompense $r$
$\mathsf{Q}(x,u) \leftarrow \mathsf{Q}(x,u) + \alpha[r + \gamma \max_{v \in \mathcal{U}} \mathsf{Q}(x',v) - \mathsf{Q}(x,u)]$
$x \leftarrow x'$
**Fin tant que**
**Fin**

FIG. 2.4 – *Algorithme du Q-Learning.*

non dirigées utilisent uniquement les valeurs de la fonction d'utilité $\mathsf{Q}$ pour choisir la commande (Sebastian Thrun 1992, L. Kaelbling, M. Littman et A. Moore 1996). Les méthodes dirigées utilisent des informations supplémentaires comme le nombre de fois où telle commande a déjà été effectuée pour donner un bonus d'exploration à certaines commandes (Nicolas Meuleau 1996, Leslie Kaelbling 1993b).

Deux méthodes non dirigées sont couramment employées :

– l'$\epsilon$-gourmand[21] choisit la commande gourmande avec une probabilité $1 - \epsilon$ et tire une commande au hasard avec une probabilité $\epsilon$,
– le tirage selon une distribution de Boltzmann associe une probabilité de sélection à chaque commande, telle que :

$$P(u|x) = \frac{e^{\mathsf{Q}(x,u)/\tau}}{\sum_v e^{\mathsf{Q}(x,v)/\tau}} \tag{2.12}$$

Dans ces méthodes, les paramètres $\epsilon$ et $\tau$ permettent de déterminer le niveau de l'exploration.

Bien que ce sujet soit intéressant, nous ne le développons pas plus car cet aspect de l'apprentissage n'est pas l'objet de cette thèse. Dans la suite, nous utilisons toujours la fonction de décision par $\epsilon$-gourmand.

### Convergence

La figure 2.4 présente l'algorithme résumant le fonctionnement du Q-Learning. La convergence de cet algorithme a été bien étudiée et est maintenant établie (Christopher Watkins et Peter Dayan 1992, Tommi Jaakkola, Michael Jordan et Satinder Singh 1994). La suite $\mathsf{Q}_n$ converge vers $\mathsf{Q}^*$ avec une probabilité égale à 1 si :

– $\mathcal{X}$ et $\mathcal{U}$ sont finis,
– la fonction $\mathsf{Q}$ est un tableau,

21. Ou $\epsilon$-glouton, terme anglais $\epsilon$-greedy.

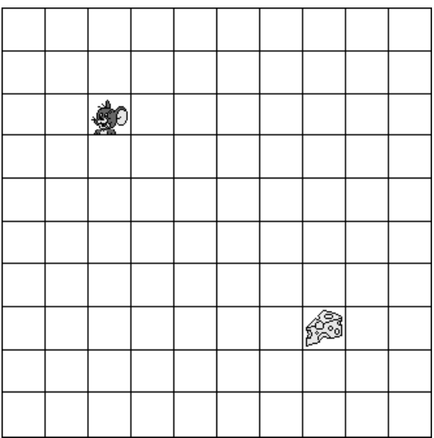


FIG. 2.5 – *Le labyrinthe 10×10 en configuration initiale.*

- chaque couple $(x,u) \in \mathcal{X} \times \mathcal{U}$ est visité un nombre infini de fois,
- $\sum_n \alpha_n(x,u) = \infty$ et $\sum_n \alpha_n^2(x,u) < \infty$,
- $\gamma < 1$ (ou $\gamma = 1$ et s'il existe un état absorbant à récompense nulle).

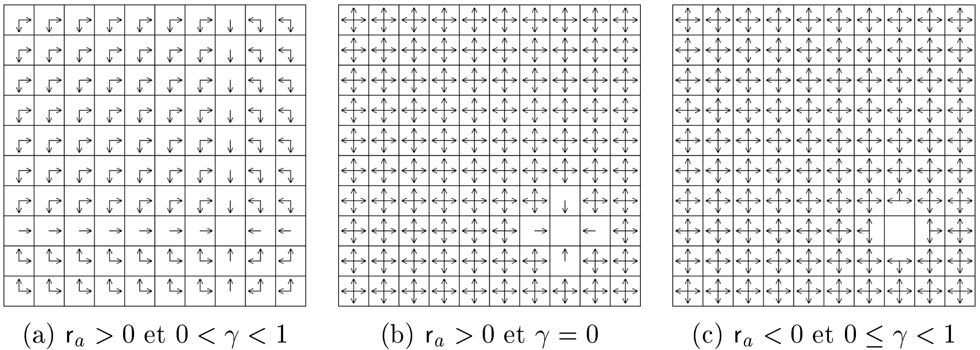
**Discussion :** Ces hypothèses sont assez fortes. Dans le cas réel, il est souvent trop long de parcourir l'espace d'états de nombreuses fois. Mais même si certaines hypothèses ne sont pas complètement vérifiées, on observe que l'algorithme se comporte de manière très robuste et converge vers des stratégies quasi-optimales.

### 2.4.2 Étude comportementale du Q-Learning

Afin de mieux cerner le fonctionnement de l'algorithme du Q-Learning, nous l'avons testé sur l'exemple simple du *labyrinthe.* Ce système est représenté par une souris évoluant sur un damier de $10 \times 10$ cases (*cf.* figure 2.5). A chaque position de la souris correspond un état. Le contrôleur peut déplacer la souris d'une case dans les quatre directions avec quatre commandes : haut, bas, droite et gauche. Le système est déterministe.

La position initiale de la souris est la case (3,3). La case (8,8) est un état absorbant noté $x_a$. Si la souris atteint cette case, l'épisode est terminé et la souris est replacée sur sa case initiale. Le chemin le plus court entre l'état initial et l'état absorbant mesure 10 cases.

La stratégie d'exploration—exploitation est l'$\epsilon$-gourmand avec $\epsilon = 0{,}1$. La fonction $\mathsf{Q}$ est initialisée à une valeur réelle $\mathsf{Q}_i$.



(a) $\mathsf{r}_a > 0$ et $0 < \gamma < 1$ (b) $\mathsf{r}_a > 0$ et $\gamma = 0$ (c) $\mathsf{r}_a < 0$ et $0 \leq \gamma < 1$

FIG. 2.6 – *Stratégies optimales pour différentes valeurs de* $\mathsf{r}_a$ *et* $\gamma$ *(les flèches représentent les commandes optimales).*

### Cas des récompenses locales

Dans ce paragraphe, seules les transitions qui mènent à l'état absorbant donnent une récompense non nulle. On pose pour tous $x$ de $\mathcal{X}$ et $u$ de $\mathcal{U}$ :

$$\mathsf{r}(x,u,x') = \begin{cases} \mathsf{r}_a & \text{si } x' = x_a \\ 0 & \text{sinon} \end{cases} \tag{2.13}$$

**Comportement avec $\mathsf{r}_a > 0$ et $0 < \gamma < 1$ :** la stratégie optimale tend à rapprocher la souris de l'état absorbant (*cf.* figure 2.6a). L'apprentissage converge rapidement vers le trajet le plus court entre l'état initial et l'état absorbant : les commandes fournies par l'algorithme conduisent à des trajectoires proches de 10 pas. La figure 2.7 montre l'évolution du nombre de pas par épisode en fonction du nombre d'épisodes. Un épisode correspond à une simulation entre l'instant où le système se trouve dans l'état initial et l'instant où le système se trouve dans l'état absorbant. Le nombre de pas est le nombre de périodes d'échantillonnage entre ces deux instants ou encore le nombre de cycles qu'a effectué l'algorithme. La courbe montre qu'après 30 épisodes, la trajectoire est quasi optimale (à $\epsilon$ près). Il faut environ 2 500 pas cumulés pour apprendre la stratégie optimale.

**Comportement avec $\mathsf{r}_a > 0$ et $\gamma = 0$ :** l'effet de la récompense est local. En effet, l'utilité optimale $\mathsf{Q}(x,u)$ d'un couple *état—commande* $(x,u)$ solution de l'équation (2.10) de Bellman s'écrit alors :

$$\mathsf{Q}^*(x,u) = \sum_{\forall x' \in \mathcal{X}} \mathsf{p}(x' \mid x, u)\mathsf{r}(x,u,x') \tag{2.14}$$

et ne dépend donc que de la première récompense et pas des récompenses futures.

Cette stratégie optimale contraint uniquement les commandes des états voisins de l'état absorbant (*cf.* figure 2.6b). Ce comportement permet d'imposer une contrainte locale. Si le système passe par un état voisin de l'état absorbant, il est attiré. Dans les autres états, il est libre de toute influence, la commande est aléatoire.

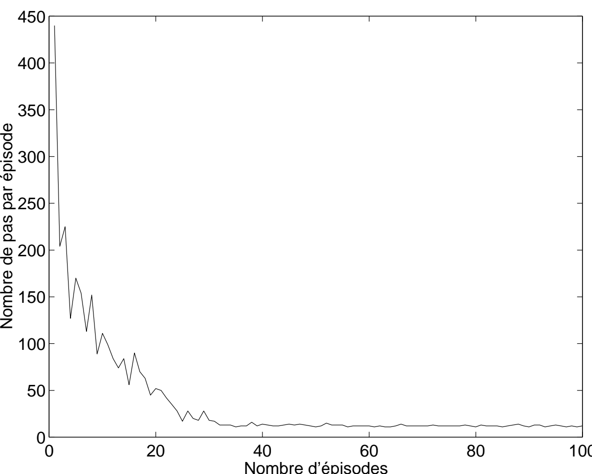



Fig. 2.7 – *Courbe d'apprentissage du Q-Learning sur l'exemple du labyrinthe 10×10 (moyenne sur 20 simulations de l'évolution du nombre de pas par épisode en fonction du nombre d'épisodes pour* $\mathsf{r}_a = 0{,}1$*,* $\gamma = 0{,}9$*,* $\alpha = 0{,}1$*,* $\epsilon = 0{,}1$ *et* $\mathsf{Q}_i = 0$*).*

**Comportement avec $\mathsf{r}_a < 0$ :** dans ce cas, quelle que soit la valeur de $\gamma$, la stratégie optimale est la même. Le système évite les transitions associées à la récompense négative (*cf.* figure 2.6c). Ce comportement est antagoniste au cas $\mathsf{r}_a > 0$ et $\gamma = 0$, il impose d'éviter localement certaines transitions.

**Comportement en début d'apprentissage :** au début de l'apprentissage, le signe de la valeur initiale $\mathsf{Q}_i$ de la fonction d'utilité influence le début de l'apprentissage. En effet, si on calcule la première mise à jour d'un couple *état—commande* $(x,u)$ avec l'équation (2.11) tel que l'état suivant $x'$ ne soit pas l'état absorbant et n'ait pas été mis à jour, on obtient :

$$\begin{aligned} \mathsf{Q}(x,u) &\leftarrow \mathsf{Q}(x,u) + \alpha[\gamma \max_{v \in \mathcal{U}} \mathsf{Q}(x',v) - \mathsf{Q}(x,u)] \\ &\leftarrow \mathsf{Q}_i + \alpha(\gamma \max_{v \in \mathcal{U}} \mathsf{Q}_i - \mathsf{Q}_i) \\ &\leftarrow \mathsf{Q}_i - \alpha(1-\gamma)\mathsf{Q}_i \end{aligned} \tag{2.15}$$

Ainsi, tant que le système n'a pas atteint beaucoup de fois l'état absorbant, les valeurs de la fonction d'utilité vont tendre vers zéro.

Si $\mathsf{Q}_i > 0$, les états déjà visités auront une valeur d'utilité inférieure à celles des états non visités ($\mathsf{Q}(x,u) < \mathsf{Q}_i$). Les états non visités seront donc plus attirants, ce qui induit une exploration au début de l'apprentissage. Cette exploration induite est plus *systématique* que l'exploration aléatoire. Elle est très utilisée pour accélérer l'apprentissage au départ.

Si $\mathsf{Q}_i < 0$, les états déjà visités auront une valeur d'utilité supérieure à celles des états non visités ($\mathsf{Q}(x,u) > \mathsf{Q}_i$). Les états visités seront donc plus attirants, ce qui induit

une moins grande exploration au début de l'apprentissage. Ce comportement, que nous nommons *piétinement*, ralentit considérablement l'apprentissage, il est donc préférable de l'éviter.

Enfin, si $\mathsf{Q}_i = 0$, les états déjà visités auront une valeur d'utilité identique à celles des états non visités ($\mathsf{Q}(x,u) = \mathsf{Q}_i$). Le comportement sera pleinement aléatoire au début de l'apprentissage.

Par ailleurs, si on calcule la première mise à jour dans le cas où l'état suivant est l'état absorbant, on obtient :

$$\mathsf{Q}(x,u) \leftarrow \mathsf{Q}_i - \alpha[(1-\gamma)\mathsf{Q}_i - \mathsf{r}_a] \tag{2.16}$$

Si $(1-\gamma)\mathsf{Q}_i > \mathsf{r}_a$, la transition menant à l'état absorbant aura une valeur d'utilité inférieure à celles des états non visités avoisinants ($\mathsf{Q}(x,u) < \mathsf{Q}_i$) et, pendant une certaine durée, le contrôleur évitera cette transition.

Si $(1-\gamma)\mathsf{Q}_i < \mathsf{r}_a$, la transition menant à l'état absorbant aura une valeur d'utilité supérieure à celles des états non visités avoisinants ($\mathsf{Q}(x,u) > \mathsf{Q}_i$) et le contrôleur sera attiré par cette transition.

Enfin, le cas $(1-\gamma)\mathsf{Q}_i = \mathsf{r}_a$ n'induit aucune contrainte temporaire.

Dans tous les cas, mises à part ces conditions, la valeur de $\mathsf{Q}_i$ n'a pas d'importance car seuls comptent les écarts relatifs des valeurs d'utilité.

### Cas des systèmes à deux récompenses

Dans ce paragraphe, nous étudions le cas plus général où les transitions qui mènent à l'état absorbant ont une récompense $\mathsf{r}_a$ et les autres transitions $\mathsf{r}_i$. On pose pour tous $x$ de $\mathcal{X}$ et $u$ de $\mathcal{U}$ :

$$\mathsf{r}(x,u,x') = \begin{cases} \mathsf{r}_a & \text{si } x' = x_a \\ \mathsf{r}_i & \text{sinon} \end{cases} \tag{2.17}$$

Dans ce cas, on observe les mêmes comportements que précédemment, mais la valeur discriminante de $\mathsf{r}_a$ n'est plus 0, mais $\mathsf{Q}_\infty$ définie par :

$$\mathsf{Q}_\infty = \frac{\mathsf{r}_i}{1-\gamma} \tag{2.18}$$

$\mathsf{Q}_\infty$ est la solution de l'équation (2.10) de Bellman dans le cas où toutes les récompenses seraient identiques ($\mathsf{r}_a = \mathsf{r}_i$). Néanmoins, si $\mathsf{r}_a \neq \mathsf{r}_i$, les valeurs d'utilité des états éloignés de l'état absorbant vont converger vers $\mathsf{Q}_\infty$. Ces états seront donc plus ou moins attirants par rapport aux états proches de l'état absorbant en fonction des valeurs relatives de $\mathsf{r}_a$ et de $\mathsf{Q}_\infty$.

**Comportement avec $\mathsf{r}_a > \mathsf{Q}_\infty$ et $0 < \gamma < 1$ :** dans ce cas, l'utilité des couples *état—commande* menant à l'état absorbant est attirante par rapport à $\mathsf{Q}_\infty$. La stratégie optimale obtenue est donc le plus court chemin vers l'état absorbant comme précédemment.

| | $\gamma = 0$ | $0 < \gamma < 1$ |
|---|---|---|
| $r_a > \frac{\mathsf{r}_i}{1-\gamma}$ | Attraction locale de $x_a$ | Attraction totale de $x_a$ |
| $r_a < \frac{\mathsf{r}_i}{1-\gamma}$ | Répulsion locale de $x_a$ | |

TAB. 2.1 – *Stratégie optimale obtenue en fonction des valeurs relatives de* $\mathsf{r}_a$, $\mathsf{r}_i$ *et* $\gamma$.

**Comportement avec $\mathsf{r}_a > \mathsf{Q}_\infty$ et $\gamma = 0$ :** la stratégie optimale contraint uniquement les commandes des états voisins de l'état absorbant.

**Comportement avec $\mathsf{r}_a < \mathsf{Q}_\infty$ :** dans ce cas, l'utilité des couples *état—commande* menant à l'état absorbant est moins attirante et la stratégie optimale obtenue est une répulsion locale.

Le tableau 2.1 résume ces résultats. Il permet d'aider le choix des récompenses en fonction de l'objectif désiré.

**Comportement en début d'apprentissage :** comme dans le cas des récompenses locales, on peut influencer le comportement en début d'apprentissage.

Le calcul de la première mise à jour telle que l'état suivant ne soit pas l'état absorbant donne :

$$\begin{aligned} \mathsf{Q}(x,u) &\leftarrow \mathsf{Q}_i - \alpha[(1-\gamma)\mathsf{Q}_i - \mathsf{r}_i] \\ &\leftarrow \mathsf{Q}_i - \alpha(1-\gamma)(\mathsf{Q}_i - \mathsf{Q}_\infty) \end{aligned} \tag{2.19}$$

Ainsi, $\mathsf{Q}_i > \mathsf{Q}_\infty$ induit une exploration systématique, $\mathsf{Q}_i < \mathsf{Q}_\infty$ un piétinement et $\mathsf{Q}_i = \mathsf{Q}_\infty$ une commande aléatoire.

De même, $(1-\gamma)\mathsf{Q}_i > \mathsf{r}_a$ induit l'évitement de l'état absorbant, $(1-\gamma)\mathsf{Q}_i < \mathsf{r}_a$ une attraction et $(1-\gamma)\mathsf{Q}_i = \mathsf{r}_a$ aucune contrainte.

Le tableau 2.2 synthétise ces résultats pour permettre de choisir le comportement de l'algorithme en début d'apprentissage.

Tous ces résultats sur les stratégies et comportements obtenus avec le Q-Learning sont généralisables à la plupart des algorithmes d'apprentissage par renforcement. Nous les utilisons dans notre approche pour le choix de tous nos paramètres et récompenses.

| | $\mathsf{Q}_i < \dfrac{r_a}{1-\gamma}$ | $\mathsf{Q}_i = \dfrac{r_a}{1-\gamma}$ | $\mathsf{Q}_i > \dfrac{r_a}{1-\gamma}$ |
|---|---|---|---|
| $\mathsf{Q}_i < \dfrac{r_0}{1-\gamma}$ | piétinement et attraction locale de $x_a$ | aléatoire | piétinement et évitement local de $x_a$ |
| $\mathsf{Q}_i = \dfrac{r_0}{1-\gamma}$ | aléatoire | aléatoire | aléatoire |
| $\mathsf{Q}_i > \dfrac{r_0}{1-\gamma}$ | exploration systématique et attraction locale de $x_a$ | aléatoire | exploration systématique et évitement local de $x_a$ |

TAB. 2.2 – *Comportements en début d'apprentissage en fonction des valeurs relatives de* $\mathsf{Q}_i$, $r_a$, $r_0$ *et* $\gamma$.

**Cas des récompenses calculées par une fonction objectif**

Pour accélérer l'apprentissage, certains auteurs définissent les récompenses comme un critère lié à une fonction objectif (Maja Mataric 1994, Yassine Faihe 1999). Cette fonction définit une mesure de chaque couple *état—commande* par rapport à un objectif. Soit $\mathsf{f}$ une fonction définie de l'espace état-commande sur $\mathbb{R}$. Le but de l'apprentissage revient à maximiser la fonction $\mathsf{f}(s,u)$.

Par exemple, dans le cas du labyrinthe, la fonction objectif peut être la distance de l'état actuel à l'état absorbant $x_a$ :

$$\mathsf{f}(x,u) = d(x,x_a) \tag{2.20}$$

On cherche à minimiser cette fonction et on pose :

$$\mathsf{r}(x,u,x') = -\Delta\mathsf{f} = \mathsf{f}(x,u) - \mathsf{f}(x',u) \tag{2.21}$$

Ainsi, la fonction récompense vaut 1 si la commande $u$ rapproche la souris de l'état terminal, 0 si la souris ne bouge pas et $-1$ si la souris s'éloigne de l'état absorbant. La fonction de récompense est recalculée à chaque commande de la souris.

Avec ces récompenses, les résultats de l'apprentissage sont évidemment meilleurs. Le premier épisode dure seulement 65 pas et l'algorithme converge en 5 épisodes. Il faut en tout 123 pas pour que l'algorithme converge.

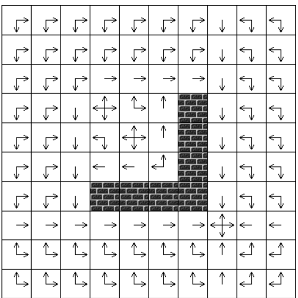


FIG. 2.8 – *Stratégie optimale pour atteindre l'état terminal dans le cas d'un labyrinthe avec des murs.*

Cet exemple est extrêmement simple. Prenons maintenant un labyrinthe légèrement plus complexe avec quelques murs au milieu. La stratégie optimale pour atteindre l'état terminal est décrite par la figure 2.8.

Dans ce cas, la fonction objectif n'est plus adaptée et induit le contrôleur en erreur. L'algorithme met plus de temps à converger, mais surtout il se bloque pendant plusieurs centaines de milliers de pas dans le coin formé par les nouveaux murs et n'en sort que par exploration. Il faut 2 358 000 pas cumulés pour obtenir une stratégie quasi-optimale.

Sur cet exemple avec des murs, l'algorithme avec une récompense locale égale à 1 pour l'état absorbant se comporte mieux. Il suffit de 6 400 pas en tout pour obtenir le même résultat !

Ainsi, l'introduction d'une fonction objectif peut être néfaste à l'apprentissage. De plus, si on peut caractériser correctement la fonction objectif, on connaît déjà une grande partie du modèle de la dynamique du système.

### Influence de la valeur des paramètres $\alpha$ et $\gamma$

Sur cet exemple, le paramètre $\alpha$ influence légèrement la vitesse de convergence. Comme le système est déterministe, on peut même fixer $\alpha$ à 1 pour aller le plus vite possible. Si le système était non déterministe, $\alpha$ permettrait de réaliser une approximation des probabilités de transition. Il peut donc s'avérer utile de réaliser des tests pour choisir la meilleure valeur de $\alpha$.

Le paramètre $\gamma$ permet normalement de donner plus ou moins d'importance aux récompenses éloignées dans le futur. En fait, ce paramètre n'a d'influence que si plusieurs transitions provoquent des récompenses, permettant alors de choisir le niveau d'*opportunisme* du contrôleur. Prenons l'exemple du labyrinthe particulier de la figure 2.9 où chaque fromage provoque une récompense de +1 et disparaît ensuite.

FIG. 2.9 – *Exemple de configuration où la valeur de $\gamma$ intervient sur le comportement de l'algorithme.*

Le contrôleur a le choix entre deux trajectoires, la première en commençant par le fromage le plus proche (à gauche), la seconde en commençant par la zone la plus dense en fromages (à droite). Dans cet exemple, l'espérance de gain de la commande « à gauche » s'écrit $\mathsf{Q}(\mathrm{G}) = \gamma^2 + \gamma^7 + \gamma^9 + \gamma^{11}$ et l'espérance de gain de la commande « à droite » vaut $\mathsf{Q}(\mathrm{D}) = \gamma^3 + \gamma^5 + \gamma^7 + \gamma^{16}$. Si $\gamma = 0{,}1$, alors $\mathsf{Q}(\mathrm{G}) > \mathsf{Q}(\mathrm{D})$ et le contrôleur prendra la première trajectoire (à gauche) délaissant les récompenses éloignées dans le futur. En revanche, si $\gamma = 0{,}8$, alors $\mathsf{Q}(\mathrm{G}) < \mathsf{Q}(\mathrm{D})$ et le contrôleur prendra la seconde trajectoire (à droite) délaissant la récompense proche. On ne peut pas généraliser car chaque cas peut créer un résultat différent (dans l'exemple $\gamma = 0{,}9$ donne $\mathsf{Q}(\mathrm{G}) > \mathsf{Q}(\mathrm{D})$). Si on choisit $\gamma$ faible, alors le contrôleur sera plutôt opportuniste et atteindra les récompenses proches, si on choisit $\gamma$ proche de 1, alors le contrôleur préférera atteindre des zones plus denses même éloignées.

#### Discussion

Le Q-Learning offre de nombreux avantages qui font de lui l'algorithme d'apprentissage par renforcement le plus utilisé, notamment :

- le Q-Learning converge dans le cadre des processus de décision markoviens déterministes et non déterministes et sous certaines conditions,
- l'algorithme requière très peu de puissance de calcul (une mise à jour par période d'échantillonnage).

Malgré son succès, le Q-Learning est loin d'être la méthode parfaite, en effet :

- pour obtenir une bonne stratégie, il faut visiter plusieurs fois quasiment tous les états,
- le Q-Learning converge lentement, ce qui le limite à de petits espaces d'états,
- l'algorithme mémorise la fonction d'utilité $\mathsf{Q}$ dans un tableau, ce qui le restreint aussi à de petits espaces d'états si la mémoire est limitée.

### 2.4.3 TD($\lambda$)

#### Principe

Introduit par Richard Sutton (1988), le *TD(λ)* recouvre une classe d'algorithmes qui mettent à jour tous les états dernièrement visités à l'aide de la différence temporelle de l'état courant. Une pondération de cette mise à jour est déterminée par l'ancienneté de la dernière visite. Dans ce sens, ces algorithmes utilisent la notion de *trace* ou d'*éligibilité*.

Une fonction d'éligibilité qualifie à l'aide d'un nombre réel l'ancienneté de la dernière visite de chaque couple *état—commande*. L'éligibilité d'un couple $(x,u)$ est notée $\mathsf{e}(x,u)$. A chaque période d'échantillonnage, les valeurs d'éligibilité diminuent de façon exponentielle. A chaque passage par un couple *état—commande* $(x,u)$, la trace est remise à jour soit par $\mathsf{e}(x,u) \leftarrow 1$, on parle alors de d'*éligibilité remplaçante* [22], soit par $\mathsf{e}(x,u) \leftarrow \mathsf{e}(x,u) + 1$ dans le cas de l'*éligibilité accumulatrice* [23]. D'après Satinder Singh

22. Termes anglais : replacing trace.
23. Termes anglais : accumulating trace.

**Début**
Initialiser $\mathsf{Q}(x,u)$ arbitrairement et $\mathsf{e}(x,u)$ à zéro pour tout $(x,u)$ de $\mathcal{X} \times \mathcal{U}$
Initialiser l'état initial $x$ et choisir une commande $u$ au hasard
**Tant que** $x$ n'est pas un état absorbant **faire**
Exécuter $u$ et observer le nouvel état $x'$ et la récompense $r$
$u' \leftarrow$Décision$(\mathsf{Q},x')$
$\delta \leftarrow r + \gamma \mathsf{Q}(x',u') - \mathsf{Q}(x,u)$
$\mathsf{e}(x,u) \leftarrow \mathsf{e}(x,u)+1$ ou $\mathsf{e}(x,u) \leftarrow 1$ *(éligibilité accumulatrice ou remplaçante)*
**Pour tout** $(y,v) \in \mathcal{X} \times \mathcal{U}$ **faire**
$Q(y,v) \leftarrow \mathsf{Q}(y,v) + \alpha\delta \mathsf{e}(y,v)$
$\mathsf{e}(y,v) \leftarrow \gamma\lambda \mathsf{e}(y,v)$
**Fin pour**
$x \leftarrow x'$, $u \leftarrow u'$
**Fin tant que**
**Fin**

FIG. 2.10 – *Algorithme du Sarsa(λ).*

et Richard Sutton (1996), l'éligibilité remplaçante peut parfois améliorer la vitesse de convergence.

### Algorithme

Il existe deux types d'algorithmes de TD($\lambda$). Le premier est dit *on-line* car il utilise les commandes effectuées pour mettre à jour $\mathsf{Q}$, le second est dit *off-line* car il recherche la commande qui maximise $\mathsf{Q}$ pour la même mise à jour sans forcément effectuer réellement cette commande.

L'algorithme on-line le plus connu est le *Sarsa(λ)* de G. Rummery et M. Niranjan (1994, 1995) (*cf.* figure 2.10). Christopher Watkins (1989) et Jing Peng (1993, 1994) ont tous deux présenté une version off-line de l'algorithme du TD($\lambda$). Baptisés *Q(λ)*, ces algorithmes généralisent l'algorithme du Q-Learning à toute valeur de $\lambda$ (*cf.* figure 2.11). Contrairement à celui de Jing Peng, le Q($\lambda$) de Christopher Watkins remet les traces à zéro quand le contrôleur effectue une commande d'exploration.

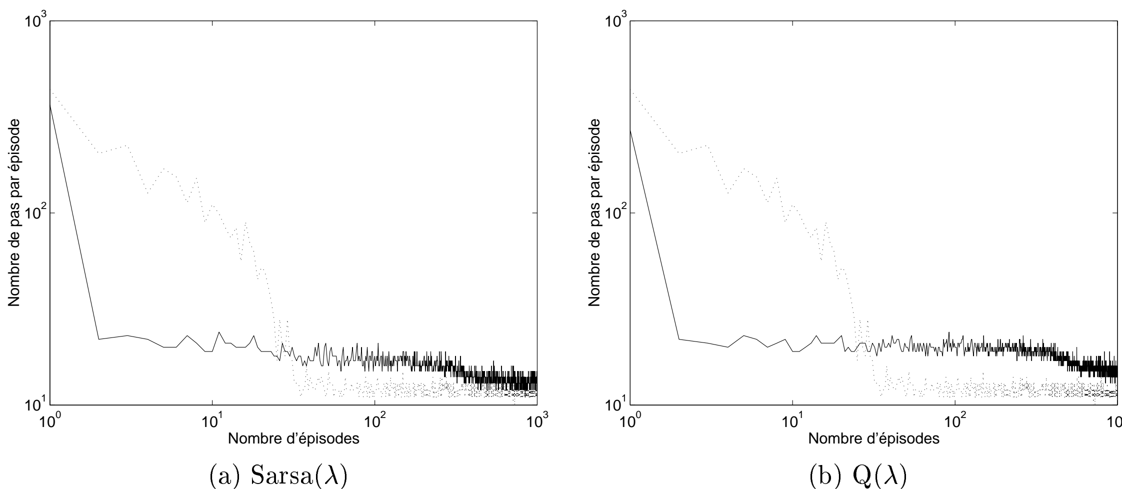
La convergence des algorithmes de TD($\lambda$) en versions on-line et off-line a été prouvée quelle que soit la valeur de $\lambda$ sous les hypothèses classiques (*cf.* §2.4.1, page 21) (Peter Dayan 1992, Jing Peng 1993, P. Dayan et T. Sejnowski 1994, John Tsitsiklis 1994, Tommi Jaakkola, Michael Jordan et Satinder Singh 1994).

### Illustration

Le test de ces algorithmes sur l'exemple précédent donne les résultats présentés en figure 2.12.

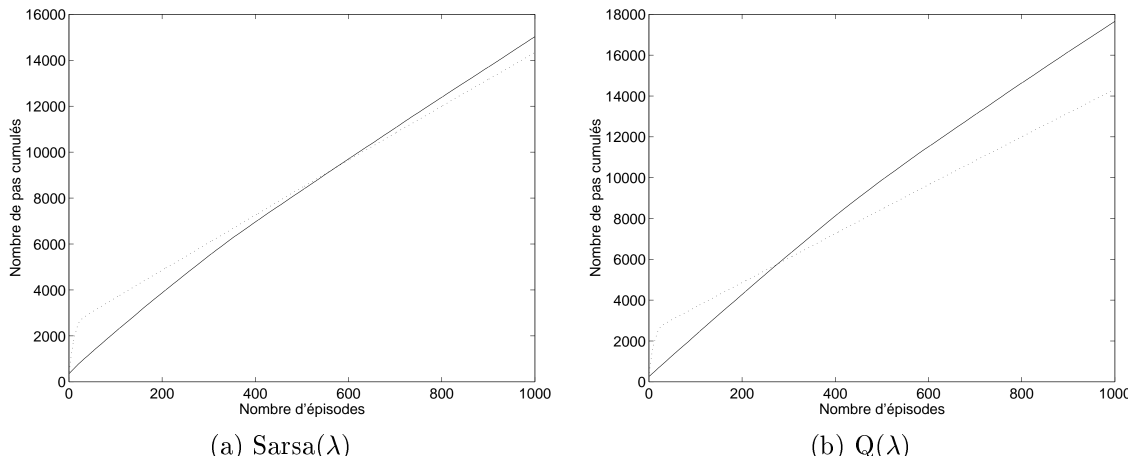
**Début**
Initialiser $\mathsf{Q}(x,u)$ arbitrairement et $\mathsf{e}(x,u)$ à zéro pour tout $(x,u)$ de $\mathcal{X} \times \mathcal{U}$
Initialiser l'état initial $x$ et choisir une commande $u$ au hasard
**Tant que** $x$ n'est pas un état absorbant **faire**
Exécuter $u$ et observer le nouvel état $x'$ et la récompense $r$
$u \leftarrow$Décision$(\mathsf{Q},x')$
$u^* \leftarrow \arg\max_{v \in \mathcal{U}} \mathsf{Q}(x',v)$
$\delta \leftarrow r + \gamma \mathsf{Q}(x',u^*) - \mathsf{Q}(X,u)$
$\mathsf{e}(x,u) \leftarrow \mathsf{e}(x,u)+1$ ou $\mathsf{e}(x,u) \leftarrow 1$ (*éligibilité accumulatrice ou remplaçante*)
**Pour tout** $(y,v) \in \mathcal{X} \times \mathcal{U}$ **faire**
$\mathsf{Q}(y,v) \leftarrow \mathsf{Q}(y,v) + \alpha\delta \mathsf{e}(y,v)$
$\mathsf{e}(y,v) \leftarrow \begin{cases} \gamma\lambda \mathsf{e}(y,v) & \text{si } u' = u^* \\ 0 & \text{sinon} \end{cases}$
**Fin pour**
$x \leftarrow x'$, $u \leftarrow u'$
**Fin tant que**
**Fin**

FIG. 2.11 – *Algorithme du Q(λ) de Christopher Watkins.*




(a) Sarsa(λ) (b) Q(λ)

FIG. 2.12 – *Courbes d'apprentissage du Sarsa(λ) et du Q(λ) (traits continus) avec des traces remplaçantes par rapport au Q-Learning (traits pointillés) (moyenne sur 20 simulations avec* $\mathsf{r}_a = 1$, $\mathsf{r}_i = 0$, $\lambda = 0{,}7$, $\gamma = 0{,}9$, $\alpha = 0{,}1$, $\epsilon = 0{,}1$ *et* $\mathsf{Q}_i = 0$*).*




(a) Sarsa($\lambda$) (b) Q($\lambda$)

FIG. 2.13 – *Courbes du nombre de pas cumulés du Sarsa($\lambda$) et du Q($\lambda$) (traits continus) avec des traces remplaçantes par rapport au Q-Learning (traits pointillés) (moyenne sur 20 simulations avec* $\mathsf{r}_a = 1$, $\mathsf{r}_i = 0$, $\lambda = 0{,}7$, $\gamma = 0{,}9$, $\alpha = 0{,}1$, $\epsilon = 0{,}1$ *et* $\mathsf{Q}_i = 0$*).*

Les deux algorithmes donnent des résultats similaires. En un coup d'horloge, ils trouvent un chemin pour aller à l'état récompensé (environ deux fois plus long que l'optimal). Malheureusement, il faut par la suite plus de temps pour trouver le plus court chemin.

Si on compare les courbes de la figure 2.13, on remarque que le nombre de pas cumulés des algorithmes TD($\lambda$) dépasse au bout d'un certain nombre d'épisodes celui du Q-Learning. L'accélération des algorithmes TD($\lambda$) est donc toute relative.

Au delà de ces résultats, nous pensons que ces algorithmes sont très sensibles au choix de la décision entre exploration et exploitation. Quand l'algorithme a trouvé un chemin, l'amélioration ne peut se faire que par exploration et l'$\epsilon$-gourmand n'est sans doute pas le meilleur compromis.

### Discussion

Il existe peu d'études comparatives entre les performances du Q-Learning, du Sarsa($\lambda$) et du Q($\lambda$). On peut néanmoins faire quelques observations.

D'une part, les avantages de ces algorithmes recoupent ceux du Q-Learning :

- le TD($\lambda$) converge dans les mêmes cas que le Q-Learning,
- le TD($\lambda$) trouve très rapidement une stratégie relativement bonne.

D'autre part, les inconvénients sont assez importants :

- une fois que l'algorithme a trouvé une stratégie acceptable, il met plus de temps que le Q-Learning pour trouver la stratégie optimale,
- le TD($\lambda$) mémorise l'éligibilité en plus de la fonction d'utilité, ce qui nécessite deux tableaux de dimension $|\mathcal{X}| \times |\mathcal{U}|$ et le limite à de petits espaces d'états,
- le TD($\lambda$) nécessite une importante puissance de calcul puisqu'il réalise $|\mathcal{X}| \times |\mathcal{U}|$ mises à jour à chaque période d'échantillonnage.

### 2.4.4 Autres méthodes directes

Il existe beaucoup d'autres méthodes directes d'apprentissage par renforcement. Sans prétendre à l'exhaustivité, nous présentons ici succinctement d'autres algorithmes qui nous semblent intéressants.

Le *AHC-Learning* cité plus haut est historiquement le premier algorithme d'apprentissage par renforcement, mais il est en pratique peu utilisé.

Le *R-Learning* de A. Schwartz (1993) est une version stochastique de la méthode par itérations sur les valeurs pour le critère moyen. Bien qu'il n'existe pas de preuve formelle de convergence du R-Learning vers une stratégie optimale, il semble en pratique qu'il présente des propriétés de vitesse de convergence plus intéressantes que le Q-Learning (Sridhar Mahadevan 1994, Sridhar Mahadevan 1996).

Enfin, il existe une version partielle du TD($\lambda$) appelée *truncated temporal differences* ou *TTD($\lambda$)* (Pawel Cichosz 1995). Cet algorithme allège les calculs en tronquant l'historique des états visités. Il ne met à jour que les $m$ derniers états visités qui sont mémorisés dans une file d'attente. Ce principe permet de réaliser des apprentissages avec des espaces d'états plus grands que ceux permis avec le TD($\lambda$) traditionnel.

### 2.4.5 Synthèse

De manière générale, les méthodes directes d'apprentissage par renforcement permettent d'obtenir une stratégie de commande proche de l'optimale. En effet, leur convergence a été prouvée dans le cadre des processus de décision markoviens déterministes ou non.

Deux inconvénients majeurs limitent leur utilisation. Tout d'abord, l'utilisation de tableaux pour stocker les valeurs est adaptée à des espaces d'états de petites tailles. Nous verrons dans la section 2.6 des approches qui évitent l'utilisation de tableaux. Deuxième inconvénient, leur vitesse de convergence relativement lente rend difficile leur utilisation avec des applications réelles car alors le temps d'expérimentation risque d'être extrêmement long. Ce second point est en partie résolu par les algorithmes par renforcement indirects.

## 2.5 Méthodes indirectes

### 2.5.1 Principe général

Les méthodes indirectes sont plus proches de la programmation dynamique dans le sens où il s'agit d'optimiser la fonction d'utilité à l'aide d'un modèle partiel du système. La figure 2.14 montre le schéma fonctionnel des méthodes indirectes. Dans ces approches, la fonction d'optimisation est indépendante de la boucle sensori-motrice et utilise indirectement les informations stockées par la fonction de modélisation qui enrichit le modèle au fur et à mesure des états rencontrés. L'avantage de cette désynchronisation est de permettre une optimisation permanente même quand la boucle sensori-motrice est momentanément stoppée.

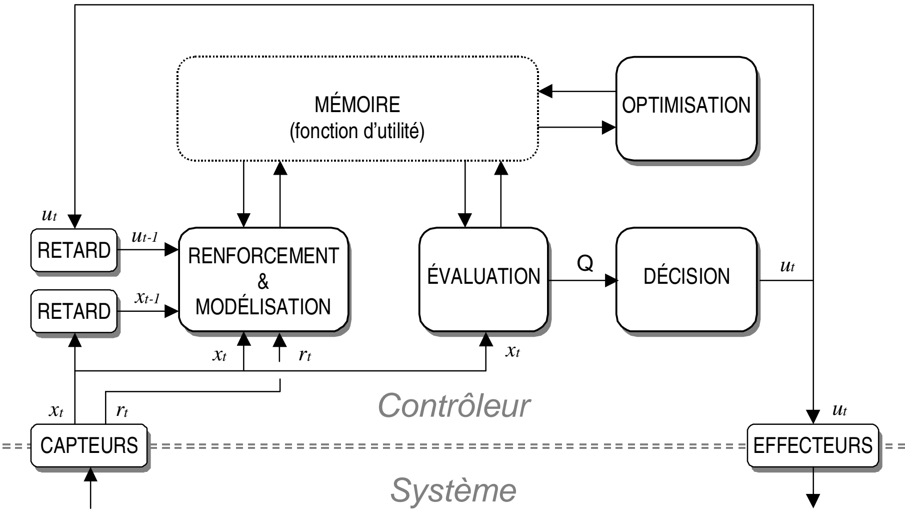



FIG. 2.14 – *Schéma fonctionnel des méthodes indirectes.*

### 2.5.2 Dyna-Q

#### Algorithme

L'algorithme *Dyna-Q* et ses variantes ont été développés en grande partie par Richard Sutton dans les années 90 (Richard Sutton 1990, Richard Sutton 1991a, Richard Sutton 1991b, Richard Sutton 1992) .

Le Dyna-Q utilise un modèle partiel et déterministe du système. Ce modèle est enrichi par les nouvelles expériences à chaque période d'échantillonnage. Noté $\mathsf{m}(x,u)$, le modèle revoit pour chaque couple *état—commande* déjà visité l'état suivant observé $x'$ ainsi que la récompense observée $r$. A l'aide de ce modèle, l'algorithme peut optimiser la fonction d'utilité à chaque période d'échantillonnage ou même indépendamment lors d'une pause dans la boucle sensori-motrice. Le nombre de mises à jour par période d'échantillonnage est noté $N$. L'algorithme du Dyna-Q est présenté en figure 2.15.

L'utilisation du Dyna-Q est restreinte au processus de décision markovien *déterministe* et avec des récompenses *immédiates*. Cette limitation rend souvent son emploi impossible dans des applications non simulées.

On peut néanmoins envisager une version non déterministe qui construirait un modèle plus complet du système à l'aide d'une matrice de taille $|\mathcal{X}|\times|\mathcal{U}|\times|\mathcal{X}|$. Même si la matrice n'était que peu remplie, l'utilisation d'un tel algorithme se limiterait à de petits espaces d'états.

#### Illustration

La figure 2.16 montre les résultats du Dyna-Q sur l'exemple du labyrinthe. L'algorithme converge très rapidement vers la stratégie de commande optimale. La solution

**Début**

Initialiser $\mathsf{Q}(x,u)$ arbitrairement et $\mathsf{m}(x,u)$ à $\emptyset$ pour tout $(x,u)$ de $\mathcal{X} \times \mathcal{U}$
Initialiser l'état initial $x$
**Tant que** $x$ n'est pas un état absorbant **faire**
$u \leftarrow$Décision$(\mathsf{Q},x)$
Exécuter $u$ et observer le nouvel état $x'$ et la récompense $r$
$\mathsf{Q}(x,u) \leftarrow \mathsf{Q}(x,u) + \alpha[r + \gamma \max_{v \in \mathcal{U}} \mathsf{Q}(x',v) - \mathsf{Q}(x,u)]$
$\mathsf{m}(x,u) \leftarrow (x',r)$
$x \leftarrow x'$
**Répéter** $N$ fois
Choisir $(y,v)$ au hasard tel que $\mathsf{m}(y,v) \neq \emptyset$
$(y',r) \leftarrow \mathsf{m}(y,v)$
$\mathsf{Q}(y,v) \leftarrow \mathsf{Q}(y,v) + \alpha[r + \gamma \max_{w \in \mathcal{U}} \mathsf{Q}(y',w) - \mathsf{Q}(y,v)]$
**Fin Répéter**
**Fin tant que**
**Fin**

FIG. 2.15 – *Algorithme du Dyna-Q.*

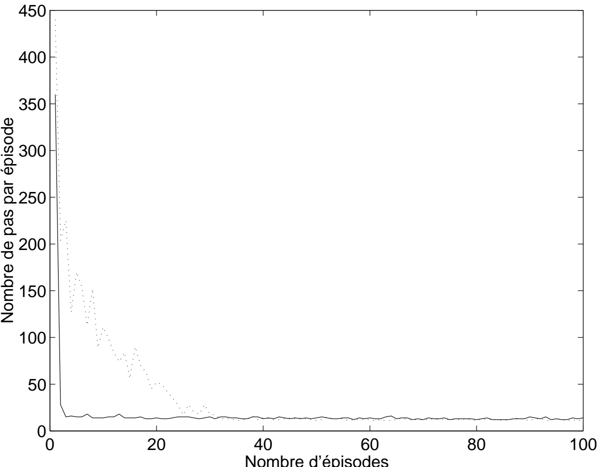



FIG. 2.16 – *Courbe d'apprentissage du Dyna-Q (trait continu) par rapport au Q-Learning (trait pointillé) (moyenne sur 20 simulations avec $N = 100$, $\mathsf{r}_a = 1$, $\mathsf{r}_i = 0$, $\gamma = 0{,}9$, $\alpha = 0{,}1$, $\epsilon = 0{,}1$ et $\mathsf{Q}_i = 0$).*

obtenue est légèrement moins bonne que l'optimale mais bien meilleure que celle trouvée par un algorithme type TD($\lambda$).

