Université Paris-Sud XI

Approches évolutionnaires pour la robotique modulaire et anticipatoire

THÈSE

Défendue le 29 septembre 2005

pour obtenir le titre de

Docteur en Sciences de l'Université Paris-Sud - Paris XI

(spécialité Informatique)

par

Nicolas Godzik

Composition du jury

Président :	Jean-Sylvain Liénard, *Directeur de Recherche CNRS, LIMSI*
Rapporteurs :	Cyril Fonlupt, *Professeur, Université du Littoral* Gilles Venturini, *Professeur, Université de Tours*
Examinateurs :	Michèle Sebag, *Directeur de Recherche CNRS, LRI* Olivier Sigaud, *Professeur, Université Paris 6*
Directeur :	Marc Schoenauer, *Directeur de Recherche INRIA Futur*

Laboratoire de Recherche en Informatique (LRI)

Résumé

L'architecture de subsomption de Brooks basée sur la modularité et sur une hiérarchie de comportements s'avéra être un grand pas vers les applications réelles après l'échec relatif des méthodes classiques. Cependant, l'inconvénient de cette méthode réside dans le fait que tout est mis en place par un concepteur humain, et plus la tâche du robot est ardue, plus la conception des modules nécessaires l'est aussi.

D'autre part, la robotique évolutionnaire, qui optimise par évolution artificielle des contrôleurs neuronaux (poids et éventuellement architecture), est une approche boîte noire par rapport aux comportements, mais requiert la définition d'une (ou plusieurs) fonction objectif dont la complexité va elle aussi croître avec la difficulté de la tâche demandée au robot.

L'approche des "contrôleurs symboliques" et des "superviseurs" proposée dans cette thèse tente d'établir un compromis entre les deux précédentes. Les sorties ne sont plus des ordres moteurs élémentaires mais des comportements plus évolués (comme "aller tout droit", "tourner à gauche", ...). Le choix peut également être effectué avec une librairie de comportements de plus haut niveau (comme "explorer", "aller vers la lumière", ...). Cette approche a été validée expérimentalement dans une expérience d'exploration, ou bien encore de recharge d'énergie.

Le second axe de recherche de cette thèse s'intéresse à l'apprentissage dans le cadre de la robotique évolutionnaire. Le baldwinisme qui consiste à modifier le génotype pendant la vie du robot (pendant l'évaluation des individus), mais qui reste sans conséquence sur le génotype lui-même pour la suite de l'évolution artificielle, est étudié ici à travers diverses expériences.

Les réseaux dits "auto-teaching", qui permettent au robot de modifier son comportement durant sa vie en s'appuyant sur un réseau modèle lui aussi obtenu par évolution, sont étudiés ici avec attention. Des défauts patents de ses propriétés de généralisation ont été mis en évidence.

Une architecture baptisée Action, Anticipation, Adaptation (AAA) a été créée pour tenter de pallier ces défauts. Cette architecture neuronale a donc trois tâches à remplir : l'action (contrôler les moteurs du robot), l'anticipation (basée sur les perceptions du robot mais également sur ses actions moteurs) et l'adaptation (basée sur la différence entre les sensations prédites au temps t et les sensations réellement observées au temps t+1). Les erreurs sur la prédiction des futures sensations du robot sont utilisées pour modifier en ligne les poids du contrôleur.

L'analyse détaillée de cette architecture d'une part a montré des capacités de généralisation plus robustes que celles des réseaux auto-teaching, et d'autre part a mis en évidence

quelques capacités d'adaptation.

Mots-clés: Robotique autonome, algorithmes évolutionnaires, contrôleurs neuronaux, robotique évolutionnaire, mécanisme d'anticipation, superviseur symbolique

Abstract

In this thesis, we propose some original approaches to autonomous robotics that are somehow intermediate between Brooks' subsumption architecture and the pure black-box Evolutionary Robotics approach. Both those approaches badly scale up with the complexity of the task of the robot.

The "symbolic controllers" use some user-defined high level behaviors as output of the controller (e.g. move forward, turn left, ...). The "supervisors" simply choose one behavior from a library of possible behaviors, that can either be user-defined or be the result of previous evolutions (e.g. explore, go to the light, ...). This approach has been validated on a homing experiment with battery recharge.

Finally, we studied the coupling between evolution and life-time learning for the robot. The "auto-teaching" architecture, proposed by Nolfi and Parisi, has been thoroughly tested, and some clear defects of its generalization properties have been demonstrated. The AAA architecture (Action, Anticipation, Adaptation) was designed to remedy those defects : It has some additional outputs that are predictions of the sensor values at next time step ; it uses the errors made by this anticipatory part, when compared to its actual sensor values, to on-line modify the weights of the controller. Some promising results have been obtained after tuning some parameters of the model, even though some long-term weakness still remains. In particular, this architecture showed some small adaptation capabilities.

Keywords: Autonomous robotics, Evolutionnary algorithm, neural network controller, evolutionnary robotics, anticipation, symbolic supervisor

Remerciements

Je tiens à remercier chaleureusement tout d'abord mon Directeur de Thèse, Marc Schoenauer, avec qui j'ai pu découvrir le monde de la recherche. Son optimisme et sa capacité à voir le bon côté des choses en toutes cirsconstances m'ont été particulièrement utiles, son aide fréquente sur les points difficiles et délicats m'a permis d'avancer plus sereinement et de mener à bien cette thèse. Je le remercie pour la confiance qu'il m'a témoigné.

Je suis très reconnaissant à Cyril Fonlupt et Gilles Venturini qui ont accepté d'être les rapporteurs de ma thèse. Je remercie aussi Olivier Sigaud pour sa participation au jury lors de ma soutenance. Enfin, je tiens à remercier tout particulièrement les deux derniers membres du jury que je connais plus pour les avoir cotoyés notamment au sein du projet Robea : tout d'abord Michèle Sebag, pour son intérêt constant et ses apports permanents tout au long de ce travail, bien au-delà du projet Robea lui-même, et même si j'ai manqué du temps suffisant pour pouvoir mettre en pratique la majorité de ses suggestions ; et enfin Jean Sylvain Lienard, le président de mon jury, dont la grande connaissance des Sciences Cognitives m'a permis de sortir du cadre purement Robotique de ma thèse et m'a inspiré de nouvelles expériences.

Je remercie maitenant les personnes que j'ai pu cotoyer à l'INRIA Rocquencourt où j'ai passé les deux premières années de ma thèse et au LRI à Orsay. Je suis reconnaissant tout particulièrement à Yann Semet de m'avoir toujours soutenu et d'avoir participé toujours avec plaisir à la finalisation de ce document jusqu'à très tard le soir.

Je tiens également à remercier grandement Nicolas Bredèche, que j'ai rencontré à mon arrivée au LRI, et qui s'est intéressé immédiatement à mon travail, j'ai très apprecié les discussions que nous avons pu avoir ensemble qui m'ont permis d'avancer plus efficacement dans la rédaction de mon manuscrit et la préparation de ma soutenance.

Je tiens à remercier aussi les personnes avec qui j'ai également passé de bons moments dans les laboratoires, notamment Amine Boumaza, Anne Auger, et Matthieu Pierres et tout particulièrement Olga Roudenko et Mary Felkin, entre autres, pour être venues assister un dimanche à la préparation de ma soutenance, dont les remarques ont été très pertinentes et m'ont permis d'améliorer la qualité de certains transparents.

Enfin, je remercie tout particulièrement Corinne, qui a fait preuve de beaucoup de patience durant cette thèse, et de compréhension durant les moments de pression et je la remercie également pour avoir relu mon manuscrit à la recherche des fautes d'orthographe qui étaient encore présentes.

Table des matières

Chapitre 1

Introduction

1.1 L'intelligence Artificielle et la Robotique

Tout au long du 20ème siècle, avec notamment l'apparition des premiers ordinateurs dans les années 40, l'intelligence artificielle a été l'objet d'un engouement grandissant de la part des scientifiques. De très nombreux chercheurs rêvèrent alors à la possibilité de simuler l'intelligence humaine. Avec des ordinateurs capables de jouer aux échecs et capables de battre des joueurs humains à de nombreux autres jeux, la fascination a été grandissante. Ainsi, on se met à rêver d'un ordinateur doté d'une intelligence comparable à celle d'un humain.

Karel Kapek, un écrivain et journaliste tchécoslovaque, inventa le mot "robot", dans sa pièce de théâtre R.U.R (Rossum's Universal Robots), qui fut créée à Prague au début de l'année 1921, et dans laquelle des robots en viennent à se révolter contre leurs concepteurs. Cette pièce connut un tel succès dans son pays, qu'elle s'expatria à travers toute l'Europe.

C'est en 1951 que Raymond Goertz du CEA en France construit le premier robot téléguidé : il s'agissait en fait d'un bras manipulateur constitué principalement de courroies et de poulies. Trois ans plus tard l'américain George Devol conçoit le premier robot programmable. En 1956, lors d'une rencontre avec Joe Engelberger, ils décidèrent de fonder la première société qui fabriquerait des robots : Unimation (de "Universal Automation"). Les ordinateurs devaient constituer en fait l'intelligence du robot (bras manipulateur à l'époque) alors que les moteurs en étaient la force musculaire.

Parallèlement, depuis une trentaine d'année l'intelligence artificielle a fasciné de très nombreux écrivains et réalisateurs de cinéma. Parmi les oeuvres portées à l'écran, on peut penser notamment à plusieurs films devenus "cultes" comme le film "War Games" sorti en 1983, où un ordinateur programmé par le professeur Falken, doué d'une intelligence

capable de communiquer, manque de peu de déclencher la troisième guerre mondiale. Dans ce film on constate que Joshua (l'ordinateur vedette du film) n'est pas seulement constitué d'une base de connaissances, mais possède également la capacité d'apprendre. Parmi les autres robots célèbres, on peut citer notamment, le très mèdiatique robot R2D2, issu du film "Star Wars", ou bien encore les robots Vincent et Bob du film "le trou noir".

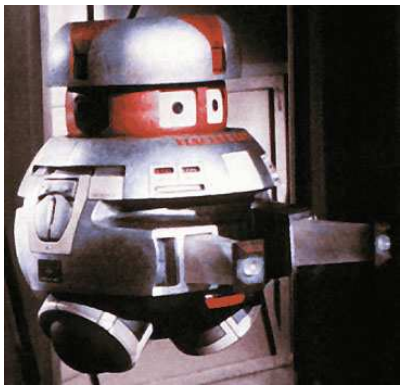


FIG. 1.1 – Le robot Vincent du film "le trou noir" de Walt Disney en 1979.

Avec la naissance des ordinateurs et leur capacité grandissante de calcul, un androïde (un robot ressemblant à un humain) semblait tout a fait réalisable. En effet, en guise de vue, une caméra pouvait bien suffire, un micro pouvait représenter l'ouïe du robot, un haut-parleur, la parole, à charge aux concepteurs de programmer le microprocesseur pour contrôler le tout. Au commencement de l'IA, le but était d'atteindre une intelligence similaire à celle d'un être humain, comme dans le film "Blade Runner" sorti aux USA en 1982, où seuls des tests très poussés permettent de différencier les robots des êtres humains.

L'intelligence symbolique fit son apparition à la fin des années 50 avec notamment MacCarthy du MIT. Cette approche consiste à manipuler des symboles, des concepts abstraits, par exemple, une voiture ou bien une porte. La culture du sens commun était elle aussi très importante, par exemple "le conducteur conduit la voiture" mais pas l'inverse, ou bien encore "la pluie tombe vers le bas". Les chercheurs de l'époque clamaient que finalement le plus dur était d'introduire tous ces symboles abstraits ainsi que ce qui les reliait aux autres symboles, et finalement le fait de reconnaître une porte pour un robot ne serait pas une tâche très ardue si les connaissances disponibles étaient présentes. Les mécanismes de déduction semblaient la partie facile de l'apprentissage par les machines. Le tentative la plus impressionnante dans cette direction est le projet CYC de Doug Lenat ([Lenat et Guha, 1990], `http ://www.cyc.com/`, voir aussi

`http ://en.wikipedia.org/wiki/Douglas_Lenat`) qui veut rassembler le plus important corpus de sens commun jamais vu dans une machine pour enfin atteindre un niveau qui permettra à la machine de continuer toute seule son apprentissage par tous les moyens dont elle pourra disposer (livres, Internet, ...).

Toutefois, lors d'expérimentations simples comme l'ouverture d'une porte par un robot, la difficulté apparut brusquement, l'ancrage du symbole "ouvrir" est en fait d'une extrême difficulté lorsque celui-ci est basculé dans le monde réel, reconnaître une voiture devient également extrêmement délicat (notons d'ailleurs qu'il s'agit d'une question à laquelle il est encore difficile de répondre aujourd'hui).

C'est au cours des années 80 que s'effectua progressivement le basculement vers une intelligence beaucoup plus "en situation". Les problèmes sont alors pris à la racine, la vision devient ascendante, les considérations de bas niveau, aussi peu "nobles" que le bruit sur les capteurs, doivent entrer en ligne de compte. R. Brooks [Brooks, 1986, Brooks, 1990] ressort aujourd'hui comme étant le précurseur de ce tournant dans l'intelligence artificielle nommée "l'intelligence artificielle située".

La conception d'un contrôleur selon Brooks, fait partie de l'approche "robotique basée sur les comportements" (*Behavior-based Robotics*) : toute la conception doit être réalisée par un expérimentateur humain qui doit procéder de manière très méthodique ; les divers comportements du robot sont ajoutés graduellement (tel par exemple l'évitement d'obstacles), et doivent être testés minutieusement afin de voir s'ils interagissent convenablement avec les comportements déjà implémentés.

1.2 Robotique Evolutionnaire

Le renouveau du domaine des réseaux de neurones artificiels, après l' "hiver" consécutif à l'article de Minsky et Papert [Hornik *et al.*, 1989], a été un autre tournant pour le contrôle en robotique. Leur identification en tant qu' "approximateurs universels", puis l'apparition de méthodes d'apprentissage des poids (e.g. la rétro-propagation du gradient [Rumelhart *et al.*, 1986]) montrant la faisabilité pratique de l'utilisation de réseaux de neurones pour la régression numérique, ont ouvert la voie à de très nombreuses applications, et ont suscité l'intérêt des roboticiens. Cependant, pour faire fonctionner les algorithmes d'apprentissage à base de gradient, des exemples donnant les sorties du contrôleur pour un grand nombre de jeux d'entrée sont nécessaires, ce qui semble incompatible avec l'idée d'un robot autonome, surtout lorsque l'on veut parler d'adaptation. Les approches de rétro-propagation "à travers le temps" [Nguyen et Widrow, 1990] ne permettent de résoudre que certains types de problèmes, et demandent de construire un modèle neuronal de la dynamique du robot, source supplémentaire d'erreur. Et, si les réseaux de neurones mis au point à l'aide des algorithmes d'apprentissage à base de gra-

dient ont connu un grand succès dans le domaine du contrôle [W. T. Miller *et al.*, 1990], leur utilisation en robotique autonome n'a pu réellement convaincre qu'avec l'utilisation d'autres algorithmes d'apprentissage permettant de s'affranchir de la nécessité d'avoir des exemples, tels les algorithmes évolutionnaires [Eiben et Smith, 2003] : la robotique évolutionnaire était née [Mondada et Floreano, 1995, Nolfi et Floreano, 2000].

Dans les années 60, plusieurs auteurs ont imaginé indépendamment des algorithmes inspirés de l'évolution des espèces de Charles Darwin [Darwin, 1859] et de la recombinaison génétique de Gregor Mendel : John Holland [Holland, 1975] invente les algorithmes génétiques, Larry Fogel la Programmation Evolutionnaire [Fogel *et al.*, 1966], et Ingo Rechenberg et Hans-Paul Schwefel les Stratégies d'Evolution [Rechenberg, 1972, Schwefel, 1981]. Après ici aussi un "hiver" de plus de 15 ans, c'est vers la fin des années 80 que le domaine re-naquit de ses cendres, avec en particulier l'ouvrage devenu séminal de David Goldberg [Goldberg, 1989]. Celui-ci, outre qu'il montra l'applicabilité concrète des algorithmes génétiques à des problèmes réels, essaya également de comprendre plus en détail cette nouvelle technique d'optimisation. Aujourd'hui, les algorithmes évolutionnaires sont une des techniques utilisées pour résoudre les problèmes d'optimisation difficiles [Eiben et Smith, 2003].

Le principe de base de cette technique, inspirée donc de la théorie de l'évolution naturelle, consiste à faire évoluer une population d"'individus". Chaque individu code une solution du problème à résoudre, et l'algorithme consiste principalement à faire subir des variations aléatoires aux individus de la population (les "variations aveugles" de la théorie de Darwin), et à sélectionner ensuite les individus les plus performants (imitant la "sélection naturelle" selon Darwin). La population est par conséquent en perpétuel mouvement et de génération en génération elle se dirige en général vers des zones de l'espace de recherche contenant des "bonnes" solutions au problème.

Cette méthode a été appliquée au début des années 1990 pour résoudre des problèmes de robotique, les premiers à s'aventurer dans cette nouvelle branche de la Robotique furent Dario Floreano et F. Mondada [Mondada et Floreano, 1995, Floreano et Mondada, 1994b]. En France, c'est Jean Arcady Meyer [Meyer et Guillot, 1991] qui popularisa cette approche de la robotique.

Les algorithmes évolutionnaires sont en effet particulièrement efficaces pour optimiser des contrôleurs type réseaux de neurones. Il s'agissait, aux débuts de la robotique évolutionnaire, de mettre en place une architecture fixe de réseaux de neurones et de trouver un vecteur de poids adéquat permettant de réaliser l'expérience souhaitée – vecteur de poids optimisé par algorithme évolutionnaire, la fonction à optimiser étant une mesure de la qualité de la façon dont le contrôleur s'acquittait de la tâche assignée.

Cependant, le problème de cette approche réside dans le fait de passer à l'échelle pour des problèmes difficiles : il faut complexifier l'architecture du réseau de neurones, mais, de ce fait, le nombre de poids augmente, le problème d'optimisation peut devenir très difficile à résoudre en un temps raisonnable, et il devient quasi-impossible d'arriver à trouver les poids satisfaisants.

C'est dans ce contexte que se situent les travaux présentés dans cette thèse : nous proposons quelques pistes pour contourner ce problème de conplexité algorithmique auquel fait face la robotique évolutionnaire.

1.3 Nos contributions

Cette thèse se situe donc dans le cadre de l'intelligence artificielle située, et en particulier toutes les expériences envisagées doivent être réalisables sur des vrais robots. L'environnement n'est pas entièrement connu à l'avance, et un des critères de succès des approches envisagées est que la dégradation des performances des contrôleurs soit la plus lente possible lorsque l'environnement se modifie par rapport à l'environnement dans lequel a eu lieu la mise au point du contrôleur. Dans ce cadre, nous avons choisi comme modèles pour les contrôleurs les réseaux de neurones, qui permettent déjà de réagir beaucoup mieux au bruit environnant que de nombreux autres modèles. Enfin, ne possédant pas d'exemples d'apprentissage, nous avons utilisé les algorithmes évolutionnaires afin d'ajuster ces contrôleurs neuronaux.

Mais au delà de l'inspiration directe des travaux de robotique évolutionnaire déja cités, la robotique basée sur les comportements de Brooks a également beaucoup inspiré notre approche : bien que très performante, cette approche exige que l'experimentateur procède de manière très méthodique, et ceci demande un temps relativement conséquent pour implanter et tester le bon fonctionnement du robot. C'est ainsi que dans notre première contribution, nous nous sommes appuyés sur cette modularité comportementale, tout en gardant à l'esprit les intérêts opportunistes d'une évolution artificielle. Dans ce cadre, nous avons proposés des "superviseurs" neuronaux, permettant, dans le cadre d'une approche élargie de la robotique basée sur les comportements, de décharger l'utilisateur de la tâche du choix du bon comportement, et/ou des liens entre les divers comportements : les superviseurs neuronaux ont justement pour but de choisir les comportements nécessaires à chaque instant de la vie du robot, en fonction de la tâche que le robot aura à accomplir.

Mais une autre composante essentielle d'un contrôleur adaptatif est la possibilté d'apprentissage en ligne : nous avons ainsi dans un deuxième temps rajouté cette capacité d'apprentissage pendant la vie du robot, qui devient ainsi capable, pendant sa vie, de modifier son comportement face à un environnement changeant, différent éventuellement de ceux rencontrés durant l'évolution. Et ce mécanisme d'adaptation dont nous avons

choisi de doter le robot est basé sur la notion d'anticipation. L'anticipation est une idée extrêmement puissante pour mesurer notre connaissance du monde qui nous entoure. L'anticipation peut être totalement passive ("je sens qu'il va pleuvoir", "je sens qu'il bluffe !") ou au contraire basée sur des actions ("si je rentre dans cette pièce, je devrais voir tel ou tel objet"). La notion de contingences sensori-motrices [O'Regan et Noë, 2001] nous permet de prédire nos futures sensations (exprimées par nos cinq sens) lorsque nous effectuons une action. Inspiré par ces contingences sensori-motrices, nous avons doté le contrôleur neuronal du robot, d'un module qui essaye de prédire les futures sensations du robot. Les erreurs d'anticipation sont alors un signal que l'environnement a au moins partiellement changé, et l'algorithme de rétro-propagation du gradient, partant de cette erreur, est alors utilisé afin de modifier les poids du réseau, en ligne.

1.4 Guide de Lecture

Les deux premiers chapitres (2, 3) de ce rapport présenteront un état de l'art (non exhaustif) dans le domaine de l'intelligence artificielle. Nous avons présenté dans le chapitre 2 quelques approches avant tout utilisées en robotique, soit dans un monde réel, soit dans un mondre discret :

- La section 2.1 présente les approches de l'intelligence artificielle dites classiques, il s'agit des premières techniques d'intelligence que les chercheurs ont mises au point.
- La section 2.2 relate l'approche beaucoup plus pratique et proche de la réalité des robots avec l'architecture de subsomption de Brooks.
- La section 2.3 explique les origines de l'apprentissage par renforcement, et décrit l'algorithme du Q-learning (section 2.3.1) qui en est un des algorithmes les plus utilisés.
- Enfin, les systèmes de classeurs sont décrits dans la section 2.4, dans laquelle une description sommaire notamment des sytèmes XCS, ACS est donnée.

Ensuite, le chapitre 3 présente les différentes techniques utilisées en robotique évolutionnaire :

- Les réseaux de neurones artificiels sont présentés dans la section 3.1, ainsi que l'algorithme d'apprentissage le plus populaire pour l'apprentissage des poids d'une topologie fixe dans le cas où des exemples sont disponibles, l'algorithme de rétro-propagation du gradient (section 3.1.3), qui nous sera utile dans le chapitre 6.
- La section 3.2 présente les origines et les différentes techniques utilisées dans les algorithmes évolutionnaires.
- La combinaison des deux techniques, réseaux de neurones et évolution, constituant la robotique évolutionnaire, est décrite dans la section 3.3. Les différentes possibilités de couplage des précédentes techniques sont présentées.

– L'évolution artificielle nécessite l'évaluation des individus afin de déterminer ceux qui sont aptes à traverser les générations. Les différentes possibilités de construire la fonction performance (fitness) sont discutées dans la section 3.4.

Les environnements logiciels et matériels que nous avons utilisés durant cette thèse sont décrits dans la section 4 : les robots Khepera, ainsi que leurs équipements, puis les particularités des différents simulateurs utilisés durant le déroulement de ces travaux.

Les deux derniers chapitres (5, 6), présentent les résultats originaux obtenus pendant cette thèse. Le premier d'entre eux (5) présente nos travaux sur l'introduction et l'utilisation de la modularité en robotique :

– La section 5.2, présente en détail le concept de contrôleur modulaire développé par S. Nolfi et D. Floreano [Nolfi et Floreano, 2000].
– Les contrôleurs symboliques sont présentés dans la section 5.3, une expérience simple d'évitement d'obstacles met en évidence les possibilités de ce type de contrôleur.
– Les superviseurs sont décrits dans la section 5.5 : une étude poussée sur une expérience d'évitement d'obstacles avec nécessité de recharge d'une batterie virtuelle, est effectuée, et les résultats montrent une bonne efficacité et une grande souplesse de ce type de contrôleur dans l'utilisation des comportements mis à sa disposition.
– Enfin, la section 5.7 présente une expérience effectuée sur un robot réel, avec l'utilisation des superviseurs.

Le dernier chapitre est consacré à l'utilisation de l'apprentissage dans la robotique évolutionnaire, via l'adjonction au contrôleur classique d'un mécanisme d'anticipation :

– L'expérience réalisé par Tuci, Harvey et Quinn est tout d'abord présentée dans la section 6.2, afin de montrer un exemple relativement impressionnant d'apprentissage en robotique évolutionnaire.
– Le réseau Autoteaching mis au point par Floreano et Nolfi est décrit dans la section 6.3. En particulier, nous avons refait de nombreux essais avec ce type de réseau pour étudier plus en détail ses capacités d'apprentissage, et en particulier sa stabilité sur le long terme.
– Une architecture relativement novatrice (AAA) est décrite dans la section 6.4. Cette architecture est étudiée en détail sur une expérience simple, dans la section 6.5, et laisse entrevoir de grandes capacités de passage à l'échelle.

Enfin, la conclusion (chapitre 7) résume nos contributions, et présente un ensemble de perspectives ouvertes par nos résultats.

Pour finir, les annexes A et B décrivent en détails les résultats obtenus à travers l'utilisation de l'architecture AAA. Ces résultats sont présentés sous formes de graphiques et de schémas explicites, qui permettent au lecteur désirant approfondir les résultats d'en effectuer une lecture rapide et efficace.

Chapitre 2

Intelligence artificielle et robotique

Qu'est ce que l'intelligence artificielle ? Il n'existe pas de réponse unanimement acceptée à cette question. Cependant, toute réponse est intimement liée à l'apparition des ordinateurs au cours du 20ème siècle, et en particulier à l'augmentation régulière et continue des puissances de calcul disponibles. Il existe de très nombreuses techniques dites d'intelligence artificielle, des techniques aujourd'hui considérées comme plutôt classiques (voir section 2.1), que l'on oppose souvent à d'autres techniques plus récentes (cf. sections 2.2, 2.3, 2.4).

Ce chapitre ne prétend pas parcourir exhaustivement toutes les techniques se disant relever de l'Intelligence Artificielle, mais va plutôt décrire certaines d'entre elles qui ont marqué (et marquent souvent encore) leur époque, des techniques qui ont été très utilisées et le sont encore aujourd'hui.

2.1 L'intelligence artificielle classique

Depuis les années 50, les chercheurs en informatique ont essayé de déterminer ce qui pourrait faire qu'un programme se rapproche de la cognition humaine, et ont progressivement opté pour une représentation très linguistique de notre monde. Les entités qui nous entourent représentant aussi bien les objets que les personnes, ainsi que les actions associées ont été symbolisées par des mots et des concepts abstraits.

Cependant, une simple liste de symboles sans lien entre eux n'a pas grand intérêt, c'est pourquoi la nécessité de relier ces différents symboles s'est évidemment imposée. Une culture du sens commun est apparue, des notions claires et sans ambiguïté du type "la pluie tombe vers le bas" ou bien "une voiture est conduite par un conducteur" en faisaient partie. Le projet CYC de Lenat et Guha de 1984 à 1990 [Lenat et Guha, 1990] a représenté la quintescence de cette approche : on a tenté de donner une immense culture de sens commun à l'ordinateur en entrant à la main de très nombreuses connaissances, puis

en se fondant sur des mécanismes de déduction génériques pour augmenter graduellement et automatiquement ce corpus de connaissances. L'idée était qu'après quelques années, le système deviendrait capable d'apprendre tout seul à partir de textes bruts, et que rapidement la masse des connaissances qu'il apprendrait par lui-même dépasserait tout ce qui aurait pu lui être donné par les experts humains. Ce projet n'a évidemment jamais réellement abouti, se heurtant à plusieurs barrières. Certes, CYC, et tous les systèmes de ce type, bien que ne possédant pas de signification réelle pour les différents symboles qu'ils manipulaient, étaient capables de raisonner à partir de ces symboles, de déduire par un processus logique de nouvelles conclusions à partir des hypothèses proposées. Mais il n'existe pas (ou du moins pas dans l'état actuel des connaissances humaines) de *représentation* des connaissances universelles – pas même les "frames" de CYC. Ceci en particulier limite aussi les possibilités de raisonnement universel.

La représentation des connaissances a été et reste un des grands sujets de recherche en intelligence artificielle et de nombreuses personnes travaillent encore dessus : comment classer les différents symboles en mémoire ? Comment faire pour que les mécanismes de déduction soient les plus performants possibles et se rapprochent de celui de l'être humain ? Ces techniques d'intelligence artificielle sont appelées intelligence artificielle symbolique – néanmoins, on utilise aussi fréquemment aujourd'hui le terme d'*IA classique*, car ce fut une des premières manifestations de l'IA, et même, pendant de nombreuses années, une des seules approches envisagées.

En parallèle, la théorie du contrôle en Robotique a commencé à émerger avec l'apparition des premiers robots programmables dans les années 50. Il s'agissait alors principalement de bras manipulateurs constitués de courroies et de poulies. La théorie du contrôle consiste à définir les équations de la trajectoire que le robot (le bras) doit suivre très précisemment. Dans le cas d'écarts constatés par rapport à la trajectoire idéale par les capteurs lors du parcours effectif (tout en considérant les bruits provenant de l'environnement et susceptibles de modifier les valeurs des capteurs), des ordres moteurs précis sont calculés puis envoyés aux moteurs. Les résultats théoriques garantissent alors la stablité de la trajectoire (et l'optimalité des commandes ainsi calculées). Cette technique de contrôle est très efficace et robuste, et ne pose aucun problème pour les bras manipulateurs notamment. Cependant, elle nécessite une connaissance parfaite du monde dans lequel évolue le robot : c'est effectivement le cas pour un bras manipulateur, mais cela est évidemment la négation même du cas d'un robot autonome se déplaçant en milieu partiellement inconnu, cas pour lequel un contrôle fixe décidé a priori ne peut qu'amener à l'échec.

Dans le cadre de l'intelligence artificielle symbolique, toutes les expériences qui ont été menées de façon concrète (sur robot réel) ou de façon abstraite (uniquement sur la manipulation de symboles) ont été faites suivant le principe de la figure 2.1. Le raisonne-

ment sur les symboles, imposé par l'approche classique, contraint le robot à distinguer par l'intermédiaire de ses capteurs les éléments de l'environnement qui l'entourent. L'identification de ces éléments, ainsi que leur position, constituent la modélisation de cet environnement dont le robot possède alors une représentation symbolique abstraite. Une fois cette modélisation effectuée, le robot va concevoir, par l'intermédiaire d'algorithmes de déduction logique, les différentes étapes nécessaires à l'obtention de son but : c'est l'étape de planification. Ensuite, les moteurs sont commandés afin d'exécuter les différentes tâches ainsi planifiées.

Décrivons par un exemple ce que pourraient être ces différentes étapes pour un robot qui devrait parcourir les bureaux d'une entreprise le soir afin de vider les poubelles dans un container qu'il transporte derrière lui :

- ***la perception*** : dans un premier temps, grâce à ses capteurs, le robot va percevoir son environnement (par l'intermédiaire de capteurs infrarouges, d'un sonar, ou d'une caméra par exemple).

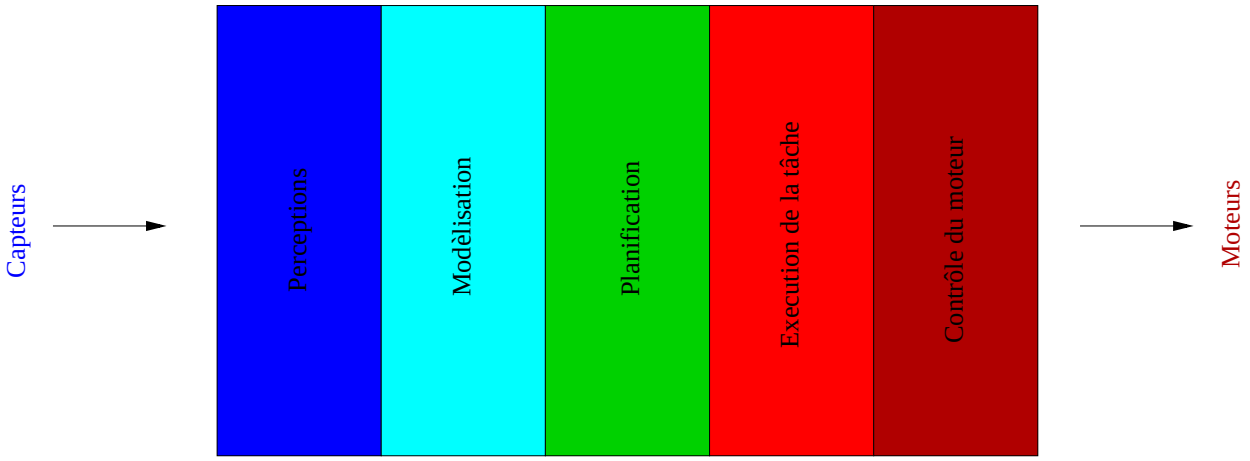


FIG. 2.1 – Manière traditionnelle de procéder en intelligence classique.

- ***la modélisation*** : en fonction des données recueillies par les capteurs, le robot va devoir essayer de reconnaître les éléments de l'environnement. Des images obtenues par caméra vont être analysées pour tenter de reconnaître des entités telles que des portes, des murs, des bureaux, des êtres humains, ... sans oublier bien sûr de repérer les poubelles qui constituent l'objectif principal de sa tâche. Celles-ci, ne doivent bien sûr pas être confondues avec d'éventuels cartons qui se trouveraient par terre, contenant des objets qu'il ne faudrait pas jeter. Une fois cette modélisation de l'environnement terminée, l'étape suivante peut être enclenchée.
- ***la planification*** : la planification se rapporte directement au but du robot voulu par le concepteur. Dans le cadre de cette expérience, le but va être de ramasser les poubelles. Lors de la précédente étape, le robot s'est situé théoriquement dans son

environnement, par exemple au milieu de deux murs parallèles rapprochés, correspondant à un couloir. La tâche de planification, résultant de développement logique sur la modélisation symbolique de l'environnement, proposera par exemple de se diriger vers la porte la plus proche, d'entrer dans le bureau, puis de se diriger vers la poubelle et enfin la vider si elle n'est pas déjà vide.

- ***l'exécution de la tâche*** : désormais, les moyens doivent être mis en oeuvre pour exécuter les différentes tâches prévues par la planification : il faut actionner les roues en précisant la direction à prendre pour se diriger vers le bureau, ou bien encore prévoir la séquence d'actions nécessaires afin d'ouvrir la porte si celle-ci se trouve être fermée. Notons que ces séquences d'actions sont codées au préalable par le concepteur du robot.
- ***le contrôle des moteurs*** : les moteurs correspondant à l'exécution de la tâche courante sont activés, un contrôle de suivi de la trajectoire prévue lors de l'étape précédente peut être activé par exemple pour qu'un bras à 4 axes, avec une pince en bout, atteigne la poignée dans de bonnes conditions et actionnent le pène. Dans ce cas là, les techniques classiques du contrôle en robotique sont utilisées.

En 1980, le philosophe Searle critique fortement cette méthode de conception symbolique de l'intelligence artificielle à travers l'exemple de la chambre chinoise [Searle, 1980]. Il imagine qu'il possède un livre rempli de séries de symboles chinois, représentant l'ensemble des questions possibles écrites en chinois avec un nombre donné de caractères, avec, en face de chaque question, les réponses associées, elles aussi écrites en chinois. Searle ne parle pas le chinois, et n'a aucune idée de ce que peuvent représenter ces symboles. Une personne va alors lui poser une série de questions écrites à la fois en anglais et en chinois. Searle répond bien entendu sans problème aux questions en anglais, et parcours le livre à la recherche des symboles chinois qui lui sont proposés, puis recopie la réponse associée. En adoptant cette stratégie, il répond aussi bien aux questions en anglais qu'à celles en chinois. Searle a désiré montrer à travers cette expérience que bien qu'il ne comprenait absolument rien à ce qu'il pouvait écrire en chinois, les réponses étaient tout de même correctes. Il a désiré montrer ainsi que le fait de travailler sur des symboles en ayant l'air intelligent ne signifie pas forcément être intelligent. Cependant, pour les partisans de l'intelligence artificielle classique, bien que Searle ne comprenne pas du tout le chinois, le système [Searle + le livre], lui, le comprend, puisqu'il est capable de répondre de manière sensée : il est donc intelligent. Des discussions farouches ont encore lieu aujourd'hui entre les partisans de l'intelligence artificielle symbolique et ses adversaires.

Il ne faudrait pas cependant passer sous silence le fait que l'intelligence artificielle classique a tout de même permis d'obtenir des résultats intéressants, tant au niveau recherche que dans plusieurs domaines d'application, notamment grâce aux systèmes experts. Les systèmes experts furent extrêmement en vogue de 1970 à 1990, et ont démontré des ca-

pacités de raisonnement certaines, à partir d'hypothèses et de règles de production, par exemple dans le domaine du diagnostic. Cependant, dès que l'on s'intéresse au domaine plus spécifique de la robotique, la critique majeure que l'on peut adresser à l'IA symbolique est que les chercheurs ont travaillé exclusivement sur les symboles en négligeant l'aspect réel de la robotique. En effet, les techniques issues de l'IA classique en robotique ont consisté à essayer de modéliser le monde réel le mieux possible, mais sans travailler directement sur lui. La planification est alors effectuée sur un monde idéal, où tous les objets entourant le robot ont été clairement identifiés, et en l'absence de tout bruit.

Lors de certaines expériences menées sur des robots réels, les chercheurs ont tenté de faire en sorte que tous ces symboles abstraits pour le robot deviennent concrets. Cependant, l'ancrage des symboles dans le monde réel est un travail particulièrement rude. Durant les prémisses de l'intelligence artificielle dans les années 50, les chercheurs pensaient qu'il serait facile de reconnaître un objet à partir d'un stock d'images, il suffisait pour cela de stocker différents types d'image comme un ours en peluche, une table, ... Cependant, ce ne fut pas aussi facile, car ces objets, lorsqu'ils sont dans le monde réel, sont orientés d'une certaine façon, soumis à des éclairements différents qui penvent ajouter des ombres, etc, et cette voie a rapidement été abandonnée, d'autant plus que le stockage de millions d'images aurait demandé un espace mémoire considérable.

Mais quels sont les éléments essentiels qui peuvent permettre aux capteurs dont dispose le robot de reconnaître tel ou tel objet ? Cette question a été posée maintes et maintes fois par de nombreux chercheurs sans recevoir une réponse valable. Prenons l'exemple concret d'une chaise, quels sont les éléments qui distinguent une chaise ? 4 pieds avec un dossier permettent de distinguer une chaise, mais un tabouret qui ne comporte pas de dossier ne doit t-il-pas être classé comme une chaise ? Ou bien encore une chaise monobloc qui serait née d'une injection plastique ? Comment différencier une chaise sans dossier d'une table ? toutes ces questions rejoignent directement la représentation des connaissances, doit-on considérer la chaise comme un objet avec ses propres attributs ? le tabouret doit il faire partie de la catégorie chaise ? ou doit on utiliser une autre représentation ? Ces questions ne sont pas encore tranchées aujourd'hui.

Outre ces difficultés de représentation, l'ancrage très difficile des symboles, mais également le temps calcul nécessaire au robot pour modéliser son environnement et exécuter la planification, ont poussé de plus en plus de chercheurs à critiquer cette approche symbolique de l'intelligence robotique et de nouvelles techniques sont apparues avec notamment l'architecture de subsomption de Brooks dans les années 80.

2.2 L'architecture de subsomption de Brooks

Rodney Brooks a écrit de nombreuses critiques [Brooks, 1990] de l'approche classique de l'IA en robotique : pour lui il n'est pas nécessaire (et il peut même être nuisible) de tenter de modéliser le monde. En effet, le robot fait partie du monde réel et va évoluer dans ce monde. Par conséquent, son comportement, loin d'être basé sur une approche symbolique abstraite, doit au contraire rester concrètement en liaison directe avec le monde réel grâce aux moyens dont il dispose (capteurs + actionneurs). Tous les éléments physiques doivent être pris en considération, comme le bruit sur les capteurs qui ne sont pas parfaits, ou bien encore le fait que les roues peuvent à tout moment patiner.

Contrairement aux techniques classiques, où le contrôleur est totalement centralisé, l'approche de Brooks s'inspire d'une nouvelle voie biologique, où le contrôle des tâches ne s'effectue pas de façon centrale mais où au contraire chaque zone du cerveau joue un rôle bien spécifique. Ainsi, certaines expériences réalisées sur des personnes accidentées ont même montré que certains malades ayant perdu l'usage de certaines zones du cerveau sont par exemple capables de nommer un objet, sans pour autant savoir quelle est sa fonction. Brooks se situe dans cette optique, à travers une approche comportementale dans laquelle les décisions ne sont plus prises de façon centrale, mais où au contraire l'ensemble des comportements du robot fonctionnent en parallèle.

Les conditions pour la robotique que Brooks pense être primordiales et qu'il désire satisfaire sont les suivantes :

- ***les buts multiples*** : le robot se retrouve assez fréquemment dans des situations où plusieurs buts possibles s'offrent à lui ; des buts de haut niveau (e.g. l'accession à une certaine zone de l'environnement) sont en concurrences avec des procédures de plus bas niveau (e.g. l'évitement d'obstacles, la nécessité de recharge des batteries).
- ***l'inexactitude des capteurs*** : tous les capteurs commettent une erreur lors de l'acquisition de leurs données, et le robot doit prendre en compte ces erreurs (inconnues) pour décider de son comportement.
- ***la robustesse*** : lorsqu'un événement inattendu se produit, comme l'arrêt du fonctionnement d'un capteur, ou bien un problème moteur, le robot doit adopter un comportement propice à la situation et ne doit en aucun cas rester bloqué.
- ***l'additivité*** : la puissance de calcul mise à disposition du robot doit être proportionnelle au nombre de comportements exigés du robot ; de même, chaque fois que de nouveaux capteurs sont ajoutés, une puissance supplémentaire est nécessaire, afin que le robot ne perde pas de temps.

Le résultat est l'architecture de subsomption [Brooks, 1986], qui diffère totalement des techniques de l'IA classique qui l'ont précédée. C'est le monde réel qui est mis en avant, au détriment des procédures abstraites de haut niveaux. L'approche adoptée est

une approche *bottom-up* : la conception du contrôleur global du robot est entreprise en partant des plus bas niveaux, qui assurent des comportements élémentaires, comme par exemple l'évitement d'obstacles ; puis de façon incrémentale, le concepteur va progressivement construire des comportements de niveaux de plus en plus haut, s'occupant de buts de plus en plus complexes, s'appuyant sur les comportements déjà existants aux niveaux inférieurs. Concrètement, l'architecture de subsomption est une architecture en couches, le concepteur crée d'abord la couche de plus bas niveau, puis s'appuie progressivement sur les couches déjà construites pour réaliser de nouvelles couches, représentant des actions de plus haut niveau.

Chaque couche est constituée de plusieurs modules, chaque module est totalement indépendant et reçoit en entrée aussi bien les données des capteurs, que des buts, ou bien encore des sorties provenant d'autres modules. Chacun d'entre eux est constitué d'un automate à états finis qui en fonction des entrées reçues proposera une sortie (ou n'en proposera pas).

Afin de mieux cerner l'architecture de subsomption, prenons l'exemple d'une tâche devant être réalisée par le robot Khépéra (décrit dans la section 4). La mission devant être accomplie par le robot consiste à se diriger vers une source lumineuse afin de saisir un objet grâce à son *gripper* (bras manipulateur). Un exemple d'architecture de subsomption pourrait être celui représenté sur la figure 2.2.

Afin que le lecteur puisse discerner plus facilement les différentes couches mises en oeuvre, celles-ci sont représentées avec des couleurs différentes : le vert représente les entrées sensorielles en provenance du robot, les différentes actions motorisées destinées aux roues ainsi qu'au bras manipulateur sont en rouge. Examinons maintenant de plus près les différentes couches :

- ***la couche 0 - bleu clair*** : cette couche est la couche de plus bas niveau, et son seul but est l'évitement d'obstacles. Les deux modules *gestion des chocs* et *calcul de la force de répulsion* reçoivent en entrée les valeurs des capteurs infrarouges en mode actif (percevant la distance approximative d'un éventuel obstacle, voir chapitre 4). La force de répulsion calculée est inversement proportionnelle à la distance du robot à d'éventuels obstacles. Cette force de répulsion est ensuite communiquée au module *évitement primaire* qui va déterminer les ordres moteurs nécessaires à l'évitement d'obstacles. Ces commandes moteurs sont ensuite transmises aux moteurs contrôlant les roues. Le module *gestion de chocs* quant à lui prend le relais, lorsque, malgré *l'évitement primaire*, le robot rentre en collision avec un obstacle (obstacle vraiment très proche d'un des capteurs, ou en contact) : une commande aléatoire est envoyée au moteur jusqu'à ce que le robot soit débloqué (i.e. suffisamment éloigné de l'obstacle).
- ***la couche 1 - jaune*** : le module *évitement d'obstacles et direction vers la lumière* prend en entrée les capteurs infrarouges du robot Khépéra permettant de déterminer

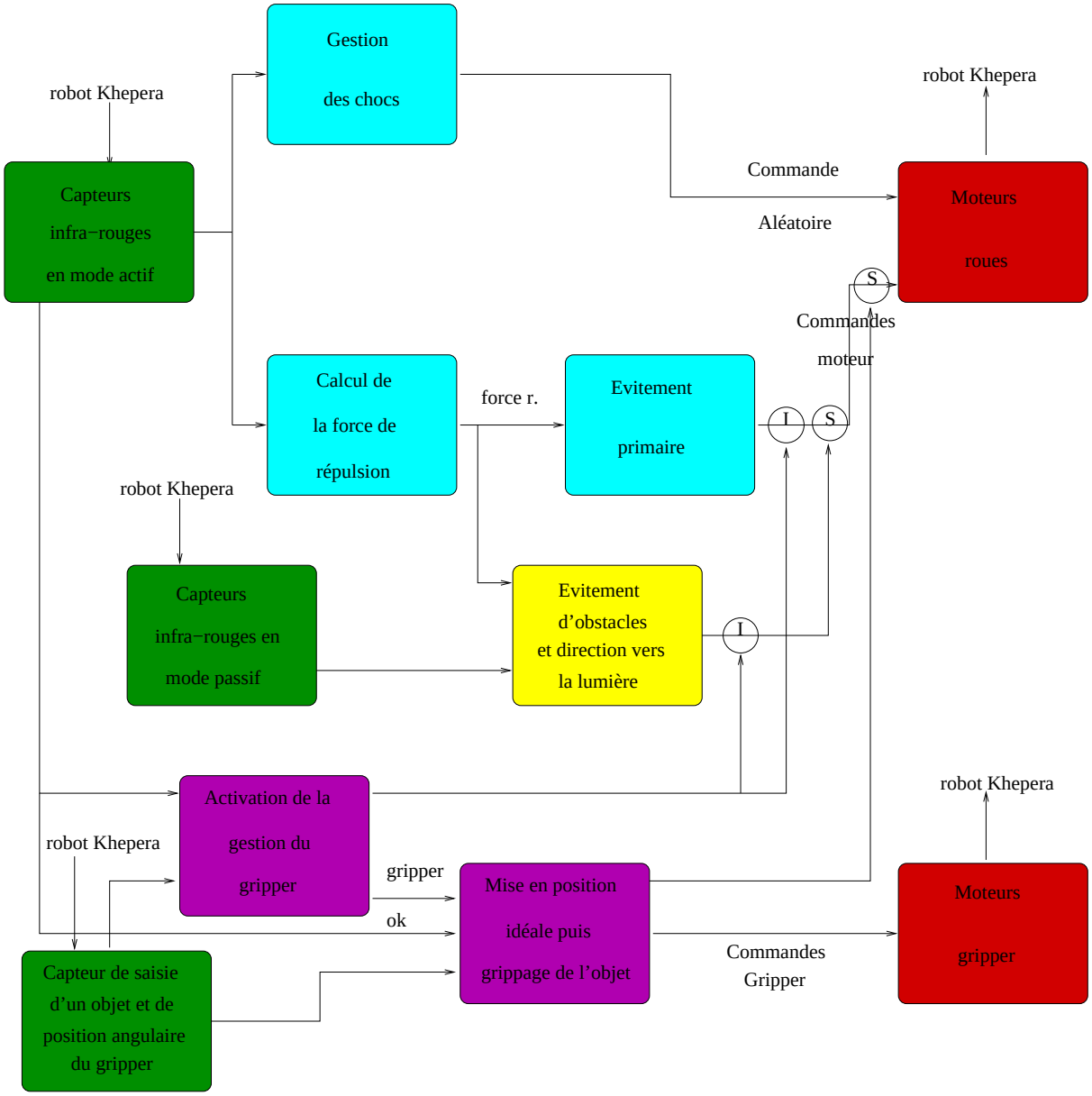


FIG. 2.2 – Un exemple d'architecture de subsomption de Brooks, cet architecture permet au robot d'aller saisir un objet situé près d'une lumière tout en évitant les obstacles.

la direction de provenance de la source lumineuse (voir ici le encore chapitre 4), ainsi que la force de répulsion venant du module *calcul de la force de répulsion* de la couche 0. Le module ici présent devra déterminer les ordres moteurs adéquats afin de permettre au robot un évitement convenable des obstacles, tout en essayant de se diriger vers la source lumineuse. Il faut noter que dans ce cas là *l'évitement primaire* de la couche 0 n'est plus effectif, sa sortie est remplacée par celle du module de la couche 1 (le "S" reliant l'entrée des moteurs indique la suppression de la sortie du module *évitement primaire* et son remplacement par la sortie du module *évitement d'obstacles et direction vers la lumière*).

- ***la couche 2 - violet*** : l'objectif de cette couche est tout simplement de vérifier les capteurs infrarouges du robot et lorsque l'analyse de ces capteurs semble signifier la présence à proximité d'un cylindre, la gestion du gripper doit être activée par l'intermédiaire de la variable "gripper ok". Notons que lorsque *l'activation de la gestion du gripper* est activée, deux ordres d'inhibition sont expédiés aux sorties des modules *évitement primaire* ainsi que le module *évitement d'obstacles et direction vers la lumière*, en effet, dans ce cas là, la gestion du gripper prend le pas sur les autres couches et le robot ne doit plus essayer d'éviter les obstacles mais au contraire de se rapprocher du cylindre correctement identifié. Des ordres sont ensuite envoyés aux moteurs du gripper en provenance du module *mise en position idéale* puis *grippage de l'objet*.

L'architecture de Brooks est résolument modulaire. De plus, comme il s'agit d'une architecture fonctionnant de façon asynchrone, à travers des modules indépendants, Brooks précise qu'un processeur peut être affecté à chacun d'entre eux, réduisant considérablement les temps de calcul dans le cas de tâches, d'environnement ou de robot très complexe.

L'architecture de subsomption rompt donc radicalement avec les techniques d'intelligence artificielle classique, et les premiers essais sur robot réel ont littéralement subjugué les chercheurs de l'époque, en permettant aux robots de Brooks d'effectuer avec rapidité et précision des tâches que les techniques classiques mettaient un temps laborieux à exécuter. Cependant, la discussion s'est alors focalisée sur l'appartenance ou non de ces travaux au domaine de l'intelligence artificielle. En effet, toute la conception est issue d'un développement humain, les couches nouvellement implantées sont testées minutieusement avant d'être connectées aux autres, et si, en apparence, le robot possède un comportement intelligent il a été en fait entièrement programmé par un ou plusieurs concepteurs humains. Les détracteurs de Brooks sur ce terrain ont comparé son architecture avec l'expérience de la chambre chinoise de Searle ...

2.3 L'apprentissage par renforcement

L'apprentissage par renforcement est une méthode d'apprentissage non supervisé, c'est-à-dire pour laquelle on ne dispose pas d'exemples d'actions avec leur conséquence immédiate quantifiée, mais seulement des récompenses correspondant à des suites d'actions – récompenses donc éventuellement différées par rapport à l'action elle-même. Ce type d'apprentissage tire également son inspiration de la biologie, et plus précisément de l'apprentissage des animaux. Parmi les travaux les plus connus sur l'apprentissage des animaux, citons ceux du célèbre biologiste Pavlov [Pavlov, 1927], et en particulier son expérience qui mit en évidence les capacités d'apprentissage des chiens. La procédure expérimentale consistait à secouer une clochette avant de donner à manger à un chien. Après plusieurs jours de ce traitement, il constata que le chien se mettait à saliver dès que la clochette retentissait, alors que la nourriture ne lui avait pas encore été servie : il avait appris à associer à la clochette l'imminente apparition de la nourriture. C'est à partir de ces expériences que les scientifiques ont commencé à parler de renforcements, exprimés ici par la nourriture.

Bien entendu, l'apparition d'algorithmes informatiques et de tests systématiques effectués en utilisant une notion de renforcement n'apparaîtra que bien plus tard. L'algorithme d'apprentissage par renforcement est un algorithme qui consiste à apprendre par une suite d'essais-erreurs : en fonction des actions tentées, le couple (sensation, action) est plus ou moins rétribué suivant ses conséquences par rapport à la tâche à réaliser. Les actions sont choisies de manière stochastique. Ainsi, au départ, toutes les actions possibles ont la même probabilité : l'exploration de l'espace des actions est privilégiée, afin d'expérimenter de nombreuses possibilités. Les actions pour lesquelles les récompenses acquises sont positives voient leurs probabilités augmenter, et auront ainsi plus de chance d'être sélectionnées par la suite. Sutton, un des pionnier de l'apprentissage par renforcement, précise dans l'ouvrage de référence [Sutton et Barto, 1998] que les chercheurs en réseaux de neurones comme Rosenblatt, Widrow et Hoff, étaient très motivés par l'apprentissage par renforcement, malgré le fait que le contexte de l'apprentissage des réseaux de neurones dans lequel ils se trouvaient en général était en fait de l'aprentissage supervisé : par exemple, dans des domaines comme la reconnaissance de formes, on dispose d'un ensemble de formes pour lesquelles on connait la classe, ce qui permet d'utiliser des algorithmes d'apprentissage utilisant directement l'erreur résultant de chaque "action", comme par exemple la rétro-propagation de l'erreur (voir section 3.1.3).

D'après Sutton, le travail qui a eu le plus d'influence en apprentissage par renforcement fut celui de Donald Michie qui, de 1961 à 1963, réalisa un système basé sur les essais et les erreurs pour apprendre à jouer au "tic-tac-toe" (Morpions 3x3) [Michie, 1961, Michie, 1963]. Il nomma son système d'apprentissage MENACE (pour

Matchbox Educable Naughts and Crosses Engine). Le principe de cet algorithme est le suivant : pour chaque position de jeu (configuration précise de croix et de ronds sur l'échiquier 3x3), on définit un nombre de boites d'allumettes égal aux différentes positions suivantes de jeux. Le nombre de perles dans chaque boite d'allumettes est proportionnel à l'efficacité que pourrait présenter le coup auquel elle correspond.Au début de la partie, toutes les boites ont autant d'allumettes. A la fin de la partie, suivant que l'issue a été favorable ou non, des perles vont être rajoutées ou retirées dans les boites d'allumettes correspondant à l'ensemble des coups joués durant cette partie.