#### Discussion

Si la supériorité du Dyna-Q en matière de vitesse de convergence est évidente, cet algorithme présente cependant des inconvénients, notamment :

- le Dyna-Q est limité au contrôle de systèmes déterministes à récompenses immédiates,
- le Dyna-Q utilise trois tableaux de dimension $|\mathcal{X}| \times |\mathcal{U}|$,
- le Dyna-Q demande une puissance de calcul importante mais néanmoins paramétrable et désynchronisable. En effet, on peut choisir le nombre de mises à jour par période d'échantillonnage et, lorsque la boucle de contrôle est momentanément interrompue, le contrôleur peut continuer à faire des mises à jour (une sorte de phase de sommeil ou de réflexion).

### 2.5.3 Autres méthodes indirectes

La plupart des autres algorithmes indirects sont inspirés du Dyna-Q. Fondés sur la même structure, ils optimisent la fonction d'utilité selon une heuristique afin de réduire le nombre de mises à jour à effectuer. Comme pour les algorithmes directs, nous limitons la présentation des algorithmes indirects à quelques algorithmes représentatifs.

Le *Priority Sweeping* ou *Queue-Dyna* a été développé simultanément et indépendamment par Andrew Moore et Christopher Atkeson (1993) et par Jing Peng et Ronald Williams (1993). Cet algorithme utilise une heuristique pour optimiser la fonction d'utilité de manière plus rapide. Cette heuristique utilise une liste triée d'états qui permet de commencer par les mises à jour dont les différences temporelles sont les plus grandes. Pour ajouter de nouveaux états à cette liste, l'algorithme utilise une matrice de transition de taille $|\mathcal{X}| \times |\mathcal{X}| \times |\mathcal{U}|$. Cette matrice permet de plus d'utiliser l'algorithme avec des systèmes non déterministes. Pour défendre l'emploi d'une telle matrice, les auteurs expliquent que chaque état n'a que quelques couples *état—commande* précédents et que dans une application en temps réel, le contrôleur n'a jamais le temps de visiter tous les états. Ainsi, la quantité d'informations à mémoriser est très inférieure à $|\mathcal{X}| \times |\mathcal{X}| \times |\mathcal{U}|$.

L'algorithme RTDP (real-time dynamic programming) (Andrew Barto, Steven Bradtke et Satinder Singh 1995) est quant à lui dédié aux systèmes comportant un état objectif dont la récompense associée est positive (les autres récompenses sont nulles). Cet algorithme cherche uniquement le chemin le plus court entre l'état initial et l'état objectif sans forcément optimiser tout l'espace d'états visité.

### 2.5.4 Synthèse

Les algorithmes indirects apportent des améliorations importantes quant à la vitesse de convergence et à la possibilité de désynchroniser l'optimisation. Ces avancées sont contrebalancées par une extrême lourdeur de la structure et des calculs. Néanmoins,

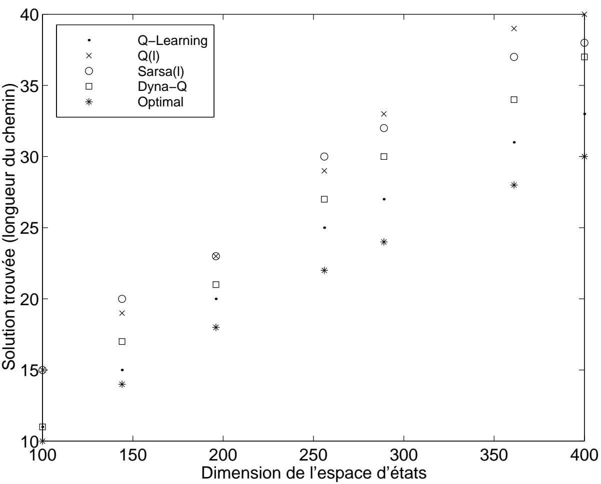



FIG. 2.17 – *Comparaison de la solution trouvée des algorithmes Q-Learning, Q(λ), Sarsa(λ) et Dyna-Q en fonction de la taille du labyrinthe (moyennes sur 20 simulations avec* $\mathsf{r}_a = 1$, $\mathsf{r}_i = 0$, $\lambda = 0{,}7$, $\gamma = 0{,}9$, $\alpha = 0{,}1$, $\epsilon = 0{,}1$ *et* $\mathsf{Q}_i = 0$*).*

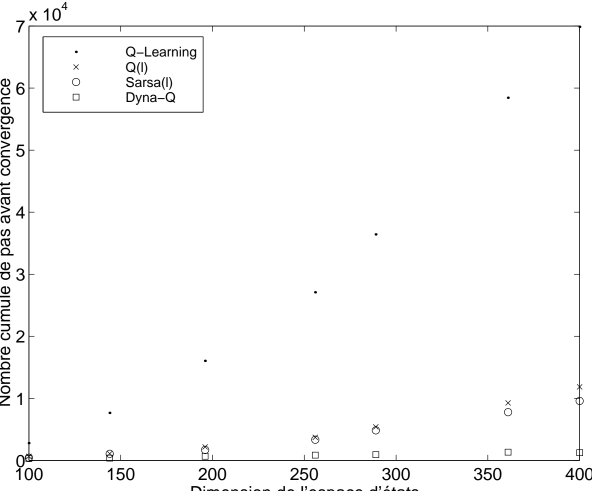



FIG. 2.18 – *Comparaison du temps d'apprentissage des algorithmes Q-Learning, Q(λ), Sarsa(λ) et Dyna-Q en fonction de la taille du labyrinthe (moyennes sur 20 simulations avec* $\mathsf{r}_a = 1$, $\mathsf{r}_i = 0$, $\lambda = 0{,}7$, $\gamma = 0{,}9$, $\alpha = 0{,}1$, $\epsilon = 0{,}1$ *et* $\mathsf{Q}_i = 0$*).*

ces méthodes sont intéressantes pour apprendre à contrôler des systèmes dont l'espace d'états est de faible dimension.

Nous avons comparé les quatre algorithmes présentés (Q-Learning, Q($\lambda$), Sarsa($\lambda$) et Dyna-Q) sur des labyrinthes de différentes tailles. Les stratégies trouvées sont plus ou moins proches de la stratégie optimale (*cf.* figure 2.17). Le Q-Learning donne les meilleures stratégies. On observe que les temps de convergence augmentent rapidement (*cf.* figure 2.18). L'algorithme qui converge le plus rapidement est sans conteste le Dyna-Q, mais au prix d'un nombre importants de calculs.

En conclusion, le défaut majeur des méthodes directes et indirectes est de ne pouvoir fonctionner qu'avec de petits espaces d'états. Au travers de cette constatation perce un autre problème : celui de la généralisation. En effet, même si un contrôleur a appris une bonne stratégie de commande, s'il rencontre un état encore inconnu, ses commandes seront purement aléatoires alors qu'il a peut-être trouvé la commande optimale dans des états voisins. Pour cette raison, beaucoup de recherches ont été effectuées sur l'utilisation de fonctions d'approximation.

## 2.6 Méthodes directes et fonctions d'approximation

L'idée de base de ces méthodes est de remplacer le tableau stockant les valeurs de $\mathsf{Q}$ par une fonction définie par un nombre de paramètres inférieur au nombre d'états. Dans cette optique, il existe deux approches différentes. La première approche, la plus employée, consiste à utiliser une fonction paramétrée différentiable. L'apprentissage s'opère alors sur les paramètres. Une seconde approche, plus complexe, s'autorise l'apprentissage de la structure même de la fonction d'approximation.

Nous proposons dans cette section une synthèse succincte de ces approches car elles ne seront pas utilisées par la suite.

### 2.6.1 Fonctions d'approximation différentiables

Soit $\mathsf{f}$ la fonction d'approximation et $\theta$ le vecteur de ses paramètres. On pose :

$$\mathsf{Q}_t(x,u) = f(\theta_t) \tag{2.22}$$

Pour mettre à jour $\mathsf{Q}$ dans ces conditions, il est naturel d'utiliser des méthodes comme la descente de gradient pour minimiser l'erreur quadratique. A chaque itération, on corrige alors les paramètres par :

$$\theta_{t+1} = \theta_t + \frac{1}{2}\alpha\nabla_{\theta_t}\left[\mathsf{Q}^d(x_t,u_t) - \mathsf{Q}_t(x_t,u_t)\right]^2 = \theta_t + \alpha\left[\mathsf{Q}^d(x_t,u_t) - \mathsf{Q}_t(x_t,u_t)\right]\nabla_{\theta_t}\mathsf{Q}_t(x_t,u_t) \tag{2.23}$$

avec $\mathsf{Q}^d(x_t,u_t)$ la nouvelle estimation et :

$$\nabla_\theta f(\theta) = \left(\frac{\partial f(\theta)}{\partial\theta(1)},\frac{\partial f(\theta)}{\partial\theta(2)},\ldots\right)^T \tag{2.24}$$

On adapte ensuite cette équation au type d'apprentissage sélectionné.

Pour le TD(0), on corrige les paramètres par :

$$\theta_{t+1} = \theta_t + \alpha \left[ r_{t+1} + \gamma \mathsf{Q}_t(x_{t+1}, u_{t+1}) - \mathsf{Q}_t(x_t, u_t) \right] \nabla_{\theta_t} \mathsf{Q}_t(x_t, u_t) \tag{2.25}$$

Pour le TD($\lambda$), on corrige les paramètres par :

$$\theta_{t+1} = \theta_t + \alpha \left[ r_{t+1} + \gamma \mathsf{Q}_t(x_{t+1}, u_{t+1}) - \mathsf{Q}_t(x_t, u_t) \right] \vec{\mathsf{e}}_t \tag{2.26}$$

avec :

$$\vec{\mathsf{e}}_t = \gamma \lambda \vec{\mathsf{e}}_{t-1} + \nabla_{\theta_t} \mathsf{Q}_t(x_t, u_t) \tag{2.27}$$

Les principales méthodes d'approximation par une fonction différentiable se décomposent en méthodes linéaires et non linéaires

### 2.6.2 Fonctions d'approximation linéaires

Dans ce cas $\mathsf{Q}_t(x_t, u_t)$ est une fonction linéaire des paramètres $\theta_t$, soit :

$$\mathsf{Q}_t(x_t, u_t) = \theta_t^T \vec{\phi}_{x,u} = \sum_{i=1}^{n} \theta_t(i) \phi_{x,u}(i) \tag{2.28}$$

On a alors :

$$\nabla_{\theta_t} \mathsf{Q}_t(x, u) = (\phi_{u,x}(1), \phi_{x,u}(2), \phi_{x,u}(3), \ldots)^T \tag{2.29}$$

L'intérêt des fonctions linéaires est l'existence d'un optimum unique $\theta^*$ pour l'erreur quadratique. De plus, on montre que la méthode du gradient associé au TD($\lambda$) converge nécessairement, mais pas forcément vers $\theta^*$ (D. Bertsekas et J. Tsitsiklis 1996, §6.3.3).

Il existe plusieurs types de codages pour la fonction $\vec{\phi}_x$ classés en deux familles les fonctions de voisinages et les fonctions à base d'états représentatifs.

#### Fonctions de voisinage

Ces fonctions associent à chaque paramètre $\theta(i)$ une région $i$ de l'espace et un paramètre binaire $\phi(i)$. On pose :

$$\phi_{x,u}(i) = \begin{cases} 1 & \text{si } (x,u) \text{ est dans la zone associée à } \theta(i) \\ 0 & \text{sinon} \end{cases} \tag{2.30}$$

L'idée la plus simple consiste à regrouper plusieurs états dans une région, diminuant ainsi la complexité (John Tsitsiklis et Ben Roy 1996). Mais cette approche pose le problème du choix des régions.

L'approche par voisinage la plus classique a été introduite par J. Albus (1975) sous le nom de *cerebellar model articulator controller* ou *CMAC*. Son principe est de définir plusieurs partitions de l'espace (circulaires ou carrées), décalées géométriquement les

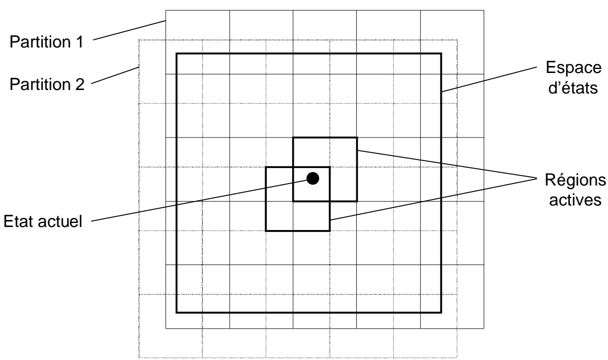



FIG. 2.19 – *Exemple de partition d'un CMAC.*

unes par rapport aux autres (*cf.* figure 2.19). Le CMAC a été très largement utilisé en apprentissage par renforcement particulièrement pour des espaces continus ou de grande taille (Christopher Watkins 1989, Doina Precup et Richard Sutton 1997). Richard Sutton (1995) a démontré son efficacité sur trois exemples de contrôles en grands espaces continus (puddle world, mountain car, acrobot) par rapport à une approche par réseau de neurones artificiels proposée par J. Boyan et A. Moore (1995).

**Fonctions à base d'états représentatifs**

Ces méthodes consistent à placer un petit nombre d'états représentatifs $c_i$ dans l'espace. La fonction d'utilité est alors réalisée par interpolation.

L'interpolation peut être fonction de la distance de l'état courant à ces états représentatifs. Ainsi les fonctions nommées *radial basis function* ou *RBF* (Richard Sutton et Andrew Barto 1998, §8.3) définissent $\phi$ telle que :

$$\phi_{x,u}(i) = \exp\left(-\frac{\|(x,u) - c_i\|^2}{2\sigma_i^2}\right) \tag{2.31}$$

où $\sigma_i$ est un paramètre associé à $c_i$ déterminant la taille de la zone d'influence de $c_i$.

Une autre méthode à base d'états représentatifs consiste à utiliser une triangularisation de l'espace d'états ($\mathcal{X} \subset \mathbb{R}^n$) par un ensemble de points $c_i$ associés aux paramètres $\theta(i)$ (D. Bertsekas et J. Tsitsiklis 1996, Rémi Munos 1997). $\phi_x(i)$ est alors définie par les coordonnées barycentriques de l'état $x$ par rapport aux $n+1$ points du simplexe qui contient $x$.

Il existe aussi des méthodes utilisant d'autres fonctions comme par exemple une influence inversement proportionnelle à la distance (J.H. Kim et al. 1997) ou une influence calculée par la distance de Hamming (nombre de bits qui diffèrent) (Sridhar Mahadevan et Jonathan Connell 1991).

On peut citer aussi la méthode non linéaire d'approximation par cartes auto-organisatrices de Kohononen (Samira Sehad 1996, Claude Touzet 1999). Les états représentatifs sont des neurones qui peuvent être déplacés dans l'espace selon des règles de voisinage

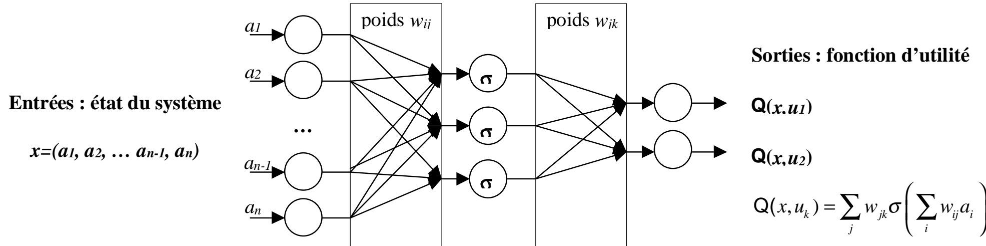



FIG. 2.20 – *Exemple de perceptron multicouche à une couche cachée.*

entre neurones. Un état intermédiaire active le neurone le plus proche qui renvoie la valeur d'utilité ou la commande à exécuter.

#### Discussion

Les méthodes d'approximation linéaires permettent de réduire sensiblement le nombre de valeurs à optimiser lors d'un apprentissage. Néanmoins, cette réduction est relative à la précision que demande la commande du système. Typiquement, les espaces d'états des systèmes commandés sont de dimension deux ou trois.

Par ailleurs, les différentes fonctions et notamment le CMAC sont des outils simples et facilement adaptables à n'importe quel système. De plus, la convergence de l'apprentissage est assurée, mais rien n'indique que la stratégie obtenue est optimale. Ces méthodes permettent enfin une généralisation locale fondée sur le voisinage qui peut dans certains cas accélérer l'apprentissage.

En conclusion, 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).

### 2.6.3 Fonctions d'approximation non linéaires

Le *perceptron multicouche* ou *PMC* est sans doute l'architecture non linéaire la plus employée en apprentissage par renforcement (*cf.* figure 2.20). Les PMC ont plusieurs avantages qui justifient leur emploi. Tout d'abord, les PMC sont des approximateurs universels : ils sont capables d'approcher avec une précision arbitraire n'importe quelle fonction régulière sur un intervalle borné. Ensuite, ils ont de grandes capacités de généralisation pouvant accélérer l'apprentissage. Enfin, leur structure est extrêmement compacte, ce qui leur permet d'approcher des fonctions sur des espaces à très grandes dimensions avec peu de paramètres.

Lin Long-Ji (1992) propose une version connexionniste de l'algorithme du Q-Learning baptisée *Q-Con*. Cette architecture utilise un PMC à une couche cachée et un seul neurone de sortie par commande. Un seul réseau est donc remis à jour à l'issue d'une commande en utilisant comme erreur la différence temporelle. G. Rummery et M. Niranjan (1994, 1995) ont ensuite proposé une extension du Q-Con à l'algorithme du Q($\lambda$) et l'ont

appliquée au contrôle d'un robot mobile. Lin Long-Ji (1993b) a aussi proposé une autre implémentation utilisant un réseau récurrent de type Elmann.

Au-delà de ces quelques implémentations initiales, il ne se dégage pas vraiment d'implémentation méthodique de l'apprentissage par renforcement connexionniste. Tous les exemples de cas de réussite sont issus d'une adaptation assez empirique des PMC et du TD($\lambda$) à l'application. On peut citer notamment :

- le célèbre TD-Gammon de Gerald Tesauro (1992, 1994,1995),
- la gestion d'une flotte d'ascenseurs par Robert Crites et Andrew Barto (1996),
- l'ordonnancement de tâches de Wei Zhang et Thomas Dietterich (1995).

Les points communs de ces trois exemples sont le grand nombre d'états des systèmes à contrôler ($> 10^{20}$ pour le TD-Gammon, $> 10^{22}$ pour la gestion d'ascenseurs) et le grand nombre d'essais avant d'obtenir une bonne stratégie (1 500 000 parties pour le TD-Gammon, simulation équivalente à 60 000 heures de fonctionnement des ascenseurs).

Des implémentations connexionnistes ont aussi été testées sans succès sur le jeu de Go par Nicol Schraudolph, Peter Dayan et Terrence Sejnowski (1994) et sur le jeu d'échecs par Sebastian Thrun (1995).

Ces résultats mitigés peuvent s'expliquer par l'absence de garantie de convergence des algorithmes de renforcement utilisant des réseaux de neurones. Sebastian Thrun et Anton Schwartz (1993) ainsi que J. Boyan et A. Moore (1995) ont montré que ces algorithmes sont peu robustes car ils sont très sensibles aux erreurs d'approximation. D'autres raisons peuvent être avancées, comme la gestion des connaissances acquises et désapprises ou la difficile interprétation du fonctionnement de ces réseaux qui rend délicate leur mise au point.

#### Discussion

Les réseaux de neurones ont prouvé que, dans certaines applications, ils permettent d'obtenir un excellent apprentissage alors que l'espace d'états est de grande dimension. Néanmoins, il n'existe pas de preuve de convergence des algorithmes, ni de méthode bien définie pour construire un réseau adapté à un système donné. Ces deux inconvénients majeurs, s'il ne sont pas définitifs, ont poussé les chercheurs vers d'autres voies.

### 2.6.4 Apprentissage de fonctions d'approximation

Afin de ne pas avoir à définir *a priori* une structure d'approximation, plusieurs approches construisent cette structure en fonction de l'expérience acquise.

On peut par exemple envisager une discrétisation dynamique de l'espace avec un pas plus fin là où une plus grande précision est nécessaire. Ces approches utilisent un arbre de décision pour diviser récursivement l'espace d'états en régions. Dans ce sens, le *parti-game* d'Andrew Moore et Christopher Atkeson (1993) génère une discrétisation à résolution variable d'un espace continu à dynamique déterministe à partir de données observées. Sur ce sujet, on peut citer aussi les travaux de David Chapman et Leslie Kaelbling (1991), d'Andrew McCallum (1995) ainsi que ceux Rémi Munos (1997).

Une autre approche consiste à représenter une stratégie par une base de règles enrichie au fur et à mesure de l'apprentissage (Seydina Ndiaye 1999). Une règle a un poids $w$ et associe une région (généralement un pavé) de l'espace d'états à une région (ou pavé) de l'espace de commandes. La commande est choisie au hasard dans la région de l'espace de commandes associée à la région de l'espace d'états contenant l'état actuel et de plus fort poids.

Enfin, la fonction d'utilité peut être approximée par des règles à inférence floue (Pierre-Yves Glorennec 1994, Min-Soeng Kim, Sun-Gi Hong et Ju-Jang Lee 1999).

### 2.6.5 Synthèse

Les algorithmes d'apprentissage par renforcement utilisant des fonctions d'approximation linéaires sont adaptés à la commande de systèmes continus dont la dimension reste modeste. Ils permettent de réduire le nombre de paramètres à optimiser et leur convergence est garantie (mais pas forcément vers la fonction d'utilité optimale).

Pour les espaces de très grande dimension, il est possible d'utiliser des structures type réseaux de neurones artificiels, mais la convergence n'est plus garantie et les méthodes sont relativement empiriques.

## 2.7 Conclusion

Dans ce chapitre, nous avons présenté les principaux algorithmes d'apprentissage par renforcement. Nous avons étudié leurs avantages et leurs inconvénients à la lumière de notre problématique. Ces algorithmes élémentaires sont clairement limités à des systèmes dont l'espace d'états est de faible dimension. Or, l'application que nous voulons commander a un espace d'états discret de grande dimension. Il est donc nécessaire de trouver une alternative adaptée à notre système, parmi les méthodes utilisant d'autres structures comme les fonctions d'approximation, les réseaux de neurones artificiels ou toute autre structure permettant l'apprentissage du contrôle de systèmes complexes.

Afin de préciser le choix de notre axe de recherche, nous avons synthétisé l'adéquation entre la structure (tableaux, fonctions d'approximation, *etc.*) à employer et les caractéristiques de l'espace d'états sous forme d'un arbre. La figure 2.21 présente l'algorithme de choix de la structure à utiliser en fonction de deux critères : la dimension et la nature de l'espace d'états.

Nous avons ajouté aux structures présentées dans ce chapitre les structures hiérarchiques. Ces architectures offrent une alternative aux réseaux de neurones. Leur principe est de découper le problème de commande globale en sous-problèmes plus rapides à apprendre. Parmi ces structures hiérarchiques, on trouve les architectures utilisant des comportements multiples présentées dans le chapitre suivant.

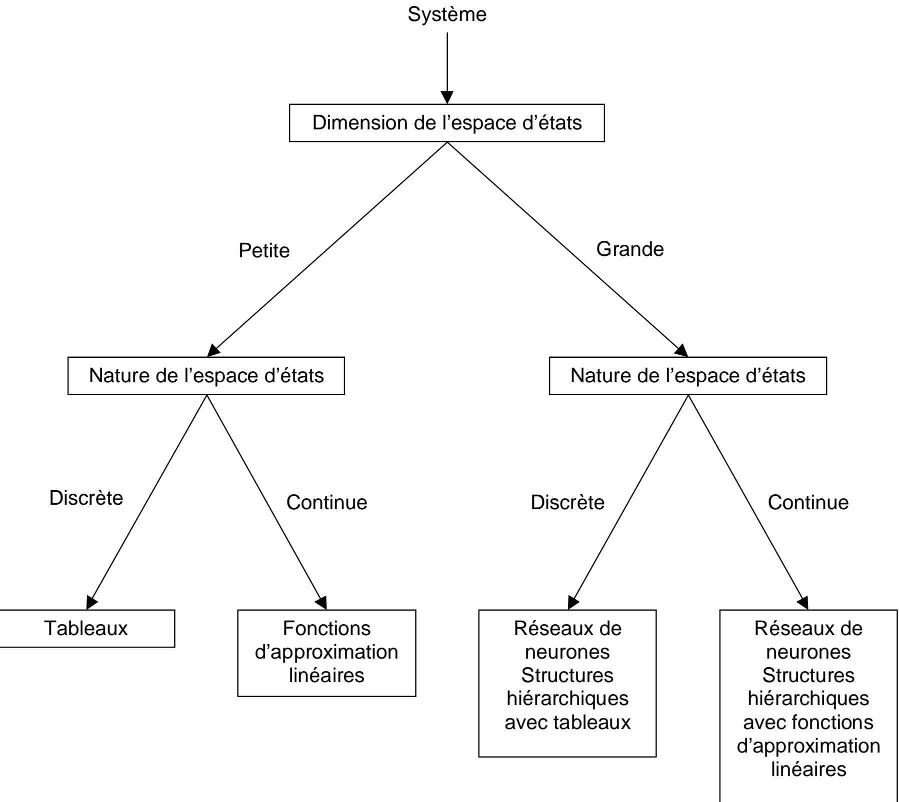



FIG. 2.21 – *Algorithme de choix de la structure en fonction de la dimension et de la nature de l'espace d'états du système.*

# Chapitre 3

# Architectures comportementales et apprentissage par renforcement

*Ce chapitre présente une manière de réduire la complexité d'un espace d'états à l'aide des architectures comportementales. Les principales architectures sans apprentissage puis les algorithmes d'apprentissage par renforcement qui utilisent ce découpage comportemental sont successivement exposés. Après une conclusion sur les méthodes comportementales, ce chapitre se termine par la définition de nos objectifs de recherche.*

## 3.1 Introduction

Les *architectures comportementales* [1] exploitent l'idée que le comportement global d'un robot ou autre système peut être réalisé par la *coordination* de plusieurs comportements élémentaires plus simples. Prenons l'exemple classique d'un robot mobile dont le comportement est d'atteindre un but ou de se déplacer au hasard si le but n'est pas visible, tout en évitant les obstacles qu'il rencontre. Ce comportement global peut être vu comme la coordination de trois comportements : atteindre un but, se déplacer au hasard et éviter les obstacles (*cf.* figure 3.1). Dans la suite, un comportement désigne un module fonctionnel autonome pouvant recevoir des informations en entrée, prendre des décisions et produire des commandes ou autres actions en sortie.

Cette méthode permet de réduire la complexité du problème global en le divisant en sous-problèmes plus simples à résoudre. Cette simplification n'est qu'apparente car la difficulté majeure est alors de définir une fonction de coordination.

Il existe deux grandes familles de techniques de coordination : les *architectures compétitives* et les *architectures coopératives*. Dans les premières, la commande effectuée est celle voulue par un unique comportement qui a été désigné par un processus de sélection [2]. Dans les secondes, la commande finale est le résultat d'un compromis ou d'une

1. Termes anglais : behaviour-based architectures.
2. C'est le principe du « gagnant rafle la mise » (the winner takes all).

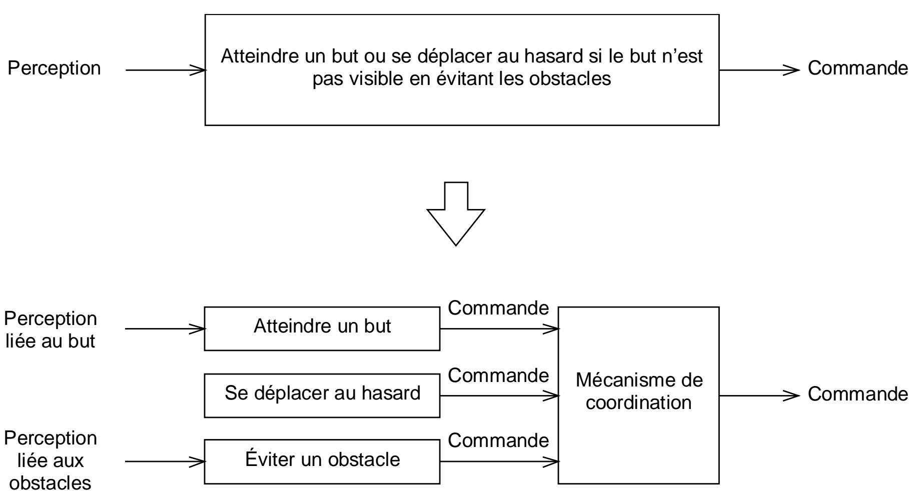



FIG. 3.1 – *Exemple d'architecture comportementale.*

fusion entre les commandes proposées par les différents comportements qui peuvent donc être tous actifs en même temps.

Beaucoup d'architectures comportementales ont été développées notamment dans le domaine de la robotique mobile. Nous nous limiterons à l'étude de quatre architectures caractéristiques. Ces descriptions sont fondées sur les articles de référence ainsi que sur les études comparatives de Toby Tyrrell (1993), Lennart Pettersson (1997), Paolo Pirjanian (1998), Ronald Arkin (1998), Pierre Arnaud (2000) et Nadine Richard (2001).

Ce chapitre présente deux architectures compétitives, puis décrit deux architectures coopératives. Il aborde enfin les architectures comportementales utilisant l'apprentissage par renforcement.

## 3.2 Architectures compétitives

### 3.2.1 Introduction

Le principe des architectures compétitives est fondé sur la sélection temporaire d'un unique comportement qui va commander le système pendant une durée déterminée. Le système de compétition peut être défini par un jeu de priorités fixes comme dans l'*architecture à subsomption*. Le système de compétition peut aussi être dynamique. Par exemple, l'*architecture à sélection d'actions* de Pattie Maes sélectionne le comportement qui a le plus grand niveau d'activité, ce niveau étant recalculé en permanence.

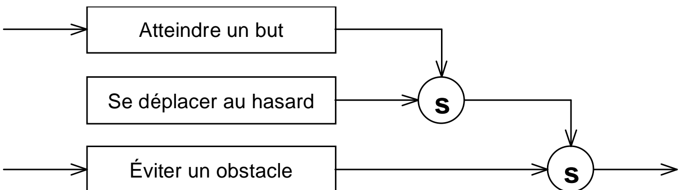



Fig. 3.2 – *Exemple d'architecture à subsomption.*

### 3.2.2 Architecture à subsomption

#### Principe

A la fin des années 1980, Rodney Brooks (1986) a introduit l'une des premières architectures comportementales : l'architecture à *subsomption*[3] .

Cette architecture est constituée de comportements indépendants et asynchrones organisés en couches appelées *niveaux de compétence*. Les couches supérieures regroupent les compétences de haut niveau (par exemple la planification) tandis que les couches inférieures regroupent les comportements simples de bas niveau (évitement d'obstacles). Les comportements d'une couche peuvent subsumer les comportements d'une couche inférieure à l'aide de deux mécanismes : l'*inhibition* et la *suppression*, représentés par des nœuds à deux entrées sur la figure 3.2. L'inhibition consiste à empêcher la diffusion de la sortie d'un comportement inférieur. La suppression permet de remplacer la sortie d'un comportement inférieur par celle du comportement supérieur.

Les expériences menées par Rodney Brooks et ses collaborateurs ont montré que cette architecture permet effectivement de construire des robots capables d'évoluer dans le monde réel (Rodney Brooks 1990, Pattie Maes et Rodney Brooks 1990).

#### Discussion

La principale critique de cette architecture concerne la difficulté à concevoir, à maintenir et à faire évoluer un système cohérent. En effet, les modules de comportements ne peuvent pas être conçus de manière complètement indépendante. Les comportements supérieurs doivent espionner les couches inférieures pour permettre une bonne coordination. Par exemple, le comportement « atteindre un but » doit savoir si un obstacle est présent ou non pour pouvoir supprimer ou non le comportement « éviter un obstacle ». L'implémentation de comportements tenant compte de l'état des autres comportements est presque aussi complexe que l'implémentation d'un contrôleur unique résolvant le même problème.

La volonté originale de Rodney Brooks d'associer priorité et niveaux de complexité est très critiquée notamment par Pierre Arnaud (2000, p. 86). Rodney Brooks (1989) lui-même a d'ailleurs discrètement introduit un nouveau nœud nommé *disqualification*[4] permettant à un comportement inférieur de supprimer un comportement supérieur.

3. Terme anglais : subsumption.
4. Terme de Rodney Brooks : defaulting.

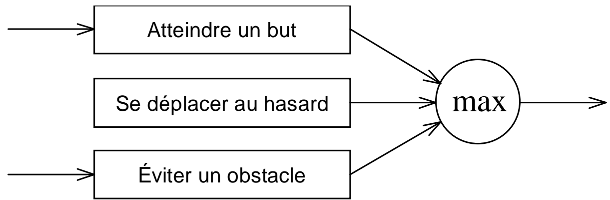



FIG. 3.3 – *Exemple simplifié de l'architecture à sélection d'actions.*

### 3.2.3 Architecture à sélection d'actions de Pattie Maes

#### Principe

L'architecture à sélection d'actions de Pattie Maes (1989) est aussi constituée de modules comportementaux distribués. Contrairement à la subsomption, il n'y a pas de priorité pré-établie, la sélection du comportement à suivre est réalisé dynamiquement.

Chaque module possède un niveau dynamique d'activation et une liste de pré-conditions nécessaire à son exécution. Il fournit en sortie une liste de conséquences positives et une liste de conséquences négatives. Les modules calculent leur niveau d'activation en fonction des connexions inter-modules reliant les sorties aux entrées et des règles de propagation de l'activation. Ce processus a pour but de favoriser une augmentation de l'activation des modules les plus adéquats à la situation.

La sélection du comportement dont la commande sera exécutée est déterminée par le niveau d'activation. Le module exécutable dont le niveau d'activation est le plus élevé est activé. En simplifiant et en omettant les connexions inter-modules, l'architecture à sélection d'actions peut être résumée par le schéma de la figure 3.3 pour l'exemple du robot mobile.

#### Discussion

Pour Toby Tyrrell (1993, p. 176), cette architecture est efficace pour gérer les situations d'urgence (comme l'évitement d'obstacles) ou obtenir un comportement opportuniste (comme atteindre un objectif proche). En revanche, le système a tendance à rentrer dans des cycles oscillants. Par exemple, dans une impasse près d'un objectif attractif, un robot mobile peut osciller indéfiniment entre le comportement « éviter un obstacle » et le comportement « atteindre un but ». Le robot peut aussi hésiter indéfiniment entre deux buts différents à atteindre. Par ailleurs, une architecture comportant un grand nombre de modules est difficile à réaliser sans provoquer des conflits internes entre comportements.

### 3.2.4 Synthèse

Les architectures compétitives montrent une bonne efficacité notamment pour gérer les situations d'urgence, mais dans des situations complexes ces architectures peuvent entrer dans des cycles oscillants.

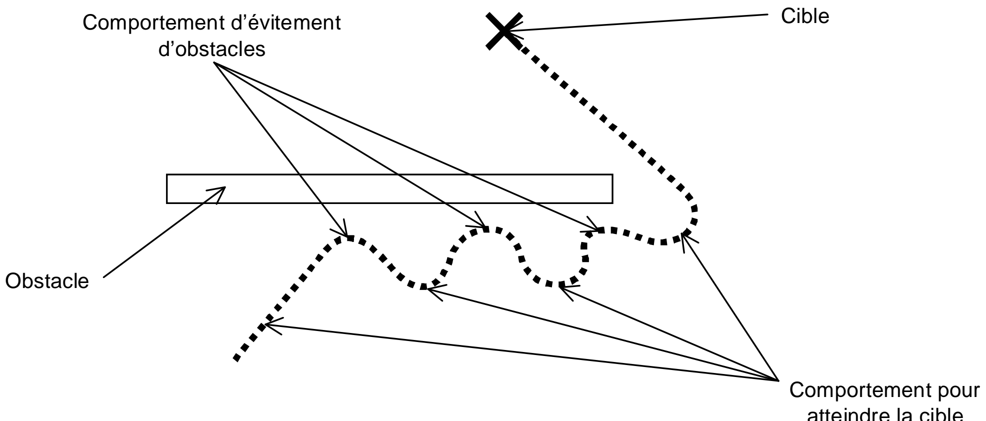



FIG. 3.4 – *Exemple de trajectoire classique produite par une architecture compétitive.*

Les trajectoires obtenues peuvent être saccadées si deux comportements produisant des commandes très différentes entrent en compétition et se trouvent activés l'un à la suite de l'autre (*cf.* figure 3.4).

En théorie, les architectures compétitives sont très souples puisque les modules sont indépendants les uns des autres, mais dans la pratique il est difficile de respecter complètement cette indépendance.

## 3.3 Architectures coopératives

### 3.3.1 Introduction

Le principe des architectures coopératives est de faire participer tous les comportements à l'élaboration de la commande. Cette coopération se fait généralement selon l'un des deux principes suivants. La première idée est d'utiliser un opérateur de fusion comme la somme pour agréger les commandes proposées par chacun des comportements. La nouvelle commande produite peut alors être complètement inédite. Les *architectures orientées schémas* utilisent ce principe. La seconde idée est de proposer à chaque comportement un choix (discret) de commandes possibles, de recueillir les préférences de chaque comportement puis de choisir la commande réalisant le meilleur compromis. L'*architecture DAMN* exploite cette idée.

### 3.3.2 Architecture orientée schémas

#### Principe

L'architecture orientée schémas a été développée à partir de la *théorie des schémas*. Cette théorie a été introduite au début du siècle pour représenter le fonctionnement du cerveau. Les schémas sont des entités autonomes et représentent l'unité de base du comportement.

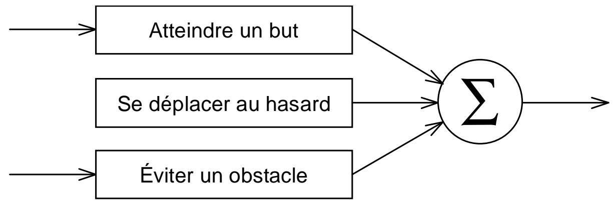



FIG. 3.5 – *Exemple d'architecture orientée schémas.*

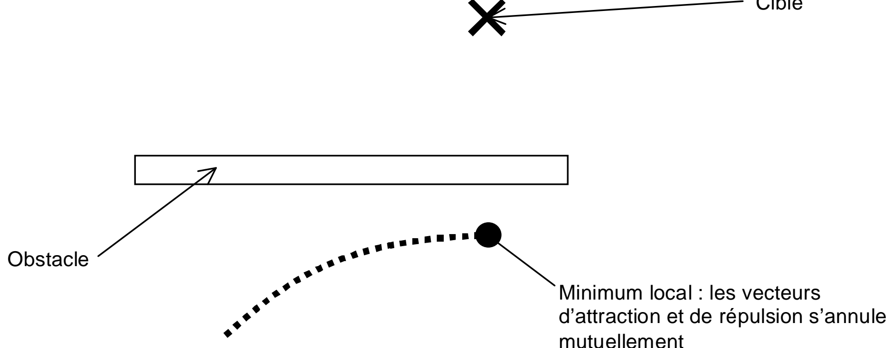



FIG. 3.6 – *Exemple de trajectoire classique produite par une architecture orientée schéma.*

Michael Arbib (1981) puis Ronald Arkin (1989) ont appliqué cette théorie à la navigation de robots autonomes. Leurs architectures sont constituées de schémas moteurs indépendants et fonctionnant en parallèle. Les schémas moteurs sont des schémas particuliers qui génèrent un vecteur de commande complet pour le robot. Ce vecteur est calculé à l'aide de méthodes à base des champs de potentiels (Oussama Khatib 1986). Généralement, les obstacles produisent des champs répulsifs tandis que les cibles génèrent des champs attractifs. Contrairement à la planification qui nécessite le calcul complet du champ vectoriel, chaque schéma ne calcule que la valeur instantanée du champ à la position courante du robot.

La coordination des schémas moteurs est simple : il suffit d'additionner l'ensemble des vecteurs pondérés par une valeur dynamique associée à chaque schéma (*cf.* figure 3.5). Le vecteur obtenu correspond à la commande envoyée aux actionneurs. Cette commande est donc le résultat de la fusion de l'ensemble des commandes proposées.

### Discussion

Cette architecture est modulaire, simple à mettre en œuvre et complètement distribuée. Les stratégies de commande obtenues sont très lisses grâce à l'utilisation des champs de potentiels (*cf.* figure 3.6).

En revanche, le résultat de la fusion des vecteurs de commande peut engendrer des commandes absurdes. Par exemple, si l'attraction d'un but contrebalance exactement la répulsion d'un obstacle, le vecteur de commande est nul et le robot ne bouge plus.

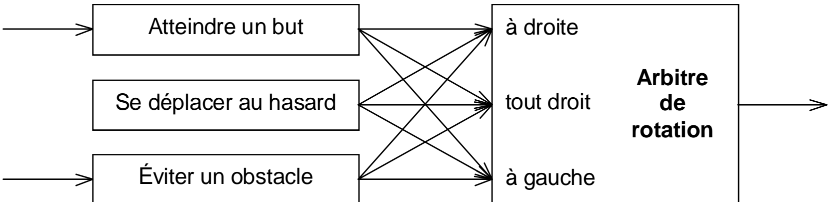



FIG. 3.7 – *Exemple d'arbitre DAMN.*

Cette méthode est particulièrement sensible aux problèmes de maxima locaux et de formation de cycles oscillants. Néanmoins, l'ajout de bruit dans le champ (Ronald Arkin 1998) ou d'une mémoire à court terme des endroits dernièrement visités (Tucker Balch et Ronald Arkin 1993) permet de limiter ces risques.

Enfin, dans des systèmes plus complexes, il peut se révéler difficile de calculer les champs de potentiels et de régler le poids de chaque schéma.

### 3.3.3 Architecture DAMN

DAMN (Distributed Architecture for Mobile Navigation) est une architecture distribuée coopérative développée par Julio Rosenblatt pour la navigation de robots mobiles (Julio Rosenblatt et David Payton 1989, Julio Rosenblatt 1997a, Julio Rosenblatt 1997b).

Cette architecture est constituée de modules comportementaux indépendants et d'*arbitres*. Chaque arbitre est responsable de la sélection d'une valeur pour une variable caractéristique de la commande comme par exemple la direction ou la vitesse du robot. Les arbitres proposent un nombre fini de valeurs pour la variable dont ils sont responsables. Un arbitre peut aussi encapsuler des actions abstraites.

Contrairement aux comportements des architectures précédentes, chaque comportement ne propose pas une unique variable en sortie, mais vote en fonction de son propre intérêt pour chaque commande proposée par les arbitres à l'aide d'une note (par exemple comprise entre $-1$ et 1). Chaque arbitre calcule alors une moyenne des notes pour chacune des commandes proposées, chaque note étant pondérée par le poids du comportement émetteur. La commande comptabilisant le plus grand score est sélectionnée pour la variable correspondante : cette commande correspond à un compromis entre les différents comportements. La pondération des votes correspond à la priorité du comportement et peut être dynamique. Contrairement à Rodney Brooks, Julio Rosenblatt donne la priorité maximale aux comportements de bas niveau comme l'évitement d'obstacles.

Julio Rosenblatt a expérimenté cette architecture sur le pilotage d'un véhicule en environnement extérieur et en temps réel. Il a montré que son architecture produit des stratégies de commande permettant d'atteindre un objectif tout en évitant les obstacles avec de bonnes trajectoires.

Toby Tyrrell (1993) a comparé plusieurs architectures comportementales dont la sélection d'actions et l'architecture DAMN sur une simulation. Les résultats expérimentaux montrent que l'architecture DAMN est supérieure selon tous les critères de performance testés.

Récemment, Julio Rosenblatt (2000) a proposé une version de DAMN fondée sur l'espérance de gain de chaque valeur des commandes. Chaque comportement calcule par programmation dynamique à l'aide d'un modèle la valeur d'utilité de chaque possibilité proposée par les arbitres par rapport à son état actuel. Ces valeurs remplacent directement les notes. Ce principe permet de choisir les commandes qui maximisent leur utilité. Les trajectoires obtenues sont encore plus courtes et plus directes.

#### Discussion

L'architecture DAMN allie la robustesse des approches compétitives et l'optimisation des trajectoires des approches orientées schémas. En effet, l'utilisation d'un arbitre ne produit jamais de commande absurde non désirée et permet au contraire de choisir le meilleur compromis. De plus, cette architecture est très modulaire et permet un mélange facile entre comportements de haut et de bas niveaux.

En revanche, elle ne résout pas le problème de maxima locaux et de formation de cycles. Les exemples donnés par Julio Rosenblatt ou Toby Tyrrell ne présentent jamais de cas limites comme par exemple une impasse près d'un objectif. Enfin, cette architecture est limitée à des systèmes dont la commande est discrète.

### 3.3.4 Synthèse

Les architectures coopératives génèrent des trajectoires plus lisses et souvent plus directes que les architectures compétitives. Elles sont complètement modulables car les comportements sont réellement indépendants. Enfin, les architectures orientées schémas sont moins robustes car elles peuvent produire des maxima locaux et entrer plus facilement dans des cycles infinis.

## 3.4 Application à l'apprentissage par renforcement

### 3.4.1 Subsomption et apprentissage par renforcement

Sridhar Mahadevan et Jonathan Connell (1991) ont implémenté une architecture à subsomption avec des modules comportementaux utilisant le Q-Learning. L'application visée est de déplacer une boîte en la poussant avec un robot mobile.

Pour réaliser cette tâche, le robot apprend en parallèle trois comportements : éviter de se coincer, pousser la boîte et chercher la boîte. Ces comportements se subsument à l'aide de nœuds suppresseurs. Sridhar Mahadevan et Jonathan Connell donnent la priorité à l'action d'éviter de se coincer puis de pousser la boîte et enfin de chercher la boîte. Chaque comportement utilise l'algorithme du Q-Learning conjugué à une fonction d'approximation (distance de Hamming ou clustering dynamique).

Les résultats de la simulation et des tests expérimentaux montrent clairement que cette architecture est plus performante que l'architecture monolithique équivalente. L'architecture développée et notamment la version utilisant la distance de Hamming convergent beaucoup plus vite et atteignent des résultats supérieurs en terme de taux de poussée par période d'échantillonnage.

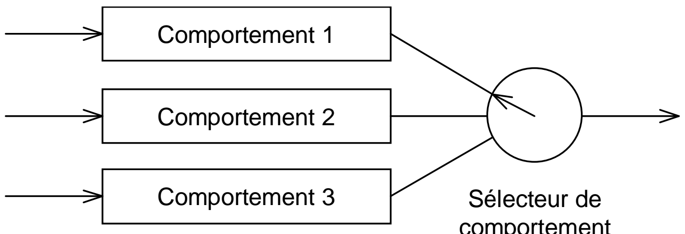



FIG. 3.8 – *Exemple d'architecture à sélecteur.*

### 3.4.2 Sélection d'actions et apprentissage par renforcement

Beaucoup de chercheurs se sont inspirés de l'architecture à sélection d'actions pour réaliser des commandes distribuées par apprentissage par renforcement. Le principe de base de ces architectures est de disposer de plusieurs modules distribués dont le comportement est déterminé par apprentissage. La coordination est ensuite réalisée par un sélecteur [5] qui choisit un unique comportement à chaque instant selon des critères variables avec les algorithmes. Cette sélection peut aussi être issue d'un processus d'apprentissage.

#### Hierarchical Learning

L'architecture de Lin Long-Ji (1993a) utilise des comportements obtenus par apprentissage ainsi qu'un sélecteur réalisé lui aussi par apprentissage. Cette architecture a été appliquée à la commande d'un robot devant retrouver son chargeur de batterie dans des bureaux. Dans ce but, Lin Long-Ji a utilisé trois comportements : suivre un mur, passer une porte et atteindre le chargeur.

Ces comportements sont appris séparément en utilisant l'algorithme du Q-Learning. Chaque comportement perçoit une partie ou la totalité de l'état du système, peut effectuer les commandes de déplacement du robot et reçoit des récompenses propres à son objectif. Ces comportements sont ensuite figés et le sélecteur apprend à choisir un des comportements par rapport à l'état global du système pour obtenir les meilleures récompenses globales possibles.

Yassine Faihe (1999) a appliqué cette approche à la simulation d'un robot devant collecter des lettres dans 3 bureaux différents, aller les poster régulièrement et aller se recharger si nécessaire.

Le problème majeur de cette architecture est qu'elle ne simplifie pas forcément l'apprentissage puisque les sélecteurs apprennent à associer un comportement à chaque état global du système. Il n'y a donc pas de simplification au niveau de la complexité, mise à part une légère réduction du nombre de commandes (égal au nombre de comportements).

L'autre critique concerne l'apprentissage des comportements. En effet, une fois appris, les comportements sont figés et le robot ne s'adapte pas si le système évolue.

5. L. Kaelbling, M. Littman et A. Moore (1996) utilisent les termes « Gated behaviour » pour désigner les architectures utilisant un sélecteur de comportement.

#### Compositional Q-Learning (C-QL)

L'approche de Satinder Singh (1992) consiste à coordonner plusieurs comportements élémentaires correspondant à une décomposition temporelle d'une tâche complexe à l'aide de sous-buts. Les comportements élémentaires sont d'abord entraînés à réussir le sous-but qui leur est attribué, puis le sélecteur apprend à choisir le bon comportement au bon moment pour réussir l'objectif global. Cette méthode a été utilisée avec succès par Chen Tham et Richard Prager (1994) pour apprendre à piloter la simulation d'un bras robot multiaxial. Cette approche effectue une décomposition temporelle qui permet d'enchaîner des stratégies de commande différentes, mais ne réduit pas la complexité de l'espace d'états. En effet, les comportements élémentaires et le sélecteur réalisent un apprentissage par rapport à l'espace d'états global du système.

### 3.4.3 Autres approches

#### Coordination hybride

Le but de cette architecture développée par M. Carreras, J. Battle et P. Ridao (2001) est d'utiliser les avantages des architectures compétitives et des architectures coopératives. Le sélecteur de comportements n'est pas réalisé par apprentissage mais utilise selon le cas un mécanisme type sélection d'actions ou un mécanisme de fusion type schémas moteurs.

L'application visée est le contrôle d'un sous-marin dont le but est de trouver et d'atteindre une cible tout en évitant les obstacles. Chaque comportement réalise un apprentissage de sa tâche à l'aide de récompenses personnalisées mais l'apprentissage est réalisé en permanence. A chaque période d'échantillonnage, les comportements calculent leur niveau d'activation en fonction de lois fixes et sans apprentissage. Si aucun comportement n'atteint le niveau d'activation 1, la commande effectuée est une combinaison des vecteurs proposés par chaque comportement. En revanche, si un comportement atteint ce niveau d'activation alors il est seul maître à bord !

Des tests ont été réalisés sur une simulation. Ils montrent que la stratégie obtenue est robuste et bien optimisée. Le principal inconvénient de cette approche est de devoir déterminer les lois d'activation de chaque comportement.

#### Hierarchical Distance to Goal (HDG)

Introduite par Leslie Kaelbling (1993a), cette architecture est dédiée à l'apprentissage d'un robot mobile qui doit atteindre un état objectif dans un espace d'états de grande dimension. Pour ce faire, Leslie Kaelbling propose de partitionner l'espace d'états en plusieurs régions. Au sein d'une région, un module bas niveau utilise un algorithme inspiré du Q-Learning[6] pour apprendre à atteindre n'importe quel état de la région.

Si le robot se trouve dans la même région que l'état objectif, alors le module bas niveau est activé avec pour but d'atteindre directement l'état objectif. Si le robot ne se trouve pas dans la même région, alors l'algorithme cherche parmi les régions voisines

6. Nommé DG-Learning par Leslie Kaelbling.

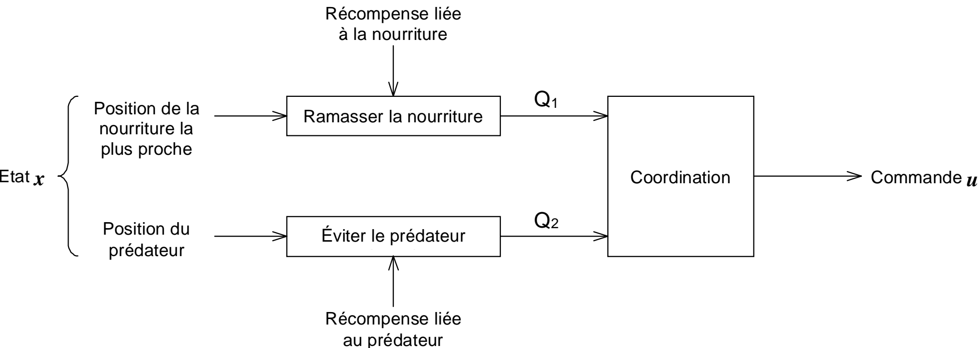



FIG. 3.9 – *Exemple d'architecture W-Learning dans le cas d'un robot dont le but est de ramasser de la nourriture en évitant un prédateur mobile.*

celle qui est la plus proche de la région objectif, puis active le module de bas niveau en lui donnant comme but un état à la frontière entre la région du robot et la région voisine sélectionnée.

L'avantage de cette architecture est de pouvoir réutiliser le comportement appris sur une région dans une autre région. Il suffit alors d'apprendre à atteindre n'importe quel état dans une petite région pour être capable d'atteindre n'importe quel objectif dans l'espace d'états entier. Cette architecture permet donc de réduire la complexité du système. Mais cette solution est très limitée. Par exemple, il faut bien connaître le système pour définir sa partition et si le comportement du système n'est pas homogène sur l'espace d'états (un mur inattendu par exemple), alors cette architecture ne fonctionne plus.

### 3.4.4 Une architecture originale : le W-Learning

#### Principe

En 1995, Mark Humphrys a introduit une architecture comportementale particulièrement intéressante appelée *W-Learning* [7] (Mark Humphrys 1995, Mark Humphrys 1996a, Mark Humphrys 1996b).

Cette architecture est constituée de plusieurs modules d'indice $i$ percevant chacun une partie du vecteur d'état $x$. Chaque module utilise l'algorithme du Q-Learning pour calculer en sortie son espérance de gain $\mathsf{Q}_i(x,u)$ pour chaque commande possible $u$. Un module de coordination détermine la commande à effectuer en fonction des valeurs de $\mathsf{Q}_i$. Chaque module de comportement perçoit alors une récompense liée à sa perception pour remettre à jour sa fonction d'utilité.

La figure 3.9 propose un exemple d'architecture W-Learning pour une version simplifiée d'un des systèmes testés par Mark Humphrys : un robot doit collecter de la nourriture en évitant un prédateur mobile [8]. Supposons que le robot puisse effectuer quatre com-

7. En rapport avec $W$ qui représente les poids des comportements (Weight en anglais).

8. 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.

| | Haut | Bas | Droite | Gauche | |
|---|---|---|---|---|---|
| Ramasser nourriture ($\mathsf{Q}_1$) | 0,1 | 0,6 | 0,7 | 0,3 | |
| Éviter prédateur ($\mathsf{Q}_2$) | 0 | 0 | $-1$ | $-1$ | |
| Maximiser la meilleure espérance | 0,1 | 0,6 | 0,7 | 0,3 | $\Rightarrow u$=droite |
| Minimiser le pire regret | 0,6 | 0,1 | 1 | 1 | $\Rightarrow u$=bas |
| Maximiser l'espérance collective | 0,1 | 0,6 | $-0{,}3$ | $-0{,}7$ | $\Rightarrow u$=bas |
| Minimiser le regret collectif | 0,6 | 0,1 | 1 | 1,4 | $\Rightarrow u$=bas |

TAB. 3.1 – *Exemple de coordination pour l'exemple de la figure 3.9.*

mandes (haut, bas, droite, gauche), le but du module de coordination est de choisir la meilleure commande en fonction des informations fournies par les comportements (les espérances de gain que chaque comportement a d'effectuer telle ou telle commande).

Mark Humphrys propose quatre méthodes pour coordonner les comportements. Toutes ces méthodes utilisent les valeurs des fonctions d'utilité $\mathsf{Q}_i$ de chaque comportement. La commande est choisie selon un des quatre principes suivants :

– maximiser la meilleure espérance[9] :

$$u = \max_{v \in \mathcal{U}} \max_i \mathsf{Q}_i(x,v) \tag{3.1}$$

– minimiser le pire regret[10] (avec $v_i = \arg\max_{v \in \mathcal{U}} \mathsf{Q}_i(x,v)$) :

$$u = \min_{v \in \mathcal{U}} \max_i \left(\mathsf{Q}_i(x,v_i) - \mathsf{Q}_i(x,v)\right) \tag{3.2}$$

– maximiser l'espérance collective[11] :

$$u = \max_{v \in \mathcal{U}} \sum_i \mathsf{Q}_i(x,v) \tag{3.3}$$

– minimiser le regret collectif[12] :

$$u = \min_{v \in \mathcal{U}} \sum_i \left(\mathsf{Q}_i(x,v_i) - \mathsf{Q}_i(x,v)\right) \tag{3.4}$$

Sur notre exemple, on peut imaginer le cas des valeurs de fonctions d'utilité du tableau 3.1. Le calcul de la commande selon les quatre méthodes donne des résultats différents.

Les deux premières méthodes font une coordination du type sélection d'actions. La commande choisie est issue d'un seul comportement. Les deux autres méthodes effectuent un compromis à la manière d'un arbitre DAMN utilisant les fonctions d'utilité.

9. Termes anglais : Maximize the best happiness.
10. Termes anglais : Minimize the worst unhappiness.
11. Termes anglais : Maximize collective happiness.
12. Termes anglais : Minimize collective unhappiness.

**Discussion**

Malgré l'originalité de la méthode, Mark Humphrys a très peu expérimenté cette architecture. Il l'a testée sur deux simulations seulement et donne peu de détails quand aux résultats expérimentaux.

On peut néanmoins dégager quelques conclusions générales. Tout d'abord, le principe général fonctionne et permet de réduire la complexité du système puisque chaque comportement effectue son apprentissage par rapport à une partie des variables d'états, donc dans un sous-espace de l'espace d'états.

Au niveau du choix des méthodes de coordination, la première ne permet pas au robot d'éviter un prédateur si ce comportement est en conflit avec la recherche de nourriture. En revanche, la seconde permet de gérer des comportements antagonistes car elle tient compte de toutes les espérances de gain positives comme négatives. Les deux autres méthodes permettent en plus de choisir le meilleur compromis entre les comportements à la manière d'un arbitre DAMN.

Quelques chercheurs ont testé cette architecture sur d'autres applications. Jackson Pauls (2001) a comparé les performances du W-Learning (avec la méthode « minimiser le pire regret ») et d'une architecture à sélection d'actions sans apprentissage sur une simulation baptisée *Pigs and People*. Cette simulation met en jeux plusieurs agents autonomes qui doivent survivre dans un environnement hostile (ressources limitées, prédateurs). Martin Hallerdal a expérimenté le W-Learning pour piloter un robot Khépéra[13] devant rapporter des objets à un point donné en évitant les obstacles (Martin Hallerdal 2001, Martin Hallerdal et John Hallam 2002). Jackson Pauls et Martin Hallerdal arrivent à la même conclusion : l'architecture est très difficile à spécifier notamment au niveau du nombre des comportements et des récompenses à leur attribuer. Dans les deux expériences, les résultats sont décevants et le W-Learning se révèle moins performant que la sélection d'actions pour l'application Pigs and People. A ce propos, on peut regretter que les opérateurs de fusion collectifs généralement plus performants n'aient pas été testés. Malgré ces résultats, Jackson Pauls et Martin Hallerdal ne remettent pas en cause leur intérêt pour le W-Learning, et Jackson Pauls suggère de rendre automatique la décomposition en comportements pour simplifier la phase de spécification.

### 3.4.5 Synthèse

Les architectures comportementales utilisant l'apprentissage par renforcement sont tributaires des avantages et inconvénients des architectures traditionnelles (robustesse, qualité de la trajectoire, modularité, *etc.*). Certaines structures permettent d'apprendre plus vite mais les approches comme le Hierarchical Learning et le C-QL ne réduisent pas la complexité du problème, car les comportements ou les sélecteurs utilisent l'espace d'états global du système.

Dans tous les cas, les récompenses sont associées à chaque comportement, ce qui limite l'architecture à des systèmes très bien cernés par l'utilisateur.

---

13. Mini-robot autonome développé par la société suisse K-TEAM.

Enfin, la division en comportements n'est pas automatique, ce qui est contraire à l'objectif même de l'apprentissage.

## 3.5 Conclusion

Les architectures comportementales permettent de réaliser la commande de systèmes complexes à l'aide de modules élémentaires plus simples à concevoir.

Les architectures compétitives sont souvent plus robustes mais produisent des trajectoires saccadées. A l'inverse, les architectures coopératives génèrent de belles trajectoires mais certaines architectures coopératives comme celles orientées schémas sont moins robustes. Parmi toutes ces architectures, l'arbitre DAMN semble être la meilleure solution de coordination. De manière générale, toutes ces architectures souffrent de la formation plus ou moins fréquente de cycles oscillants qui piègent le robot.

Beaucoup d'architectures utilisant l'apprentissage par renforcement ont été développées. Malgré son faible succès, le W-Learning est pour nous une des architectures les plus intéressantes. Elle permet de réduire considérablement la complexité de l'espace d'états et utilise une coordination proche de celle d'un arbitre DAMN. Pour la rendre plus souple, il serait nécessaire de rendre la décomposition automatique et de résoudre l'épineux problème de la définition des récompenses par rapport aux comportements.

Notre projet de recherche découle de ces conclusions. L'approche que nous avons développée a pour objectif de répondre à trois enjeux principaux :

- créer une architecture constituée de multiples comportements simples à apprendre dans le but de réduire la complexité du système,
- créer une architecture auto-constructive qui évite de devoir spécifier précisément les comportements et les récompenses associées à ces comportements,
- créer une architecture dont la fonction de coordination ne génère pas de maximum local ni de cycle oscillant.

Le chapitre suivant introduit la structure et le principe de fonctionnement de l'architecture développée. Cette architecture est validée au chapitre 5 sur l'exemple du labyrinthe. Le chapitre 6 présente une amélioration de la vitesse d'apprentissage de l'architecture obtenue en remplaçant l'algorithme d'apprentissage initial par un algorithme indirect. Enfin, le chapitre 7 propose une modification de la fonction de prise de décision de l'architecture qui permet d'éviter la création de maximum local.

# Chapitre 4

# Q-Learning parallèle

*Ce chapitre présente l'approche parallèle du Q-Learning en s'appuyant sur l'exemple du labyrinthe. Il introduit le principe général de cette approche, puis définit des notations et une architecture qui se veulent très ouvertes. Il explique en détail le fonctionnement du contrôleur parallèle et ouvre les problématiques générées par cette approche.*

## 4.1 Introduction

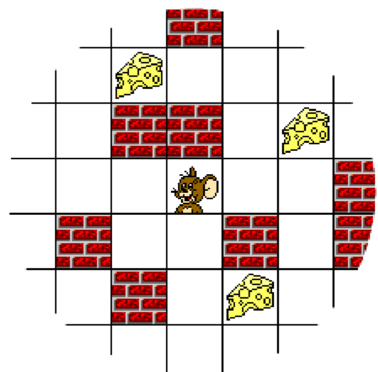
Le but de l'approche parallèle est de répondre aux trois objectifs énoncés précédemment : réduire la complexité du système, créer une architecture auto-constructive et coordonner les comportements sans générer de minimum local.

De plus, nous recherchions une approche rigoureuse, cohérente et efficace pour notre application tout en gardant une grande ouverture à l'évolution et à la réutilisation pour la commande d'autres systèmes. L'idée nous est venue de considérer l'état du système comme une juxtaposition de plusieurs états markoviens et indépendants décrivant le comportement d'un élément du système. Pris séparément, ces comportements élémentaires seraient particulièrement simples à apprendre. Il faut donc trouver le moyen d'apprendre en parallèle chaque comportement puis de regrouper ces informations pour décider des commandes à effectuer.

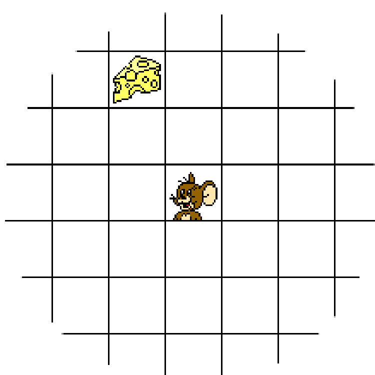
Ce chapitre se compose de trois parties. La première introduit le principe de base de l'architecture parallèle. La deuxième définit précisément le cadre de travail en proposant un vocabulaire spécifique. Enfin, la troisième partie explicite dans les grandes lignes le fonctionnement d'un contrôleur parallèle.

## 4.2 Idée fondatrice

Afin d'illustrer notre propos, nous utiliserons l'exemple simple du labyrinthe (*cf.* §2.4.2, page 22). Imaginons que la souris ait à retrouver des fromages dans un labyrinthe discret encombré de murs et dont la configuration puisse changer (*cf.* figure 4.1a). Comme



(a) Situation globale dans le repère lié à la souris



(b) Une perception élémentaire dans le repère lié à la souris

FIG. 4.1 – *Exemple de labyrinthe.*

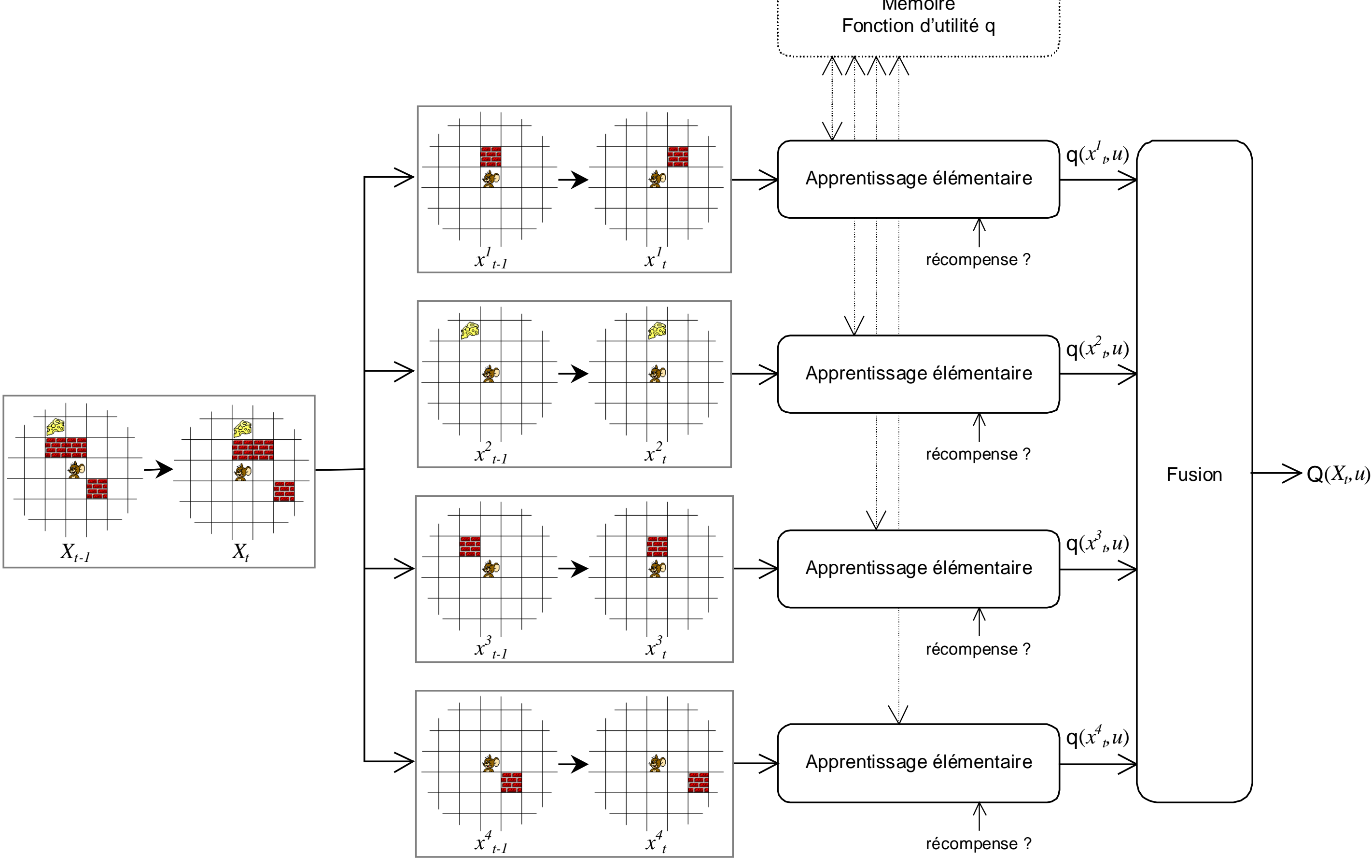
nous l'avons vu précédemment (*cf.* §1.1.3, page 3), l'espace d'états d'un tel système est de cardinal $m^n$ avec $m$ le nombre de positions possibles pour un objet par rapport à la souris et $n$ le nombre d'objets. Il n'est pas envisageable d'utiliser un algorithme conventionnel type Q-Learning dans ces conditions.

En revanche, s'il n'y avait qu'un seul objet (fromage ou mur) dans la grille (*cf.* figure 4.1b), la souris pourrait rapidement apprendre la stratégie optimale vis-à-vis de cet objet (atteindre le fromage ou éviter le mur). Par exemple, un contrôleur élémentaire utilisant l'algorithme du Q-Learning pourrait apprendre cette tâche. Dans ce cas, il optimiserait une fonction d'utilité élémentaire $\mathsf{q}$. Cette fonction fournirait l'espérance de gain d'effectuer une commande par rapport à l'état élémentaire englobant le type et la position d'un seul objet dans un repère lié à la souris (ce que nous appellerons la *perception* de l'objet).

Dans le cas d'un labyrinthe avec plusieurs objets, notre idée est de réaliser des apprentissages élémentaires en parallèle pour chaque objet, comme si l'objet était seul (*cf.* figure 4.2). Pour chaque objet, l'apprentissage élémentaire optimise la même fonction d'utilité élémentaire $\mathsf{q}$ en un point différent et génère une estimation de l'espérance de gain vis-à-vis de cet objet pour chaque commande. Avec ces estimations élémentaires, le contrôleur reconstruit une estimation de l'espérance de gain *globale* $\mathsf{Q}$ des commandes par rapport à l'ensemble des objets.

En résumé, le principe de l'approche parallèle est d'apprendre comment se comporte un objet (dans un repère lié à la souris) et de réutiliser cette expérience pour des situations mettant en jeu plusieurs objets. Cette méthode permet de réduire la complexité du système. Elle soulève trois problèmes principaux :

- suivre le déplacement de chaque objet par rapport à la souris pour pouvoir remettre à jour la fonction d'utilité $\mathsf{q}$ puisque ce calcul nécessite la connaissance de la position précédente et de la position actuelle d'un objet (*cf.* figure 4.2b),




(a) Changement de *situation* : suite à la dernière commande les objets se sont déplacés vers la droite par rapport à la souris.

(b) Chaque *perception* d'objet est traitée séparément. Pour suivre le déplacement de chaque objet, on réalise une *mise en correspondance temporelle des objets.*

(c) Chaque transition d'objet permet d'optimiser la même fonction d'utilité q en un point différent. L'espérance de gain vis-à-vis de chaque perception est ensuite calculée.

(d) Une étape de fusion calcule l'espérance de gain *globale de chaque commande* pour permettre le choix de la commande à effectuer.

FIG. 4.2 – *Principe de l'architecture parallèle.*

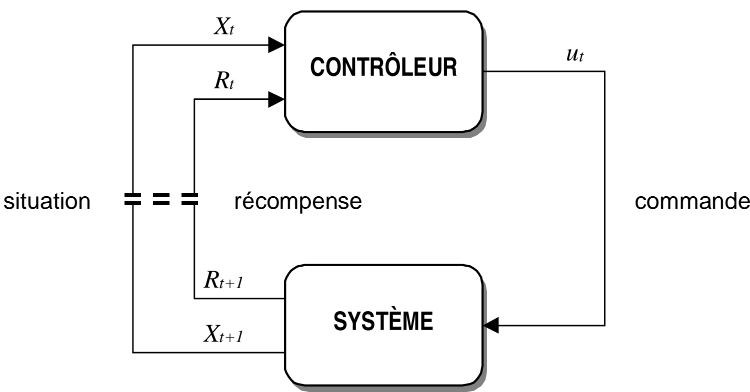



FIG. 4.3 – *Boucle sensori-motrice.*

- trouver un moyen d'attribuer automatiquement les récompenses qui sont déterminées au niveau global pour ne pas avoir à spécifier des comportements dédiés au sein du contrôleur (*cf.* figure 4.2c).
- reconstituer par une étape de fusion l'espérance de gain *globale* Q des commandes vis-à-vis de la situation *globale* du système (*cf.* figure 4.2d).

Nous allons maintenant décrire notre approche plus formellement et expliquer les solutions que nous proposons pour répondre à ces problèmes.

## 4.3 Définitions et hypothèses fondamentales

Notre approche se situe dans le cadre classique d'une boucle sensori-motrice (*cf.* §2.3.2, page 14). A chaque période d'échantillonnage, le contrôleur reçoit des informations $X_t$ sur l'état du système ainsi qu'un signal de récompense $R_t$ puis décide d'une commande $u_t$ qui agit sur le système (*cf.* figure 4.3).

En revanche, l'approche parallèle est dédiée au contrôle de systèmes particuliers dont les états sont appelés *situations*.

### 4.3.1 Situations et perceptions

Reprenons l'exemple du labyrinthe présenté par la figure 4.2. La situation globale du système comporte quatre objets. On la décompose en quatre perceptions élémentaires. Chaque perception caractérise le type (fromage ou mur) et la position *dans un repère lié à la souris* de chaque objet (la souris est donc immobile dans le repère considéré). Ces deux informations sont regroupées au sein d'une variable notée $x_t^i$ et appelée perception de l'objet $i$ à l'instant $t$. L'ensemble des valeurs (positions et types) que peut prendre une perception est noté $\mathcal{X}$ et appelé *espace des perceptions* (ou *ensemble des perceptions* si ce n'est pas un espace vectoriel). Dans notre exemple, la perception $x_t^2$ du fromage se note $x_t^2 = (0{,}2{,}\text{fromage})$.

La situation globale du système regroupe l'ensemble des perceptions élémentaires $x_t^i$ à l'instant $t$ dans un ensemble noté $X_t$. Dans notre exemple, la situation est l'ensemble

des quatre perceptions, soit :

$$X_t = \{(1,1,\text{mur}),(0,2,\text{fromage}),(0,1,\text{mur}),(2,-1,\text{mur})\} \tag{4.1}$$

Dans le cas général, on note :

$$X_t = \{x_t^i \,|\, 1 \leq i \leq n_t, x_t^i \in \mathcal{X}\} \tag{4.2}$$

avec $n_t$ le cardinal de la situation $X_t$. Les perceptions d'une situation ne sont pas ordonnées et le cardinal $n_t$ de la situation peut varier dans le temps. L'espace d'états du système est donc de la forme $\mathcal{X}^{n_t}$.

Beaucoup de systèmes peuvent être appréhendés sous cette forme. De manière générale, ce cadre s'adapte particulièrement bien aux systèmes dont les variables d'état appartiennent à un même espace vectoriel. Néanmoins, si les capteurs sont hétérogènes, il suffit de regrouper dans une seule variable toutes les informations décrivant la perception quel que soit le capteur. On peut aussi utiliser plusieurs $\mathcal{X}_i$ au lieu d'un unique $\mathcal{X}$. Néanmoins, pour des soucis de simplification des notations, nous utilisons un unique $\mathcal{X}$ dans cette thèse.

### 4.3.2 Commandes

A chaque période d'échantillonnage, le contrôleur envoie une commande $u_t$ aux actionneurs. L'ensemble des commandes est noté $\mathcal{U}$.

### 4.3.3 Récompenses

Dans le cadre classique de l'apprentissage par renforcement, le signal de récompense est un scalaire. Dans notre cas, ce choix peut provoquer des situations ambiguës. Par exemple, si la souris atteint un fromage, action récompensée par $+1$, mais qu'à cause de cette action, la souris passe aussi sur un piège, action punie par $-1$, le contrôleur va par exemple recevoir la somme des récompenses (et des punitions), c'est-à-dire 0. Il nous semble que cette situation n'est pas équivalente aux situations où le contrôleur reçoit 0 parce que la souris ne touche aucun des pièges et des fromages.

Pour cette raison, nous avons choisi une définition plus large du signal de récompense : à chaque période d'échantillonnage, le contrôleur reçoit un multi-ensemble[1] $R_t$ de scalaires supposés non ordonnés. Le cas non ordonné est un cas d'étude général qui suppose que le contrôleur ne connaît pas d'emblée à quelles perceptions sont associées les récompenses. On peut alors parler de récompenses non étiquetées *a priori*. Ce cas de figure correspond par exemple à un enfant qui serait puni sans la moindre explication par rapport à sa situation courante. C'est alors à l'enfant de comprendre à quel élément de la situation se rapporte la punition pour éviter à l'avenir qu'elle ne se reproduise. Le cas ordonné où les récompenses sont classées dans le même ordre que leur perception

1. Ensemble dans lequel un même élément peut apparaître plusieurs fois. Autrement dit, la multiplicité d'un élément d'un multi-ensemble peut être supérieure à 1.

correspondante dans la situation est donc un cas particulier du cas envisagé et serait beaucoup plus informatif pour le contrôleur.

Le signal de récompense est un multi-ensemble de la forme :

$$R_t = \{r_t^i \,|\, 1 \leq i \leq \min(n_{t-1},n_t), r_t^i \in \mathbb{R}\} \tag{4.3}$$

Par exemple, si la souris décrite ci-dessus passe sur un piège au moment où elle atteint un fromage, $R_t$ contiendra deux récompenses non nulles, l'une positive et l'autre négative, et $n_t - 2$ récompenses nulles ($R_t = \{-1,1,0,\ldots,0\}$) correspondant aux autres éléments perçus.

Cette définition traduit l'idée qu'il ne peut y avoir qu'une récompense scalaire par perception. Dans le cas où $n_{t-1} \neq n_t$, les perceptions qui apparaissent ou disparaissent sont ignorées car il n'y a pas de transition associée.

### 4.3.4 Hypothèses fondamentales

On suppose que chaque perception $x_t^i$ d'une situation $X_t$ est une variable indépendante qui satisfait les hypothèses de Markov. En d'autres termes, si la perception $x_t^i$ était l'unique élément de l'état du système $X_t$, ce système serait markovien.

Par exemple, 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 commande effectuée, quelles que soient les positions des autres fromages. Les perceptions élémentaires de ce système peuvent être considérées comme markoviennes.

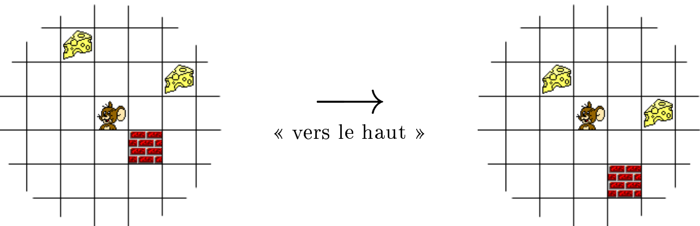
On suppose donc qu'il existe une fonction $\mathsf{p}$ décrivant les probabilités de transition entre perceptions, telle que $\mathsf{p}(x'|u,x)$ donne la probabilité d'obtenir la perception $x'$ après avoir effectué la commande $u$ à partir de la perception $x$. De même, il existe une fonction de récompense immédiate $\mathsf{r}_t(x,u,x') \in \mathbb{R}$ qui qualifie la dernière transition de perception définie par une perception $x$, la commande effectuée $u$ et la nouvelle perception $x'$.

Comme dans le cadre de l'apprentissage par renforcement, dans notre approche, ces deux fonctions sont inconnues. De plus, comme les perceptions d'une situation et les récompenses ne sont pas ordonnées, il faut reconstituer les déplacements des perceptions dans le repère lié à la souris et retrouver les récompenses correspondant à ces transitions. En d'autres termes, à l'instant $t$, le contrôleur reçoit deux situations $X_{t-1}$ et $X_t$ désordonnées ainsi qu'un signal de récompense $R_t$ désordonné. Il s'agit de retrouver les perceptions et récompenses qui qualifient la transition d'un même objet. Par exemple, il faut retrouver le déplacement relatif de chaque mur après la commande $u_{t-1}$ et la récompense associée à chaque déplacement élémentaire.

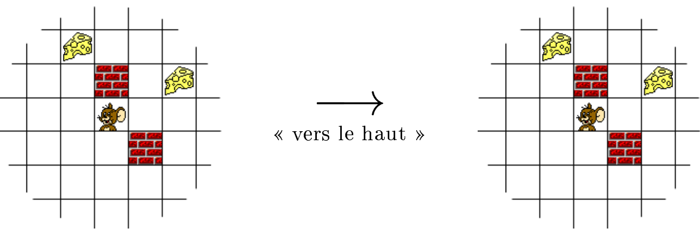
L'étape qui consiste à retrouver les deux perceptions d'une transition est appelée *mise en correspondance temporelle*. L'étape qui associe une récompense scalaire à cette transition est nommée *attribution des récompenses*.

#### Discussion

L'hypothèse que les perceptions sont des variables markoviennes est difficile à satisfaire complètement. En effet, des interactions entre les perceptions peuvent survenir, ce




(a) Dans cette situation, la commande « vers le haut » provoque un mouvement des perceptions vers le bas




(b) Dans cette situation, la commande « vers le haut » ne provoque pas de mouvement des perceptions bien que trois de ces perceptions soient dans la même position que dans le cas précédent.

FIG. 4.4 – *Les interactions entre la souris et les murs nuisent à l'indépendance des perceptions.*

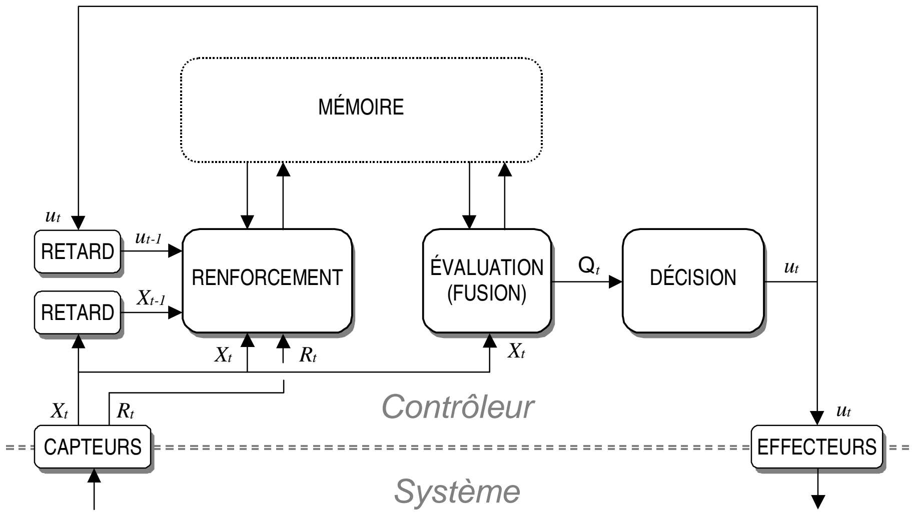



FIG. 4.5 – *Schéma fonctionnel de l'approche parallèle.*

**Entrée :** $X_{t-1}$, $u_{t-1}$, $X_t$, $R_t$
**Sortie :** $u_t$
**Début**
  $C_t$ ←MiseEnCorrespondance($X_{t-1}$,$X_t$)
  $A_t$ ←Attribution($C_t$,$R_t$,$u_{t-1}$,q)
  RenforcementParallèle($A_t$,$u_{t-1}$,q)
  Q$_t$ ←Fusion($X_t$,q)
  $u_t$ ←Décision(Q$_t$)
**Fin**

FIG. 4.6 – *Algorithme du Q-Learning parallèle.*

qui nuit à leur indépendance. Par exemple, la présence de murs peut modifier le déplacement relatif entre la souris et les fromages, car les fromages ne se déplacent plus si la souris est bloquée par un mur (*cf.* figure 4.4). Toute la question est de savoir si ces rares interactions nuiront à la stabilité de l'algorithme, ce que nous étudierons dans le chapitre suivant.

## 4.4 Fonctionnement d'un contrôleur parallèle

Le principe de fonctionnement du contrôleur parallèle est fondé sur le schéma classique de l'apprentissage par renforcement (*cf.* figure 4.5). Le contrôleur parallèle est constitué des quatre fonctions principales : la fonction de renforcement, la fonction de mémorisation, la fonction d'évaluation ici appelée fonction de *fusion* et la fonction de décision.

Dans un contrôleur parallèle, la mémoire associe une estimation de l'espérance de gain à chaque perception $x$ et non pas à une situation $X$. Elle mémorise donc la valeur d'utilité $\mathsf{q}(x,u)$ pour toutes les perceptions possibles de $\mathcal{X}$ et commandes de $\mathcal{U}$. La fonction de renforcement procède alors à un apprentissage de type Q-Learning pour chaque perception en utilisant *toujours* la même fonction d'utilité $\mathsf{q}$. Rappelons qu'une perception $x$ caractérise complètement un objet et notamment son type (mur, fromage, piège, *etc.*)

Cette méthode permet effectivement de réduire la complexité du système puisque l'apprentissage s'effectue sur l'ensemble des perceptions $\mathcal{X}$ et non sur l'ensemble des états du système $\mathcal{X}^{n_t}$ (*cf.* §4.3.1, page 62).

La fonction de fusion calcule une estimation de l'espérance de gain globale $\mathsf{Q}(X_t,u)$ pour la situation $X_t$ et les commandes $u \in \mathcal{U}$ à l'aide des valeurs d'utilité $\mathsf{q}(x_t^i,u)$. Enfin, la fonction de décision détermine la commande en fonction de critères d'exploitation et d'exploration exactement comme dans le Q-Learning classique.

Le fonctionnement général de ce contrôleur est synthétisé par l'algorithme de la figure 4.6 dont nous allons maintenant étudier les fonctions en détail.

### 4.4.1 Fonction de mémorisation

La mémoire stocke la valeur d'utilité de chaque perception. Elle utilise pour cela une fonction $\mathsf{q}$ qui à chaque couple *perception—commande* associe une valeur réelle, soit :

$$\mathsf{q} : \mathcal{X} \times \mathcal{U} \to \mathbb{R} \tag{4.4}$$

*A priori*, la fonction $\mathsf{q}$ peut être réalisée par n'importe quel type de mémorisation. Si les dimensions de $\mathcal{X}$ et de $\mathcal{U}$ ne sont pas trop importantes, il est possible d'utiliser un tableau statique ou dynamique. Sinon, on peut envisager d'utiliser une fonction d'approximation linéaire (CMAC, RBF, *etc.*).

Nous utilisons dans notre cas un tableau statique. Avant de commencer l'apprentissage, la fonction $\mathsf{q}$ est initialisée à 0 pour tout $(x,u)$ de $\mathcal{X} \times \mathcal{U}$.

### 4.4.2 Fonction de renforcement

Le rôle de la fonction de renforcement est d'optimiser l'estimation de l'espérance de gain $\mathsf{q}(x,u)$ d'une perception $x$. Pour ce faire, nous utilisons l'équation de mise à jour du Q-Learning. Cette équation met à jour la fonction $\mathsf{q}$ pour chaque perception $x_{t-1}$ de la situation $X_{t-1}$, soit :

$$\mathsf{q}(x_{t-1},u_{t-1}) \leftarrow \mathsf{q}(x_{t-1},u_{t-1}) + \alpha\big[r_t + \gamma \max_{v\in\mathcal{U}} \mathsf{q}(x_t,v) - \mathsf{q}(x_{t-1},u_{t-1})\big] \tag{4.5}$$

Le problème est que l'on ne connaît pas *a priori* le successeur $x_t$ de la perception $x_{t-1}$. En effet, les perceptions ne sont pas ordonnées à l'intérieur des situations. Par exemple, la première perception $x_t^1$ de $X_t$ n'est pas forcément le successeur de la première perception $x_{t-1}^1$ de $X_{t-1}$. Il est donc nécessaire de créer un outil pour apparier les perceptions de la situation $X_{t-1}$ avec celles de la situation $X_t$.

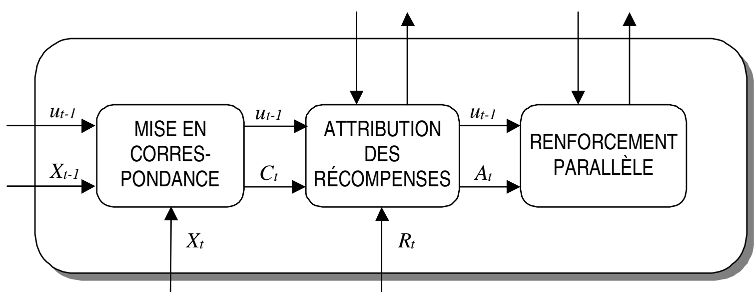



FIG. 4.7 – *Nouvelles fonctionnalités de la fonction de renforcement présentée par la figure 4.5.*

D'autre part, comme l'ensemble $R_t$ des récompenses n'est pas ordonné, le scalaire $r_t$ ne correspond pas forcément à la transition $(x_{t-1},x_t)$. Il faut retrouver la transition qui correspond à ce scalaire.

Pour pallier ces problèmes, le bloc de renforcement est enrichi par de nouvelles fonctions (*cf.* figure 4.7). Une fonction de *mise en correspondance temporelle* relie les perceptions de $X_{t-1}$ avec leur successeur de $X_t$. Une fonction d'*attribution des récompenses* détermine les perceptions responsables des récompenses obtenues. Enfin, une fonction de *renforcement parallèle* met à jour la mémoire.