Par la suite, en 1968, Michie et Chambers décrivirent un autre système d'apprentissage par renforcement appelé GLEE (*Game Learning Expectimaxing Engine*) couplé au contrôleur BOXES [Michie et Chambers, 1968]. Il parvinrent, grâce à ces deux systèmes, à résoudre le problème du pendule inversé, c'est-à-dire à contrôler un pendule inversé afin qu'il reste stable le plus longtemps possible. Sutton précise dans son livre [Sutton et Barto, 1998] que c'est cet ensemble de travaux qui l'influença pour ses travaux ultérieurs [Barto *et al.*, 1983, Sutton, 1984]. Ces expériences de Michie constituent pour lui, un des meilleurs exemples des débuts de l'apprentissage par renforcement. Sutton considère également qu'un autre des pionniers de l'apprentissage par renforcement est Harry Klopf ([Klopf, 1972, Klopf, 1975, Klopf, 1982]), dont les idées ont notamment permis de comprendre la différence fondamentale existant entre l'apprentissage supervisé et l'apprentissage par renforcement.

Nous allons maintenant détailler l'algorithme d'apprentissage par renforcement le plus connu et sans doute le plus utilisé en robotique, le Q-learning.

2.3.1 Le Q-learning

Le Q-learning a été développé par Watkins en 1989 [Watkins, 1989]. Dans cet algorithme, à l'instar de ce qu'a fait Michie avec ses petites boîtes d'alumettes, il s'agit de stocker, pour chaque situation possible (e.g. présence d'un obstacle à gauche du robot, réception d'un message particulier, ...) les différentes qualités associées à l'exécution de chaque action possible (e.g. aller tout droit, tourner de 90 degrés à droite, envoyer un message ...). Un exemple d'un tel tableau est illustré sur la figure 2.3. Au commencement de l'expérience, toutes les composantes de ce tableau sont initialisées à zéro : pour une situation donnée, il n'y a pas de raison logique pour effectuer une action plutôt qu'une autre, et on va procéder de manière aléatoire par des choix uniformes.

L'algorithme relatif au Q-learning est illustré sur la figure 2.4. On suppose donc qu'il existe un tableau $Q(s, a)$ donnant la qualité espérée de l'action a effectuée dans la situation s. Il y a quatre phases principales dans le Q-learning :

– Le robot se situe dans une situation donnée s_t, il s'agit de déterminer l'action a^* à accomplir qui maximise la récompense espérée, c'est-à-dire telle que $Q(s_t,a^*)$ soit

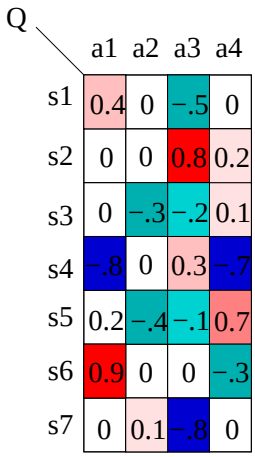


FIG. 2.3 – Un exemple de table de qualité pour le Q-learning pour les différents couples (sensation, action).

maximale. Pour cela, il suffit de regarder dans la table Q (voir par exemple figure 2.3). Cependant, que ce soit pour régler les cas d'indétermination (par exemple, au commencement de la phase d'apprentissage, tous les éléments de cette table sont initialisés à zéro) ou pour conserver un minimum d'exploration et éviter qu'une des actions ne soit systématiquement choisie trop tôt dans le processus d'apprentissage, la procédure de choix de l'action possède une composante stochastique. Au fur et à mesure que l'apprentissage progresse, l'importance de cette partie stochastique s'estompe, ce qui permet alors une répétabilité des choix et renforce les séries d'actions menant aux plus fortes récompenses.

– Le robot va alors exécuter l'action choisie à l'étape précédente, et se retrouver dans la situation s_{t+1}. En fonction de cette nouvelle situation, ainsi que de la nouvelle situation du monde (e.g. cylindre attrapé, but de marqué dans un match de robot-soccer, ...) , un renforcement r défini au préalable dans une fonction de renforcement par l'utilisateur (e.g. +1 si le robot a marqué un but, -1 si le robot a poussé la balle en dehors des cages de but et 0 sinon) est attribué à l'action entreprise. Il est important de noter que ce renforcement peut être nul s'il est impossible d'estimer l'amélioration de la qualité de la situation s_{t+1} par rapport à la situation s_t.
– La qualité est alors mise à jour de la manière suivante :
$Q(s_t, a) = Q(s_t, a) + \beta(r + \gamma.max_a(Q(s_{t+1}, a') - Q(s_t, a)))$,
terme qui représente l'amélioration maximale de la qualité que l'on peut espérer dans la situation s_{t+1}. Les paramètres β et γ sont des constantes dans $]0, 1]$ définies par l'utilisateur.
– Enfin, la mise à jour de la table de qualité est effectuée et on revient à l'étape 1.

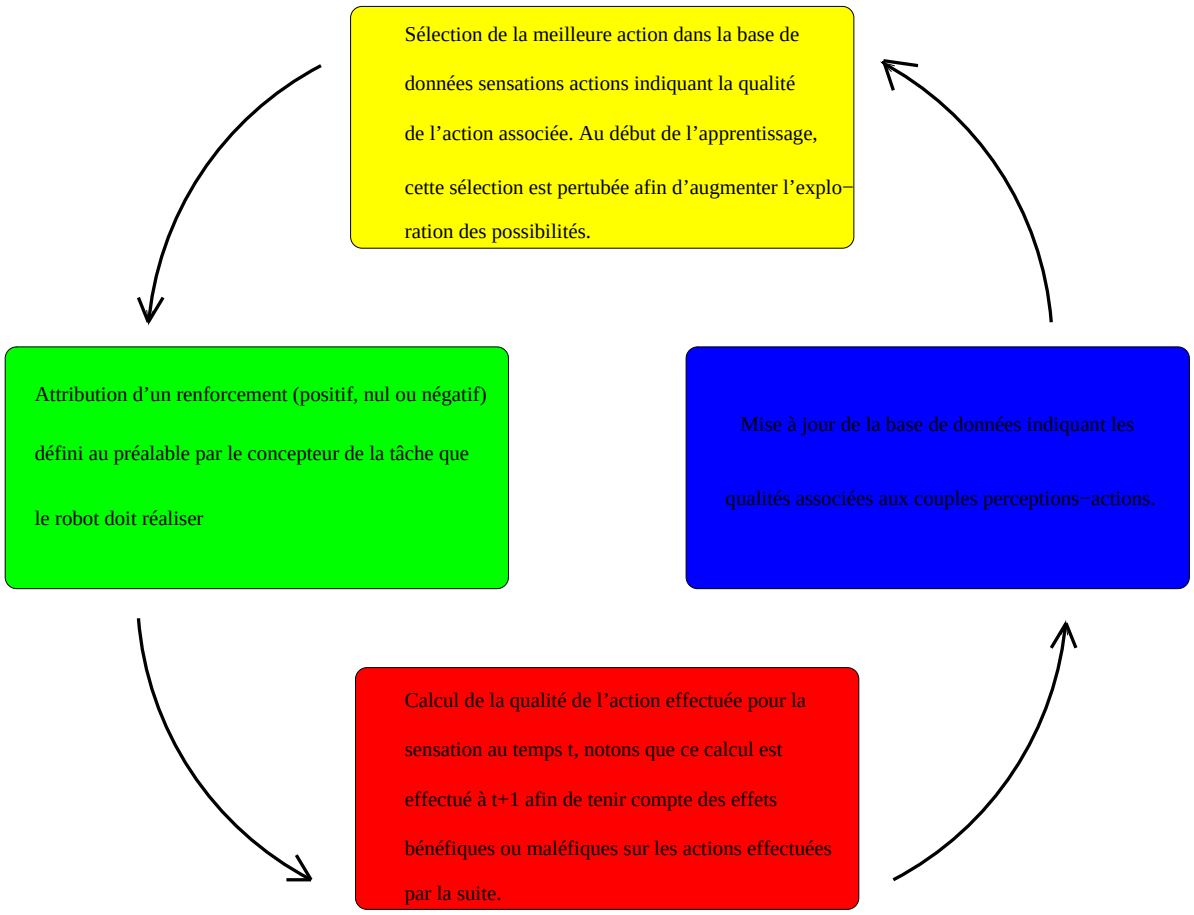


FIG. 2.4 – Les différentes étapes de l'algorithme du Q-learning dans l'apprentissage par renforcement.

L'apprentissage par renforcement est une technique très utilisée aujourd'hui, malgré des défauts connus. D'une part, comme il a été dit plus haut, il est courant que peu de situations permettent d'obtenir un renforcement non nul. Pour reprendre l'exemple du robot-soccer, si on ne donne de récompense qu'aux actions entraînant un but ou une sortie de balle, il est probable que le système n'évoluera jamais : lorsque toutes les valeurs de Q sont nulles (situation initiale), aucune mise à jour n'en est faite tant que le renforement r reste nul. Seule l'action ayant alors entraîné le but ou la sortie de balle verra sa qualité changer. Lors de la phase de jeu suivante, les actions menant à cette action de qualité non nulle verront à leur tour leur qualité changer, et ainsi de suite : de très nombreuses phases de jeu seront nécessaire avant que ne se dégagent des suites d'actions répétables de par leurs qualités positives (ou leur absence de qualités négatives, sans doute, dans le cadre du robot-soccer).

Un autre problème majeur de l'apprentissage par renforcement est le fait que toutes

les expériences doivent être décrites dans un monde discretisé, alors que les capteurs du robot, et d'une manière générale le monde dans lequel il évolue, prennent des valeurs réelles. Une discrétisation du monde est alors soit très grossière, soit entraîne rapidement une explosion de la taille de la table de qualité, bien que des travaux récents aient proposé plusieurs méthodes de discrétisation adaptatives pour pallier cet inconvénient [Munos et Moore, 1999], voire l'utilisation de fonctions de qualité continues [Munos, 2000]. De plus, le problème du bruit, crucial en robotique, n'est pas explicitement pris en compte par la technique du Q-learning.

2.4 Les systèmes de classeurs

Les systèmes de classeurs ont été introduits par John Holland en 1976 [Holland, 1975] en parallèle avec les algorithmes génétiques. Un classeur représente en fait une règle, constituée d'un couple (condition, action). Tout comme l'apprentissage par renforcement, ces systèmes sont utilisés dans un environnement idéal discrétisé, voire même binarisé : seules deux valeurs sont possibles pour l'ensemble des conditions ainsi que pour l'ensemble des actions. Ainsi les capteurs fonctionnent souvent en mode tout ou rien (présence d'un obstacle (1) ou non (0) sur une case adjacente), et les ordres moteurs permettent de se déplacer instantanément sur la case située devant le robot, ou sur une case immédiatement voisine (en 4-connexité). Nous allons voir plus loin que les techniques d'apprentissage utilisées sont également voisines de celles de l'apprentissage par renforcement.

Un exemple de classeur basique utilisé pour les robots est visible sur la figure 2.5. Dans cet exemple, le robot a 4 capteurs (dans l'ordre, par exemple, gauche, devant, droite, derrière), et la situation décrite dans la condition de la règle est qu'un obstacle est visible sur la droite du robot, et qu'il n'y a pas d'obstacle devant ni à gauche du robot. La partie action de cette règle est un virage à droite pour le robot (les moteurs étant dans l'ordre gauche, droite). La dernière partie de la règle est sa *force*, valant ici 0.2. Cette règle peut se traduire par "S'il y a un obstacle à droite, pas d'obstacle devant ni à gauche, et qu'il y ait ou non un obstacle derrière, tourne à gauche".

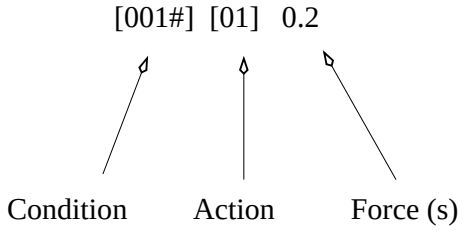


FIG. 2.5 – Un exemple de classeur.

2.4.1 Fonctionnement d'un système de classeurs

Le système maintient une liste de *messages* qui constitue la communication entre le monde (le robot) et le système de classeurs lui-même. Voici comment se déroule le traitement d'un système de classeurs à chaque cycle :

- le robot indique sur la liste de messages l'état de ses capteurs.
- les classeurs dont la partie condition correspond au message sont sélectionnés. Le *matching* est fait en considérant que 0 et 1 se correspondent exactement, et que le caractère "joker" # correspond indifféremment à 0 ou à 1.
- un tirage aléatoire similaire au tirage de la roulette des algorithmes évolutionnaires (cf figure 3.5) est effectué sur les forces, favorisant les classeurs ayant la plus grande force. L'action associée au classeur gagnant est expédiée sur la liste des messages.
- Le message, indiquant l'action moteur à accomplir, est récupéré par les actionneurs du robot qui exécutent les ordres moteurs.

Par rapport au Q-learning décrit section 2.3.1, remarquons qu'il n'est pas nécessaire que l'ensemble des classeurs décrive exactement l'ensemble des situations possibles dans les parties conditions : d'une part, il y a en général une "règle par défaut" qui est activée lorsqu'aucun classeur n'est sélectionné car les conditions courantes ne matchent pas la condition courante. D'autre part, l'utilisation du caractère joker permet d'avoir des règles plus ou moins générales, diminuant par conséquence la taille de la base de règles.

Par contre, les forces des systèmes de classeurs sont modifiées par un algorithme assez proche de l'algorithme de Q-learning (section 2.3.1), ainsi que l'a remarqué Dorigo en 94 [Dorigo et Bersini, 1994]. C'est l'algorithme dit de la *bucket brigade* [Holland, 1975] (terme qui peut être traduit par la "chaîne des seaux"). Il s'agit d'une technique de récompense différée. Chaque classeur symbolise un être humain, et l'objectif commun de l'ensemble des classeurs est de former une chaîne dans le but d'aller éteindre un incendie en se passant des seaux plus ou moins pleins d'eau. L'objectif est de faire parvenir la chaîne à un point d'eau qui constitue la récompense. Dans le système de classeur, cette récompense est progressivement rétro-propagée à travers toute la chaîne par l'intermédiaire d'offres (proportionnelles à leurs forces) émises par les classeurs. Les classeurs constituant des chaînes ne parvenant pas à obtenir de récompense de la part de l'environnement vont ainsi être progressivement écartés par une baisse graduelle de leur force.

2.4.2 L'évolution des systèmes de classeurs

Si le fait de pouvoir décrire des situations générales par un seul classeur, ou d'avoir relaché la contrainte de couverture de l'ensemble des situations possibles par le système de classeur permet de réaliser un gain substantiel en complexité du système lui-même, il est par contre nécessaire de prévoir des algorithmes permettant d'ajuster le nombre et la

généralité d'un système de classeurs. Compte-tenu de la nature de l'espace de recherche (chaînes de caractères 0, 1, ou #), les algorithmes génétiques, ou plus précisément des variantes adaptées des algorithmes génétiques) se sont naturellement imposés depuis les premiers travaux de J. Holland.

Une introduction plus complète sur les algorithmes évolutionnaires et génétiques est disponible dans la section 3.2. Le principe général est inspiré de la théorie de l'évolution de Darwin [Darwin, 1859] : une population d'individus est soumise aux deux facteurs de l'évolution que sont la sélection "naturelle" et les variations "aveugles". Les individus les plus adaptés survivent et se reproduisent via des transformations stochastiques.

Il existe deux variantes principales de l'évolution des systèmes de classeurs, qui diffèrent fondamentalement au niveau de la représentation non pas d'un classeur, mais du système complet. Un classeur (un génotype) est donc toujours constitué d'une chaîne de caractères (0, 1 ou #), éventuellement complétée de paramètres réels (force comme dans l'exemple précédent, justesse de l'anticipation dans les ACS présentés section 2.4.4, ...). Par contre, deux écoles s'affrontent en ce qui concerne la manière de manipuler un ensemble de classeurs constituant le système complet :

- **La représentation dite de *Pittsburgh*** : introduite par [Smith, 1980], elle considère qu'un individu est un systéme de classeurs complet (aussi appelé une base de règles). L'algorithme génétique permet de faire évoluer cette population de base de règles en appliquant des opérateurs de variation (croisement, mutation) entre bases. Le principal inconvénient de cette méthode est d'une part la complexité en taille (une population contient un grand nombre d'individus, chacun contenant un grand nombre de classeurs), et d'autre part la difficulté qu'il y a à introduire des opérateurs de variation spécifiques (tels que prendre le classeur le plus général entre 2 individus). Il en résulte, en général, une faible vitesse de convergence vers des solutions acceptables.
- ***La représentation dite de Michigan*** : introduite par [Holland et Reitman, 1978], cette approche manipule en fait une population de classeurs, et le système complet est la population totale (ou éventuellement un de ses sous-ensembles). La difficulté n'est plus alors dans la conception des opérateurs de variation (voir plus loin) mais dans l'attribution d'une fitness partielle à chaque classeur à partir des résultats du système complet. C'est là qu'intervient l'algorithme de la Bucket Brigade par exemple, utilisé pour créditer les classeurs de la force qui va ensuite servir à la sélection au sein de l'algorithme d'évolution.

2.4.3 Le système de classeurs XCS

La similitude algorithmique entre l'algorithme de Bucket Brigade et le Q-learning déjà notée, outre qu'elle rapproche considérablement l'approche système de classeur et appren-

tissage par renforcement, implique également que les deux algorithmes risquent de souffrir des mêmes problèmes. Ainsi, on retrouve dans les classeurs un des problèmes pratiques du Q-learning, à savoir le manque de rétribution des classeurs situés en début de chaîne (loin de la récompense). Or, lors de l'évolution de systèmes de classeurs avec une représentation de type Michigan, la sélection des classeurs se fait selon leur force, et ainsi les classeurs situés en bout de chaîne (proche de la récompense) vont être considérablement favorisés. Il peut alors rapidement devenir problématique de sélectionner des classeur suffisamment divers pour constituer des chaînes complètes.

Pour pallier ce défaut, les systèmes de classeurs XCS [S.W.Wilson, 1995] ne basent plus la sélection sur la contribution espérée à la récompense (la force des classeurs classiques décrits dans la section précédente) mais maintient trois composantes numériques en sus des parties condition et action : la récompense espérée R, une estimation de l'erreur moyenne absolue sur R, et une *fitness*, qui est une estimation de la précision relative de la récompense espérée R. C'est sur cette fitness que se fait ensuite la sélection lors de l'étape évolutionnaire. En d'autres termes, un classeur qui prédira convenablement sa récompense, même si celle-ci est faible, aura plus de chance d'être sélectionné qu'un classeur prédisant mal une grande récompense (ce qui n'était pas le cas dans le cadre des LCS classiques).

Lors du déroulement de l'algorithme XCS, au moment du choix d'une action, si plusieurs classeurs satisfont les conditions, alors le choix est effectué en fonction de la prédiction de la récompense de ce classeur, comme dans le cas des LCS classiques. Le déroulement s'effectue donc de manière classique, l'algorithme de la Bucket Brigade continue d'être utilisé pour mettre à jour les récompenses espérées, avec en sus la mise à jour des taux d'erreurs comparés des prédictions et des récompenses réelles, ainsi que de la fitness de chaque classeur. Grâce à cette technique, tous les classeurs peuvent être sélectionnés, aussi bien les classeurs de début de chaîne que ceux situés en fin.

Signalons enfin qu'il est possible d'avoir dans la partie action des coefficients d'une fonction linéaire des entrées, et qu'on peut ainsi résoudre avec des systèmes de classeurs XCS des problèmes de régression à valeurs réelles [Wilson, 2000].

2.4.4 Les systèmes de classeurs anticipatoires

Les ACS (Anticipatory Classifier Systems) ont été introduits par [Stolzmann, 1998]. Une des différences majeures avec les autres systèmes de classeurs se situe au niveau de la création de nouveaux classeurs : ce ne sont plus des algorithmes de type génétiques ou évolutionnaires qui sont utilisés, mais des heuristiques spécifiques. Le principe du système de classeur ACS est représenté sur la figure 2.6.

Dans ce nouveau type de classeur, l'accent est mis autant sur la capacité des classeurs à entraîner des récompenses pour le robot qu'à anticiper sur les futurs états de ses capteurs compte-tenu de l'action envisagée. Un terme binaire additionnel est ajouté après les parties

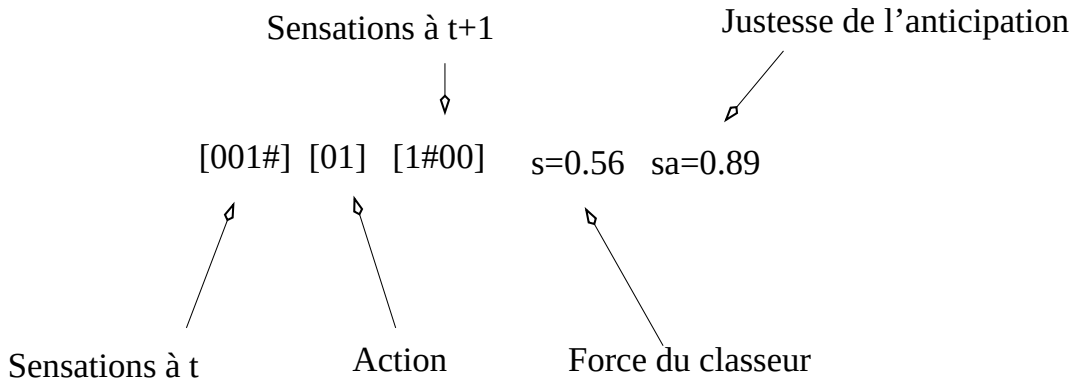


FIG. 2.6 – Le principe du classeur anticipatoire ACS

conditions et actions, qui a la même structure que le terme de condition puisqu'il tente de prédire quelles seront les sensations (conditions) au temps $t + 1$. Et au terme de force des classeurs LCS classiques, qui est ici encore la récompense espérée R, et représente toujours la qualité du classeur en vue de l'obtention d'une éventuelle récompense finale, est ajoutée une force A qui représente la capacité du classeur à anticiper correctement ses futures sensations.

Rappelons que le signe # dans S_t est le caractère "joker", indiquant que l'attribut en question peut prendre indifféremment la valeur 0 ou 1. Par contre, le caractère # dans la partie anticipatoire (les prédictions par le classeur pour S_{t+1}) signifie le "passage à travers" (*passthrough*) de l'attribut correspondant : sa valeur dans S_{t+1} reste inchangée par rapport à sa valeur dans S_t. Sur l'exemple de la figure 2.6, la valeur du deuxième attribut dans S_t est de 0, par conséquent, la valeur du deuxième attribut dans la prédiction de S_{t+1} représentée par # sera aussi 0.

La mise à jour des forces des classeurs est similaire à celle utilisée dans l'algorithme Bucket Brigade, avec une différence notable : si une récompense r est prévue à partir d'un endroit précis de l'environnement, cette récompense ne sera pas seulement rétro-propagée sur les classeurs qui ont permis d'atteindre cette récompense, mais elle augmentera également la force de tous les classeurs qui anticipent pour S_{t+1} une situation représentant le même état des capteurs que sur la case de récompense.

En ce qui concerne, la mise à jour de la force d'anticipation, les classeurs actifs sont testés sur le robot (en simulation), les sensations à t+1 réellement obtenues sont comparées aux sensations anticipées. Si les sensations correspondent, alors la force d'anticipation s_a est augmentée, si elles ne correspondent pas, on tente de créer un nouveau classeur possédant les bons paramètres d'anticipation (on constate donc ici que l'utilisation d'algorithmes évolutionnaires n'est plus nécessaire contrairement aux classeurs classiques ou de type XCS précédemment décrit). Cependant, si la création d'un tel classeur n'est pas

réalisable, alors la force d'anticipation du classeur étudié baisse.

Après avoir laissé l'algorithme évoluer un moment, certains des classeurs finissent par possèder une grande force d'anticipation. Lorsque cette force d'anticipation atteint une valeur dépassant un certain seuil (par exemple 0.9), le classeur est dénommé *classeur sûr*, indiquant que l'utilisation de classeur va conduire à partir des actions précisées dans le classeur dans plus de 90% des cas, aux sensations anticipées par ce classeur à l'instant d'avant. Indiquer une position finale que le robot doit atteindre (sensations désirées) suffit alors à ce dernier pour planifier le chemin qu'il va emprunter, puisqu'il suffit pour cela de construire la chaîne minimale de classeurs sûrs, à partir des sensations initiales du robot jusqu'aux sensations finales (le but).

Le type de classeurs apparaît donc comme idéal dans des expériences d'apprentissage latent. De nombreuses expériences d'apprentissage latent ont été menées par les biologistes, notamment avec des rats. Tolman, en 1948, a fait évoluer des rats dans un labyrinthe avec des couleurs différentes aux différentes extrémités. Après que le rat se soit promené dans le labyrinthe, sans but, on le place dans une case colorée avec une récompense (nourriture). Ensuite, lorsque le rat se retrouve au début du labyrinthe, il se dirige directement vers la case de la même couleur que celle ayant contenu de la nourriture. Le rat a donc emmagasiné une très bonne connaissance du monde, puisqu'il est capable d'aller directement dans la case colorée (représentant une sensation particulière). L'analogie avec les systèmes de classeurs ACS est flagrante : lorsque le robot explore son environnement, il fortifie au fur et à mesure sa connaissance du monde, qu'il est capable de mettre en valeur par la construction d'une chaîne de classeurs sûrs. Lorsque les environnements deviennent non markoviens (impossibilité de prédire correctement les sensations futures à plusieurs reprises, dûe à une ambiguïté dans la perception (deux cases différentes peuvent engendrer les mêmes sensations à l'instant t mais le fait d'exécuter la même action pour les deux, peut engendrer des sensations futures différentes.)), le formalisme ACS est capable d'utiliser des séquences d'actions permettant de se retrouver sur une situation non ambiguë.

Ces techniques d'apprentissage par renforcement, et par système de classeur, ont fait leur preuve dans les mondes discrets, cependant peu d'expérimentations ont été tentées sur des robots réels : le nombre de sensations disponibles explose avec la discrétisation, et les bruits qui apparaissent sur les capteurs perturbent l'apprentissage.

Dans cette thèse, bien que la majorité des expérimentations aient été effectuées en simulation, la simulation tend à modéliser aux mieux les contraintes engendrées par le passage au réel. C'est pourquoi nous avons utilisé la technique de la robotique évolutionnaire utilisant des techniques d'évolution artificielle et des réseaux de neurones artificiels.

2.5 Conclusion

Nous avons passé en revue dans ce chapitre les techniques récentes issues de l'Intelligence Artificielle utilisées pour le contrôle en robotique. Si l'architecture de subsomption de Brooks a constitué un grand progrès par rapport à l'utilisation des méthodes de l'IA symbolique, elle n'en demande pas moins un gros travail d'analyse et de mise au point de la part du programmeur, puisqu'il doit décomposer lui-même la tâche que le robot doit accomplir en un ensemble hiérarchisé de sous-tâches entre lesquelles il doit aussi fixer les dépendances. Enfin, la notion d'adaptativité, par rapport à des incertitudes ou des changements dans l'environnement, semble difficile à intégrer dans l'architecture de subsomption, dans la mesure où la hiérarchie entre les divers modules est complètement figée – c'est la base de l'architecture. Seuls des petits changements de comportement semblent possibles dans ce contexte.

Au contraire, les techniques utilisant l'apprentissage par renforcement modifient par définition le comportement du robot durant sa vie, en fonction des récompenses accumulées durant les étapes antérieures, et semblent donc à même d'une part d'atteindre l'adaptativité, d'autre part de court-circuiter l'étape de mise au point modulaire manuelle de l'architecture de subsomption (elles peuvent en particulier être appliquées directement à un contrôleur global de type réseau de neurones tels ceux décrits dans le chapitre suivant). Cependant, dans le cas du Q-learning appliqué à un réseau de neurones, comme dans le cas des systèmes de classeurs, les méthodes inspirées de l'apprentissage par renforcement souffrent de deux défauts : d'une part, le temps d'apprentissaage du robot est souvent très long, et en particulier il faut de très nombreux essais dans un enviromment donné avant d'obtenir un comportement raisonnable – et ceci n'est pas toujours souhaitable voire possible. D'autre part, ces approches nécessitent une discrétisation préalable de l'espace de recherche des contrôleurs, qui aboutit dans les cas réalistes à une explosion de l'espace des (états,actions) rendant son exploration difficile et (encore plus) coûteuse.

Mais une troisième voie existe, consistant à utiliser des techniques d'évolution artificielle, i.e. inspirées de l'évolution naturelle des populations biologiques selon Darwin, pour faire évoluer des contrôleurs basés sur des réseaux de neurones artificiels. L'utilisation des réseaux de neurones, approximateurs universels connus pour leurs bonnes capacités de généralisation, permet de s'affranchir de l'étape de hiérarchisation de l'ensemble des tâches à accomplir, mais aussi de prévoir des mécanismes spécifiques d'adaptation indispensables dans le cadre de la robotique autonome. Et l'utilisation de l'évolution artificielle, en découplant la mise au point du contrôleur, qui se déroule à l'échelle de temps de l'évolution, de l'apprentissage et de l'adaptation en-ligne, qui a elle lieu à une échelle de temps beaucoup plus petite, celle de la "vie" du robot, permet d'obtenir des robots immédiatement opérationnels - même si c'est au prix d'un temps de calcul hors-ligne par-

fois important. Ces techniques vont être présentées au chapitre suivant, et sont la base de travail de toutes les réalisations de cette thèse.

Chapitre 3

La robotique évolutionnaire

Ce chapitre introduit les notions essentielles de robotique évolutionnaire, l'approche de la robotique autonome basée d'une part sur les réseaux de neurones comme modèles de contrôleurs, d'autre part sur les techniques d'évolution artificielle comme outils de mise au point de ces contrôleurs.

Les réseaux de neurones artificiels (et même les perceptrons multi-couches à fonction d'activation sigmoïdale) sont connus pour être des approximateurs universels [Hornik *et al.*, 1989], c'est-à dire qu'ils sont capables d'approcher au plus près, avec suffisamment de neurones, toute fonction, en particulier, donc, la fonction donnant les valeurs des actuateurs du robot en fonction des sensations, ou entrées des capteurs. D'autre part, les réseaux de neurones sont également connus pour leur capacité à bien généraliser, c'est-à-dire qu'ils intègrent implicitement une notion de régularité de la fonction qu'ils apprennent : la fonction approximée est plutôt lisse, ce qui les rend aussi moins sensibles aux bruits sur les capteurs que d'autres modèles possibles pour les contrôleurs des robots.

Dans le cadre de l'apprentissage supervisé, la mise au point (souvent baptisée "entrainement") des réseaux de neurones est en général réalisée via l'utilisation de méthodes à base de gradient (telle la célèbre "rétro-propagation du gradient"). Cependant, dans le cadre de la robotique, il n'existe pas (ou très peu) d'exemples permettant d'utiliser l'erreur commise sur un ensemble d'entrées particulières du réseau, et l'utilisation de méthodes à base de gradient est impossible (on citera pour mémoire la méthodes appelée *backprop through time* [Nguyen et Widrow, 1990], qui consiste à "dérouler" le temps, mais nécessite d'une part un modèle neuronal de la dynamique du robot, et d'autre part ne peut concerner que quelques pas de temps au delà desquels il faut disposer de ce que devrait être l'état du réseau, deux conditions qui la rende inapplicable en pratique). C'est pourquoi, dans le cadre de la robotique autonome, les algorithmes évolutionnaires sont le plus souvent utilisés pour la mise au point des contrôleurs neuronaux.

Inspirés de la théorie de l'évolution des espèces de Darwin, les algorithmes

évolutionnaires peuvent travailler sur des espaces de recherche non classiques (ce qui permet par exemple de les utiliser pour optimiser à la fois l'architecture et les poids d'un réseau de neurones, bien que nous n'ayons pas profité de cette souplesse dans cette thèse, voir section 3.3.3). Ils ne nécessitent que de connaître les valeurs prises par la fonction à optimiser (algorithmes d'ordre 0), et ils sont par ailleurs très robustes pour échapper aux minima locaux. Ce sont ces deux dernières propriétés qui justifient à elles seules le choix de cette technique pour la mise au point de contrôleurs en robotique autonome adaptative.

Nous allons tout d'abord présenter dans ce chapitre les concepts de base des méthodes de réseaux de neurones artificiels, section 3.1, puis les outils d'optimisation que sont les algorithmes évolutionnaires, section 3.2. Nous discuterons ensuite de l'optimisation des réseaux de neurones utilisant les algorithmes évolutionnaires, justifiant en particulier notre choix de ne faire évoluer que les poids d'un réseau d'architecture fixe. Enfin, la dernière section sera consacrée à des considérations générales sur les fonctions performance en robotique évolutionnaire.

3.1 Les réseaux de neurones artificiels

3.1.1 Historique

Les premiers neurones artificiels ont été développés dans les années 40, par Mc Culloch et Pitts [McCulloch et Pitts, 1943]. Inspirés par les neurones de notre système nerveux, ils définirent un neurone comme une unité de calcul plus ou moins activée par les activités des autres neurones auxquels il est relié par des connexions – connexions et poids correspondant aux synapses et poids synaptiques de notre propre système cognitif. Ces connexions peuvent être excitatrices ou bien inhibitrices (par exemple, dans leur première conception, les neurones artificiels de Mc Culloch et Pitts ne sont activés que s'ils sont assez excités). Les expériences réalisées par Mc Culloch et Pitts ont démontré l'énorme potentiel contenu par ces neurones artificiels et leur capacité à apprendre semblait tout à fait impressionnante à l'époque, bien que les algorithmes d'apprentissage utilisés fussent relativement simples (règles linéaires).

En 1949, toujours inspiré par notre système cognitif, Donald Hebb, met au point l'apprentissage hebbien [Hebb, 1949]. La modification des valeurs du poids de chaque connexion est réalisée par le produit des valeurs d'activation des deux neurones (amont et aval) attachés à la cronnection.

Le premier véritable réseau ordonné de neurones vit le jour dans les années 50 avec le perceptron de Rosenblatt (cf fig. 3.1) : ce réseau est constitué de deux couches, la couche du bas représentant les entrées du réseau et la couche du haut représentant les sorties. Le perceptron multi-couches viendra un peu plus tard, les neurones sont alors organisés en

plusieurs couches, et, s'ils peuvent théoriquement réaliser des approximations de fonctions non linéaires, malgré de nombreux essais, les capacités d'apprentissage en environnement non-linéaires des réseaux de neurones restent inexistantes. C'est ainsi qu'en 1969, Minski et Papert [Minsky et Papert, 1969] prouvèrent qu'en l'état des connaissances des algorithmes d'apprentissage, le perceptron,même multi-couches, était totalement incapable d'apprendre des concepts non linéaires (en particulier la fonction XOR). Ce fut le début de l'*hiver* des réseaux de neurones, la communauté scientifique ne voyant pas d'intérêt à utiliser une technique aussi limitée.

Cette limitation d'usage prit fin avec la publication d'un nouvel algorithme d'apprentissage, la rétro-propagation du gradient (parfois aussi appelé rétropropagation de l'erreur) [Cun, 1987, Rumelhart et McClelland, 1986] (nous détaillerons plus en détails cet algorithme dans la section 3.1.3). Grâce à cet algorithme, à base de descente de gradient, l'apprentissage supervisé des réseaux de neurones permit au perceptron multi-couches de révéler enfin toutes ses possibilités. De nombreuses applications furent réalisés avec succès dans les années 1990, notamment en reconnaissance des formes (et plus spécifiquement des caractères). De nombreuses applications vinrent confirmer que la technique avait atteint l'âge adulte [Tesauro et Sejnowski, 1987]. Ainsi les réseaux de neurones sont depuis plusieurs années utilisés pour la lecture des codes postaux à la poste américaine.

3.1.2 Types d'architectures

Un neurone artificiel est donc une unité de calcul ayant plusieurs entrées et une sortie, calculée en fonction des entrées. Cette sortie est éventuellement dupliquée et envoyée en entrée de plusieurs autres neurones. Chaque connection entre deux neurones est munie d'un poids, qui module la force du signal entrant dans le neurone suivant. Les entrées de certains neurones proviennent directement de l'extérieur du réseau. Vu comme une boîte noire, le réseau constitue une fonction de $\mathbb{R}^d$ dans $\mathbb{R}^{d'}$, d étant le nombre d'entrées du réseau et d' le nombre de sorties.

La première leçon issue des succès applicatifs est que l'architecture choisie pour une application donnée des réseaux de neurones est un facteur de réussite important, avant même l'algorithme d'apprentissage des poids. Ainsi, le réseau de reconnaissance des chiffres de la poste américaine comporte plusieurs dizaines de milliers de neurones et de connexions, dont la mise au point a nécessité des centaines d'hommes-mois. On distingue deux grands types d'architecture, les réseaux à propagation directe (*feedforward*), et les réseaux récurrents, suivant la présence ou non de boucles dans le graphe de connection des neurones.

Classiquement, en robotique, la couche d'entrée représente les sensations obtenues par le robot à travers ses capteurs, comme par exemple différents pixels d'une image caméra, ou bien encore de valeurs de capteurs infrarouges. Cette information est ensuite propagée à travers les différentes couches, et la couche de sortie décide des actions motrices devant

être accomplies par le robot.

3.1.2.1 Les réseaux feedforward

Le réseau de neurones à propagation directe (voir une exemple figure 3.1) est appelé ainsi car, pour une assignation des valeurs des entrées du réseau, les sorties peuvent être calculées directement, en un seul passage, en calculant de proche en proche les activations et les sorties des neurones. L'illustration la plus commune de ce réseau est le perceptron multi-couches, dans lequel le graphe du réseau est organisé en couche, chaque neurone d'une couche étant relié à tous les neurones de la couche suivante.

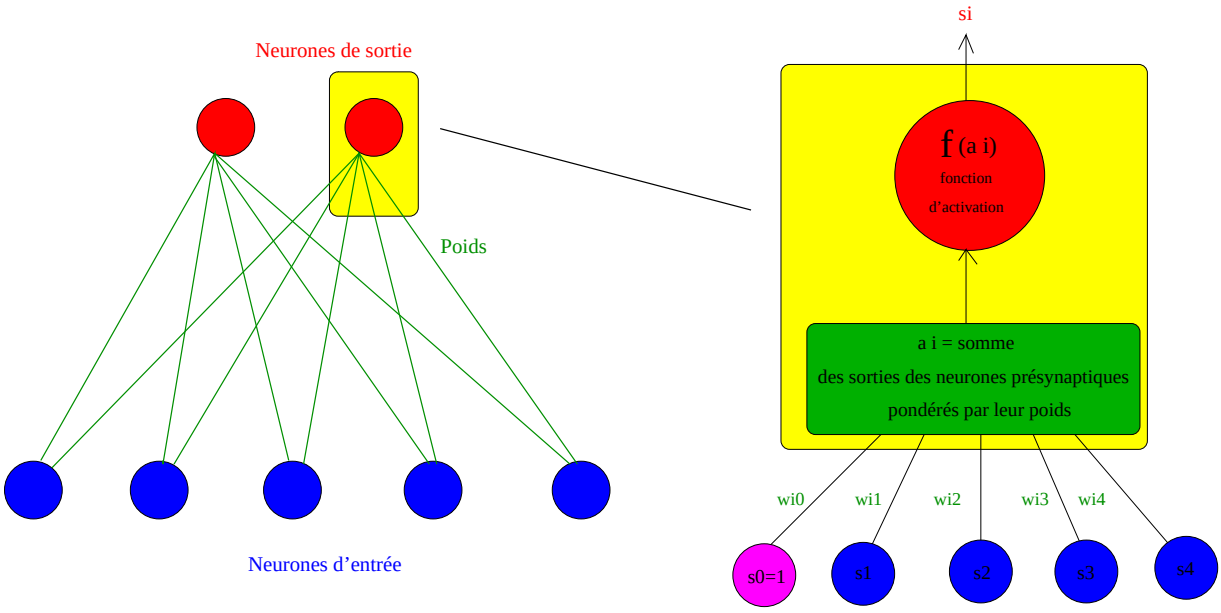


FIG. 3.1 – Sur la gauche, un exemple de réseau à propagation directe, le perceptron de Rosenblatt, à droite, le calcul de la sortie d'un neurone.

Rentrons un peu plus dans les détails d'un neurone pour comprendre comment le mécanisme d'activation des neurones est effectué. Deux neurones sont reliés par un synapse muni d'un poids, synapse reliant la sortie du premier neurone, aussi appelé neurone pré-synaptique, à l'entrée du deuxième neurone, ou neurone post-synaptique. Le calcul de la sortie d'un neurone s_i (cf figure 3.1 à droite) est effectué en utilisant les valeurs de sortie des différents neurones pré-synaptiques pour tous les synapses entrant dans le neurone, pondérées par les valeurs des poids des synapses qui les relient. Ces poids synaptiques (w_{ij}), possèdent une valeur soit positive soit négative, et on appelera ces connexions des connexions respectivement excitatrices ou inhibitrices. La somme pondérée des entrées du neurone constitue l'activation du neurone a(i) :

$$a_i = \sum_{j=1}^{N} w_{ij} s_j \tag{3.1}$$

où j représente l'indice du neurone pré-synaptique, et N le nombre de neurones pré-synaptiques. Il faut également noter la présence d'un neurone virtuel (non compté dans N), ou neurone de biais, dont la sortie est uniformément égale à 1 (sortie maximale), représenté par le neurone s_0 sur la figure 3.1. Ce biais va servir à fournir une activation soit positive, soit négative en l'absence d'autres activations notables.

Il ne reste plus qu'à calculer la valeur de sortie du neurone :

$$s_i = f(a_i) \tag{3.2}$$

où f est la *fonction d'activation* du neurone. Plusieurs types de fonction d'activation peuvent être utilisés, mais la plus utilisée reste la sigmoïde (cf fig. 3.2 et la formule 3.3), dans laquelle k représente en fait la pente à l'origine de la sigmoïde (cette fonction possède une sortie bornée entre 0 et 1).

$$f(x) = \frac{1}{1 + e^{-kx}} \tag{3.3}$$

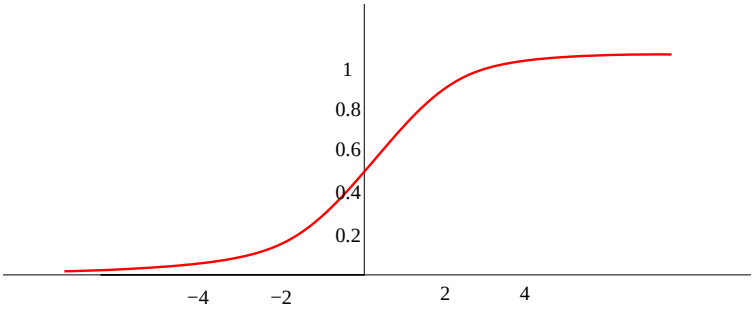


FIG. 3.2 – la sigmoïde.

3.1.2.2 Les réseaux récurrents

Pour certains types d'expérience, un processus de mémorisation peut s'avérer bénéfique, voire nécessaire. Dans ce cadre là, on peut utiliser un réseau récurrent, pour lequel il existe des boucles de connection entre les neurones : les activations des neurones constituent une mémoire entre une mise à jour des activations de l'ensemble du réseau et la suivante.

En effet, dans le cas de réseaux récurrents, la mise à jour des activations du réseau ne dépend plus uniquement des entrées du réseaux, mais également des sorties de chacun

des autres neurones auxquels son entrée est reliée. L'utilisation principale de ce type de réseau consiste alors, dans le cadre d'un temps discrétisé, à mettre à jour simultanément les activations de l'ensemble des neurones au temps $t+1$ en fonction de l'ensemble des sorties au temps t et des entrées au temps $t+1$ (l'initialisation à $t=0$ est en général faite en mettant les activations à 0). Les sorties du réseau sont alors la réponse du réseau au temps $t+1$.

A noter qu'il est possible aussi d'utiliser les réseaux de neurones récurrents pour l'approximation de phénomènes stationnaires : les mises à jour synchrones des activations sont itérées pour les mêmes entrées, jusqu'à stabilisation (éventuelle). Mais nous n'entrerons pas dans les détails de cette utilisation ici, puisque les fonctions (contrôleurs) que nous cherchons à identifier dépendent du temps : la robotique évolutionnaire utilise donc exclusivement l'approche par mise à jour synchrone à chaque pas de temps décrite dans le paragraphe précédent.

Un exemple de réseau récurrent très utilisé en robotique évolutionnaire est présenté figure 3.3 : il s'agit du réseau dit de type Elman [Elman, 1990], qui a une architecture semblable à celle d'un perceptron multi-couche, sauf que la couche cachée est totalement récurrente, i.e. tous les neurones sont reliés à tous les autres.

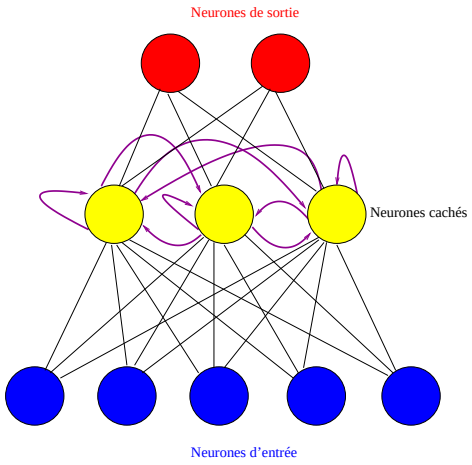


FIG. 3.3 – Un exemple de réseau récurrent, ici en réseau de type Elman (1990) constitué d'une couche cachée totalement récurrente, tous les neurones de cette couche sont reliés aux autres.

Nous allons maintenant passer en revue rapidement quelques méthodes classiques d'apprentissage des réseaux de neurones.

3.1.3 La rétro-propagation du gradient

L'algorithme de rétro-propagation du gradient est probablement l'algorithme d'apprentissage supervisé le plus populaire pour les réseaux de neurones. C'est notamment

grâce à lui que l'intérêt pour la technique neuronale est revenu à la fin des années 80. En effet, l'erreur commise sur les (nombreux) exemples d'apprentissage peut être (rétro)-propagée de proche en proche, à partir de la dernière couche du réseau (la sortie) vers les couches précédentes. Cela permet de calculer explicitement le gradient de la fonction d'erreur, qui est quadratique par rapport aux poids du réseau, et d'utiliser une des méthodes de minimisation à base de gradient pour la minimisation de cette erreur. Plus précisément, la rétro-propagation du gradient est une variante du gradient stochastique dans laquelle les exemples sont utilisés l'un après l'autre, dans un ordre aléatoire, avec mise à jour des poids au fur et à mesure. Nous allons maintenant détailler cet algorithme : en effet, bien que la situation en robotique évolutionnaire ne permette pas d'utiliser cette technique pour l'apprentissage des poids du contrôleur neuronal, du fait de l'absence d'exemples, nous utiliserons la rétro-propagation pour l'adaptation par anticipation décrite dans le chapitre 5 de cette thèse.

Prenons l'exemple d'un perceptron multi-couches. La couche de sortie indique le résultat de la fonction que l'on cherche à approcher sur les exemples disponibles. On peut penser par exemple à un algorithme de reconnaissance de caractères qui prendrait en entrée les différents pixels d'une image, et la couche de sortie aurait pour rôle de décoder le caractère associé.

Situons-nous sur la dernière couche du réseau de neurones. Grâce aux exemples d'apprentissage, l'erreur quadratique peut être calculée :

$$E = \frac{1}{2}\sum_i (d_i - s_i)^2 \tag{3.4}$$

d_i représente ici la sortie désirée pour le neurone de sortie i et s_i représente la sortie réellement obtenue à travers la propagation du réseau de neurones.

L'objectif de cet algorithme est de déterminer l'influence des poids sur l'erreur commise en sortie, par conséquent, il suffit de calculer le gradient de l'erreur par rapport aux poids :

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial a_i}.\frac{\partial a_i}{\partial w_{ij}} \tag{3.5}$$

où a_i représente l'activation du neurone associé.

$$\frac{\partial E}{\partial a_i} = -(d_i - s_i).f'(a_i) \tag{3.6}$$

Considérons que la couche immédiatement précédente comporte k neurones, on a alors :

$$\frac{\partial a_i}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}[\sum_k w_{ik}s_k] = s_j \tag{3.7}$$

Les gradients de l'erreur par rapport aux poids reliés à la couche de sortie peuvent désormais être calculés.

Intéressons-nous maintenant à la couche cachée qui précède immédiatement la couche de la sortie. Les indices i correspondent désormais à cette couche cachée, et l'indice j à la couche précédant cette couche, l'indice k quand à lui indicera les neurones de la couche de sortie.

Procédons de la même manière que précédemment, et calculons le gradient de l'erreur par rapport à l'activation des neurones de cette couche cachée :

$$\frac{\partial E}{\partial a_i} = [\sum_k \frac{\partial E}{\partial a_k}.\frac{\partial a_k}{\partial s_i}].\frac{ds_i}{da_i} = [\sum_k \frac{\partial E}{\partial a_k}.w_{ki}].f_i'(a_i) \tag{3.8}$$

Or $\frac{\partial E}{\partial a_k}$ a déjà été calculé précédemment, lors du calcul de la couche qui suit la couche courante, et va donc pouvoir être utilisé ici. C'est en cela que cet algorithme est appelé "rétro-propagation du gradient".

Il reste seulement à calculer la dérivée de la fonction f – et d'un point de vue purement pratique, l'intérêt de l'utilisation de la sigmoïde prend un sens nouveau, puisque cette fonction a une dérivée très simple à calculer si l'on connait déjà ses valeurs (et elles aussi ont déjà été calculées). En effet :

$$f'(x) = k.f(x).(1 - f(x)) = k.s_i.(1 - s_i) \tag{3.9}$$

Une fois calculés les gradients, il suffit d'utiliser la technique de la descente du gradient et de modifier les poids du réseau de neurones dans la direction opposée la direction du gradient :

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} \tag{3.10}$$

où η est une constante définie par l'utilisateur, appelé taux d'apprentissage (*learning rate*).

Cet algorithme est très efficace dans le cadre de l'apprentissage supervisé. Il souffre cependant de quelques défauts : d'une part, comme tout algorithme de gradient, et malgré la nature (faiblement) stochastique du fait de la présentation des exemples un par un dans un ordre aléatoire, il peut facilement être piégé dans des minima locaux : d'autre part, il est en général assez lent, et il est nécessaire de présenter plusieurs fois l'ensemble des exemples disponibles. Plusieurs manière de pallier ce dernier défaut ont été proposées, parmi lesquelles nous avons choisi (au chapitre 5) une des plus classiques, celle consistant a rajouter un terme d'inertie, qui permet d'augmenter les variations des poids en privilégiant la direction déjà prise au cycle précédent :

$$w_{ij}(t+1) := w_{ij}(t) - \eta \frac{\partial E}{\partial w_{ij}} + \alpha \delta w_{ij}(t) \tag{3.11}$$

avec $\delta w_{ij}(t) = w_{ij}(t) - w_{ij}(t-1)$, et où α est le moment d'inertie (*momentum*), un autre paramètre à définir par l'utilisateur.

3.2 Les algorithmes évolutionnaires

Les algorithmes évolutionnaires sont inspirés de la théorie de l'évolution de Darwin [Darwin, 1859] qui veut que les individus les plus adaptés à leur environnement survivent et se reproduisent par variations aveugles. Plus formellement, ce sont des algorithmes d'optimisation stochastique d'ordre 0, c'est-à-dire n'utilisant que des valeurs prises par la fonction à optimiser (à l'exclusion de valeurs des dérivées). Mais les points les plus caractéristiques de ces algorithmes sont d'une part leur souplesse d'emploi, puisqu'ils peuvent optimiser des fonctions définies sur des espaces quelconques, et d'autre part leur robustesse face aux optima locaux. De plus, les algorithmes évolutionnaires ne sont pas restreints aux espaces de recherche paramétriques (dans lesquels la solution est cherchée sous forme d'une vecteur de paramètres de taille fixe). Une présentation détaillée et actuelle de ces algorithmes se trouve dans l'ouvrage de Eiben et Smith [Eiben et Smith, 2003]. Ces algorithmes sont aujourd'hui couramment utilisés pour trouver l'optimum de fonctions difficiles (ayant par exemple de nombreux optima locaux), ou définies sur des espaces non-standards.

Le principe général des algorithmes évolutionnaires est illustré sur la figure 3.4 : une *population* d'*individus* (ensemble de points de l'espace de recherche, chacun représentant une solution potentielle au problème d'optimisation posé) évolue suivant les principes Darwiniens. Dans un premier temps, la population est **initialisée**, en général de manière aléatoire (sachant qu'il est possible de biaiser le choix de la population initiale en fonction des connaissances du problème [Surry et Radcliffe, 1996]). Les individus ainsi générés sont alors **évalués**, c'est-à-dire que l'on calcule la valeur de la fonction objectif en ces points. Nous rentrons alors dans la boucle de l'algorithme, appelée boucle *générationnelle*, et cette population initialisée constitue les *parents*.

Le **critère d'arrêt** est le plus souvent un nombre fixe de générations, ou un nombre fixé de générations sans amélioration de la meilleure performance dans la population. En cas d'arrêt de l'algorithme, le meilleur individu de la population de parents est rendu comme solution.

L'étape suivante de l'algorithme est l'étape de **sélection**, durant laquelle certains individus vont être sélectionnés pour devenir des géniteurs, c'est-à-dire qu'ils sont autorisés à se reproduire. La sélection favorise les individus réalisant les meilleures valeurs de la

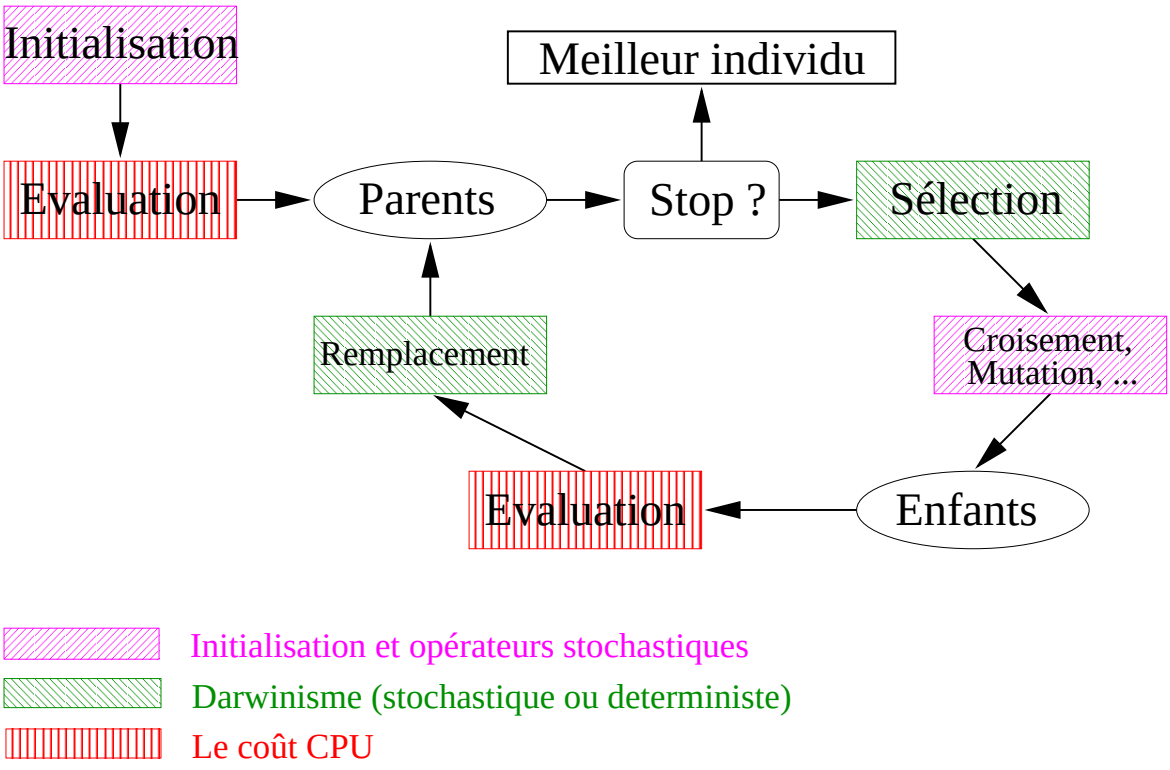


FIG. 3.4 – le principe de l'algorithme évolutionnaire.

fonction objectif (les plus grandes si l'on est en train de maximiser, les plus faibles si l'on minimise). Cette sélection peut être stochastique ou déterministe.

L'étape de reproduction consiste à appliquer à ces géniteurs des opérateurs de **variation** qui les modifient de manière stochastique. On distingue deux types d'opérateurs suivant le nombre d'opérandes : les opérateurs de croisement construisent un ou deux enfants à partir de deux parents ; les opérateurs de mutation génèrent un enfant à partir d'un parent.

La population des enfants est alors à son tour **évaluée** – on calcule la valeur de la fonction objectif en ces nouveaux points. Enfin, certains individus parmi les parents et les enfants sont sélectionnés (toujours en favorisant les meilleurs) afin de constituer la population des parents à la génération suivante. C'est l'étape dite de *remplacement*. Une nouvelle génération commence alors, et la boucle générationnelle est bouclée. Les générations vont ainsi se succèder jusqu'à la satisfaction du critère d'arrêt.

Il est important de souligner que dans la plupart des applications des algorithmes évolutionnaires, et en particulier en robotique évolutionnaire, l'étape prenant le plus de temps de calcul est l'étape d'évaluation (en robotique, elle fait appel à un simulateur qui doit tester le robot en situation, durant un certain nombre de pas de temps). Il faut en

tenir compte afin d'ajuster la taille de la population et le nombre d'enfants désiré au temps que l'on est prêt à consacrer pour une optimisation.

3.2.1 Historique des algorithmes évolutionnaires

Historiquement, on distingue 4 grandes classes d'algorithmes évolutionnaires :

- Les algorithmes génétiques : mis au point par John Holland dans les années 60 [Holland, 1975], puis popularisés par David Goldberg [Goldberg, 1989] à la fin des années 80, ils s'appuient avant tout sur une représentation binaire des individus. Ils sont particulièrement plébiscités par la communauté des utilisateurs des algorithmes évolutionnaires aux États-Unis.
- Les stratégies d'évolution : mis au point par Ingo Rechenberg [Rechenberg, 1972] et Hans-Paul Schwefel [Schwefel, 1981] dans les années 60 en Allemagne, ces algorithmes s'appuient sur une représentation réelle des individus et sur l'opérateur de mutation gaussienne. Les SE les plus performantes utilisent les mutations autoadaptatives, dans lesquelles chaque individu porte avec lui les paramètres de la mutation gaussienne qui lui sera appliquée – paramètres eux-même soumis à mutation [Bäck, 1995].
- La programmation évolutionnaire : mis au point par Larry Fogel dans les années 60 également [Fogel *et al.*, 1966], la programmation évolutionnaire s'est tout d'abord intéressée à l'évolution d'automate à états finis, pour notamment l'approximation des séries temporelles. Cependant, popularisés à son tour par David Fogel [Fogel, 1995], cet algorithme a également été utilisé dans de nombreux autres champs d'application. Les caractéristiques de l'évolution au sein de la programmation évolutionnaire en font des algorithmes très proches des stratégies d'évolution.
- Enfin, dernière instance différenciée d'algorithme évolutionnaire, la programmation génétique [Banzhaf *et al.*, 1998] s'intéresse à l'évolution de programmes, souvent représentés par des arbres. Proposée par Cramer [Cramer, 1985], elle a surtout été popularisée par Koza au début des années 90 [Koza, 1992, Koza, 1994, Koza et al., 1999].

3.2.2 Le dilemme exploration vs exploitation

Si les méthodes d'optimisation numérique classique à base de descente de gradients sont très efficaces sur les fonctions régulières sans optima locaux, elles sont le plus souvent irrémédiablement attirées par le premier optimum local qu'elles rencontrent : ce sont des algorithmes de recherche locale uniquement. À l'inverse, les marches aléatoires, et les algorithmes de Monte-Carlo de base essaient une succession de points de l'espace de recherche tirés au hasard : ils peuvent sans doute visiter tous les bassins d'attraction des optima

locaux, mais se révèlent incapables d'en trouver les optima.

Les algorithmes évolutionnaires tentent de marier les deux avantages : d'une part **explorer** largement l'espace de recherche, pour ne pas rester bloqués près des minima locaux ; mais également être capables d'**exploiter** les meilleurs résultats trouvés à un instant donné afin de les améliorer localement. Tous les paramètres d'un algorithme évolutionnaire permettent ainsi de modifier l'équilibre entre exploitation et exploration, et cet équilibre doit guider le choix des valeurs de ces paramètres par l'utilisateur.

Dans le cadre de cette thèse, nous avons choisis de travailler sur des espaces réels (l'ensemble des poids d'un réseau de neurones) en utilisant les stratégies d'évolution à mutation adaptative (décrites section 3.2.4.2) car elles permettent à la fois d'explorer largement dans un premier temps l'espace de recherche, mais aussi de rendre de plus en plus précises les mutations au fur et à mesure de l'avancement de l'algorithme, permettant ainsi la convergence fine de l'algorithme.

3.2.3 Les étapes darwiniennes : la sélection et le remplacement

Avant toute chose, éclaircissons les termes que nous allons utiliser par la suite. L'espace de recherche dans lequel l'optimum est recherché est appelé Ω, un individu est un élément X de Ω. L'espace Ω est souvent appelé espace génotypique : c'est l'espace dans lequel s'effectue la recherche, et surtout dans lequel sont définis les opérateurs de variation (croisement et mutation). Dans de nombreux cas d'application des algorithmes évolutionnaires, toutefois, l'espace génotypique n'est pas l'espace sur lequel est définie la fonction objectif. On appelle ce dernier l'espace phénotypique, ou espace des comportements. On parle également souvent du codage pour désigner l'application de l'espace phénotypique dans l'espace génotypique, et du décodage pour l'opération inverse.

Cependant, dans cette thèse, nous avons exclusivement utilisé des représentations en nombres réels, dans lesquelles les deux espaces génotypique et phénotypique sont confondus, et nous ne ferons donc pas la distinction dans la suite. En particulier, nous supposerons que la fonction objectif, à optimiser, est définie sur l'espace Ω, et nous appelerons indifféremment *performance* ou *fitness* la valeur de F en un point de Ω.

Enfin, la population à l'etape i, constituée de P éléments de Ω, sera notée π_i.

3.2.3.1 La sélection

L'étape de sélection doit permettre de choisir les individus qui vont ensuite se reproduire au moyen des opérateurs de variation. Par définition, la sélection est un opérateur d'exploitation : le fait de sélectionner les individus les plus adaptés entraîne forcément une perte certaine de diversité. Par conséquent, une sélection trop forte (favorisant trop les meilleurs individus au détriment des plus mauvais) peut conduire à une convergence

prématurée de l'algorithme, vers un optimum local. Inversement, le fait de ne pas appliquer de sélection assez forte peut empêcher toute convergence. Cette étape doit donc être calibrée avec soin.

La sélection peut être déterministe – seuls les n premiers individus de la population par ordre de fitness seront autorisés à se reproduire – ou au contraire stochastique. Un exemple de sélection stochastique est la sélection par *roulette* : le principe de la roulette consiste à tirer P fois un parent parmi $\pi_t = X_1, ..., X_p$, de telle sorte que la probabilité pour un individu d'être sélectionné soit proportionnelle à sa performance. La probabilité de sélection pour un individu i est donc de $\frac{F(X_i)}{\sum_j F(X_j)}$. Tout en favorisant les individus les plus performants, ce mode de sélection, à l'inverse de la sélection déterministe, laisse une petite chance aux individus de faible performance d'être sélectionnés, préservant ainsi la diversité de la population.

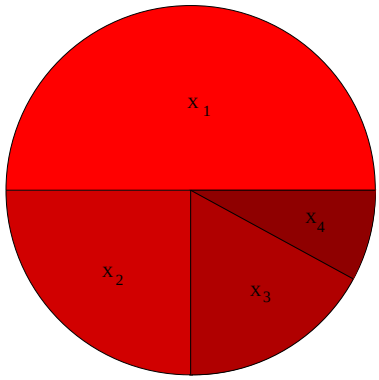


FIG. 3.5 – La roulette, Pour P=4 et $F(X_i) = \{60, 30, 20, 10\}$

Mais le mode de sélection stochastique le plus utilisé est la sélection par *tournoi* : pour choisir un parent en vue de la reproduction, on tire uniformément T parents (T est la raille du tournoi, fixée par l'utilisateur), et le meilleur de ces T est sélectionné. Le paramètre T permet de régler la force de la sélection, les grandes valeurs favorisant les meilleurs individus, la plus petite valeur étant $T = 2$, qui permet de sélectionner également de nombreux individus de performance moyenne.

3.2.3.2 Le remplacement

L'étape de remplacement constitue l'autre étape du darwinisme artificiel, et sert à déterminer quels individus, parmi les parents de la génération courante et leurs enfants, seront les parents de la génération suivante. À la différence de l'étape de sélection, durant laquelle les individus peuvent être sélectionnés plusieurs fois (si l'on utilise la sélection par roulette par exemple), lors de l'étape de remplacement, un individu est sélectionné

une fois – et il survit alors à la génération suivante – ou pas du tout, et il disparait définitivement de l'évolution en cours. Tout comme la sélection, l'étape de remplacement peut être déterministe ou stochastique.

On distingue deux catégories principales de remplacement, suivant que les futurs parents sont sélectionnés parmi les enfants seulement, ou bien parmi les parents courants et leurs enfants. On peut donc citer les deux remplacements déterministes utilisés dans les stratégies d'évolution (la taille de la population est courament appelée μ dans le domaine des stratégies d'évolution) :

- $(\mu, \lambda) - ES$: les prochains μ parents sont les meilleurs parmi les λ enfants. Ceci peut entraîner de meilleurs résultats de convergence, mais il est également possible que la performance du meilleur parent décroisse d'une génération à la suivante, si tous les enfants ont des mauvaises performances.
- $(\mu + \lambda) - ES$: les prochains μ parents sont les meilleurs parmi les μ parents et les λ enfants. Ceci permet une meilleure robustesse, garantissant en effet que les meilleurs individus sont conservés d'une génération à l'autre. En revanche, ce mode de remplacement augmente, par rapport au précédent, les risques de convergence prématurée.

Nous ne détaillerons pas ici de remplacement stochastique, n'ayant utilisé que les deux remplacements déterministes ci-dessus dans cette thèse.

3.2.4 Les opérateurs de variation

Les opérateurs de variations sont regroupés en deux catégories : les opérateurs de croisement agissent sur deux parents et donnent naissance à un ou deux enfants, et les opérateurs de mutation qui agissent sur un unique parent pour donner un enfant. Ces opérateurs sont "aveugles" au sens darwinien du terme, c'est-à-dire qu'ils ne tiennent aucun compte lors de leur application de la fitness des individus auxquels ils sont appliqués – sachant que ces individus ont au préalable été sélectionnés en fonction de cette fitness par l'étape de sélection décrite plus haut.

Dans le cadre de cette thèse, nous avons travaillé uniquement dans le cadre des stratégies d'évolution manipulant des génotypes qui sont des vecteurs de variables réelles. Nous ne nous sommes par conséquent pas intéressés aux algorithmes génétiques, codés de manière binaire. Nous ne décrirons donc que les opérateurs de variation (opérateurs de croisement et de mutation) sur des espaces réels. En particulier, l'espace Ω sera $\mathbb{R}^d$, où d est le nombre de coordonnées du génotype (ou un sous-ensemble).

3.2.4.1 Le croisement

La métaphore biologique du croisement est la reproduction sexuée : deux individus sont sélectionnés (i.e. choisis aléatoirement parmi les *géniteurs*). Un nouvel individu est créé en recombinant les caractéristiques de ces deux parents. Dans le cadre général de vecteurs de caractéristiques, un premier croisement possible consiste à échanger les caractéristiques correspondantes entre les deux parents, créant ainsi deux enfants (le fait qu'un ou deux enfants soient créés semble n'avoir aucune influence sur les résultats globaux de l'algorithme – mis à part le fait qu'il faut bien sûr effectuer deux fois plus de croisements pour obtenir globalement le même nombre d'enfants).

Cependant, dans le cadre continu, des opérateurs plus adaptés ont été proposés, qui sont ceux que nous utiliserons. Ces croisements sont à base de combinaisons linéaires des deux parents, et sont appelés suivant les communautés *croisement arithmétique* (au sein des algorithmes génétiques avec codage réel), ou *croisement intermédiaire* (au sein de la communauté des Stratégies d'Évolution).

Dans le croisement arithmétique simple, un réel α est tiré uniformément dans $[0, 1]$, et l'enfant de deux individus X et Y (appartenant à $\mathbb{R}^d$) est défini par

$$(\overrightarrow{X}, \overrightarrow{Y}) \rightarrow \alpha\overrightarrow{X} + (1 - \alpha)\overrightarrow{Y}$$

avec

$$\alpha = U([0, 1])$$

alors que dans le croisement intermédiaire généralisé, un réel α_i est tiré uniformément dans $[0, 1]$ pour chaque coordonnée, et l'enfant de X et Y est défini par

$$(\overrightarrow{X}, \overrightarrow{Y}) \rightarrow \overrightarrow{\alpha_i X_i} + (1 - \alpha_i)\overrightarrow{Y_i}$$

avec

$$\alpha_i = U([0, 1])$$

De plus, afin de limiter le caractère contractant des opérateurs de croisement arithmétiques ci-dessus (un algorithme qui n'a comme opérateur que le croisement, sans aucune sélection, finit en moyenne sur le barycentre de la population), il est possible d'étendre l'intervalle sur lequel sont choisis les coefficients de la combinaison linéaire, en les choisissant par exemple dans $[-0.5, 1.5]$.

3.2.4.2 Les mutations auto-adaptatives

La mutation d'un individu le modifie aléatoirement pour donner naissance à un unique enfant. Un point essentiel pour le succès des algorithmes évolutionnaires est l'ergodicité

de la mutation, i.e. le fait que l'on peut par une suite de mutations joindre deux points quelconques de l'espace de recherche. La mutation est le seul garant de la globalité de la recherche : c'est le principal opérateur d'exploration. Toutefois, lorsque l'opérateur de mutation a une "force" variable (comme c'est le cas pour l'opérateur de mutation gaussienne auto-adaptatif décrit ci-dessous), la mutation peut aussi être l'opérateur d'exploitation.

Dans le cadre de génotypes réels, l'opérateur de mutation le plus efficicace et le plus utilisé est donc la mutation gaussienne auto-adaptative, que nous allons maintenant décrire.

Le principe de base de la mutation gaussienne est d'ajouter un bruit gaussien centré aux variables que l'on désire muter :

$$X_i := X_i + N(0, \sigma)$$

Mais tout l'art de la mutation gaussienne réside dans le choix du paramètre σ, donnant la variance de la perturbation gaussienne, et donc la "force" de la mutation : environ 67% des tirages seront compris entre -σ et +σ (et plus de 99% entre -3σ et 3σ) – mais il existe également une probabilité strictement positive de tirer tout nombre réel positif ou négatif (la "queue" de la distribution ne s'annule jamais). Tous les ingrédients sont donc présents pour pouvoir faire de cette mutation un opérateur soit d'exploration (pour les grandes valeurs de σ) soit d'exploitation (pour les petites valeurs de σ). Il faut par contre trouver la bonne loi de mise à jour de σ pour atteindre un bon compromis entre ces deux comportements.

Une première idée pour ce faire fut de muter plus fortement les mauvais individus (puisqu'il ne sert à rien de les exploiter, autant explorer encore plus) et faiblement les bons individus (pour exploiter l'espace de recherche autour d'eux). Cette idée, utilisée dans les premiers temps de la Programmation Évolutionnaire [Fogel *et al.*, 1966], se révèla difficile à mettre en oeuvre dans le cadre de variables réelles.

La première approche *adaptative*, c'est-à-dire dans laquelle la décision est prise au vu de la situation courante, fut la célèbre règle des 1/5 de Ingo Rechenberg [Rechenberg, 1972] : soit τ la proportion de mutations réussies (c'est-à-dire pour lesquelles la fitness de l'enfant a été meilleure que celle du parent) dans les dernières T générations. On peur montrer rigoureusement que, sur la fonction "sphère" (minimisation de la norme du vecteur $\overrightarrow{X}$), le réglage optimal de la mutation correspond à une proportion d'environ 0.2, et varie inversement proportionnellement à σ [Schwefel, 1981]. On en déduit par extrapolation à toute fonction qu'il faut augmenter σ si τ est trop grand, et le diminuer dans le cas inverse.

Cependant, cette approche possède quelques faiblesses, et ne prend pas en compte les caractéristiques locales du paysage de fitness, du fait notamment que la même valeur de σ est utilisée pour toute la population, et pour toutes les composantes du

génotype. Pour pallier ce défaut, et plus généralement pour se débarrasser élégamment de la tâche fastidieuse du règlage des paramètres de la mutation, Rechenberg et Schwefel [Rechenberg, 1972, Schwefel, 1981] ont proposé de rendre la mutation *auto-adaptative* : chaque individu possède ses propres paramètres de mutation, qui sont eux-mêmes sujets à mutation avant d'être utilisés pour la mutation des variables elles-mêmes.

Bien que la sélection s'effectue sur les valeurs de fitness, dans lesquelles les paramètres de la mutation n'interviennent pas, les individus qui survivront seront ceux qui auront à la fois les bonnes valeurs pour les variables du problème (sur lesquelles se fait la sélection), mais aussi les bonnes valeurs des paramètres de mutation, sinon ils seront immanquablement dépassés par d'autres, mieux adaptés aux caractéristiques locales du paysage de fitness. De manière informelle, lorsque le gradient (dans une direction) est fort, il faut faire de "petites" mutations, et inversement. De là le succès de la méthode :

- des mutations successives avec des paramètres de mutation aberrants ne peuvent par être constamment réussies
- les individus qui survivent longtemps résultent de nombreuses mutations successives réussies
- les individus qui survivent longtemps doivent donc avoir de "bons" paramètres de mutation.

Il existe trois types de mutations auto-adaptatives, suivant la structure des paramètres de la mutation gaussienne :

- La mutation *isotropique*, dans laquelle chaque individu possède un scalaire σ qui est utilisé pour l'ensemble des composantes du vecteur lors de la mutation. Plus précisément, la mutation de l'individu $(\overrightarrow{X}, \sigma)$ s'effectue en deux temps, mutation de σ, suivant une loi log-normale (pour des raisons de symétrie multiplicative autour de 1, puis mutation des variables X_i, en utilisant la nouvelle valeur de σ.

$$\sigma := \sigma \; exp(\tau N(0,1)) X_i := X_i + N(0,\sigma)$$

  où τ est un paramètre utilisateur (mais que l'on peut qualifier du deuxième ordre, et dont l'influence sur les résultats est faible).
- La mutation *anisotropique*, dans laquelle les paramètres de la mutation sont un vecteur de valeurs, qui représentent les déviations dans chacune des directions canoniques. La mutation s'effectue aussi en deux temps,

$$\kappa = \tau N(0,1) i = 1 \ldots N \qquad \sigma_i := \sigma_i \; exp(\kappa + \tau' N(0,1)) \qquad X_i := X_i + N(0,\sigma_i)$$

  où τ et τ' sont encore une fois des paramètres (de second ordre) utilisateur.
- Les mutations *corrélées* dans lesquelles les paramètres de la mutation sont une ma-

trice de covariance complète, et que nous ne détaillerons pas ici, dans la mesure où nous avons utilisés les mutations anisotropique lors des travaux de cette thèse.

Les valeurs préconisées par Schwefel [Schwefel, 1981], basée sur des études théoriques sur la fonction sphère en grande dimension [Schwefel, 1981], sont les suivantes :

$$\tau \propto \frac{1}{\sqrt{2\sqrt{N}}}\tau' \qquad \propto \frac{1}{\sqrt{2N}} \qquad \beta = 0.0873 \ (=5^o)$$

3.2.5 Algorithme utilisé

L'algorithme que nous avons utilisé durant cette thèse est le suivant :

Initialiser t à 1 // *première génération*
Initialiser $P(1)$ // *population de μ individus tirés uniformément*
// *constituant la première génération* $P(1) = \{X_1(1), \ldots, X_\mu(1)\}$
Evaluer $P(1)$ // *i.e., calculer* $f(X_i(t))$
Tant que le critère d'arrêt de l'algorithme n'est pas vérifié, faire :
    Pour j allant de 1 à λ faire // λ étant le nombre d'enfants désirés
        Avec probabilité p_{cross}
            Croiser aléatoirement deux individus de la population pour obtenir un nouvel individu
        Sinon
            Choisir aléatoirement un individu de la population
        Avec probabilité p_{mut}
            Muter l'individu $X_{(j\_modulo\_\mu)}$ en utilisant la mutation auto-adaptative anisotropique
        Sinon le garder inchangé
            Rajouter cet individu dans la population des enfants $P'(t)$
    Fin pour
    Evaluer $P'(t) = \{X'_1(t), \ldots, X'_\lambda(t)\}$ //évaluer les enfants
    Sélectionner μ individus parmi $P'(t)$ (Moteur $(\mu, \lambda) - ES$)
    $t = t + 1$ //nouvelle génération
Fin tant que

Notre critère d'arrêt sera toujours soit un nombre donné de générations, soit l'uniformisation de la population ("convergence" vers un individu unique).

3.3 L'évolution artificielle des réseaux de neurones

Nous présentons enfin ici les diverses techniques utilisées pour la mise au point de réseaux de neurones par évolution artificielle. Signalons que, si la robotique est un domaine où cette technique a été très utilisée, de nombreux domaines ont aussi eu recours à cette

manière d'optimiser des réseaux de neurones, qui est une des rares possibilités qui restent ouvertes lorsque les méthodes de type gradient sont inapplicables, en général du fait de l'absence d'exemples.

De plus, la souplesse des algorithmes évolutionnaires permet leur utilisation sur des espaces de recherche non standards : dans le cadre des réseaux de neurones, cela a permis l'optimisation non plus seulement des poids d'un réseau (ce qui revient à de l'optimisation paramètrique), mais également de l'architecture (et des poids) d'un réseau de neurones, à condition de trouver une représentation (et les opérateurs d'évolution adaptés) pour les architectures que l'on veut optimiser. Nous allons rapidement passer en revue ici quelques-unes de ces techniques, puis discuter de l'opportunité de l'utilisation des techniques les plus avancées d'optimisation des réseaux de neurones dans le cadre de la robotique évolutionnaire.

3.3.1 L'évolution des seuls poids synaptiques

C'est évidemment l'approche la plus simple à mettre en œuvre de l'optimisation de réseaux de neurones par algorithmes évolutionnaires : les seuls paramètres à optimiser sont les poids du réseau. N'oublions pas cependant que le rôle du concepteur sera de définir au préalable une architecture pour le réseau, ce qui demande un grand savoir-faire et une expérience certaine. Prenons en effet le cadre de la robotique évolutionnaire : un petit réseau, avec un faible nombre de poids, sera plus facilement optimisé par l'algorithme évolutionnaire. Cependant, si la tâche à réaliser est complexe, on voit intuitivement qu'un réseau très simple ne suffira pas. Il faut donc adapter la taille du réseau et la topologie du réseau à la complexité de la tâche à résoudre. Même en se restreignant aux perceptrons multi-couches (voir section 3.1.2.1), il faut déterminer le nombre de couches et le nombre de neurones par couche.

Cette approche a été utilisée dans de nombreux domaines pour lesquels des exemples ne sont pas disponibles. Outre la robotique évolutionnaire, (voir les nombreuses citations de cette thèse), on peut citer des problèmes de contrôle [Ronald et Schoenauer, 1994], d'identification de lois, ...

3.3.2 L'évolution des architectures

Il existe de nombreuses possibilités de représentation pour l'évolution des réseaux de neurones. Nous allons en passer rapidement en revue quelques grandes familles.

Il faut tout d'abord distinguer deux types d'approche concernant l'évolution de l'architecture des réseaux de neurones, suivant que l'on optimise par évolution artificielle simultanément l'architecture et les poids des connexions, ou bien uniquement l'architecture, les poids des connexions étant optimisés dans une deuxième phase, en général par

un algorithme non-évolutionnaire.

En effet, compte-tenu de l'influence de l'architecture du réseau de neurones sur les résultats même dans un cadre d'apprentissage supervisé (à partir d'exemples), une des utilisation principales des méthodes d'évolution de l'architecture d'un réseau de neurones reste la classification : l'architecture est optimisée par évolution, et les poids sont ensuite ajustés par un algorithme de rétro-propagation standard, la fitness de l'architecture étant donnée par l'erreur de classification après apprentissage des poids par rétro-propagation [Yao, 1999].

Cette approche présente toutefois quelques inconvénients :

- quel que soit l'algorithme d'apprentissage utilisé, une phase d'initialisation aléatoire des poids est nécessaire. Or, de ce fait, l'algorithme d'apprentissage des poids peut converger vers différents minima locaux, entrainant de fait, une performance différente pour l'individu à chaque nouvelle évaluation.
- le fait d'utiliser un algorithme d'apprentissage différent peut entrainer une performance différente.

Pour ces raisons, et puisque de toute facon le cadre de la robotique évolutionnaire ne se prête pas à cette approche, nous ne développerons pas plus avant cette approche, nous concentrant sur les approches faisant évoluer simultanément l'architecture et les poids du réseau.

Dans ce cadre, on distingue deux types d'approche, suivant la représentation choisie pour le réseau : le codage direct manipule des neurones et des connexions, alors que l'approche développementale manipule un programme qui, exécuté, donnera le réseau de neurones.

3.3.2.1 Le codage direct

Le codage direct constitue un codage explicite de l'architecture à utiliser. L'approche la plus utilisée commence par définir un nombre maximal de neurones, et le génotype est dans ce cas la matrice de connectivité que constituent les poids du réseau, c'est à dire la carte des poids qui doivent être reliés à tel neurone ou à un autre. Si l'on ne fait évoluer que la connectivité, le génotype est une matrice de booléens, sinon, on doit faire évoluer une matrice de poids. La figure (3.6) présente un exemple très simple de codage direct avec la matrice de connectivité associée.

Le problème majeur de cette représentation est le passage à l'échelle, en effet si le génotype possède une taille raisonnable pour des réseaux de neurones très simples, sa taille augmente considérablement lorsque celui ci augmente.

Des solutions dans lesquelles l'architecture est représentée de manière plus concise ont également été proposées. Les représentations choisies sont alors *ad hoc*, et la plus grande difficulté est de définir un opérateur de croisement entre deux réseaux

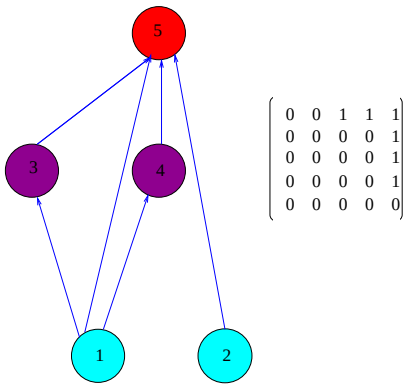


FIG. 3.6 – Evolution d'architecture neuronale via un codage direct.

n'ayant pas du tout la même architecture, certaines approches [Angeline *et al.*, 1993, Fadda et Schoenauer, 1995, Yao et Liu, 1997] n'utilisaient en fait que des opérateurs de mutation.

Une approche un peu différente a été proposée plus récemment, dans laquelle l'architecture est peu à peu augmentée par évolution (les opérateurs de mutation ne peuvent qu'ajouter des neurones ou des connexions) [Stanley et Miikkulainen, 2002]. Ceci semble donner de bons résultats pour des problèmes de classification ainsi que pour des problèmes de contrôle.

Quoique d'une manière moindre que les approches encodant la matrice de connectivité complète, ces approches souffrent encore du problème du passage à l'échelle.

3.3.2.2 Approche développementale

L'approche développementale consiste à faire évoluer non pas le réseau de neurones directement, mais un programme qui va ensuite construire le réseau à partir d'un embryon. Notons que cette approche est beaucoup plus proche de la réalité biologique, dans la mesure où les connexions du système nerveux ne sont certainement pas codées individuellement dans le génome des individus.

L'outil utilisé pour faire évoluer de tels programmes est la programmation génétique, qui a précisément été mise au point pour l'évolution de programmes au début des années 90 (initialement, des programmes LISP [Koza, 1992]). Le génotype codant le programme est représenté par une structure d'arbre. Cette approche a permis à J. Koza d'obtenir de superbes résultats dans le cadre du développement de circuits analogiques de plus en plus performants [Koza et al., 1999], (re)découvrant par exemple des brevets déposés il n'y a que quelques années. A noter cependant que ces résultats n'ont pu être obtenus qu'à l'aide de populations de plusieurs centaines de milliers d'individus réparties sur des clusters de plusieurs dizaines voire centaines d'ordinateurs.

Cependant, le premier exemple d'une approche développementale de l'évolution des réseaux de neurones a été réalisée indépendamment de la programmation génétique, par l'approche dite du codage cellulaire de F. Gruau [Gruau, 1994]. Cette approche est également appelée *embryogénèse artificielle*, et a été reprise ensuite par J. Koza [Koza et al., 1999] dans le cadre de la conception de cicruits analogiques déjà citée.

Le codage cellulaire repose sur la donnée d'un embryon (par exemple, un réseau de neurone ... sans neurones, reliant directement les entrées aux sorties du réseau dans les travaux originaux de F. Gruau) et la donnée d'opérateurs de développement (e.g. division parallèle, séquentielle, ...). L'arbre représente alors un programme qui, appliqué à l'embryon, va donner la solution du problème cherché (le réseau de neurones dans l'approche de Gruau). A noter que F. Gruau a également introduit la récursion dans la programmation de tels arbres, permettant ainsi l'évolution de modularité au sein des programmes.

Dans le cadre de la robotique, l'opérateur n'a qu'à définir les entrées sensorielles et les sorties des actuateurs. Cependant, l'évolution de l'architecture a finalement été peu utilisée en robotique évolutionnaire (et ce point sera développé dans la section suivante). Citons les travaux de J. Kodjabachian [Kodjabachian et Meyer, 1998], qui s'est inspiré fortement du codage cellulaire de Gruau afin de développer une architecture pour un robot mobile hexapode. En particulier, l'introduction de la modularité lui a permis de trouver des architecture ayant factorisé les contrôleurs pour des paires de pattes du robot. Floreano et Urzelai [Floreano et Urzelai, 2001] ont également proposé d'utiliser la morphogénèse pour la robotique évolutionnaire. Plus récemment, S. Doncieux [Doncieux, 2003] a appliqué le même type de codage, tout en se focalisant sur une modularité prédeterminée (fonctions dérivées, ...) afin de contrôler un ballon dérigeable.

Nous allons revenir dans la section suivante sur cette question préalable à l'évolution de contrôleurs neuronaux en robotique évolutionnaire : faut-il faire évoluer l'architecture du réseau ?

3.3.3 Évolution des poids ou de l'architecture ?

Face à un problème de robotique évolutionnaire, et ayant décidé d'utiliser les réseaux de neurones comme contrôleurs pour le robot, et l'évolution artificielle comme méthode d'optimisation de ces contrôleurs (voir la discussion dans l'introduction de ce chapitre), il reste à décider si nous allons fixer une architecture à la main, et en faire uniquement évoluer les poids, ou si nous allons utiliser les techniques décrites dans la section précédente pour faire évoluer aussi l'architecture.

Le premier argument qui vient à l'esprit est évidemment qu'il vaut mieux chercher la solution dans un espace plus grand car on est sûr qu'il y existe de meilleures solutions – ce qui ferait pencher vers l'optimisation de l'architecture (et des poids). Cependant, et ceci n'est en rien spécifique à la robotique évolutionnaire, chercher dans un espace beaucoup

plus grand peut tout simplement rendre tout recherche inopérante, et il faut trouver en fait un compromis entre la taille de l'espace de recherche et l'efficacité de l'algorithme d'optimisation. L'exemple des résultats de Koza sur les circuits analogiques est édifiant, et il n'est pas question de faire évoluer des populations de plusieurs centaines de milliers d'individus pendant plusieurs dizaines de générations.

D'autre part, le programmeur a toujours quelques connaissances sur le comportement attendu du robot : l'expérience à réaliser nécessite-t-elle de la mémoire ? quels sont les capteurs qui vont être utiles pour cette expérience ? une capacité d'adaptation est-elle nécessaire ? Ceci peut entraîner la mise en place d'une architecture suffisamment satisfaisante pour espérer conduire à une solution du problème. Ainsi, par exemple, si de la mémoire est nécessaire au robot, il est clair qu'un réseau à propagation directe ne sera pas suffisant, et qu'il faudra utiliser un réseau avec des connexions récurrentes. Dans ce contexte, il serait maladroit de laisser un algorithme généraliste "découvrir" ce que nous savons a priori, ce qui élimine les approches totalement "boite noire" de l'optimisation de l'architecture du réseau.

Même si aucun calcul précis n'a été effectué sur le sujet, nous avons essayé de tracer une courbe explicitant notre point de vue sur la question (fig. 3.7) : bien qu'il existe de meilleures solutions dans des espaces de recherche laissant libre l'architecture du réseau, il nous semble peu probable qu'un algorithme évolutionnaire à ressources limitées puisse arriver à les découvrir du fait de la trop grande taille de l'espace de recherche. On risque alors de n'obtenir que des solutions sous-optimales.

A l'inverse, le choix de ne faire évoluer que les poids d'une architecture fixée permet d'utiliser des algorithmes évolutionnaires très performants (les Stratégies d'Evolution, voir section 3.2.4.2), et doit permettre de trouver, dans les limites fixées par l'architecture, une solution quasi-optimale en un temps raisonnable. Cependant, reste au programmeur le choix, crucial, de l'architecture.

Nous l'avons vu, certaines caractéristiques de la tâche à accomplir peuvent guider ce choix, donnant des indications sur le type d'architecture. Mais cela n'aide pas à choisir le complexité voulue dans le type en question : si l'on a choisi un perceptron multi-couche, faut-il 2 ou 3 couches cachées (ou plus encore) ? et combien de neurones par couche (à part les couches d'entrée et de sortie, pour lesquelles le nombre de neurones est imposé par le problème) ?

Il semble clair que la complexité du réseau de neurone doit dépendre de la complexité de la tâche demandée au robot ... ce qui n'apporte pas d'information quantitative ! On peut alors avoir recours à des principes généraux, tel le principe du rasoir d'Occam, la principe de la description de taille minimale, le principe de bon sens du *keep it simple*, qui, tous, plaident pour la complexité minimale suffisante pour résoudre le problème posé. Pour des tâches simples, ce principe et quelques essais avec des architectures connues

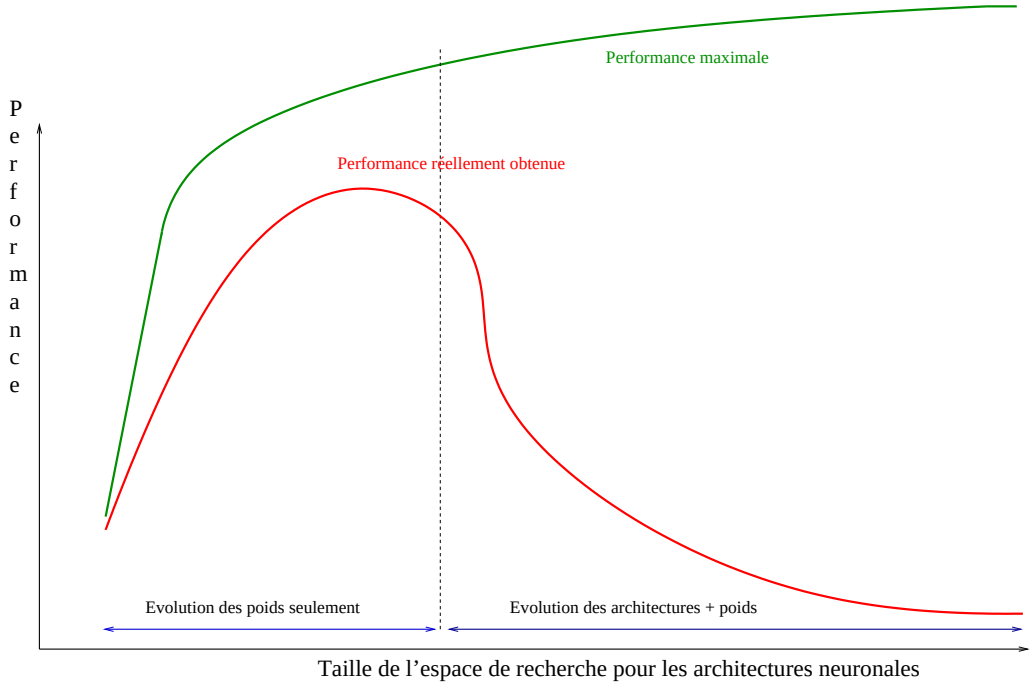


FIG. 3.7 – Comportement probable d'une recherche évolutionnaire dans l'espace des réseaux de neurones : il n'est pas sûr qu'un espace plus grand permette d'obtenir de meilleurs résultats en un temps raisonnable

pour leurs capacités de généralisation et leur robustesse permettent de s'en tirer, mais se repose alors le problème du passage à l'échelle, de la décomposition de tâches complexes en plusieurs tâches plus simples, comme proposée par Brooks 2.2 : ce sera justement l'objet de l'approche par superviseurs que nous proposons et décrivons en détail dans le chapitre 5.

En résumé, nous avons donc choisi d'utiliser des architectures connues (perceptron multi-couche, et sa modification récurrente proposée par Elman [Elman, 1990] dite réseau de type Elman) dont nous optimisons les poids par stratégies d'évolution, comme briques de base de nos contrôleurs neuronaux, les architectures plus complexes étant constituées d'enchaînements de telles briques, soit obtenus de manières elles aussi évolutionnaires (chapitre 5), soit imbriquées à la main dans le cadre de l'architecture d'anticipation proposée au chapitre 6.

3.4 L'évaluation des performances en robotique évolutionnaire

Une fois choisie l'architecture du réseau, et mis en place l'algorithme (voir section 3.2.5), il reste toutefois une dernière étape de conception avant de pouvoir commencer l'évolution : l'écriture de la fonction fitness. Cette fonction doit répondre à plusieurs objectifs. D'une part, son optimum doit être atteint aux points solutions du problème de robotique initialement posé. Mais d'autre part, elle doit aussi permettre l'évolution graduelle à partir de contrôleurs aléatoires – et donc incapables de réaliser la moindre tâche – vers les-dites solutions. Ainsi, une fonction fitness qui vaudrait 1 lorsque le robot réussit sa tâche et 0 sinon ne permettra jamais d'obtenir le moindre résultat ! L'évaluation de la performance des individus revêt donc une importance capitale dans la réussite ou non de l'expérience.

La façon de décrire la fitness peut être explicitée dans un espace à trois "dimensions" [Nolfi et Floreano, 2000]. Différentes positions de fitness dans cet espace (voir figure 3.8) peuvent aboutir à un résultat similaire du comportement du robot. Cependant, des résultats complètement différents, voire même totalement inattendus, peuvent être obtenus simplement en changeant la configuration de la fitness, par exemple en laissant plus de liberté au robot pour parvenir à son but. Nous allons maintenant passer en revue les trois dimensions, essayant de dégager des idées générales pour une conception intelligente d'une fonction fitness pour une problème donné.

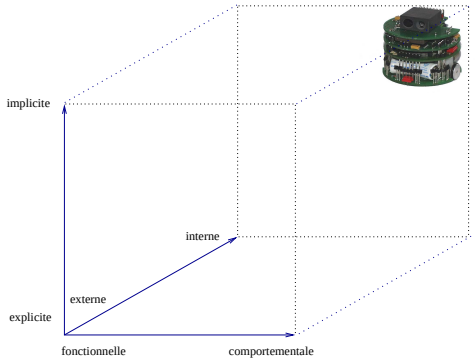


FIG. 3.8 – Le paysage de fitness suivant les trois dimensions.

3.4.1 La dimension fonctionnelle - comportementale

Pour illustrer cette dimension, prenons l'exemple très simple [Jacobi, 1997] d'un labyrinthe en T, où le robot dans sa position de départ est éclairé par une lumière soit sur sa gauche, soit sur sa droite, qui lui indique la direction qu'il devra prendre une fois arrivé

au bout du couloir. La lumière s'éteint dès que le robot l'a dépassée. Une fitness *fonctionnelle* correspondrait à une mesure réalisée sur le contrôleur lui-même, en l'occurence ici un réseaux de neurones. On pourrait par exemple étudier la dynamique du réseau, plus précisément les capacités de mémorisation de tel ou tel neurone, permettant au robot de prendre ou non la bonne direction au bout du couloir. A l'inverse, la dimension *comportementale* est quant à elle totalement liée au comportement désiré du robot. Ainsi, pour cette expérience, on va simplement noter si le robot a pris le bon chemin en fonction du côté où la lumière lui était apparue, sans se préoccuper de la façon avec laquelle il a procédé.

Utiliser une fitness fonctionnelle revient souvent à imposer des biais à l'évolution (par exemple, imposer que tel ou tel neurone serve de mémoire au côté où est apparu la lumière interdit d'autres types de mémorisation), et peut se révéler pénalisant si on demande ensuite au robot d'autres tâches simultanément.

Mais inversement, quelques expériences que nous avons réalisées dans le contexte du couloir en T de Jakobi montrent que le fait d'utiliser une fitness comportementale peut induire des comportements tout à fait inattendus. En effet, si cette expérience semble tout à fait indiquée pour étudier les capacités de mémorisation de tel ou tel réseau, nous nous sommes rendus compte que certaines solutions, loin de mémoriser quoi que ce soit, se contentaient de modifier la direction du robot sous la lumière de telle sorte qu'il "rebondisse" contre les murs (sans les toucher !) exactement le nombre de fois nécessaires pour se retrouver du bon côté une fois atteint le bout du couloir – et ainsi prendre le bon embranchement. Bien sûr, ceci n'était possible que parce que la longueur du couloir était fixe, mais il s'agit d'un exemple qui illustre bien d'une part l'opportunisme de l'évolution, et d'autre part la nécessité d'analyser en détail les résultats obtenus avant de tirer des conclusions hâtives.

3.4.2 La dimension explicite - implicite

Cette dimension décrit le fait que la fitness dépend directement de variables accessibles au robot ou au contraire est un renforcement arrivant après un certain temps et sans lien apparent avec les variables (du robot ou même du programmeur).

Pour rendre ce propos plus . . . explicite, prenons l'exemple de l'expérience d'exploration d'environnement avec une contrainte d'énergie [Floreano et Mondada, 1996]. Il s'agit pour le robot d'éviter les obstacles dans l'environnement tout en gérant une batterie (virtuelle). Le robot doit à la fois explorer l'environnement mais aussi aller se recharger avant d'avoir épuisé son énergie. Dans ce cas, deux façons existent de coder cette fitness, soit de façon explicite en privilégiant les robots étant à une distance de plus en plus petite pour finir sur la zone de batterie lorsque le besoin s'en fait sentir (dans ce cas là, la distance du robot à la zone de recharge sera explicitement codée), et soit de façon implicite en ne spécifiant tout

simplement rien dans la fitness ayant un rapport avec la batterie. Dans ce dernier cas, la sélection va s'opérer naturellement puisque les robots n'ayant pas réussi à aller recharger leur batterie vont "vivre" moins longtemps, et les individus développant des compétences pour recharger leur batterie seront sélectionnés en priorité.

3.4.3 La dimension externe - interne

Les robots possèdent leurs propres capteurs, ceux-ci ne permettent d'avoir qu'une vision partielle du monde qui les entourent. Dans le choix des termes qui composent la fitness, on peut faire en sorte, afin de guider le robot vers le but désiré, d'utiliser des informations que le robot seul ne pourrait pas obtenir. Prenons encore une fois un exemple concret pour illustrer ce propos. Considérons un environnement contenant un cylindre posé à l'intérieur de l'arène, et supposons que nous voulons que le robot se positionne à coté du cylindre. La formulation externe pour la résolution de cette expérience consisterait à minimiser la distance entre le robot et le cylindre. Par opposition, la formulation interne consiste à décrire la façon dont le robot va ressentir la présence du cylindre sur son côté, avec tel ou tel capteur activé par exemple. Si la formulation externe du problème reste très facile à implémenter lorsque l'environnement et le robot sont simulés, il n'en est pas de même lorsque l'évolution du robot doit être exécuté dans le monde réel. Bien entendu, il est en théorie possible de placer une caméra au dessus de l'arène pour déterminer les positions des différents robots, mais ceci n'est cependant pas facile à mettre en oeuvre. Lorsque l'on souhaite mettre l'accent sur le passage au robot réel, comme Nolfi et Floreano [Nolfi et Floreano, 2000] il convient de privilégier des formulations internes des fitness, qui permettent ainsi à tout instant au robot de l'évaluer in situ.

3.4.4 Exemple : l'évitement d'obstacle

La robotique autonome, par définition, souhaite obtenir du robot un comportement non-supervisé. La liberté du robot doit au maximum être préservée. C'est pour cela, que la fitness idéale lorsqu'on parle de robotique autonome est de créer une fonction fitness qui soit à la fois comportementale, implicite et interne (figure 3.8) afin de laisser libre court à l'opportunisme de l'évolution artificielle.

Prenons l'exemple de la tâche de base de tout robot autonome digne de ce nom : l'évitement d'obstacle. La fonction performance d'évitement d'obstacle présentée ici [Nolfi et Floreano, 2000] est adaptée au robot Khépéra présenté dans le chapitre suivant. L'objectif de cette fitness est de faire aboutir un comportement pour le robot lui permettant d'aller relativement vite dans son environnement en évitant de rester longtemps près des murs (dans le but de ne jamais les heurter).

La fitness proposée par Floreano et Mondada [Floreano et Mondada, 1994a] est la

suivante :

$$fitness = V(1 - \sqrt{\Delta v})(1 - i) \tag{3.12}$$

- V est la somme des vitesses des deux roues non signées, afin de permettre aussi bien la marche avant que la marche arrière ;
- Δv est la valeur absolue de la différence des vitesses signées (chaque vitesse est comprise entre -0,5 et 0,5 ;
- i est l'activation maximale parmi l'ensemble des capteurs que le Khépéra possède.

Le premier terme de la fitness favorise une vitesse rapide du Khépéra, le deuxième terme défavorise fortement tout individu qui aurait tendance à tourner sur lui-même, et le dernier terme a pour but d'empêcher les individus de heurter les obstacles en les poussant à s'en éloigner. En fait, ce dernier terme va avant tout défavoriser les individus tentés par un comportement "main courante" qui consisterait à aller vite en effectuant simplement le tour de l'arène tout proche des murs. Avec ce terme, une dimension exploratoire est ajoutée dans le comportement final désiré pour les robots.

Les résultats d'une évolution artificielle basique permettent de constater les éléments suivants lors des générations initiales :

- Dans les premères générations, la plupart des individus tournent sur place. Mais en l'absence du deuxième terme ci-dessus, les individus tournant sur place très rapidement ont une excellente performance ...
- Les meilleurs individus de la population avancent, mais très lentement
- L'évitement d'obstacle n'est pas efficace (la plupart des robots qui avancent finissent dans un mur).
- La performance dépend du point de départ, en effet, la position du robot initiale est aléatoire et les individus plus éloignés des murs accumuleront plus de performance par rapport aux autres, car ils ne savent toujours pas éviter les murs.

Après quelques générations (une vingtaine), on constate que :

- Les robots évitent les obstacles
- Ils ne tournent plus sur place (grâce au deuxième terme de la fitness).

Leur vitesse va ensuite s'accélèrer progressivement à travers les générations, pour atteindre, en une centaine de génération, un palier qui, en réel (Mondada et Floreano [Floreano et Mondada, 1994a] ont effectué cette première évolution uniquement sur robots réels), n'est pas la vitesse maximale. Par contre, le simulateur ne prenant pas en compte l'inertie des robots, la vitesse atteinte lors d'évolutions en simulation est nettement plus grande.

3.5 Conclusion

Nous avons présenté dans ce chapitre les ingrédients de base de la robotique évolutionnaire, d'une part les réseaux de neurones, utilisés comme modèles pour le contrôleur, d'autre part les algorithmes évolutionnaires, outils de mise au point basés sur le darwinisme artificiel. Nous avons vu que l'effort du programmeur était déplacé de la décomposition de la tâche en un ensemble hiérarchique de sous-tâches d'un point de vue structurel, préalable nécessaire par exemple à l'utilisation de l'architecture de subsomption, à la définition d'une fonction performance, ou d'une suite de fonctions de performances permettant d'amener le robot à accomplir la tâche désirée, c'est-à-dire une décomposition fonctionnelle de la tâche souhaitée. Nous allons voir dans les chapitres suivants des exemples d'utilisation de ces concepts, dans lesquels nous allons tout d'abord (ré-)introduire une notion de modularité, mais en laissant l'évolution trouver les dépendances et hiérarchies entre les divers modules (chapitre 5), puis au sein de laquelle nous allons spécifier un mécanisme d'anticipation devant permettre une plus grande adaptation aux changements d'environnement (chapitre 6).

Chapitre 4

L'environnement matériel et logiciel

Dans ce chapitre, nous allons présenter brièvement les outils tant matériels que logiciels que nous avons utilisés dans cette thèse.

La robotique évolutionnaire a longtemps été agitée par le débat "Simulation vs expérimentation sur robots réels" [Nolfi *et al.*, 1994b]. Les arguments pour l'expérimentation réelle sont bien connus : c'est la seule manière de prendre en compte tous les aléas inévitables (glissements des roues, bruits des capteurs, lumière ambiante, ...), puisqu'aucun simulateur, aussi précis soit-il, ne peut espérer être assez fidèle pour supprimer le "fossé de la réalité" (*the reality gap*) qui fera capoter une expérience, pourtant réussie en simulation, lors de son transfert sur le robot réel (on se souvient des énormes ressources de calcul mobilisées par Karl Simms pour tenter de simuler de manière réaliste les mouvements de corps dans des fluides [Sims, 1994], mais qui, de l'avis unanime, étaient encore très loin de la réalité).

Les arguments pour réaliser les évolutions de contrôleurs en simulation plutôt que sur le vrai robot sont de fait basés uniquement sur les contraintes matérielles qu'impose le robot réel : temps d'expérience, tout d'abord, puisqu'une expérience, même dans le cas d'une tâche aussi simple que l'évitement d'obstacle, prend plusieurs jours [Nolfi *et al.*, 1994b] ; les modalités pratiques ensuite, puisqu'il faudra, pour des expériences un peu plus complexes, compter avec la remise en place de l'environnement (par exemple dans le cas de l'expérience du ramasseur de poubelles décrite section 5.2).

C'est pourquoi tous les travaux récents en robotique évolutionnaire effectuent l'évolution en simulation, en prenant toutefois soin d'ajouter du bruit aux capteurs et même éventuellement aux actuateurs du robot, afin de minimiser les "surprises" lors du passage au robot réel. Et les travaux de N. Jacobi [Jacobi, 1998] ont même permis de relativiser l'impact de la pauvreté des simulations sur les résultats – pauvreté d'ailleurs

nécessaire pour garder des temps de calcul raisonnables.

Il n'en reste pas moins que la validation ultime pour des travaux en robotique évolutionnaire reste le passage sur robot réel, qui seul peut démontrer la justesse d'une hypothèse ou la pertinence d'une approche. C'est pourquoi il est très important, même lorsque l'on effectue des évolutions en simulation, d'avoir une plate-forme matérielle cible et de ne pas simuler un robot virtuel totalement irréaliste.

Dans cette optique, et même si peu des expériences présentées dans cette thèse ont pu de fait être validées en réel (voir section 5.7.4 pour une discussion plus approfondie des problèmes rencontrés), nous avons dès le début de cette thèse opté pour la plate-forme matérielle la plus utilisée par les chercheurs du domaine, le robot Khepera (section 4.1).

Nous allons commencer par le présenter rapidement, puis nous évoquerons les deux plate-formes logicielles que nous avons successivement utilisées dans le déroulement de cette thèse.

4.1 Le Khépéra

4.1.1 Le robot

FIG. 4.1 – La base du Khépéra version 2.

Toutes les expériences de cette thèse (en réel ou en simulation) ont donc été effectuées sur le robot Khépéra. Il s'agit d'un robot très petit (de diamètre 55mm), peu cher (surtout par rapport aux robots disponibles lors de sa sortie au début des années 1990), et avant tout très pratique, car il peut être installé sur un simple bureau et sans compétences particulières en electronique ou en mécanique. Le modèle *Khepera II* est représenté sur la figure 4.1 – la différence avec le modèle I, que nous avons utilisé dans une grande partie de la thèse, tient de manière visible dans la coque de protection ajoutée autour du robot (la

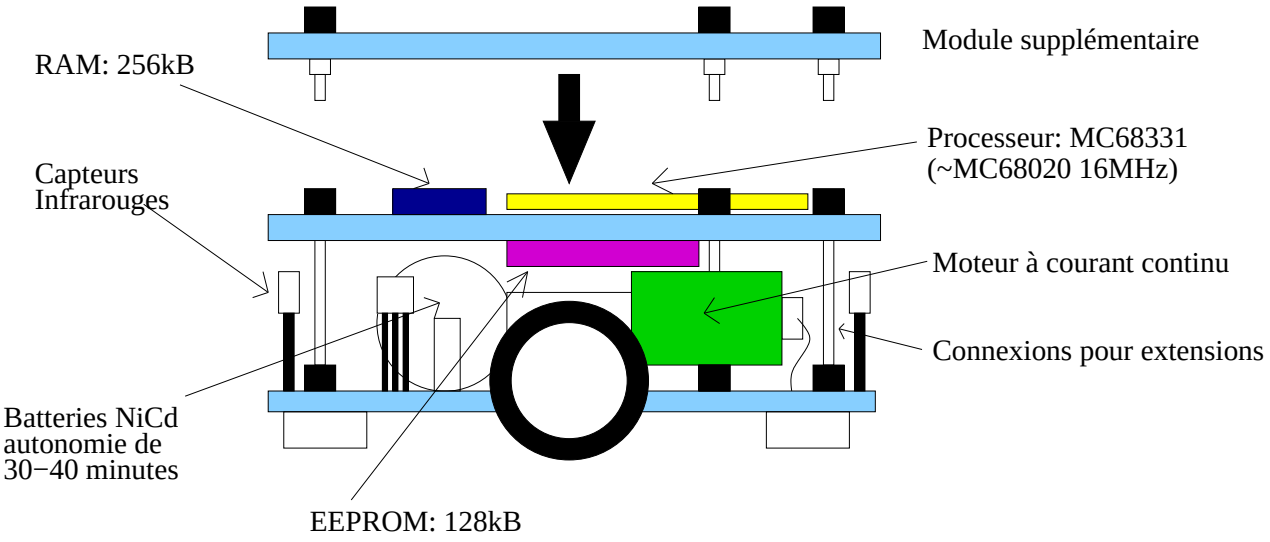


FIG. 4.2 – Le schéma du Khépéra vu de profil.

figure 4.4 montre le modèle I), et de manière moins visible dans l'amélioration importante de la qualité des capteurs infra-rouges.