### Fonction de mise en correspondance temporelle

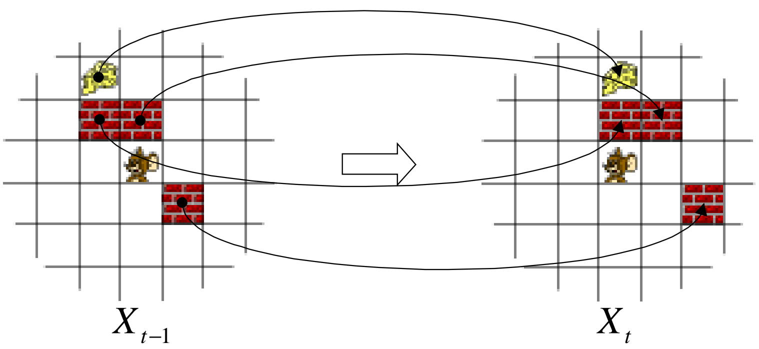
Le rôle de la mise en correspondance est d'apparier les perceptions de la situation $X_{t-1}$ avec celles de la situation $X_t$. Pratiquement, la mise en correspondance doit reconstituer l'évolution de chaque perception. Par exemple dans le labyrinthe, elle doit reconstituer le déplacement relatif de chaque objet par rapport à la souris entre deux instants $t$ (*cf.* figure 4.8). La mise en correspondance est dite *exacte* si elle correspond à la réalité de l'évolution des perceptions, ce qui est généralement invérifiable si on ne connaît pas le comportement du système.

On note $\mathsf{c}$ la fonction de mise en correspondance et on pose :

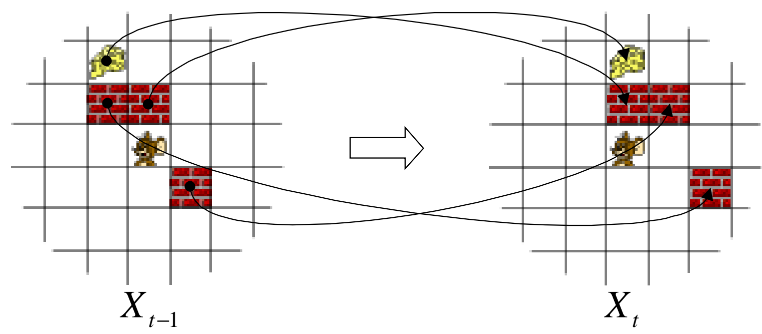
$$C_t = \mathsf{c}(X_{t-1},X_t) \tag{4.6}$$

La sortie de cette fonction est un ensemble $C_t$ de couples $c_t^i$. Chaque couple $c_t^i$ est constitué d'une perception $x_{t-1}^j$ de $X_{t-1}$ et de son successeur supposé $x_t^k$ appartenant à $X_t$, donc $c_t^i$ est de la forme $(x_{t-1}^j,x_t^k)$.

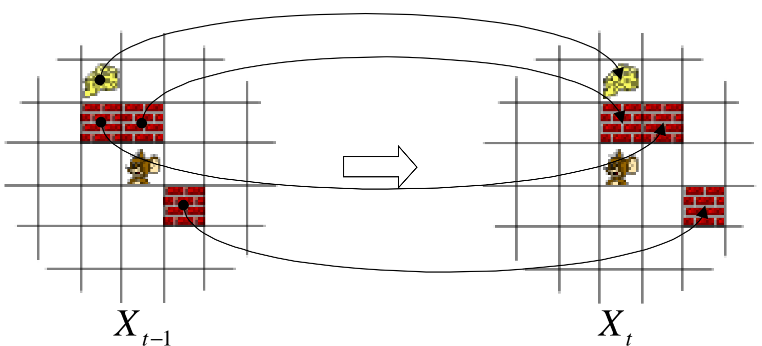
Afin d'obtenir une bonne solution en un faible temps de calcul, nous avons opté pour une méthode heuristique très simple. Il s'agit de prendre au hasard une perception de la situation $X_t$ et de l'associer à la perception la plus proche (en terme de distance) appartenant à $X_{t-1}$, puis de recommencer jusqu'à avoir apparié toutes les perceptions. Si un nouvel objet entre ou sort du champ de vision, $X_{t-1}$ et $X_t$ n'auront pas le même cardinal ($n_{t-1} \neq n_t$). Dans ce cas, les perceptions qui n'ont pas été associées sont ignorées par la mise en correspondance pendant une période d'échantillonnage, le temps que




(a) Mise en correspondance exacte




(b) Mise en correspondance erronée




(c) Mise en correspondance erronée

FIG. 4.8 – *Exemples de mises en correspondance.*

**Entrée :** $X_{t-1}$, $X_t$
**Sortie :** $C_t$
**Début**
  $C_t \leftarrow \emptyset$
  **Si** $|X_{t-1}| < |X_t|$ **alors**
    **Tant que** $X_{t-1} \neq \emptyset$ **faire**
      Retirer au hasard une perception $x$ de $X_{t-1}$
      $x' \leftarrow \arg\min_{y \in X_t} \|x - y\|$ *(les ex æquo sont départagés par tirage au sort)*
      $X_t \leftarrow X_t \setminus \{x'\}$
      $C_t \leftarrow C_t + \{(x,x')\}$ *(concaténation)*
    **Fin tant que**
  **Sinon**
    **Tant que** $X_t \neq \emptyset$ **faire**
      Retirer au hasard une perception $x'$ de $X_t$
      $x \leftarrow \arg\min_{y \in X_{t-1}} \|x' - y\|$ *(les ex æquo sont départagés par tirage au sort)*
      $X_{t-1} \leftarrow X_{t-1} \setminus \{x\}$
      $C_t \leftarrow C_t + \{(x,x')\}$ *(concaténation)*
    **Fin tant que**
  **Fin si**
**Fin**

FIG. 4.9 – *Algorithme de la fonction de mise en correspondance temporelle.*

$X_{t-1}$ et $X_t$ retrouvent le même cardinal. L'algorithme de cette heuristique est décrit en détail par la figure 4.9. Il distingue deux cas de figure : si $|X_{t-1}| < |X_t|$ il prend les perceptions de $X_{t-1}$ et les associe à celles de $X_t$, sinon il prend les perceptions de $X_t$ et les associe à celles de $X_{t-1}$. Cette méthode permet implicitement d'ignorer les perceptions qui viennent d'apparaître ou de disparaître.

Cette méthode implique la définition d'une métrique sur $\mathcal{X}$. Cette définition doit être cohérente avec le fonctionnement du système. Par exemple, la distance entre une perception fromage et une perception mur doit être beaucoup plus grande que la distance entre deux perceptions de même type dans la même configuration spatiale. En effet, dans le cas contraire, le changement de type serait aussi facile que le déplacement d'un objet, ce qui n'est pas réaliste. Par exemple, pour le labyrinthe, si $i_1$, $j_1$ et $k_1$ sont les coordonnées et le type de la perception $x$ (avec par exemple, $k_1 = 1$ pour un mur et $k_1 = 2$ pour un fromage), et $i_2$, $j_2$ et $k_2$ ceux de la perception $y$, on utilisera une distance[2] de la forme :

$$\|x - y\| = |i_1 - i_2| + |j_1 - j_2| + K|k_1 - k_2| \tag{4.7}$$

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

Cette méthode heuristique n'est évidemment pas exacte et commet parfois des erreurs d'appariement, notamment quand les perceptions sont regroupées. Nous verrons par la suite si l'algorithme global est affecté ou non par ces erreurs de mise en correspondance. On pourra améliorer cette fonction en trouvant un critère plus pertinent ou même imaginer une mise en correspondance par apprentissage.

### Fonction d'attribution des récompenses

Le rôle de cette fonction est de déterminer les perceptions en rapport avec l'obtention des récompenses $R_t$. Par exemple, la souris trouve un fromage et reçoit alors l'ensemble de récompenses $\{1,0,0,0\}$. La perception du fromage qui a provoqué cette récompense positive est celle qui est actuellement à la même position que la souris. Mais cette information est issue de notre connaissance globale du système. A son niveau, la souris doit apprendre la raison de cette récompense (*cf.* figure 4.10). Autre exemple, si la souris est sur une case encadrée de deux murs et qu'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), la proximité n'est donc pas discriminante pour l'attribution de la récompense négative obtenue.

Le contrôleur doit donc apprendre l'association des récompenses et des perceptions à l'aide des seules informations dont il dispose. C'est le rôle de la fonction d'attribution des récompenses.

Soit $\mathsf{a}$ la fonction d'attribution, on pose :

$$A_t = \mathsf{a}(C_t,R_t) \tag{4.8}$$

2. Cette définition de la distance suppose que le type soit numérique ou étiquetable. Si ce n'est pas le cas, il faut envisager une autre définition de la distance.

FIG. 4.10 – *Illustration du problème de l'attribution des récompenses : à quelle perception la récompense obtenue correspond-elle ?*

Le résultat de cette fonction est un ensemble $A_t$ de triplets $a_t^i$. Ces triplets $a_t^i$ associent les deux membres d'un couple de perceptions à la récompense attribuée à ce couple. Ils sont donc de la forme $(x_{t-1}^j, x_t^k, r_t^l)$.

L'idée que nous avons développée utilise la seule information disponible : la fonction d'utilité $\mathsf{q}$. La valeur d'utilité $\mathsf{q}(x_t,u_t)$, qui représente l'espérance de gain du couple *perception—commande* $(x_t,u_t)$, est constituée pour une grande part de l'espérance de la récompense *immédiate*. Cette affirmation est d'autant plus vraie quand les récompenses non nulles sont rares ou quand $\gamma$ n'est pas trop proche de 1. En quelque sorte, on peut dire que l'espérance de gain $\mathsf{q}(x_t,u_t)$ est une estimation très grossière de la récompense *immédiate* $r_{t+1}$ qui va être obtenue si on effectue la commande $u_t$ lorsqu'on observe la perception $x_t$. Ainsi, si la fonction d'utilité est proche de l'optimale, les valeurs qu'elle donne des perceptions de $X_t$ associées à la commande $u_t$ correspondent à peu près aux récompenses que l'on obtiendra à l'instant $t+1$.

A partir de cette constatation, nous avons développé un algorithme d'attribution des récompenses fondé sur la fonction d'utilité. Il consiste à trouver une mise en correspondance entre récompenses et perceptions qui minimise la somme des écarts entre les valeurs des récompenses et les valeurs d'utilité des perceptions. On cherche donc une fonction $\mathsf{a}$ d'attribution des récompenses telle que, pour toute autre fonction d'attribution $\mathsf{h}$, on ait :

$$\sum_{\forall(x,x',r)\in\mathsf{a}(C_t,R_t)} |\mathsf{q}(x,u_{t-1}) - r| \leq \sum_{\forall(x,x',r)\in\mathsf{h}(C_t,R_t)} |\mathsf{q}(x,u_{t-1}) - r| \tag{4.9}$$

**Entrée :** $C_t$, $R_t$, $u_{t-1}$
**Entrée-sortie :** q
**Sortie :** $A_t$
**Début**
  $A_t \leftarrow \emptyset$
  **Tant que** $R_t \neq \emptyset$ **faire**
    $r \leftarrow \min R_t$
    $R_t \leftarrow R_t \setminus \{r\}$
    $(x,x') \leftarrow \arg\min_{(y,y') \in C_t} \mathsf{q}(y,u_{t-1})$ *(les ex æquo sont départagés par tirage au sort)*
    $C_t \leftarrow C_t \setminus \{(x,x')\}$
    $A_t \leftarrow A_t + \{(x,x',r)\}$ *(concaténation)*
  **Fin tant que**
**Fin**

FIG. 4.11 – *Algorithme de la fonction d'attribution des récompenses.*

Par exemple, si le contrôleur, dans la situation de la figure 4.2, décide de faire monter la souris, elle ne va pas bouger et le système va rester dans la même situation. L'ensemble des récompenses obtenues est alors $\{-1,0,0,0\}$. Le contrôleur calcule les espérances de gain des perceptions et cherche l'attribution des récompenses qui minimise la somme des écarts. Ainsi, la perception dont la valeur d'utilité est la plus faible reçoit la récompense $-1$, les autres perceptions recevant 0. De même, si la souris atteint le fromage et obtient $R = \{0,0,1,0\}$, la récompense 1 est attribuée à la perception dont la valeur d'utilité est la plus grande, les autres perceptions recevant 0.

Par définition, les récompenses obtenues $R_t$ sont aussi nombreuses que les couples de perceptions $C_t$. Pour trouver une fonction d'attribution optimale, il suffit d'associer la plus petite des récompenses avec le couple de perceptions dont la valeur d'utilité est la plus faible, d'enlever chacun de ces deux éléments des ensembles $R_t$ et $C_t$ correspondants, puis de recommencer ce 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 4.11.

Il n'est pas établi que cette méthode adjointe à l'apprentissage de la fonction d'utilité converge vers une attribution sans erreur et permette aussi à la fonction d'utilité de converger. Ce principe de mise en correspondance des récompenses sera testé dans le chapitre suivant.

### Fonction de renforcement parallèle

Une fois la mise en correspondance et l'attribution des récompenses effectuées, la fonction de renforcement parallèle met à jour la mémoire avec les informations contenues

**Entrée :** $A_t$, $u_{t-1}$
**Entrée-sortie :** $\mathsf{q}$
**Début**
**Pour tout** $(x,x',r)$ de $A_t$ **faire**
$\mathsf{q}(x,u_{t-1}) \leftarrow \mathsf{q}(x,u_{t-1}) + \alpha[r + \gamma \max_{v\in\mathcal{U}} \mathsf{q}(x',v) - \mathsf{q}(x,u_{t-1})]$
**Fin pour**
**Fin**

FIG. 4.12 – *Algorithme de la fonction de renforcement parallèle.*

dans l'ensemble de triplets $A_t$. Pour cela, elle utilise l'équation de mise à jour du Q-Learning avec chaque triplet $(x,x',r)$ de $A_t$, soit :

$$\forall(x,x',r) \in A_t,\ \mathsf{q}(x,u_{t-1}) \leftarrow \mathsf{q}(x,u_{t-1}) + \alpha[r + \gamma \max_{v\in\mathcal{U}} \mathsf{q}(x',v) - \mathsf{q}(x,u_{t-1})] \quad (4.10)$$

La figure 4.12 présente l'algorithme de cette fonction.

#### Convergence de la fonction d'utilité $\mathsf{q}$

L'exploration de l'espace des perceptions est garantie par la stratégie de décision de l'$\epsilon$-gourmand (*cf.* §2.4.1, page 20). Si les perceptions sont réellement des variables markoviennes et si les étapes de mise en correspondance et d'attribution ne font pas d'erreur, alors la fonction d'utilité $\mathsf{q}$ convergera vers la fonction d'utilité optimale. Malheureusement, les fonctions de mise en correspondance et d'attribution ne sont pas parfaites et donc les trois hypothèses ne sont pas toujours vérifiées. Cela ne signifie pas que la fonction ne convergera pas, mais qu'il n'existe aucune assurance de cette convergence.

### 4.4.3 Fonction de fusion

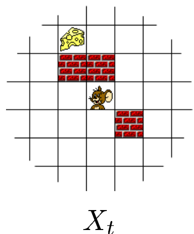
#### Définition du gain dans le cadre parallèle

La nouvelle définition du signal de récompense impose de redéfinir le critère global à maximiser, à savoir le gain (*cf.* §2.3.4, page 15). La définition suivante du gain semble cohérente avec la définition classique :

$$\mathsf{G}(t) = \sum_{k=0}^{\infty} \gamma^k \sum_{\forall r \in R_{t+k}} r \quad (4.11)$$

La fonction d'utilité globale notée $\mathsf{Q}^\pi(X,u)$ qui correspond à l'espérance de gain d'effectuer la commande $u$ dans la situation $X$ en suivant la stratégie $\pi$ est alors toujours définie par :

$$\mathsf{Q}^\pi(X,u) = E_\pi\{\mathsf{G}(t) \,|\, X_t = X, u_t = u\} \quad (4.12)$$



$X_t$

| | $u^1$ | $u^2$ | $u^3$ | $u^4$ |
|---|---|---|---|---|
| $\mathsf{Q}^*(X_t,u)$ | 0,6561 | 0,4783 | 0,6561 | −0,4095 |
| $\mathsf{q}^*(x_t^1,u)$ | 0 | 0 | 0 | 0 |
| $\mathsf{q}^*(x_t^2,u)$ | 0,6561 | 0,6561 | 0,81 | 0,81 |
| $\mathsf{q}^*(x_t^3,u)$ | 0 | 0 | 0 | −1 |
| $\mathsf{q}^*(x_t^4,u)$ | 0 | 0 | 0 | 0 |
| $\sum$ | 0,6561 | 0,6561 | 0,81 | -0,19 |

FIG. 4.13 – *Comment estimer $\mathsf{Q}_t$ en fonction des valeurs d'utilité de chaque perception de $X_t$ ? (avec $u_1$ = droite, $u_2$ = bas, $u_3$ = gauche, $u_4$ = haut et $\gamma = 0{,}9$).*

**Entrée :** $X_t$, q
**Sortie :** $\mathsf{Q}_t$
**Début**
  **Pour tout** $v$ de $\mathcal{U}$ **faire**
    $\mathsf{Q}_t(v) \leftarrow \sum_{\forall x \in X_t} \mathsf{q}(x,v)$
  **Fin pour**
**Fin**

FIG. 4.14 – *Algorithme de la fonction de fusion (opérateur somme).*

Il reste à définir un opérateur de fusion capable d'estimer $Q_t(X_t,u)$, ce qui, comme nous allons le voir, est très délicat.

### Fonction de fusion

Le rôle de la fonction de fusion est de calculer une estimation de l'espérance de gain *globale* $\mathsf{Q}_t(X_t,u)$ de chaque commande $u$ pour la situation globale $X_t$ du système. Comme la mémoire stocke l'espérance de gain des commandes par rapport à une perception, le contrôleur dispose des estimations de l'espérance de gain $\mathsf{q}(x_t^i,u)$ de chaque perception $x_t^i$. A partir de ces valeurs, il faut trouver un moyen de calculer une estimation de l'espérance de gain globale $\mathsf{Q}_t(X_t,u)$.

Par exemple, dans la situation de la figure 4.13, la fonction d'utilité $\mathsf{q}^*$ donne les estimations reportées dans le tableau. Dans ce cas, comment estimer l'espérance de gain globale de chaque commande ? Il faut trouver un opérateur de fusion pour calculer au mieux $\mathsf{Q}(X_t,u)$ à partir des valeurs $\mathsf{q}(x_t^i,u)$. Cet opérateur sera noté f et on pose :

$$\mathsf{Q}_t(X_t,u) = \mathsf{f}(\{\mathsf{q}(x,u) \mid \forall x \in X_t\}) \tag{4.13}$$

La discussion est ouverte et de nombreux opérateurs sont envisageables. Dans un premier temps, nous avons choisi d'étudier l'opérateur *somme*. Par la suite, au vu des résultats obtenus, nous reviendrons sur la définition même de la fonction de fusion.

**Entrée :** $\mathsf{Q}_t$
**Sortie :** $u_t$
**Début**
$\xi \leftarrow$ nombre réel aléatoire de $[0;1[$
**Si** $\xi < \epsilon$ **alors**
$u_t \leftarrow$ commande aléatoire de $\mathcal{U}$
**Sinon**
$u_t \leftarrow \arg\max_v \mathsf{Q}_t(v)$ *(les ex æquo sont départagés par tirage au sort)*
**Fin si**
**Fin**

FIG. 4.15 – *Algorithme de la fonction de décision ($\epsilon$-gourmand).*

L'opérateur *somme* consiste à calculer la somme des espérances de gain de chaque perception, soit :

$$\mathsf{Q}_t(X_t,u) = \sum_{\forall x \in X_t} \mathsf{q}(x,u) \tag{4.14}$$

L'algorithme synthétique de la fonction de fusion est donné par la figure 4.14.

Cet opérateur s'inspire du mode de fonctionnement d'un arbitre DAMN (*cf.* §3.3.3, page 51) où chaque comportement note chacune des commandes. Il correspond au principe de maximisation de l'espérance collective du W-Learning (*cf.* §3.4.4, page 55). Enfin, nous avons choisi l'opérateur somme car il nous semble être l'opérateur le plus cohérent avec la définition du gain (*cf.* équation (4.11)). Cependant, il ne permet pas d'obtenir l'espérance de gain globale exacte (il suffit pour s'en rendre compte de faire le calcul avec l'exemple donné par le tableau 4.13).

### 4.4.4 Fonction de décision

La fonction de décision se base sur les valeurs d'utilité globales $\mathsf{Q}_t(X_t,u)$ de chaque commande $u$ de $\mathcal{U}$ afin de déterminer la commande à effectuer. Le choix est influencé par des critères d'exploration et d'exploitation (*cf.* §2.4.1, page 20).

Les stratégies de décision classiques s'adaptent parfaitement à l'approche parallèle. Nous utilisons l'$\epsilon$-gourmand dont l'algorithme est décrit en figure 4.15 car il permet de maîtriser précisément le taux d'exploration, ce qui est fort utile lors de l'étude des comportements de l'algorithme.

## 4.5 Conclusion

Ce chapitre nous a permis de définir notre cadre de travail et d'établir les principes de base sous-jacents à notre idée initiale de parallélisation.

Ainsi, les états des systèmes commandés sont particuliers et structurés en ensembles de perceptions appelés situations. D'autres part, nous avons élargi la définition du signal de récompense pour supprimer les ambiguïtés que peut cacher un signal scalaire.

Nous avons ensuite présenté l'architecture parallèle. Cette architecture utilise une fonction d'utilité qui stocke des estimations de l'espérance de gain vis-à-vis des perceptions élémentaires et *non pas par rapport à une situation globale*, ce qui permet de réaliser l'apprentissage sur l'ensemble $\mathcal{X}$ des perceptions et non sur l'espace d'états global du système $\mathcal{X}^n$ ($n$ désigne le nombre de perceptions élémentaires d'une situation). Une fonction de fusion est chargée de reconstituer à l'aide de la fonction d'utilité élémentaire l'espérance de gain globale de chaque commande. Cette architecture nécessite l'ajout de nouvelles fonctionnalités pour permettre la mise en correspondance temporelle des perceptions et l'attribution des récompenses.

La mise en correspondance utilise une heuristique qui n'est pas exacte et peut donc commettre des erreurs d'appariement. L'attribution des récompenses est réalisée à l'aide d'estimations des récompenses fondées sur la fonction d'utilité, principe dont la convergence n'est pas établie. En conséquence, rien ne prouve que la fonction d'utilité converge bien vers l'optimale et que le contrôleur produise une bonne stratégie de commande. Dans le chapitre suivant, nous expérimenterons donc cette architecture pour valider son fonctionnement et déterminer ses avantages et ses inconvénients.

# Chapitre 5

# Application du Q-Learning parallèle à l'exemple du labyrinthe

*Ce chapitre présente les résultats expérimentaux obtenus sur l'exemple du labyrinthe. Les tests montrent que le Q-Learning parallèle est stable et permet d'apprendre plus vite quand le nombre de perceptions est plus important. En revanche, dans certains cas particuliers, la fonction de fusion peut générer momentanément des maxima locaux, cependant, grâce à l'apprentissage, le contrôleur est plus robuste sur ce point que les architectures comportementales.*

## 5.1 Introduction

L'objectif de ce chapitre est de valider le principe parallèle sur l'exemple du labyrinthe. La simplicité du labyrinthe, sans être trop réductrice, permet de bien comprendre le fonctionnement de l'algorithme.

Ce chapitre définit tout d'abord les outils de mesure qui permettront d'observer et de comparer la performance de l'algorithme en fonction des paramètres et des situations. Ensuite, nous étudierons successivement les résultats obtenus avec un labyrinthe sans mur et les résultats obtenus avec un labyrinthe encombré de murs.

## 5.2 Définition du cadre expérimental

### 5.2.1 Labyrinthe de test

Nous avons choisi un labyrinthe de 400 cases ($20 \times 20$) sur lequel sont disposés au hasard des fromages et des murs. La souris peut se déplacer de case en case à l'aide de quatre commandes discrètes ($\mathcal{U}$ ={haut, bas, droite, gauche}). Le système est déterministe. A chaque instant $t$, la souris reçoit la situation $X_t$ du système qui contient les perceptions $x_t^i$ des objets. Chaque perception caractérise le type $k$ et les coordonnées $(i,j)$ d'un objet dans un *repère lié à la souris* ($\mathcal{X} = \{(i,j,k) \,|\, -19 \leq i \leq 19,\, -19 \leq j \leq 19, 1 \leq k \leq 2,\}$).

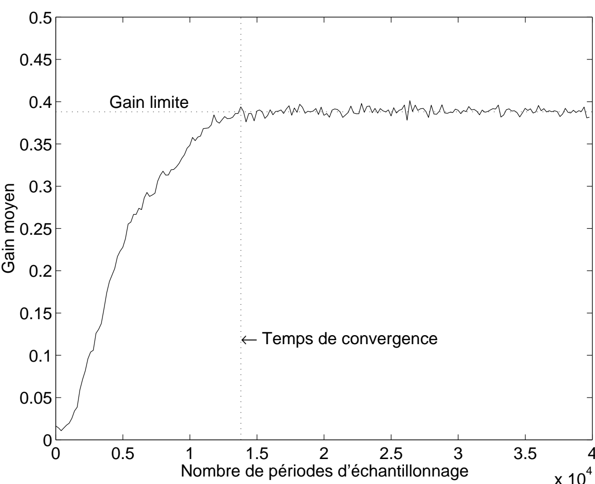



FIG. 5.1 – *Évolution du gain moyen en fonction du nombre de périodes d'échantillonnage (moyenne sur 100 simulations avec 5 fromages, $\alpha = 1$, $\gamma = 0{,}5$ et $\epsilon = 0{,}1$).*

La souris a donc toujours la position (0,0) dans le repère utilisé. Afin de limiter les effets de bord, les objets sont disposés à plus de 5 cases du bord du labyrinthe.

Si à l'instant $t$, la souris atteint une case pourvue d'un fromage, une récompense égale à 1 est ajoutée à l'ensemble $R_t$ de récompenses. A l'instant $t+1$, le fromage a disparu et est replacé à l'instant $t+2$ au hasard sur une case libre du labyrinthe.

Si à l'instant $t$, la souris tente de passer sur un mur, elle reste sur sa case et une récompense égale à $-1$ est ajoutée à l'ensemble $R_t$ de récompenses. Les murs sont fixes.

Au début de chaque test, la fonction $\mathsf{q}$ est initialisée à $\mathsf{q}_0 = 0$ pour obtenir une exploration parfaitement aléatoire au début de l'apprentissage (*cf.* §2.4.2, page 22). L'annexe A décrit le fonctionnement du logiciel de simulation développé.

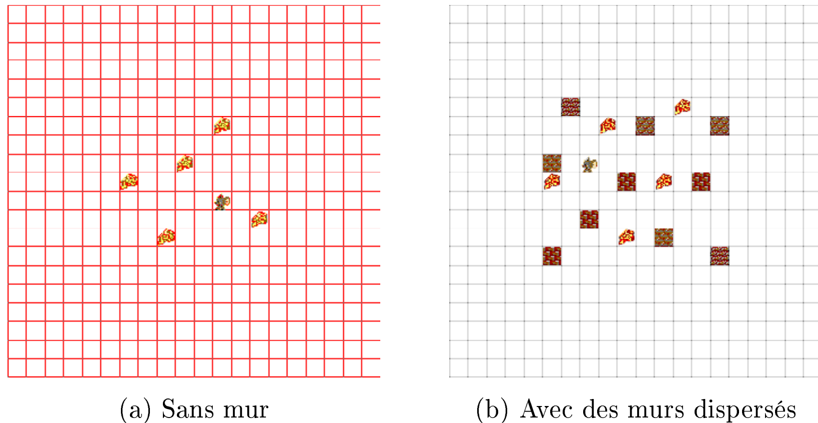
### 5.2.2 Définition des outils de mesure

La courbe de la figure 5.1 est la moyenne sur un grand nombre de simulations des évolutions du gain moyen en fonction du nombre de périodes d'échantillonnage. Le gain moyen $\mathsf{G}_m(t)$ est défini par :

$$\mathsf{G}_m(t) = \frac{1}{T}\sum_{k=0}^{T-1} \mathsf{G}(t+k) \tag{5.1}$$

avec $T$, un petit nombre de périodes d'échantillonnage (typiquement $T = 200$) et $\mathsf{G}$ le gain défini au chapitre 4 par l'équation (4.11). L'utilisation du gain moyen permet de lisser le tracé de la courbe.

Pour étudier et comparer les résultats de l'algorithme, nous avons défini deux mesures de performance : le gain limite et le temps de convergence (*cf.* figure 5.1). Le gain limite correspond à la valeur moyenne du gain moyen après convergence. Le temps de



(a) Sans mur (b) Avec des murs dispersés

FIG. 5.2 – *Labyrinthes de test.*

convergence correspond au nombre de périodes d'échantillonnage effectuées avant d'atteindre (ou de dépasser) pour la première fois le gain limite. Comme l'évolution du gain moyen est irrégulière, la mesure du temps de convergence est peu précise (par exemple, un pic du gain moyen dépassant le gain limite avant la convergence peut engendrer une mesure du temps de convergence en deçà de la réalité).

## 5.3 Labyrinthe avec un seul type d'objet

Cette partie présente les résultats de simulations dans le cas où le labyrinthe comporte plusieurs fromages mais pas de mur (*cf.* figure 5.2a).

### 5.3.1 Stabilité

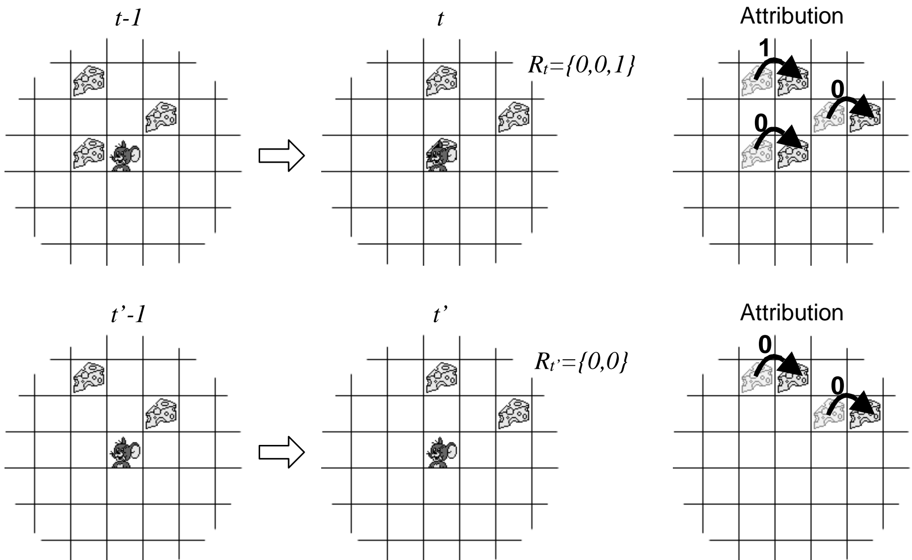
Malgré les erreurs de mise en correspondance et d'attribution qui peuvent survenir, le contrôleur est stable et le Q-Learning parallèle fonctionne bien. Sans vouloir prouver la stabilité de notre algorithme, nous en donnons deux explications vis-à-vis de la mise en correspondance et de l'attribution des récompenses.

#### Mise en correspondance

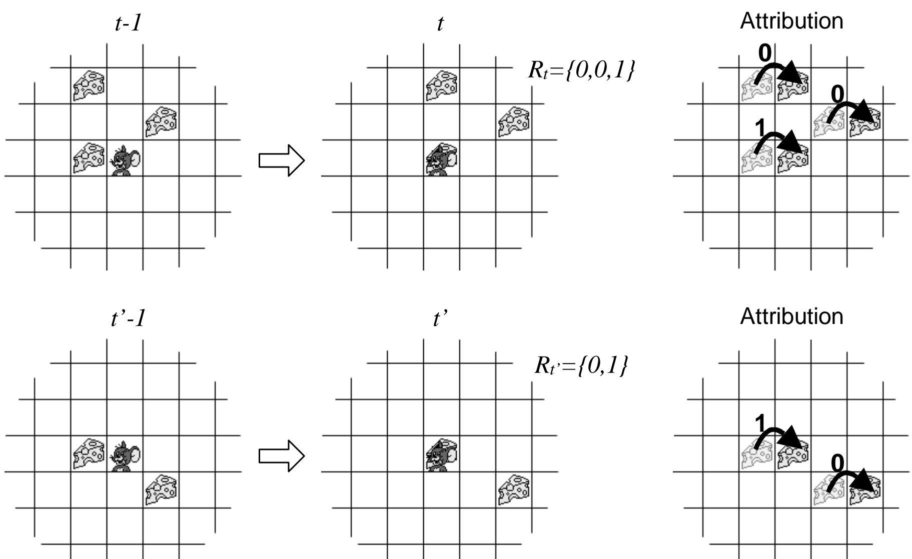
Les erreurs de mise en correspondance ne posent pas de problème car elles n'apparaissent pas toujours aux mêmes endroits. Tout se passe alors pour le contrôleur, comme si les transitions entre perceptions étaient légèrement non déterministes, ce qui ne nuit pas à la stabilité du Q-Learning utilisé par la fonction de renforcement.

#### Attribution des récompenses

Le principe de l'attribution des récompenses fonctionne bien. Le succès de cette méthode s'explique par un processus d'auto-vérification lié au principe même de l'algorithme.




(a) Premier scénario : à l'instant $t$, la souris trouve un fromage et attribue la récompense positive à la perception située aux coordonnées $(-1{,}2)$ (dans le repère lié à la souris). A l'instant $t' > t$, la souris se retrouve dans la même situation vis-à-vis d'une perception 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 $t$, la souris trouve un fromage et attribue la récompense positive à la perception située aux coordonnées $(-1{,}0)$. A l'instant $t' > t$, la souris se retrouve dans la même situation vis-à-vis d'une perception qui doit donner une récompense positive. Le contrôleur obtient alors une récompense positive. L'attribution précédente était donc correcte et le contrôleur attribue de nouveau la récompense positive à la perception située aux coordonnées $(-1{,}0)$.

FIG. 5.3 – *Illustration du processus d'auto-vérification de l'attribution des récompenses.*

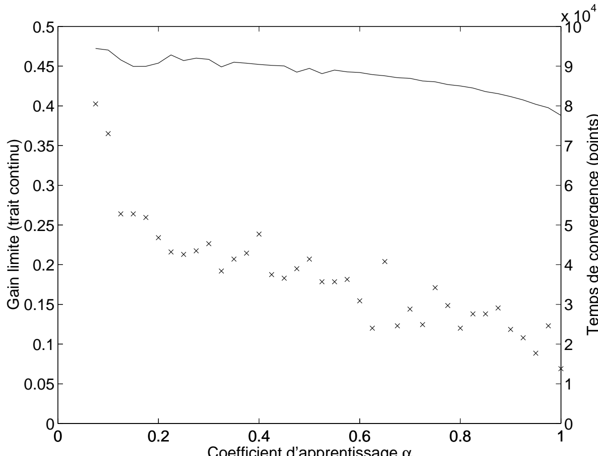



FIG. 5.4 – *Influence du coefficient d'apprentissage $\alpha$ sur le gain limite et sur le temps de convergence (moyenne sur 100 simulations avec 5 fromages, $\gamma = 0{,}5$ et $\epsilon = 0{,}1$).*

Au début de l'apprentissage, la fonction d'utilité contient des valeurs identiques pour tous les couples *perception—commande* $(x,u)$ ($\mathsf{q}_0 = 0$). Aussi, lors de l'attribution, les récompenses sont distribuées au hasard aux couples de perceptions. 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 perceptions. Cette attribution a donc peu de chances d'être correcte (*cf.* figure 5.3a). Comme la récompense est positive, le contrôleur va essayer d'obtenir à nouveau la perception qui, selon lui, a provoqué une récompense positive (comportement d'attraction globale, *cf.* §2.4.2, page 22). Après un certain nombre de périodes d'échantillonnage, le système va donc se retrouver dans une situation présentant la perception qui, d'après le contrôleur, fournira une récompense positive en effectuant la même commande que précédemment. Si l'attribution était erronée, alors les récompenses obtenues seront nulles, et la perception en question se verra attribuer une récompense nulle. En revanche, si l'attribution était correcte (*cf.* figure 5.3b), le contrôleur reçoit bien la récompense positive attendue et le processus d'attribution donnera à nouveau la récompense positive à la même perception. Ce processus d'*auto-vérification* assure donc une certaine stabilité de la fonction d'attribution.

### 5.3.2 Influence des paramètres

La qualité de l'apprentissage dépend souvent des paramètres choisis. Ainsi, dans cette section, nous étudions leur influence. Nous ne traiterons pas de l'influence du coefficient d'exploration $\epsilon$ car notre architecture est similaire sur ce point aux autres algorithmes. Le dilemme exploration—exploitation est une autre problématique qui n'est pas abordée dans cette thèse (*cf.* §2.4.1, page 20).

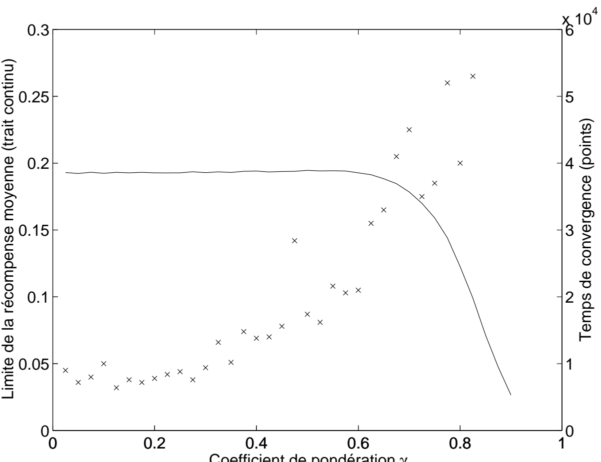



FIG. 5.5 – *Influence du coefficient de pondération $\gamma$ sur la limite de la récompense moyenne et sur le temps de convergence (moyenne sur 100 simulations avec 5 fromages, $\alpha = 1$ et $\epsilon = 0{,}1$).*

### Influence du coefficient d'apprentissage $\alpha$

La figure 5.4 montre l'influence de $\alpha$ sur le temps d'apprentissage et le gain limite. Comme le système est déterministe, le coefficient d'apprentissage $\alpha$ n'a pas beaucoup d'influence sur le comportement de l'algorithme. A moins de choisir $\alpha$ proche de 0, le temps d'apprentissage et le gain limite sont quasiment constants. Néanmoins, si $\alpha$ est grand, alors le temps d'apprentissage est légèrement plus court et la stratégie obtenue moins bonne. Cette observation rejoint le dilemme exploration—exploitation. En effet, soit le contrôleur prend plus de temps pour converger et peut donc visiter plus d'états permettant ainsi d'obtenir une meilleure stratégie ($\alpha$ petit), soit le contrôleur converge rapidement vers une stratégie de moins bonne qualité car il aura visité moins d'états ($\alpha$ grand).

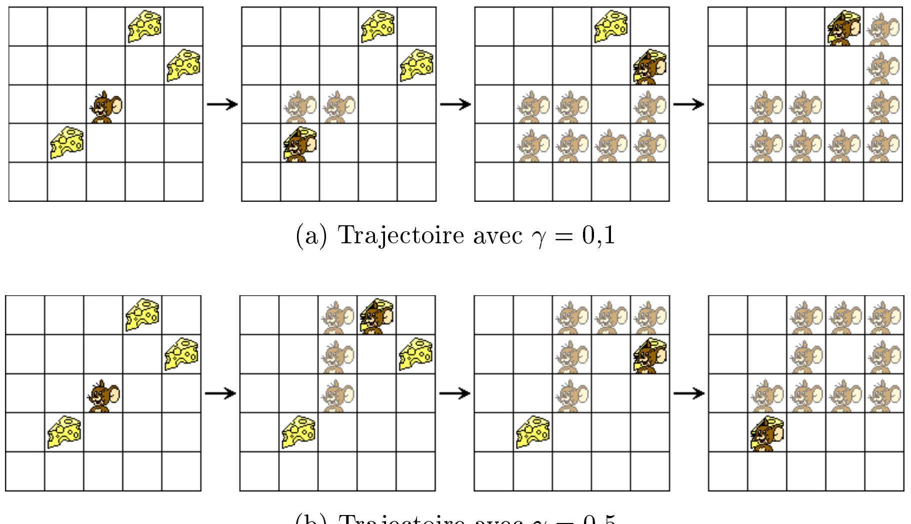
### Influence du coefficient de pondération $\gamma$

Comme le gain dépend de $\gamma$, nous préférons étudier la récompense moyenne obtenue au cours de la simulation pour étudier l'influence de $\gamma$. Nous définissons la récompense moyenne par :

$$R_m(t) = \frac{1}{T} \sum_{k=0}^{T-1} \sum_{\forall r \in R_{t+k}} r \qquad (5.2)$$

Dans le cas où il n'y a que des fromages, ce critère correspond au nombre moyen de fromages que la souris atteint par période d'échantillonnage.

La figure 5.5 montre l'évolution du temps d'apprentissage et de la limite de la récompense moyenne en fonction de $\gamma$. Pour des valeurs inférieures à 0,7, le coefficient $\gamma$ n'a



(a) Trajectoire avec $\gamma = 0{,}1$

(b) Trajectoire avec $\gamma = 0{,}5$

FIG. 5.6 – *Influence du coefficient de pondération $\gamma$ sur le comportement du contrôleur.*

pas d'influence sur les performances de l'algorithme. En revanche, les valeurs proches de 1 font augmenter le temps d'apprentissage et chuter la limite de la récompense moyenne.

Nous pensons que cette dégradation des performances est due à la faible dimension du système étudié. En effet, le nombre de cases à parcourir entre chaque fromage est relativement faible (typiquement 5 à 6 cases avec 5 fromages dans le labyrinthe) et quand $\gamma$ est proche de 1, la fonction d'utilité ne décroît plus assez vite et ses valeurs ne sont plus assez discriminantes. Ainsi, au niveau de la fusion, le choix de la commande est alors difficile puisque toutes les directions ont une bonne espérance de gain. De même, au niveau de l'attribution, qui est aussi basée sur les valeurs de la fonction d'utilité, il est par exemple difficile de retrouver la perception correspondant à une récompense positive puisque toutes les perceptions ont une espérance de gain proche.

Mis à part cet effet inhérent au système, $\gamma$ permet de déterminer le comportement du contrôleur comme dans le cas des algorithmes classiques (*cf.* §2.4.2, page 22). La figure 5.6 montre deux types de comportements. Si $\gamma$ est faible, le contrôleur privilégie les récompenses proches. Son comportement est opportuniste. Au contraire, si $\gamma$ est plus grand, le contrôleur préférera atteindre des zones plus denses en récompenses.

### 5.3.3 Influence du nombre de fromages

La figure 5.7 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 qui décroît de façon importante lorsque le nombre de fromages augmente. En fait, la présence de plusieurs fromages permet au contrôleur d'expérimenter un plus grand nombre de perceptions à chaque période d'échantillonnage, ainsi il met à jour la fonction d'utilité plusieurs fois et pour des perceptions différentes. Ce résultat est très

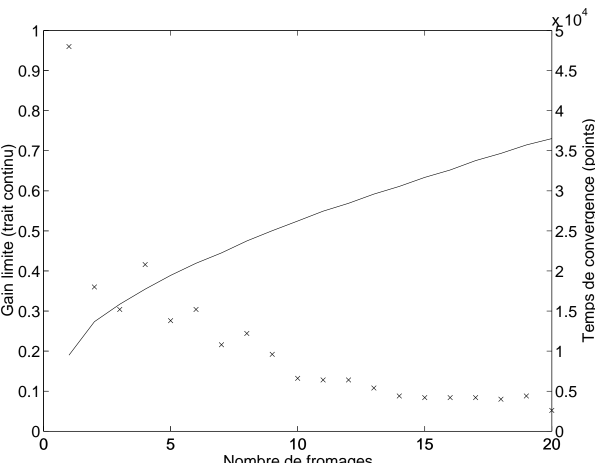



FIG. 5.7 – *Influence du nombre de fromages sur le gain limite et sur le temps de convergence (moyenne sur 100 simulations avec $\alpha = 1$, $\gamma = 0{,}5$ et $\epsilon = 0{,}1$).*

important pour envisager un apprentissage sur un système réel : le grand nombre d'objets devient un *avantage* pour le contrôleur.

L'autre observation concerne le gain limite qui augmente avec le nombre de fromages. Cette observation n'est pas surprenante. 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.

### 5.3.4 Synthèse

Sur cet exemple simple, le Q-Learning parallèle est stable et permet de réduire la complexité du système. De plus, le nombre d'objets n'est plus un problème, mais participe à un gain important de vitesse de convergence.

Il est difficile de dire si les stratégies obtenues sont proches ou non de l'optimale. Néanmoins, les observations directes du mouvement de la souris montrent que le contrôleur ne fait pas de détour entre les fromages, ce qui nous laisse penser que les trajectoires obtenues sont de bonne qualité. L'opérateur de fusion semble donc convenir dans ce cas.

## 5.4 Labyrinthe avec deux types d'objets

Dans cette partie, nous étudions les résultats de simulations dans le cas où des murs dispersés sont introduits dans le labyrinthe en plus des fromages (*cf.* figure 5.2b).