Quelques spécifications techniques sont données sur la figure 4.2, représentant un Khepera vu de profil. Le Khépéra comporte 2 moteurs électriques, qui sont associés à deux réducteurs et permettent de mettre en mouvement le Khépéra. La rotation s'effectue en donnant des vitesses différentes aux deux moteurs.

Le module de base comporte également 8 capteurs infrarouges, dont la position est détaillée sur la vue de dessus du Khépéra montrée figure 4.3. Ces capteurs permettent de détecter les obstacles assez proches du robot, le Khépéra commençant à voir les obstacles aux environs de 7cm de distance – dans le meilleur des cas, si les murs sont clairs.

Deux capteurs sont situés sur le côté gauche du robot, deux sur le devant du robot, deux sur le côté droit, et deux sont situés à l'arrière du robot. On peut regretter que les capteurs ne soient pas mieux répartis autour du robot, mais ils sont clairement configurés pour éviter les obstacles au mieux lorsque le robot avance.

Ces capteurs peuvent fonctionner en deux modes distincts :

- le *mode actif* permet de détecter les obstacles : des rayons infrarouges sont envoyés sur les éventuels obstacles à proximité, ces rayons sont ensuite plus ou moins réfléchis suivant la couleur et l'éclairage des murs, et les capteurs infrarouges renvoient alors une valeur dépendant de la présence plus ou moins proche des obstacles.
- le *mode passif* sert à mesurer la lumière ambiante, et permet de détecter une source lumineuse ou une source de chaleur à une distance relativement importante (par rapport à la taille du robot).

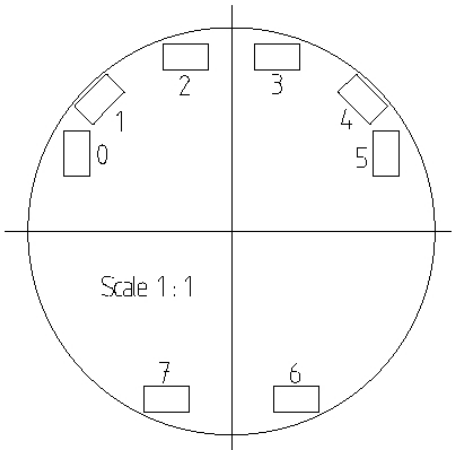
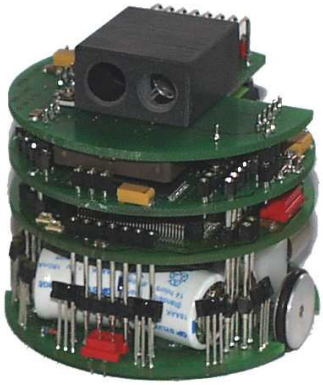
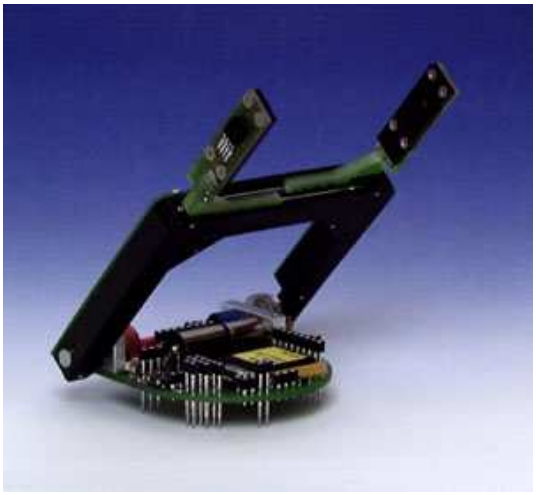


FIG. 4.3 – Les positions des capteurs sur le Khépéra 2, image extraite de sa documentation.

D'autre part, ce robot a été conçu de façon modulaire, et de nombreux modules disponibles peuvent être ajoutés et empilés sur la base.



Le Khépéra muni de la caméra à 1 dimension



Le bras manipulateur du Khépéra

FIG. 4.4 – Quelques modules additionnels du Khépéra.

Dans le cadre de cette thèse, nous avons utilisé le module caméra à 1 dimension (figure 4.4 à gauche) qui permet de "voir" 64 pixels en niveaux de gris décrivant la bande horizontale située à la hauteur de la caméra 1D, et permet par exemple de détecter sans problème des cylindres noirs, ou bien encore des codes barres grossiers situés à une faible distance du Khépéra.

Le module *gripper* (figure 4.4 à droite) a également été utilisé (voir par exemple section

5.2), il permet de saisir des objets de faibles poids et de les transporter sur une autre partie de l'environnement.

4.1.2 La commande du Khépéra

Il existe trois façons de commander le Khépéra :

- En téléchargeant directement un programme sur le Khépéra après l'avoir cross-compilé pour le processeur interne du robot, ou programmé directement en assembleur de ce processeur. Le principal problème de cette approche est la très faible mémoire RAM embarquée sur le robot (de 16Ko à 256 Ko selon les versions). Ainsi, seuls des programmes très simples (e.g. de type évitement d'obstacle) peuvent être implémentés. De plus, dans ce cas là, le robot fonctionne uniquement sur ses batteries et a une autonomie réduite (de l'ordre de 45 minutes au maximum).
- En utilisant un cordon reliant le port série de l'ordinateur et le Khépéra. Il est alors possible d'envoyer sur le port série des ordres directement compréhensibles par le robot, comme par exemple les vitesses moteurs, ou de le questionner sur les valeurs de ses capteurs (infrarouges, caméra, ...). Par rapport à un programme transféré directement sur le Khépéra, cette méthode présente deux avantages : elle permet de commander le robot avec de plus importantes puissances de calcul et capacités mémoire (celles de l'ordinateur), et les problèmes d'autonomie disparaissent car le courant continu passe également à travers le cordon. Par contre, un des principaux inconvénients est la presque impossibilité de réaliser des expériences impliquant plusieurs robots. En effet, les cables s'emmêlent rapidement et empêchent les robots de se mouvoir normalement (même si l'équipe de Dario Floreano a réussi à mettre au point un dispositif permettant de mettre en action deux robots sans problème de cables emmelés).
- La dernière possibilité est d'utiliser le module radio, récemment disponible pour le Khépéra, qui permet de commander les robots à distance et surtout sans fil. Dans ce cas, le problème d'autonomie ressurgit, mais cela semble être la solution idéale dans le cas d'une expérience utilisant plusieurs robots Khépéra.

Lors des expériences effectuées en réel, nous avons utilisé la solution avec cordon. En effet, des modules radio n'ont été acquis que très récemment, et n'ont pas été encore utilisés. De plus cette méthode permet d'éviter les problèmes d'autonomie et de recharges régulières à faire des batteries.

Comme il a été dit dans l'introduction de ce chapitre, le fait de réaliser des essais en robot réel représente un temps conséquent et non négligeable, c'est en partie pourquoi nous avons réalisé la totalité de nos expériences en simulation, la validation sur robot réel n'étant effectuée dans le meilleur des cas, (voir section 5.7.4) qu'a posteriori, sur le(s) meilleur(s) individu(s) obtenu(s) par l'évolution en simulation.

4.2 Les simulateurs

Deux simulateurs différents ont été utilisés durant le déroulement de la thèse, *EvoRobot*, basé sur le simulateur fourni par S. Nolfi et avec lequel la plupart des expériences rapportées dans l'ouvrage [Nolfi et Floreano, 2000] ont été effectuées, et *WOB*, développé au sein de l'équipe TAO. Nous allons les décrire tour à tour, donnant leurs caractéristiques, et décrivant nos contributions à leur développement pour la réalisation des expériences décrites dans les chapitres 5 et 6 de cette thèse.

4.2.1 Evorobot

La première partie du travail de la thèse (fin 2001-2002), ainsi d'ailleurs que mon travail de DEA, et les expériences réalisées dans le cadre du MODEX de robotique de l'École Polytechnique (voir section 5.7.4 pour un exemple) ont été effectuées à partir d'un logiciel basé sur le simulateur de Khépéra Open-Source *Evorobot*, créé par Stefano Nolfi, doté d'une interface graphique et fonctionnant sous Windows.

Mes contributions ont été tout d'abord de porter le logiciel sous Linux (en utilisant la librairie *OpenGL* pour le graphisme), puis ensuite de l'interfacer avec la librairie d'évolution artificielle *EO* (*Evolving Object*, également diffusée en Open-Source) [Keijzer *et al.*, 2002] développée par un ensemble de chercheurs européens au sein du réseau d'excellence *EvoNet*. En effet, les routines d'évolution présentes dans le logiciel original ne comportait que l'évolution de chaînes de bits – et il est maintenant unanimement reconnu que l'évolution de paramètres réels (les poids des contrôleurs neuronaux dans le cas présent) est beaucoup plus efficace quand une représentation réelle est utilisée. La librairie EO offre toute la souplesse nécessaire à de telles évolutions, avec en particulier la possiblité d'utiliser les mutations gaussiennes auto-adaptatives dont la puissance n'est plus à démontrer dans le cadre de l'optimisation paramètrique (de variables réelles, donc).

Cependant, la très faible modularité dans la conception du simulateur initial (écrit en C, et visiblement modifié à de très nombreuses reprises sans vision d'ensemble très structurée), entraînait d'énormes difficultés pour y implanter un contrôleur différent de ceux qui étaient fournis, ne serait-ce qu'en terme d'architecture des réseaux de neurones : seuls les réseaux de neurones fournis pouvaient être réutilisés facilement.

L'adaptation sous Linux a profité d'une conception un peu plus orientée objet, du moins en ce qui concerne la partie contrôleur robotique, qui a été totalement repensée. En particulier, tous les contrôleurs dérivaient cette fois d'une seule et unique classe, ce qui permettait d'ajouter relativement facilement de nouveaux contrôleurs. De plus, les méthodes de calculs de ces contrôleurs (comme les réseaux feedforward, récurrents, ...) ont été reprogrammées. À noter qu'il existe de nombreuses librairie de réseaux de neurones artificiels, très facilement configurables grâce à une conception très orientée objet.

Cependant, nous avons choisi ici d'écrire les primitives de base concernant les réseaux de neurones en langage C afin de pouvoir réellement les optimiser pour obtenir une vitesse d'exécution la plus rapide possible. Une des conséquences de ce choix est que la conception d'une nouvelle architecture prend un peu plus de temps. Mais le gain en temps calcul lors d'une évolution artificielle compense largement ce léger inconvénient, et ne diminue en rien le gain en facilité de configuration de l'architecture des réseaux, entièrement paramétrables à l'exécution (nombre d'entrées et de sorties, nombre de couches cachées, nombre de neurones par couches cachées, ...) via le parser utilisé par la librairie EO.

De plus, des éléments permettant de prendre en compte les modules additionnels du Khepera ont été ajoutés : programmation sommaire et grossière du gripper, prise en compte détaillée et réaliste de la caméra 1D. Néanmoins quelques difficultés ont persisté, qui étaient très difficilement corrigeables :

- La gestion des capteurs était simplifiée dans le logiciel de S.Nolfi, afin de gagner du temps. Mais du coup, il était impossible, par exemple, de détecter des petites ouvertures dans un mur continu de l'environnement, et certaines expériences impliquant de telles petites ouvertures, que pourtant les capteurs infrarouges détectaient parfaitement sur le robot réel, devenaient impossible à mettre en oeuvre en simulation.
- L'implantation de superviseurs (voir section 5.5) contrôlant d'autres architectures de supervision, était pratiquement impossible, ou alors aurait demandé pour chaque nouveau superviseur un gros effort de programmation spécifique à la librairie – ce qui est précisément le contraire de l'esprit ayant présidé au développement de l'architercture "superviseur".
- La gestion multi-robots était particulièrement difficile à mettre en oeuvre pour deux robots, et complètement impossible lorsque les robots étaient au nombre de trois ou plus.
- L'introduction de modèles de robots autres que les Khépéra était absolument à exclure, toute la conception était en effet axée uniquement sur ce type de robot.
- L'implantation en OpenGL avait repris exactement l'implantation graphique utilisée sous Windows à l'origine, et n'était pas dissociable du simulateur. La conséquence la plus grave (outre que la maintenance en était rendue difficile) était qu'il n'existait pas de version sans graphique, ce qui interdisait de lancer le programme sur les clusters de calcul, ceux-ci étant habituellement dépourvus de toutes librairies d'affichage graphique.

4.2.2 Le simulateur WOB – World in a Bottle

Afin de ne plus être freiné par ces problèmes de modularité logicielle, et fort de l'expérience obtenue par la mise au point du précédent, un nouveau simulateur a été totalement repensé et recodé durant le premier semestre de l'année 2003. Sa conception

et sa réalisation ont été réalisées en collaboration avec Jérémie Mary, un autre membre de l'équipe TAO, et a donné naissance au simulateur *World in bottle* (*WOB*).L'environnement a été totalement repensé, et dans *WOB*, tant les éléments de l'environnement (murs, cylindres, ...) que les robots eux-même sont considérés comme un ensemble de formes élémentaires consituant l'environnement, et pouvant être fixes ou mobiles. Ainsi, tout nouveau type de robot peut être facilement introduit, chaque robot possèdant ses propres attributs, et ses propres réactions et interactions avec l'environnement. Ainsi, le type, le nombre et les positions des capteurs, de même que la façon dont le robot se déplace, peuvent être redéfinis aisément, et chaque robot peut alors interagir de façon propre aussi bien avec les éléments de l'environnement qu'avec les autres robots (puisqu'ils ne sont finalement que des éléments particuliers de l'environnement). Le nombre de robots utilisables en même temps est donc cette fois illimité (sauf par la puissance de calcul, puisque bien sûr le temps calcul augmente quadratiquement avec le nombre d'objets présents dans l'environnement).

Dans *WOB*, les contrôleurs robotiques sont cette fois totalement découplés du simulateur. L'interface entre la partie simulateur et la partie contrôleur étant clairement spécifiée, il est très facile de créer et d'utiliser de nouveaux contrôleurs. De plus, chaque contrôleur possède désormais un nom, ce qui permet de constituer une librairie de contrôleurs pouvant être appelés à tout moment par le simulateur : la conception de superviseurs est devenue une tâche très facilement réalisable, de même que la supervision d'autres architectures de supervision qui supervisent elles-mêmes d'autres architectures .... Enfin, les capteurs infrarouges sont maintenant beaucoup mieux simulés, et intéragissent de façon réaliste tant avec les autres robots qu'avec tous les éléments de l'environnement.

La librairie d'évolution artificielle *EO* est toujours utilisée et la librairie OpenGL est toujours la base de tout le graphisme, mais elle peut maintenant être désactivée à la fois lors de la compilation, ce qui permet de lancer le simulateur en mode non graphique (sur des clusters par exemple), mais aussi lors de l'exécution, ce qui permet un gain de temps calcul non négligeable lors des évolutions tout en autorisant à tout moment une éventuelle visualisation du meilleur résultat déjà obtenu.

D'autres éléments ont été encore ajoutés par rapport au précédent simulateur utilisé :

- la possibilité de prendre le contrôle des robots par des raccourcis claviers à la fois en simulation et en réel.
- un algorithme de rétro-propagation de l'erreur (voir chapitre 6) compatible avec les architectures de réseaux de neurones existants.
- la possibilité d'utiliser du baldwinisme ou du lamarckisme lors de l'évolution (cf. chapitre 6).
- une gestion beaucoup plus fidèle et précise du gripper utilisé par le Khépéra.

L'ensemble de ces travaux de développement logiciel constitue la base sans laquelle

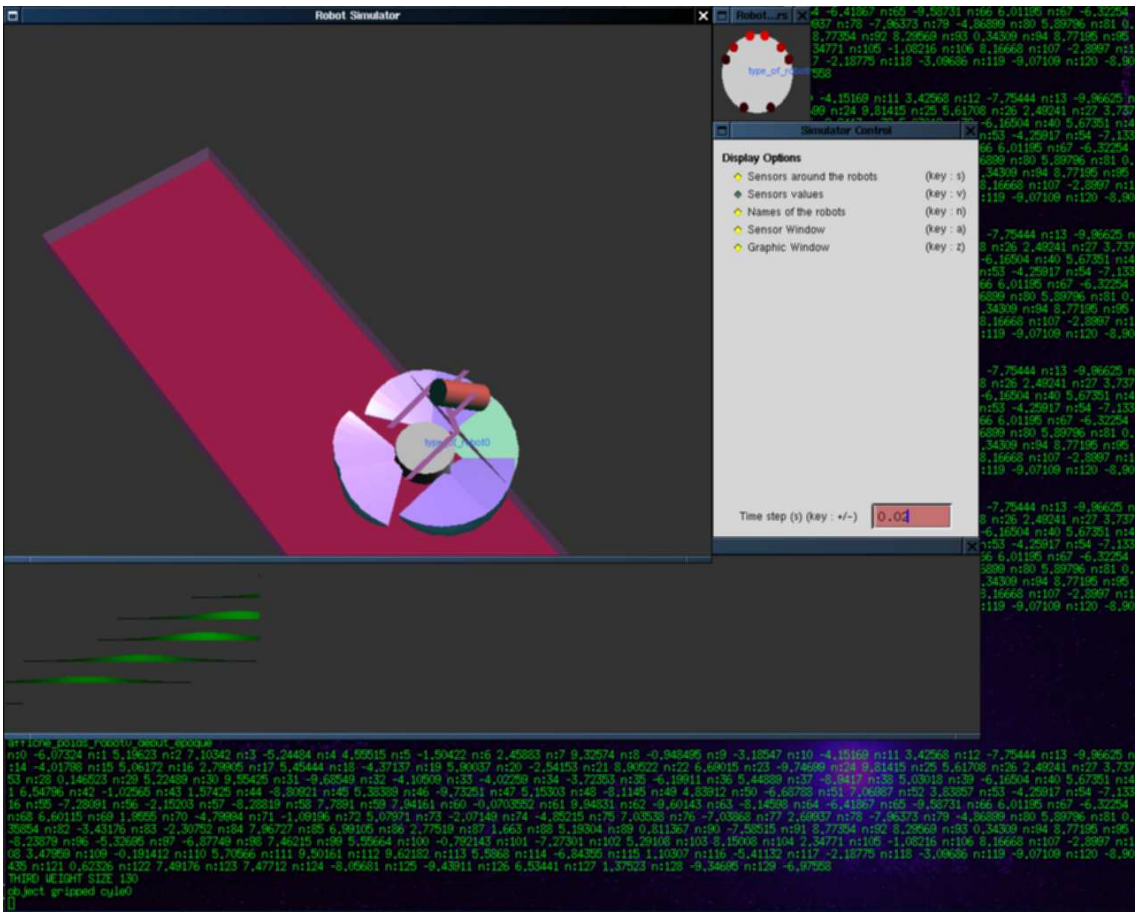


FIG. 4.5 – Un screenshot du simulateur WOB.

aucune des expérimentations qui vont être présentées dans la suite n'auraient pu être réalisées. Tous ces développements sont par ailleurs disponibles en code source pour une éventuelle réutilisation par d'autres chercheurs. Signalons que les élèves du MODEX de l'École Polytechnique continuent d'utiliser cet ensemble logiciel bien que j'ai cessé d'encadrer ces Travaux Pratiques depuis 2003.

Chapitre 5

Les contrôleurs experts en robotique évolutionnaire

Ce chapitre présente nos premières contributions à la robotique évolutionnaire, qui consistent à essayer de trouver un compromis entre l'approche de subsomption de Brooks et l'approche totalement boîte noire adoptée en général par la robotique évolutionnaire. L'utilisation de "contrôleurs symboliques", dont les sorties ne sont pas forcément directement reliées aux actuateurs du robot, mais à des contrôleurs de plus haut niveau programmés à la main, est la première approche proposée. Elle sera ensuite généralisée à la prise en compte de contrôleurs de haut niveau, pouvant eux-mêmes avoir été construits par évolution artificielle, pour permettre un meilleur passage à l'échelle lorsque la complexité de la tâche à accomplir pour le robot augmente.

Nous commencerons au préalable, section 5.1, par discuter des problèmes posés par le passage à l'échelle en robotique autonome, en terme de complexité de la tâche à accomplir pour le robot, et suivant les deux points de vue de la modularité et de la quantité d'intervention de l'expert humain nécessaire à l'obtention de résultats satisfaisants. La section 5.2 se concentrera sur des travaux de D. Floreano et S. Nolfi proposant un modèle de réseau modulaire, travaux que nous avons répétés afin de pouvoir les analyser en détail. La section 5.3 présentera les *contrôleurs symboliques*, qui utilisent des primitives prédéfinies de bas niveau pour l'évolution de comportements de plus haut niveau. Nous présenterons ensuite des comportements simples obtenus à l'aide de cette approche (section 5.4). Mais les limitations de cette approche par rapport au passage à l'échelle nous amèneront à introduire les "superviseurs", contrôleurs pouvant à chaque pas de temps choisir parmi des comportements d'une librairie donnée. La section 5.6 proposera une validation de l'approche superviseurs sur une expérience plus complexe, pour laquelle les résultats seront comparés à ceux d'approches classiques de type boite noire. Les problèmes posés par cette

approche superviseur, comme la robustesse par rapport à la généralisation et au choix des primitives de la librairie, seront discutés au vu d'expériences complémentaires. Enfin, la section 5.7 concluera le chapitre avec un exemple de mise en oeuvre sur robot réel de résultats pour lesquels l'approche superviseur s'est révélée indispensable à l'obtention de résultats satisfaisants.

5.1 Le problème du passage à l'échelle

5.1.1 La modularité

Comme nous l'avons vu dans les chapitres précédents, il existe deux grandes familles d'approches pour la robotique autonome.

D'une part, les approches déterministes, requérant une importante intervention humaine, dont le meilleur exemple est l'architecture de subsomption de R. Brooks évoquée dans la section 2.2, dans laquelle les tâches du robot sont décomposées manuellement dans une hiérarchie de sous tâches indépendantes.

À l'opposé, la robotique évolutionnaire mentionnée dans le chapitre 3 est généralement perçue comme une approche de type boite noire : des contrôleurs (par exemple de type réseaux de neurones) sont optimisés en vue d'obtenir le comportement souhaité en utilisant le paradigme darwinien de l'évolution artificielle. Dans ce cadre, le travail du concepteur consiste uniquement à définir la fonction de fitness qui amènera l'évolution à sélectionner les comportements souhaités.

Toutefois, quoique pour des raisons différentes, le passage à l'échelle, en terme de complexité de la tâche à accomplir, de ces deux approches reste très délicat. L'efficacité de l'approche déterministe, basée sur le découpage de la tâche que le robot doit accomplir par un expert, est limitée précisément par le facteur humain : il est tout de même très difficile de décomposer des tâches complexes comportant de nombreuses sous-tâches distinctes dans le cadre de l'architecture de subsomption, dans la mesure où la mise au point des intéractions entre les diverses sous-tâches constitue en soi un problème dont la complexité croit exponentiellement avec le nombre de sous-tâches.

La difficulté du passage à l'échelle en complexité dans le cadre de la robotique évolutionnaire est, elle, liée au choix de la fonction fitness, ou des fonctions fitness successives qu'il faut en général imaginer : le problème du bootstrap (le fait que tous les robots initialisés aléatoirement sont également nuls pour accomplir la tâche souhaitée, empêchant tout simplement l'optimisation de démarrer) fait qu'il faut soit construire une fonction fitness faite de plusieurs composantes délicates à mettre au point, soit construire une suite de fonctions fitness qui va permettre d'obtenir des comportements de plus en plus sophistiqués aboutissant au comportement visé. Dans les deux cas, il s'agit d'un vrai travail de décomposition, certes basé sur les comportements et non sur les interactions

entre différentes parties fonctionnelles du contrôleur comme dans l'architecture de subsomption, mais qui tend à devenir de plus en plus difficile à mesure que la complexité du but visé augmente.

De plus, l'approche de type boite noire de la robotique évolutionnaire rend extrêmement difficile la compréhension et l'interprétation des résultats, que ce soit des succès ou des échecs, interdisant ainsi toute capitalisation d'expertise passée pour une éventuelle réutilisation future sur une expérience différente.

Une approche modulaire a été proposée par Stefano Nolfi et Dario Floreano. Cette approche essaie de palier un des principaux défauts des approches de type boite noire en donnant la possibilité à l'évolution de choisir parmi différents modules. Bien qu'accélérant l'évolution pour certaines tâches, cette approche présente quelques problèmes que nous détaillerons dans la section 5.2.

L'approche que nous proposons dans cette thèse essaye de trouver quelques compromis entre les deux extrêmes mentionnés précédemment. Elle est basée sur la remarque suivante : pour espérer réellement apporter une réponse évolutionnaire au problème du passage à l'échelle, l'un des problèmes les plus critiques en robotique autonome, il faut utiliser le même mécanisme tout au long du chemin de complexité, des tâches primitives à des comportements plus complexes.

D'autre part, la difficulté d'intelligibilité des solutions conçues par l'approche évolutionnaire vient du fait que les "langages" des contrôleurs sont de bas niveau (par exemple, les consignes de vitesses des moteurs du robot). L'utilisation comme sorties de comportements certes encore basiques, mais de plus haut niveau (e.g. tourner à gauche, à droite, aller tout droit), comme proposée dans la section 5.3, devrait nous permettre de mieux comprendre la relation entre les sorties du contrôleur et le comportement résultant. De plus, cette approche nous permettra de construire récursivement des comportements de plus haut niveau à partir de ces simples comportements, permettant de résoudre des problèmes de plus en plus complexes.

Les expériences mises en œuvre dans ce chapitre manipulent les deux aspects décrits précédemment. Mais nous allons tout d'abord dans la section suivante revisiter ces arguments à la lumière de l'importance de l'effort humain prérequis pour le succès de l'une ou l'autre approche.

5.1.2 Inné ou acquis

Les deux tendances opposées en robotique autonome décrites dans la sous-section précédente peuvent également être appréhendées comme les aspects cognitifs "inné vs acquis" au niveau de la mise au point du contrôleur : l'intervention humaine correspond à la partie "innée", et les contrôleurs non explicitement programmés – et donnant d'ailleurs des résultats pas toujours prévus – en étant la partie "acquise".

De ce point de vue, l'architecture de subsomption de Brooks se situe à l'un des extrêmes, côté "inné", puisque toutes les compétences données au robot sont programmées par le concepteur, à savoir les comportements de bases à attribuer au robot ainsi que la façon de les combiner et de les faire interagir de manière à obtenir des comportements complexes. Cette manière de procéder permet d'obtenir de bonnes performances pour des tâches très spécifiques. Cependant, lorsque l'on passe à une tâche plus complexe, il faut en général revoir une bonne partie de l'architecture. Rappelons tout de même que le but initial de Brooks était d'atteindre l'intelligence des insectes [Brooks, 1991], mais qu'il a dû revoir cet objectif à la baisse du fait ... de sa complexité trop importante.

Du côté de l'inné, on trouve également les modèles de sélection d'actions (e.g. Spreading Activation Network [Maes, 1989]. Cependant, de telles approches ont deux points faibles : le premier est qu'une telle architecture est biologiquement discutable, tandis que le second concerne le problème de l'autonomie ; en effet, si le fait de remplacer les réflexes de bas niveau par des décisions à propos des comportements de plus haut niveau (utiliser tel ou tel comportement à tel moment) semble prometteur, la façon de programmer les interactions de tels réflexes dans un monde changeant et au moins en partie inconnu nous semble pouvoir être plus facilement résolu par évolution artificielle que de manière déterministe (cet argument sera plus développé section 5.3).

A l'extrême "acquis" du débat inné/acquis se trouvent la plupart des travaux en robotique évolutionnaire : a priori, tout biais de la part du designer est considéré comme néfaste pour l'opportunisme de l'évolution et l'autonomie du robot. A noter qu'une telle position est également défendue dans le domaine du design optimal [Bentley, 1999], domaine dans lequel de très bonnes solutions souvent inattendues ont pu être obtenues en limitant l'intervention humaine dans le cadre du design évolutionnaire. Il faut toutefois signaler que de nombreuses décisions sont en fait prises par le designer en amont de l'évolution, dont bien sûr le choix de la représentation, qui est crucial dans le domaine du design.

Dans le domaine de la robotique évolutionnaire, l'idée précédente a été mise en valeur par la revisite récente de Tuci, Harvey et Quinn [Tuci *et al.*, 2002] d'une expérience proposée initialement par Yamauchi et Beer [Yamauchi et Beer, 1994] (cette expérience sera décrite en détail dans la section 6.2.2) : le robot doit agir différemment (soit aller vers la lumière soit s'en éloigner) en fonction d'une variable aléatoire cachée. Le robot doit donc apprendre à la première époque l'état de cette variable aléatoire, et agir en conséquence dans les époques suivantes. Alors que dans l'expérience originale, l'architecture du contrôleur était soigneusement conçue pour le problème, l'architecture du contrôleur de Tuci et al. est un réseau de neurones CTRNN (voir encore une fois section 6.2.2) totalement connecté, et donc sans connaissance particulière du domaine, et l'évolution agit sur tous les poids. De plus, Tuci et al. n'utilisent pas de renforcement

explicite. Néanmoins, les résultats obtenus par cette récente approche sont bien meilleurs que ceux obtenus dans l'expérience originale, et, de l'avis des auteurs, ceci est obtenu grâce à leur approche totalement boîte-noire, dans la mesure où aucune modularité n'a pu être constatée dans les réseaux solutions a posteriori.

Néanmoins, alors que les concepteurs ont décidé d'utiliser une architecture modulaire soigneusement mise au point dans la première expérience, l'effort dans l'approche de Tuci et al. a été reporté sur la fonction fitness (par exemple, la zone de récompense sous la lumière doit se produire seulement une fois sur deux, ou bien encore, la récompense pour se diriger vers la lumière doit être supérieure à celle pour la fuir, pour briser la symétrie, ...). Par conséquent, cette approche demande de fait également une grande partie d'inné, puisque, pour la création de la fonction fitness, l'intervention humaine est requise de manière très subtile.

Comme souvent, une bonne solution est sans doute, dans ce débat "inné/acquis", une approche intermédiaire qui serait de doter le robot de comportements innés complexes, issus de la connaissance humaine, mais dans laquelle les choix entre ces comportements et leurs interactions ne nécessiteraient pas d'intervention humaine supplémentaire : l'évolution pourrait alors évoluer aussi bien vers des comportements clairement modulaires (choix d'un comportement inné dans des situations clairement repérables) ou un comportement plus proche de la boîte noire (basculements répétés et rapides entre plusieurs comportements sans rapport apparent avec une situation donnée).

C'est la proposition que nous avons faite (voir section 5.3) : le concepteur va aider l'évolution de contrôleurs complexes en mettant à la disposition de l'optimisation des comportements simples, soit codés manuellement soit obtenus par évolution artificielle, laissant l'évolution arranger ces blocs de construction ensemble. Un important effet de bord est que le concepteur pourra, sans doute, plus facilement comprendre les résultats d'une expérience, grâce à une plus grande intelligibilité des contrôleurs.

Cependant, d'autres solutions intermédiaires dans la perspective inné/acquis ont été proposées il y a quelques années, tels les contrôleurs modulaires développés par Nolfi et Floreano [Nolfi et Floreano, 2000]. Ils constituent un premier pas, en partant de l'approche évolutionnaire "classique" purement boîte noire, vers l'approche que nous avons développée, dans la mesure où l'architecture du contrôleur neuronal est choisie par le concepteur en fonction du problème à résoudre, et c'est l'évolution qui construit simultanément les comportements des modules et le mécanisme de choix entre les modules. Nous allons donc tout d'abord présenter cette architecture en essayant de comprendre ses propriétés et leurs incidences sur les performances du robot.

5.2 Le contrôleur modulaire de Nolfi et Floreano

Le contrôleur modulaire mis au point par Stefano Nolfi et Dario Floreano [Nolfi et Floreano, 2000] utilise deux modules identiques "parallèles". Cependant, avant de décrire son architecture en détail, nous allons présenter la tâche sur laquelle cette achitecture a été utilisée, dans la mesure où de nombreux choix des auteurs sont dictés par le problème à résoudre.

5.2.1 Le ramasseur de poubelles

L'expérience de robotique autonome sur laquelle a été testée l'architecture modulaire présentée section 5.2.2 est celle dite du "ramasseur de poubelles". Une illustration de cette expérience est présentée sur la figure 5.1 : le robot Khepera est dans ce cas équipé d'un bras manipulateur (*gripper*) lui permettant de saisir des objets, et les "poubelles" sont ici des cylindres de taille suffisamment petite pour que le robot puisse les saisir avec le-dit gripper.

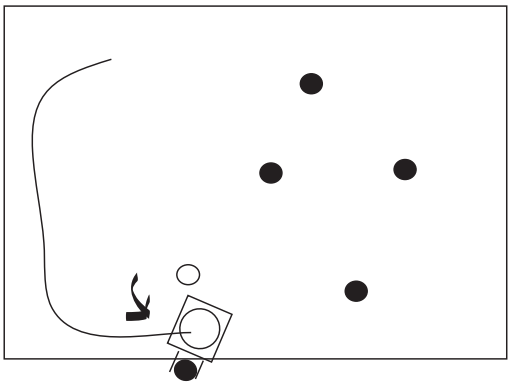


FIG. 5.1 – Expérience du "garbage collector", le robot Khepera doit saisir les objets cylindriques se situant sur l'arène et les déposer à l'extérieur.

À chaque époque, cinq cylindres sont disposés aléatoirement sur l'arène, et le robot doit en saisir un avec le gripper et le déposer en dehors de l'arène (les murs sont suffisamment bas). Le robot dispose de 15 époques pour mettre au maximum 15 cylindres à l'extérieur. La performance du robot est déterminée de la manière suivante : le robot ne marque aucun point tant qu'il n'a pas saisi de cylindre ; lorsque que le robot a saisi un cylindre, et afin de l'inciter à se déplacer, il reçoit une fitness correspondante à celle de l'évitement d'obstacles décrite dans la section 3.4.4 ; enfin, un bonus de 10000 points lui est attribué lorsqu'il parvient à déposer un cylindre à l'extérieur de l'arène. Par conséquent, la fitness maximale qu'un robot peut récolter à la fin des 15 époques correspond à un peu plus de 150000 points, soit 150000 plus l'ac-

cumulation des points marqués entre la saisie du cylindre et son dépot en dehors de l'arène.

Un point important dans le cadre de la suite de ce chapitre concerne la manipulation du gripper : alors que les commandes du moteur du bras manipulateur lui permettent de se positionner au degré près, les auteurs ont choisi de ne considérer au niveau du contrôleur que les macro-actions consistant à ramasser ou à relacher un objet. En effet, l'expérience consistant à manoeuvrer le gripper pour arriver à des macros-actions nécessite une grande dextérité et demanderait un temps d'apprentissage par l'évolution très long : ces manoeuvres sont donc programmées de manière déterministe, et le robot doit juste choisir s'il saisit l'objet situé à proximité et en face de lui, ou s'il relâche l'objet en sa possession – ou seulement s'il choisit de ne faire ni l'une ni l'autre de ces actions.

Au niveau des sorties du contrôleur, deux neurones spécialisés ont été introduits en plus des neurones moteurs dans la couche de sortie du réseau. Si la sortie de ces neurones dépasse un certain seuil, alors le robot va exécuter l'action correspondante (soit prendre l'objet, soit le relâcher). En cas de conflit, il exécutera l'action correspondant à la sortie des deux neurones commandant le gripper ayant la plus grande valeur.

Signalons enfin que, dans le cadre de cette expérience, il est aisé d'imaginer au moins un type de modularité qui devrait fonctionner : supposons que le robot dispose des comportements "exploration de l'environnement pour repérer un cylindre" et "aller le plus près possible d'un mur et ouvrir son gripper", alors une solution au problème du ramasseur d'ordures consiste à programmer le robot de telle sorte qu'il explore tant qu'il n'a pas trouvé de cylindre, et qu'il bascule vers l'autre comportement dès qu'il a chargé un cylindre. C'est ce genre de remarque qui a motivé l'introduction du réseau modulaire.

5.2.2 L'architecture modulaire

L'architecture du réseau modulaire est présentée sur la figure 5.2. Quelques commentaires s'imposent pour expliquer ce schéma assez difficile à comprendre : on constate qu'il existe 4 parties bien distinctes, correspondant aux quatres actuateurs que le contrôleur doit commander : les vitesses des moteurs (gauche et droit), et les commandes du gripper décrites dans la sous-section précédente (ramassage ou relâche d'un objet). Chacune de ces parties comporte 4 sorties reliées à l'ensemble des entrées (pas de couche cachée).

Faisons un zoom sur les quatres sorties de l'une de ces parties (figure 5.3) : parmi ces 4 neurones de sortie, on distringue 2 neurones de sélection et deux neurones de commande de l'actuateur considéré. Considérons par exemple la partie commandant le moteur de gauche. Les deux neurones de sélection S1 et S2, reliés donc aux entrées (capteurs du robot), vont permettre de choisir lequel des deux neurones de commande O1 et O2 va réellement envoyer la commande au moteur, de la manière suivante : si $S1 > S2$ alors O1

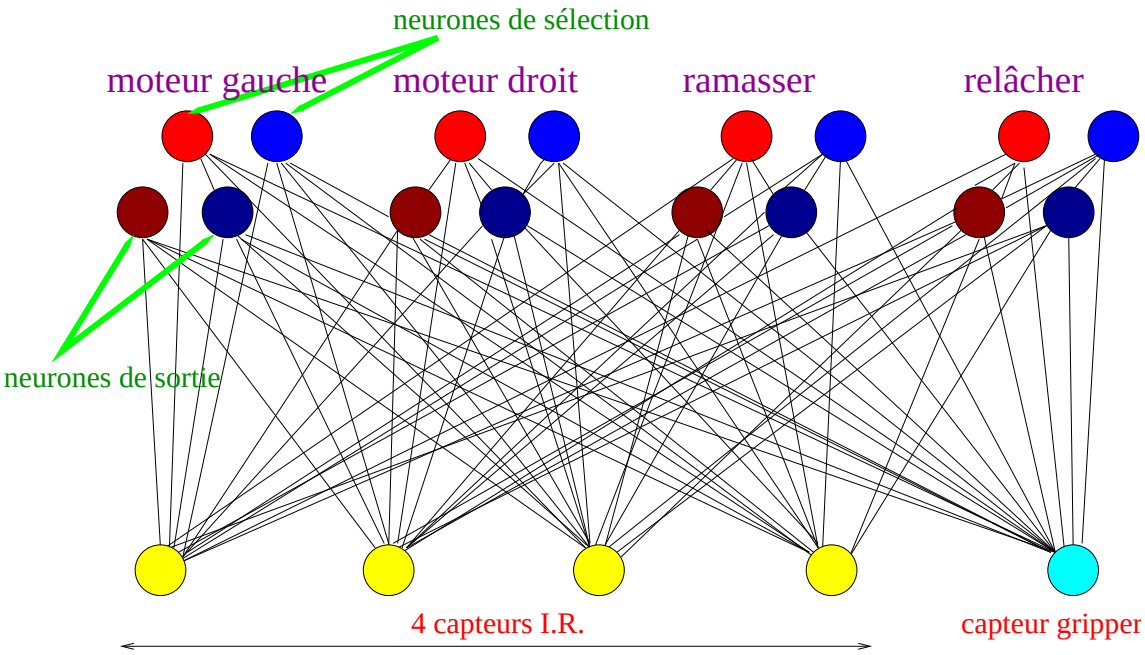


FIG. 5.2 – Le réseau modulaire de Nolfi et Floreano

commandera le moteur gauche, dans le cas contraire, ce sera O2 qui le commandera. Le choix est refait à chaque pas de temps.

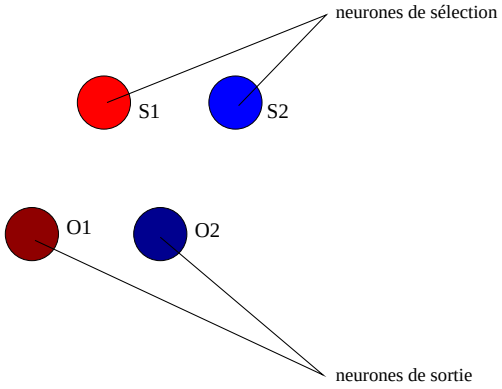


FIG. 5.3 – Zoom sur un bloc constituant une des sorties du réseau modulaire.

5.2.3 Résultats et discussions

Nous avons refait des expériences sur le robot “ramasseur d’ordures” suivant les indications données dans [Nolfi et Floreano, 2000], c’est-à-dire en utilisant l’architecture modulaire décrite ci-dessus. À titre de comparaison, nous avons également utilisé une architecture identique mais sans modularité, utilisant un réseau simple sans couche cachée reliant

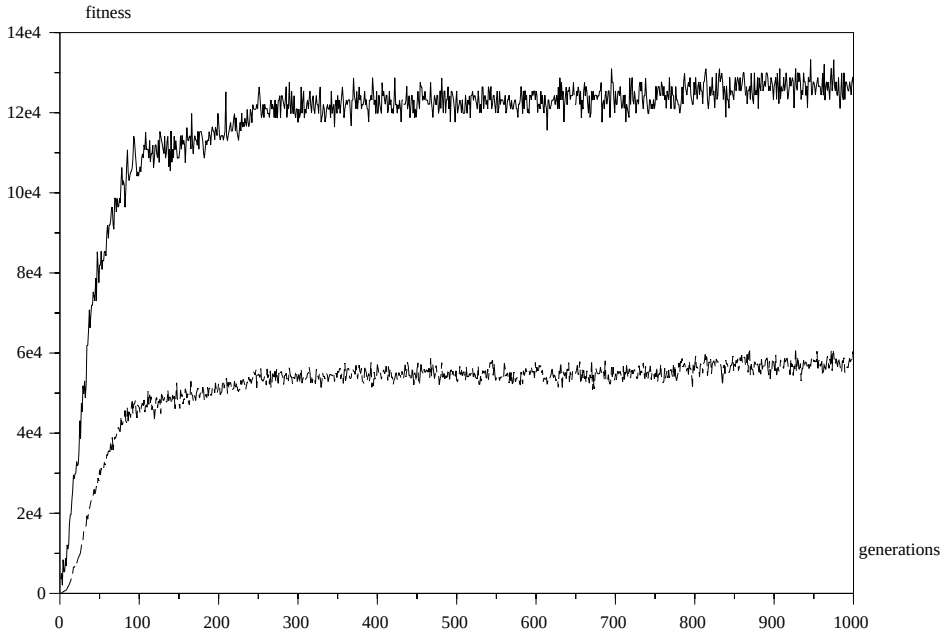
directement les entrées et les quatres sorties (les deux moteurs et les neurones déclencheurs du gripper). Les résultats, en terme d'évolution de la fitness au cours des générations, sont visibles sur la figure 5.4. Les résultats des deux architectures sont assez similaires, avec une meilleure performance tout de même pour les réseaux modulaires (+10000 environ sur la fin pour les meilleurs individus). Il est à noter également que le réseau modulaire, bien qu'il ait beaucoup plus de poids à optimiser que le réseau direct, progresse plus vite. On retrouve donc des résultats similaires à ceux obtenus dans [Nolfi et Floreano, 2000], où il est précisé que ce réseau modulaire permet d'accélérer un peu plus la progression du meilleur individu dans l'évolution artificielle.

Cependant, dans l'expérience originale décrite dans [Nolfi et Floreano, 2000], les résultats n'ont pas été analysés en profondeur, en particulier au niveau des activations des neurones de sélection, et il est difficile au vu des résultats publiés de répondre à la question "Quel est réellement l'apport de la modularité' ?" C'est pour cela que nous avons étudié les meilleurs individus obtenus pour les réseaux modulaires. Afin d'étudier ces réseaux de manière équitable, les cylindres ont été placés à des endroits identiques pour tous les individus testés à chaque nouvelle époque (mais différents d'une époque à l'autre). De plus ceci facilite beaucoup la représentation des résultats. Une illustration du principal résultat obtenu par cette analyse est présenté sur la figure 5.5.

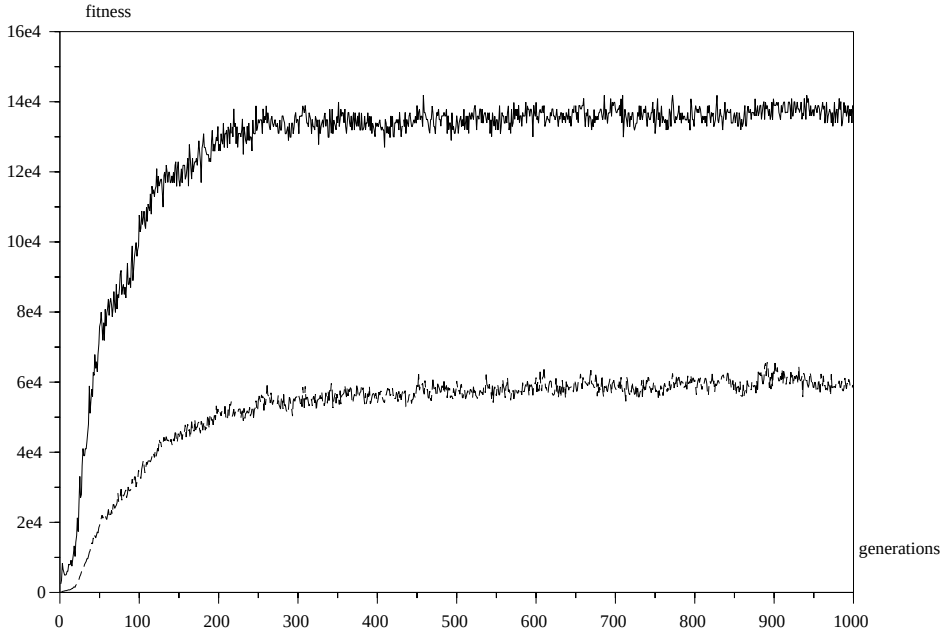
Le but de cette série de tests était d'en savoir un peu plus sur la manière dont fonctionnent les neurones de sélection. Sur la figure 5.5, le dessin du haut représente la prise d'un des cylindres par le Khepera : les points gris et les croix grises représentent les positions du robot à chaque nouveau cycle et l'intensité avec laquelle le neurone déclencheur de prise d'objets est déclenché. Plus la croix est grande, plus l'intensité est importante, sachant qu'elle doit être suffisamment grande pour dépasser le seuil permettant de déclencher la prise du cylindre. On peut apercevoir, quoiqu'assez difficilement (en utilisant une bonne loupe, cela semble évident !), une très petite croix noire, qui correspond à l'utilisation de l'autre neurone de sortie (donc également l'autre neurone de sélection) sur ce seul pas de temps.

Sur la figure du bas, suivant le même principe, on observe le comportement du neurone servant à relâcher l'objet en dehors de l'arène, et on assiste en fait au même phénomène hégémonique d'un des neurones de sortie commandant l'action préétablie de façon déterministe pour la manoeuvre de relâchement d'objet. Cette fois ci, on peut distinguer, mais très difficilement également, deux points noirs à proximité des cylindres de droites. Des expériences similaires ont été réalisées pour les moteurs, avec des résultats tout à fait similaires.

Concrètement, tout ceci signifie, que l'évolution semble avoir tendance à privilégier tel ou tel module, et à ne plus utiliser que celui-là lors de l'ensemble des tests. Cela peut expliquer, le fait que l'évolution progresse plus vite, en effet, le nombre de combinaison



Un perceptron simple sans couche cachée, la performance du meilleur individu et de la moyenne des invidus sont affichés.



Le réseau modulaire

FIG. 5.4 – Les résultats pour l'expérience du ramassage d'ordure.

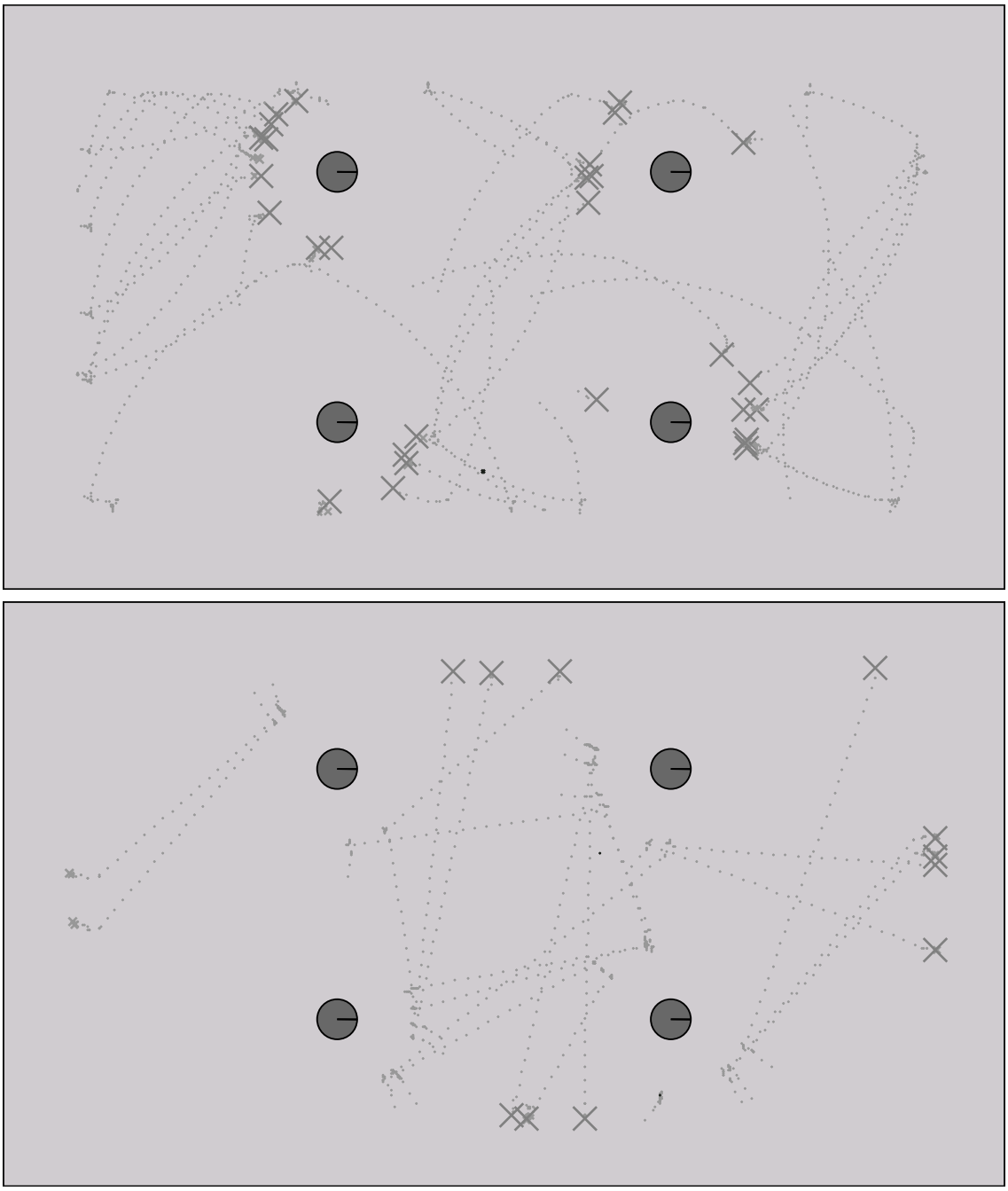


FIG. 5.5 – Analyse de la modularité effective durant une époque du meilleur individu obtenu lors de l'expérience du "ramasseur de poubelles"

possible avec 8 modules (4×2) est de $2^4 = 16$ par conséquent, les chances pour que le bon échantillon de poids soit trouvé dès les premières générations sont 16 fois plus importantes que pour un réseau classique. Cela peut également suffire à expliquer la très légère différence de performance du réseau modulaire sur le réseau classique.

Ces nouveaux éléments pris en compte indiquent que ces réseaux modulaires (bien que l'idée semble séduisante à la base) sont finalement comparables à des réseaux classiques, pour lesquels on aurait lancé 2^n runs en parallèle (n étant le nombre de sous parties du réseau) au lieu d'un seul. Il semble donc qu'une fois de plus l'évolution ait trouvé un raccourci inattendu au travers de l'utilisation du réseau modulaire, raccourci qui n'utilise pas du tout en fait la modularité telle qu'elle avait été prévue par les concepteur du réseau !

Partant de ces constatations, nous avons réfléchi à une manière de garder à l'esprit ce type de modularité en demandant à la fois au concepteur de fournir plus d'information de haut niveau, sous la forme de modules spécialisés, mais en laissant plus de souplesse à l'évolution pour arranger les choix entre les modules que dans le réseau modulaire de Nolfi et Floreano, et surtout en essayant de rester le plus indépendant possible par rapport à la tâche demandée, afin que l'approche puisse se généraliser facilement.

5.3 Les contrôleurs symboliques

5.3.1 Raisonnement

Nous nous plaçons donc dans le cadre de la robotique autonome évolutionnaire décrit au chapitre 3, et considérons les deux objectifs décrits dans le début de ce chapitre : d'une part, pouvoir aider l'évolution en lui fournissant des briques de base évoluées contenant ce que nous pensons devoir être utile comme comportements simples (ou un peu moins) pour construire le comportement que nous cherchons à atteindre, mais sans devoir préciser les liens ni les séquences d'enchainements de ces comportements de base ; d'autre part, essayer d'obtenir in fine des résultats soient assez intelligibles pour permettre leur interprétation par le concepteur, facilitant ainsi leur mise au point, et la généralisation à d'autres comportements cibles.

Nous nous restreignons au cadre de contrôleurs neuronaux (qu'ils soient à propagation directe ou récurrents), mais il est clair que d'autres types de contrôleurs pourraient bénéficier de la technique proposée ici. Les entrées des contrôleurs sont donc les capteurs du robot (infrarouge, caméra, ...), et in fine, le contrôleur doit générer des commandes pour les actionneurs du robot (e.g. les vitesses de moteurs de gauche ou de droite du robot pour un robot Khépéra, éventuellement les ordres pour le gripper s'il y a lieu).

Dans l'approche "classique", les valeurs des sorties du réseau de neurone sont directement (après une simple normalisation) les valeurs des commandes cherchées (vitesses des moteurs dans l'intervalle $[-20, 20]$ pour le Khepera par exemple. Le contrôleur résultant

est alors comparable à un programme en langage machine, c'est à dire très difficile à interpréter. De plus, supposons que le déplacement en ligne droite représente un sous-comportement utile (ce qui peut sembler raisonnable de supposer dans de nombreuses situations, et pour des comportements cibles très différents). Et bien non seulement il sera difficile de dire quand un contrôleur neuronal donnant des consignes pour les moteurs se déplace effectivement en ligne droite, mais de plus la "découverte" du déplacement en ligne droite pourra le cas échéant nécessiter un certain temps d'évolution – voire ne jamais être exactement atteinte. On voit ainsi de nombreux cas de solutions obtenues par évolution de contrôleurs neuronaux, et dont on sait qu'il faut qu'ils atteignent un point de leur environnement rapidement, effectuer des déplacements peu rectilignes. D'un autre côté, la programmation déterministe d'un tel déplacement est évidemment triviale.

5.3.2 L'architecture symbolique

Dans ce cadre, nous proposons d'utiliser comme sorties du contrôleur des comportements de plus haut niveau que les valeurs des vitesses des moteurs, tels le déplacement en ligne droite mentionné ci-dessus. Plus spécifiquement, dans un premier temps, nous dotons le robot de 4 actions possibles programmées de manière déterministe : Aller tout droit en avant, tourner à droite, tourner à gauche, ou aller tout droit en arrière. De plus, dans le but de permettre plus de flexibilité, chacune de ces actions est modulée par un paramètre (dans notre cas la vitesse de déplacement linéaire pour les déplacements rectilignes, et la vitesse de rotation pour les autres). Le contrôleur donne alors, à chaque pas de temps, d'une part le choix de l'action qui va être exécutée, d'autre part la valeur du paramètre correspondant. C'est un module déterministe qui transforme alors couple (action, paramètre) en consignes pour les moteurs.

Le contrôleur symbolique proposé est représenté de manière synthétique figure 5.6 : il comporte 8 sorties prenant des valeurs dans [0,1]. Les 4 premières sorties sont utilisées pour spécifier quelle action va être utilisée. Plus précisément, le robot effectuera l'action i, avec $i = Argmax\{sortie(j), j = 0 \dots 3\}$. Le paramètre correspondant est alors la valeur de la sortie $i + 4$. Les valeurs des commandes pour les actuateurs sont calculés par le module (prédéfini par le concepteur) correspondant à l'action choisie.

À noter que les sorties 4 à 7, qui sont les paramètres des actions 0 à 3, sont calculées à chaque pas de temps, mais que seule la valeur du paramètre de l'action choisie est utilisée à un pas de temps donné : quelques essais de réseaux à 5 sorties, les quatres sorties du choix et un unique paramètre qui serait appliqué à l'action choisie nous ayant rapidement convaincu qu'il était nécessaire de séparer les paramètres – ce qui était prévisible compte-tenu de la sémantique différente d'une vitesse linéaire et d'une vitesse de rotation par exemple.

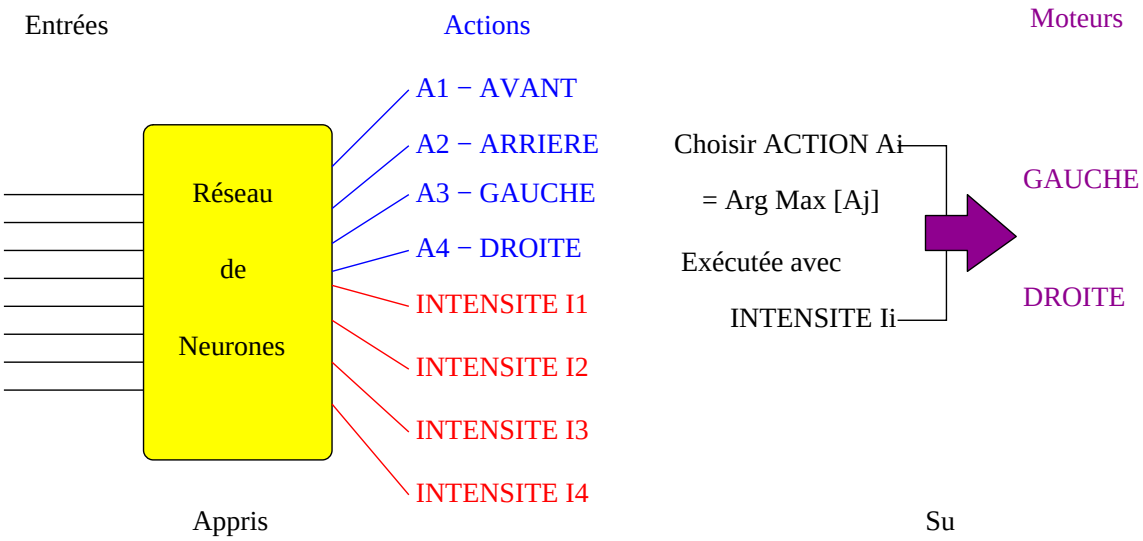


FIG. 5.6 – Le réseau symbolique.

5.3.3 Discussions

Utiliser des langages et représentations de plus haut niveau pour les contrôleurs a des conséquences à la fois sur la taille de l'espace de recherche et sur la modularité du contrôleur.

A première vue, il semble que la taille de l'espace de recherche soit augmentée, du fait que le contrôleur possède 8 sorties au lieu de 2 dans un contrôleur classique. Toutefois, les 8 sorties ne sont qu'une combinaison des 2 sorties moteurs. D'un autre côté, l'utilisation d'un contrôleur symbolique peut être vue comme l'ajout de contraintes sur l'espace de recherche, ce qui entraîne une réduction de la taille de la partie de l'espace de recherche en fait explorée. On peut également rapprocher cet argument de celui utilisé par Vapnik comme fondation de la théorie de l'apprentissage statistique [Vapnik, 1998], où réécrire le problème d'apprentissage dans un espace de plus grande dimension le rend en fait plus simple (linéairement séparable ... pour un bon choix de l'espace de grande dimension, bien sûr). De plus, le paysage de fitness engendré par l'utilisation d'un contrôleur symbolique comprend de nombreux plateaux neutres, puisque seule la plus grande valeur des premières sorties est prise en compte. Or la neutralité est souvent invoquée comme bénéfique pour l'évolution, permettant de s'échapper plus facilement d'optima locaux [Harvey, 1993].

D'un autre côté, les primitives de haut niveau des contrôleurs symboliques engendrent plus de modularité. D'après Dawkins [Dawkins, 1988], la probabilité de construire un système complexe par un processus aléatoire augmente avec le degré de modularité. Mais nous reviendrons sur ce point plus en profondeur dans la section 5.5.

Enfin, il est bon de noter que ce principe a déjà été utilisé, sans être explicité ni étudié

en tant que tel, en robotique évolutionnaire. Ainsi, dans l'expérience du "ramasseur de poubelle" décrite section 5.2, les auteurs ont choisi ce mode d'action pour contrôler la pince du robot Khépéra : les sorties des contrôleurs dédiées au gripper sont des actions de haut niveau (e.g. saisir l'objet, relâcher l'objet,...), et non les commandes des moteurs des pinces.

5.4 Apprendre un comportement simple : l'évitement d'obstacle

La suite de ce chapitre va être consacrée à des validations expérimentales de l'approche proposée ci-dessus (et de sa généralisation). Nous décrivons dans cette section tout d'abord les conditions expérimentales communes à toutes les expériences qui vont suivre, avant d'aborder la validation de l'approche des contrôleurs symboliques sur une première expérience d'évitement d'obstacle.

5.4.1 Le protocole expérimental

Comme dans l'ensemble des travaux de cette thèse, le génotype est constitué des poids du réseau de neurones. Ces poids évoluent dans $[-1, 1]$, en utilisant une (30,150)-ES (Stratégie d'Évolution : chacun des 30 parents donne naissance à 5 enfants, et les meilleurs des 150 enfants deviennent les 30 parents de la génération suivante) avec une faible probabilité de croisement et des mutations gaussiennes auto-adaptatives (voir section 3.2.4). Nous avons choisi comme critère d'arrêt un nombre maximal de génération fixé à 250, qui représente un temps machine de 1h à 3h suivant l'expérience à réaliser sur un ordinateur relativement récent (Pentium de fréquence $>$ 2GHz).

Tous les résultats sont présentés sur une moyenne de 10 essais indépendants. Chaque évaluation de performance est effectuée sur 10 époques, et chaque époque dure de 150 à 1000 pas de temps (dépendant de l'expérience), en commençant par une position aléatoire.

5.4.2 Description de l'expérience

L'expérience d'évitement d'obstacle s'est déroulée dans deux environnements différents visibles sur la figure 5.7 :

- Un environnement séparé par une croix (situé à gauche sur la figure 5.7).
- Un environnement un peu plus compexe, où le robot possède peu de place pour se déplacer, et doit bien faire attention à éviter les murs.

Le comportement recherché est ici un simple comportement d'évitement d'obstacle, et la fonction fitness utilisée est définie comme

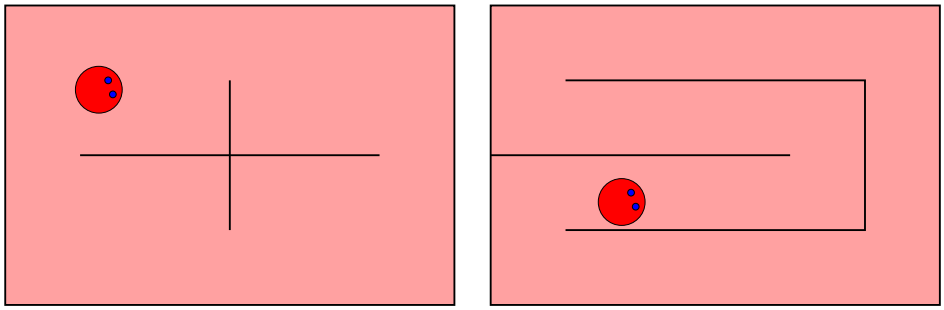


FIG. 5.7 – Les deux environnements testés avec le contrôleur symbolique pour l'expérience de l'évitement d'obstacle

$$\mathcal{F} = \sum_{epoch} \sum_{t} |V(t)|(1 - \sqrt{\delta V(t)})$$

où V(t) est la vitesse moyenne au temps t, et $\delta V(t)$ représente la valeur absolue de la différence entre la vitesse des deux roues.

La différence entre cette fitness et la fitness originale de Nolfi et Floreano [Nolfi et Floreano, 2000] est l'absence des valeurs des capteurs infrarouges dans la fitness. En effet, le comportement d'évitement d'obstacle est ici implicitement récompensé, car dès que le robot rentre en contact avec un mur, l'évaluation de l'époque courante est arrêtée, l'accumulation de fitness est ainsi faite sur moins de pas de temps, et le robot est ainsi indirectement pénalisé.

Le contrôleur symbolique à 8 sorties décrit section 5.3.2 (et figure 5.6) est comparé dans cette expérience à un contrôleur classique à 2 sorties – les commandes des moteurs. Les deux réseaux possèdent 8 entrées, à savoir les 8 capteurs infrarouges du Khépéra en mode actif, c'est-à-dire permettant de détecter les obstacles.

Pour cette tâche simple, nous n'avons utilisé que des architectures de perceptrons multi-couches. Par contre, le réglage du nombre de couches et de neurones a fait l'objet d'expériences préliminaires, dont les résultats principaux sont donnés dans le tableau 5.1 : avec seulement deux couches, les contrôleurs symboliques obtiennent des meilleurs résultats que les contrôleurs classiques. En augmentant le nombre de neurones de la couche cachée, les performances des deux contrôleurs augmentent mais les performances du contrôleur symbolique restent meilleures. Finalement, pour établir une comparaison équitable en terme de nombre de poids, nous avons choisi d'utiliser 14 neurones cachés pour le contrôleur symbolique et 20 pour le contrôleur classique, le nombre de poids étant alors quasiment identique (216 vs 221).

Architecture	Classique	Symbolique
8-2 / 8-6	861 ± 105	1030 ± 43
8-8-2 / 8-8-6	1042 ± 100	1094 ± 55
8-20-2 / 8-14-6	1220 ± 41	1315 ± 6
8-20-2 / 8-14-6*	1132 ± 55	1197 ± 16

TAB. 5.1 – Résultats comparatifs des contrôleurs classiques et symboliques. * signifie que le robot Khépéra a été testé dans le second environnement (plus complexe)

5.4.3 Résultats et discussions

Les premiers résultats pour les contrôleurs symboliques ont été surprenants : la moitié des runs, même sans couche cachée, a trouvé une faille opportuniste dans l'évolution, rendue possible par l'absence d'inertie dans le simulateur : un comportement optimal est en effet obtenu dans ce cas par une succession rapide des comportements "aller vers l'avant" et "aller vers l'arrière" à vitesse maximale – et par conséquent évitant ainsi les obstacles !

À noter que les contrôleurs classiques ont réussi également à exploiter cette faille, mais uniquement lorsque nombre de neurones cachés est augmenté, et en autorisant un intervalle pour les poids 10 fois plus grands ($[-10, 10]$).

Nous avons donc décidé d'une part d'éliminer le comportement "aller vers l'arrière", réduisant ainsi le nombre de sortie du contrôleur symbolique à 6 neurones, et d'autre part, dans le but d'éviter définitivement cette faille, de modifier la fonction fitness pour qu'elle augmente uniquement lorsque le robot va vers l'avant. Cette modification n'a pas altéré le classement des contrôleurs : le contrôleur symbolique continue de dépasser les performances du contrôleur classique. Cet avantage diminue cependant avec l'augmentation du nombre de neurones cachés alors que la variance des résultats baisse, pour les deux réseaux (voir tableau 5.1).

Ces résultats valident l'approche des contrôleurs symboliques : de meilleurs résultats sont obtenus qu'avec un contrôleur classique. Toutefois, il semble clair que le principal obstacle à l'utilisation de l'approche évolutionnaire pour des tâches beaucoup plus complexes soit le passage à l'échelle discuté section 5.1 qui n'est pas vraiment réduit par l'approche des contrôleurs symboliques. En effet, l'utilisation de nombreux comportements de base experts va nécessiter un réseau de neurones très grand – et d'autre part l'idée des paramètres que nécessite chaque action de base ne s'applique sans doute pas telle quelle à d'autres comportements experts que le concepteur pourrait vouloir ajouter à ceux proposés section 5.3. L'idée d'un processus de conception réentrant, dans lequel on pourrait utiliser des comportements conçus par évolution en tant que brique de base pour la construction de comportements plus complexes semble à ce moment incontournable.

5.5 Les superviseurs

Pour palier les limitations constatées ci-dessus, nous proposons l'architecture dite des "superviseurs", qui consiste à fournir à un module contrôleur une librairie de contrôleurs préexistants dans laquelle il va choisir, à chaque pas de temps, un des modules pour donner les consignes aux actuateurs du robot. La figure 5.9 montre une instanciation du concept de superviseur.

Les entrées du module de supervision sont les valeurs de capteurs disponibles (ou un sous-ensemble de ces valeurs, un autre exemple est donné à la section 5.7). Il y a autant de sorties du module superviseur que de contrôleurs disponibles dans la librairie. À chaque pas de temps, la sortie qui prend la valeur la plus importante décide du module contrôleur qui va être utilisé pour envoyer les consignes aux actuateurs. Le principe reprend la base de l'approche "contrôleurs symboliques" présentée section 5.3.2, avec une différence essentielle que les contrôleurs de la librairie sont des contrôleurs à part entière, c'est-à-dire qu'ils donnent des consignes pour les actuateurs en fontion des valeurs des senseurs. Cette différence a deux conséquences essentielles :

- il est possible (comme nous le verrons plus loin) d'utiliser dans la librairie aussi bien des contrôleurs programmés de manière déterministe par le concepteur que des contrôleurs obtenus par évolution artificielle, ou par tout autre méthode d'apprentissage ;
- en particulier, il est possible (même si cela n'a pas encore été expérimenté) d'utiliser un contrôleur superviseur, résultat d'une évolution préliminaire, à son tour comme l'un des contrôleurs de la librairie servant à construire un autre superviseur – dit alors "de plus haut niveau" (même si la notion de niveau n'est pas bien définie).

Cette approche répond donc au moins en partie aux problèmes discutés dans la section 5.1 concernant le passage à l'échelle de l'approche évolutionnaire en robotique : la modularité devient une réalité, mais sans que cela alourdisse l'intervention humaine en terme de description fonctionnelle des relations entre les différents modules. Bien sûr, l'expert a encore du travail (fournir au superviseur un ensemble suffisant de contrôleurs), mais la partie fastidieuse et combinatoire de mise au point des liens entre ces contrôleurs est laissée à l'évolution, et risque moins d'être biaisée par des a priori humains. En ce sens, on peut faire l'analogie suivante : l'approche superviseur est à l'approche sélection d'action classique ce que la programmation déclarative est à la programmation séquentielle.

Un des problèmes-clé de cette approche, et qui pourrait remettre en question le gain que nous pensons atteindre en terme de passage à l'échelle, est celui du choix des contrôleurs de base, et de la dépendance des résultats par rapport à la librairie fournie. Ce point sera discuté plus en détail section 5.6.4. Mais commençons par valider l'approche superviseur par une première expérience, plus complexe que l'évitement d'obstacle, l'exploration avec

recharge de batterie.

5.6 Evolution d'un comportement complexe

5.6.1 Exploration et recharge de batterie

Le comportement recherché est dérivé de l'expérience "passage par la maison" (homing experiment) pour la première fois proposée dans [Floreano et Mondada, 1994b], combinant l'exploration de l'environnement avec une gestion de l'énergie. Le robot est équipé d'un accumulateur d'énergie (virtuel), l'énergie diminue linéairement avec le temps, et est totalement épuisée après 285 cycles. Une aire de recharge est signalée par une lumière dans l'arène (cf. figure 5.8). Aucun obstacle se trouve dans l'arène. Dans notre expérience, la position du robot ainsi que la zone de recharge sont tirées aléatoirement au début de chaque époque (dans l'expérience originale, la zone de recharge se trouvait toujours au même endroit, dans un coin de l'arène).

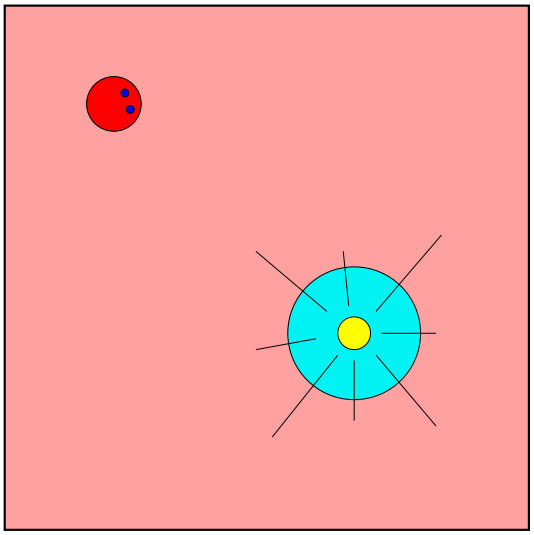


FIG. 5.8 – L'expérience du passage par la maison, le robot doit se déplacer rapidement dans son environnement tout en s'approchant de la zone représentée par la lumière pour recharger ses batteries.

La performance du robot est proportionnelle à sa vitesse moyenne, mais seulement lorsque le robot n'est pas situé dans la zone de recharge. Dans l'expérience originale [Floreano et Mondada, 1994b], les batteries étaient instantanément rechargées lorsque le robot entrait dans la zone de recharge. La recharge ici est proportionnelle au temps que le robot passe dans cette zone (une recharge complète prend 100 pas de temps). De plus, l'aire de recharge n'est pas directement "visible" par le robot, alors qu'elle était signalée par un disque noir que le robot pouvait détecter avec un capteur de sol. Toutes ces différences

augmentent la complexité de la tâche.

5.6.2 Les architectures

Dans cette expérience, nous avons utilisé une architecture de superviseur telle que nous allons la décrire maintenant. Son nombre de sorties est égal au nombre de comportements disponibles. Dans un premier temps, nous avons considéré les 4 comportements suivants, obtenus par évolution : une fitness visant à obtenir un comportement simple a été utilisée avec les conditions expérimentales qui seront décrites par la suite pour chaque comportement recherché ; le meilleur comportement obtenu en fin d'évolution a été sauvegardé pour être utilisé comme brique de base du superviseur. Bien entendu, des comportements programmés de manière déterministe auraient pu être utilisés également (comme ce sera le cas lors de l'étude de robustesse section 5.6.4). Voici la liste des comportements utilisés en sortie du surperviseur (voir figure 5.9) pour cette expérience :

- *L'évitement d'obstacle* : Ce comportement a été obtenu par une évolution telle qu'elle est décrite notamment dans l'expérience de la section précédente 5.4.2.
- *Le suivi de lumière* : La fitness utilisée pour aboutir à ce comportement est le nombre de fois que le robot atteint la zone de recharge sous la lumière durant 10 époques (sans aucune idée d'énergie). Un simple perceptron à 2 couches a été suffisant pour l'obtention d'un suivi de lumière efficace.
- *Stop* : Ce comportement est le résultat d'une évolution durant laquelle on cherche à minimiser la vitesse du centre du robot. Il faut noter que ce comportement est très facile à obtenir par évolution : un simple réseau à 2 couches a été utilisé, et très peu de générations sont nécessaires pour arriver à un tel comportement, et que le comportement obtenu est le plus souvent un robot qui tourne sur lui-même (ce sera le cas ci-dessous). On aurait même facilement pu programmer de manière déterministe ce comportement, mais l'idée est aussi ici de mettre en évidence le caractère réentrant de l'approche proposée.
- *Balayeur* : Le but recherché est ici un robot qui explore le plus largement possible son environnement – sans heurter les murs. L'aire est discrétisée en petits carrés, et la performance est le nombre de carrés différents ayant été visités par le robot durant 10 époques. Ce comportement a nécessité l'utilisation d'un réseau de type Elman (perceptron multi-couche avec une couche cachée totalement récurrente), avec 5 neurones dans la couche cachée.

Deux types de superviseurs ont été testés : les superviseurs dits *classiques*, dans lesquels tous les comportements sont obtenus par des contrôleurs classiques (c'est-à-dire ayant 2 sorties qui sont les commandes directes des moteurs), et des superviseurs *symboliques*, qui utilisent comme briques de base des contrôleurs symboliques tels que décrits dans la section 5.3. Les comportements de base utilisés par les deux types de superviseurs

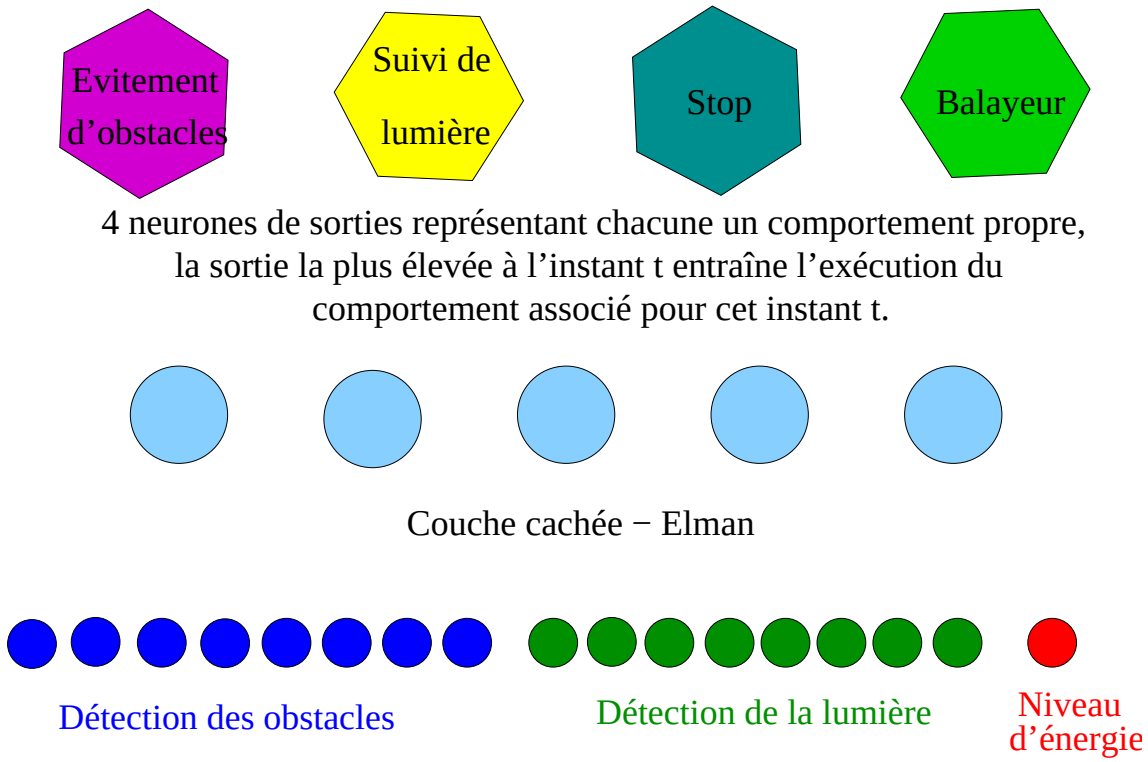


FIG. 5.9 – L'architecture du réseau de supervision : le superviseur.

sont donc le résultat d'évolutions différentes. A noter qu'il serait possible de mélanger des comportements de base classiques et symboliques, puisque une fois la décision prise par le superviseur, le robot reçoit de toute façon des ordres moteurs, soit directement du contrôleur (cas classique) soit via une "traduction" déterministe des sorties du contrôleur (cas symbolique). Mais les expériences décrites ici ne concernent que des superviseurs à comportements de base homogénes, tous classiques ou tous symboliques.

Les réseaux de neurones utilisés pour la supervision sont des réseaux de type Elman (perceptron à 3 couches, la couche cachée, composée de 5 neurones, étant totalement récurrente).

À titre de comparaison, les expériences ont également été effectuées avec un contrôleur de type Elman direct, soit avec des sorties classiques (appelé "Elman classique" dans la suite) soit avec des sorties symboliques (et donc appelé "Elman symbolique").

Pour l'expérience de recharge de batterie, tous les contrôleurs considérés ont 17 entrées : 8 capteurs infrarouges en mode actif pour l'évitement d'obstacle, les mêmes 8 capteurs infrarouges, mais en mode passif, pour la détection de la lumière et le niveau de batterie, qui constitue le dernier "capteur". Le nombre de sorties est de 2 pour un contrôleur classique (les vitesses des moteurs), et de 6 pour les contrôleurs symboliques, qui utilisent les trois

comportements de base : “aller tout droit”, “tourner à droite”, “tourner à gauche”. Les superviseurs ont eux 4 sorties (qu’ils soient classiques ou symboliques) qui permettent le choix entre les 4 comportements évolués : “évitement d’obstacle”, “suivi de lumière”, “stop” et “le balayeur” (figure 5.9).

5.6.3 Premiers résultats

Les statistiques sur 10 essais indépendants peuvent être vus sur les figures 5.10 et 5.11. Trois critères peuvent être utilisés pour comparer les performances des 4 architectures : la meilleure performance, la variance de ces meilleures performances, et enfin la rapidité avec laquelle ces meilleures performances sont atteintes.

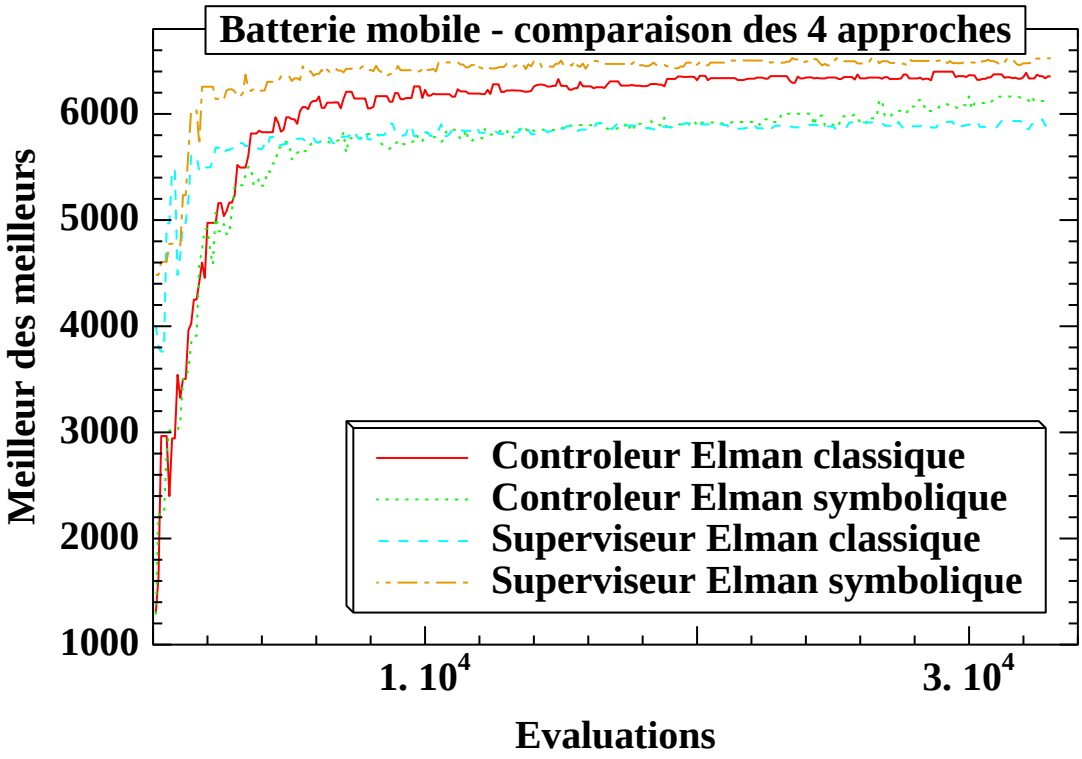


FIG. 5.10 – Résultats pour l’expérience de batterie, le meilleur des meilleurs sur les 10 runs.

La meilleure performance est obtenue par les superviseurs symboliques. De plus, la variance en est très faible (la moyenne de la meilleure performance est 6442 ± 28). Notons que dépasser une performance de 6400 constitue presque la performance maximale, et le comportement est pratiquement optimal entre la navigation et les phases de recharge.

L’architecture ayant la deuxième meilleure performance est le contrôleur classique. Mais alors que la moyenne est significativement plus basse, l’écart-type est 10 fois plus élevé (6044 ± 316, avec le meilleur à 6354). Les contrôleurs symboliques et les superviseurs

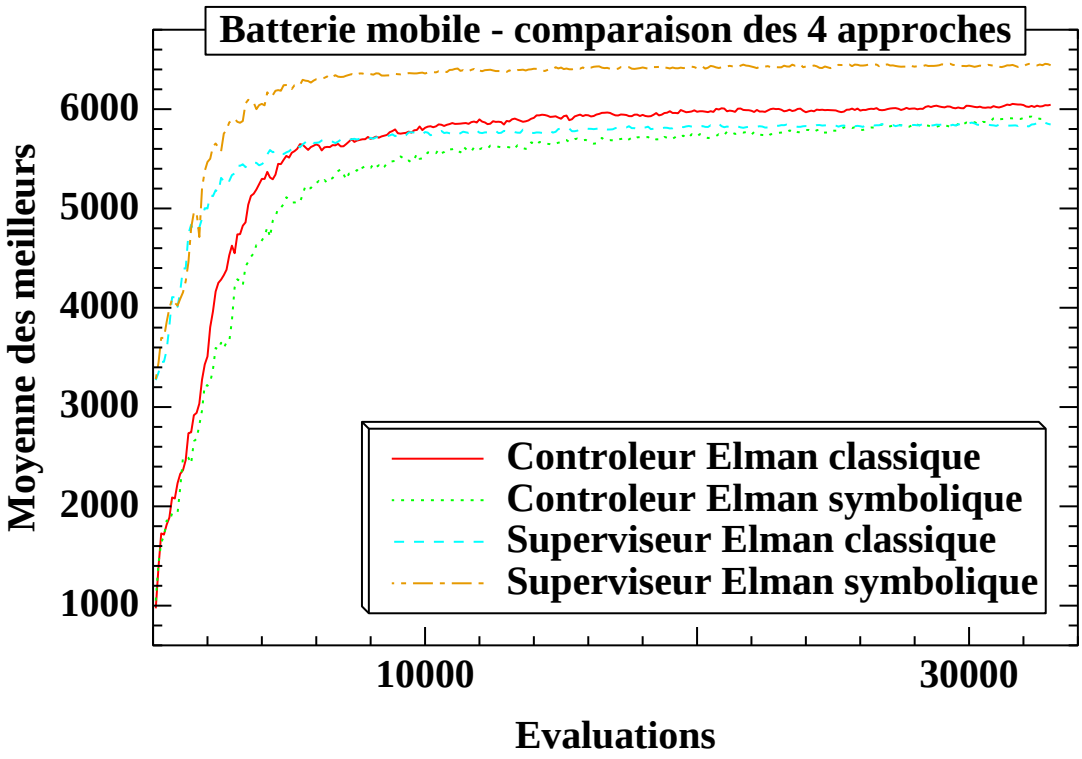


FIG. 5.11 – Résultats pour l'expérience de batterie, la moyenne des 10 essais.

classiques viennent ensuite, avec respectivement la meilleure performance à 5902 ± 122 et 5845 ± 27.

Commentons maintenant plus en détails ces résultats. Tout d'abord, les deux architectures de superviseurs montrent des performances très élevées (environ 3200) dès la population initiale. Mais de telles performances sont obtenues parce que les meilleurs superviseurs aléatoires sont ceux qui utilisent uniquement l'évitement d'obstacle tout le long de la première période de décharge de batterie. À l'inverse, dépourvus de toute connaissance initiale, les contrôleurs directs mettent beaucoup plus de temps pour atteindre de telles performances (plus de 2000 évaluations). Ce résultat est tout à fait encourageant, dans la mesure où le but de l'introduction des superviseurs est précisément de pouvoir transmettre du savoir acquis dans un cadre à un autre cadre. Mais il faut également éviter que de tels comportements n'inhibent pas totalement la recherche : il s'agit d'un optimum local, et il se pourrait qu'il soit difficile pour l'évolution d'en sortir, et que les individus qui dévieraient de ce comportement soient ensuite systématquement éliminés par la sélection.

Une autre remarque est que la variance pour les deux types de superviseurs est très basse (indépendamment de leurs performances). De plus, il semble que cette faible variance

n'est pas seulement vraie pour la performance mais également au niveau du comportement : alors que les superviseurs symboliques explorent l'environnement jusqu'à ce que leur énergie devienne dangereusement basse, et se dirigent alors vers la zone de lumière, y restant jusqu'à ce que le niveau de batterie ait de nouveau atteint le maximum, la plupart (mais pas tous) des contrôleurs directs semblent simplement rester proches de la zone de recharge, y rentrant quand le niveau de batterie est faible, mais pas toujours au même niveau, et par un endroit différent à chaque reprise.

Une constatation inattendue, surtout au vu des résultats de la section 5.4.3, est la faible performance des contrôleurs symboliques. Une possible explication est l'existence des plateaux de neutralité, discutée dans la section 5.3.3 : ces plateaux permettent de s'échapper plus facilement des optima locaux, cependant ils ralentissent également le processus d'apprentissage. On peut également voir sur la figure 5.10 que l'architecture des contrôleurs symboliques est la seule qui semble encore augmenter sa meilleure performance sur la fin des générations. Pour tester cette hypothèse, l'expérience a été relancée pour 250 générations supplémentaires, et effectivement la performance des contrôleurs symboliques a continué d'augmenter, atteignant finalement plus de 6200), alors que les performances des autres architectures ont continué à stagner.

Les comportements évolués ont été examinés plus en détail. La figure 5.12 montre une courbe typique du nombre d'appels de chaque comportement basique par le meilleur superviseur symbolique évolué pendant une évaluation de fitness. Premièrement, il apparaît que les deux architectures de superviseurs utilisent principalement le comportement d'évitement d'obstacles, et n'utilisent en fait jamais le comportement "balayeur", ni, ce qui est plus surprenant, le comportement "suivi de lumière" : quand ils aperçoivent la lumière, ils tournent sur eux-même en utilisant, le comportement "stop" (qui consiste en fait à tourner très rapidement sur place), et se dirigent ensuite vers la lumière en utilisant le comportement d'évitement d'obstacles pour aller tout droit (la lumière est fixe ici). Une fois sur la zone de batterie, ils utilisent (à bon escient ici) leur comportement "stop" jusqu'à ce que leur batterie soit rechargée à plus de 90%.

Pour expliquer ce phénomène, examinons de manière plus approfondie les comportements de la librairie : de fait, les vitesses de déplacement du robot pour le comportement de suivi de lumière et le comportement balayeur sont moins grandes que celles du comportement d'évitement d'obstacles. Or la vitesse est cruciale dans l'expérience de recharge de batterie (beaucoup plus que pour les expériences de suivi de lumière ou de balayeur). Cela montre que le superviseur possède la capacité de sélectionner les comportements réellement efficaces pour la tâche à accomplir, même si ce ne sont pas ceux que le concepteur avait pensé utiles, tout en étant capable de recréer par assemblage de comportements de base un nouveau comportement (ici un suivi de lumière plus efficace que celui proposé).

5.6.4 Réactions face à de nombreux comportements disponibles

Un des principaux problèmes posé par l'approche proposée est de savoir si le "bon" comportement sera disponible pour le superviseur. En effet, même si nous avons vu que le superviseur peut, et c'est heureux, créer le comportement manquant par combinaison des comportements dont il dispose, nous avons vu également qu'il est bénéfique d'avoir dans ses briques de base des comportements déjà experts pour accélérer l'évolution. Une solution possible est de proposer le plus large choix possible de comportements. Mais la question qui se pose est alors bien sûr de savoir si le superviseur sera ou non capable de retenir uniquement les comportements utiles, et ne sera pas noyé sous l'avalanche de comportements inutiles. Dans le but de vérifier ceci, l'expérience de gestion d'énergie a été répétée, mais en ajoutant à la panoplie de base du superviseur de nombreux comportement inutiles, voire même parfois nuisibles, aux 4 comportements basiques.

Dans un premier temps, 4 comportements ont été ajoutés aux comportements existants, à savoir :

- *La marche aléatoire* : il s'agit d'un comportement programmé, qui donne des consignes aléatoires aux deux moteurs, tout en évitant, afin de rester un tant soit peu réaliste, les variations trop brutales de vitesse qui auraient pu sinon être invoquées pour la non-utilisation de ce comportement. Ce comportement est évidemment considéré a priori comme inutile ...
- *La fuite de la lumière* : il s'agit d'un comportement évolué à l'aide d'une fitness consistant à minimiser les valeurs des capteurs de lumière (infrarouges passifs). Dans le contexte de la recharge de batterie, ce comportement est bien entendu nuisible.
- *Foncer dans les murs* : c'est un autre comportement obtenu par évolution, la fitness consistant à minimiser le temps avant le premier crash dans un mur. L'évolution de ce comportement a été remarquablement rapide ! Quel que soit le but final recherché (sauf peut-être pour des épreuves de stock-car) ce comportement est a priori nuisible.
- *Coller aux murs* : l'évolution de ce comportement a été réalisée avec pour but de maximiser les valeurs des capteurs infrarouges en mode actif (détectant les obstacles), tout en évitant les crashs dans les murs. Le meilleur comportement typiquement se dirige droit vers un mur, et reste ensuite très proche dudit mur sans quasiment bouger. Dans le contexte de la recharge de batterie avec exploration, ce comportement devrait être lui aussi nuisible.

Les premières générations montrent des performances plus basses, et avec une plus grande variance, que dans l'expérience initiale. Ceci est relativement normal, car dans les individus des premières populations, tous les comportements sont bien sûr utilisés avec a priori les mêmes probabilités. Toutefois, la courbe de la meilleure fitness (non montrée) rattrape les courbes de la figure 5.10, et après 150 générations, la différence est difficilement distinguable. Enfin, si l'on regarde la fréquence d'utilisation de chaque comportement, il

apparaît clairement que même si certains comportements nuisibles sont utilisés (voir figure 5.13), ils le sont bien moins que les comportements réellement utiles (voir figure 5.12).

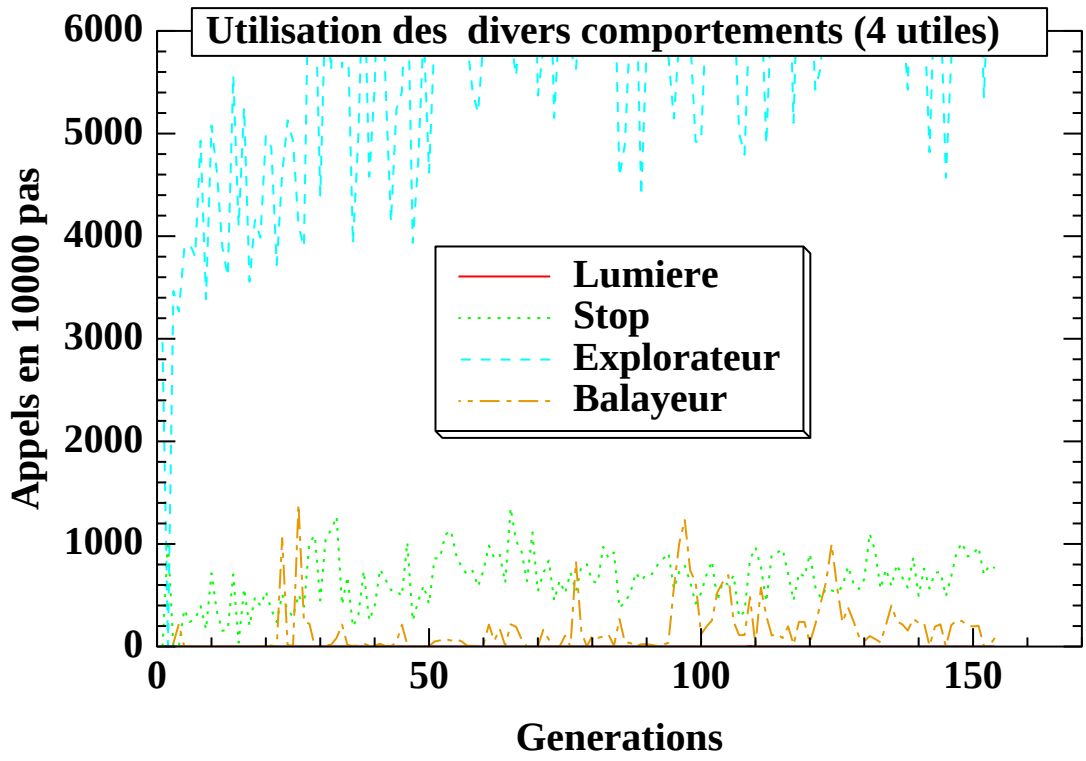


FIG. 5.12 – Nombre d'appels des 4 comportements originaux durant 10000 pas de temps pour le meilleur superviseur symbolique.

Ces bon résultats de stabilité ont été confirmés en cette fois ajoutant 20 nouveaux comportements inutiles (les 4 comportements précédents répétés 5 fois indépendamment). Les résultats sont très similaires aux précédents, avec bien entendu une utilisation un peu plus fréquente de l'ensemble de ces comportements inutiles, mais sans nuire à la bonne réalisation de l'expérience au final.

5.6.5 La généralisation

Un autre point essentiel des approches en robotique évolutionnaire est la capacité de généralisation des contrôleurs obtenus – ou en d'autres termes, avec quelle rapidité les résultats se dégradent-ils lorsque l'on change graduellement l'environnement dans lequel le robot évolue.

Plusieurs expériences ont été effectuées dans le but de tester les capacités de généralisation des superviseurs construits dans ce qui précède. Les 10×4 (10 pour le nombre d'essais et 4 pour le nombre de contrôleurs différents) meilleurs contrôleurs ob-

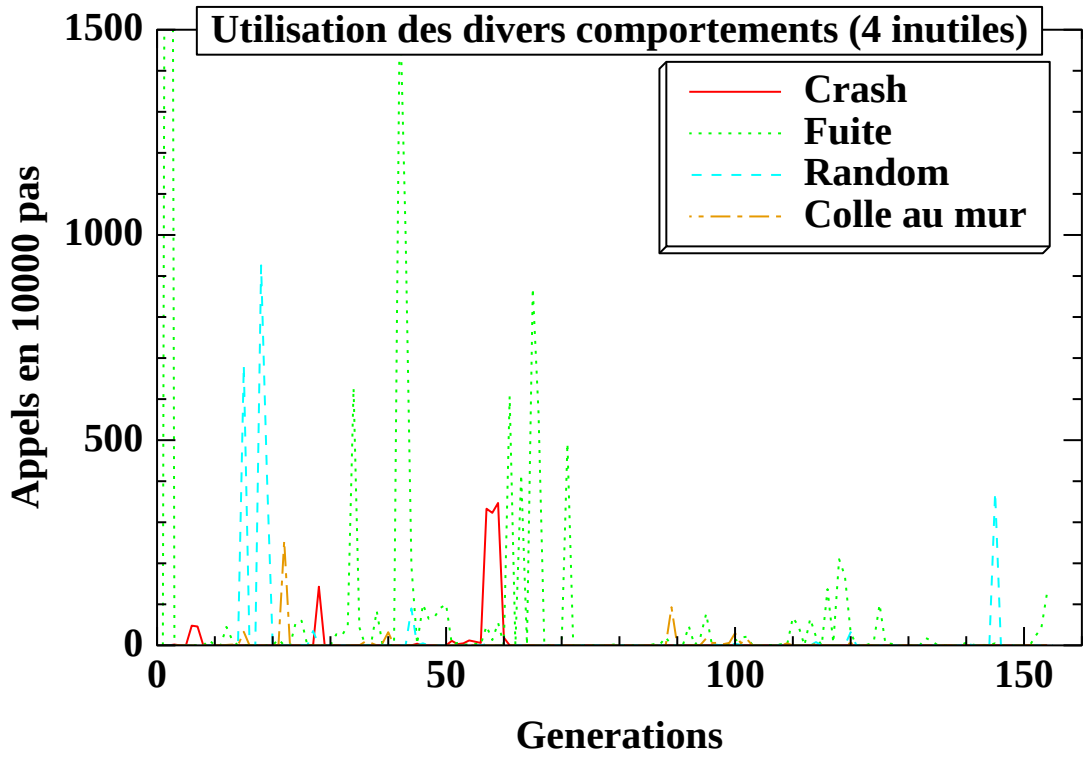


FIG. 5.13 – Nombre d'appels des 4 comportements additionels (inutiles ou nuisibles) durant 10000 pas de temps pour le meilleur superviseur symbolique.

tenus dans l'expérience précédente ont ainsi été testés dans de nouveaux environnements expérimentaux.

Premièrement, quelques obstacles ont été ajoutés dans l'environnement (l'arène était vide d'obstacles dans les expériences décrites plus haut). Mais aucun contrôleur n'a été capable d'aller dans la zone de recharge de batterie lorsque cette zone se trouvait de l'autre côté d'un obstacle gênant le passage. Il est probable que ce changement d'environnement est trop brutal par rapport aux conditions de l'évolution. Toutefois lorsque les évolutions ont été lancées depuis le début avec les obstacles, les performances restent légèrement inférieures, mais très similaires, aux performances déjà obtenues, ce qui est compréhensible du fait de la gène occasionnée par la présence de l'obstacle.

Nous avons obtenu des résultats plus intéressants lorsque les robots ont été placés dans une arène trois fois plus large que l'arène initiale. Les meilleurs résultats de généralisation sont obtenus par les contrôleurs classiques avec seulement une petite baisse de fitness (quelques %), et une centaine de générations supplémentaires lui permettent de retrouver ses meilleures performances.

En terme de capacité de généralisation pour cette phase, les contrôleurs symboliques

arrivent en deuxième position, perdant environ 10 % de leurs performances originales, puis retrouvant ce même niveau, voire un niveau sensiblement supérieur, après 150 générations d'évolution dans le nouvel environnement.

De manière surprenante, les deux architectures de supervision, placées dans le nouvel environnement plus grand, n'arrivent pas à atteindre de nouveau leur performance initiale, même après une évolution supplémentaire. Les superviseurs symboliques perdent approximativement 12.5 % de leur performance, et n'en recouvrent que la moitié, alors que les superviseurs classiques perdent plus de 20 % et ne les retrouvent jamais.

Ces résultats peuvent sans doute être expliqués partiellement par les comportements obtenus initialement avant la généralisation : les architectures de contrôleurs directs donnent par évolution des comportements qui restent principalement proches de la zone de recharge, en tournant pratiquement autour, et de ce fait ne sont pas vraiment gênés par l'augmentation de la taille de l'arène. Les architectures de supervision, à l'inverse, utilisent beaucoup plus leurs comportements d'exploration. De ce fait, lorsque l'arène est plus grande, ils déclenchent le mécanisme de retour vers la zone de recharge alors qu'ils sont trop loin de celle-ci, et n'ont pas le temps de la rejoindre à temps. Les résultats les plus surprenants sont bien sûr leur incapacité à retrouver leurs meilleures performances après de nombreuses générations supplémentaires – c'est-à-dire à ressortir du bassin dans lequel l'évolution dans une petite arène les a entraînés. Cependant il faut noter que lorsque l'évolution est relancée depuis le début, les superviseurs symboliques affichent toujours des performances supérieures aux autres comportements, qu'ils aient été le résultat d'une unique évolution ou d'une évolution dans une petite arène suivie d'une évolution dans une grande arène.

La différence de comportements avant généralisation explique également les résultats obtenus dans une autre expérience de généralisation. Cette fois-ci, on modifie les paramètres de la batterie, doublant le temps nécessaire à sa recharge, ou doublant la consommation d'énergie, l'autonomie de la batterie étant alors diminuée de moitié (ces deux expériences de généralisation donnent les mêmes résultats).

Ici, les résultats des superviseurs symboliques sont clairement bien meilleurs que ceux des autres contrôleurs : en effet, pour tous les superviseurs symboliques obtenus par évolution initiale, le robot reste dans la zone de recharge jusqu'à ce que son niveau de batterie soit remonté au maximum en utilisant le comportement "stop", et ce quelle que soit la durée de cette recharge.

Curieusement, ce n'est pas le cas de la plupart des superviseurs classiques qui, bien qu'ils pourraient utiliser leur comportement "stop", ne parviennent pas à atteindre l'arène de recharge. D'un autre côté, les deux architectures de contrôleurs directs, ne s'arrêtant jamais sur la zone de recharge, ont beaucoup de mal à s'adapter aux nouvelles conditions.

Toutefois, alors que les contrôleurs symboliques parviennent à survivre pour la moitié des essais, aucun des contrôleurs classiques n'y parvient.

5.7 Une expérience validée sur robot réel

La validation ultime en robotique, et plus particuliérement en robotique évolutionnaire où les évolutions sont simulées, est bien sûr la validation sur robot réel. Dans le contexte de l'utilisation de robots Khepera, nous considérons ici le fait que le contrôleur contrôle effectivement un vrai robot, même si les ordres sont en fait générés sur le PC pilote et non cross-compilés pour être téléchargés sur le robot, le câble étant ensuite retiré.

Cette section présente une expérience utilisant le gripper du Khepera, dans laquelle l'utilisation des superviseurs a accru considérablement la performance des robots, et qui a été validée sur le robot réel. Par manque de temps, cette expérience n'a pas été étudiée en détail, par conséquent, les résultats détaillés ne seront pas fournis. Cependant, les résultats obtenus sont assez explicites et suffisamment encourageants pour figurer dans cette thèse. A noter que cette expérience a été réalisée dans le cadre d'un projet des modules expérimentaux d'électronique (MODEX) de l'Ecole Polytechnique, que j'ai encadré pendant les trois années de ma thèse (les élèves consacraient à ce projet 1 jour par semaine pendant 4 mois). En particulier, dans ce cadre-là, un soin tout particulier a été apporté au calibrage des capteurs notamment, cette étape étant un passage obligé afin que le passage sur robot réel se passe au mieux.

5.7.1 Le ramasseur d'ordure à autonomie limitée

Cette expérience est fortement inspirée de l'expérience du "ramasseur de poubelles" décrite en détail dans la section 5.2.1. Il s'agit pour le robot de retrouver des cylindres dans l'arène et les déposer à l'extérieur, mais une difficulté additionnelle est introduite par l'ajout d'une batterie (virtuelle) comme dans l'expérience décrite section 5.6.1 : le robot possède cette fois-ci une autonomie limitée, et ne peut guère déposer plus d'un objet à l'extérieur de l'arène sans aller se recharger dans les zones prévues à cet effet, situées cette fois aux quatres coins de l'arène (voir fig. 5.14). Comme dans l'expérience de recharge de la section 5.6.1, le robot doit rester dans les zones de recharge un certain temps pour être totalement rechargé.

Les senseurs du robot sont les 8 capteurs infrarouges en mode actif (détection d'obstacles), un indicateur de niveau de batterie, le capteur situé dans la pince indiquant s'il y a ou non un objet saisi par la pince, et un capteur infrarouge dirigé vers le sol, afin que le robot puisse détecter qu'il se trouve dans une zone de recharge (il n'y a pas ici de lumière qui indiquerait au robot la zone de recharge comme dans l'expérience de recharge de batterie de la section 5.6.1).

Les actuateurs du robot sont les deux moteurs, ainsi que les commandes du gripper – ces commandes sont déjà mises sous la forme “de haut niveau” décrite section 5.2 : une commande de saisie et une commande d'ouverture de la pince.

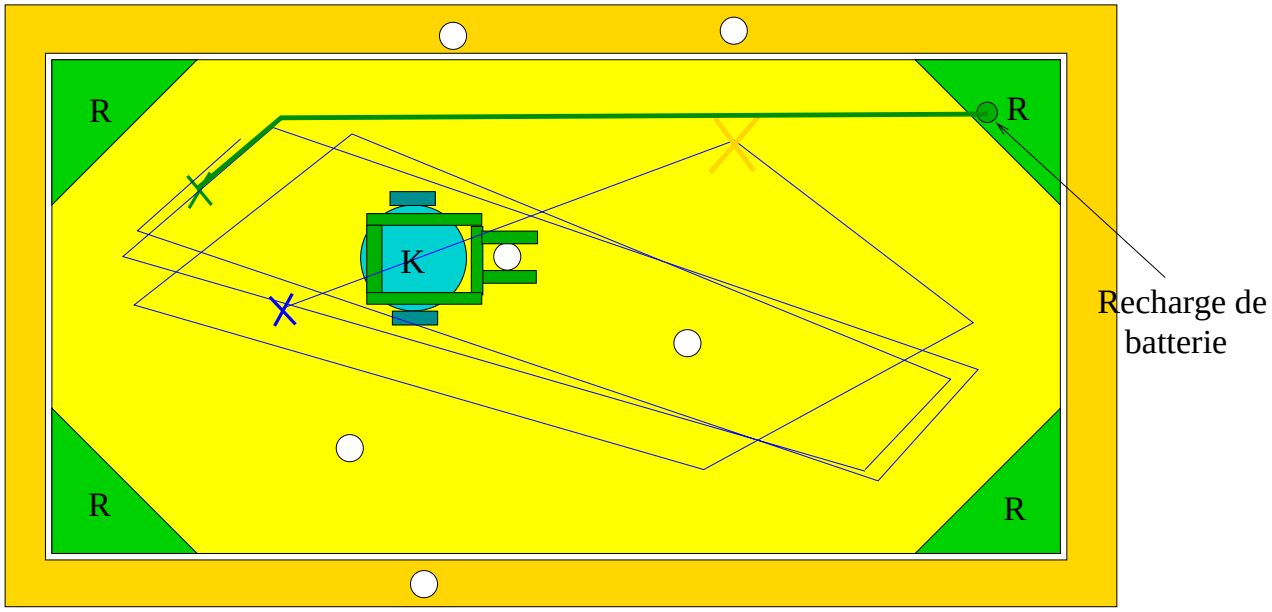


FIG. 5.14 – L'expérience du gripper et des zones de recharge, la croix bleue représente le début de la trajectoire du robot, la trajectoire du comportement de recherche et du dépôt du cylindre à l'extérieur est en bleue, la croix orange représente la position du Khépéra lorsque celui-ci dépose le cylindre à l'extérieur de l'arène, la croix verte représente un décrochage dans la trajectoire, le superviseur vient de permuter dans le comportement de recharge, où cette fois-ci le robot doit raser les murs pour parvenir aux zones de recharge symbolisées par des triangles verts.

5.7.2 Les architectures

Dans un premier temps, un contrôleur classique, possédant deux sorties moteurs et les deux sorties du gripper a été utilisé. Mais aucun résultat probant n'a pu être obtenu dans ce contexte avec ce type de contrôleur. En effet, les zones de recharge étant situées dans les coins, le robot doit s'approcher très fortement des murs pour les atteindre, et cette tactique nuit à l'exploration qui lui permettrait de déceler la majorité des cylindres, qui se trouvent à l'intérieur de l'arène. Dans le meilleur des cas, le Khépéra parvient en effet à attraper et à déposer à l'extérieur de l'arène un des cylindres, mais il ne va pas recharger sa batterie, et n'a alors plus le temps d'en attraper un autre et de le déposer.