### 5.4.1 Comparaison avec le cas précédent

La figure 5.8 montre une comparaison entre les apprentissages sur le labyrinthe avec et sans mur. D'une part, la présence des murs augmente sensiblement le temps d'apprentissage. Cette augmentation est due à l'attribution des récompenses qui est plus complexe

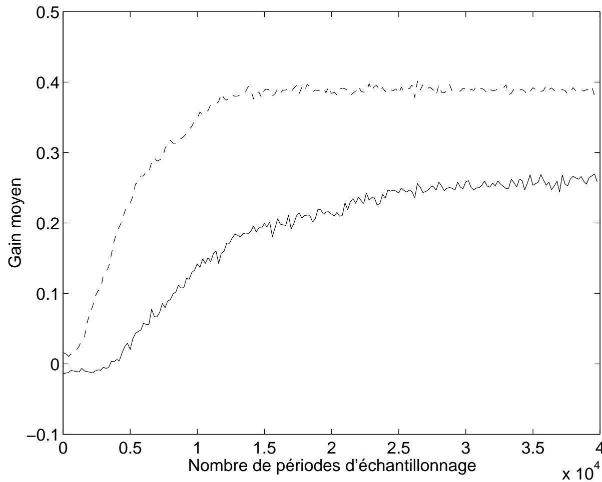



FIG. 5.8 – *Comparaison de l'évolution du gain moyen avec un labyrinthe contenant 10 murs (trait continu) et avec un labyrinthe sans mur (trait pointillé) (moyenne sur 100 simulations avec 5 fromages,* $\alpha = 1$*,* $\gamma = 0{,}5$ *et* $\epsilon = 0{,}1$*).*

car le contrôleur doit apprendre à différencier l'effet des murs et des fromages. D'autre part, le gain limite obtenu est inférieur aux résultats précédents. 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, ce qui montre que dans ce cas, la stratégie obtenue est sous-optimale.

### 5.4.2 Influence des paramètres

#### Influence du coefficient d'apprentissage $\alpha$

Le coefficient $\alpha$ n'a pas la même influence que dans le cas du labyrinthe sans mur (*cf.* figure 5.9). Le gain limite augmente régulièrement avec $\alpha$ tandis que le temps d'apprentissage diminue lentement.

#### Influence du coefficient de pondération $\gamma$

La courbe de la figure 5.10 montre un tracé similaire au cas du labyrinthe sans mur. Il faut éviter de choisir $\gamma$ proche de 1 car le labyrinthe est petit. En outre, le coefficient de pondération permet de modifier le comportement d'opportunisme du contrôleur.

### 5.4.3 Apprentissage progressif

De manière générale, la présence de murs ralentit l'apprentissage. Ce résultat est dû à l'attribution plus complexe des récompenses. En effet, le contrôleur doit apprendre à distinguer deux classes de perceptions qui produisent des récompenses différentes. Cet

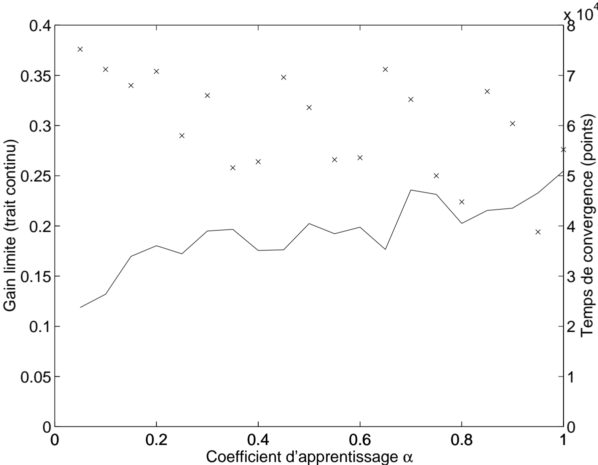



FIG. 5.9 – *Influence du coefficient d'apprentissage $\alpha$ sur le gain limite et sur le temps de convergence (moyenne sur 100 simulations avec 5 fromages, 10 murs, $\gamma = 0{,}5$ et $\epsilon = 0{,}1$).*

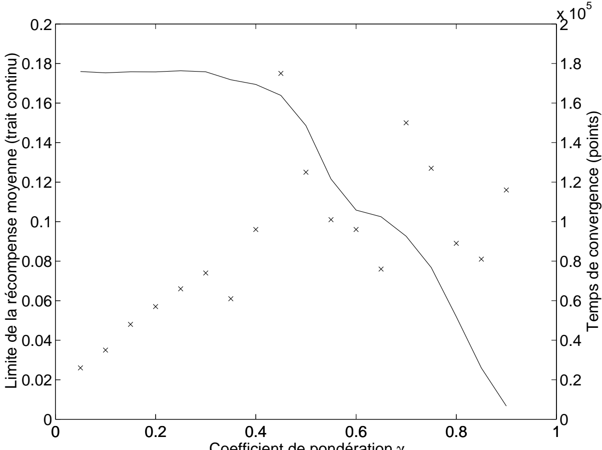



FIG. 5.10 – *Influence du coefficient de pondération $\gamma$ sur la limite de la récompense moyenne et sur le temps de convergence (moyenne sur 100 simulations avec 5 fromages, 10 murs, $\alpha = 0{,}1$ et $\epsilon = 0{,}1$).*

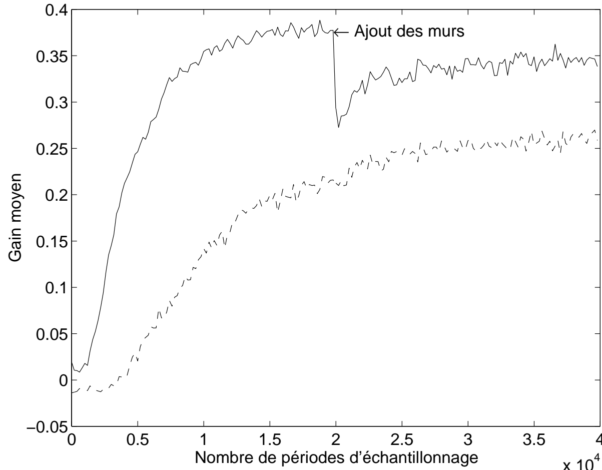



FIG. 5.11 – *Comparaison de l'apprentissage progressif (trait continu) et de l'apprentissage normal (trait pointillé) (moyenne sur 100 simulations avec 5 fromages, 10 murs,* $\alpha = 1$*,* $\gamma = 0{,}5$ *et* $\epsilon = 0{,}1$*).*

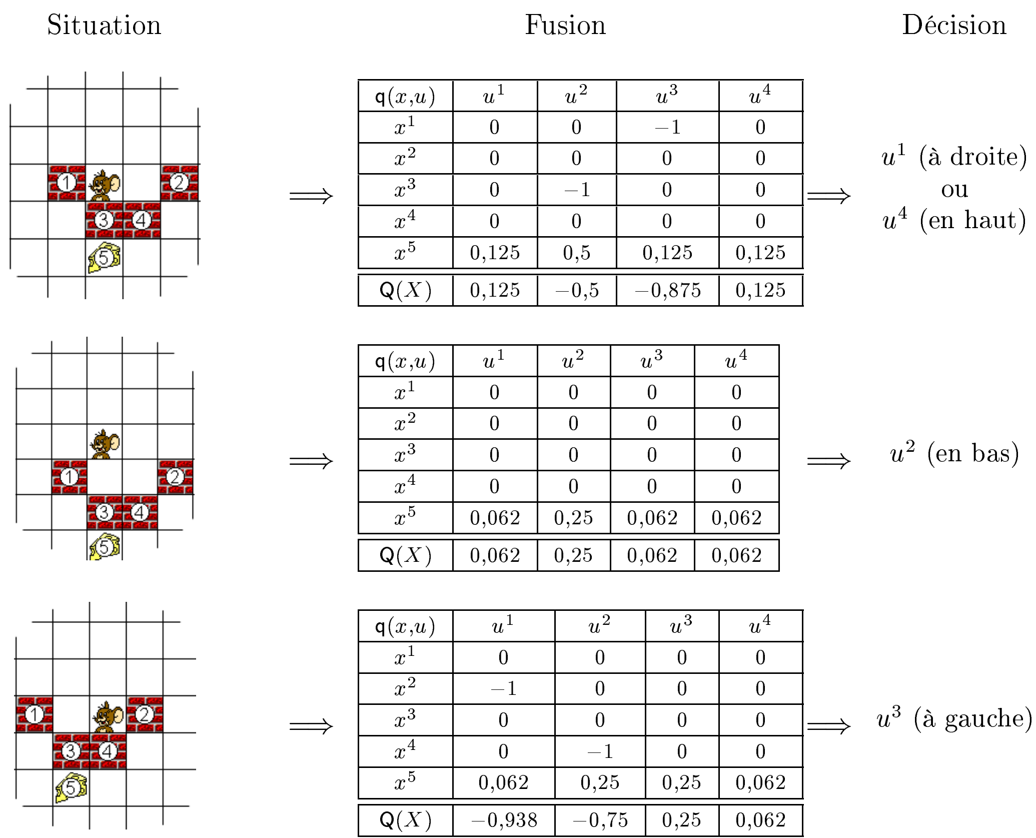
apprentissage est d'autant plus long que, si le contrôleur attribue par erreur une récompense négative à une perception, il aura tendance, par la suite, à éviter d'obtenir à nouveau cette perception. Il ne pourra donc pas vérifier immédiatement l'attribution de cette récompense négative comme cela se passe pour des récompenses positives.

Pour pallier ce problème, il suffit de procéder avec « pédagogie » : au lieu d'exiger du contrôleur qu'il apprenne tout en même temps, il est préférable de lui présenter d'abord un labyrinthe contenant uniquement des fromages, puis un labyrinthe contenant aussi des murs. Avec cette méthode progressive, l'apprentissage est beaucoup plus rapide (*cf.* figure 5.11) et la stratégie obtenue est de meilleure qualité.

### 5.4.4 Problème de fusion

La présence des murs révèle un inconvénient de la fonction de fusion. Si les murs sont agencés de manière à créer une impasse, la souris a tendance à y être piégée. La fonction de fusion génère un maximum local pour la fonction d'utilité globale, ce qui entraîne la formation d'un cycle de quelques états (*cf.* figure 5.12, voir aussi le logiciel de démonstration sur le CD-ROM décrite dans l'annexe A).

Néanmoins, par rapport aux 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 commandes menant aux états du cycle deviennent en effet de moins en moins attractives puisque le contrôleur ne reçoit plus de récompense. Le contrôleur finit donc par choisir d'autres commandes plus prometteuses. La courbe de la figure 5.13 illustre ce phénomène. Elle montre clairement deux modes de fonctionnement : les instants où la souris se bloque obtenant un gain négatif et les instants où la souris n'est pas bloquée et obtient un gain positif.




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

| $q(x,u)$ | $u^1$ | $u^2$ | $u^3$ | $u^4$ |
|---|---|---|---|---|
| $x^1$ | 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 |
| $Q(X)$ | 0,062 | 0,25 | 0,062 | 0,062 |

| $q(x,u)$ | $u^1$ | $u^2$ | $u^3$ | $u^4$ |
|---|---|---|---|---|
| $x^1$ | 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 |
| $Q(X)$ | $-0{,}938$ | $-0{,}75$ | 0,25 | 0,062 |

FIG. 5.12 – *Exemple de formation d'un cycle entre trois états (estimations des valeurs de q réalisées avec $\gamma = 0{,}5$).*

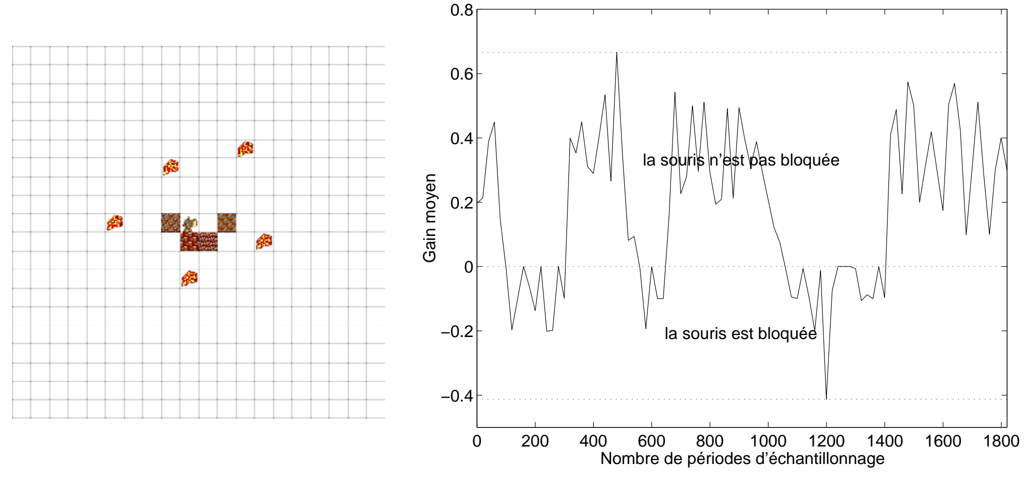



FIG. 5.13 – *Évolution du gain dans un labyrinthe avec 5 fromages et une impasse de 4 murs au centre du labyrinthe (avec $\alpha = 1$, $\gamma = 0{,}5$ et $\epsilon = 0{,}1$).*

### 5.4.5 Synthèse

Le temps d'apprentissage est plus long avec la présence de murs car le contrôleur doit apprendre à différencier les perceptions. Malgré tout, si on procède avec méthode, en ne présentant que des fromages puis des fromages et des murs, l'apprentissage est plus rapide et converge vers de meilleures stratégies.

Par ailleurs, les murs peuvent créer des situations complexes (une impasse par exemple) où la souris va se bloquer momentanément. La fonction de fusion produit dans ce cas un maximum local. Cependant, le processus d'apprentissage permet au contrôleur de s'échapper du maximum local au bout d'un certain temps, temps d'autant plus long que l'impasse est « profonde ».

## 5.5 Conclusion

Les résultats expérimentaux sur le labyrinthe montrent que le Q-Learning parallèle est stable et performant.

D'une part, la fonction d'attribution des récompenses fonctionne bien grâce à un processus d'auto-vérification de l'attribution. D'autre part, un grand nombre d'objets n'est plus un problème mais permet d'augmenter le nombre des expériences du contrôleur à chaque période d'échantillonnage, accélérant ainsi l'apprentissage.

En revanche, la fonction de fusion donne de bons résultats uniquement si les murs sont éloignés les uns des autres. En effet, elle peut créer momentanément des maxima locaux si les murs sont regroupés et forment une impasse. Ce problème sera résolu au chapitre 7 grâce à une nouvelle fonction de fusion fondée sur la modélisation introduite au chapitre 6.

Enfin, si le parallélisme a permis de raccourcir sensiblement le temps d'apprentissage, les temps obtenus sont encore trop importants pour envisager directement le contrôle d'une application réelle comme le WIMS. Nous avons donc modifié notre architecture de manière à la rendre encore plus dynamique en utilisant un algorithme de renforcement fondé sur le Dyna-Q.

# Chapitre 6

# Dyna-Q parallèle et application à la commande du macro-manipulateur WIMS

*Ce chapitre décrit le fonctionnement de l'algorithme du Dyna-Q parallèle et présente les résultats expérimentaux obtenus sur l'exemple du labyrinthe et sur le macro-manipulateur WIMS. Ces études montrent que le Dyna-Q parallèle accélère suffisamment l'apprentissage pour permettre la commande de systèmes réels comme le macro-manipulateur WIMS.*

## 6.1 Introduction

L'objectif du Dyna-Q parallèle est d'augmenter la vitesse de convergence de l'apprentissage pour envisager la commande d'applications réelles telles que le macro-manipulateur WIMS décrit dans le chapitre 1.

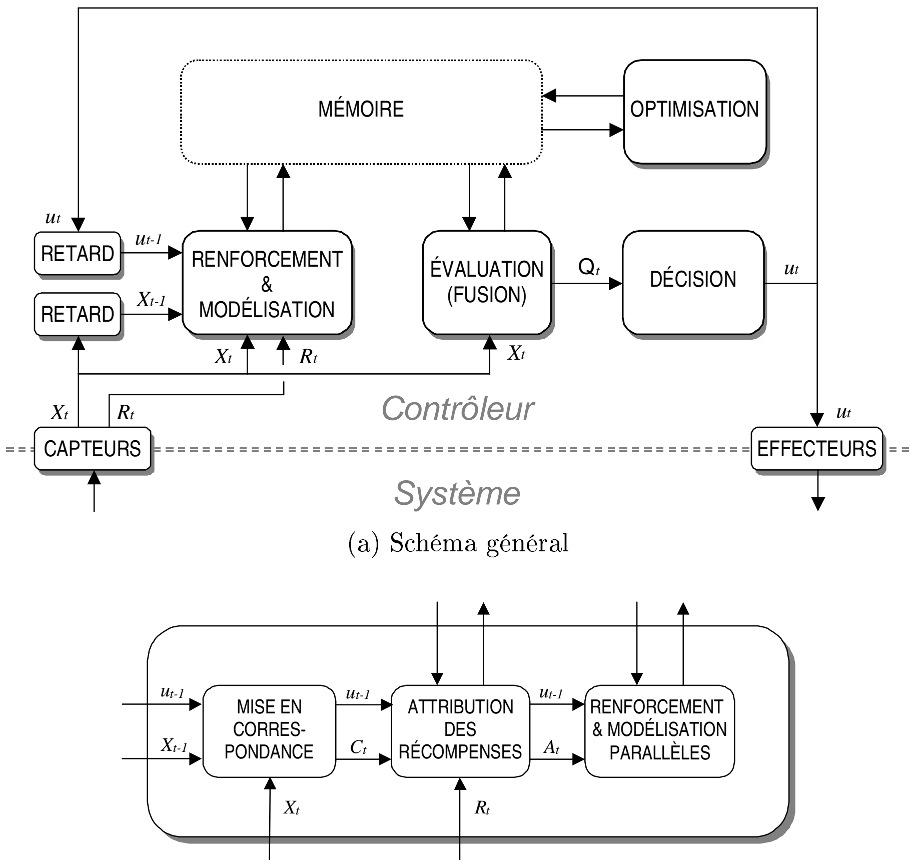
A la différence du Q-Learning parallèle, le Dyna-Q parallèle utilise une méthode d'apprentissage par renforcement indirecte pour optimiser la fonction d'utilité $\mathsf{q}$. Comme son nom l'indique, il est fondé sur l'algorithme de Richard Sutton (*cf.* §2.5.2, page 34).

Ce chapitre présente l'algorithme du Dyna-Q parallèle ainsi que les tests expérimentaux réalisés sur le labyrinthe et sur le macro-manipulateur WIMS.

## 6.2 Dyna-Q parallèle

### 6.2.1 Introduction

Comme le Dyna-Q original de Richard Sutton ne permet pas la commande de systèmes non déterministes, nous l'avons modifié pour le rendre capable de contrôler des systèmes légèrement stochastiques. L'idée développée est de mémoriser pour chaque couple *perception—commande* visité non seulement le dernier couple *perception—récompense*




(a) Schéma général

(b) Détail de la fonction de renforcement et modélisation

FIG. 6.1 – *Schéma fonctionnel du Dyna-Q parallèle.*

observé, mais aussi les $h$ derniers couples. Cette solution est moins coûteuse en espace mémoire que le stockage de la matrice complète de transition.

### 6.2.2 Fonctionnement

Le Dyna-Q parallèle utilise une méthode indirecte : une fonction de modélisation enrichit un modèle partiel du système au fur et à mesure des expériences, puis une fonction d'optimisation met à jour la fonction d'utilité à l'aide de ce modèle. La figure 6.1 synthétise le fonctionnement du Dyna-Q parallèle et la figure 6.2 présente son algorithme global. Les fonctions de mise en correspondance, d'attribution des récompenses, de renforcement parallèle, de fusion et de décision sont identiques à celles du Q-Learning parallèle. Seules diffèrent les fonctions de mémorisation, d'optimisation asynchrone et de modélisation parallèle.

**Entrée :** $X_{t-1}$, $u_{t-1}$, $X_t$, $R_t$
**Sortie :** $u_t$
**Début**
$C_t \leftarrow$MiseEnCorrespondance($X_{t-1}$,$X_t$)
$A_t \leftarrow$Attribution($C_t$,$R_t$,$u_{t-1}$,q)
RenforcementParallèle($A_t$,$u_{t-1}$,q)
ModélisationParallèle($A_t$,$u_{t-1}$,m)
Optimisation(m,q)
Q$_t \leftarrow$Fusion($X_t$,q)
$u_t \leftarrow$Décision(Q$_t$)
**Fin**

FIG. 6.2 – *Algorithme du Dyna-Q parallèle.*

### 6.2.3 Fonction mémorisation

Outre la fonction d'utilité $q$, la mémoire contient un modèle noté m qui permet de stocker un historique de $h$ couples *perception—récompense* $(x',r)$ pour chaque couple *perception—commande* $(x,u)$. Ces couples *perception—récompense* $(x',r)$ correspondent aux perceptions et aux récompenses associées à une perception $x$ par la fonction de mise en correspondance et par la fonction d'attribution des récompenses lors des derniers passages où le contrôleur observait la perception $x$ et où la commande $u$ a été effectuée. Le nombre $h$ de couples $(x',r)$ mémorisés pour un couple $(x,u)$ donné est appelé dimension de l'historique. $h$ est identique pour tous les couples $(x,u)$ de $\mathcal{X} \times \mathcal{U}$.

Le modèle $\mathsf{m}(x,u)$ est donc un multi-ensemble[1] de couples dont le cardinal est au plus $h$. Si $h = 5$, on a par exemple :

$$\mathsf{m}(x,u) = \big\{(x'_{t_1},r_{t_1}),(x'_{t_2},r_{t_2}),(x'_{t_3},r_{t_3}),(x'_{t_4},r_{t_4}),(x'_{t_5},r_{t_5})\big\} \tag{6.1}$$

avec $t_1 > t_2 > t_3 > t_4 > t_5$. Chaque couple $(x'_{t_i},r_{t_i})$ correspond à une observation passée de la perception et de la récompense obtenues quand le contrôleur a effectué la commande $u$ alors qu'il observait la perception $x$. Les couples peuvent être différents à cause de la nature non déterministe du système.

La dimension $h$ de l'historique est choisie en rapport avec le degré de non déterminisme du système. Si les transitions des perceptions sont complètement déterministes, on peut choisir $h = 1$, sinon il faut trouver une valeur adaptée au système.

Avant de commencer l'apprentissage, les fonctions q et m sont respectivement initialisées à zéro et à vide pour tout $(x,u)$ de $\mathcal{X} \times \mathcal{U}$.

### 6.2.4 Fonction de modélisation parallèle

1. Ensemble dans lequel un même élément peut apparaître plusieurs fois. Autrement dit, la multiplicité d'un élément d'un multi-ensemble peut être supérieure à 1.

**Entrée :** $A_t$, $u_{t-1}$
**Entrée-sortie :** m
**Début**
  **Pour tout** $(x,x',r)$ de $A_t$ **faire**
    **Si** $|\mathsf{m}(x,u_{t-1})| \geq h$ **alors**
      Retirer le couple le plus ancien de $\mathsf{m}(x,u_{t-1})$
    **Fin si**
    $\mathsf{m}(x,u_{t-1}) \leftarrow \mathsf{m}(x,u_{t-1}) + \{(x',r)\}$ *(concaténation)*
  **Fin pour**
**Fin**

FIG. 6.3 – *Algorithme de la fonction de modélisation parallèle.*

A chaque instant $t$, la fonction de modélisation utilise les informations fournies par la fonction d'attribution pour enrichir le modèle. Pour chaque triplet $(x,x',r)$ de l'ensemble $A_t$, elle ajoute par concaténation à $\mathsf{m}(x,u_{t-1})$ le couple $(x',r)$. Si le cardinal de $\mathsf{m}(x,u_{t-1})$ est supérieur à $h$, alors elle retire de $\mathsf{m}(x,u_{t-1})$ le couple le plus ancien. Cet algorithme est décrit par la figure 6.3.

### 6.2.5 Fonction d'optimisation asynchrone

Le stockage d'un historique pour chaque couple permet de calculer une approximation des probabilités $\mathsf{p}$ de transition en utilisant la multiplicité d'un couple, soit :

$$\mathsf{p}(x'|x,u) \approx \sum_{\forall (y,r) \in \mathsf{m}(x,u) | y=x'} \frac{1}{|\mathsf{m}(x,u)|} \tag{6.2}$$

A l'aide de cette approximation, la fonction d'utilité peut être optimisée sans utiliser le coefficient d'apprentissage $\alpha$, même dans le cas où le système n'est pas déterministe. L'équation de mise à jour pour un couple $(x,u)$ s'écrit alors :

$$\mathsf{q}(x,u) \leftarrow \sum_{\forall (y,r) \in \mathsf{m}(x,u)} \frac{1}{|\mathsf{m}(x,u)|} \left[ r + \gamma \max_{v \in \mathcal{U}} \mathsf{q}(y,v) \right] \tag{6.3}$$

Les fonctions d'optimisation et de renforcement vont donc utiliser cette nouvelle équation.

La figure 6.4 présente l'algorithme de cette fonction. Le nombre $N$ règle le nombre de mises à jour par appel. *A priori*, plus $N$ est grand, plus l'apprentissage est rapide, mais le temps de calcul nécessaire peut alors être trop long pour réaliser un contrôle en temps réel. Dans ce cas, il faut choisir une valeur qui réalise un bon compromis entre vitesse de convergence et temps de calcul par période d'échantillonnage. Si la fréquence d'échantillonnage est faible, une autre solution peut être envisagée : il s'agit de réaliser les calculs d'optimisation en temps masqué pendant l'exécution de la dernière commande.

**Entrée :** m
**Entrée-sortie :** q
**Début**
  **Répéter** $N$ fois
    Choisir $(x,u)$ au hasard tel que $\mathsf{m}(x,u) \neq \emptyset$
    $$\mathsf{q}(x,u) \leftarrow \sum_{\forall (y,r) \in \mathsf{m}(x,u)} \frac{1}{|\mathsf{m}(x,u)|} \left[ r + \gamma \max_{v \in \mathcal{U}} \mathsf{q}(y,v) \right]$$
  **Fin Répéter**
**Fin**

FIG. 6.4 – *Algorithme de la fonction d'optimisation asynchrone.*

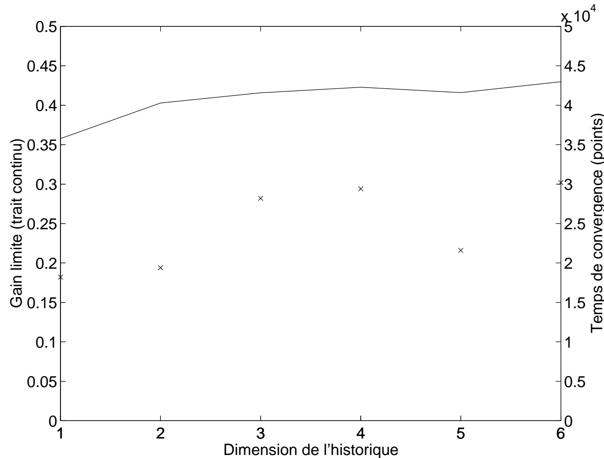



FIG. 6.5 – *Influence de la dimension h de l'historique sur le gain limite et sur le temps de convergence (moyenne sur 50 simulations avec 5 fromages, 10 murs, $N = 3000$, $\gamma = 0{,}5$ et $\epsilon = 0{,}1$).*

## 6.3 Application du Dyna-Q parallèle à l'exemple du labyrinthe

### 6.3.1 Influence du coefficient de pondération $\gamma$

Le coefficient de pondération $\gamma$ joue le même rôle avec le Q-Learning parallèle et avec le Dyna-Q parallèle : il permet de régler le degré d'opportunisme du contrôleur.

### 6.3.2 Influence de la dimension $h$ de l'historique

La figure 6.5 montre l'influence de la dimension $h$ de l'historique sur le gain limite et sur le temps de convergence. $h$ n'a quasiment pas d'influence sauf lorsqu'il est égal à 1. Dans ce cas, le gain limite obtenu est inférieur.

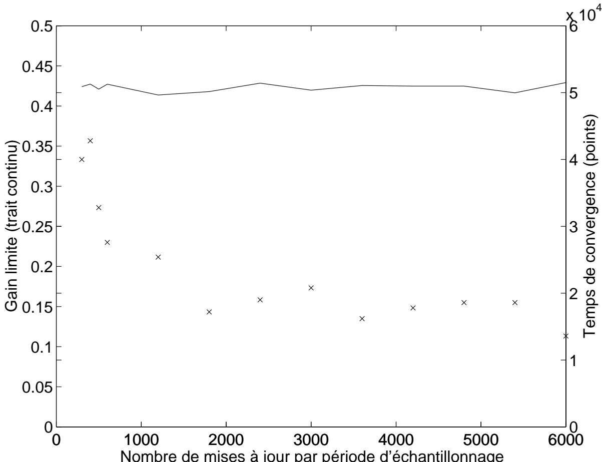



FIG. 6.6 – *Influence du nombre de mises à jour par période d'échantillonnage sur le gain limite et sur le temps de convergence (moyenne sur 40 simulations avec 5 fromages, 10 murs,* $h = 5$*,* $\gamma = 0{,}5$ *et* $\epsilon = 0{,}1$*).*

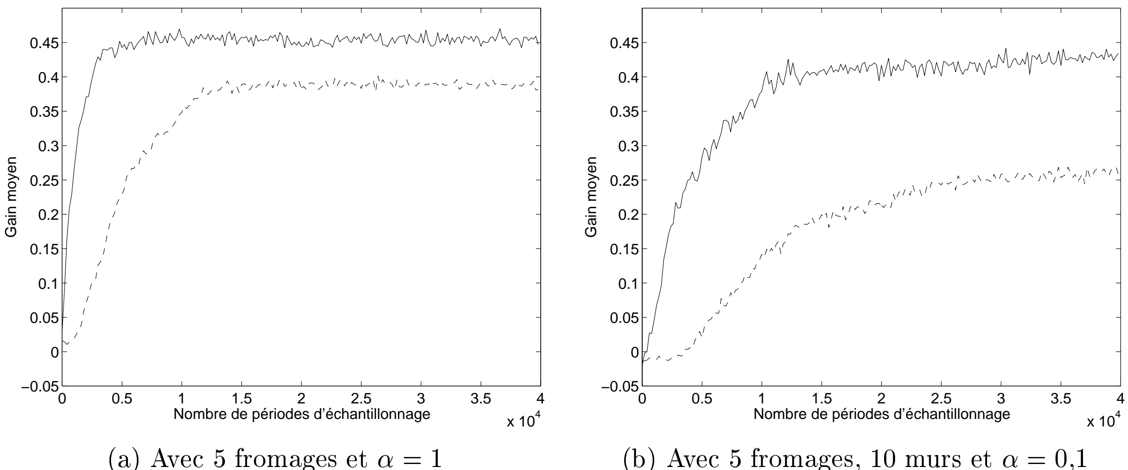
Cette observation s'explique par les interactions qui peuvent se produire entre la souris et les murs. Ces interactions rendent les transitions des perceptions légèrement non déterministes. Par exemple, si la souris est bloquée par un mur, les fromages ne bougent plus par rapport à elle quand elle essaie d'avancer, ce qui n'est pas habituel (*cf.* figure 4.4, page 65). Ainsi à cause des murs, les perceptions peuvent être figées, ce qui engendre des transitions non déterministes. A cause de ces interactions, il vaut mieux choisir, même pour un système *a priori* déterministe, une dimension d'historique strictement supérieure à 1.

### 6.3.3 Influence du nombre $N$ de mises à jour

D'après la courbe de la figure 6.6, plus le nombre de mises à jour est grand, plus le contrôleur apprend vite. La décroissance du temps d'apprentissage est très rapide puis s'estompe vers 2000 mises à jour. Il est donc inutile d'effectuer un très grand nombre de mises à jour. Il suffit de choisir $N$ aux alentours de 2000 pour obtenir une bonne vitesse de convergence sans perdre de temps en calculs superflus.

### 6.3.4 Comparaison du Q-Learning parallèle et du Dyna-Q parallèle

La figure 6.7 montre les résultats comparatifs entre le Q-Learning parallèle et le Dyna-Q parallèle. Le Dyna-Q parallèle converge plus rapidement et vers une meilleure stratégie. L'amélioration du temps de convergence était attendue car les méthodes indirectes comme le Dyna-Q permettent de réaliser un plus grand nombre de mises à jour par période d'échantillonnage. Habituellement, cette augmentation de la vitesse se fait au détriment de la qualité de la stratégie de commande obtenue car le contrôleur a moins de temps pour visiter les états (*cf.* §2.4.3, page 29). Dans notre approche, la présence de




(a) Avec 5 fromages et $\alpha = 1$

(b) Avec 5 fromages, 10 murs et $\alpha = 0{,}1$

FIG. 6.7 – *Comparaison de l'évolution des gains moyens obtenus avec le Dyna-Q parallèle (traits continus) et avec le Q-Learning parallèle (traits discontinus) (moyenne sur 40 simulations avec $N = 2000$, $h = 5$, $\gamma = 0{,}5$ et $\epsilon = 0{,}1$).*

plusieurs perceptions permet au contrôleur de visiter continuellement d'autres perceptions, tout en satisfaisant au critère de la maximisation de la somme des récompenses. *Le Dyna-Q parallèle permet donc non seulement de réaliser une amélioration importante de la vitesse de convergence, mais aussi de trouver de meilleures stratégies de contrôle.*

### 6.3.5 Synthèse

Le Dyna-Q parallèle nécessite une puissance de calcul plus importante que le Q-Learning parallèle mais apporte en contrepartie un important gain en vitesse de convergence. De plus, les stratégies obtenues sont plus efficaces. Ces améliorations permettent d'envisager le contrôle d'applications réelles telles que le macro-manipulateur WIMS.

Enfin, l'emploi d'un modèle partiel sauvegardant un historique limité pour chaque couple *perception—commande* permet d'optimiser la fonction d'utilité sans utiliser de coefficient d'apprentissage $\alpha$. Le choix du coefficient $\alpha$ est alors remplacé par celui du paramètre $h$.

## 6.4 Application à la commande du macro-manipulateur WIMS

### 6.4.1 Définition du cadre expérimental

#### Boucle sensori-motrice

La figure 6.8 résume les principales fonctions du système WIMS décrit dans le premier chapitre et dans l'annexe B. La commande reçue provoque le déplacement de l'aimant *via* une table de micro-positionnement deux axes. Grâce au champ magnétique généré par l'aimant, l'outil suit le déplacement de l'aimant. L'outil interagit alors avec les objets

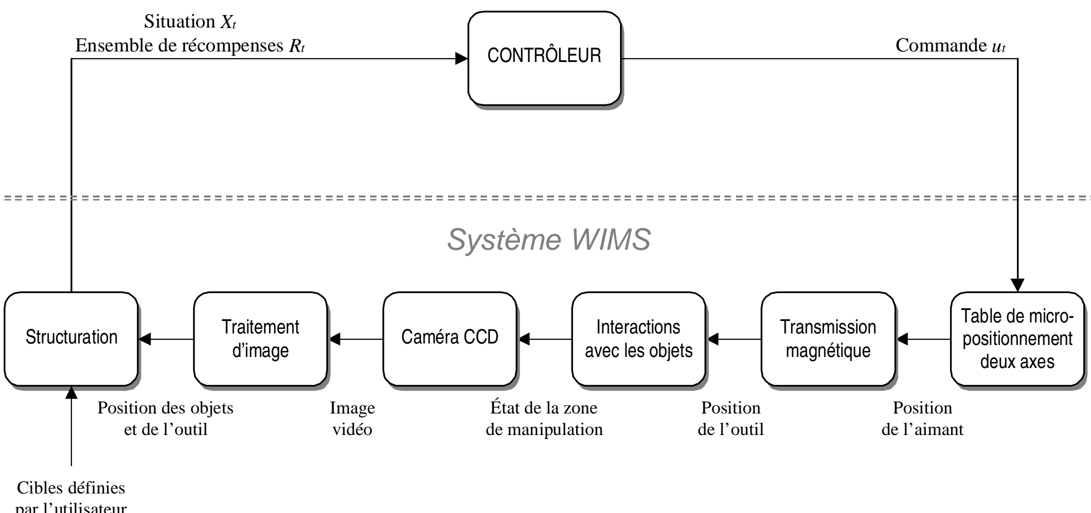



FIG. 6.8 – *Boucle sensori-motrice de commande du macro-manipulateur WIMS.*

présents dans la zone de manipulation. Une caméra CCD transmet des images de la zone de manipulation à un module de traitement d'image. Le traitement de l'image permet d'extraire la position du barycentre de chaque objet ainsi que celle du barycentre de l'outil. Enfin, le module de structuration transforme les informations issues du module de traitement d'image et les cibles définies par un utilisateur extérieur en une situation et un ensemble de récompenses.

### Définition des commandes

Nous avons retenu l'emploi de 8 commandes discrètes permettant de déplacer l'outil dans les quatre directions et selon deux pas différents : un pas fin pour les positionnements précis et un pas grossier pour effectuer des grands déplacements. Les commandes fines déplacent l'outil d'environ 1,9 mm (soit environ 11 pixels), ce qui correspond à peu près à la moitié du diamètre d'un objet. Les commandes grossières déplacent l'outil d'environ 5,7 mm (soit environ 33 pixels).

Le choix de ces valeurs est motivé par un double objectif :

- pouvoir effectuer des déplacements conséquents à chaque période d'échantillonnage pour obtenir une vitesse d'évolution importante dans les zones vides,
- garantir une précision de positionnement suffisante vis-à-vis des objets et des cibles.

De plus, nous avons réduit l'ensemble des commandes à 8 commandes élémentaires pour que le contrôleur apprenne plus rapidement, mais on peut envisager l'ajout d'autres commandes comme par exemple des déplacements plus fins, plus importants ou en diagonale...

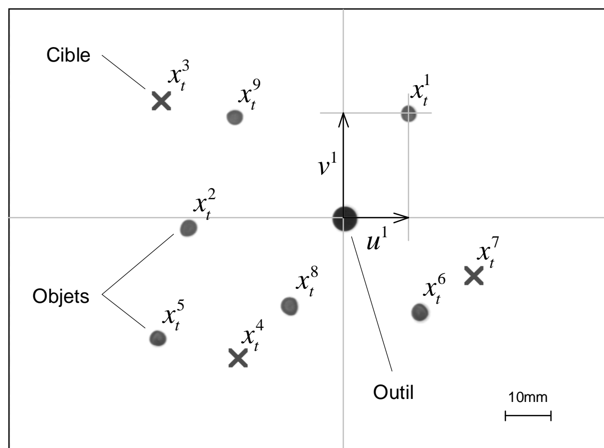



FIG. 6.9 – *Exemple de situation de manipulation sur le WIMS.*

**Définition des perceptions**

La figure 6.9 montre une image vidéo de la zone de manipulation sur laquelle sont ajoutées les cibles virtuelles définies par l'utilisateur.

Une perception $x_t^i$ d'une situation $X_t$ décrit la position $(u^i,v^i)$ dans le repère lié à l'outil et le type $w^i$ d'un objet sous la forme d'un triplet $(u^i,v^i,w^i)$ (objet réel : $w^i = 1$, ou cible virtuelle : $w^i = 2$). L'ensemble $\mathcal{X}$ des perceptions correspond à toutes les positions possibles dans le repère lié à l'outil et aux différents types (objet ou cible).

L'image est découpée en cases selon un pas régulier. Ce pas caractérise la résolution de positionnement des objets et des cibles. Le pas le plus fin peut valoir un seul pixel (soit environ 0,173 mm), mais il est inutile de choisir une résolution supérieure au plus petit déplacement de l'outil. Dans notre cas, nous utilisons donc un pas de 11 pixels (soit environ 1,9 mm) qui correspond aux commandes de déplacements fins.

Avec ce découpage, la position d'une cible ou d'un objet $(u^i,v^i)$ peut varier entre les points extrêmes $(-68,-50)$ et $(68,50)$ dans le repère lié à l'outil. Les coordonnées sont des nombres entiers relatifs qui correspondent au nombre de cases qui séparent une cible ou un objet de l'outil. Le cardinal de $\mathcal{X}$ est égal à 26 730 perceptions ($\mathcal{X} = \{(u,v,w) \mid -68 \leq u \leq 68, -50 \leq j \leq 50, 1 \leq w \leq 2,\}$, donc $|\mathcal{X}| = 2\,(68+68-1)\,(50+50-1) = 26\,730$). Sans la parallélisation, le cardinal de l'espace d'états serait de la forme $[2(68+68-1)(50+50-1)]^n$ avec $n$ objets.

**Définition des récompenses**

Nous avons choisi les deux objectifs suivants :
- l'outil doit atteindre les cibles définies par l'utilisateur,
- l'outil doit éviter les objets réels.

Le premier objectif se traduit par un comportement d'attraction globale (*cf.* §2.4.2, page 22). Pour obtenir ce comportement, on définit les récompenses vis-à-vis d'une cible

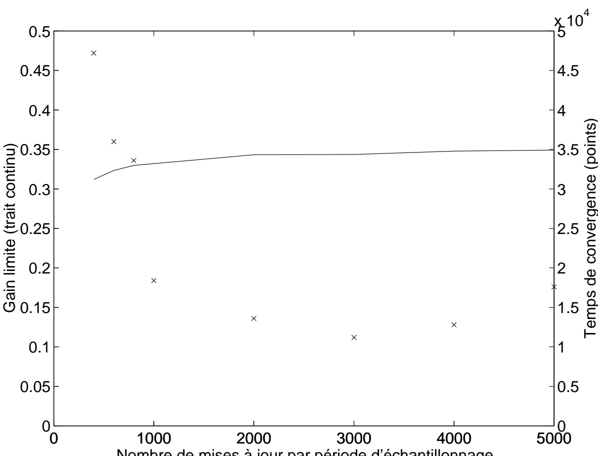



FIG. 6.10 – *Influence du nombre de mises à jour par période d'échantillonnage sur le gain limite et sur le temps de convergence (moyenne sur 10 simulations avec 10 cibles,* $h = 10$*,* $\gamma = 0{,}5$ *et* $\epsilon = 0{,}1$*).*

telles que si à l'instant $t$, l'outil atteint la cible, une récompense positive égale à 1 est ajoutée à l'ensemble $R_t$ des récompenses, sinon une récompense nulle est ajoutée à l'ensemble des récompenses. Quand l'outil a atteint une cible, celle-ci disparaît à la période d'échantillonnage suivante et est replacée à une position aléatoire au coup d'après.

Le second objectif correspond à un comportement de répulsion locale (*cf.* §2.4.2, page 22). Pour obtenir ce comportement, on définit les récompenses vis-à-vis d'un objet telles que si l'outil s'approche à moins de 11 pixels d'un objet (soit environ une marge de sécurité de 1,9 mm), une récompense négative égale à $-1$ est ajoutée à l'ensemble des récompenses, sinon une récompense nulle est ajoutée à l'ensemble des récompenses.

**Simulation**

Nous avons eu recours à une simulation logicielle afin de pouvoir tester les algorithmes avant de les utiliser avec le système réel. Cette simulation reproduit les déplacements de l'outil, le phénomène d'hystérésis entre l'aimant et l'outil, et les interactions entre l'outil et les objets. Les effets de stick-slip (*cf.* §1.1.2, page 3) ne sont pas modélisés. Le logiciel développé est décrit en annexe B.

L'objectif de la simulation n'est pas de reproduire fidèlement le système, mais bien d'avoir un outil pour pouvoir valider un algorithme avant de le tester sur le système réel.

### 6.4.2 Apprentissage avec uniquement des cibles

**Influence des paramètres (tests avec le système simulé)**

A l'aide de la simulation du système, nous avons étudié les influences du nombre de mises à jour et de la dimension de l'historique sur l'apprentissage. Les résultats sont

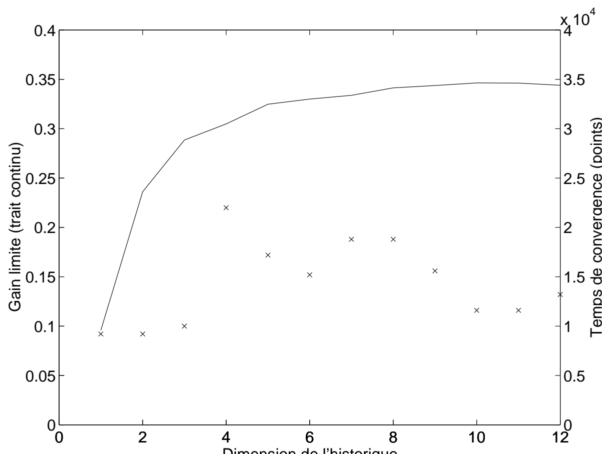



FIG. 6.11 – *Influence de la dimension de l'historique sur le gain limite et sur le temps de convergence (moyenne sur 10 simulations avec 10 cibles,* $N = 2000$*,* $\gamma = 0{,}5$ *et* $\epsilon = 0{,}1$*).*

qualitativement similaires aux courbes obtenues avec le labyrinthe (*cf.* figures 6.10 et 6.11).

Le temps d'apprentissage diminue rapidement avec l'augmentation du nombre de mises à jour. A partir de 2000 mises à jour, cette diminution ralentit. Il est donc souhaitable de limiter le nombre de mises à jour à quelques milliers ($N = 2000$ par exemple).