Afin de pallier ce problème, une approche de type superviseur a été utilisée (cf. fig. 5.15). Tout d'abord, un comportement de recherche de cylindre avec dépôt à l'extérieur a été construit par évolution, mais sans aucune idée d'énergie ni d'autonomie pour le robot. Parallèlement, un comportement optimal de recharge de batterie a aussi été mis au point

par évolution : à partir d'un point aléatoire dans l'arène, le robot doit parvenir dans un temps limité à accéder aux zones de batterie. À noter que le meilleur moyen découvert par l'évolution pour ce faire est de foncer vers un des murs, puis de le longer de très près jusqu'à ce que son capteur de sol soit activé.

5.7.3 Les passages sur les robots réels

Ces MODEX (modules expérimentaux) ont également été l'occasion d'expérimenter avec les élèves le passage de la simulation informatique à une expérimentation sur un robot réel :

- L'étape principale consiste à essayer de calibrer au mieux les capteurs du robot dans ses interactions avec le monde réel. La notion d'activation des capteurs vis à vis de leur distance face aux obstacles nécessite un gros travail préparatoire afin d'ajuster au mieux la fonction d'activation dans le simulateur.
- Les capteurs possèdent un biais non négligeable (notamment sur la deuxième version des Khépéra, qui, certes, ont la capacité de détecter les obstacles de plus loin, mais au prix de biais plus importants). Or ce biais n'est pas fixe. La solution envisagée a consisté à utiliser un tirage gaussien centrée autour d'une valeur moyenne mesurée, mais cette solution n'est pas totalement satisfaisante, et il reste encore bien difficile de simuler ce phénomène de manière satisfaisante.
- Mais de plus la calibration est très variable en fonction des conditions extérieures. Un changement de luminosité sur l'arène, par exemple, change la valeur de calibration des capteurs (ceci a été expérimenté à nos dépens lors de changements de temps entre deux séances : le Khépéra, pourtant bien calibré lors de la première journée, ne l'était plus du tout par la suite. Et ceci étant d'autant plus vrai pour des expériences utilisant une source lumineuse devant être détectée sur l'arène par les robots. L'idéal pour limiter ces dérives aurait été bien entendu de disposer d'une pièce fermée, sans fenêtre, et dédiée aux expérimentations des robots Khépéra.
- Enfin la qualité des contacts sur les fils reliant le robot au PC maître a entrainé de nombreuses perturbations rendant quelquefois les valeurs lues durant les phases de calibrations totalement incompréhensibles . . . et il a fallu un certain temps pour penser à tout simplement changer le fil et voir ces problèmes se résorber ! Il y a fort a parier que si nous avions disposé de communications radio, entre les PCs et les Khépéra (modules disponibles depuis peu), ces pertes de temps auraient pu être totalement évitées.

Il a donc été impossible de réaliser des calibrations efficaces et pérennes. C'est pourquoi nous avons plutot privilégié dans le cadre de cette thèse les expériences en simulation, où il s'agissait de valider l'efficacité d'architectures originales, et pour lesquelles une calibration exacte n'est pas indispensable (même si celle-ci a tenté d'être la plus fidèle à la réalité). Les

validations sur robots réels, qui représentent un gros travail à chaque nouvelle expérience, ont plutôt été effectuées uniquement par les élèves des MODEX de l'Ecole Polytechnique.

5.7.4 Une validation sur robot réel

Une fois l'évolution terminée, et le comportement du meilleur individu (en simulation) jugé satisfaisant, ce meilleur comportement a été porté sur le Khepera (comme déja mentionné, le Khepera restait relié au PC, mais celui-ci se contentait de transmettre aux actionneurs du Khepera, à chaque pas de temps, les ordres issus du contrôleur en action).

En ce qui concerne le capteur de sol, sur le modèle I du Khepera, il était possible de tordre un des capteurs infra-rouges (arrière par exemple) afin qu'il pointe vers le sol et devienne ainsi un vrai capteur de sol. Mais, outre le danger pour le matériel que représentait une telle solution, elle n'est de toute façon plus possible avec la version II du Khepera à cause de la coque qui lui a été rajoutée. Pour pallier cette difficulté, l'astuce suivante a été employée : un programmeur était chargé d'appuyer sur une touche du clavier dès qu'il voyait le robot sur un des coins noirs de l'arène, simulant ainsi le fait que le capteur de sol venait de passer de 0 à 1023.

Notons enfin que cette entrée représentant le capteur de sol s'est avérée inutile dans le réseau superviseur dans la mesure où elle est présente (et utilisée) dans le contrôleur expert spécialisé dans la localisation et la recharge de batteries.

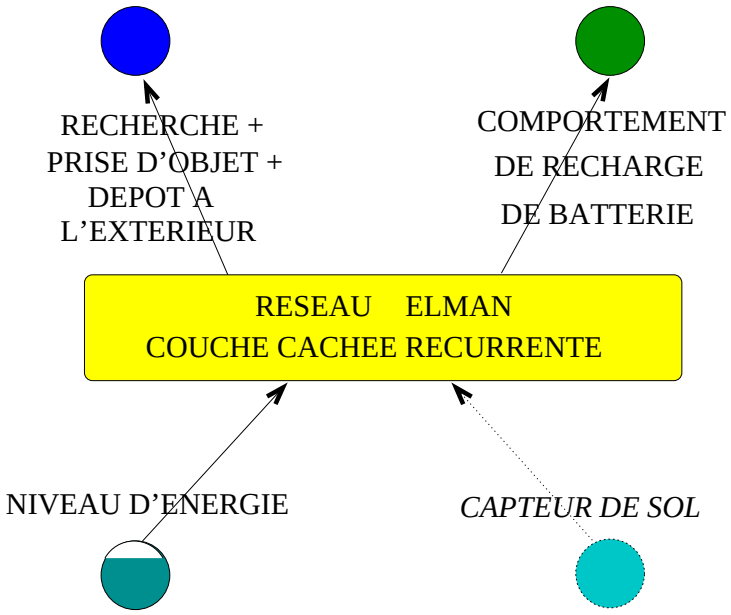


FIG. 5.15 – Le superviseur utilisé pour commander le robot, à noter que l'entrée capteur de sol, n'est pas obligatoire.

Le superviseur qui a été utilisé disposait d'une architecture simplifiée, puisque ne com-

portant que deux entrées (et de fait, une seule s'est avérée nécessaire, comme discuté plus haut). Plusieurs architectures pour le réseau superviseur ont été testées, et la plus performante s'est avérée être un réseau de type Elman, i.e. comportant une couche cachée totalement récurrente. En effet, un réseau totalement réactif enclenche les différents comportements à un seuil précis et invariable : lorsque le robot est suffisamment rechargé (le niveau d'énergie est supérieur au seuil) il ressort de la zone de batterie. Mais il recommence alors à se décharger, et son niveau d'énergie repasse quasiment instantanément sous le seuil. Par conséquent, le robot se met à osciller à la limite de la zone de recharge. L'utilisation d'un réseau Elman permet de créer un déclenchement des réseaux annexes en hystérésis (2 seuils différents). Des résultats satisfaisants ont ainsi pu être obtenus.

5.8 Conclusion du chapitre

L'expérimentation de deux architectures a été effectuée au sein de ce chapitre, les contrôleurs symboliques, qui permettaient au réseau de contrôle de basculer sur des comportements de plus haut niveau (aller tout droit, à droite, à gauche, en arrière) que des commandes moteurs classiques, et d'un autre côté les architectures superviseurs, où le réseau de neurones principal (le superviseur) pouvait choisir à chaque cycle des comportements de haut niveau indépendants et totalement adaptés à la tâche spécifique à accomplir (comme l'exploration d'un environemment, se diriger vers une source lumineuse, ...).

Les contrôleurs symboliques ont montré une plus grande efficacité que les réseaux classiques sur l'expérience simple d'évitements d'obstacle, mais semblent mettre plus de temps à dévoiler leur potentiel lorsqu'il s'agit d'une expérience plus difficile (comme celle de la recharge de batterie). Le disposition de cette architecture ajoute des contraintes sur l'espace de recherche et implique également l'existence de plateaux de neutralités. Tout ceci semblant consistuer à la fois un avantage et un inconvénient selon l'expérience à réaliser. L'inconvénient principal semble être la difficulté de progression rapide lors de l'évolution artificielle (comme il a été vu dans l'expérience de recharge de batterie).

Les superviseurs semblent être une bonne solution lorsque l'expérience commence à atteindre une difficulté importante. L'architecture de supervision de contrôleurs déjà indépendants, a réalisé les meilleures performances sur l'expérience de recharge de batterie. De plus, une des interrogations principales était de savoir comment choisir les comportements à utiliser et pour quel type d'expérience ; l'ajout de nombreux comportements nuisibles a montré que l'évolution possédait la capacité de sélection des comportements principalement utiles pour l'expérience. Ceci constitue un élement clef dans la conception des superviseurs, on peut imaginer que posséder une collection importante de comportements différents permettrait à l'évolution de choisir les bons comportements pour réaliser de nouvelles expériences. Par rapport à l'approche de Brooks, ceci représente une difficulté

moindre et un temps terriblement moins important pour le concepteur. Le fait d'utiliser une collection de comportements permet d'augmenter largement la compréhension du superviseur généré, ce qui semble difficile dans le cadre des approches totalement boites noires, où l'évolution d'un réseau (parfois complexe) est laissé à l'évolution artificielle...

Chapitre 6

Agir, Anticiper, s'Adapter

Ce chapitre s'ouvre (section 6.1) sur une discussion de l'apprentissage en robotique évolutionnaire, en particulier en ce qui concerne les liens entre l'évolution et l'apprentissage. Nous présenterons ensuite deux expériences mettant en jeu une interaction entre apprentissage et évolution qui ont été à la source de nos motivations : la section 6.2.2 sera consacrée à une expérience proposée initialement par Yamauchi et Beer [Yamauchi et Beer, 1994], et revisitée récemment par Tuci, Harvey et Quinn [Tuci *et al.*, 2002]. Dans cette expérience, le robot doit trouver une récompense dont la position dépend d'une variable cachée. La deuxième expérience montrant l'évolution de contrôleurs dotés de capacités d'apprentissage est basée sur les réseaux dits *auto-teaching* proposés par Nolfi et Parisi [Nolfi et Parisi, 1997], et vise à obtenir un contrôleur capable de s'adapter à des changements de couleur des murs de son environnement. Nous nous attarderons sur cette expérience dans la section 6.3, et mettrons en évidence l'instabilité dans le long terme des réseaux résultats. Cela nous amènera à proposer l'architecture *AAA*, titre de ce chapitre, que nous détaillerons section 6.6, avant de présenter les résultats obtenus sur le problème des murs noirs/ murs blancs (section 6.7).

Une partie des travaux de ce chapitre a été publiée à la conférence *PPSN'04* [Godzik *et al.*, 2004], et a également été présenté au *Workshop on Anticipatory Robotic* en marge de la conférence *SAB'2004*.

6.1 Apprentissage et évolution

Commençons par définir précisément ce que nous allons appeler apprentissage dans ce chapitre. En effet, il existe deux échelles de temps dans les techniques de robotique évolutionnaire : l'échelle de l'évolution et celle de la "vie" du robot, i.e. son évaluation à une génération donnée. Du point de vue du concepteur en robotique évolutionnaire,

on peut parler d'apprentissage pour les résultats obtenus aux deux échelles de temps : les poids du réseau contrôleur sont bien "appris" au cours de l'évolution. Toutefois, pour éviter toute confusion, nous avons soigneusement évité jusque-là, et continuerons dans la suite de cette thèse, à utiliser le terme d'apprentissage à l'échelle de l'évolution, le réservant aux phénomènes ayant lieu durant la "vie" du robot. On peut parler d'une vision anthropomorhique de l'apprentissage, limité à l'apprentissage de l'individu durant sa vie.

Nous considérerons donc dans toute la suite l'apprentissage comme une modification du comportement de l'individu pendant sa vie, indépendamment de toute évolution. Plus précisément, l'apprentissage sera caractérisé par un changement du comportement du robot dû à des modifications des caractéristiques de son contrôleur (par exemple les poids du réseau de neurones) ayant lieu durant l'évaluation dudit contrôleur.

Quand on mêle apprentissage et évolution, deux écoles s'affrontent depuis longtemps, les tenants de l'effet Baldwin et les partisans du Lamarckisme. Baldwin [Baldwin, 1896] est convaincu que c'est la *capacité d'apprentissage* qui va influencer la sélection des individus dans l'évolution naturelle. Pour lui, le plus important est de pouvoir s'adapter au monde dans lequel on vit. Mais les éventuelles modifications du génotype engendrées par l'apprentissage durant la vie de l'individu ne sont en aucun cas transmises aux enfants. À l'inverse, le lamarckisme (d'après Lamarck [Lamarck, 1914]) préconise de reporter ces modifications génotypiques dans le génotype des parents afin qu'il puisse être transmis aux enfants. Bien que moins réaliste biologiquement (une image du Lamarckisme pourrait être qu'un individu qui a fait beaucoup de musculation dans sa vie va donner naissance à un bébé très musclé !), le lamarckisme est très utilisé en algorithmique évolutionnaire, et a donné naissance à ce que l'on appelle aujourd'hui les *algorithmes mémétiques*, qui ont obtenu d'énormes succès dans le domaine de l'optimisation combinatoire : dans un algorithme mémétique, l'apprentissage prend la forme d'une optimisation locale, et c'est bien l'individu modifié par cette optimisation locale qui est utilisé pour générer des enfants.

Le contexte de la robotique évolutionnaire est très différent, puisqu'ici le but ultime recherché est précisément d'obtenir des contrôleurs qui s'adpatent à un monde changeant ou inconnu. Il est donc primordial de sélectionner des individus pour leur capacité à s'adapter : le génotype transmis aux enfants sera donc toujours le génotype de parents avant apprentissage – la fitness, elle, dépendant fortement de la capacité d'adaptation montrée par ce génotype.

Dans le cadre des contrôleurs neuronaux, nous avons relevé deux types de mécanismes d'apprentissage qui semblent efficaces, proposés dans des travaux antérieurs. Chacun a démontré des capacités d'apprentissage sur une expérience particulière.

Nous allons considérer en premier lieu les réseaux CTRNNs (Continous Time Recurrent Neural Network), dans lesquels l'apprentissage est basé uniquement sur la modification de la dynamique du réseau, qui permet de modifier le comportement du robot pendant sa vie. Une démonstration en a été faite par Tuci et al. [Tuci *et al.*, 2002] sur une expérience dans laquelle une cible et une lumière sont positionnées aléatoirement d'un côté ou de l'autre de l'arène lors d'une première époque durant laquelle le robot doit apprendre (mémoriser) si la cible et la lumière sont du même côté ou non. Le robot est par la suite noté sur sa faculté à aller directement sur la cible pendant les 14 époques suivantes : tout la capacité d'apprentissage est par conséquent basée sur la mémorisation effectuée durant la première époque.

Cependant, si cette expérience montre effectivement des capacités de mémorisation (donc d'apprentissage), elle ne met pas en évidence des capacités d'adaptation (de généralisation à un environnement changeant ou inconnu). L'autre approche de contrôleurs réellement adaptatifs qui nous a motivé est celle de l'architecture "Auto-Teaching" proposée dans le cadre de la robotique évolutionnaire par Nolfi et Parisi [Nolfi et Parisi, 1997]. Dans cette approche, le contrôleur est constitué de deux parties, optimisées simultanément par algorithme évolutionnaire. Durant la vie du robot (son évaluation), la première partie, appelée le réseau modèle, est figée. L'autre partie du contrôleur, le réseau agent, a pour valeurs initiales les valeurs déterminées par l'évolution artificielle et contrôle les moteurs du robot. Mais ce réseau agent est lui-même modifié pendant l'évaluation du robot en fonction des différences avec les sorties du réseau modèle, par un algorithme de type rétro-propagation du gradient. De cette façon, l'évolution construit un système de décision dynamique, et, pendant la vie du robot, le réseau modèle constitue un attracteur pour le réseau agent qui se modifie en conséquence.

Cependant, pour des raisons de coût-calcul, la période d'apprentissage durant l'évolution des robots représente seulement une petite partie de ce que pourrait être la vie du robot s'il était par exemple mis en exploitation dans le monde réel. Par conséquent, les changements dynamiques potentiels du réseau ne sont examinés que sur une très courte période pendant l'évolution artificielle. Il est possible (et c'est ce que nous avons constaté) que l'évolution artificielle fasse une fois de plus preuve d'opportunisme en sélectionnant des contrôleurs qui sont efficaces à court terme et désastreux à long terme.

Nous nous sommes donc intéressés aux réseaux auto-teaching, en ayant commencé par faire beaucoup d'efforts pour reproduire aussi fidèlement que possible les conditions expérimentales des travaux originaux de Nolfi et Parisi [Nolfi et Parisi, 1997]. Bien que les résultats n'aient pas pu être exactement reproduits, des phénomènes intéressants sont apparus relativement à la stabilité à long terme des contrôleurs. Des expériences inten-

sives montrent très fréquemment que les réseaux auto-teaching malgré de relatives bonnes performances durant la période d'apprentissage, divergent au fur et à mesure que le temps augmente au delà du temps d'apprentissage fixé lors de l'évolution artificielle pour évaluer la performance de l'individu.

De fait, le problème majeur des réseaux auto-teaching est que la rétro-propagation ne s'appuie sur rien de concret puisque le réseau modèle est lui-même le résultat de l'évolution : rien n'empêche les comportements des robots de dériver lorsque leur temps de vie dépasse celui utilisé pour la sélection durant l'évolution.

Nous concentrant sur le problème de l'apprentissage pour l'adaptation, nous proposons une autre source d'adaptation inspiré des modèles des contingences sensorimotrices [O'Regan et Noë, 2001]. L'adaptation de l'architecture est centré sur un module d'anticipation qui a pour tâche de prédire le prochain état des sensations du robot (les valeurs des capteurs) en fonction de l'état courant et des actions de l'agent. Quand l'état des capteurs devient disponible, le module d'anticipation dispose d'un signal d'erreur fiable (la différence entre les prévisions et ce qui s'est ensuite réellement passé) qui ainsi peut être utilisé pour modifier les poids du réseau. Les implantations de cette architecture sont appelés AAA, pour Action, Anticipation et Adaptation.

Le fait de faire des prédictions sur les sensations futures du robot n'est pas complètement nouveau dans la littérature : Nolfi et Parisi [Nolfi *et al.*, 1994a] ont déjà proposé en 1994 une telle approche, en essayant de voir si celle-ci permettait d'améliorer les performances de l'évolution. Dans la littérature, cette architecture s'avère être la plus proche de la nôtre, c'est pourquoi nous la décrirons en détail en soulignant bien les différences essentielles existant entre notre approche et la leur.

6.2 Apprentissage d'une variable cachée par CTRNNs

6.2.1 Les CTRNNs

Les "Continous time recurrent neural networks" (CTRNNs) sont des réseaux récurrents totalement connectés, dans lesquels, en dehors des neurones d'entrée (sensations fournies par les capteurs du robot), tous les neurones sont reliés à tous les autres, y compris eux-mêmes et aux neurones de sortie (consignes pour les moteurs dans le cas présent).

L'état du neurone i à l'instant t est décrit par l'équation différentielle suivante :

$$\frac{da_i}{dt} = \frac{1}{\tau_i}(-a_i + \sum_{j=1}^{N} w_{ij} F_j + \sum_{k=1}^{S} w_{ik} I_k) \tag{6.1}$$

avec :

$$F_j = \frac{1}{1 + exp(a_j + \beta_j)} \tag{6.2}$$

L'indice j décrit l'ensemble des indices des neurones a l'exception des sensations (neurones d'entrée), tandis que l'indice k décrit seulement l'ensemble des sensations (entrées représentées par la lettre I). a_u décrit l'activation du neurone u. Enfin, F_u représente la sortie du neurone u via une fonction sigmoïde classique, β_u représentant le biais du neurone u associé. Cette équation différentielle est résolue par une méthode numérique classique (typiquement la méthode d'Euler).

La mise à jour de l'activation du neurone se fait à l'aide de la discrétisation suivante (Δt étant le pas de temps) :

$$a_i(n+1) = a_i(n) + \frac{\Delta t}{\tau_i}(-a_i(n) + \sum_{j=1}^{N} w_{ij} F_j + \sum_{k=1}^{S} w_{ik} I_k) \tag{6.3}$$

En résumé, les réseaux CTRNNs sont des réseaux totalement récurrents dont la dynamique est différente des réseaux classiques de type Elman, et qui, de par les différentes constantes de temps que chaque neurone possède (les paramètres τ_i de l'équation 6.3) ont une capacité de mémorisation plus importante que les réseaux récurrents discrets classiques (par exemple de type Elman), dans lesquels le fait que tous les neurones aient la même constante de temps entraîne une décroissance exponentielle uniforme sur l'ensemble du réseau de la mémoire. Une démonstration de ces capacités de mémorisation a été donnée par Tuci et al [Tuci *et al.*, 2002], et nous allons maintenant en résumer les résultats en détaillant surtout les conditions expérimentales.

6.2.2 Apprentissage par modifications de la dynamique interne du réseau

[Tuci *et al.*, 2002] ont testé un réseau CTRNN sur une expérience fortement inspirée de [Yamauchi et Beer, 1994] dans laquelle le robot doit faire preuve de capacités d'apprentissage pour réussir. Dans cette expérience (voir figure 6.1), le robot, lors de la première époque, doit déterminer si la cible qu'il doit atteindre se situe à proximité de la source lumineuse ou si au contraire, elle se situe à l'autre extrémité de l'arène. Il doit donc mettre en oeuvre une mémoire relativement stable afin qu'elle puisse subsister durant les 14 époques suivantes, époques durant lesquelles le robot va être noté (il ne reçoit aucune note à la première époque).

Lors de l'expérience originale de [Yamauchi et Beer, 1994], l'utilisation directe de réseaux CTRNNs n'avait pas fonctionné, et Yamachi et Beer avaient dû manuellement mettre au point une architecture neuronale composée de modules spécialisés. Par contre, [Tuci *et al.*, 2002] ont réussi avec l'architecture totalement connectée décrite plus haut, à

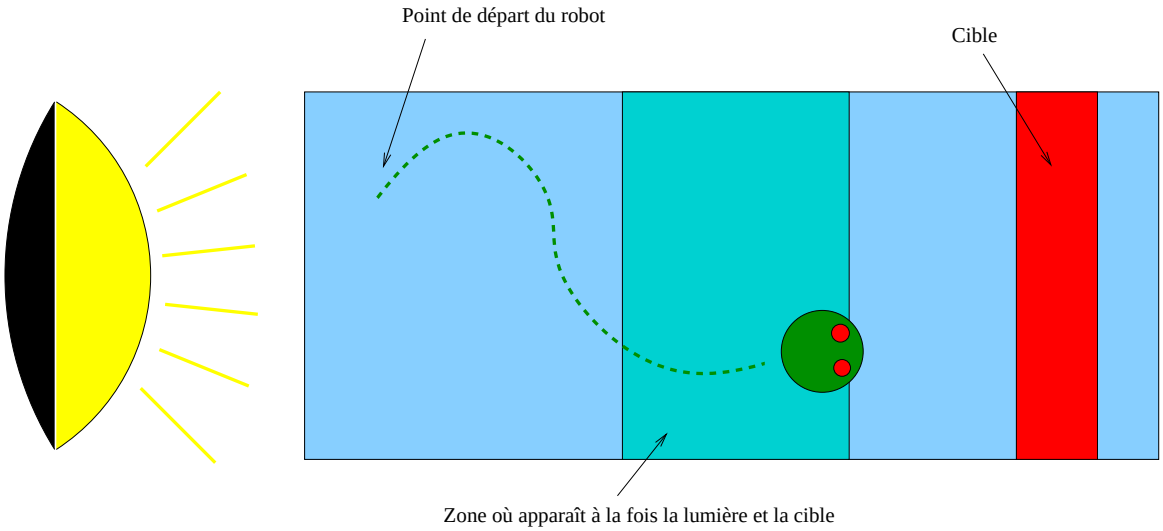


FIG. 6.1 – Expérience de Tuci, Harvey et Quinn.

ce que la moitié de leurs essais aboutissent à un réseau résolvant complètement le problème en trouvant la récompense dans les 14 époques qui suivent la première. Cependant, il faut noter que la fonction fitness mise au point était très peu intuitive et spécialement bien travaillée pour cette expérience. Cela prouve d'une part que de tels type de réseaux sont capables d'apprentissage sous forme de mémorisation d'une variable cachée (la récompense est-elle sous la lumière ou de l'autre côté ?), mais d'autre part que l'intervention humaine est encore nécessaire, quoique sous une forme déclarative plutôt que fonctionnelle (ce point a été discuté plus en détail au chapitre précédent).

Signalons que d'autres travaux partagent le même esprit d'apprentissage : dans l'expérience réalisée par [Blynel et Floreano, 2003], des CTRNNs sont utilisés pour savoir si le robot doit tourner à gauche ou à droite dans un labyrinthe en T, la première époque servant à déterminer, comme dans l'expérience ci-dessus, de quel côté la cible se situe.

6.3 Les réseaux auto-teaching et le comportement à long terme

La deuxième expérience qui nous a inspiré est celle décrite par Nolfi et Parisi [Nolfi et Parisi, 1993], que nous allons décrire dans cette section, en tentant d'analyser les résultats obtenus plus en détail que dans l'article original.

6.3.1 Un expérience avec les réseaux auto-teaching

L'architecture du réseau auto-teaching (cf. fig. 6.2) est constituée de deux modules ayant des entrées et des topologies identiques, ici deux réseaux à propagation directe sans couche cachée. Durant l'évolution, l'ensemble des poids des deux modules est soumis à l'évolution. Par contre, pendant la vie du robot, les poids du premier module (ou réseau *modèle*, sont fixés. Le second module, ou réseau *contrôleur*, est tout d'abord utilisé pour donner les consignes aux moteurs du Khepera, et la fitness partielle pour le pas de temps courant est calculée. Par la suite, les deux sorties du réseau contrôleur sont comparées à celles du réseau modèle, et les différences constatées (les erreurs d'anticipation) sont utilisées pour modifier les poids du réseau contrôleur à l'aide d'une itération de l'algorithme de rétro-propagation du gradient. Durant l'évolution, la vie du robot (calcul de fitness et apprentissage) dure 10 époques.

Comme on peut le voir sur la figure représentant le réseau auto-teaching (fig. 6.2), le contrôleur possède 4 entrées : chaque neurone d'entrée représente la moyenne de deux capteurs du robot (représentés sur la figure 4.3) : le neurone de gauche représente la moyenne des deux capteurs situés sur la gauche du robot, le deuxième la moyenne des capteurs situés sur le devant du robot, le suivant est la moyenne des capteurs de droite, et le dernier neurone d'entrée correspond à la moyenne des deux capteurs situés à l'arrière du robot. Le neurone de biais n'est pas représenté sur la figure, mais les biais sont bien présents, ce qui donne un total de 16 poids entre les entrées et les sorties, plus quatre poids de biais reliés aux sorties moteurs, soit un total de 20 poids.

Au début de chaque époque, la cible et la position du robot sont tirées aléatoirement. Le robot explore une arène de 60 $\times$ 20 cm, durant 500 pas de temps. Le robot doit alors trouver les 2 cm de diamètre de la cible : son centre doit être positionné sur la cible pour que l'époque courante cesse et qu'un bonus lui soit donné. Ainsi, lorsque la cible est trouvée, la fitness augmente de 500 moins le nombre de cycles qui ont été nécessaires pour atteindre la cible. Lorsque le robot entre en collision avec un mur, l'époque courante se termine, et aucune fitness n'est récoltée pour cette époque. Notons que la cible n'est pas visible pour le robot, et que la tâche demandée au robot est donc une tâche d'exploration.

Pour examiner la façon dont les réseaux auto-teaching s'adaptent aux environnements changeants, les couleurs de murs changent alternativement à chaque génération entre la couleur blanche et noire (rappelons que les réponses des capteurs infra-rouges sont très différentes pour des murs blancs et pour des murs noirs : les murs blancs sont détectés de beaucoup plus loin que les noirs). Une idée plus précise des distances est donnée sur la figure 6.11 où les valeur d'activation des capteurs dans le simulateur WOB sont représentés en fonction de la distance et la couleur blanche ou noire des murs.

Pour se situer dans les mêmes conditions que [Nolfi et Parisi, 1997], nous avons utilisé un algorithme évolutionnaire de type ES (cf. section 3.2) dans lequel 20 parents donnent

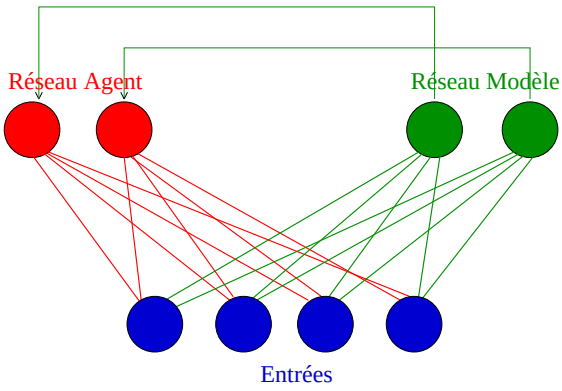


FIG. 6.2 – La réseau auto-teaching.

naissance à 100 enfants. Les poids sont codés dans une représentation réelle et leur valeur est limitée à [-10,10]. Pour coller aux mêmes conditions, un taux de mutation par poids a été introduit et fixé à 10 %. Toutefois, dans le but d'améliorer les performances, quelques petites modifications ont été nécessaires : nous avons utilisé un (20,100)-ES en utilisant une mutation gaussienne avec un écart-type de 0.5 à la place d'une mutation uniforme d'amplitude double, nous avons également utilisé un croisement avec une probabilité de 40 %, absent dans l'expérience originale (nos résultats en ont été légèrement améliorés, sans encore rattraper les résultats publiés, voir plus loin).

6.3.2 Les résultats de l'évolution et la période de généralisation

Nous avons échoué dans la tentative de reproduction exacte des résultats de [Nolfi et Parisi, 1997], remarquant en particuier une très grande variation de la variance de la performance par rapport aux positions de départ du robot et de la cible. Dans de nombreux cas dans notre contexte le robot ne trouve tout simplement jamais la cible – et en effet, il apparaît clairement difficile qu'un robot puisse trouver pendant 10 époques la cible alors que sa position et celle de la cible sont aléatoires dans une arène de cette taille.

Nous avons effectué 12 essais de 1000 générations chacun pour le réseau auto-teaching sur l'expérience de recherche de cible. Les résultats du meilleur individu des 12 essais après les 1000 générations sont visibles sur la figure 6.9. Dans le détail de ces 12 essais, au bout de 1000 générations, on trouve parmi les essais :

- 1 *très mauvais* résultat (fitness stagnant a 1000)
- 3 *mauvais* résultats (fitness d'environ 1500)
- 3 *bons* résultats (fitness aux alentours de 2000)
- 5 *très bons* résultats (fitness proches de 2500)

soit moins de 50 % des essais qui se rapprochent réellement de l'optimum.

Mais le point le plus intéressant concerne le comportement de ces mêmes réseaux au-

delà des 10 époques qui ont été utilisées pour mesurer la performance durant l'évolution. Plusieurs essais ont donc été effectués en obervant le comportement du robot durant 50000 à 100000 époques.

On trouvera des exemples de telles expériences dans l'article [Godzik *et al.*, 2004], mais le manque de place n'a pas permis de publier des analyses poussées ... De plus, lors de la publication de cet article un léger bug dans la mise à jour des poids subsistait, et même si l'instabilité des architectures auto-teaching n'est pas mise en cause, la stabilité des architectures AAA n'est pas aussi éclatante que dans l'article, les résultats exacts seront publiés dans ce chapitre.

Analysons en détail un exemple typique d'une telle expérience : La couleur des murs est changée toutes les 10000 époques, la première couleur utilisée est le blanc. Le nombre d'époque maximal pour ce test de généralisation est fixé à 50000 époques. Les résultats obtenus par les 12 réseaux résultats de 12 évolutions mentionnées ci-dessus sont les suivants :

- 8 réseaux parmi les 12 présentent des collisions avec les murs extrêmement rapidement (tous avant 50 époques, et certains beaucoup plus vite encore). Parmi ces contrôleurs désastreux (et avec les "notations" précédentes), on retrouve le très mauvais, 2 des 3 mauvais, 1 des bons et surtout 4 des très bons (dont le meilleur après l'évolution !).
- à l'inverse, les 4 autres réseaux ne rencontrent pratiquement jamais les murs en 50000 époques – ceci concerne donc l'un des mauvais, 2 des bons et un seul des très bons.

La figure 6.3 permet de visualiser le phénomène pour l'un des individus désastreux. Il faut la lire de la manière suivante : chaque graduation de l'axe des x représente une époque. Une barre positive (vers le haut) signifie que le robot n'a pas rencontré le mur durant les 500 pas de temps de l'époque (la hauteur de la barre donne le temps mis par le robot à trouver la cible, 500 voulant donc dire qu'il ne l'a pas trouvée). Inversement, une barre négative (vers le bas) indique que le robot a heurté un mur – la longueur de la barre donnant le temps auquel ce choc est arrivé.

On voit donc sur la figure 6.3 que le robot en question commence à heurter les murs dès la onzième époque. Cette figure est assez typique de tous les réseaux mentionnés ci-dessus, seule la première époque de collision varie, mais est toujours inférieure à 50 époques, la majorité étant bien inférieure.

De plus, en observant les trajectoires des 12 réseaux au bout de 50000 époques, on constate que

- 2 observent des boucles très resserées – ce qui ne les empèche pas de heurter les murs ;
- 6 vont systématiquement tout droit (où les trajectoires présentent une légère courbure) et donc se crashent dans le mur en face de la position de départ ;

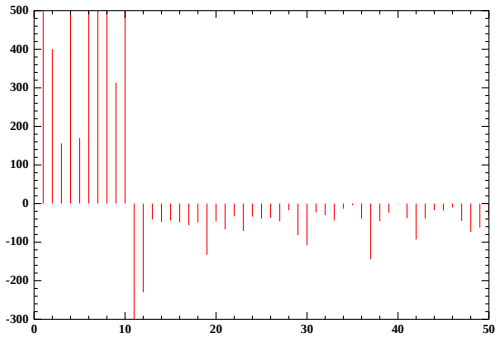


FIG. 6.3 – Performance assez typique d'un essai présentant des crashs d'un réseau auto-teaching lorsque l'on prolonge sa "vie" au delà des 10 époques utilisées durant l'évolution. Dans le cas montré, on assiste à un choc dès la 11ème époque. Voir le texte pour l'explication de la reprśentation.

- 3 donnent de très bonnes trajectoires, similaires à la trajectoire de la première époque sur la figure 6.4.
- le dernier va tout droit, puis s'arrête devant le premier mur qu'il rencontre et se met à tourner en rond.

Un essai assez typique parmi les 4 dans lesquels le Khépéra ne heurte pratiquement aucun mur est présenté sur la figure 6.4. Pour chacune des époques citées, sont représentées sur la trajectoire du robot les erreurs sur chacune des sorties (la différence entre la sortie du réseau contrôleur et celle du réseau moteur – une couleur plus foncée indiquant une erreur plus grande).

Le comportement du Khepera à la première époque, c'est-à-dire celui au vu duquel l'évolution a sélectionné cet individu par rapport aux autres (figure en haut à gauche) consiste à effectuer des arcs de cercles, ce qui est effectivement une bonne tactique pour essayer de trouver la zone cible. Un zoom de ce qui se passe lorsque le Khépéra s'approche du mur (eu haut à droite), montre que la première sortie moteur (celle commandant le moteur gauche) a quelques difficultés pour correspondre à celle du réseau modèle (couleur noire des petits carrés, montrant une forte erreur).

Le robot va alors légèrement modifier son comportement de virage à la proximité des murs, et, à partir de la 200ème époque, le comportement est totalement stabilisé et ressemble désormais fortement à celui observé lors de la dernière époque. On constate également sur la figure 6.4 quelques petits heurts avec le mur quelquefois, lorsque le robot ne parvient pas à tourner du fait de sa légère modification de comportement, mais ceci reste peu fréquent. De plus, les erreurs à proximité des murs ont pratiquement totalement disparu, et ne réapparaissent que faiblement dans quelques cas, en fonction de l'angle d'attaque du robot et de sa position initiale. Un coup d'oeil sur l'évolution des poids et

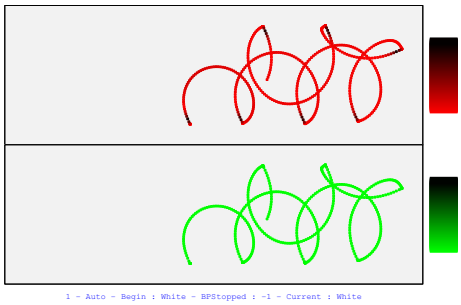
sur l'erreur moyenne (figure 6.5) montre que le réseau présente quelquefois une très faible erreur sur la deuxième sortie, ce qui permet difficilement aux poids de continuer à bouger. Ainsi, sur la figure montrant la modification à travers les époques de la différence entre poids similaires pour les deux réseaux parallèles (6.5-gauche), on constate que 5 poids du réseau sont totalement identiques à ceux du réseau modèle dès les toutes premières époques (différences tendant rapidement vers 0).

Les 5 autres courbes sur cette même figure donnent l'impression de stagner mais lorsque l'on regarde de plus près, elles se rapprochent en fait très lentement de zéro : on se rapproche ainsi très très lentement d'une convergence totale du réseau agent vers le réseau modèle. Cependant les quelques très faibles erreurs, empêchant une progression plus rapide des différences des poids vers zéro, indiquent très précisément que le comportement des deux réseaux modèle et contrôleur sont extrêmement proches.

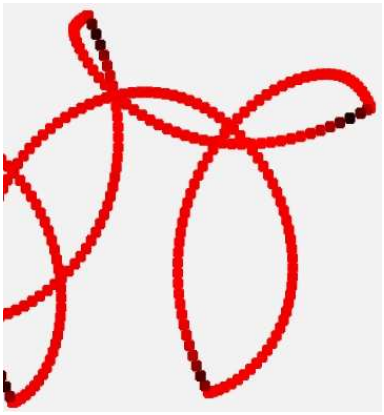
Ceci peut être constaté effectivement sur la figure 6.4 où le comportement à l'époque 50000 et celui à l'époque 200 sont parfaitement identiques – et également très proches du comportement initial, puisque les arcs de cercles se sont simplement un peu agrandis. L'évolution a donc fait en sorte que, même si les poids des deux réseaux ne sont pas parfaitement identiques, les comportements, en revanche, le sont quasiment. A noter que les 4 essais parmi les 12 montrant une stabilité à long terme par rapport à l'évitement des murs présentent la même stabilité phénotypique.

Pour terminer cette analyse des trajectoires et des erreurs faites par les réseaux auto-teaching, précisons que trois essais parmi les douze (deux ne heurtant pratiquement aucun mur, et un qui les heurte pourtant) présentent une convergence totale des poids des deux réseaux modèle et contrôleur avant la 50000ème époque. Un exemple peut être vu sur la figure 6.6. Tous les autres essais présentent une stabilisation très rapide de l'erreur, et les poids similaires entre les deux réseaux semblent converger très lentement vers 0.

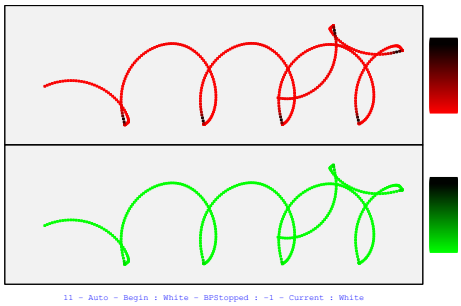
Ce que l'on peut conclure sur l'approche par réseaux auto-teaching, c'est que, bien sûr, tout dépend du réseau modèle. Le problème vient du fait que l'évolution ne teste pas le comportement du réseau modèle, puisqu'en 10 époques, le réseau contrôleur n'a pas le temps de se rapprocher beaucoup du réseau modèle, mais teste ce qui se passe lors des premières étapes de ce rapprochement. Et l'opportunisme bien connu de l'évolution est capable d'utiliser ce phénomène transitoire pour générer des comportements qui ne sont satisfaisants que sur cette courte période de 10 époques. Mais ensuite, et dans tous les essais, le comportement du réseau contrôleur va tendre très rapidement, en quelques centaines d'époques maximum, vers le comportement du réseau modèle. Par conséquent, si le réseau modèle implique un comportement de crash pour le robot, alors le robot va aller très rapidement heurter les murs : à l'inverse, si le réseau modèle implique un comportement qui évite les murs (par exemple par un comportement de "sur-place"),



Epoque 1

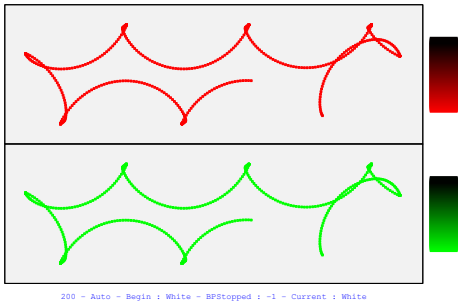


Zoom sur la première époque, on constate qu'il existe une grosse erreur sur la première sortie (noir) lorsque le khépéra s'approche du mur

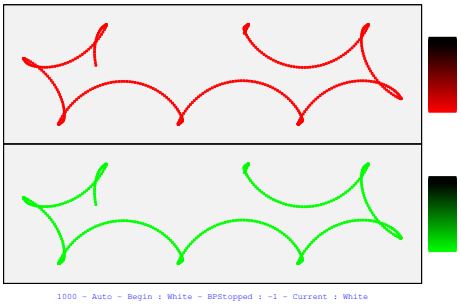


Epoque 11

Epoque 100

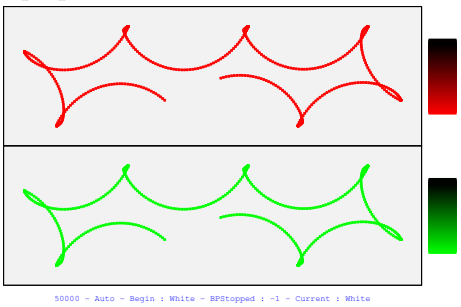


Epoque 200



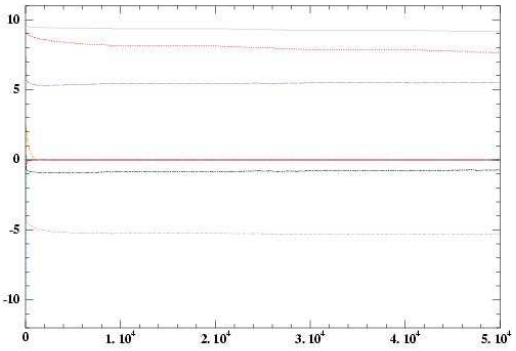
Epoque 1000

Epoque 20000

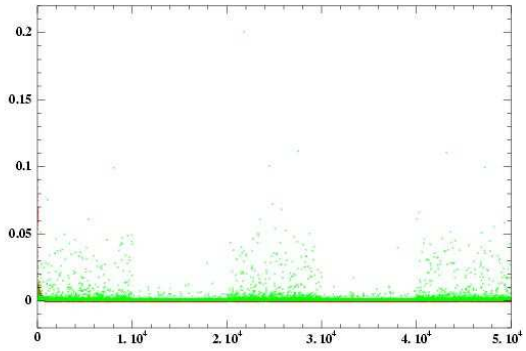


Epoque 50000

FIG. 6.4 – Exemple détaillé d'un essai avec l'architecture autoteaching, le comportement du robot, malgré quelques variations et quelques discrètes prises de murs, reste similaire de la première à la dernière époque.



différences de poids



erreurs sur les deux sorties

FIG. 6.5 – Autoteaching poids et erreurs, progressant **très** lentement

alors le robot tournera très rapidement sur place. Mais dans tous les cas, sauf hasard miraculeux, le comportement du réseau modèle n'a aucune raison d'être un comportement d'exploration du type de celui qui est recherché.

Bien entendu, revenant à la motivation initiale de l'approche auto-teaching, toute capacité d'adaptation est totalement à exclure pour ce type d'approche : la présentation de murs noirs ou blancs ne peut que modifier la vitesse de modification des poids, les exemples d'apprentissage étant plus nombreux avec les murs noirs puisque le robot va aller plus près des murs lors de son virage pour les éviter. Mais encore une fois, le comportement pourra sembler meilleur durant quelques époques (et c'est même cela qui entraînera la sélection de ce réseau parmi les autres), mais ce ne sera pas dû à un quelconque phénomène d'adaptation.

Ainsi, l'approche autoteaching permet au robot d'avoir un comportement donné pendant quelques époques (celui du réseau contrôleur initial), puis de migrer très rapidement vers le réseau modèle, qui constitue son comportement définitif. Pour que ceci ait une utilité, il faudrait pour le moins que le comportement dudit réseau modèle soit également partie prenante de la fitness. Et il faudrait surtout trouver des applications nécessitant ce type de double comportement.

6.4 L'anticipation

Robert Rosen [Rosen, 1985] a donné une des premières définitions d'un système anticipatoire :

"C'est un système qui contient un modèle prédictif de lui-même et/ou de son environnement, qui lui permet de changer son état à un instant donné en accord avec les prédictions du modèle concernant l'instant suivant."

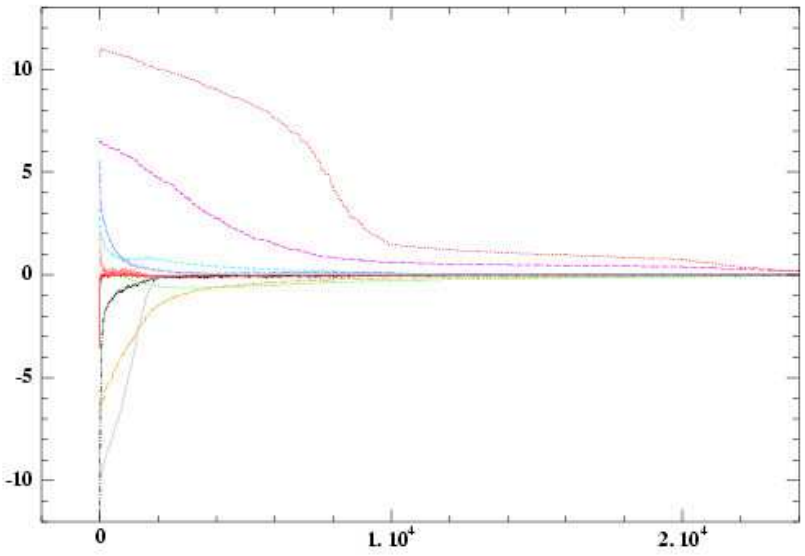


FIG. 6.6 – Un des 3 exemples présentant une convergence complète des poids, les deux réseaux modèle et contrôleur sont en quelques milliers d'époques totalement identiques

Cette définition nous aide à comprendre qu'un système anticipatoire ne s'intéresse pas seulement au présent ni au passé à travers les connaissances qu'il a pu apprendre, mais il va se servir également du futur, ou plus précisément de ce qu'il prédit pour le futur. Lorsqu'on pratique l'anticipation, on va essayer de prédire les modifications qui vont apparaître soit au niveau de son propre contrôleur soit au niveau de l'environnement. Ces prédictions sont réalisées à partir des connaissances acquises et de la vision de soi-même ou du monde environnant du moment.

L'anticipation comporte deux formes, elle peut être effectuée sous une forme passive ou active. Prenons l'exemple d'un appartement situé près d'une ligne de métro : le sol tremblotant indique l'imminent passage d'un métro, et le fait d'entendre les tremblements permet d'anticiper sur le fait que le passage du métro ne va plus se faire attendre très longtemps. C'est de l'anticipation passive.

La forme active de l'anticipation consiste à essayer de prédire les conséquences de ses propres actions. Beaucoup d'expérience sur les animaux dans le but de comprendre leur comportement anticipatoire (actif) ont été menées au cours du 20ème siècle. Parmi celles-ci, on peut citer celle de B.F. Skinner [Skinner, 1948] qui en 1948 étudia les pigeons : il les nourrit à intervalle régulier, et entre les différents renforcements que constituait cet apport de nourriture alors que la faim commençait à se faire sentir, des rituels tels se toucher la tête avec le bec, ou bien encore feindre le picorement, s'installèrent rapidement. B.F. Skinner en déduisit que les pigeons associaient ces rituels à l'apparition de la nourriture,

et ils anticipaient donc sur le fait que ces actions permettaient à la nourriture d'apparaître.

Parmi les approches actives de l'anticipation, les contingences sensorimotrices occupent une place de choix. Ces dernières, d'après Kevin O'Regan et A. Noe [O'Regan et Noë, 2001], constituent en fait des régularités que l'on peut prédire à l'intérieur de nos sensations (comme par exemple la vue, ou le toucher) lorsqu'une action particulière est entreprise (comme avancer, ou bien encore tourner la tête). Les contingences sensorimotrices permettent de définir la façon avec laquelle on peut percevoir un objet dans ce contexte. Ainsi, une chaise n'est pas seulement vue comme une entité avec 4 pieds et un dossier, mais elle est considérée de façon beaucoup plus dynamique comme un objet sur lequel on peut s'asseoir sans risque.

6.5 Architectures neuronales pour l'anticipation

Avant d'introduire notre proposition pour une architecture neuronale pour l'anticipation, nous allons rappeler dans la section 6.5.1 les travaux de Nolfi, Elman et Parisi [Nolfi *et al.*, 1994a] qui constituent, selon nous, l'architecture se rapprochant le plus de celle que nous proposerons section 6.6.

6.5.1 L'architecture de Nolfi, Elman et Parisi

L'idée d'impliquer de l'anticipation dans un contrôleur de robot date de quelques années, et a été explorée à plusieurs reprises (see e.g. [Tani et Kukumura, 94]). Nous allons nous concentrer ici sur celle proposée dans [Nolfi *et al.*, 1994a], afin d'en dégager les différences avec celle que nous proposerons dans la suite de ce chapitre (nous la présenterons en détails section 6.6).

L'expérience décrite dans [Nolfi *et al.*, 1994a] est inscrite dans un monde discret, dans lequel le robot doit effectuer une tâche de recherche de nourriture : il doit parvenir à atteindre un maximum de cases contenant de la nourriture. L'architecture anticipatoire décrite dans [Nolfi *et al.*, 1994a] est présentée sur la figure 6.7 : les deux sensations courantes du robot correspondent respectivement à l'angle relatif entre le robot et la nourriture la plus proche, puis la distance du robot à celle-ci. Outre ces entrées externes à l'instant t, les actions effectuées à l'instant précédent $t-1$ sont également données en entrées du réseau. Enfin, une couche cachée de 7 neurones est reliée à la fois à la couche de sortie standard comprenant les actions à exécuter à l'instant t, à savoir, aller tout droit, à gauche, à droite, ou ne rien faire, mais aussi à deux neurones qui essayent de prédire les sensations à l'instant $t+1$.

Un algorithme de rétro-propagation de l'erreur est utilisé pour modifier les poids reliés aux sensations prédites ainsi que tous les poids reliant la couche cachée à la couche d'entrée.

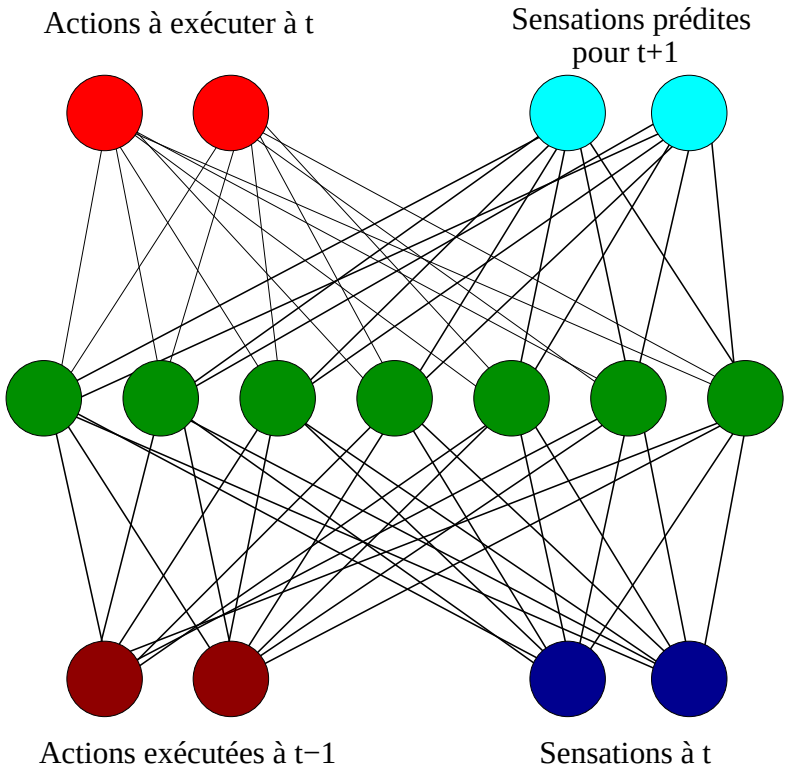


FIG. 6.7 – Architecture anticipatoire présentée par Nolfi, Elman et Parisi en 1994

Sur la figure 6.7, tous les poids étant modifiés par la rétro-propagation de l'erreur apparaissent légèrement plus foncés que ceux qui ne le sont pas, à savoir ceux reliant la couche cachée du réseau et les actions à exécuter.

Les auteurs présentent des expériences réalisées avec des réseaux possédant ou non la couche d'anticipation. Les résultats pour les meilleurs individus sont comparables, la courbe correspondant au réseau sans anticipation progressant légèrement plus vite, probablement grâce à un nombre de poids moins élevé, et du fait que l'apprentissage dû à la modification des poids par la rétro-propagation doit mettre un certain temps à bien se mettre en place. Cependant, cette forme d'apprentissage permettant de modifier les poids permet d'augmenter nettement la performance moyenne de toute la population, bien que la performance des meilleurs individus ne soit que marginalement augmentée. Les auteurs en déduisent cependant que l'apprentissage modifiant les poids du réseau (durant la "vie" du robot) a permis d'augmenter les performances de l'évolution.

Les auteurs constatent également, dans ce papier, que l'évolution permet d'améliorer les performances de l'apprentissage : des comparaisons sont notamment effectuées à ce sujet entre un contrôleur avec des poids totalement aléatoires, un réseau évolué sans anticipation auquel on aurait rajouté de l'apprentissage, et le réseau évolué avec sa capa-

cité anticipatoire. Le contrôleur ayant obtenu les meilleures performances est celui qui a évolué avec sa capacité d'apprentissage. Enfin, dans ce papier, les capacités anticipatoires du réseau sont relativement satisfaisantes (bonnes prédictions des sensations).