Le gain limite obtenu augmente avec la dimension de l'historique. La variation du gain est importante car le système est non déterministe (à cause de l'hystérésis entre l'aimant et l'outil). La courbe se stabilise pour des dimensions supérieures à 10. Nous retenons donc cette valeur pour les expérimentations suivantes ($h = 10$).

### Résultats avec le système simulé

La figure 6.12 montre la courbe d'apprentissage sur le système simulé. Environ 12 000 périodes d'échantillonnage sont nécessaires pour d'obtenir une bonne stratégie de commande. Sur le système réel, une période d'échantillonnage dure 500 ms. Ainsi, le contrôleur devrait apprendre à piloter le système réel en moins de deux heures (sous l'hypothèse que le temps de convergence avec le système réel soit comparable à celui obtenu avec le système simulé).

La figure 6.13 présente un exemple classique de trajectoire que l'on obtient après l'apprentissage. L'outil atteint successivement les cibles sans perdre de temps mis à part de légers piétinements dus à l'hystérésis inhérent au système.

### Résultats avec le système réel

La courbe 6.14a montre l'évolution du gain limite moyen obtenu avec le système réel. Le contrôleur trouve une stratégie correcte en 7 000 périodes d'échantillonnage, soit environ une heure d'essais. La courbe est beaucoup plus irrégulière que les courbes

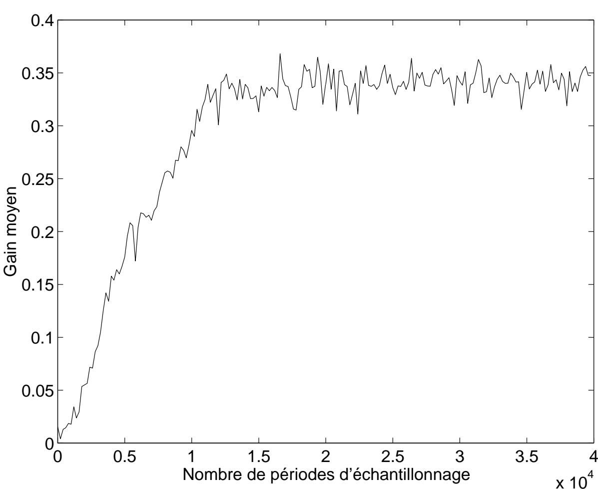



FIG. 6.12 – *Évolution du gain moyen avec le système simulé avec 10 cibles (moyenne sur 10 simulations avec* $h = 10$, $N = 2000$, $\gamma = 0{,}5$ *et* $\epsilon = 0{,}1$*).*

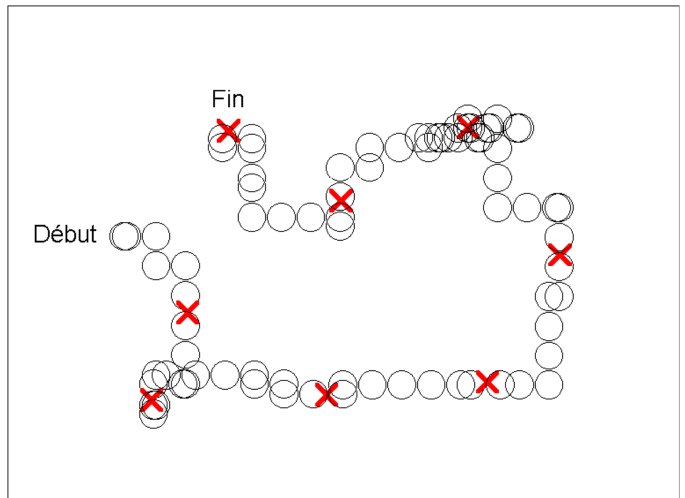
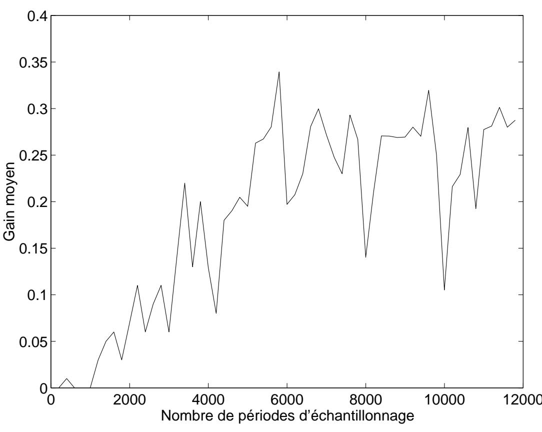
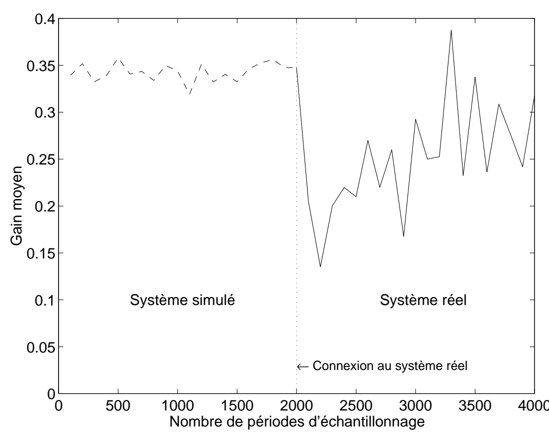



FIG. 6.13 – *Exemple de trajectoire de l'outil sur le système simulé après apprentissage (avec* $h = 10$, $N = 2000$, $\gamma = 0{,}5$ *et* $\epsilon = 0{,}1$*).*




(a) Apprentissage sur le système réel




(b) Apprentissage sur le système réel après un apprentissage sur la simulation

FIG. 6.14 – *Évolution du gain moyen avec le système réel avec 10 cibles (avec* $h = 10$*,* $N = 2000$*,* $\gamma = 0{,}5$ *et* $\epsilon = 0{,}1$*).*

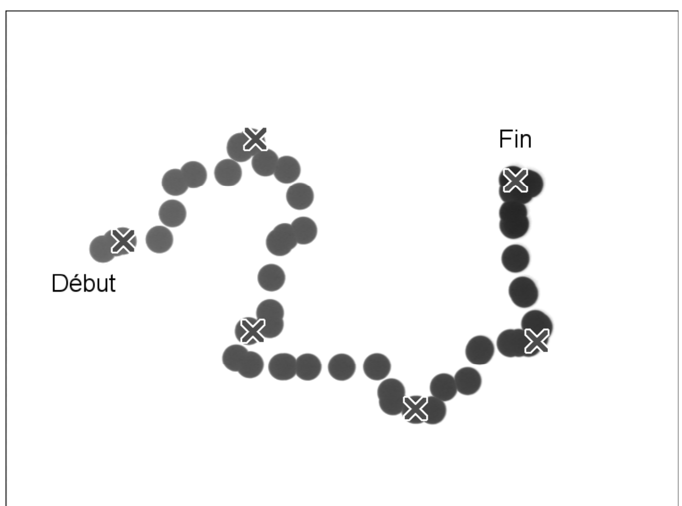



FIG. 6.15 – *Exemple de trajectoire de l'outil sur le système réel après apprentissage (avec* $h = 10$*,* $N = 2000$*,* $\gamma = 0{,}5$ *et* $\epsilon = 0{,}1$*).*

obtenues par simulation car elle est le résultat d'un unique essai et non une moyenne de nombreux essais comme c'est le cas pour les simulations. Par exemple, une courbe issue d'un unique essai peut montrer des chutes momentanées du gain moyen. En effet, comme le processus de positionnement des cibles est stochastique, il peut arriver que le contrôleur ait atteint toutes les cibles d'une zone et que les nouvelles cibles soient regroupées à l'opposé de l'aire de manipulation. Ainsi, le contrôleur n'obtient plus de récompense pendant qu'il dirige l'outil vers les cibles, ce qui entraîne un gain moyen quasiment nul pendant ce temps.

Après l'apprentissage, les trajectoires obtenues (*cf.* figure 6.15) montrent que le contrôleur parvient à contrôler réellement et efficacement le système (voir aussi la vidéo sur le CD-ROM décrite dans l'annexe C). Par ailleurs, on remarque que le contrôleur utilise les grands déplacements lorsqu'il est loin d'une cible et les déplacement fins lorsqu'il en est proche.

Afin d'accélérer le processus d'apprentissage avec le système réel, la mémoire du contrôleur peut être initialisée avec la fonction d'utilité et le modèle obtenus après apprentissage sur le système simulé. Le contrôleur adaptera ensuite ces données au système réel qui est sans doute différent du système simulé. La courbe de la figure 6.14b montre l'évolution du gain moyen obtenu par le contrôleur sur le système réel après un apprentissage sur le système simulé. Pendant la phase d'adaptation qui dure environ un millier de périodes d'échantillonnage (environ 8 minutes), le contrôleur obtient moins de récompenses car il apprend le comportement du système réel. Cette méthode permet donc d'accélérer l'apprentissage sur le système réel, mais suppose que l'on dispose d'une simulation du système.

### 6.4.3 Apprentissage avec des cibles et des objets

Après que le contrôleur a appris à atteindre correctement les cibles, on ajoute des obstacles sur l'aire de manipulation. La figure 6.16 montre la réaction du gain moyen à l'ajout des obstacles. A la fin de l'apprentissage, le contrôleur sait éviter les obstacles comme le montre la figure 6.17 (voir aussi la vidéo sur le CD-ROM décrite dans l'annexe C). Malgré tout, les gains obtenus sont inférieurs car l'outil doit parcourir plus de chemin entre chaque cible.

### 6.4.4 Test de robustesse

Ce paragraphe a pour objectif de montrer la capacité du contrôleur à s'adapter à une modification importante du comportement du système. Dans ce but, nous avons tourné le manipulateur de 45° par rapport à l'axe de l'image vidéo après que le contrôleur a appris à atteindre les cibles. Cette rotation dévie la trajectoire de l'outil. Comme le montre la courbe de la figure 6.18, après un certain temps le contrôleur s'est adapté au changement et obtient des résultats similaires à ceux précédant la rotation.

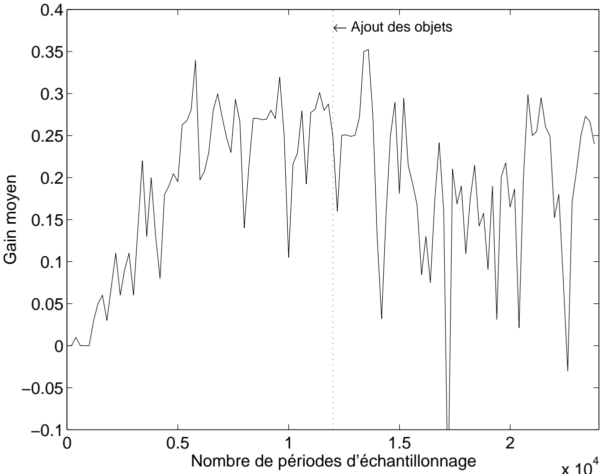



Fig. 6.16 – *Évolution du gain moyen avec le système réel après l'ajout d'obstacles et après apprentissage avec 10 cibles (avec* $h = 10$*,* $N = 2000$*,* $\gamma = 0{,}5$ *et* $\epsilon = 0{,}1$*).*

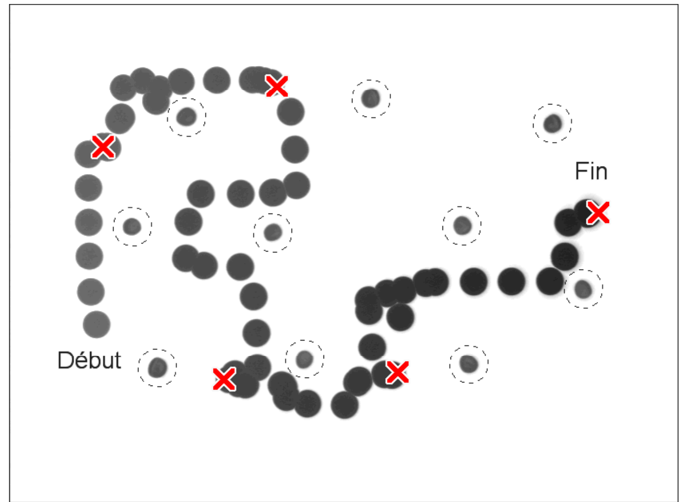



Fig. 6.17 – *Exemple de trajectoire de l'outil sur le système réel après apprentissage (avec* $h = 10$*,* $N = 2000$*,* $\gamma = 0{,}5$ *et* $\epsilon = 0{,}1$*).*

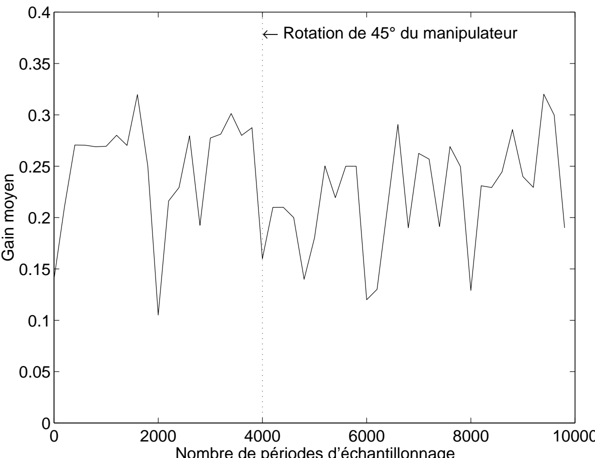



FIG. 6.18 – *Évolution du gain moyen avec le système réel après la rotation de 45° du manipulateur (avec $h = 10$, $N = 2000$, $\gamma = 0{,}5$ et $\epsilon = 0{,}1$).*

### 6.4.5 Synthèse

A l'aide de la simulation, nous avons déterminé des valeurs pour la dimension $h$ de l'historique et le nombre $N$ de mises à jour. Ces valeurs sont choisies de manière à obtenir des performances suffisantes de l'algorithme sans dépenser trop de temps de calcul.

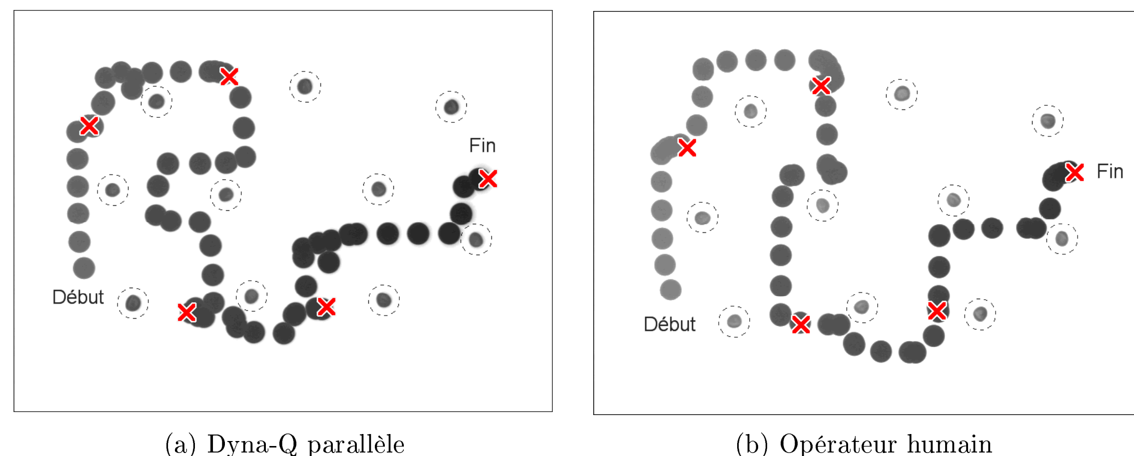
Les expériences menées avec le système WIMS réel montrent que le contrôleur est capable d'apprendre à commander le macro-manipulateur de manière satisfaisante pour atteindre des cibles en évitant des obstacles. En outre, le contrôleur s'adapte rapidement à un changement sensible du comportement du système comme une rotation de 45° du manipulateur.

## 6.5 Conclusion

Les tests effectués sur le labyrinthe montrent que le Dyna-Q parallèle permet non seulement d'apprendre plus vite mais aussi d'obtenir de meilleures stratégies de contrôle que le Q-Learning parallèle.

L'amélioration de la vitesse de convergence permet d'apprendre directement sur le système réel et le contrôleur est capable de s'adapter rapidement à un changement sensible du comportement du système.

La qualité des stratégies obtenues est difficile à juger : comme le modèle du système est inconnu, il est impossible de calculer une stratégie optimale de référence. Si l'on compare une trajectoire réalisée par le contrôleur après apprentissage et une trajectoire réalisée manuellement par un opérateur humain (*cf.* figure 6.19), la trajectoire humaine est plus directe mais la différence est peu importante. Par exemple, le nombre de commandes est comparable (54 pour le Dyna-Q parallèle contre 49, soit un écart de 10%). En fait, même pour un opérateur entraîné, la commande est très difficile à cause de l'hystérésis entre




(a) Dyna-Q parallèle (b) Opérateur humain

FIG. 6.19 – *Comparaison des trajectoires de l'outil générées par le contrôleur Dyna-Q parallèle et un opérateur humain entraîné.*

l'aimant et l'outil. Nous estimons donc que les stratégies obtenues par apprentissage sont satisfaisantes.

Néanmoins, comme pour le Q-Learning parallèle, la fonction de fusion peut créer momentanément des maxima locaux si les obstacles forment une impasse. Nous allons donc étudier dans le chapitre suivant une méthode de fusion plus élaborée.

# Chapitre 7

# Dyna-Q parallèle et fusion de modèles

*Ce chapitre présente un principe de fusion de modèles qui permet d'éviter la fusion des espérances de gain des perceptions et les problèmes inhérents de maxima locaux. Cette méthode fusionne en amont les modèles de transition des perceptions pour permettre d'effectuer une étape de planification avant tout choix d'une commande. L'étape de planification établit les séquences de commandes futures qui maximisent les récompenses en fonction d'un modèle global de l'environnement obtenu par fusion de modèles élémentaires. Si ce modèle global est suffisamment enrichi par le biais des modèles élémentaires, l'étape de planification permettra d'établir des séquences de commandes qui ne bloquent pas le système dans des maxima locaux. Les essais expérimentaux montrent que ce principe fonctionne correctement à condition que la mise en correspondance soit robuste.*

## 7.1 Introduction

Il nous faut désormais résoudre le problème de fusion des espérances de gain des perceptions qui survient lorsque le contrôleur est confronté à des situations de type impasse. Dans ces configurations, la fonction de fusion par somme des espérances de gain peut créer des maxima locaux qui entraînent le contrôleur dans un cycle répétitif.

La fonction de fusion que nous proposons dans ce chapitre utilise le modèle de transition $\mathsf{m}$ généré par la fonction de modélisation du Dyna-Q pour établir un modèle global du système qui permettra de calculer une meilleure estimation de l'espérance de gain de chaque commande grâce à une étape de planification.

Ce chapitre se compose de quatre parties : la première expose le principe de la fusion de modèles sur un exemple de situation, la deuxième décrit son fonctionnement dans le cas général, les deux dernières présentent respectivement les résultats expérimentaux obtenus sur l'exemple du labyrinthe et les comportements obtenus avec des impasses.

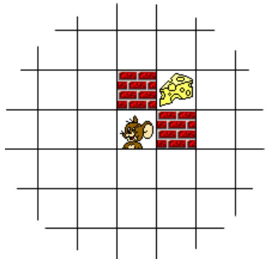


FIG. 7.1 – *Exemple de situation.*

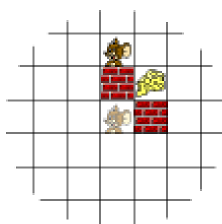
## 7.2 Idée fondatrice

Pour illustrer notre propos, nous nous appuyons sur l'exemple du labyrinthe dans la situation de la figure 7.1. Si le contrôleur utilise le principe de fusion par somme des espérances de gain décrit au chapitre 4, la souris est bloquée par les deux murs. En effet, si la souris s'écarte de cette position, elle aura tendance à y revenir car la fonction de fusion par somme crée un maximum local. La question est de savoir si le contrôleur peut trouver un moyen de calculer une stratégie de commande sans maximum local avec les connaissances qu'il a sur le système.

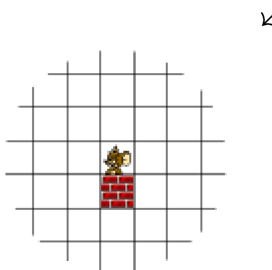
Pour un humain qui connaît le comportement de la souris vis-à-vis des murs et des fromages, la solution est évidente : pour atteindre le fromage, il faut contourner les murs. En fait, nous analysons rapidement la situation : la souris ne peut pas traverser un mur, par contre elle peut se déplacer sur les cases vides. Nous en déduisons les déplacements possibles de la souris dans les différentes cases du labyrinthe pour trouver le chemin le plus court parmi les différentes possibilités.

Notre idée de fusion de modèles est fondée sur ce principe : à chaque fois que le contrôleur aura besoin d'effectuer une commande, il va réaliser auparavant une « expérience de pensée » en imaginant la souris en train d'évoluer dans l'environnement. Il va alors déterminer la meilleure séquence de commandes possible pour maximiser ses récompenses en fonction des informations qu'il possède. Il va ensuite choisir la première commande de cette séquence. La conséquence de cette commande va enrichir le modèle que possède le contrôleur sur l'évolution de ses perceptions et le processus décrit recommencera. Une étape de planification des commandes du contrôleur a donc lieu avant toute nouvelle commande. Cette étape de planification pourra évidemment être plus ou moins approfondie en fonction d'un certain nombre de contraintes comme le traitement en temps réel. Ce point sera discuté plus loin dans le chapitre.

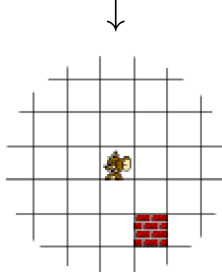
Le modèle $\mathsf{m}$ dont dispose le contrôleur est le modèle de transition des perceptions construit au fur et à mesure de ses expériences. Il décrit donc l'évolution des perceptions du contrôleur lorsqu'il effectue une commande. Dans ce modèle, la souris est immobile et les perceptions évoluent par rapport à elle. Pour pouvoir imaginer la souris en train d'évoluer dans l'environnement, le contrôleur a donc besoin d'« inverser » ce modèle en supposant son environnement devenu statique. Notons au passage que l'approche déve-



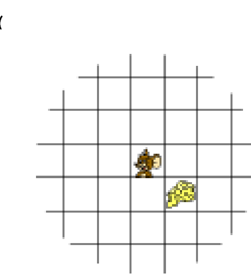
Position imaginaire (0,2) dans la situation de la figure 7.1



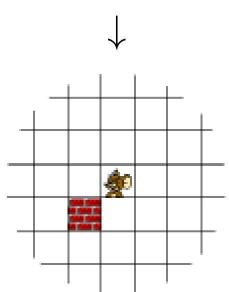
Le contrôleur recevrait la perception $x^1 = (0,-1,\text{mur})$



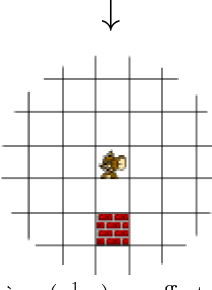
Le contrôleur recevrait la perception $x^2 = (1,-2,\text{mur})$



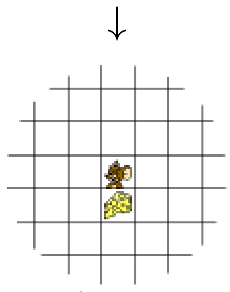
Le contrôleur recevrait la perception $x^3 = (1,-1,\text{fromage})$



D'après $\mathsf{m}(x^1,u)$, en effectuant la commande $u$, le contrôleur obtiendrait la perception $y^1 = (-1,-1,\text{mur})$ et la récompense 0



D'après $\mathsf{m}(x^1,u)$, en effectuant la commande $u$, le contrôleur obtiendrait la perception $y^2 = (0,-2,\text{mur})$ et la récompense 0



D'après $\mathsf{m}(x^1,u)$, en effectuant la commande $u$, le contrôleur obtiendrait la perception $y^3 = (0,-1,\text{fromage})$ et la récompense 0

La perception $x^1$ se déplacerait dans le labyrinthe selon le vecteur $\overrightarrow{\Pi(x^1)\Pi(y^1)} = (-1,-1)-(0,-1) = (-1,0)$ et obtiendrait la récompense 0

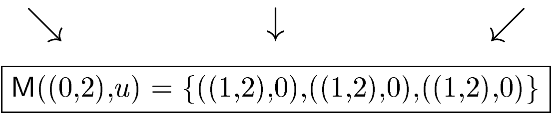
La perception $x^2$ se déplacerait dans le labyrinthe selon le vecteur $\overrightarrow{\Pi(x^2)\Pi(y^2)} = (0,-2)-(1,-2) = (-1,0)$ et obtiendrait la récompense 0

La perception $x^3$ se déplacerait dans le labyrinthe selon le vecteur $\overrightarrow{\Pi(x^3)\Pi(y^3)} = (0,-1)-(1,-1) = (-1,0)$ et obtiendrait la récompense 0

La souris se déplacerait dans le labyrinthe selon le vecteur $\overrightarrow{\Pi(y^1)\Pi(x^1)} = (1,0)$ et obtiendrait la récompense 0

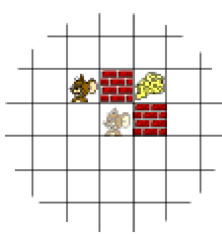
La souris se déplacerait dans le labyrinthe selon le vecteur $\overrightarrow{\Pi(y^2)\Pi(x^2)} = (1,0)$ et obtiendrait la récompense 0

La souris se déplacerait dans le labyrinthe selon le vecteur $\overrightarrow{\Pi(y^3)\Pi(x^3)} = (1,0)$ et obtiendrait la récompense 0

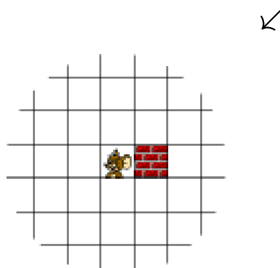


$\mathsf{M}((0,2),u) = \{((1,2),0),((1,2),0),((1,2),0)\}$

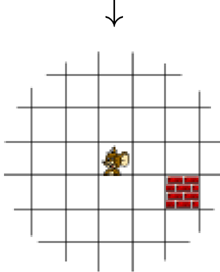
FIG. 7.2 – *Illustration du fonctionnement de la fonction de fusion de modèles pour la position (0,2) de la situation 7.1.*



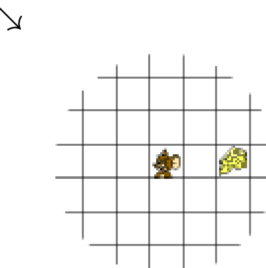
Position imaginaire $(-1,1)$ dans la situation de la figure 7.1



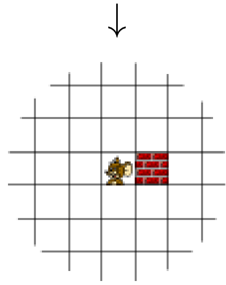
Le contrôleur recevrait la perception $x^1 = (1,0,\text{mur})$



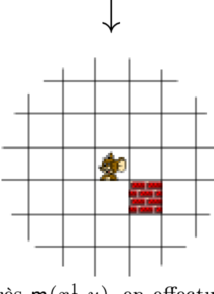
Le contrôleur recevrait la perception $x^2 = (2,-1,\text{mur})$



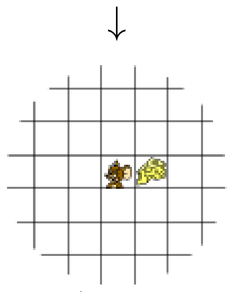
Le contrôleur recevrait la perception $x^3 = (2,0,\text{fromage})$



D'après $\mathsf{m}(x^1,u)$, en effectuant la commande $u$, le contrôleur obtiendrait la perception $y^1 = (1,0,\text{mur})$ et la récompense $-1$



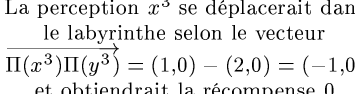
D'après $\mathsf{m}(x^1,u)$, en effectuant la commande $u$, le contrôleur obtiendrait la perception $y^2 = (1,-1,\text{mur})$ et la récompense 0



D'après $\mathsf{m}(x^1,u)$, en effectuant la commande $u$, le contrôleur obtiendrait la perception $y^3 = (1,0,\text{fromage})$ et la récompense 0

La perception $x^1$ se déplacerait dans le labyrinthe selon le vecteur $\overrightarrow{\Pi(x^1)\Pi(y^1)} = (1,0) - (1,0) = (0,0)$ et obtiendrait la récompense $-1$

La perception $x^2$ se déplacerait dans le labyrinthe selon le vecteur $\overrightarrow{\Pi(x^2)\Pi(y^2)} = (1,-1) - (2,-1) = (-1,0)$ et obtiendrait la récompense 0

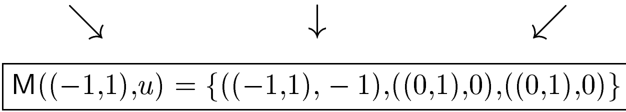


La perception $x^3$ se déplacerait dans le labyrinthe selon le vecteur $\overrightarrow{\Pi(x^3)\Pi(y^3)} = (1,0) - (2,0) = (-1,0)$ et obtiendrait la récompense 0

La souris se déplacerait dans le labyrinthe selon le vecteur $\overrightarrow{\Pi(y^1)\Pi(x^1)} = (0,0)$ et obtiendrait la récompense $-1$

La souris se déplacerait dans le labyrinthe selon le vecteur $\overrightarrow{\Pi(y^2)\Pi(x^2)} = (1,0)$ et obtiendrait la récompense 0

La souris se déplacerait dans le labyrinthe selon le vecteur $\overrightarrow{\Pi(y^3)\Pi(x^3)} = (1,0)$ et obtiendrait la récompense 0



$$\mathsf{M}((-1,1),u) = \{((-1,1),-1),((0,1),0),((0,1),0)\}$$

FIG. 7.3 – *Illustration du fonctionnement de la fonction de fusion de modèles pour la position (-1,1) de la situation 7.1.*

loppée dans ce chapitre ne peut s'appliquer à un environnement dynamique[1].

Dans ce qui suit, nous allons étudier une façon de reconstruire, à partir du modèle $\mathsf{m}$, un modèle global d'évolution de la souris dans son environnement perceptuel.

Imaginons que la souris soit dans la position (0,2) dans la situation de la figure 7.1. Le contrôleur recevrait alors trois perceptions : $(0,-1,\text{mur})$, $(1,-2,\text{mur})$ et $(1,-1,\text{fromage})$ (*cf.* figure 7.2). On cherche à établir ce que serait son déplacement si le contrôleur effectuait la commande $u =$ « à droite ». D'après le modèle de transition $\mathsf{m}$, le contrôleur peut prévoir les nouvelles perceptions[2] qu'il observerait après avoir effectué la commande $u$, soit : $(-1,-1,\text{mur})$, $(0,-2,\text{mur})$ et $(0,-1,\text{fromage})$. Les récompenses qu'il obtiendrait seraient $\{0{,}0{,}0\}$. Le contrôleur peut alors calculer le vecteur de déplacement que suivrait chaque perception dans le labyrinthe (ce vecteur s'écrit $\overrightarrow{\Pi(x)\Pi(x')}$ pour la perception $x$, $\Pi(x)$ étant une fonction qui renvoie les coordonnées de la perception $x$). Enfin, le contrôleur peut en déduire le vecteur de déplacement de la souris par rapport à chacune des perceptions (ce vecteur est l'opposé du vecteur de déplacement de la perception soit $\overrightarrow{\Pi(x')\Pi(x)}$).

Dans cet exemple, la souris se déplacerait d'une case vers la droite en obtenant une récompense nulle par rapport à chaque perception. Le contrôleur a donc construit le modèle de déplacement de la souris pour la position (0,2) et la commande « à droite » (voir en bas de la figure 7.2).

Maintenant, imaginons que la souris soit dans la position $(-1{,}1)$ toujours dans la situation de la figure 7.1. Dans cette position, le contrôleur recevrait trois perceptions : $(1{,}0,\text{mur})$, $(2,-1,\text{mur})$ et $(2{,}0,\text{fromage})$ (*cf.* figure 7.3). D'après le modèle de transition $\mathsf{m}$, si le contrôleur effectuait la commande $u =$ « à droite », la première perception ne bougerait pas et donnerait une récompense égale à $-1$. Comme l'évolution de chaque perception est traitée indépendamment, les deux autres perceptions se déplaceraient vers la gauche en donnant chacune une récompense nulle. Après inversion du modèle perceptuel et concaténation des couples obtenus, le modèle de déplacement de la souris est alors 33% de chance de rester sur place et d'obtenir une récompense égale à $-1$ et 66% de se déplacer vers la droite et d'obtenir une récompense nulle (voir en bas de la figure 7.3).

Grâce à cette méthode que nous appelons *fusion de modèles*, le contrôleur peut donc construire à chaque instant un modèle global de déplacement $\mathsf{M}$ de la souris dans le labyrinthe tel qu'il est perçu à cet instant à l'aide des informations qu'il a collectées depuis le début de l'apprentissage. Le contrôleur a alors la possibilité de planifier par programmation dynamique une stratégie de commande dans $\mathsf{M}$ qui pourra être optimale par rapport aux connaissances acquises.

1. Néanmoins, si le contrôleur possède un modèle temporel de l'évolution dynamique des perceptions, une extension de l'approche proposée est vraisemblablement possible. L'étape de planification devra alors prendre en compte l'évolution temporelle future de l'environnement.

2. On suppose que la dimension $h$ de l'historique est fixée à 1.

## 7.3 Fonctionnement dans le cadre général

### 7.3.1 Définitions

Dans le cadre général, le nouveau modèle $\mathsf{M}$, que l'on qualifiera de *global*, est défini sur un ensemble bien particulier distinct de l'ensemble des perceptions $\mathcal{X}$ et fortement lié au système. Par exemple, dans le labyrinthe, les déplacements de la souris sont plans et discrets. Le modèle global sera donc défini sur un ensemble discret de points du plan correspondant aux cases du labyrinthe.

Cet ensemble est appelé *la carte* et noté $\mathcal{C}$. Pour des applications de navigation comme la nôtre, la carte est un ensemble de points du plan correspondant à la discrétisation du système. Pour d'autres systèmes, la carte n'est pas forcément plane.

L'espace vectoriel (plan ou hyperplan) des points de la carte est noté $\Pi$. La figure 7.4 donne une représentation graphique de la carte par rapport à l'ensemble des perceptions $\mathcal{X}$.

Pour faire le lien entre l'ensemble des perceptions $\mathcal{X}$ et la carte $\mathcal{C}$, on définit une fonction $\Pi(x)$ qui associe à toute perception $x$ de $\mathcal{X}$ son projeté sur le plan (ou l'hyperplan) $\Pi$. Le plan (ou l'hyperplan) $\Pi$ est orienté dans l'espace de façon cohérente avec les directions des vecteurs d'évolution des perceptions. Pour le labyrinthe, le plan $\Pi$ correspond à celui du labyrinthe.

Notons que la carte n'est pas forcément égale à $\Pi(\mathcal{X})$. L'étape de planification pourra donc se faire sur une carte volontairement restreinte afin d'obtenir un temps de calcul compatible avec une exécution en temps réel. Néanmoins, si le contrôleur doit contourner de grands obstacles, la carte devra être suffisamment étendue si on veut que l'étape de planification trouve une succession de commandes conduisant au contournement [3].

On définit enfin le point $O$ de la carte qui correspond au projeté de l'origine du repère défini sur $\mathcal{X}$. Dans le cas du labyrinthe, le point $O$ correspond à la position de la souris.

### 7.3.2 Algorithmes

Dans un souci de clarté des algorithmes, nous avons divisé la fonction de fusion en deux parties (*cf.* figure 7.5).

La première étape consiste à créer le modèle global $\mathsf{M}$ d'évolution dans la carte. Cette fonction est appelée *fusion de modèles*. Elle produit en sortie la fonction $\mathsf{M}$ qui associe à chaque couple *position imaginaire—commande* $(p,u)$ de la carte $\mathcal{C}$, un multi-ensemble de couples *position—récompense* $(p',r)$. L'ensemble de ces couples $\{(p',r)\} = \mathsf{M}(p,u)$ décrit les positions et récompenses que l'on obtiendrait en imaginant d'effectuer la commande $u$ dans la position $p$. Notons que $\mathsf{M}$ est un modèle non déterministe puisqu'à partir d'un couple $(p,u)$, on obtient un multi-ensemble de couples $\{(p',r)\}$ qui définit des probabilités de transition en fonction de la multiplicité de chaque couple.

La seconde étape est l'étape de planification proprement dite. Puisque le modèle $\mathsf{M}$ est parfaitement connu, l'étape de planification correspond à une étape de programmation

3. Se pose donc ici le problème de la détermination de la topologie des frontières de la carte $\mathcal{C}$, cette topologie pouvant être de plus dynamique pour s'adapter à la « forme » du maximum local. Ce problème complexe est l'objet de perspectives de recherches (*cf.* chapitre 8).

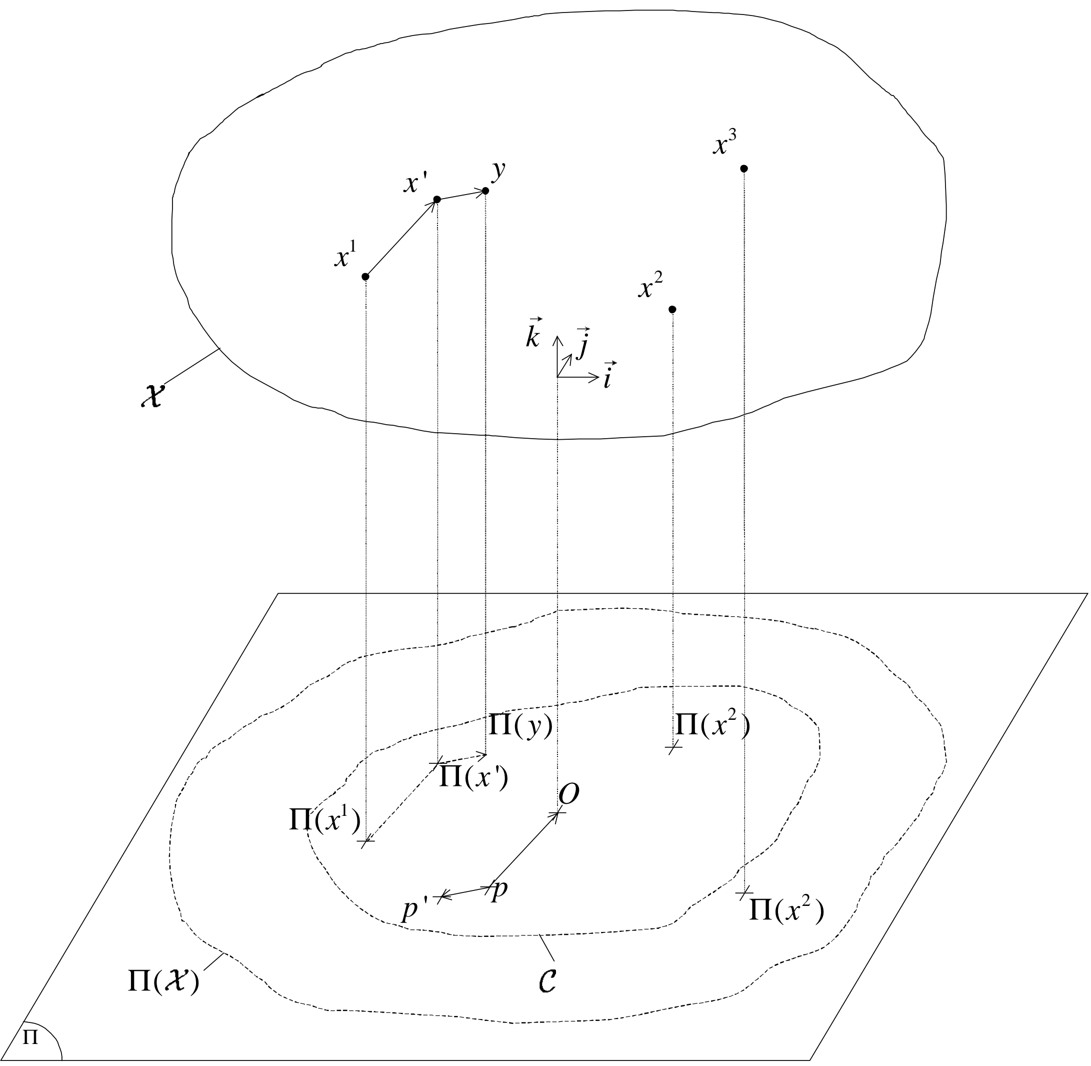



FIG. 7.4 – *Représentation des projections et des translations de la fonction de fusion de modèles.*

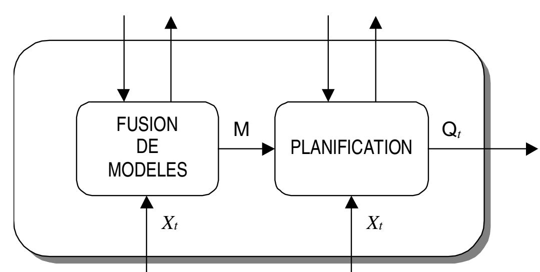



FIG. 7.5 – *Nouvelles fonctionnalités de la fonction de fusion présentée au chapitre 4 par la figure 4.5.*

dynamique classique sur le modèle $\mathsf{M}$. Cette étape établit donc pour *chaque position imaginaire* $p$ de la carte $\mathcal{C}$, l'espérance de gain optimale $\mathsf{Q_c}(p,u)$ de chaque commande $u$. Comme l'objectif du contrôleur est de sélectionner une commande $u$ adaptée à la situation courante du système, la fonction de planification doit fournir en sortie les espérances de gain $Q_t(u)$ pertinentes pour le choix de la commande $u$. Cette sortie correspond à l'ensemble des espérances $\mathsf{Q_c}(O,u)$ puisque $O$ correspond dans la carte à la position courante du système ($O$ est le projeté de l'origine du repère défini sur l'ensemble $\mathcal{X}$ des perceptions).

A l'aide de l'espérance de gain $\mathsf{Q}_t(u)$ de chaque commande $u$ possible, le contrôleur va sélectionner une commande en utilisant l'algorithme habituel de l'$\epsilon$-gourmand. Cette commande va induire une nouvelle transition du système qui va permettre à l'instant suivant d'affiner le modèle $\mathsf{M}$. Par conséquent, plutôt que de suivre par la suite les commandes optimales préalablement planifiées, il est préférable de relancer une étape de planification prenant en compte les nouvelles informations obtenues. Autrement dit, une étape de planification va avoir lieu à chaque période d'échantillonnage avant le choix de la commande.

### Fonction de fusion de modèles

L'algorithme de fusion de modèles est fondé sur le principe énoncé en début de chapitre. Pour chaque point $p$ de la carte $\mathcal{C}$ et pour chaque commande $u$ de $\mathcal{U}$, il estime les évolutions dans la carte et les récompenses obtenues par rapport à chaque perception $x$ de $X_t$. L'algorithme complet est donné par la figure 7.6. La représentation graphique de la figure 7.4 permet d'en suivre le déroulement.

L'algorithme fonctionne selon le principe suivant répété pour chaque perception $x$ de $X_t$.

Soit $p$ un point de la carte et $u$ une commande. Tout d'abord, l'algorithme cherche les perceptions que recevrait le contrôleur dans la position $p$. Il faut pour cela se replacer dans le référentiel lié à $p$. L'algorithme effectue ainsi la translation de la perception $x$ selon le vecteur $\overrightarrow{pO}$.

Si la perception $x + \overrightarrow{pO}$ appartient à $\mathcal{X}$, l'algorithme recherche dans la mémoire s'il connaît une transition de la perception $x' = x + \overrightarrow{pO}$ pour la commande $u$ (i.e. $\mathsf{m}(x',u) \neq \emptyset$). Dans ce cas, pour chaque couple *perception—récompense* $(y,r)$ de $\mathsf{m}(x',u)$, c'est-à-dire pour tout l'historique, il calcule le vecteur de déplacement de la perception dans la carte, soit $\overrightarrow{\Pi(x')\Pi(y)}$. Il en déduit que le point $p$ se déplacerait selon le vecteur opposé. Si le point obtenu $p' = p - \overrightarrow{\Pi(x')\Pi(y)}$ reste dans les limites de la carte, l'algorithme ajoute par concaténation ce nouveau point et la récompense $r$ au modèle global $\mathsf{M}(p,u)$. Notons que, comme dans l'exemple de la figure 7.4, il se peut qu'un même couple $(p',r)$ soit ajouté plusieurs fois au multi-ensemble. La multiplicité des éléments de $\mathsf{M}(p,u)$ définit alors très simplement un modèle probabiliste *via* une approche fréquentiste, en comptant les occurrences.

Lorsque l'algorithme a réalisé ce processus pour chaque point $p$ de $\mathcal{C}$ et commande $u$ de $\mathcal{U}$, on obtient le modèle global courant du système $\mathsf{M}$.

**Entrée :** $X_t$, m
**Sortie :** M
**Début**
  **Pour tout** point $p$ de $\mathcal{C}$ **faire**
    Initialiser M$(p,u)$ à $\emptyset$ pour tout $u$ de $\mathcal{U}$
    **Pour tout** $x$ de $X_t$ **faire**
      **Si** $x + \overrightarrow{pO} \in \mathcal{X}$ **alors**
        $x' \leftarrow x + \overrightarrow{pO}$
        **Pour tout** $u$ de $\mathcal{U}$ **faire**
          **Si** m$(x',u) \neq \emptyset$ **alors**
            **Pour tout** $(y,r)$ de m$(x',u)$ *(i.e. pour tout l'historique)* **faire**
              **Si** $p - \overrightarrow{\Pi(x')\Pi(y)} \in \mathcal{C}$ **alors**
                M$(p,u) \leftarrow$ M$(p,u) + \{(p - \overrightarrow{\Pi(x')\Pi(y)},r)\}$ *(concaténation)*
              **Fin si**
            **Fin pour**
          **Fin si**
        **Fin pour**
      **Fin si**
    **Fin pour**
  **Fin pour**
**Fin**

FIG. 7.6 – *Algorithme de la fonction de fusion de modèles.*

**Entrée :** $X_t$, M
**Entrée-sortie :** Q$_c$
**Sortie :** Q$_t$
**Début**
*(Initialisation des valeurs d'utilité des positions limites)*
**Pour tout** $(p,u)$ de $\mathcal{C} \times \mathcal{U}$ **faire**
**Si** $\mathsf{M}(p,u) = \emptyset$ **alors**
$$\mathsf{Q_c}(p,u) \leftarrow \sum_{\forall x \in X_t \mid x+\overrightarrow{pO} \in \mathcal{X}} \mathsf{q}(x+\overrightarrow{pO},u)$$
**Fin si**
**Fin pour**
*(Optimisation de la fonction d'utilité Q$_c$)*
**Répéter** $N_c$ fois
**Pour tout** $(p,u)$ de $\mathcal{C} \times \mathcal{U}$ **faire**
**Si** $\mathsf{M}(p,u) \neq \emptyset$ **alors**
$$\mathsf{Q_c}(p,u) \leftarrow \sum_{\forall (p',r) \in \mathsf{M}(p,u)} \frac{1}{|\mathsf{M}(p,u)|} \left[ r + \gamma \max_{v \in \mathcal{U}} \mathsf{Q_c}(p',v) \right]$$
**Fin si**
**Fin pour**
**Fin Répéter**
*(Évaluation des valeurs d'utilité de chaque commande de $\mathcal{U}$)*
**Pour tout** $u$ de $\mathcal{U}$ **faire**
$\mathsf{Q}_t(u) \leftarrow \mathsf{Q_c}(O,u)$
**Fin pour**

**Fin**

FIG. 7.7 – *Algorithme de la fonction de planification.*

### Fonction de planification

La fonction de planification correspond à une étape de programmation dynamique sur M fondée sur l'algorithme d'itérations sur les valeurs (*cf.* §2.3.5, page 17). En pratique, cette étape va consister à optimiser une fonction d'utilité liée à la carte et notée Q$_c$ à l'aide du modèle M. Cette fonction d'utilité globale $\mathsf{Q_c}(p,u)$ associe à chaque position imaginaire $p$ de la carte $\mathcal{C}$ l'espérance de gain imaginaire d'effectuer la commande $u$.

Dans l'algorithme d'itérations sur les valeurs, l'équation de mise à jour (équation de Bellman) s'écrit :

$$\mathsf{Q_c}{}_t(p,u) = \sum_{\forall p' \in \mathcal{C}} \mathsf{p}(p' \mid p, u) \left[ \mathsf{r}(p,u,p') + \gamma \max_{v \in \mathcal{U}} \mathsf{Q_c}(p',v) \right] \tag{7.1}$$

Comme le modèle $\mathsf{M}$ est défini par un multi-ensemble, les probabilités $\mathsf{p}$ de transition sont calculées en utilisant la multiplicité d'un couple, soit :

$$\mathsf{p}(p'|p,u) = \sum_{\forall (g,r)\in \mathsf{M}(p,u) | g=p'} \frac{1}{|\mathsf{M}(p,u)|} \tag{7.2}$$

De même, la fonction de récompense $\mathsf{r}(p,u,p')$ correspond à la moyenne des récompenses que l'on obtiendrait en atteignant la position $p'$, soit :

$$\mathsf{r}(p,u,p') = \frac{\sum_{\forall (g,r)\in \mathsf{M}(p,u) | g=p'} r}{\sum_{\forall (g,r)\in \mathsf{M}(p,u) | g=p'} 1} \tag{7.3}$$

L'équation[4] de mise à jour s'écrit alors :

$$\mathsf{Q_c}_t(p,u) = \sum_{\forall (p',r)\in \mathsf{M}(p,u)} \frac{1}{|\mathsf{M}(p,u)|} \left[ r + \gamma \max_{v\in\mathcal{U}} \mathsf{Q_c}_{t-1}(p',v) \right] \tag{7.4}$$

Notons que $r$ correspond bien à la récompense associée à la position $p'$ et non à la fonction de récompense $\mathsf{r}(p,u,p')$.

Par exemple, si pour une position $p$ et une commande $u$, le modèle $\mathsf{M}$ donne l'ensemble $\{(p_1,-1),(p_2,2),(p_2,1)\}$, l'équation de mise à jour s'écrit :

$$\mathsf{Q_c}_t(p,u) = \frac{1}{3}(-1 + \max_{v\in\mathcal{U}} \mathsf{Q_c}_{t-1}(p_1,v)) + \frac{1}{3}(2 + \max_{v\in\mathcal{U}} \mathsf{Q_c}_{t-1}(p_2,v)) \tag{7.5}$$

$$+ \frac{1}{3}(1 + \max_{v\in\mathcal{U}} \mathsf{Q_c}_{t-1}(p_2,v)) \tag{7.6}$$

$$= \frac{1}{3}(-1 + \max_{v\in\mathcal{U}} \mathsf{Q_c}_{t-1}(p_1,v)) + \frac{2}{3}(1{,}5 + \max_{v\in\mathcal{U}} \mathsf{Q_c}_{t-1}(p_2,v)) \tag{7.7}$$

Cette équation traduit bien le modèle décrit par le multi-ensemble : à partir de la position $p$ et en effectuant la commande $u$, la probabilité d'obtenir la position $p_1$ et la récompense $-1$ est de 0.33, la probabilité d'obtenir la position $p_2$ et la récompense 1,5 (qui correspond à la moyenne des deux récompenses associées à $p_2$) est de 0.66.

Avec cette définition de la fonction de récompense $\mathsf{r}(p,u,p')$ par l'équation (7.3), le gain que l'algorithme cherche à maximiser est toujours défini par :

$$\mathsf{G_c}(t) = \sum_{k=0}^{\infty} \gamma^k \mathsf{r}_{t+k}(p_{t+k},u_{t+k},p_{t+k+1}) \tag{7.8}$$

4. Cette équation a déjà été utilisée dans la fonction d'optimisation du Dyna-Q parallèle avec le modèle $\mathsf{m}$ (*cf.* équation (6.3)).

et donc l'espérance de gain imaginaire $\mathsf{Q_c}$ s'écrit :

$$\mathsf{Q_c}^{\pi_c}(p,u) = E_{\pi_c}\{\mathsf{G_c}(t) \,|\, p_t = p, u_t = u\} \tag{7.9}$$

L'algorithme de la fonction de planification, donné par la figure 7.7, se compose de trois parties.

Tout d'abord, le contrôleur calcule des valeurs d'utilité des positions limites qui correspondent aux positions $p$ dont on ne connaît pas de transition sortante ($\mathsf{M}(p,u) = \emptyset$). Cette étape est nécessaire dans le cas où aucune perception n'a son projeté dans la carte. Elle permet alors au contrôleur de prendre une décision en fonction des perceptions éloignées en utilisant l'algorithme de fusion par somme. Ce calcul n'intervient que si la carte est de petite dimension par rapport au projeté de $\mathcal{X}$.

L'algorithme optimise ensuite $N_c$ fois l'ensemble des couples *position—commande* à l'aide de l'équation (7.4). Pour obtenir une bonne stratégie, il n'est pas nécessaire de réaliser un grand nombre d'optimisations. En effet, l'optimisation de la fonction d'utilité globale $\mathsf{Q_c}$ à l'instant $t$ va démarrer à partir de la fonction $\mathsf{Q_c}$ préalablement optimisée à l'instant $t-1$. De plus, et c'est là un avantage majeur de l'approche, le modèle global $\mathsf{M}$ à l'instant $t$ est très proche de celui calculé à l'instant $t-1$ car l'état du système évolue généralement peu entre deux instants $t-1$ et $t$. La fonction d'utilité $\mathsf{Q_c}$ est donc déjà proche de l'optimale et quelques cycles suffisent à sa convergence. Ce nombre restreint $N_c$ de cycles nécessaires à la convergence permet en pratique d'envisager la planification en temps réel à chaque période d'échantillonnage.

Enfin, une fois la fonction d'utilité $\mathsf{Q_c}$ optimisée, il reste à évaluer l'espérance de gain $\mathsf{Q}_t(u)$ de chaque commande $u$ possible dans l'état actuel du système. Cet état courant correspond toujours à l'origine du repère dans lequel sont représentées les perceptions. Comme l'origine de ce repère est associée au point $O$ de la carte, l'espérance de gain $\mathsf{Q}_t(u)$ de chaque commande $u$ correspond à la valeur de la fonction d'utilité globale $\mathsf{Q_c}$ de la position définie par $O$, soit :

$$\mathsf{Q}_t(u) = \mathsf{Q}_c(O,u) \tag{7.10}$$

## 7.4 Application à l'exemple du labyrinthe

### 7.4.1 Influence des paramètres

#### Influence de la taille de la carte

La taille de la carte n'a d'importance que par rapport à la dimension des impasses. En effet, le rôle de la carte est de pouvoir trouver un chemin qui contourne un obstacle. Si l'obstacle est plus grand que la carte, la découverte d'un chemin de contournement n'est pas possible. La taille de la carte définit donc la taille maximum des obstacles que le contrôleur saura contourner.

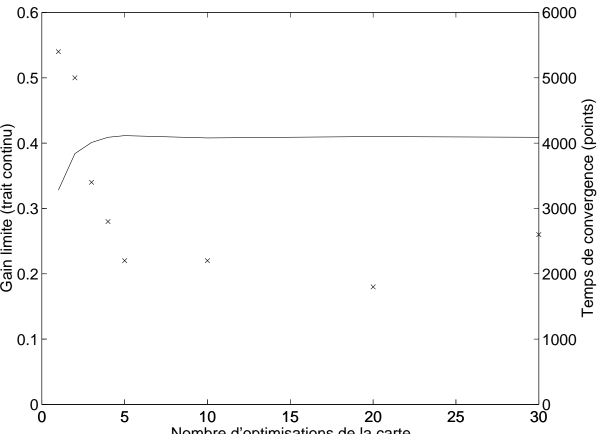



FIG. 7.8 – *Influence du nombre $N_c$ d'optimisations de la carte sur le gain limite et sur le temps de convergence (moyenne sur 10 simulations avec 5 fromages, $\mathcal{C} = 21 \times 21$, $h = 5$, $N = 2000$, $\gamma = 0{,}5$ et $\epsilon = 0{,}1$).*

Dans la suite, nous avons choisi une taille de carte de 21 par 21 cases, soit un quart de la surface du projeté de $\mathcal{X}$. Le contrôleur peut ainsi « imaginer » des déplacements de 10 cases dans chaque direction par rapport à la souris.

#### Influence du nombre $N_c$ d'optimisations de la carte

Les courbes du gain limite et du temps d'apprentissage de la figure 7.8 montrent que le nombre $N_c$ d'optimisations de la carte n'a pas besoin d'être grand. Cinq optimisations de la carte à chaque fusion suffisent pour obtenir une bonne stratégie de commande en permanence. La raison de cette rapidité de l'optimisation est que la carte change peu à chaque période d'échantillonnage. Les perceptions ne se sont pas beaucoup déplacées par rapport à la souris (d'une case par exemple) et il suffit de quelques mises à jour pour optimiser la fonction d'utilité.

#### Influence du coefficient de pondération $\gamma$

La courbe de la figure 7.9 montre que la valeur de $\gamma$ n'a pas d'influence sur la limite de la récompense moyenne et sur le temps d'apprentissage. Les valeurs proches de 1 ne posent plus de problème car cette fois, le calcul de l'espérance de gain des commandes utilise un algorithme de programmation dynamique classique.

### 7.4.2 Comparaison des fonctions de fusion

La figure 7.10 montre une comparaison des courbes d'apprentissage obtenues avec et sans fusion de modèles. Les résultats sont comparables. Avec 5 fromages, la fonction de fusion par somme est légèrement supérieure, mais lorsqu'on ajoute des murs dispersés, la fusion de modèles permet une convergence plus rapide et des gains équivalents.

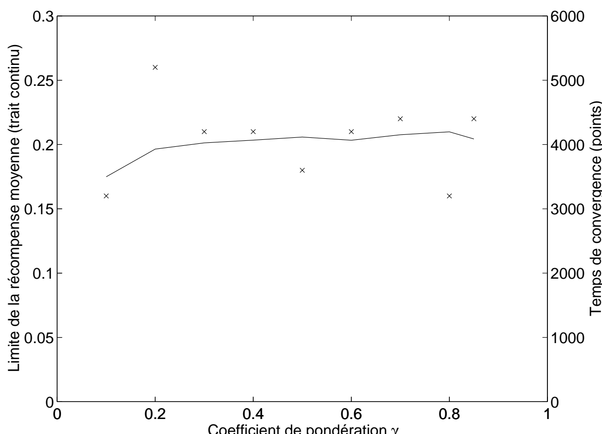
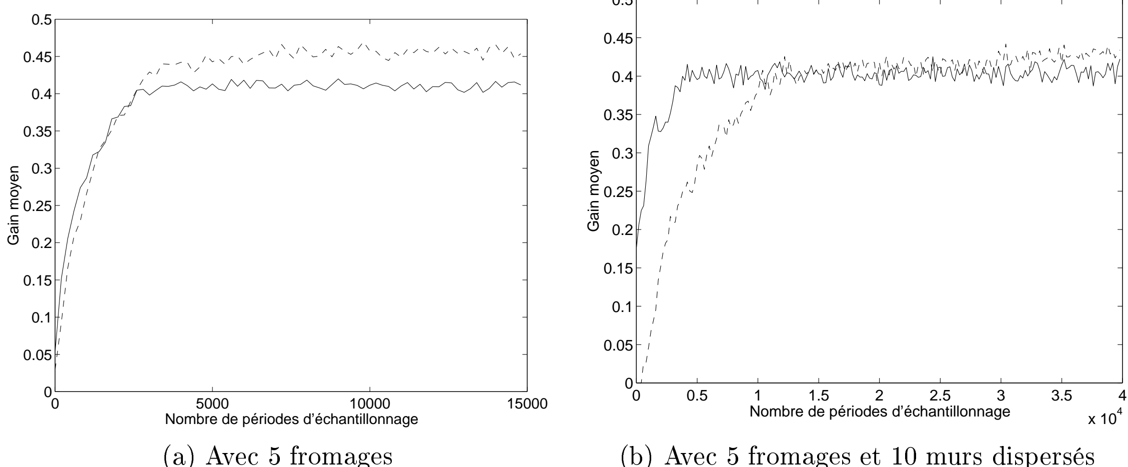



FIG. 7.9 – *Influence du coefficient de pondération $\gamma$ sur la limite de la récompense moyenne et sur le temps de convergence (moyenne sur 10 simulations avec 5 fromages, $\mathcal{C} = 21 \times 21$, $N_c = 10$, $h = 5$, $N = 2000$ et $\epsilon = 0{,}1$).*




(a) Avec 5 fromages

(b) Avec 5 fromages et 10 murs dispersés

FIG. 7.10 – *Comparaison de l'évolution des gains moyens obtenus avec le Dyna-Q parallèle avec fusion de modèles (traits continus) et avec le Dyna-Q parallèle sans fusion de modèles (traits discontinus) (moyenne sur 40 simulations avec $\mathcal{C} = 21 \times 21$, $N_c = 10$, $N = 2000$, $h = 5$, $\gamma = 0{,}5$ et $\epsilon = 0{,}1$).*

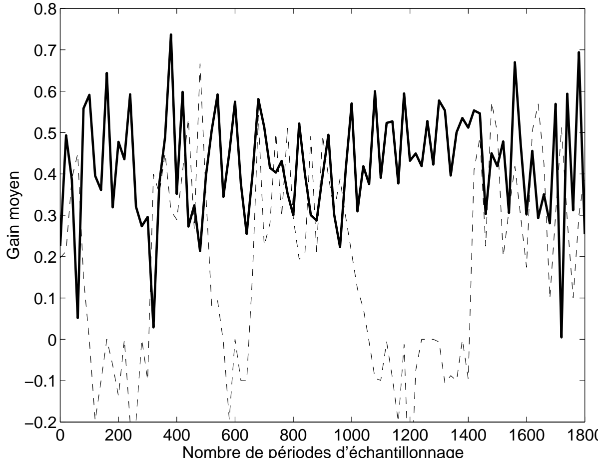



FIG. 7.11 – *Comparaison des gains obtenus dans un labyrinthe avec 5 fromages et une impasse de quatre murs avec fusion de modèles (traits continus) et sans fusion de modèles (traits discontinus) (avec* $\mathcal{C} = 21 \times 21$*,* $N_c = 10$*,* $N = 2000$*,* $h = 5$*,* $\gamma = 0{,}5$ *et* $\epsilon = 0{,}1$*).*

La supériorité de la fusion de modèles se révèle surtout quand le labyrinthe comporte des impasses.

## 7.5 Tests sur des labyrinthes avec impasses

### 7.5.1 Impasse à quatre murs

Dans le cas du labyrinthe avec une impasse à quatre murs (*cf.* §5.4.4, page 89), la fusion de modèles permet au contrôleur de sortir la souris de l'impasse. Si l'on compare l'évolution des gains moyens (*cf.* figure 7.11), on constate que le gain est plus stable avec la fusion de modèles, car la souris n'est jamais bloquée dans l'impasse. Quelques valeurs proches de zéro témoignent d'un court passage de la souris dans l'impasse.

### 7.5.2 Conséquences des erreurs de la mise en correspondance temporelle

Dans certains cas comme celui de la figure 7.12, la souris peut encore se bloquer. Cette fois, ce n'est plus la fonction de fusion qui est en cause mais la fonction de mise en correspondance. En effet, si la fonction de mise en correspondance commet une erreur en appariant temporellement des perceptions éloignées spatialement, cela se traduira par une erreur de modélisation dans $\mathsf{m}$, et donc aussi dans le modèle inverse $\mathsf{M}$. La fonction de planification « croira » alors qu'il existe un passage possible permettant de déplacer plus vite le système[5] et donc d'atteindre plus rapidement un fromage. Elle pourra donc amener le système à tenter de réaliser une commande impossible telle que par exemple

5. Comme un trou noir reliant deux positions de l'espace...

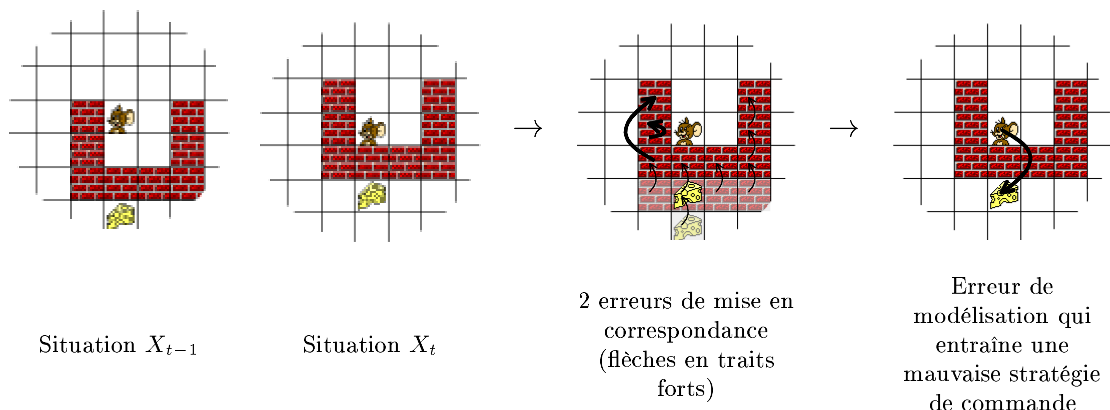
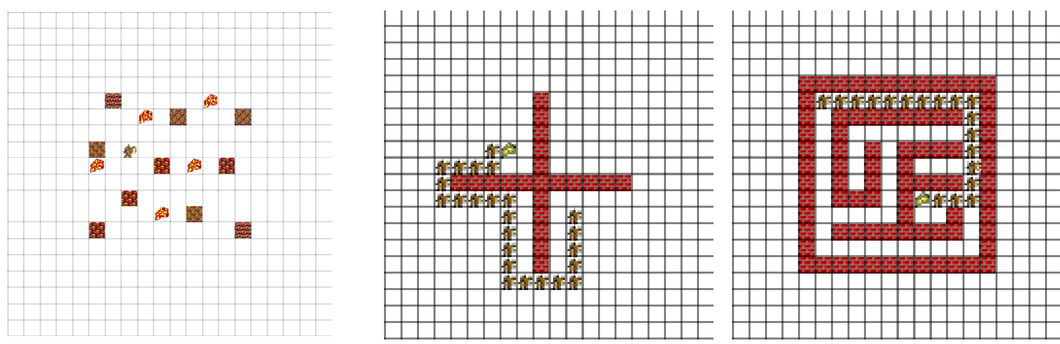



FIG. 7.12 – *Illustration de l'influence d'une erreur de mise en correspondance sur la fusion de modèles.*

passer instantanément de part et d'autre d'un mur. Cette commande se traduira par une récompense négative, le système restant bloqué dans la même configuration. Néanmoins, comme le modèle $\mathsf{M}$ est évolutif, la probabilité de cette transition impossible diminuera peu à peu si l'erreur d'appariement n'apparaît plus. La pertinence de l'étape de planification est donc directement liée à la robustesse de la fonction de mise en correspondance. On retrouve donc ici le problème classique de la spécification de l'étape de planification si le contrôleur possède des capteurs de perception imprécis et/ou incertains.

Pour vérifier que la mise en correspondance est bien à l'origine de ces nouveaux blocages, nous avons implanté une mise en correspondance *ad hoc* qui est toujours exacte dans le cas particulier du labyrinthe. Avec cette mise en correspondance, le contrôleur fonctionne parfaitement et la souris sort de n'importe quelle impasse dans la limite de la dimension de la carte comme l'illustre la figure 7.13 (voir aussi le logiciel de démonstration sur le CD-ROM décrit dans l'annexe A). Cette figure illustre bien toute la puissance du Dyna-Q parallèle avec fusion de modèles. Le contrôleur est préalablement entraîné sur un labyrinthe très simple comportant 5 fromages et 10 murs dispersés. Une fois l'apprentissage terminé, le contrôleur sait comment se comporte n'importe quel mur et n'importe quel fromage lorsqu'il effectue une commande donnée. Autrement dit, il a établi son modèle $\mathsf{m}$ de transition des perceptions. On présente alors au contrôleur de nouveaux environnements. En utilisant uniquement son modèle $\mathsf{m}$ et donc *sans faire le moindre réapprentissage*, le contrôleur est capable de reconstruire un modèle global $\mathsf{M}$ grâce à la fonction de fusion de modèles. Ce dernier devient alors capable de récolter tous les fromages sans hésitation[6] et ceci quel que soit la complexité de l'environnement. On constate de plus qu'il évite parfaitement toutes les impasses dépourvues de fromage. Le contrôleur est donc capable de généraliser à n'importe quel environnement complexe ce qu'il a appris sur un environnement donné très simple. A la limite, un seul mur et un

6. Aux explorations près qui peuvent « ralentir » le contrôleur. Si ce dernier est correctement entraîné et que l'environnement n'évolue pas, il est possible de diminuer le taux des explorations.



(a) Labyrinthe d'entraînement

(b) Labyrinthes de test avec des obstacles de taille maximale par rapport aux dimensions de la carte

FIG. 7.13 – *Exemples de trajectoires obtenues sans réapprentissage avec un contrôleur à fusion de modèles entraîné sur un labyrinthe avec 5 fromages et 10 murs dispersés.*

seul fromage suffisent pour l'entraînement du contrôleur, mais on perd alors l'accélération de l'apprentissage obtenue par le biais des renforcements parallèles induits par les perceptions multiples.

## 7.6 Conclusion

La méthode de fusion de modèles associée au Dyna-Q parallèle permet de reconstruire un modèle global d'évolution du système à partir des observations des transitions associées aux perceptions. Ce modèle global est alors utilisé dans une étape de planification qui passe par l'optimisation d'une fonction d'utilité globale. Cette optimisation utilise la programmation dynamique et demande évidemment plus de calculs que l'opérateur de fusion « somme » pour estimer l'espérance de gain de chaque commande possible à un instant donné. Néanmoins, peu d'itérations de programmation dynamique sont nécessaires à chaque période d'échantillonnage car le modèle global à l'instant $t$ est proche de celui de l'instant $t-1$. Les tests montrent que cette méthode permet effectivement une meilleure évaluation de l'espérance de gain de chaque commande. Contrairement à la fusion par sommation, elle ne crée pas de maximum local si le modèle global est suffisamment enrichi par le biais des modèles élémentaires. En revanche elle est beaucoup plus sensible aux erreurs de mise en correspondance que la fusion par sommation car le contrôleur devient très opportuniste et utilise immédiatement dans l'étape de planification toutes les informations qu'il acquiert. Si ces informations sont erronées, la planification ne sera donc pas pertinente. Une fonction de mise en correspondance robuste est donc indispensable avec ce type d'approche.

# Chapitre 8

# Conclusion et perspectives

## 8.1 Bilan des travaux

L'objectif applicatif de la thèse était de concevoir une commande par apprentissage du manipulateur WIMS et plus particulièrement de générer des trajectoires pour atteindre une cellule cible en évitant les autres cellules quelle que soit leur configuration spatiale.

Après avoir étudié les différents algorithmes d'apprentissage par renforcement et les architectures comportementales, nous nous sommes orientés vers une architecture fondée sur la parallélisation de l'algorithme du Q-Learning.

Cette nouvelle architecture de contrôle par apprentissage est adaptée à la commande de systèmes dont l'état est de grande dimension et peut se décomposer en une situation. Une situation est définie par le produit cartésien de plusieurs variables markoviennes appelées perceptions appartenant à un unique ensemble $\mathcal{X}$.

Le fonctionnement de l'architecture parallèle consiste à réaliser un apprentissage type Q-Learning pour chaque perception en utilisant toujours une même fonction d'utilité définie sur l'ensemble $\mathcal{X}$. Cette méthode permet de réduire la complexité du système et d'effectuer un plus grand nombre de mises à jour par période d'échantillonnage, ce qui accélère l'apprentissage. En revanche, pour choisir une commande, il est nécessaire de regrouper les espérances de gain données par la fonction d'utilité pour chaque perception. Une première approche possible consiste à utiliser une fonction de fusion des espérances de gain de chaque perception. Nous avons ainsi étudié l'opérateur de fusion « somme ».

Cette architecture donne de bons résultats sur l'exemple du labyrinthe, mais le temps d'apprentissage est trop long pour apprendre à commander un système réel. Le Q-Learning peut cependant être remplacé par un algorithme dont la convergence est plus rapide. Cet algorithme est fondé sur le Dyna-Q que nous avons adapté à la commande de systèmes non déterministes. Cette nouvelle architecture, baptisée Dyna-Q parallèle, permet non seulement de réaliser une amélioration importante de la vitesse de convergence, mais aussi de trouver de meilleures stratégies de contrôle. Les expérimentations montrent que l'apprentissage est alors possible directement sur le système en temps réel et sans utiliser de simulation.

| | Q-Learning parallèle avec fusion par somme | Dyna-Q parallèle avec fusion par somme | Dyna-Q parallèle avec fusion de modèles |
|---|---|---|---|
| Mémoire nécessaire (nombre de variables) | $\lvert\mathcal{X}\rvert \times \lvert\mathcal{U}\rvert$ | $(1+2h) \times \lvert\mathcal{X}\rvert \times \lvert\mathcal{U}\rvert$ | $(1+2h) \times \lvert\mathcal{X}\rvert \times \lvert\mathcal{U}\rvert + (1+2h\lvert X\rvert) \times \lvert\mathcal{C}\rvert \times \lvert\mathcal{U}\rvert$ |
| Nombre de mises à jour par période d'échantillonnage | $\lvert X\rvert$ | $\lvert X\rvert + N$ | $\lvert X\rvert + N + N_c\lvert\mathcal{C}\rvert \times \lvert\mathcal{U}\rvert$ |
| Vitesse de convergence | lente | rapide | très rapide |
| Sensibilité aux situations complexes (impasses) | très sensible | très sensible | robuste |
| Sensibilité aux erreurs de mise en correspondance temporelle | robuste | robuste | sensible |

TAB. 8.1 – *Comparatif des caractéristiques générales des trois architectures développées (avec $\mathcal{X}$ l'ensemble des perceptions, $\mathcal{U}$ l'ensemble des commandes, $X$ une situation (état global du système), $h$ la dimension de l'historique mémorisé par le modèle* m *du Dyna-Q, $\mathcal{C}$ la carte (ensemble des positions utilisées par la méthode de fusion de modèles), $N$ le nombre de mises à jour par période d'échantillonnage du modèle* m *et $N_c$ le nombre d'optimisations totales de la carte $\mathcal{C}$ par période d'échantillonnage).*

La fonction de fusion par somme des espérances de gain fonctionne bien si les obstacles sont relativement éloignés les uns des autres. Si ce n'est pas le cas, elle peut créer des maxima locaux qui entraînent le système dans un cycle répétitif. Pour pallier ce problème, nous avons proposé une autre fonction de fusion qui, cette fois, synthétise un modèle plus global du système à partir du modèle de transition des perceptions. A chaque période d'échantillonnage, la commande choisie est la première commande optimale proposée par l'étape de planification effectuée sur le modèle global. Cette nouvelle fonction de fusion permet de sortir de ces maxima locaux à condition que la fonction de mise en correspondance soit robuste. L'application en temps réel est possible car la planification nécessite peu de cycles de programmation dynamique, le modèle global évoluant peu entre deux périodes d'échantillonnage.

Le tableau 8.1 synthétise les caractéristiques générales des trois architectures développées.

## 8.2 Perspectives

Nos perspectives de recherches s'articulent autour de deux thèmes : d'une part les évolutions potentielles de l'architecture parallèle et d'autre part ses applications possibles, notamment l'étude de la possibilité d'effectuer des tâches de convoyage de cellules par poussée avec le manipulateur WIMS.

### 8.2.1 Évolutions de l'architecture parallèle

#### Mise en correspondance

Nous avons montré au chapitre 7 que pour obtenir une bonne stratégie de contrôle, il est nécessaire que la mise en correspondance soit de bonne qualité.

La mise en correspondance est un domaine de recherche à part entière. Nous n'avons pas abordé ce sujet au cours de la thèse, mais nous pensons que pour améliorer les performances de l'architecture parallèle, il serait intéressant d'étudier les différentes méthodes qui existent dans ce domaine, notamment la possibilité de réaliser cette mise en correspondance par apprentissage. La fonction apprendrait alors à mettre en correspondance les perceptions sans connaissance ni hypothèse initiale sur le système.

#### Modélisation pour les systèmes continus

Pour la commande de systèmes continus, la discrétisation linéaire n'est pas toujours la meilleure solution. Il est parfois plus adapté d'avoir une précision de discrétisation variable, plus fine aux endroits où l'on a besoin de précision et plus grossière là où l'on n'en a pas besoin.

Les travaux de Andrew Moore et Christopher Atkeson (1993) sur l'algorithme du parti-game poursuivis par David Chapman et Leslie Kaelbling (1991), Andrew McCallum (1995) et Rémi Munos (1997) apportent la possibilité d'augmenter ou de réduire le nombre d'états en fonction du besoin (*cf.* §2.6.4, page 42). Ces travaux sont tout à fait compatibles avec notre approche.

#### Fusion

La fonction de fusion présente de nombreuses possibilités d'évolution :

- l'optimisation de la carte et de la fonction d'utilité serait effectuée de manière plus efficace en utilisant des algorithmes fondés sur des heuristiques de mise à jour. Des algorithmes comme le priority sweeping de Andrew Moore et Christopher Atkeson (1993) permettent de réduire sensiblement le nombre des mises à jour (*cf.* §2.5.3, page 36). Dans ce domaine, l'idéal serait évidemment de disposer de processeurs dédiés à la programmation dynamique.
- on pourrait envisager l'utilisation d'une carte à taille variable : la taille diminue quand les cibles sont faciles à atteindre et grandit lorsque le système est bloqué, permettant ainsi de trouver une solution de sortie.

– la carte pourrait permettre d'imaginer le déplacement des perceptions qui sortent du « champ de vision » du contrôleur dans le cas classique où l'univers est plus grand que l'espace perçu. L'évolution des perceptions qui sortent du champ de vision pourrait être estimée dans une carte plus grande à l'aide du modèle de transition des perceptions. Cela nécessiterait une grande puissance de calcul mais permettrait par exemple de contourner de très gros obstacles : le contrôleur pourrait prévoir de retrouver la perception d'une cible de l'autre côté de l'obstacle car il imaginerait la trajectoire de la cible en dehors de son champ de vision. La perception de la cible pourrait alors sortir momentanément de son champ de vision.

### 8.2.2 Domaines d'application

Cette architecture est conçue pour apprendre à piloter des systèmes dont l'état est composé de plusieurs variables markoviennes. Plusieurs domaines rentrent dans ce cadre, notamment les problématiques de sélection de l'action, les problématiques de navigation et de vision.

#### Sélection de l'action

La problématique de la sélection de l'action est de trouver la meilleure commande à effectuer en fonction de nombreuses informations sur le système et de plusieurs objectifs souvent concurrents. Dans le cadre de l'apprentissage, le W-Learning proposait une première réponse à ce genre de problème que l'architecture parallèle généralise en quelque sorte. La sélection de l'action est donc tout à fait le type d'application envisageable pour l'architecture parallèle.

#### Navigation

La navigation est le domaine de prédilection de l'architecture parallèle puisqu'elle a été développée pour cette application. Au-delà de la commande du manipulateur WIMS, l'architecture parallèle pourrait être utilisée pour le contrôleur d'autres applications de navigation, comme par exemple :

– apprendre à commander un robot mobile dont les perceptions sont issues de capteurs hétérogènes renseignant sur l'environnement du robot (murs environnants, direction et éloignement d'un objectif ou d'un chargeur de batterie, *etc.*),
– apprendre à commander un robot jouant au football et percevant de nombreuses informations comme la position des joueurs de son équipe, des joueurs de l'équipe adverse, des buts et du ballon,
– en restant dans le domaine ludique, apprendre à commander un personnage non joueur [1] d'un jeu vidéo qui reçoit de nombreuses informations sur l'environnement et les autres personnages.

1. Un personnage non joueur (PNJ) est une créature artificielle du jeu vidéo qui est dirigée par l'ordinateur et qui est souvent en concurrence avec les personnages dirigés par les joueurs humains.

#### Vision

Dans le domaine de la vision, on peut considérer chaque pixel comme une perception, c'est-à-dire une information regroupant couleur et position. La situation du système est alors constituée par l'image vidéo entière.

A condition de disposer d'un calculateur suffisamment puissant, on pourrait utiliser directement l'architecture parallèle avec des perceptions ainsi définies et sans aucun traitement d'image. Les applications visées pourraient être par exemple :

- l'apprentissage de la recherche et du suivi de pixels de couleur dans une image (comme le fait le chien Aïbo de Sony avec sa balle de jeu rouge ou l'apprentissage du suivi d'une ligne de couleur par un robot),
- l'apprentissage de la navigation d'un robot à partir d'images vidéo de son environnement à 360° (avec un objectif grand angle type fish-eye).

### 8.2.3 Étude du convoyage de cellules par poussée

Un autre objectif du projet de micromanipulation WIMS est de pouvoir déplacer des cellules en les poussant avec le micro-outil. Cet objectif peut impliquer deux problématiques différentes : pousser un objet dans une direction et pousser un objet vers une position cible.

#### Apprendre à pousser un objet dans une direction

Nous avons testé le Dyna-Q parallèle sur cette première tâche. En récompensant le contrôleur quand il pousse un objet vers la droite, on obtient la génération de trajectoires effectuant le convoyage des objets vers la droite. La stratégie de commande est obtenue avec un temps d'apprentissage similaire à celui de la recherche de cible. Le contrôleur arrive rapidement à déplacer les cellules avec l'outil. La figure 8.1 montre un exemple de trajectoire de convoyage (voir aussi la vidéo sur le CD-ROM décrite dans l'annexe C).

#### Apprendre à pousser un objet vers une position cible

Cet objectif est beaucoup plus complexe que le premier. En effet, il s'agit de pousser une cellule tantôt vers la gauche, la droite, le haut ou le bas en fonction de la position relative de la cible. Par exemple, si la cellule est à droite de la cible, il faut la pousser vers la gauche. Si elle est à gauche, il faut la pousser vers la droite. De plus, s'il faut contourner un obstacle, le contrôleur peut être amené à pousser la cellule dans le sens contraire du sens habituel. Ce problème est donc plus complexe qu'il n'y paraît.

Si l'on reste dans la logique parallèle, la cellule à déplacer va constituer une perception et la position cible une autre. Le comportement que le contrôleur doit adopter par rapport à la perception « cellule » dépend de la perception « cible ». Dans l'état actuel de l'architecture, le contrôleur ne peut pas modifier son comportement vis-à-vis d'une perception en fonction de la position d'une autre. Il ne pourra donc pas apprendre cette tâche de poussée.

FIG. 8.1 – *Exemple de convoyage vers la droite de trois objets après apprentissage (avec $h = 10$, $N = 2000$, $\gamma = 0{,}5$ et $\epsilon = 0{,}1$).*

Le problème de la poussée vers une position cible est pour nous d'une complexité de niveau 2 car le comportement du contrôleur dépend simultanément de deux perceptions et il faut la conjonction de deux perceptions pour provoquer une récompense (la cellule et la cible au même endroit). En comparaison, les objectifs précédents étaient de niveau 1 car chaque perception intervenait directement sur le comportement du contrôleur et sur le signal de récompense.

Il faut donc faire évoluer l'architecture parallèle pour intégrer la possibilité d'établir des liens entre deux perceptions et pourquoi pas entre trois, quatre ou plus.

## 8.3 Conclusion

Nous avons montré dans cette thèse la capacité de l'architecture parallèle à apprendre à contrôler un système réel relativement simple sans aucune simulation et en un temps acceptable.

Ces résultats satisfaisants nous motivent à penser que l'apprentissage par renforcement représente une voie prometteuse pour la commande de systèmes complexes dont on ne connaît pas le comportement et notamment dans des domaines où l'humain ne peut intervenir directement avec facilité comme c'est le cas pour la microrobotique.

# Annexe A

# Logiciel de simulation du labyrinthe

Le logiciel que nous avons développé pour tester nos algorithmes sur l'exemple du labyrinthe est disponible sur le CD-ROM de démonstration joint au mémoire. Pour charger le logiciel, il suffit d'exécuter le fichier `Labyrinthe.exe` présent dans le répertoire nommé \`Labyrinthe` (uniquement sous un environnement Windows).

Ce logiciel a été développé en C++ avec Borland Builder 4.0. Il se compose de trois parties distinctes : une application appelée *système* qui gère la simulation du labyrinthe, une application appelée *contrôleur* qui gère les algorithmes d'apprentissage par renforcement et une application appelée *pupitre de test* qui permet de faire fonctionner les deux premières parties ensemble et de réaliser des mesures comme le gain ou le temps de convergence.

## A.1 Application *système*

La fenêtre de l'application *système* (*cf.* figure A.1) permet de visualiser l'état du labyrinthe et de suivre les déplacements de la souris (si l'option *animation système* est activée dans la fenêtre *pupitre de test*).

En cliquant dans le labyrinthe, l'utilisateur peut à sa guise ajouter ou enlever des murs (clique gauche) et des fromages (clique droit).

Enfin, le menu déroulant intitulé *fichier* permet de charger ou sauvegarder un labyrinthe.

## A.2 Application *contrôleur*

La fenêtre de l'application *contrôleur* (*cf.* figure A.2) permet la visualisation de l'état du contrôleur (si l'option *animation contrôleur* est activée dans la fenêtre *pupitre de test*) ainsi que le réglage de ses différents paramètres.

Le groupe *visualisation* permet de sélectionner le type de l'information affichée. Cinq options sont possibles :

- l'option *perceptions* active l'affichage des perceptions reçues par le contrôleur sous la forme de points colorés,

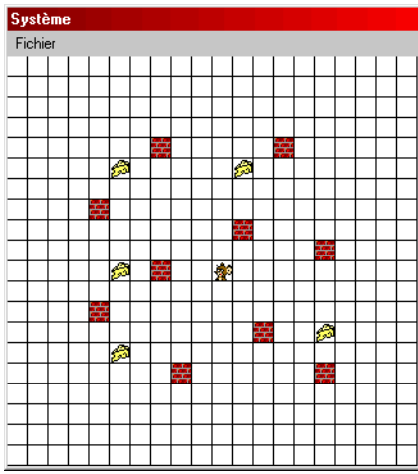



FIG. A.1 – *Fenêtre de l'application système du labyrinthe.*

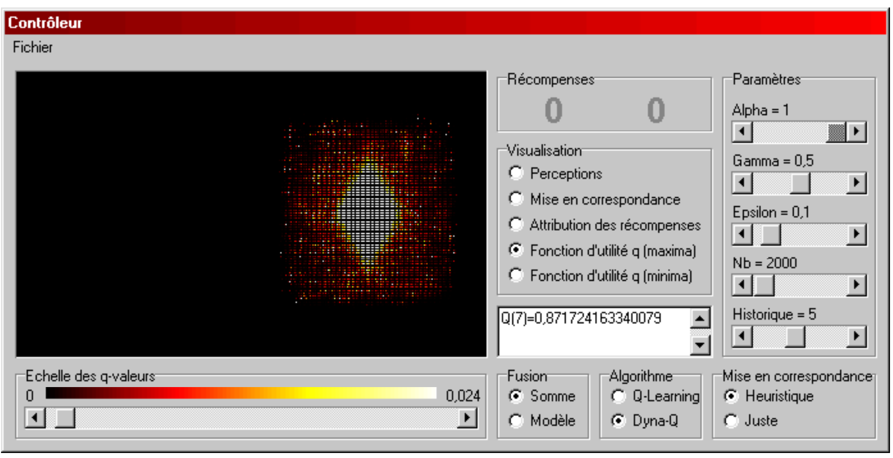



FIG. A.2 – *Fenêtre de l'application contrôleur.*

- l'option *mise en correspondance* active l'affichage de traits reliant les couples de perceptions créés par la fonction de mise en correspondance,
- l'option *attribution des récompenses* active l'affichage de la récompense attribuée à un couple de perceptions sous la forme d'un point si la récompense est nulle, d'un signe - si la récompense est négative et d'un signe + si la récompense est positive,
- l'option *fonction d'utilité q (maxima)* active l'affichage des valeurs maximales de la fonction d'utilité ($\max_u q(x,u)$) pour chaque perception $x$ à l'aide de points colorés selon l'échelle des couleurs située en bas à droite de la fenêtre,
- l'option *fonction d'utilité q (minima)* active l'affichage des valeurs minimales de la fonction d'utilité ($\min_u q(x,u)$) pour chaque perception $x$ à l'aide de points colorés selon l'échelle des couleurs située en bas à droite de la fenêtre.

Le groupe *échelle des q-valeurs* permet de modifier l'échelle des couleurs pour l'affichage de la fonction d'utilité.

Le groupe *récompenses* affiche la somme des récompenses négatives et positives courantes.

Le groupe *paramètres* permet de choisir les paramètres de l'apprentissage :
- *Alpha* correspond au coefficient d'apprentissage $\alpha$,
- *Gamma* correspond au coefficient de pondération des récompenses futures $\gamma$,
- *Epsilon* correspond au taux d'exploration de l'$\epsilon$-gourmand,
- *Nb* correspond au nombre $N$ de mises à jour par période d'échantillonnage,
- *Historique* correspond à la dimension $h$ de l'historique mémorisé par le modèle $\mathsf{m}$.

Les groupes *fusion*, *algorithme* et *mise en correspondance* permettent de sélectionner les algorithmes utilisés par les différentes fonctions du contrôleur.

La boîte de texte affiche les estimations de l'espérance de gain globale $Q_t(u)$ pour la situation courante et chaque commande $u$.

Enfin, le menu déroulant intitulé *fichier* permet de charger ou sauvegarder la mémoire du contrôleur ainsi que l'exportation dans un fichier texte des valeurs de la fonction d'utilité.

## A.3 Application *pupitre de test*

Le pupitre de test contrôle l'ensemble du logiciel (*cf.* figure A.3).

Le groupe *cycle* permet de lancer ou de stopper l'exécution de la boucle sensori-motrice. Les animations du système et du contrôleur peuvent être activées (au détriment de la vitesse d'exécution).

Le groupe *cycle automatique* permet de lancer plusieurs fois la boucle sensori-motrice sans animation. Le nombre de boucles est précisé dans le bloc de texte.