Ce réseau nous a fortement inspiré car il apparaît clairement que le fait d'introduire des capacités anticipatoires permet d'améliorer les performances de l'évolution. Néanmoins, deux ingrédients essentiels nous semblent manquer dans ce réseau. D'une part, les poids reliant la couche cachée du réseau et les neurones moteurs ne sont pas modifiés par l'algorithme de rétro-propagation de l'erreur. D'autre part, les sensations prédites n'utilisent pas les actions effectives qui vont être réalisées à l'instant t (que l'on connnait puisqu'on est en train de les calculer), mais uniquement les valeurs de la couche cachée, qui elles-même n'utilisent que les actions à l'instant $t-1$. Enfin, les deux types de réseau (moteurs et anticipatoire) ne sont pas découplés et possèdent de nombreux poids en commun. Prenant en compte toutes ces considérations nous avons essayé de construire un réseau anticipatoire qui nous semblait répondre à ces manques de l'architecture ci-dessus – de plus, nous nous sommes situés dans un monde continu, malgré l'accroissement en complexité que cela peut entrainer. Cela a donné l'architecture AAA, présentée dans la section suivante.

6.6 Action, Anticipation, Adaptation

6.6.1 Raisonnement

L'idée de base des contingences sensorimotrices est la connaissance des conséquences de ses actions sur le monde, c'est-à-dire sa capacité d'anticipation. Il s'agit d'une certaine connaissance du monde, que l'on pourrait qualifier de procédurale par opposition à la connaissance déclarative contenue par exemple dans une carte du monde – comme "au nord de la porte se trouve l'escalier" comparé à "arrivé en haut de l'escalier, si je tourne à gauche je vais me trouver devant la porte".

Ceci étant, il suffit que l'environnement soit modifié pour que ces contingences (et donc les conséquences de ses propres actions) ne soient plus les mêmes. Dans ce cas là, apparaissent des erreurs d'anticipation, et la détection même de ces erreurs devrait permettre de réaliser que l'environnement a effectivement changé, et de modifier son comportement en conséquence : une forme d'adaptation est ainsi a priori possible via l'anticipation.

A noter cependant qu'il y a néanmoins de nombreuses situations parmi lesquelles des erreurs d'anticipation vont inévitablement apparaître sans que cela signifie pour autant que l'environnement a changé, mais qui sont dues à l'existence d'alias perceptifs (*perceptual aliasing*). Ainsi, par exemple, les limitations physiques des capteurs entraînent de fréquentes situations d'alias perceptif : si le robot, muni uniquement de capteurs infrarouges, se trouve suffisamment loin de tout obstacle pour qu'aucun capteur ne soit activé,

il n'a aucun moyen de prévoir le moment où un premier capteur va brusquement signaler un obstacle, et l'erreur d'anticipation est inévitable.

Cependant, dans le cas d'une connaissance parfaite du monde, les cas de changement imprévisibles des sensations constituent des points singuliers dans l'espace des sensations possibles, transitions entre deux plages d'états permettant l'anticipation. Dans ce contexte, dans la majorité des cas, les erreurs d'anticipation peuvent être très instructives, révélant que l'on ne se trouve pas dans l'état dans lequel on pense être – ou que, justement, sa connaissance du monde n'est pas parfaite et qu'il faut la mettre à jour.

Ainsi, l'idée de l'architecture AAA est que son mécanisme d'adaptation ne doit être appliqué que lorsque cela est nécessaire. D'autre part, ce mécanisme d'adaptation doit être basé autant que possible sur des erreurs représentant quelque chose de concret, et donc de vérifiable. En d'autres termes, plutôt que construire un modèle déclaratif du monde, l'idée est de donner au robot un modèle procédural qui lui permettrait de prédire les conséquences de ses propres actions. La plus simple description de ces conséquences, dans la logique des contingences sensori-motrices, peut se faire à travers les valeurs anticipées des capteurs du robot.

D'autre part, d'un point de vue algorithmique maintenant, plaçons-nous dans la même logique que celle ayant conduit à la naissance des réseaux auto-teaching (section 6.3) : nous voulons modifier les poids de notre contrôleur neuronal durant la vie du robot, en utilisant un algorithme de rétro-propagation du gradient, à partir donc d'une erreur commise par une partie du réseau contrôleur, appelé réseau modèle. Or il a été montré section 6.3 que l'utilisation d'un réseau modèle qui est lui-même ajusté par l'évolution n'avait aucune raison de conduire à un comportement qui se stabilise à long terme sur le comportement souhaité, même si sur le court terme il existait plusieurs manière pour l'évolution de résoudre le problème posé. Mais si on remplace ce réseau modèle, quelque peu arbitraire, par un réseau qui prédit les sensations du robot à l'instant suivant, on dispose alors, en tout cas au pas de temps suivant, d'une mesure indiscutable de la justesse de cet oracle, puisqu'il suffit de comparer les sensations prédites avec les vraies sensations constatées du robot. On peut alors utiliser cette erreur comme base de l'algorithme de rétro-propagation du gradient qui va modifier les poids du réseau.

C'est dans cette optique là que nous avons créé une architecture capable d'apprentissage en se basant sur des capacités anticipatoires. Comme dans l'approche auto-teaching, les poids initiaux du réseau de neurones contrôleur sont mis au point par évolution artificielle, et sont ensuite modifiés pendant le test du robot (en ligne).

L'architecture neuronale AAA a donc trois tâches à remplir : l'action (contrôler les moteurs du robot), l'anticipation (basée sur les perceptions du robot comme dans l'approche

décrite section 6.5.1, mais également sur ses actions moteurs) et l'adaptation (basée sur la différence entre les sensations prédites au temps t et les sensations réellement observées au temps $t + 1$).

6.6.2 Le réseau de neurones

L'architecture AAA est ainsi composée de deux réseaux de neurones (cf. figure 6.8) : comme pour le réseau auto-teaching, un second réseau de neurones, le réseau *modèle*, est ajouté au contrôleur principal (le réseau *agent*), et son but est de prédire les valeurs des capteurs du robot au pas de temps suivant. Les entrées de ce réseau sont à la fois la couche cachée du réseau agent (les *perceptions*) et les sorties moteurs (les *actions*), afin que la prédiction soit établie en utilisant à la fois les perceptions du robot et les actions qu'il va entreprendre – puisqu'elles sont disponibles.

Les entrées globales du réseau AAA sont les valeurs des capteurs (les *sensations*), et éventuellement, les valeurs des erreurs d'anticipation sur les capteurs au temps précédent (les *Deltas*).

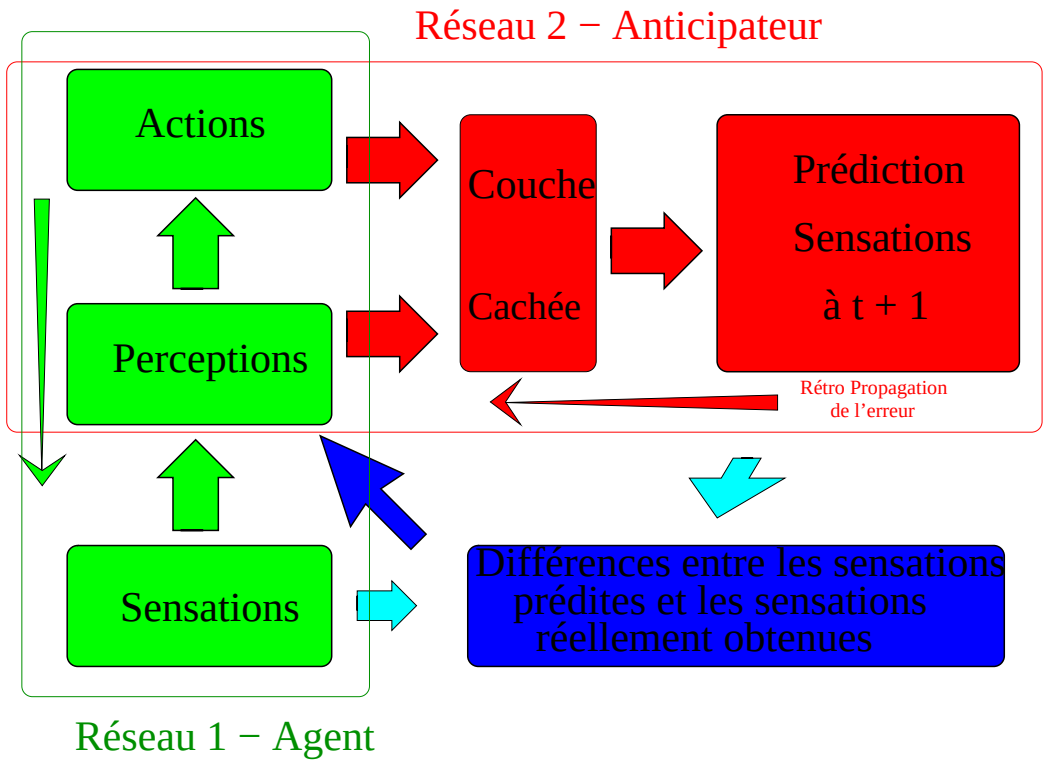


FIG. 6.8 – L'architecture AAA utilisée, le premier réseau (à gauche) constitue le réseau agent, le deuxième réseau (en haut) consitue le réseau anticipateur. La rétro-propagation de l'erreur modifie les poids des deux réseaux.

Tous les poids des deux réseaux sont soumis à évolution : pendant la vie du robot (son

évaluation), les prédictions sont comparées aux valeurs exactes reçues par les capteurs au prochain pas de temps, et les résultats de ces comparaisons sont utilisés par l'algorithme de rétro-propagation du gradient, qui ajuste à la fois les poids du réseau modèle et ceux du réseau agent, comme indiqué sur la figure 6.8.

Dans l'expérience considérée dans la section suivante, et qui reprend l'expérience de Nolfi et Parisi décrite dans la section 6.3, le Khepera possède 4 "capteurs" (en fait, 4 paires de capteurs infrarouges), et 2 sorties (les deux consignes pour les moteurs). Par conséquent, le réseau agent doit avoir 4 entrées, et la partie anticipatoire du réseau modèle doit avoir 4 sorties. D'autre part, nous avons choisi arbitrairement (après quelques essais initiaux, cette valeur n'a plus été modifiée) de mettre 5 neurones dans chacune des couches cachées, celle du réseau modèle et celle du réseau agent.

En considérant que chaque neurone possède un biais, le réseau résultant possède donc 5×5 (flèche 1 + biais sur la figure 6.8) (il faut rajouter 4×5 si on utilise la partie Différence (Delta) (flèche 6)) + 6×2 (flèche 2 + biais) poids pour le réseau agent, ainsi que 8×5 (flèches 3 et 4 + biais) + 6×4 (flèche 5) poids pour le réseau modèle. Tous ces poids sont soumis à la rétro-propagation basée sur les erreurs sur la prédiction des capteurs.

6.7 Architecture AAA, adaptation et robustesse

6.7.1 Conditions expérimentales

Nous avons utilisé ici l'expérience de recherche de plot, la même expèrience que celle utilisée pour le test des réseaux auto-teaching (section 6.3). La performance des individus est toujours évaluée sur 10 époques, le temps calcul pour une expérience est de l'ordre de deux jours, ce qui n'a pas permis d'effectuer de très nombreux essais. Cependant quelques résultats intéressants sont apparus.

6.7.2 Principaux résultats

Douze essais ont donc été effectués pour le réseau AAA de base, ainsi que douze autres pour le réseau AAA comportant en entrées supplémentaires les différences d'anticipation constatées (les *deltas*). L'ensemble des résultats détaillés est fourni dans les annexes A (AAA de base) et B (AAA avec deltas).

Une première synthèse des résultats du point de vue de l'évolution, c'est-à-dire représentant la fitness (évaluée sur 10 époques tout au long de l'évolution) en fonction du nombre d'évaluations, est présentée figure 6.9 : on constate immédiatement que les performances de l'architecture AAA de base sont largement supérieures à celles de l'auto-teaching, et légèrement supérieures à celles de l'architecture AAA avec deltas.

Le détail des résultats montre que les 12 essais de l'architecture AAA de base ont été très bons (avec les critères définis section 6.3, c'est-à-dire atteignant une fitness d'environ 2500). Dans le cas de l'architecture AAA avec deltas, 11 essais ont été très bons, mais un a été très mauvais (sa fitness stagne à 1000, voir B.7). Quant à la manière dont ils atteignent leur fitness maximale au cours des générations, si certains essais montrent quelques difficultés pour décoller et stagnent à 1000 pendant plusieurs dizaines de générations, mais après 300 générations tous atteignent 2500 – à l'exception bien entendu de celui qui ne dépasse jamais 1000.

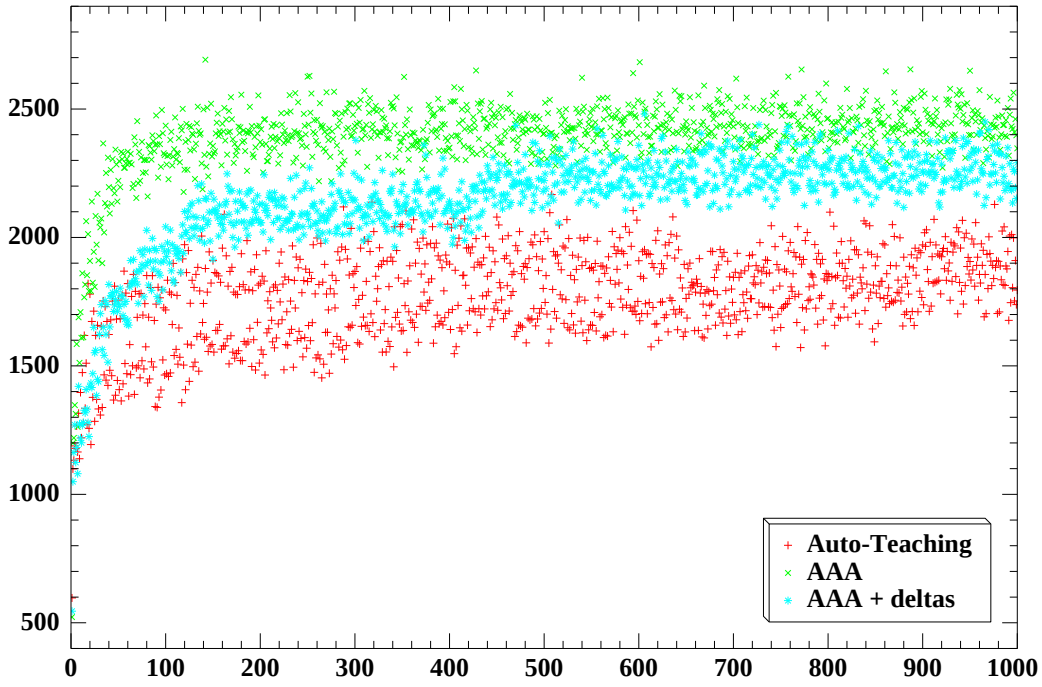


FIG. 6.9 – Résultats pour les 3 architectures, autoteaching, AAA, et AAA + deltas

6.7.3 Résultats sur le long terme

La principale critique que nous avons émise à propos des réseaux auto-teaching concerne leur comportement à long terme, qui, n'étant pas pris en compte dans la fitness, n'a aucune raison d'être stable ni de continuer à être en rapport avec le problème posé. Nous avons donc testé, dans les mêmes conditions que les expériences faites sur les réseaux auto-teaching en section 6.3, les 24 réseaux obtenus après évolution des architectures AAA. Tous les essais en généralisation ont été effectués sur 200000 époques (soit 4 fois plus que le nombre d'époques utilisé pour le test des architectures auto-teaching), la

couleur des murs était changée toutes les 10000 époques et la première couleur est le blanc. Tous les résultats détaillés sont visibles en annexes A et B.

Les résultats concernant un réseau donné sont présentés en 2 pages. La première page de chaque essai est composé de 6 figures, la figure en haut à gauche représente l'état du réseau à la fin des 200000 époques : les neurones verts représentent les neurone de biais (dont la sortie est toujours égale à 1), les neurones de couleur rouge sont les neurones classiques et enfin les neurones violets sont uniquement réservés aux neurones d'entrée représentant les deltas. Les poids sont repésentés par les traits de connexion entre neurones, avec les conventions suivantes : la couleur représente le signe (bleu pour négatif et rouge pour positif), et la taille du trait donne une indication de la valeur du poids (l'épaisseur augmente tous les multiples de 2, avec une limite à 12 pour permettre une bonne lecture visuelle des poids (certains peuvent atteindre des valeurs de l'ordre de 30 ou 40 en valeur absolue).

La figure en haut à droite représente la modification des poids du réseau d'anticipation à travers les époques. La figure au milieu à gauche indique la modification des poids du réseau de contrôle à travers les époques.

La figure au milieu à droite indique l'erreur moyenne commise sur chaque prédiction des entrées à $t + 1$: ces erreurs sont calculées en additionnant toutes les erreurs obtenues pendant l'époque, et en divisant cette somme par le nombre de cycles effectifs (i.e. si le robot heurte un mur au bout de 20 cycles, le nombre de cycles effectifs est 20). La figure juste en dessous représente l'erreur maximale obtenue lors de l'époque en question.

Enfin, la dernière figure, en bas à gauche, utilise la même representation que celle de la figure 6.3 section 6.3.2 : un trait de l'histogramme positif indique le temps que que le robot a mis pour trouver la cible (cette valeur est égale à 500 si la cible n'a pas été trouvée) ; s'il est négatif, ceci indique le temps avant le crash contre un mur.

La deuxième page représente des morceaux choisis de trajectoires du robot lors d'une époque précisée, les couleurs représentant l'importance des erreurs de prédictions – plus la couleur est foncée, plus l'erreur est grande.

6.7.4 Analyse des résultats

Signalons tout d'abord que, parmi les 24 essais (comprenant à la fois ceux de l'architecture AAA de base et ceux de l'architecture AAA avec deltas), aucun n'a de comportement aussi catastrophique que les comportements constatés pour les réseaux auto-teaching, heurtant les murs dés la onzième époque. Dix essais en particulier ne heurtent pas du tout les murs en 200000 époques (voir A.7, A.8, A.9, A.10, A.11, A.12, B.2, B.4, B.8, B.10).

Par contre, une dizaine de réseaux se mettent à heurter les murs avant 5000 époques (c'est le cas des essais A.1, A.2, A.3, A.4, A.5, A.7, B.4, B.7, B.11, B.12) et certains

commencent même à heurter les murs avant 500 époques).

Dans les cas où le robot ne heurte pas du tout les murs, on pourrait croire à un comportement stable du début à la fin de la part du khépéra. Cependant, en y regardant de plus près, les erreurs ne sont pas nulles, et les poids continuent à se modifier. Le robot s'enferme progressivement dans un comportement qui, en quelques centaines d'époques, commencent à tourner légèrement tout en évitant les murs, puis les virages s'accentuent, le robot finissant par tourner en rond. Une fois dans cette phase, les poids ne se modifient plus en profondeur, en tout cas pas suffisamment pour que le comportement du robot se modifie : cette phase, que l'on retrouve assez fréquemment, semble être un bassin d'attraction très large dans lequel le réseau contrôleur se trouve assez facilement piégé. Il est d'autant plus difficile d'en sortir qu'une fois que le robot se met à tourner sur lui-même, il se retrouve la plupart du temps à une distance importante des murs, et les futures sensations se réduisent à prédire le bruit des capteurs, qui reste (malgré la mauvaise qualité de ceux-ci) tout à fait négligeable, et ne suffit pas à modifier suffisamment les poids pour espérer voir le comportement évoluer.

Les autres comportements asymptotiques sont constitués de crashs en allant tout droit, ou en imprimant une légère courbe à la trajectoire du robot. Finalement l'essai le plus stable où le comportement initial est le plus préservé est l'essai B.5 dans lequel le comportement reste quasiment identique à celui de la première époque et ne dévie que très peu, même au bout de 200000 époques.

Ce problème nous semble en fait inhérent au principe de l'algorithme de rétropropagation du gradient. En effet, si l'on regarde en détail les poids allant vers les sorties du réseau en fin de période de généralisation (après 200000 époques), on s'apperçoit que ces poids sont largement et majoritairement bleu, ce qui sature de fait les sorties du neurone à zéro. Dans le cas de la sigmoïde, les sorties deviennent rapidement, lorsque la valeur absolue de la somme pondérée des entrées augmente, très proches de 0 ou de 1. D'une part cela permet d'avoir des sorties tranchées, qui facilite le contrôle robotique. Mais en contrepartie, dans toutes les étapes du calcul du gradient, la dérivée de la sigmoïde qui intervient dans la mise à jour des poids devient terriblement proche de zéro. Ceci entraîne un gradient très faible, et malgré les erreurs d'anticipation (visibles sur les deux graphiques reproduisant les erreurs moyennes et maximales), la modification des poids s'effectue à une vitesse extrêmement lente et ne parvient pas à modifier significativement le comportement : les poids restent pratiquement inchangés.

Néanmoins, on constate en ce qui concerne la 4ème sortie (toujours sur cet essai B.5) qu'une centaine d'époques a suffit pour faire basculer la sortie de ce neurone de 1 à 0. Il suffit par exemple de regarder les trajectoires pour s'en apercevoir : au début, la 4ème trajectoire est totalement noire (échec total dans l'anticipation) puis devient rapidement

normale (violet-rose). Il s'agit bien de la sortie la plus instable puisque l'équilibre sur cette 4ème sortie entre les poids positifs et les poids négatifs n'est pas loin. Si les erreurs sur la 4ème sortie (jaune sur le graphique et représentant les capteurs arrières du robot) étaient plus importantes, quelques milliers d'époques supplémentaires devraient permettre des modifications fortes des poids correspondants.

Il ne faut bien entendu pas négliger le fait que les erreurs faibles dans la grande majorité des cas sont induites par la nature même de l'expérience : les capteurs du robot sont en effet la plupart du temps non activés par la présence des murs, et il suffit alors de prédire des sorties faibles pour atteindre une erreur minime.

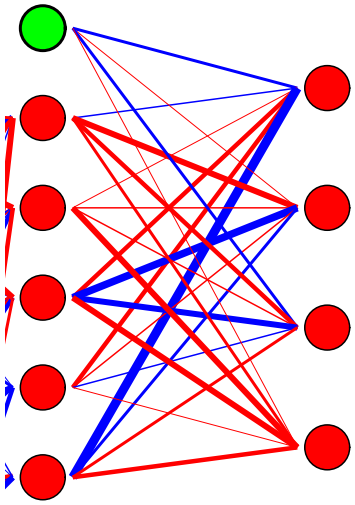
Cet essai (B.5) est cependant un cas isolé puisque dans ce cas, les 3 sorties principales (les plus sollicités pour les évitement des murs) sont saturées du fait d'une somme de poids importante. Dans de nombreux autres essais, cependant, on assiste au même phénomène, mais à l'échelle d'une seule sortie. La figure 6.10 explique pourquoi certaines des entrées ne parviennent pas à être anticipées correctement tout au long des 200000 époques : l'activation arrivant au neurone de prédiction associé est très proche de 1 ou de 0, car les valeurs absolues des poids arrivant à ces neurones sont grandes (traits largement rouges (positif) ou bleus (négatif) suivant les cas.

Ce qu'il faut retenir avant tout, dans tous ces essais, c'est qu'à quelques rares exceptions près, aucun comportement n'est stable très longtemps et le comportement du robot finit toujours par diverger du comportement initial. Bien que dans de nombreux essais, il n'existe pratiquement aucun crash, cela n'empêche pas le robot de modifier son comportement en profondeur pour finir par tourner en rond et faire du sur-place. Il ne semble pas y avoir de différences sensibles à ce propos entre les deux types de réseaux – AAA de base et AAA avec deltas. On constate d'ailleurs sur les réseaux AAA avec deltas, que l'évolution a même tendance à privilégier largement les poids provenant des entrées standards au détriment de ceux mettant en jeu les deltas eux-mêmes, pour assurer un comportement stable au robot.

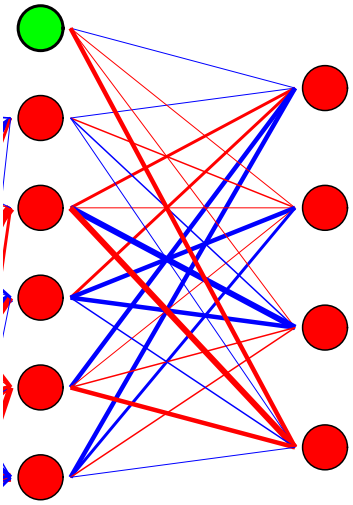
6.7.5 L'adaptation aux murs noirs et blancs

Mais revenons maintenant sur ce qui était à l'origine la principale motivation de l'expérience de Nolfi et Parisi que nous avons reprise, à savoir l'adaptation à un changement d'environnement, ici le changement de couleur des murs.

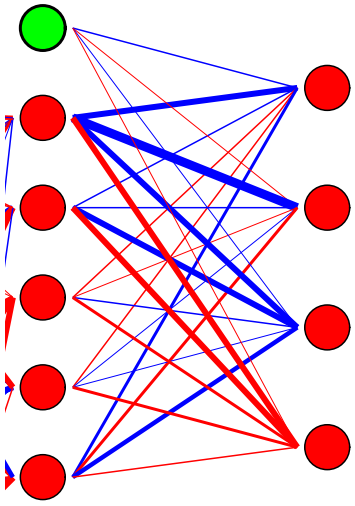
Sur les figures montrant les erreurs commises par le réseau anticipatoire, on constate une différence marquée entre les erreurs commises dans le cas de murs noirs et celles commises dans le cas de murs blancs (ceci est visible sur tous les essais). On pourrait à première vue penser que cela provient d'une phase d'adaptation aux murs noirs ou aux murs blancs. Malheureusement, il semble qu'il n'en est rien.



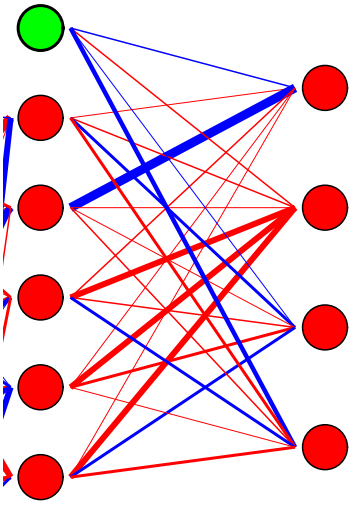
Essai A.3 sortie saturée 4



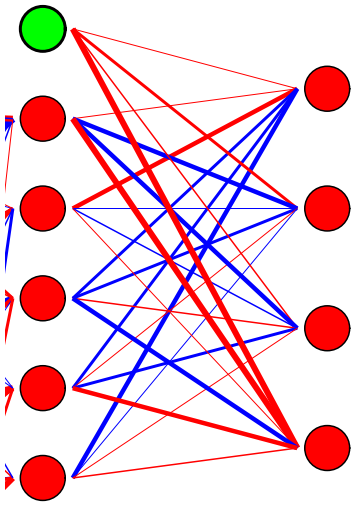
Essai A.12 sortie saturée 4



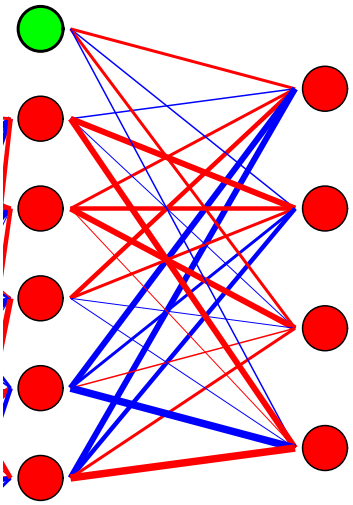
Essai B.6 sortie saturée 4



Essai B.10 sortie saturée 2



Essai B.11 sortie saturée 4



Essai B.12 sortie saturée 3

FIG. 6.10 – Cas les plus marquant où une des sorties anticipe faussement la plupart du temps pour cause d'activations saturées

La figure 6.11 montre les valeurs que peut prendre un capteur infrarouge dans le simulateur en fonction de sa distance au mur (à noter que les capteurs sont légèrement en retrait dans la carcasse du robot, la distance exacte entre le mur et le robot est sans doute d'environ 2 mm plus petite), et la figure 6.12 explique en détail le fonctionnement du calcul des capteurs infrarouges dans le simulateur : un capteur infrarouge émet ses rayons dans un cône d'angle précis, en 2 dimensions. Pour le simulateur, une discrétisation de l'angle en plusieurs faisceaux a été adoptée. Chaque faisceau va apporter sa contribution, en fonction de sa distance au mur, à la valeur du capteur totale. Pour toutes les expériences présentées dans ce travail et utilisant le simulateur *WOB*, une valeur de 60 degrés a été adoptée pour le cône de détection, discrétisée en 20 faisceaux.

Dans le cas de murs noirs, la distance pour une activation maximale est très faible et l'activation dimimue fortement en l'espace de quelques millimètres seulement (voir figure 6.11) : il faut vraiment que le capteur infrarouge soit pratiquement collé aux murs noirs pour être activé à fond. Enfin, il faut rappeler que les 8 capteurs du Khépéra sont couplés deux par deux – les deux capteurs situés sur la gauche du robot, les deux capteurs de face, les deux capteurs situés sur la droite et enfin les deux capteurs situés à l'arrière du robot. Les possibilités que deux capteurs, même situés côte à côte, soient activés pleinement simulatenément, dans le cas des murs noirs, sont extrêmement faibles. Cependant, un examen détaillé de certaines courbes d'erreurs avec les murs noirs montre que les sorties peuvent néanmoins être très proches de 1, ce qui ne peut arriver que pour quelques valeurs particulières de l'angle d'attaque du robot par rapport au mur (par exemple lorsque le Khépéra arrive parfaitement face au mur, avec une direction orthogonale à la direction du mur).

La figure 6.13 explique comment le Khépéra réagit face aux murs respectivement blancs et noirs : le robot est guidé par le réseau agent, et durant l'évolution, il va apprendre à tourner lorsque les sensations atteignent certaines valeurs. Mais de fait, ces valeurs sont totalement indépendantes de la couleur des murs, noire ou blanche.

Ainsi, quasiment tous les essais (A.1, A.2, A.3, A.4, A.5, A.6, A.7, A.8, A.9, A.10, A.11, A.12, B.1, B.2, B.3, B.7, B.9, B.11, B.12) montrent des inflexions similaires : changement de direction générale des poids aux changements de couleurs des murs, un poids par exemple qui est en train de descendre va changer de direction de façon instantanée et se mettre à monter, et inversement. Tous les poids ne présentent pas cette caractéristique mais ceux qui la présentent sont indifféremment dans le réseau anticipation et dans le réseau agent.

Lorsque le robot passe d'un environnement avec murs blancs à un environnement avec murs noirs (toutes les 10000 époques), un changement dans les exemples d'apprentissage a effectivement lieu. On pourrait imaginer, logiquement, que, si le robot avait un comportement suffisamment efficace pour tourner devant des murs blancs en s'en rapprochant

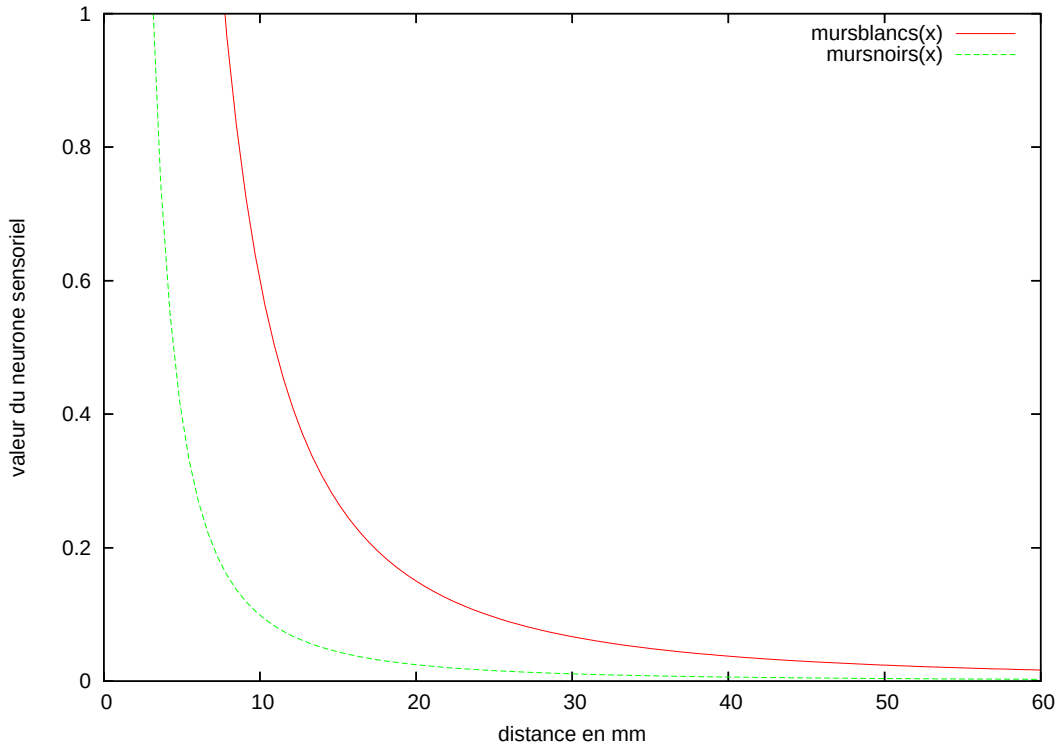


FIG. 6.11 – L'activation des neurones d'entrée dans le simulateur WOB, qui varie avec l'inverse du carré de la distance, auquel il faut ajouter un petit bruit gaussien. L'activation est saturée à 1 et ne peut être inférieure à 0. A noter que les murs noirs utilisés ne le sont pas totalement, mais qu'ils réfléchissent 6 fois moins les rayons infrarouges que les murs blancs.

suffisamment pour bien explorer l'environnement, le passage aux murs noirs devrait engendrer des crashs, ce qui ferait aussi apparaitre de nouveaux exemples d'apprentissage. Cependant, notamment, dans les essais où le robot ne heurte pratiquement jamais les murs (comme par exemple l'essai A.12), certains poids connaissent systématiquement le type d'inflexions décrit ci-dessus. Cette différence dans les exemples d'apprentissage provient bien des erreurs d'anticipation sur les états capteurs.

Néanmoins, de nombreux poids, bien que sensibles aux changements de couleur, montrent des tendances générales nettes, à la baisse ou à la hausse, sur l'ensemble des époques. Un tel phénomène peut être vu par exemple sur l'essai A.4, où le poids de couleur noire tout en haut dans le détail des poids du réseau contrôleur montre une tendance nette à la baisse tout en subissant quelques variations locales dues aux au changement de couleurs : cela signifie que, dans les deux cas (pour les murs noirs comme pour les murs blancs), des gradients assez forts subsistent malgré les changements de couleur, puisque les tendances à la baisse ou à la hausse sont préservées. Dans d'autres cas, cependant, la baisse suscitée par une certaine couleur de murs n'est pas égale à la montée générée

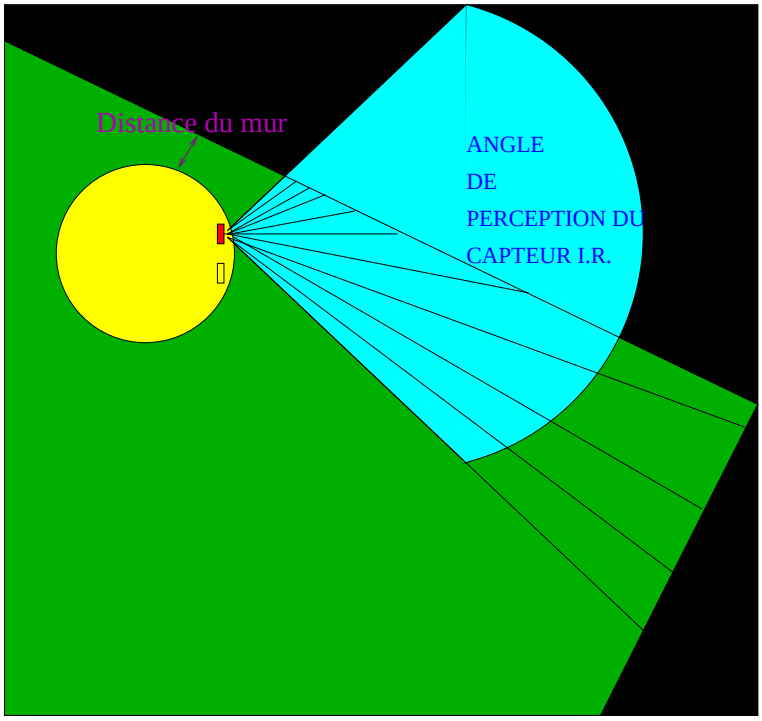


FIG. 6.12 – Les valeurs des capteurs sont calculées en faisant la somme de toutes les valeur données par les faisceaux, dans l'angle sensoriel donné

par l'autre couleur, et cette dissymétrie entraîne une baisse ou une montée globale à long terme. Ces baisses ou montées généralisées sur l'ensemble des époques contribuent donc à un changement radical de comportement du robot, qui peut ainsi, s'il s'agit d'un poids relié à un neurone moteur, saturer un neurone de sortie (ou les deux) à 1 ou à 0, et par là même engendrer une dérive de trajectoire dans des virages accentués pour finir par tourner sur place (voir encore, si les deux neurones de sortie sont bloqués à 1, à pousser le robot à aller tout droit dans le mur le plus proche).

Mais on constate également qu'il peut se faire que, bien que l'inflexion périodique des poids au changement de couleur existe, le poids n'est pas du tout stabilisé. C'est le cas tout à fait impressionnant de l'exemple A.3, où certains poids connaissent les plus grandes variations de tous les essais lors d'un passage d'une couleur à une autre. Les poids n'étant pas stabilisés, il est facile de déduire que le comportement obtenu avant le changement de couleur n'était pas encore stable, et qu'il faudrait plus de temps à la rétro-propagation pour stabiliser l'ensemble du comportement.

Cependant ces petites modifications de poids ne permettent pas d'obtenir des modifications du comportement du robot avec des sorties moteurs aussi tranchées. Le robot peut seulement faire la différence près des murs, grâce aux principes de contingences sensori-motrices. Nous avons ainsi mis en évidence le gros problème de l'expérience de recherche

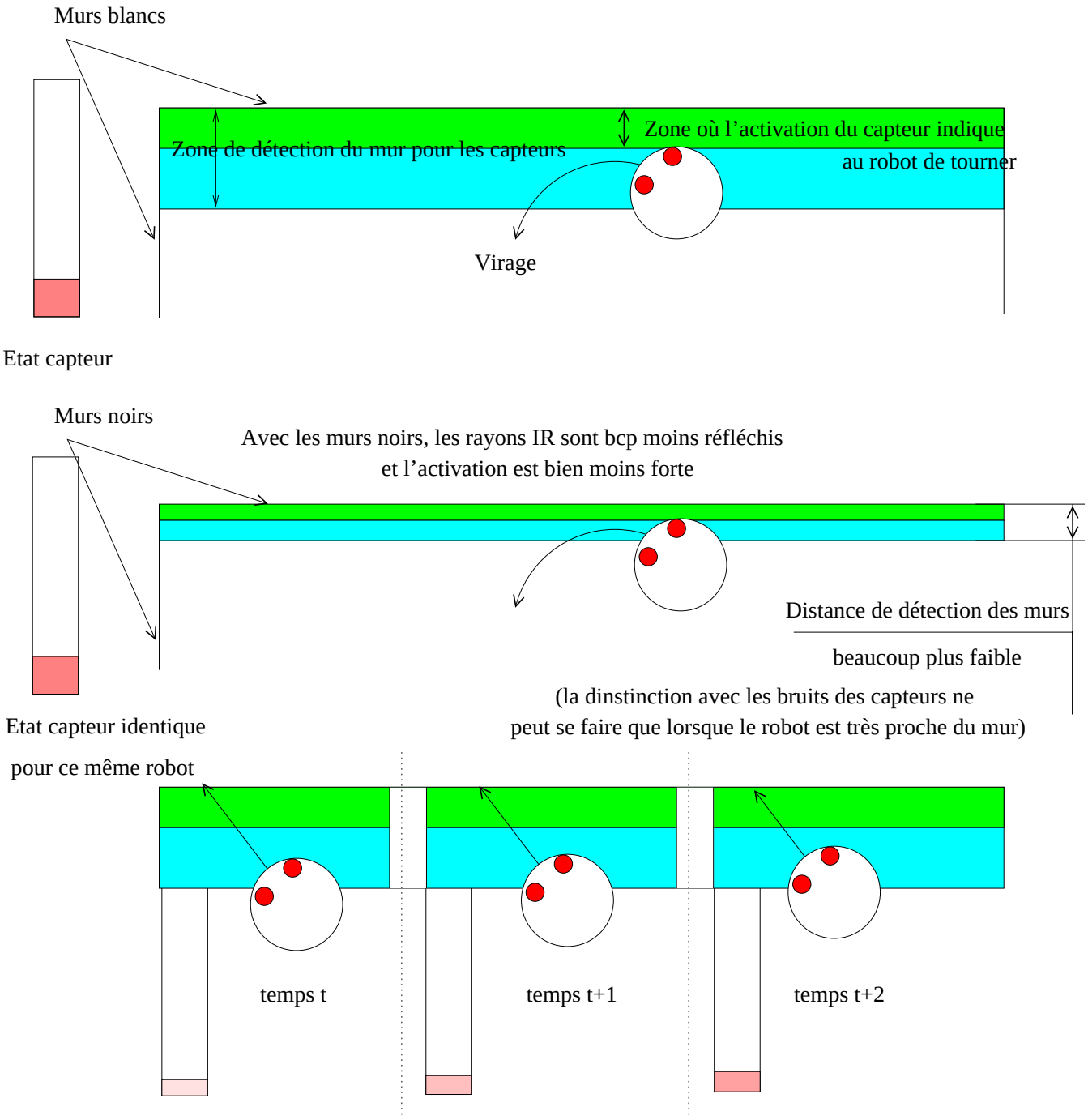


FIG. 6.13 – Les phénomènes sensoriels au niveau des murs

de cible, c'est-à-dire le fait que les évènements "intéressants" (proximité des murs) sont très peu fréquents comparé à l'immense base d'exemple d'apprentissage correspondant au robot situé loin des murs, au milieu de l'arène, là où ses capteurs sont quasiment inactivés. La seule prédiction valable est alors que les capteurs sont restés quasiment nuls, ce qui arrive la plupart du temps. En aucun cas ceci ne peut permettre une réelle adaptation dans des délais raisonnables.

6.7.6 La robustesse

Nous avons également voulu tester la robustesse de l'architecture AAA par rapport à un autre type de changement d'environnement, complètement imprévu celui-là (c'est-à-dire ne s'étant jamais produit lors de l'évolution, à la différence des changements de couleur des murs) qui concerne les capteurs du robot.

Deux séries de tests ont été mis en place pour simuler une panne d'un des capteurs :

– le capteur donne toujours la même valeur (blocage du capteur).

– le capteur fournit désormais une valeur aléatoire (endommagement du capteur).

Des tests ont été effectués parmi pour tous les réseaux issus des essais du réseau AAA, ainsi que sur les réseaux auto-teaching. A la 500ème époque, on déclenche le dysfonctionnement du capteur.

Les réseaux auto-teaching ont montré qu'ils tendaient toujours vers le réseau modèle, quel que soit le comportement de celui-ci en présence de la panne – ce qui n'est pas du tout surprenant, et constitue donc une incapacité de ce type d'architecture à pouvoir réagir lorsqu'un évènement imprévu apparaît.

Dans le premier cas où le capteur est fixe et produit une valeur fixe, tous les réseaux AAA ont montré une énorme capacité à rapidement anticiper ce dernier de façon quasi parfaite et rapide. Bien que ceci soit très intéressant d'un point de vue de l'anticipation, cette adaptation très rapide (quelques centaines d'époques) à cette nouvelle situation empêche le réseau contrôleur de se modifier en profondeur. Celui-ci restant identique, le Khépéra rencontre très rapidement les murs, ne pouvant réagir à cette nouvelle situation.

Le deuxième type d'expérience avec un capteur aléatoire entraîne un comportement totalement différent. Dans ce cas, on assiste à la situation inverse de la précédente : la valeur du capteur étant aléatoire, elle devient par là même totalement impossible à anticiper. De plus dans le cas d'un capteur rendu fou (voir figure 6.14), le comportement du robot est terriblement perturbé par cette entrée du réseau ne signifiant plus rien et le comportement en est très affecté. La figure 6.14 nous montre une trajectoire totalement chaotique du robot. La seule façon pour le robot de se stabiliser serait d'annuler les effets de ce capteur. Malheureusement, la rétro-propagation a peu de chance d'y parvenir et risque fort d'opérer des mouvements cycliques, oscillant entre des erreurs négatives et positives ...

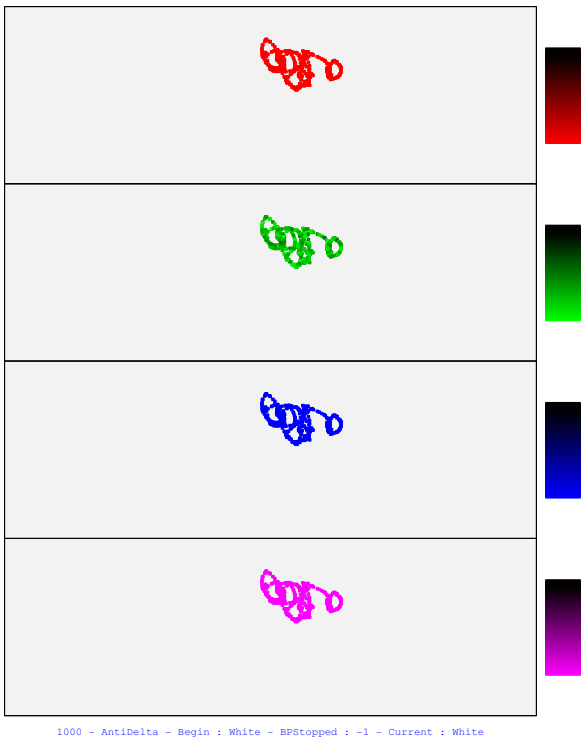


FIG. 6.14 – Cas où le deuxième capteur est rendu fou

6.8 Paramétrage de l'apprentissage et perspectives

Il semble donc que l'expérience proposée par Nolfi et Parisi [Nolfi et Parisi, 1997], même si elle a permis de montrer une certaine robustesse dans le long terme de l'architecture AAA par rapport à l'architecture auto-teaching, soit en fait inappropriée pour tester les capacités d'adaptation d'une architecture de contrôle d'un robot autonome.

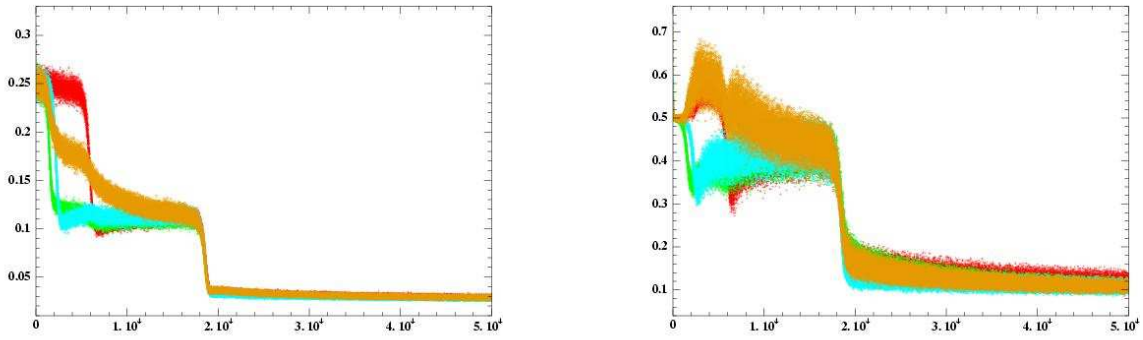
Nous avons cependant voulu tester les capacités d'apprentissage pur de l'architecture AAA, et surtout voir si le paramétrage de cet apprentissage pouvait être amélioré pour augmenter la vitesse d'apprentissage, garante d'une adaptation éventuelle plus rapide, et donc, sans doute, plus efficace.

6.8.1 Capacités d'apprentissage de l'architecture AAA

Dans l'expérience du recherche de plot de la section précédente, très peu d'exemples sont finalement présentés au mécanisme d'apprentissage, du fait que le Khépéra ne voit les murs qu'à une faible distance. Afin de vérifier si l'architecture AAA que nous avons utilisée est tout simplement capable d'anticipation pour des sensations plus variées, nous avons remplacé les entrées sensorielles par des valeurs aléatoires comprises entre 0 et 1. Le

réseau modèle avait pour tâche de prédire (de retrouver) ces valeurs aléatoires des entrées.

A noter que cette expérience a été inspirée par l'expérience dite "du diabolo", dans laquelle un réseau de neurones, composé d'une seule couche cachée de taille inférieure au nombre d'entrées, et d'une couche de sortie de même taille que la couche d'entrée, doit retrouver en sortie les valeurs des entrées (il ne s'agit donc plus ici d'anticipation). Le but de l'expérience du diabolo est de tester la capacité du réseau à compresser ses entrées en moins de "variables" (les neurones de la couche cachée) tout en étant ensuite capable de les décompresser de façon satisfaisante.



Erreurs moyennes Erreurs maximales

FIG. 6.15 – Expérience inspirée du diabolo, cas typique avec paramètres habituels

Cette expérience a ici été effectuée sur les architectures sans deltas, avec pour poids initiaux les valeurs résultant des évolutions sur l'expérience de recherche de plot. Une époque correspond toujours à 500 cycles. Les résultats typiques (erreur de prédiction des entrées par le réseau modéele) sont présentés sur la figure 6.15.

La figure de gauche représente l'erreur moyenne, qui baisse assez rapidement et atteint un minimum aux alentours de 20000 époques pour une erreur inférieure à 0.05 et ceci pour les 4 entrées. Les erreurs maximales connaissent également une forte baisse aux alentours de 20000 époques pour stationner ensuite aux environs de 0.1, ce qui constitue une bonne performance. Ceci montre donc que ce type de réseau est tout à fait capable de retrouver ses entrées. Une petite réserve toutefois concerne le passage à l'échelle : si le nombre d'entrées augmentait de façon conséquente, il faudrait certainement rajouter quelques neurones dans les couches cachées (du réseau agent, et du réseau d'anticipation).

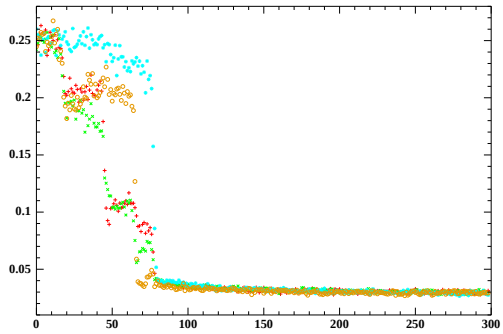
Ce test de vérification a donc été concluant. Bien que ce test ne cherche pas à anticiper, puisqu'on se contente de retrouver les valeurs des entrées, ceci permet de donner une bonne idée de ce que pourrait être un apprentissage en vue d'une anticipation des futures sensations. Et dans ce cas, on remarque que la vitesse d'apprentissage (près de 20000 époques), reste très lente, ce qui constitue un temps relativement important si l'on imagine que l'architecture AAA est portée sur des robots réels. A noter que les résultats des

annexes confirment cette tendance, puisque le robot met toujours un temps relativement important pour trouver un comportement stable. Nous allons voir si les paramètres de la rétro-propagation des erreurs peuvent être ajustés pour pallier ce défaut.

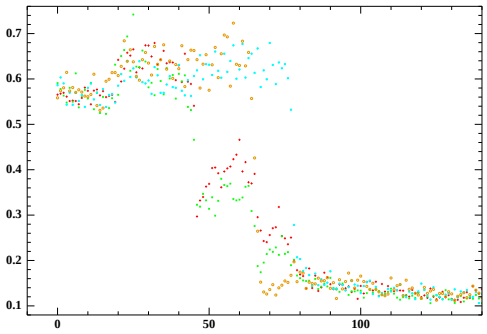
6.8.1.1 Amélioration de la vitesse d'apprentissage

Un coup d'oeil sur la figure 6.10 permet de constater qu'à de nombreuses reprises, une des sorties du réseau a du mal à se modifier du fait d'un gradient trop faible du neurone associé. Ceci peut s'expliquer par le fait que l'activation du neurone est trop forte en valeur absolue, et sa sortie est alors trop proche de 0 ou 1, ce qui donne une dérivée pour la sigmoïde très proche de zéro et inhibe toute progression rapide de l'algorithme.

Nous avons donc effectué des tests empiriques sur l'expérience d'apprentissage précédente : pour augmenter la pente aux endroits de la sigmoïde où l'activation est très présente, le paramètre α (la pente de la sigmoïde à l'origine), a été considérablement diminué. Cependant, la dérivée de la sigmoïde est elle aussi fortement diminuée par cette baisse du paramètre alpha, aussi avons-nous dans ce cas augmenté le taux d'apprentissage, sans quoi l'algorithme restait bien trop lent.



Erreurs moyennes



Erreurs maximales

FIG. 6.16 – Expérience inspirée du diabolo bis, cas typique lorsqu'on diminue fortement la pente à l'origine de la sigmoïde (0.01 au lieu de 1) et que l'on augmente fortement le taux d'apprentissage (500 au lieu de 0.2)) et le taux d'inertie (0.8 au lieu de 0.2).

Les résultats sont visibles sur la figure 6.16, pour la même architecture que pour l'expérience précédente – seuls les paramètres d'apprentissage ont été changés. On constate, comme prévu, une augmentation importante de la vitesse d'apprentissage, puisque les mêmes erreurs sont maintenant obtenues aux alentours de 70 époques au lieu des 20000 précédentes ! A noter que 70 époques est le minimum, obtenus pour plusieurs jeux de paramètres, et pour plusieurs réseaux parmi les 12 réseaux testés. Certains autres parmi les 12 essais ont mis un peu peu plus de 100 époques pour atteindre le même niveau d'erreur, mais aucun n'a dépassé 150. Cette progression reste donc tout à fait remarquable, et le

gain obtenu en coût calcul est considérable.

6.8.2 Retour à l'expérience d'exploration

Au vu des résultats des sections ci-dessus, il est légitime de penser qu'en utilisant des paramètres d'optimisation plus efficaces pour l'apprentissage pur, des phénomènes d'adaptation pourraient apparaître suffisamment rapidement, en entraînant des comportements différents de la part du robot suivant la couleur des murs lorsque l'optimisation a atteint un minimum local paraissant stable.

Malheureusement, le fait de modifier les paramètres du réseau et de l'apprentissage, et notamment la pente de la sigmoïde, a un prix important et de fait totalement prohibitif. En effet, une dizaine de tests a été ainsi effectuée sur l'expérience de recherche de plot avec les paramètres ayant montré les meilleurs résultats en terme de vitesse d'apprentissage, et aucun des essais n'est parvenu à dépasser la piètre performance de 1000 !

L'explication en est relativement simple : la modification de la pente à l'origine de la sigmoïde engendre une presque impossibilité pour le robot d'obtenir des sorties tranchées. Du coup, une petite modification des entrées engendre une très petite modification des sorties et le robot éprouve toutes les difficultés pour arriver à tourner.

Il faudrait donc trouver un compromis entre réactivité du réseau en mode contrôle, et vitesse d'apprentissage en mode adaptation.

Chapitre 7

Synthèse et conclusions

7.1 Résultats et contributions

Dans cette thèse, deux aspects de la robotique évolutionnaire ont été étudiés, le premier portait sur l'étude de la modularité par une approche symbolique, et le deuxième sur une possible application des contingences sensori-motrices à l'anticipation en robotique évolutionnaire.