Le groupe *mode manuel* permet à l'utilisateur de guider le contrôleur. L'apprentissage se poursuit mais l'utilisateur décide lui-même des commandes à effectuer.

Le groupe *test performance* permet de réaliser un enregistrement du gain moyen au cours de la simulation.

Le groupe *test paramètre* permet de mesurer le gain limite et le temps d'apprentissage pour différentes valeurs du paramètre choisi.

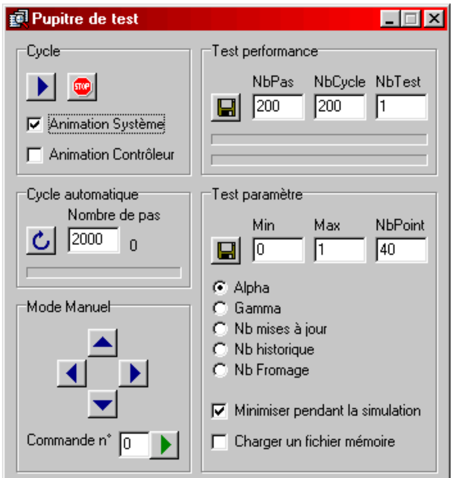



FIG. A.3 – *Fenêtre de l'application pupitre de test.*

## A.4 Description des exemples automatiques de démonstration du logiciel

Des versions automatiques de démonstration du logiciel sont disponibles sur le CD-ROM joint au mémoire. Le tableau A.1 en donne la liste.

| Aperçu | Description | Fichier |
|---|---|---|
|  | Apprentissage du Dyna-Q parallèle avec des fromages : le contrôleur apprend à attraper les fromages en quelques dizaines de secondes. | `Labyrinthe1.exe` |
|  | Trajectoire de la souris après apprentissage avec des cibles et des obstacles. | `Labyrinthe2.exe` |
|  | Illustration du problème des impasses avec la fusion somme. | `Labyrinthe3.exe` |
|  | Résolution du problème des impasses avec la fusion de modèles et une mise en correspondance exacte. | `Labyrinthe4.exe` |
|  | Résolution du problème des impasses avec la fusion de modèles et une mise en correspondance exacte. | `Labyrinthe5.exe` |

TAB. A.1 – *Liste des exemples de démonstration du logiciel.*

# Annexe B

# Plateforme de macro-manipulation WIMS

La plateforme de macro-manipulation est constituée d'un macro-manipulateur permettant de déplacer l'outil au milieu des objets et d'un système informatique équipé d'une carte d'acquisition vidéo et d'une carte de commande reliée au macro-manipulateur. La figure B.1 présente une vue d'ensemble de la plateforme.

## B.1 Macro-manipulateur

Le macro-manipulateur permet de déplacer un outil par l'intermédiaire d'un aimant permanent situé sous la zone de manipulation (figure B.2) et fixé à une table de positionnement deux axes.

### B.1.1 Outil et objets

L'outil est un cylindre en acier de 5 mm de diamètre et de 2,5 mm de haut. Pour être clairement visible par la caméra, l'outil est peint en noir.

Les objets sont des billes de plastique normalement destinées à l'injection. Leur diamètre est compris entre 3 et 4 mm. Pour pouvoir distinguer les billes de l'outil, les billes sont de couleur grise.

### B.1.2 Table de positionnement de l'aimant

La partie mécanique de la table de positionnement (*cf.* figure B.3) a été réalisée lors d'un projet de fin d'études par des étudiants de l'E.N.S.M.M. (École Nationale Supérieure de Mécanique et des Microtechniques). Elle permet de déplacer l'aimant permanent sous la plaque de verre. Elle peut effectuer des déplacements selon deux axes orthogonaux et horizontaux. Les axes sont pilotés par des moteurs à courant continu non asservis. Le tableau B.1 résume les caractéristiques techniques de la table de positionnement.

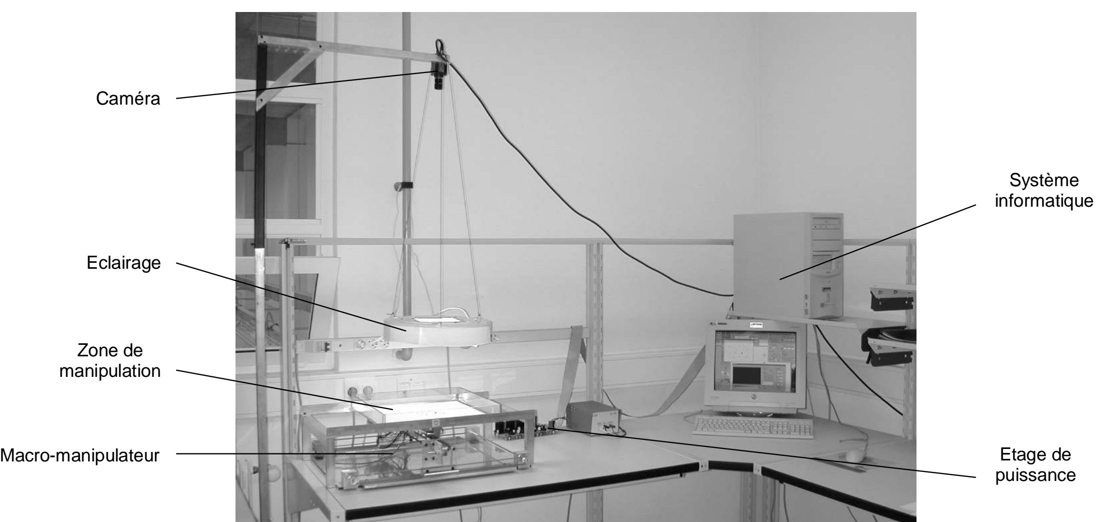
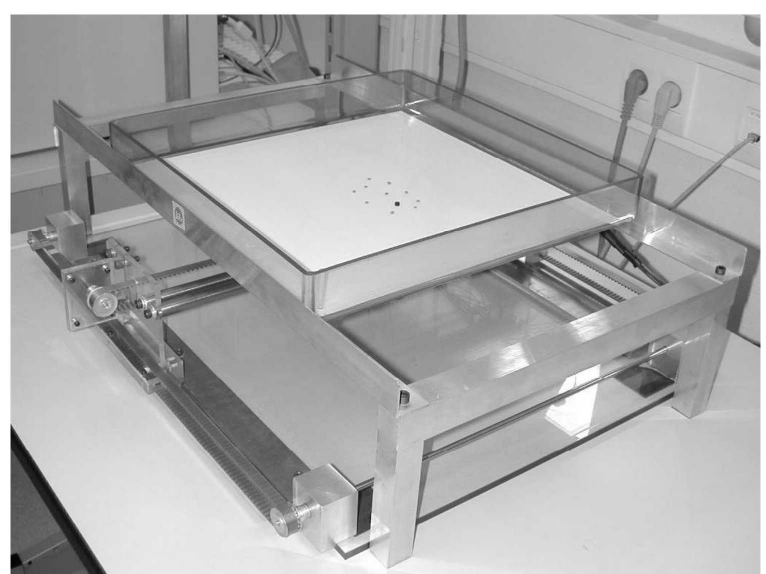


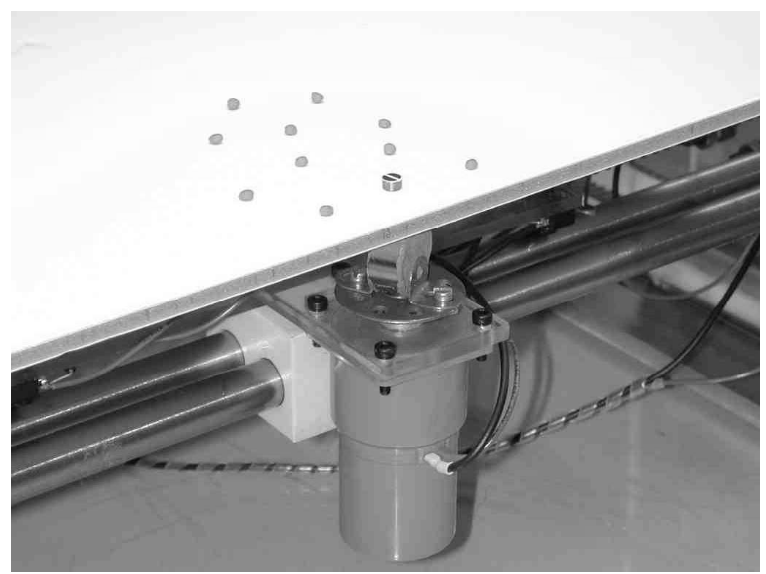

FIG. B.1 – *Vue d'ensemble de la plateforme de macro-manipulation WIMS.*

| | Axe 1 | Axe 2 |
|---|---|---|
| Course | 400 mm | 300 mm |
| Vitesse maximum | 60 mm/s | |
| Motorisation | Moteur à courant continu *Premotec* | |
| Tension nominale | 6 V | |
| Courant maximum | 360 mA / moteur | |

TAB. B.1 – *Caractéristiques de la table de positionnement.*



(a) Macro-manipulateur



(b) Détail de la liaison magnétique aimant—outil

Fig. B.2 – *Macro-manipulateur WIMS.*

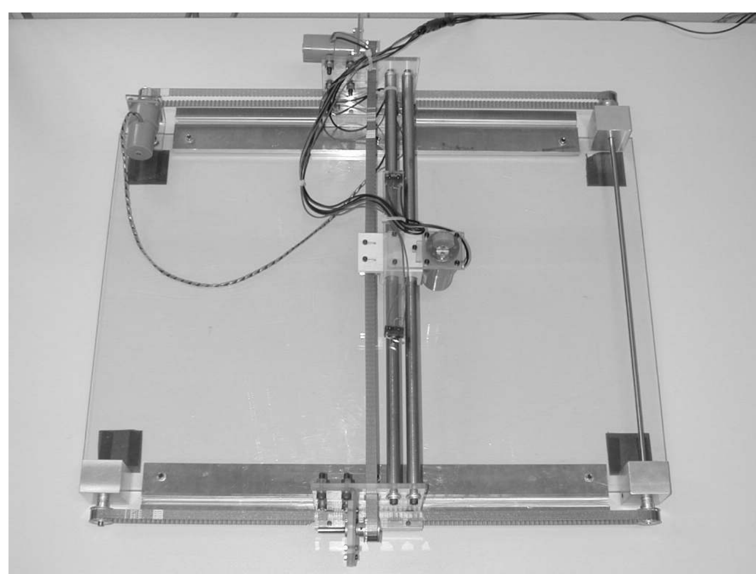


FIG. B.3 – *Table de positionnement deux axes.*

| Caméra | Sony XC-8500CE Progressive |
|---|---|
| Résolution | 752 × 560 |
| Niveaux de gris | 256 |
| Fréquence maximum | 25 images/s |
| Image | Non entrelacée |
| Objectif | 50 mm |
| Échelle | 0,173 mm/pixel |
| Carte d'acquisition | Matrox Meteor II |
| Traitement vidéo | Matrox Imaging Library 6.0 |

TAB. B.2 – *Caractéristiques du matériel d'acquisition vidéo.*

### B.1.3 Électronique de puissance

Les moteurs à courant continu de la table de positionnement sont pilotés par l'intermédiaire d'un étage de puissance que nous avons réalisé au laboratoire (*cf.* figure B.4). Cet amplificateur est fondé sur l'utilisation des amplificateurs opérationnels de type SGS-Thomson L165.

## B.2 Système informatique

### B.2.1 Acquisition vidéo

Une caméra vidéo reliée à une carte d'acquisition vidéo permet d'obtenir des images de la zone de manipulation. Les axes de la table de positionnement et de l'image vidéo

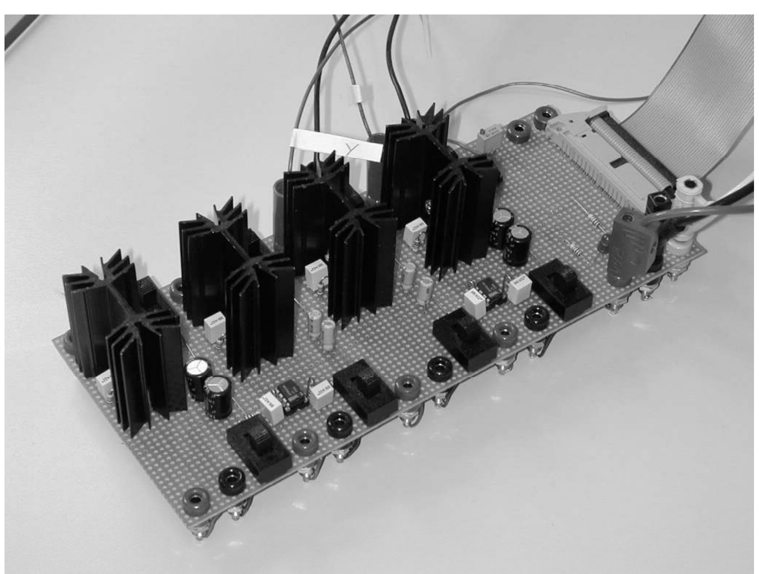


FIG. B.4 – *Étage de puissance pour l'alimentation des moteurs à courant continu.*

sont parallèles. Les caractéristiques du matériel d'acquisition sont décrites dans le tableau B.2.

A partir des images vidéo, une étape de traitement permet d'extraire la position du barycentre de l'outil et des objets (*cf.* figure B.5).

### B.2.2 Calculateur

Le micro-ordinateur utilisé pour le traitement vidéo ainsi que pour le contrôle est un Pentium III cadencé à 450 MHz, et équipé de 192 Mo de mémoire vive. Le système d'exploitation est Windows 98SE.

### B.2.3 Carte de commande

Le micro-ordinateur est équipé d'une carte à 8 sorties analogiques (Keithley DDA-08/16) destinées à commander les moteurs de la table de positionnement par l'intermédiaire de l'étage d'amplification.

## B.3 Logiciel

Comme le logiciel de simulation du labyrinthe, le logiciel que nous avons développé pour tester nos algorithmes sur le macro-manipulateur WIMS se compose de trois parties : une application *système* qui gère le macro-manipulateur et les applications *contrôleur* et *pupitre de test*. Les applications *contrôleur* et *pupitre de test* sont identiques à celles employées avec le labyrinthe (*cf.* annexe A).

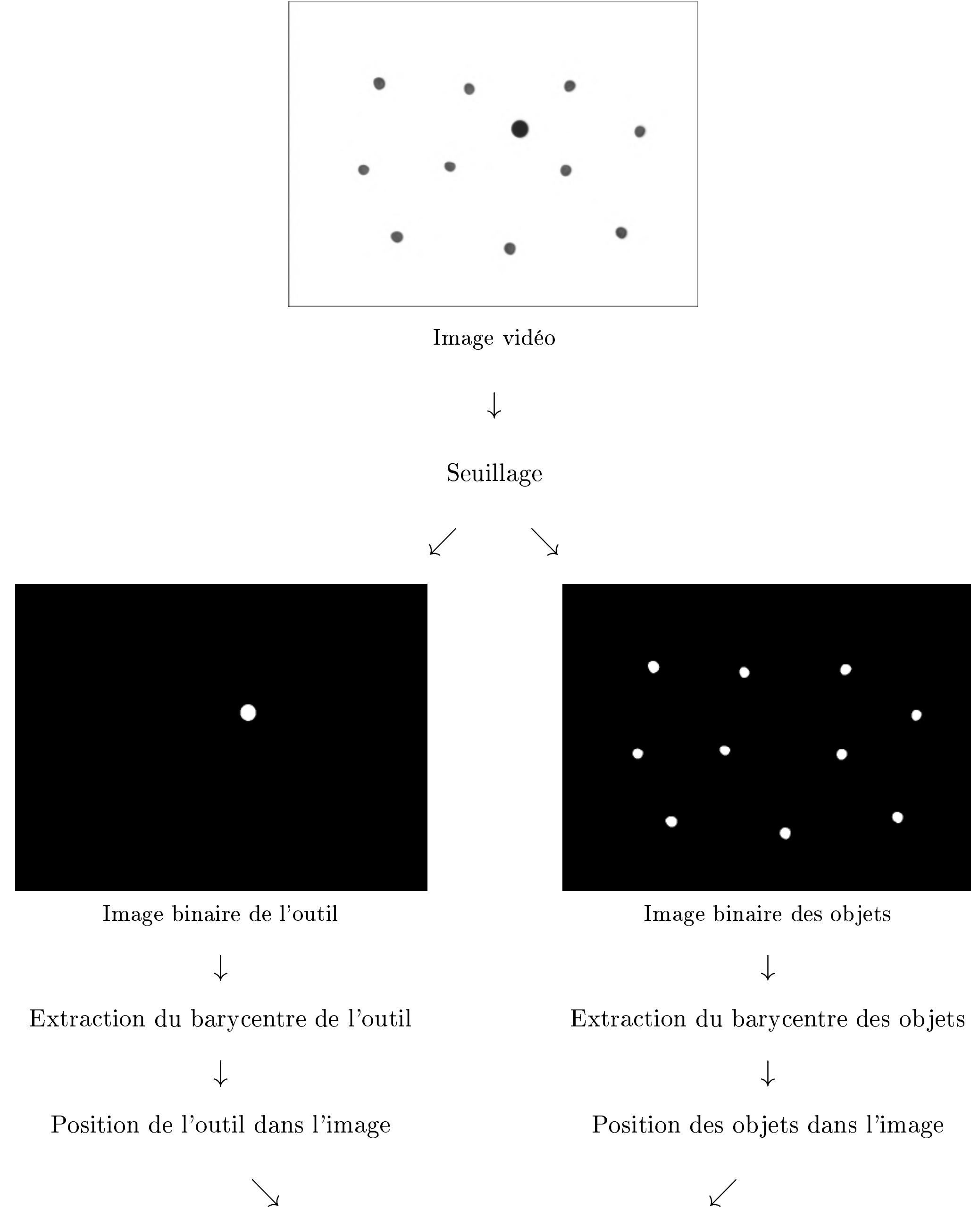



FIG. B.5 – *Traitement de l'image vidéo.*

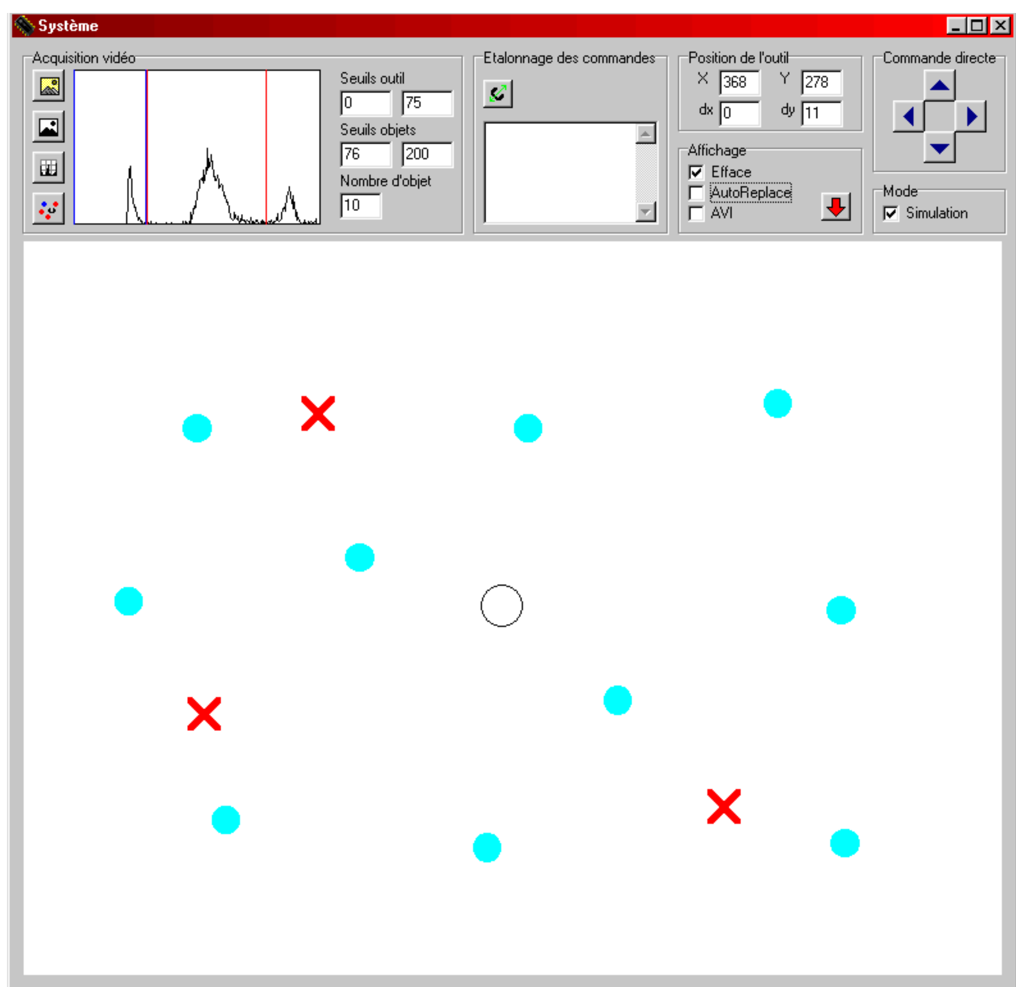



FIG. B.6 – *Fenêtre de l'application système du macro-manipulateur WIMS.*

Outre la visualisation graphique de l'état du système, la fenêtre de l'application *système* présente 6 groupes de fonctions (*cf.* figure B.6) :

- le groupe d'*acquisition vidéo* permet de provoquer l'affichage de l'image vidéo et de l'image binaire, de visualiser l'histogramme des niveaux de gris de l'image et de régler les seuils de capture de l'outil et des objets,
- le groupe *étalonnage des commandes* permet de régler l'échelle entre les commandes moteurs et les déplacements sur l'image vidéo,
- le groupe *position de l'outil* donne la position courante de l'outil ainsi que sa variation par rapport à la position précédente,
- le groupe *affichage* permet de choisir le mode d'affichage (effacer les objets entre chaque commande ou laisser les traces), de préciser si une cible virtuelle est replacée automatiquement quand l'outil l'a atteinte et d'activer l'enregistrement des images vidéo dans un fichier AVI,
- le groupe *commande directe* permet d'effectuer des déplacements manuels de la table de positionnement,
- enfin, le groupe *mode* permet de passer en mode simulé ou en mode réel.

# Annexe C

# Description des enregistrements vidéo

Des enregistrements vidéo montrant les trajectoires obtenues après apprentissage sont disponibles sur le CD-ROM joint au mémoire. Le tableau C.1 en donne la liste.

| Aperçu | Description | Fichier |
| --- | --- | --- |
| | Exemple de trajectoire de l'outil sur le système réel après apprentissage avec uniquement des cibles (avec $h = 10$, $N = 2000$, $\gamma = 0{,}5$ et $\epsilon = 0{,}1$). | `video1.avi` |
| | Exemple de trajectoire de l'outil sur le système réel après apprentissage avec des cibles et des obstacles (avec $h = 10$, $N = 2000$, $\gamma = 0{,}5$ et $\epsilon = 0{,}1$). | `video2.avi` |
| | Exemple de convoyage vers la droite de trois objets après apprentissage (avec $h = 10$, $N = 2000$, $\gamma = 0{,}5$ et $\epsilon = 0{,}1$). | `video3.avi` |

TAB. C.1 – *Liste des enregistrements vidéos.*

# Bibliographie

ALBUS, J. S. (1975). « A New Approach to Manipulator Control: The Cerebellar Model Articulator Controller (CMAC) », *Trans. of the ASME, J. Dynamic Systems, Meaasurement, and Control*, **97**(3): 220–227.

ARBIB, Michael A. (1981). « Perceptual structures and distributed motor control », *Handbook of physiology, Section 2: the nervous system, vol. II, Motor control, Part 1*, pp. 1449–1480.

ARKIN, Ronald C. (1989). « Motor schema–based mobile robot navigation », *International Journal of Robotics Research*, **8**(4): 92–112.

ARKIN, Ronald C. (1998). *Behavior–Based Robotics*, The MIT Press.

ARNAUD, Pierre (2000). *Des moutons et des robots : Architecture de contrôle réactive et déplacements collectifs de robots*, Presses polytechniques et universitaires romandes.

BALCH, Tucker et ARKIN, Ronald C. (1993). « Avoiding the past: a simple but effective strategy for reactive navigation », *Proc. of the IEEE International Conference on Robotics and Automation*, Vol. 1, Atlanta, GA, pp. 678–685.

BARTO, Andrew G. , BRADTKE, Steven J. et SINGH, Satinder P. (1995). « Learning to act using real–time dynamic programming », *Artificial Intelligence*, **72**: 81–138.

BARTO, Andrew G. , SUTTON, Richard S. et ANDERSON, Charles W. (1983). « Neuronlike elements that can solve difficult learning control problems », *IEEE Transactions on Systems, Man, and Cybernetics*, **13**(5): 835–846. Reprinted in J. A. Anderson and E. Rosenfeld, Neurocomputing: Foundations of Research, MIT Press, Cambridge, MA, 1988.

BELLMAN, Richard (1957a). *Dynamic Programming*, Princeton University Press, Princeton, NJ.

BELLMAN, Richard (1957b). « A Markov decision process », *Journal of Mathematical Mechanics*, **6**: 679–684.

BERTSEKAS, D. P. et TSITSIKLIS, J. N. (1996). *Neural Dynamic Programming*, Athena Scientific, Belmont, MA.

BOYAN, J. A. et MOORE, A. W. (1995). « Generalization in reinforcement learning: Safely approximating the value function », *Proc. of the Advances in Neural Information Processing Systems 7*, The MIT Press, Cambridge, MA.

BROOKS, Rodney A. (1986). « A Robust Layered Control System for a Mobile Robot », *IEEE Journal of Robotics and Automation*, **2**(1): 14–23. also MIT AI Memo 864, September 1985.

BROOKS, Rodney A. (1989). « A Robot That Walks; Emergent Behaviors from a Carefully Evolved Network », *Memo No. 1091*, MIT AI Lab.

BROOKS, Rodney A. (1990). « Elephants Don't Play Chess », *Robotics and Autonomous Systems*, **6**: 3–15.

CARRERAS, M. , BATTLE, J. et RIDAO, P. (2001). « Hybrid Coordination of Reinforcement Learning–based Behaviors for AUV Control », *Proc. of the International Conference on Intelligent Robots and Systems*, Maui, Haiwaii, USA.

CHAPMAN, David et KAELBLING, Leslie Pack (1991). « Input generalization in delayed reinforcement learning: An algorithm and performance comparisons », *Proc. of the International Joint Conference on Artificial Intelligence*, Sydney, Australia.

CICHOSZ, Pawel (1995). « Truncated temporal differences: On the efficient implementation of TD($\lambda$) for reinforcement learning », *Journal of Artificial Intelligence Research (JAIR)*, **2**: 287–318.

CRITES, Robert H. et BARTO, Andrew G. (1996). « Improving elevator performance using reinforcement learning », *Advances in Neural Information Processing Systems 8*, The MIT Press, Cambridge, MA.

DAYAN, P. et SEJNOWSKI, T. (1994). « TD($\lambda$) converges with probability 1 », *Machine Learning*, **14**: 295–301.

DAYAN, Peter (1992). « The convergence of TD($\lambda$) for general $\lambda$ », *Machine Learning*, **8**: 341–362.

FABIANI, Patrick , FARGES, Jean-Loup et GARCIA, Frédérick (2001). « Décision dans l'incertain », *Cours SUPAÉRO*, ONERA–CERT / INRA–UBIA.

FAIHE, Yassine (1999). *Hierarchical Problem Solving using Reinforcement Learning: Methodology and Methods*, PhD thesis, University of Neuchâtel, Departement of Computer Science, Switzerland.

GAUTHIER, Michaël et PIAT, Emmanuel (2002). « Force study applied to a biological objects planar micromanipulator », *Journal Européen des Systèmes Automatisés*, **à paraître**.

GLORENNEC, Pierre-Yves (1994). « Fuzzy Q–Learning and dynamical fuzzy Q–Learning », *Proc. of the 3th IEEE Fuzzy systems conference*, Orlando.

HALLERDAL, Martin (2001). « *Investigating W–learning on a Khepera robot* », Master's thesis, University of Edinburgh.

HALLERDAL, Martin et HALLAM, John (2002). « Behaviour Selection on a Mobile Robot using W–learning », *Proc. of the Seventh International Conference on the Simulation of Adaptive Behavior: From Animals to Animats 7*, Edinburgh, UK.

HOWARD, Ronald A. (1960). *Dynamic Programming and Markov Processes*, The MIT Press, Cambridge, MA.

HUMPHRYS, Mark (1995). « W–learning: Competition among selfish Q–learners », *Technical Report 362*, University of Cambridge.

HUMPHRYS, Mark (1996a). « Action Selection methods using Reinforcement », *From Animals to Animats 4: Proc. of the Fourth International Conference on Simulation of Adaptive Behavior (SAB–96)*, Massachusetts, USA, pp. 135–44.

HUMPHRYS, Mark (1996b). *Action Selection methods using Reinforcement Learning*, PhD thesis, University of Cambridge, Computer Laboratory.

JAAKKOLA, Tommi , JORDAN, Michael I. et SINGH, Satinder P. (1994). « On the convergence of stochastic iterative dynamic programming algorithms », *Neural Computation*, **6**(6).

KAELBLING, L. P. , LITTMAN, M. L. et MOORE, A. W. (1996). « Reinforcement Learning: A Survey », *Journal of Artificial Intelligence Research*, **4**: 237–285.

KAELBLING, Leslie Pack (1993a). « Hierarchical learning in stochastic domains: Preliminary results », *Proc. of the Tenth International Conference on Machine Learning*, Morgan Kaufmann, Amherst, MA.

KAELBLING, Leslie Pack (1993b). *Learning in Embedded Systems*, The MIT Press, Cambridge, MA.

KHATIB, Oussama (1986). « Real–time Obstacle Avoidance for Manipulators and Mobile Robots », *The International Journal of Robotics Research*, **5**(1).

KIM, J.H. , SUH, I.H. , OH, S.R. , CHO, Y.J. et CHUNG, Y.K. (1997). « Region–based Q–learning using Convex Clustering Approach », *Proc. of the IEEE International Conference on Intelligent Robots and Systems*, pp. 601–607.

KIM, Min-Soeng , HONG, Sun-Gi et LEE, Ju-Jang (1999). « On–Line Fuzzy Q–Learning with Extended Rule and Interpolation Technique », *Proc. of the IEEE International Conference on Intelligent Robots and Systems*, pp. 757–762.

KLOPF, Harry A. (1972). « Brain function and adaptive systems—A heterostatic theory », *Technical Report AFCRL–72–0164*, Air Force Cambridge Research Laboratories. A summary appears in Proceedings of the International Conference on Systems, Man, and Cybernetics, 1974, IEEE Systems, Man, and Cybernetics Society, Dallas, TX.

KLOPF, Harry A. (1975). « A comparison of natural and artificial intelligence », *SIGART Newsletter*, **53**: 11–13.

LITTMAN, M. L. , DEAN, T. L. et KAEBLING, L. P. (1995). « On the complexity of Solving Markov Decision Problems », *Proc. of the UAI'95*, Vol. 11, pp. 394–402.

LONG-JI, Lin (1992). « Self–improving reactive agents based on reinforcement learning, planning and teaching », *Machine Learning*, **8**: 293–321.

LONG-JI, Lin (1993a). « Hierarchical learning of robot skills by reinforcement », *Proc. of the International Conference on Neural Networks.*

LONG-JI, Lin (1993b). *Reinforcement Learning for Robots Using Neural Network*, PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA15213.

MAES, Pattie (1989). « How to Do the Right Thing », *Connection Science Journal*, **1**(3): 291–323. Also MIT AI–Memo 1180.

MAES, Pattie et BROOKS, Rodney A. (1990). « Learning to coordinate behaviors », *Proc. of the Eighth National Conference on Artificial Intelligence*, Morgan Kaufmann, pp. 796–802.

MAHADEVAN, Sridhar (1994). « To discount or not to discount in reinforcement learning: A case study comparing R–learning and Q–learning », *Proc. of the Eleventh International Conference on Machine Learning*, Morgan Kaufmann, San Francisco, CA, pp. 164–172.

MAHADEVAN, Sridhar (1996). « Average reward reinforcement learning: Foundations, algorithms, and empirical results », *Machine Learning*, **22**(1): 159–196.

MAHADEVAN, Sridhar et CONNELL, Jonathan (1991). « Automatic programming of behavior-based robots using reinforcement learning », *Research Report NY10598*, IBM T.J. Watson Research Center.

MATARIC, Maja J. (1994). « Reward functions for accelerated learning », dans W. W. COHEN et H. HIRSH (éds), *Proc. of the Eleventh International Conference on Machine Learning*, Morgan Kaufmann.

MCCALLUM, Andrew Kachites (1995). *Reinforcement Learning with Selective Perception and Hidden State*, PhD thesis, Department of Computer Science, University of Rochester.

MEULEAU, Nicolas (1996). *Le Dilemme entre Exploration et Exploitation dans l'Apprentissage par renforcement*, PhD thesis, Cemagref, Université de Caen.

MICHIE, D. et CHAMBERS, R. A. (1968). « BOXES: An experiment in adaptive control », *Machine Intelligence 2*, pp. 137–152.

MICHIE, Donald (1961). « Trial and error », *Science Survey, Part 2*, pp. 129–145.

MOORE, Andrew W. et ATKESON, Christopher G. (1993). « Prioritized sweeping: Reinforcement learning with less data and less real time », *Machine Learning*, **13**.

MUNOS, Rémi (1997). *L'apprentissage par renforcement : Etude du cas continu*, PhD thesis, Cemagref, Ecole des Hautes Etudes en Sciences Sociales.

NDIAYE, Seydina Moussa (1999). *Apprentissage par renforcement en horizon fini : application à la génération de règles pour la conduite de culture*, PhD thesis, Institut National de la Recherche Agronomique, Université Paul Sabatier, Toulouse.

PAULS, Jackson (2001). « Pigs and People », *Project report*, University of Edinburgh, Division of Informatics.

PENG, Jing (1993). *Efficient Dynamic Programming–Based Learning for Control*, PhD thesis, Northeastern University, Boston, MA.

PENG, Jing et WILLIAMS, Ronald J. (1993). « Efficient learning and planning within the Dyna framework », *Adaptive Behaviour*, **1**(4): 437–454.

PENG, Jing et WILLIAMS, Ronald J. (1994). « Incremental multi-step Q-learning », *Proc. of the Eleventh International Conference on Machine Learning*, Morgan Kaufmann, San Francisco, CA, pp. 226–232.

PETTERSSON, Lennart (1997). « Control System Architectures for Autonomous Agents — A Survey Study », *Technical report*, Department of Machine Design, KTH.

PIRJANIAN, Paolo (1998). *Multiple Objective Action Selection and Behavior Fusion Using Voting*, PhD thesis, Institute of Electronic Systems, Aalborg University, Denmark.

PRECUP, Doina et SUTTON, Richard S. (1997). « Exponentiated gradient methods for reinforcement learning », *Proc. of the 14th International Conference on Machine Learning*, Morgan Kaufmann, pp. 272–277.

RICHARD, Nadine (2001). *Description de comportements d'agents autonomes évoluant dans des mondes virtuels*, PhD thesis, ENST, Paris.

ROSENBLATT, Julio K. (1997a). *DAMN: A Distributed Architecture for Mobile Navigation*, PhD thesis, Carnegie Mellon University Robotics Institute, Pittsburgh, PA.

ROSENBLATT, Julio K. (1997b). « The Distributed Architecture for Mobile Navigation », *Journal of Experimental and Theoretical Artificial Intelligence*, **9**(2/3): 339–360.

ROSENBLATT, Julio K. (2000). « Optimal Selection of Uncertain Actions by Maximizing Expected Utility », *Autonomous Robots*, **9**(1): 17–25.

ROSENBLATT, Julio K. et PAYTON, David (1989). « A Fine–Grained Alternative to the Subsumption Architecture for Mobile Robot Control », *Proc. of the IEEE/INNS International Joint Conference on Neural Networks*, Vol. 2, Washington DC, pp. 317–324.

RUMMERY, G. A. (1995). *Problem Solving with Reinforcement Learning*, PhD thesis, Cambridge University, Cambridge, England.

RUMMERY, G. A. et NIRANJAN, M. (1994). « On–Line Q–learning using connectionist systems », *Technical report CUED/F–INFENG/TR 166*, Cambridge University Engineering Department, Cambridge, England.

SAMUEL, Arthur L. (1959). « Some studies in machine learning using the game of checkers », *IBM Journal of Research and Development*, **3**: 211–229. Reprinted in by E.A. Feigenbaum and J. Feldman, editors, Computers and Thought, pp. 71–105, McGraw–Hill, New York, 1963.

SCHRAUDOLPH, Nicol N. , DAYAN, Peter et SEJNOWSKI, Terrence J. (1994). « Temporal difference learning of position evaluation in the game of Go », dans J. D. COWAN, G. TESAURO et J. ALSPECTOR (éds), *Proc. of the Advances in Neural Information Processing Systems 6*, Morgan Kaufmann, San Mateo, CA, pp. 817–824.

SCHWARTZ, A. (1993). « A reinforcement learning method for maximizing undiscounted rewards », *Proc. of the Tenth International Conference on Machine Learning*, Vol. 10, Morgan Kaufmann, Amherst, Massachusetts, pp. 298–305.

SEHAD, Samira (1996). *Contribution à l'étude et au développement de modèles connexionnistes à apprentissage par renforcement : application à l'acquisition de comportements adaptatifs*, PhD thesis, Université de Montpellier II.

SINGH, Satinder P. et SUTTON, Richard S. (1996). « Reinforcement learning with replacing eligibility traces », *Machine Learning*, **22**: 123–158.

SINGH, Satinder Pal (1992). « Transfer of learning by composing solutions of elemental sequential tasks », *Machine Learning*, **8**(3): 323–340.

SUTTON, Richard S. (1984). *Temporal Credit Assignment in Reinforcement Learning*, PhD thesis, University of Massachusetts, Amherst, MA.

SUTTON, Richard S. (1988). « Learning to predict by the method of temporal differences », *Machine Learning*, **3**: 9–44.

SUTTON, Richard S. (1990). « Integrated architectures for learning, planning, and reacting based on approximating dynamic programming », *Proc. of the Seventh International Conference on Machine Learning*, Morgan Kaufmann, San Mateo, CA, pp. 216–224.

SUTTON, Richard S. (1991a). « Dyna, an Integrated Architecture for Learning, Planning, and Reacting », *Working Notes of the 1991 AAAI Spring Symposium*, pp. 151–155. and SIGART Bulletin, 2:160-163.

SUTTON, Richard S. (1991b). « Planning by Incremental Dynamic Programming », *Proc. of the Ninth Conf. on Machine Learning*, pp. 353–357.

SUTTON, Richard S. (1992). « Adapting Bias by Gradient Descent: An Incremental Version of Delta–Bar–Delta », *Proc. of the Tenth National Conf. on Artificial Intelligence*, The MIT Press, pp. 171–176.

SUTTON, Richard S. (1995). « Generalization in reinforcement learning: Successful examples using sparse coarse coding », *Advances in Neural Information Processing Systems 8*, The MIT Press, pp. 1038–1044.

SUTTON, Richard S. et BARTO, Andrew G. (1998). *Reinforcement Learning: An Introduction*, The MIT Press, Cambridge.

TESAURO, Gerald (1992). « Practical issues in temporal difference learning », *Machine Learning*, **8**: 257–277.

TESAURO, Gerald (1994). « TD-Gammon, a self-teaching backgammon program, achieves master-level play », *Neural Computation*, **6**(2): 215–219.

TESAURO, Gerald (1995). « Temporal Difference Learning and TD–Gammon », *Communications of the Association for Computing Machinery*, **38**(3).

THAM, Chen K. et PRAGER, Richard W. (1994). « A modular Q-learning architecture for manipulator task decomposition », *Proc. of the Eleventh International Conference on Machine Learning*, Morgan Kaufmann, San Francisco, CA.

THORNDIKE, Edward L. (1911). *Animal Intelligence*, Hafner, Darien, CT.

THRUN, Sebastian (1995). « Learning to play the game of chess », dans G. TESAURO, D. S. TOURETZKY et T. K. LEEN (éds), *Proc. of the Advances in Neural Information Processing Systems 7*, The MIT Press, Cambridge, MA.

THRUN, Sebastian et SCHWARTZ, Anton (1993). « Issues in using function approximation for reinforcement learning », dans M. MOZER, P. SMOLENSKY, D. TOURETZKY, J. ELMAN et A. WEIGEND (éds), *Proc. of the Connectionist Models Summer School*, Lawrence Erlbaum, Hillsdale, NJ.

THRUN, Sebastian B. (1992). « The role of exploration in learning control », *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approache*, **22**: 29–94.

TOUZET, Claude (1999). *L'apprentissage par renforcement, Connexionisme et applications*, C. Jutten, Masson. A paraître.

TSITSIKLIS, John N. (1994). « Asynchronous stochastic approximation and Q-Learning », *Machine Learning*, **16**(3).

TSITSIKLIS, John N. et ROY, Ben Van (1996). « Feature-based methods for large scale dynamic programming », *Machine Learning*, **22**(1).

TURING, Allan M. (1950). « Computing Machinery and Intelligence », *Mind*, **59**: 433–460. Reprinted in E. A. Feigenbaum and J. Feldman editors, Computers and Thought, pp. 11–35, McGraw–Hill, New York, 1963.

TYRRELL, Toby (1993). *Computational mechanisms for action selection*, PhD thesis, University of Edinburgh.

WATKINS, Christopher J.C.H. (1989). *Learning from Delayed Rewards*, PhD thesis, Cambridge University, Cambridge, England.

WATKINS, Christopher J.C.H. et DAYAN, Peter (1992). « Technical Note: Q–Learning », *Machine Learning*, **8**: 279–292.

ZHANG, Wei et DIETTERICH, Thomas G. (1995). « A reinforcement learning approach to job–shop scheduling », *Proc. of the IJCAI95*.

# Résumé

En microrobotique, la commande des systèmes est délicate car les phénomènes physiques liés à l'échelle microscopique sont complexes. Les méthodes dites d'apprentissage par renforcement constituent une approche intéressante car elles permettent d'établir une stratégie de commande sans connaissance *a priori* sur le système. Au vu des grandes dimensions des espaces d'états des systèmes étudiés, nous avons développé une approche parallèle qui s'inspire à la fois des architectures comportementales et de l'apprentissage par renforcement. Cette architecture, basée sur la parallélisation de l'algorithme du Q-Learning, permet de réduire la complexité du système et d'accélérer l'apprentissage. Sur une application simple de labyrinthe, les résultats obtenus sont bons mais le temps d'apprentissage est trop long pour envisager la commande d'un système réel. Le Q-Learning a alors été remplacé par l'algorithme du Dyna-Q que nous avons adapté à la commande de systèmes non déterministes en ajoutant un historique des dernières transitions. Cette architecture, baptisée Dyna-Q parallèle, permet non seulement d'améliorer la vitesse de convergence, mais aussi de trouver de meilleures stratégies de contrôle. Les expérimentations sur le système de manipulation montrent que l'apprentissage est alors possible en temps réel et sans utiliser de simulation. La fonction de coordination des comportements est efficace si les obstacles sont relativement éloignés les uns des autres. Si ce n'est pas le cas, cette fonction peut créer des maxima locaux qui entraînent temporairement le système dans un cycle. Nous avons donc élaboré une autre fonction de coordination qui synthétise un modèle plus global du système à partir du modèle de transition construit par le Dyna-Q. Cette nouvelle fonction de coordination permet de sortir très efficacement des maxima locaux à condition que la fonction de mise en correspondance utilisée par l'architecture soit robuste.



**Mots-clés :** commande par apprentissage, processus décisionnels de Markov, programmation dynamique, apprentissage par renforcement, Q-Learning, Dyna-Q, architecture comportementale, microrobotique, micromanipulation.



# Abstract

In the microrobotics field, the control of systems is difficult because the physical phenomena connected to the microscopic scale are complex. The reinforcement learning methods constitute an interesting approach because they allow to draw up a control policy without any knowledge of the system. With regard to the large dimensions of the state spaces of the studied systems, we developed a parallel approach which is inspired by the behaviour-based architectures and by the reinforcement learning. This architecture is based on parallel Q-Learning algorithms. It allows to reduce the system complexity and to speed up the learning process. On the gridworld example, the results are good but the learning time is too long to control a real system. Then, the Q-Learning algorithm was replaced by the Dyna-Q algorithm which we adapted to the control of no deterministic systems by using a chronological account of the last transitions. This architecture, called parallel Dyna-Q, allows to increase the convergence speed and also to find better control policies. The experiments done with the real manipulation system show that the learning is possible in real time without no need of simulations. The behaviours co-ordination function works well if the obstacles are separated from each others. If that is not case, it can create local maxima which trap temporarily the system in a cycle. So, we developed another co-ordination function which creates a more global model of the system from the model of transition built with the Dyna-Q algorithm. This new co-ordination function allows to go out of local maxima if the temporal pattern matching function used by the architecture is sturdy.



**Keywords:** control by learning, Markovian decision process, dynamic programming, reinforcement learning, Q-Learning, Dyna-Q, behaviour-based architecture, microrobotics, micromanipulation.