La première partie de nos contributions (chapitre 5) a été de proposer une approche évolutionnaire de la modularité. Après avoir constaté les défauts des approches modulaires "à la main" de la littérature, en montrant lors de tests poussés que la modularité n'était pas vraiment utilisée, et que l'évolution avait plutôt tendance à privilégier un réseau plutôt qu'un autre dans l'expérience "garbage collector", nous avons proposé une approche située à mi-chemin entre l'architecture de subsomption de Brooks 2.2 et les approches de type boîtes noires courantes en robotique évolutionnaire [Nolfi et Floreano, 2000]. Il s'agit d'un compromis intéressant entre des contrôleurs totalement déterministes, tels ceux proposés par l'architecture de subsomption, et très difficile à mettre en oeuvre, et l'approche totalement boite noire dans laquelle l'évolution est laissée sans contrainte, mais sans guide non plus.

Les contrôleurs symboliques, qui mettent en oeuvre des comportements élémentaires simples comme, le fait pour le robot Khépéra de tourner sur lui-même d'un côté ou de l'autre, d'aller tout droit ou d'aller en marche arrière, ont permis d'obtenir dans de simples expériences d'évitement d'obstacle, des performances comparativement meilleures que celles d'un réseau de neurones équivalent classique commandant les sorties moteurs de bas niveau.

L'étape suivante a été concrétisée par la création de superviseurs (section 5.5). Un superviseur est constitué d'un réseau de neurones standard et a pour tâche, à chaque cycle de la vie du robot, de choisir entre des comportements de très haut niveau (comporte-

ments tout à fait autonomes comme l'évitement d'obstacles, l'exploration de l'environnement,...). Chaque sortie du superviseur symbolise l'utilisation d'un des comportements de haut niveau associé, et la sortie la plus forte décide du comportement qui va être exécuté. Les résultats obtenus sur l'expérience d'exploration avec zones de recharge s'est avérée très encourageante, les meilleurs résultats ont d'ailleurs été obtenus par l'utilisation des superviseurs. A noter que d'une part le superviseur n'est pas gêné par de nombreux comportements inutiles ou nuisibles – ce qui permet de conseiller, dans le doute, de mettre à sa disposition le plus de comportements raisonnables possible. D'autre part, un résutat inattendu a été que l'architecture de supervision a été capable de trouver un comportement permettant d'aller vers la lumière plus efficace que celui proposé (obtenu par évolution), en utilisant à chaque cycle, les meilleurs comportements qu'elle avait à disposition.

Une autre expérience (section 5.7.1) montrant l'intérêt de ces superviseurs, consistait à évacuer des cylindres hors de l'arène tout en ayant une autonomie limitée. L'évolution classique sans superviseur ne parvenait pas à trouver de comportements efficaces (prise des cylindres ET recharge). L'utilisation de superviseurs avec deux comportements adaptés a montré l'émergence d'un comportement adéquat. De plus, cette expérience a été portée avec succès sur des Khépéra réels. Compte-tenu des difficultés techniques que cela a représenté, cela n'a pas été renouvelé pour les autres expériences, bien que nous soyons bien conscients que le passage aux robots réels constitue la validation ultime de toute expérience en robotique simulée.

La deuxième partie de nos contributions a été consacrée à l'adaptation, et à l'apprentissage en ligne, durant la vie du robot. Nous avons tout d'abord testé intensivement l'architecture auto-teaching [Nolfi et Parisi, 1997]. Cette architecture est constituée d'un réseau modèle et d'un réseau de contrôle, et la rétro-propagation de l'erreur est utilisée pour modifier les poids du réseau modèle en prenant exemple sur les sorties du réseau modèle. La tâche demandée à ces robots est la recherche d'une cible dans deux environnements différents, l'un avec des murs de couleur noire, l'autre avec des murs blancs. Le comportement à long terme des réseaux auto-teaching (sur des périodes de temps bien plus grandes que celles utilisées pour l'évolution) a montré une convergence rapide (en quelques centaines d'époques) du réseau de contrôle sur le réseau modèle. Le comportement obtenu est très stable et s'approche fortement de celui du réseau modèle. Malheureusement, ce réseau modèle n'a été, lui, testé que par la dynamique qu'il insuffle au réseau de contrôle, et le comportement qu'il induit en direct n'est pas prévisible. Ainsi, dans une majorité de cas, le robot se met à heurter les murs. Une première conclusion est que les réseaux auto-teaching ne présentent que très peu d'intérêt.

Nous avons alors proposé, pour pallier ces défauts à long terme, et tenter de retrouver de réelles facultés d'adaptation, l'architecture d'anticipation AAA (section 6.6). C'est une

architecture constituée de deux réseaux, un réseau de contrôle d'une part, et un réseau d'anticipation d'autre part, qui à partir des perceptions (couche cachée) du premier réseau et des sorties moteurs tente de prédire les sensations futures observées par le robot. L'algorithme de rétro-propagation est alors utilisé pour modifier les poids de ces deux réseaux partir des sensations réellement observées au pas de temps suivant.

Les tests effectués sur la même expérience que celle utilisée pour le réseau auto-teaching ont montré tout d'abord une meilleure stabilité à long terme, ainsi qu'une tendance des réseaux à l'adaptation à leur environnement. Cependant, les comportements bougent énormément au fil des époques, et bien que de nombreux individus continuent à éviter les murs, ils adoptent souvent après des temps longs des comportements non prévus, comme de se mettre à tourner en rond. Le problème, ici comme dans les réseaux auto-teaching, est que ces comportements asymptotiques ne sont absolument pas pris en compte dans la fitness durant l'évolution, car chaque évaluation durerait alors beaucoup trop longtemps. Les derniers essais suggèrent qu'il faudrait sans doute que la modification des poids soit beaucoup plus rapide pour que l'effet d'adaptation soit plus important, et donc puisse être testé aussi durant l'évolution – mais modifier les fonctions de transfert des neurones, première idée dans ce sens, ne semble pas non plus être une solution, puisque cela empêche toute évolution vers des fitness acceptables.

7.2 Perspectives

Une autre possibilité pour tenter d'une part d'accélérer l'apprentissage, et d'autre part de préserver la stabilité (qui manque par exemple quand on essaye de modifier les pentes des sigmoïdes), mais qui n'a pu être essayée faute de temps, consisterait bien sûr à transformer le réseau AAA en superviseur AAA, en reprenant le concept des superviseurs symboliques (voir section 5.3) au sein de l'architecture d'anticipation. Cela permettrait de faire d'une pierre deux coups : d'une part accélérer l'évolution (si on extrapole les résultats du chapitre 5), et donc peut-être permettre d'aller plus loin durant l'évaluation du robot) ; d'autre part, stabiliser le comportement long terme, en utilisant comme sorties du superviseur des comportements stables (aller vers la lumière, tourner à gauche, explorer, ...), mais surtout non-modifiables quelle que soit la durée de l'expérience. Dans ce contexte, la modification des poids risquerait moins d'entraîner des comportements aberrants comme des collisions multiples ou la fâcheuse tendance à tourner en rond.

Dans le cas des superviseurs symboliques, c'est la sortie la plus importante qui permet de savoir quel comportement sera exécuté à l'instant t. Ainsi, même si les sorties du superviseurs ne varient pas beaucoup, elles peuvent varier suffisamment pour modifier de façon significative à moyen terme le comportement.

Une autre méthode pour améliorer l'algorithme pourrait être d'imaginer un delai et

une décroissance relaxée des erreurs d'anticipation. Dans l'état actuel, lorsque le robot parcours l'arène loin des murs, il ne se passe rien, et les poids n'en sont pas modifiés, même si il y a eu des grandes erreurs d'anticipation aux pas de temps immédiatement précédents. En "étalant" les erreurs dans le temps, le réseau aurait à sa disposition plus de temps pour apprendre, et pas uniquement les quelques instants qu'il passe à proximité des murs. Afin d'optimiser encore plus le processus d'adaptation, on peut également imagner d'ajuster le taux d'apprentissage à l'erreur et d'introduire une diminution du taux d'apprentissage si l'erreur devient faible, afin de déclencher dans ce cas une recherche locale approfondie et ne laisser que la place aux bruits dans les erreurs d'anticipation.

Une autre piste à explorer vient du fait que les erreurs se figent assez rapidement (moins de 100 époques de 500 cycles sur l'expérience du diabolo) : le réseau semble être dans un minimum local, ce qui peut laisser augurer de certaines possibilités de mémorisation à long terme au sein de ce type de réseau.

Reprenons l'expérience de Tuci, Harvey et Quinn (cf. section 6.2.2), où le robot doit, lors de la première époque, repérer si la cible et la lumière sont du même côté ou pas, puis, durant les 14 époques époques suivantes, le robot doit aller directement vers la lumière en appliquant ce qu'il a appris lors de la première époque.

Il faut, pour que l'expérience réussisse, à la fois que le robot puisse modifier très rapidement certains poids, après avoir "réalisé qu'il s'était trompé la première fois, mais également qu'il soit capable de mémoriser de manière durable l'association entre la lumière et la cible.

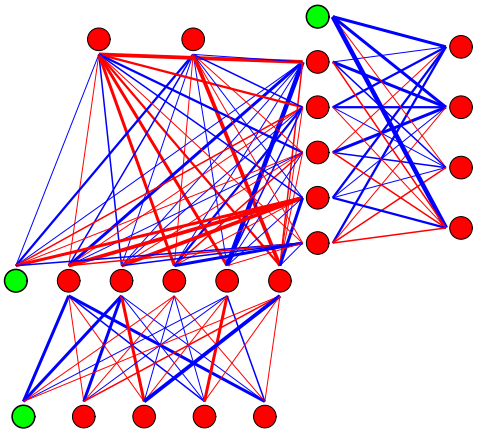
Il est légitime de penser que cette expérience peut fonctionner car cette fois, l'ensemble de l'expérience est exécutée durant l'évolution. Cependant, il faut que les poids se modifient assez rapidement pour que le cas échéant le comportement supervisé ait le temps d'être permuté dans la première époque. L'évolution doit pour cela créer un réseau dont la configuration des poids est capable de s'ajuster en moins de 500 cycles. Pour améliorer les chances de fonctionnement on peut imaginer des techniques d'amélioration de la vitesse dans la rétro-propagation comme l'algorithme de Quickprop.

Mais il faut aussi que le réseau se retrouve dans un bassin d'attraction suffisamment vaste pour qu'il ne risque pas d'en ressortir trop vite, avant le prochain signal de renforcement, la prochaine cible trouvée. Et la convergence observée dans les dernières expériences du chapitre 6 peut donner à penser qu'il est possible d'obtenir une mémoire suffisamment stable via l'anticipation – ce qui n'est pas le cas, dans les réseaux récurrents, dans lesquels la mémoire (l'activité des neurones due à un signal ancien) décroit exponentiellement. Il serait d'ailleurs intéressant de tester les réseaux obtenus par Tuci, Harvey et Quinn à très long terme . . . Bien entendu tout ceci reste à tester, mais l'anticipation pourrait permettre d'obtenir une mémoire à long terme réellement stable.

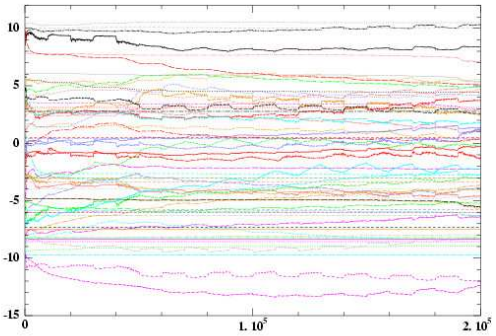
Annexe A

Les résultats détaillés avec l'architecture AAA

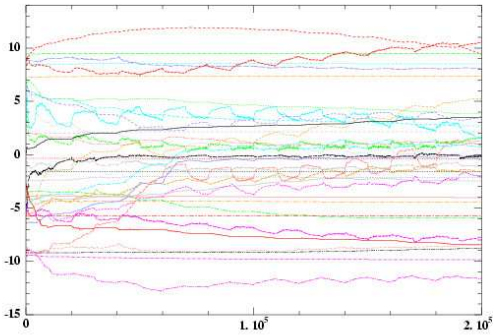
A.1 essai 1



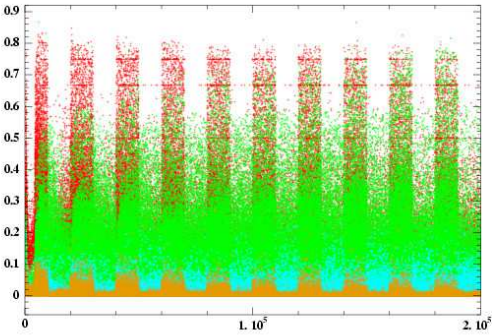
Les poids du réseau à la fin



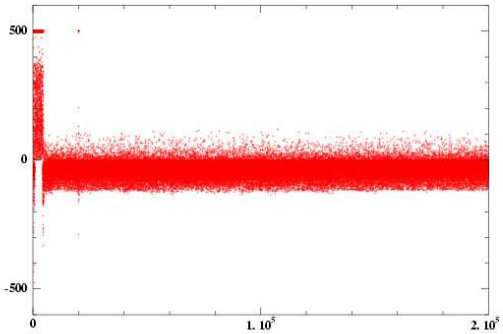
Detail des poids à travers les époques du réseau d'anticipation



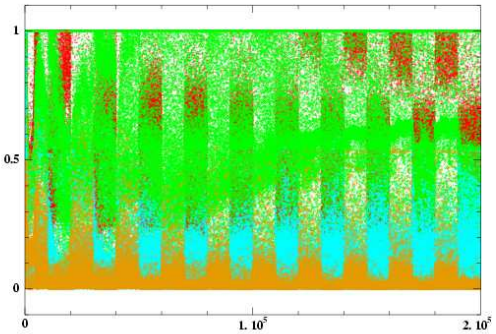
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

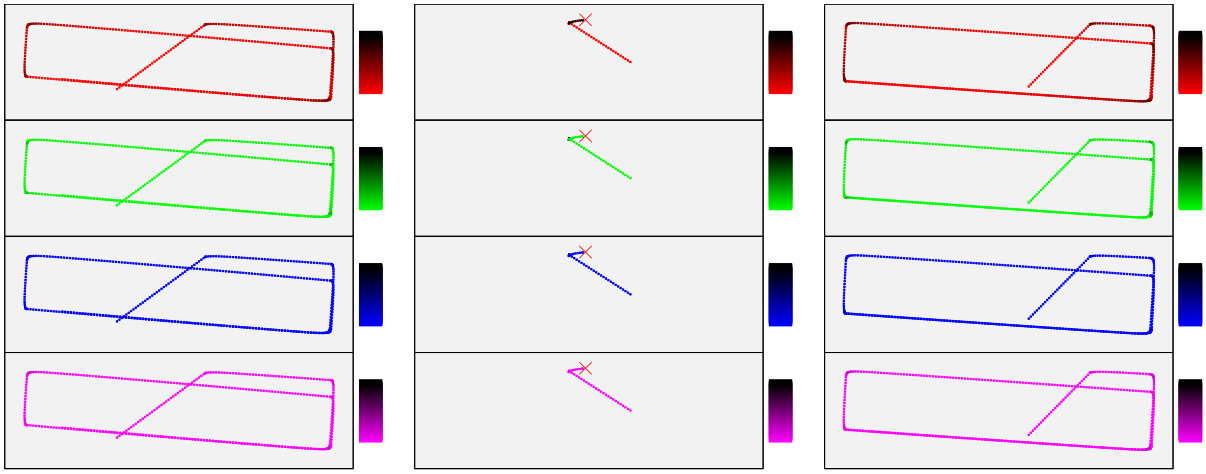


Evolution de la fitness à travers les époques

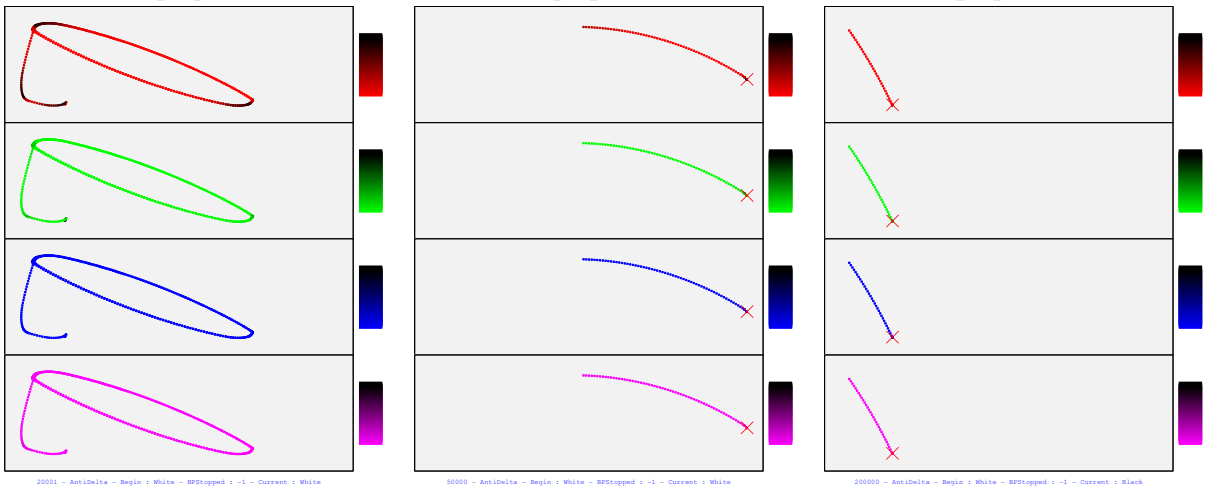


Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. A.1 – Essai 1



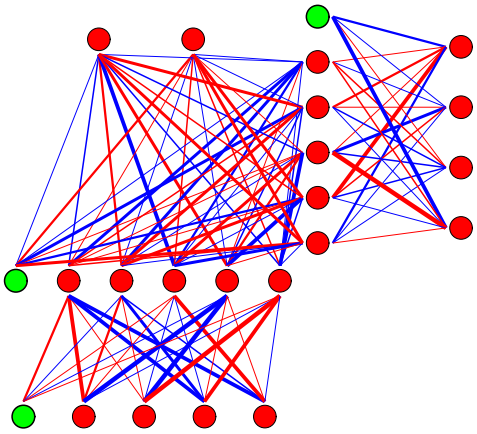
Epoque 1 Epoque 500 Epoque 1000



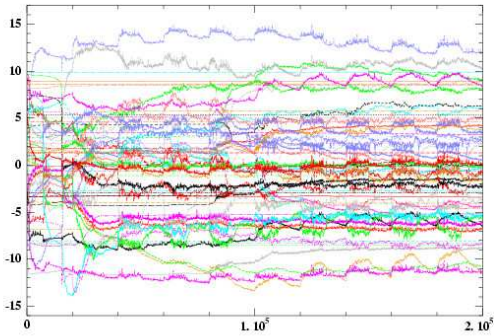
Epoque 20001 Epoque 50000 Epoque 200000

FIG. A.2 – Essai 1

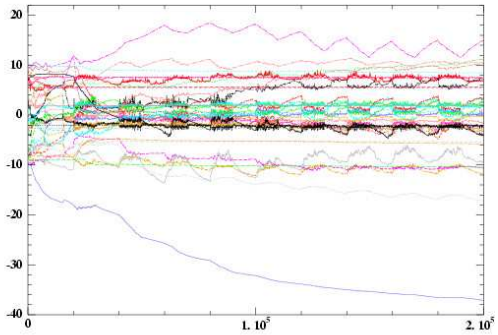
A.2 essai 2



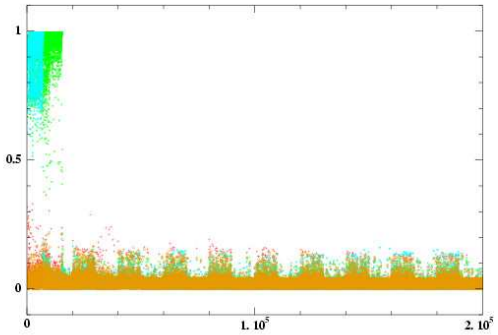
Les poids du réseau à la fin



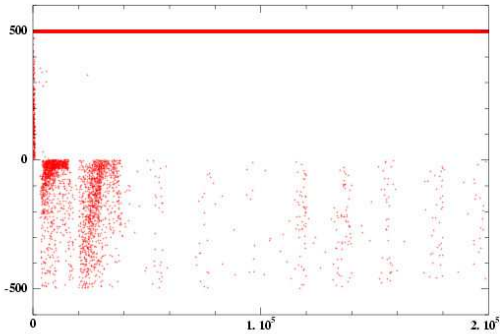
Detail des poids à travers les époques du réseau d'anticipation



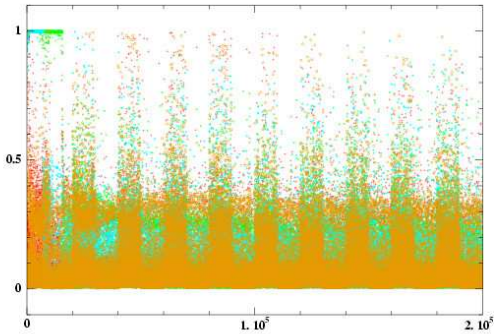
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

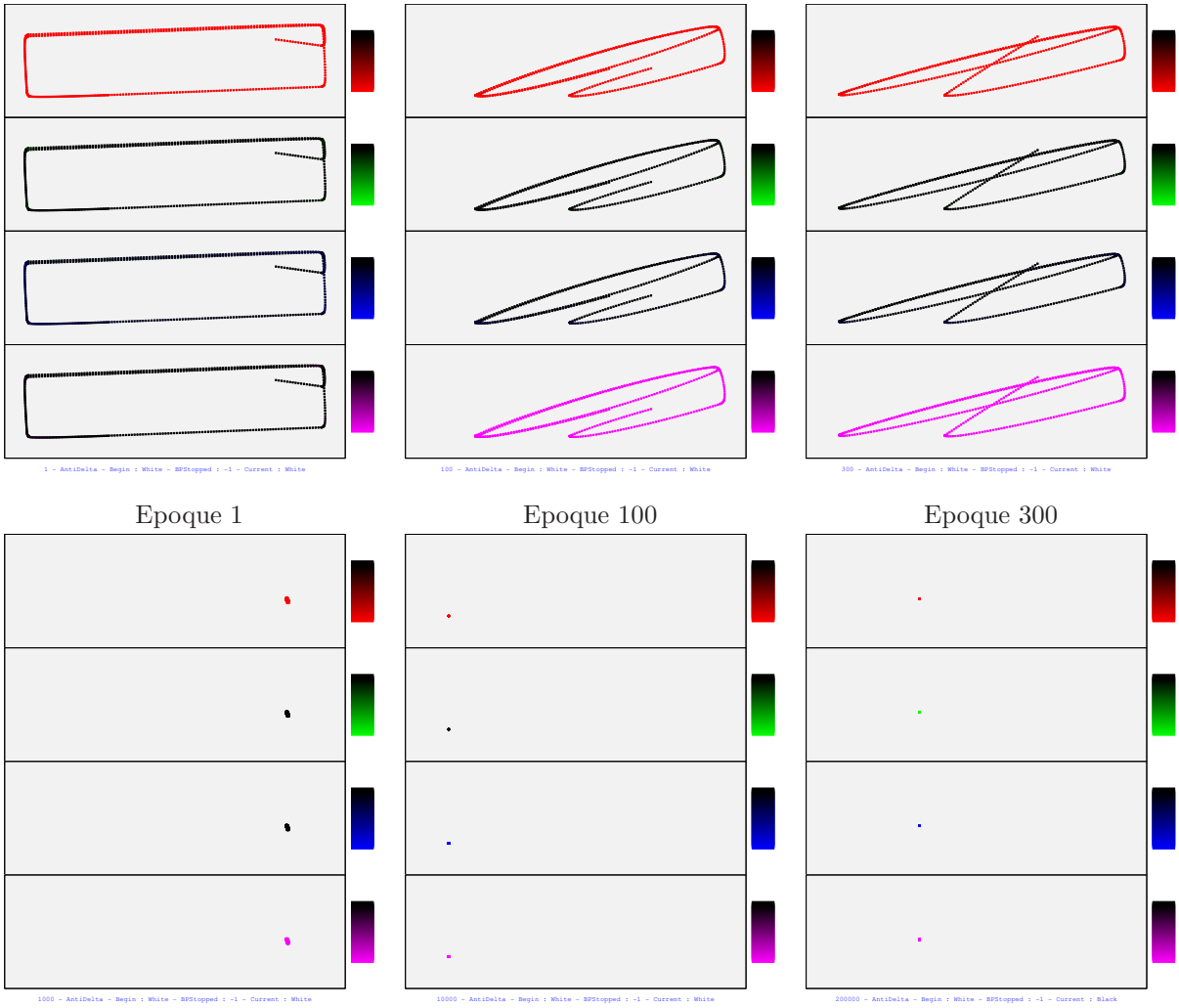


Evolution de la fitness à travers les époques



Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. A.3 – Essai 2

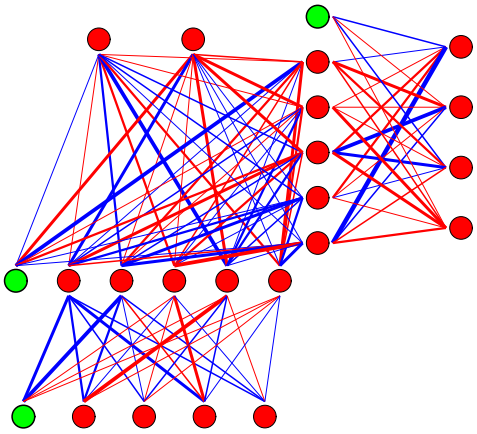


Epoque 1 | Epoque 100 | Epoque 300

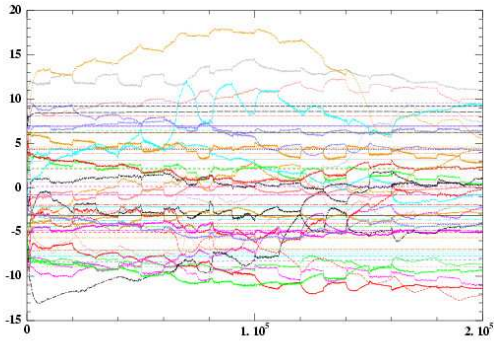
Epoque 1000 | Epoque 10000 | Epoque 200000

FIG. A.4 – Essai 2

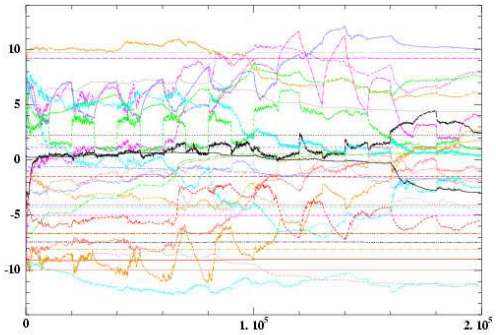
A.3 essai 3



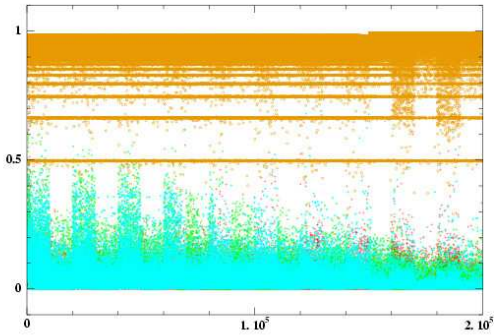
Les poids du réseau à la fin



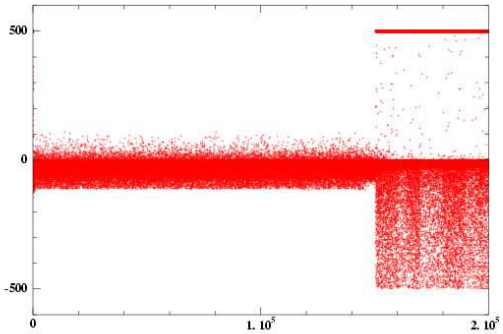
Detail des poids à travers les époques du réseau d'anticipation



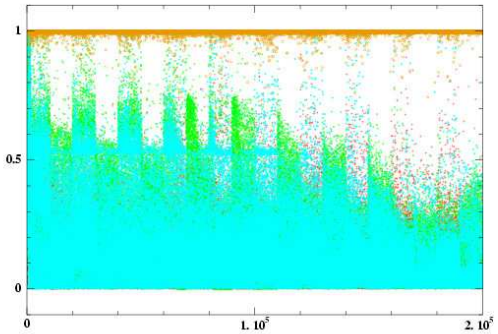
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

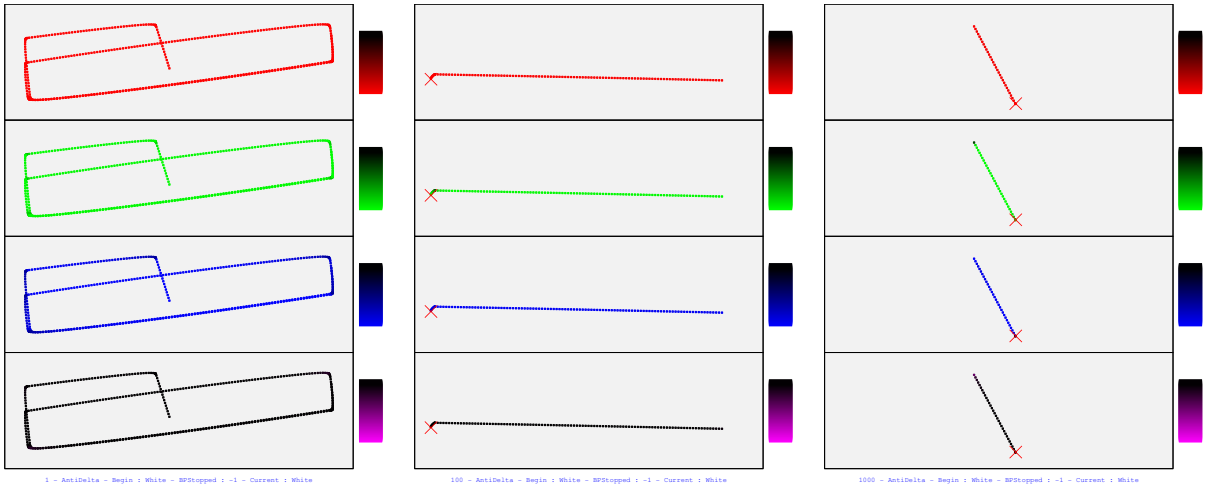


Evolution de la fitness à travers les époques

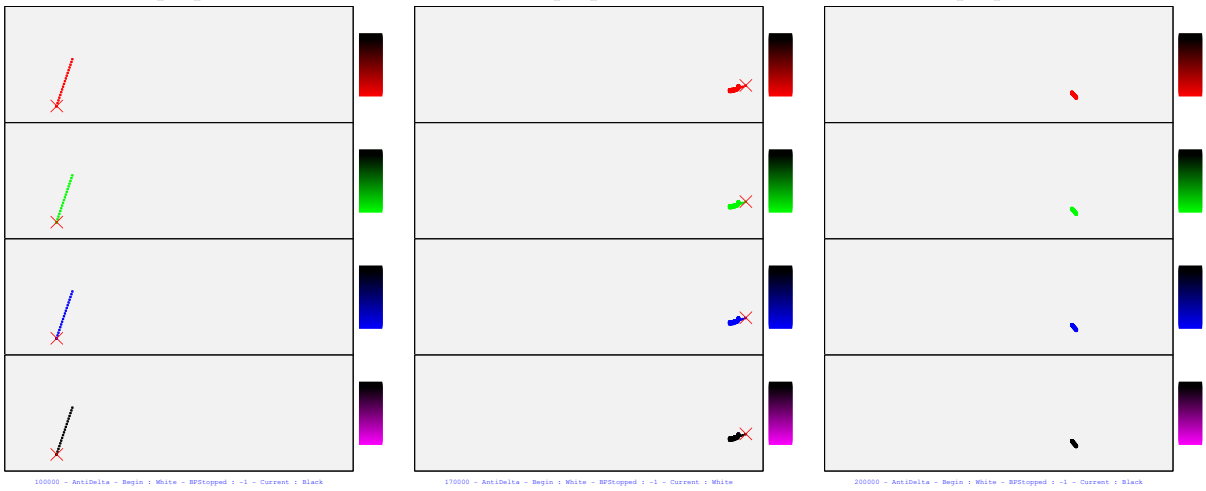


Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. A.5 – Essai 3



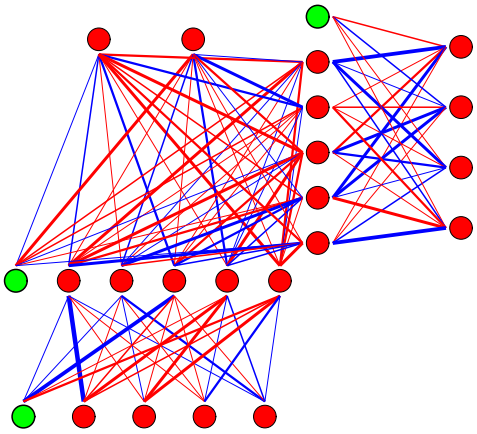
Epoque 1 | Epoque 100 | Epoque 1000



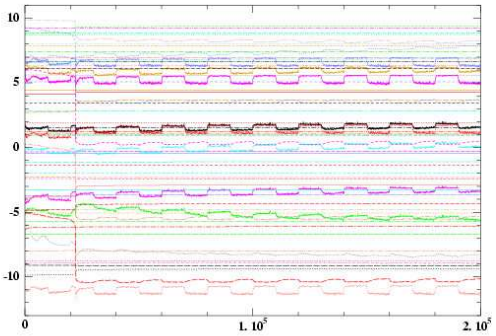
Epoque 100000 | Epoque 170000 | Epoque 200000

FIG. A.6 – Essai 3

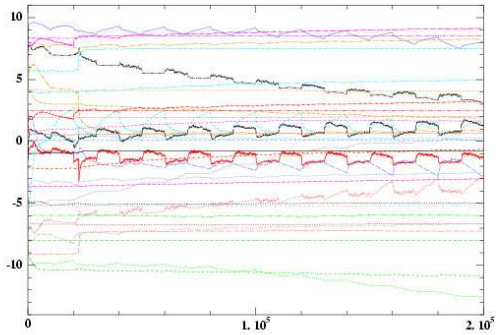
A.4 essai 4



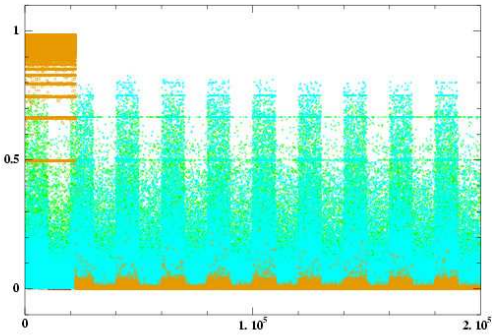
Les poids du réseau à la fin



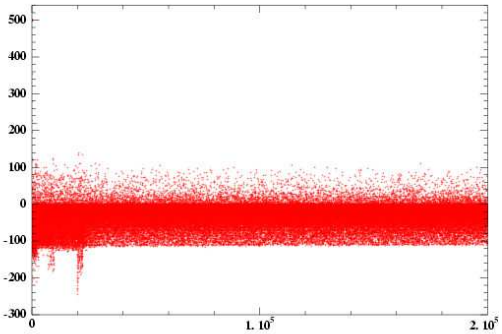
Detail des poids à travers les époques du réseau d'anticipation



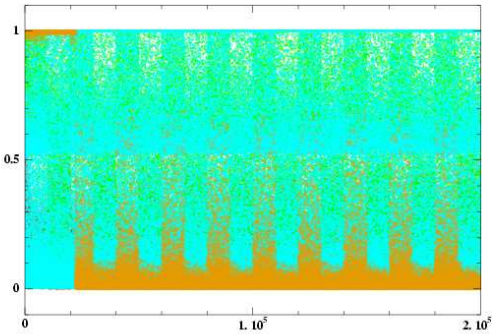
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

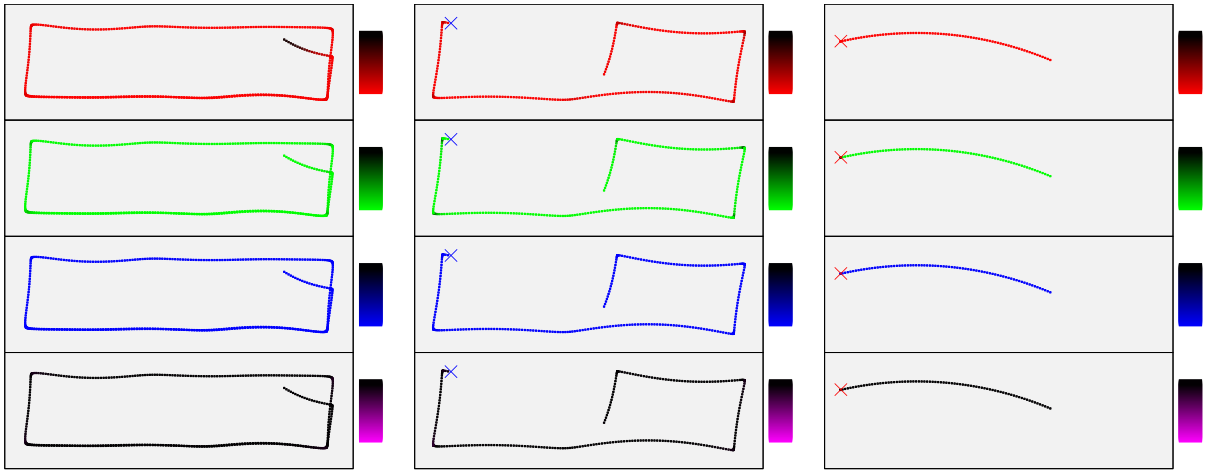


Evolution de la fitness à travers les époques

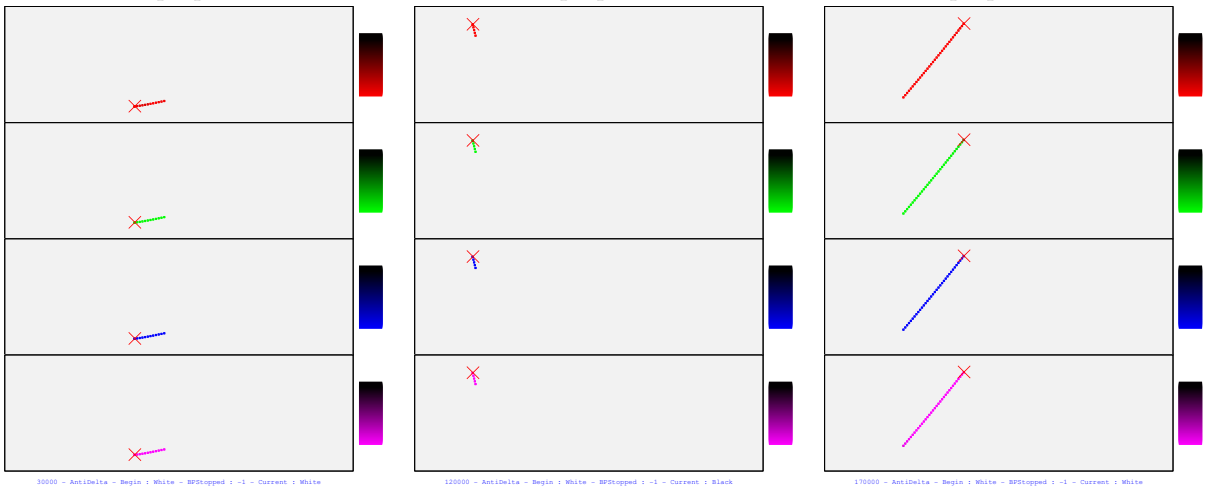


Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. A.7 – Essai 4



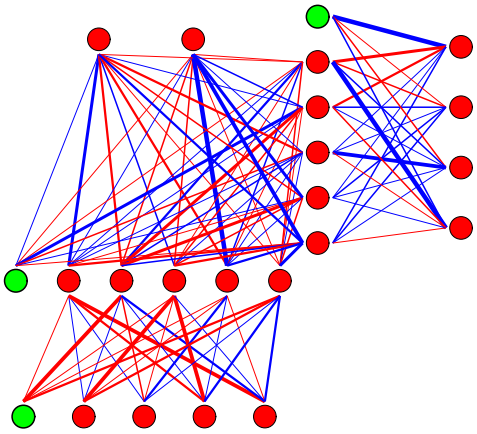
Epoque 1 | Epoque 10 | Epoque 20000



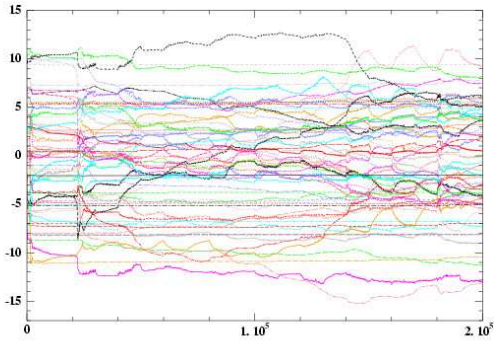
Epoque 20000 | Epoque 120000 | Epoque 170000

FIG. A.8 – Essai 4

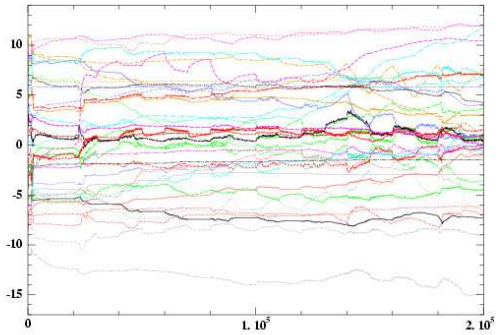
A.5 essai 5



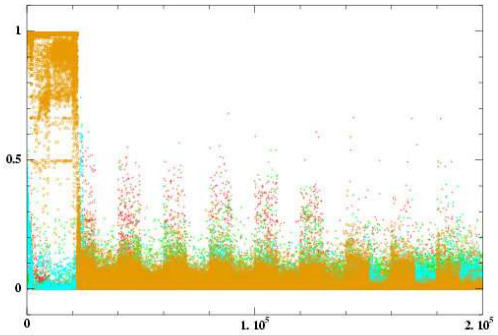
Les poids du réseau à la fin



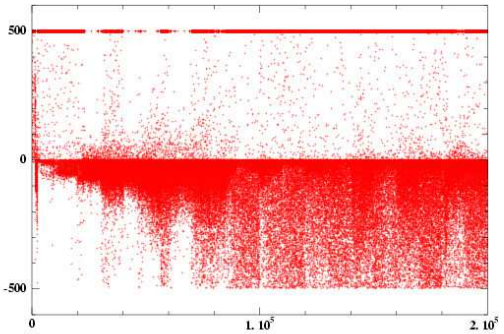
Detail des poids à travers les époques du réseau d'anticipation



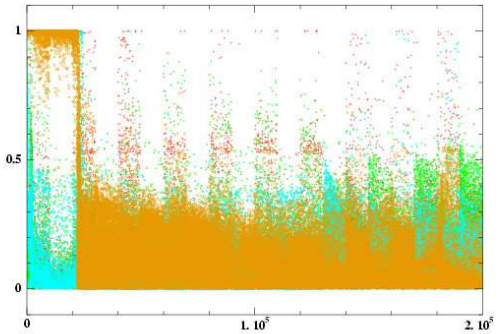
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

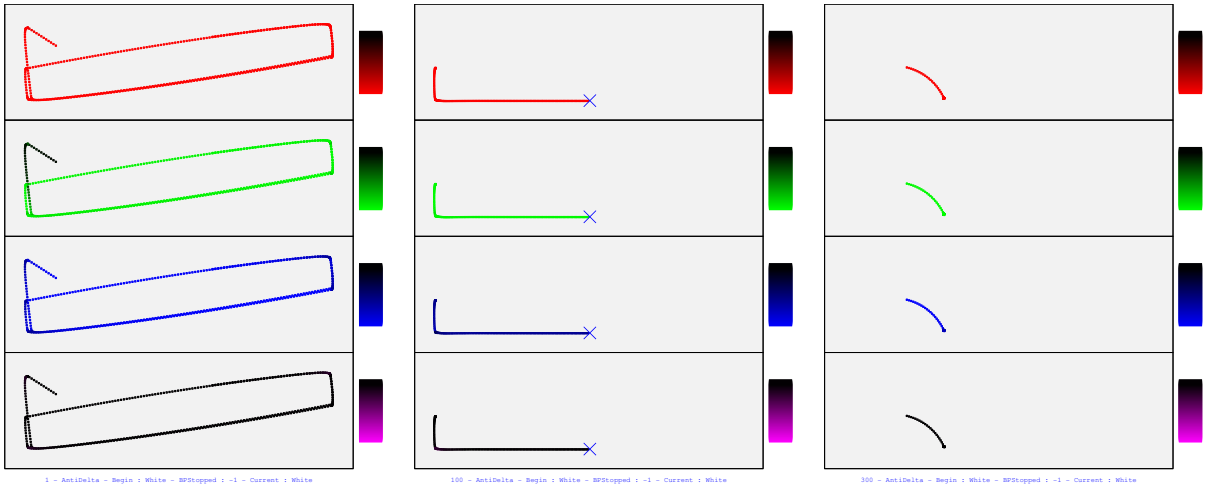


Evolution de la fitness à travers les époques

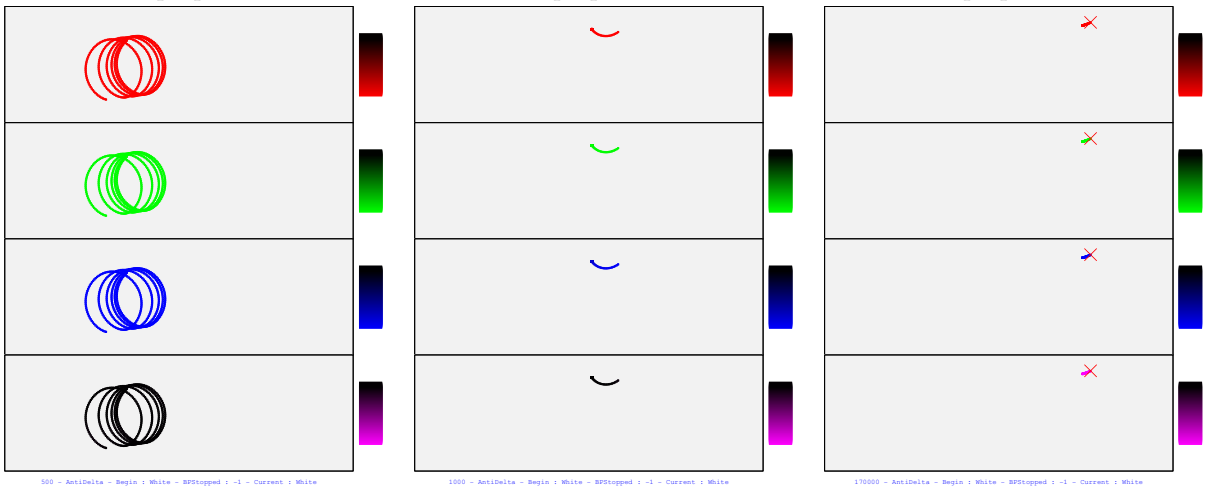


Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. A.9 – Essai 5



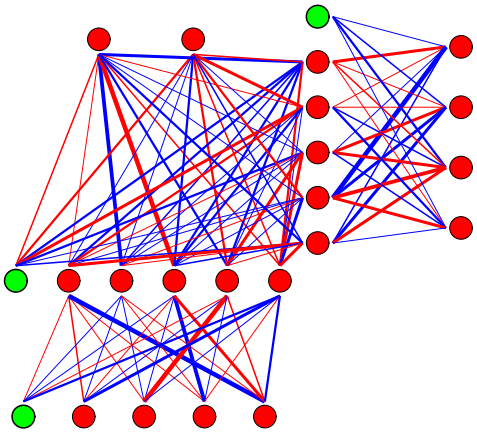
Epoque 1 | Epoque 100 | Epoque 300



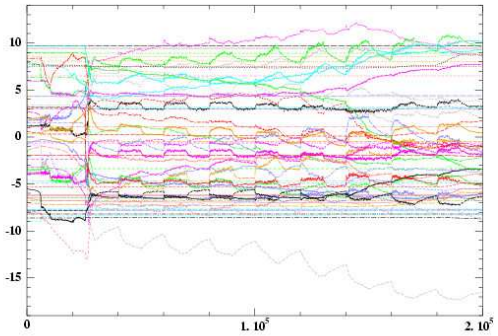
Epoque 500 | Epoque 1000 | Epoque 170000

FIG. A.10 – Essai 5

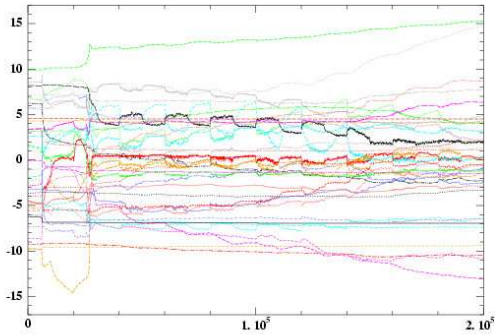
A.6 essai 6



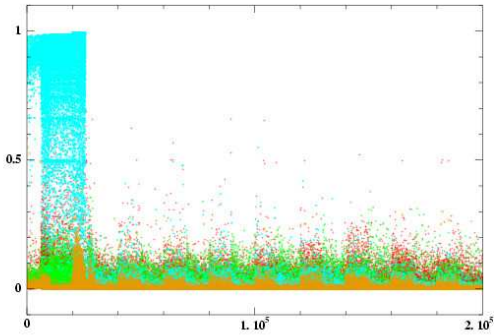
Les poids du réseau à la fin



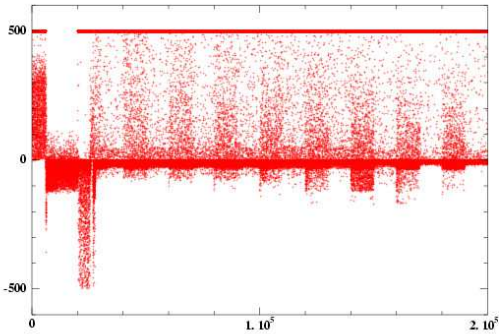
Detail des poids à travers les époques du réseau d'anticipation



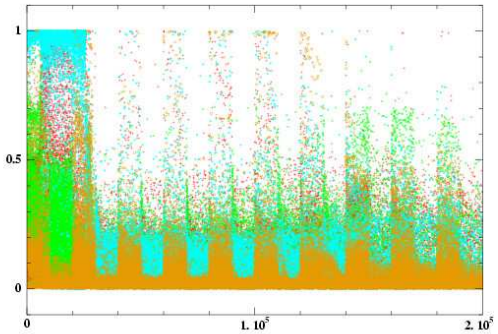
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

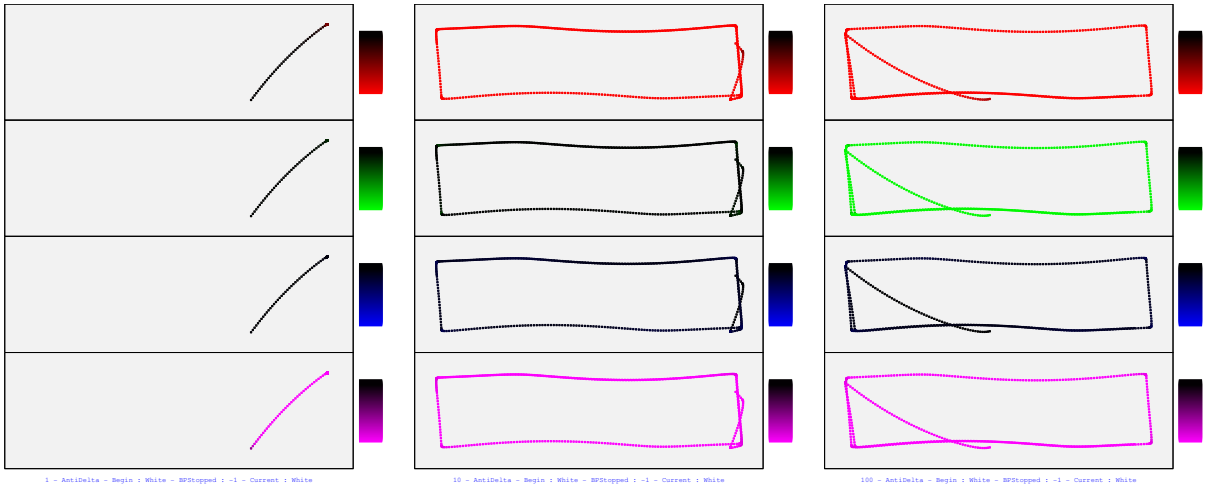


Evolution de la fitness à travers les époques

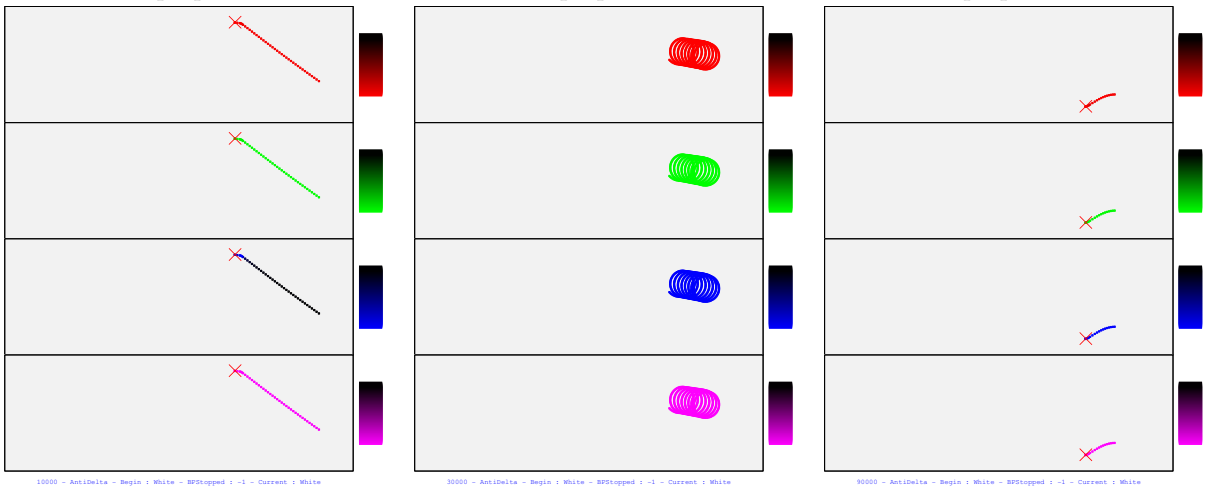


Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. A.11 – Essai 6



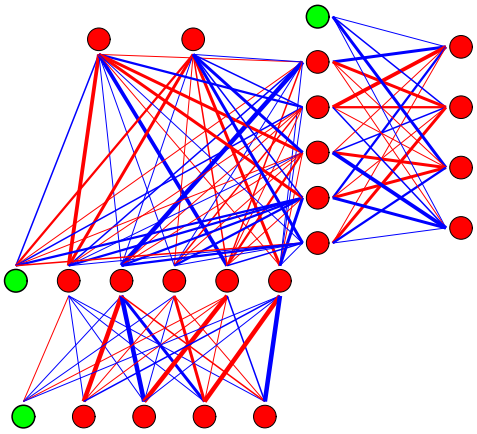
Epoque 1 | Epoque 10 | Epoque 100



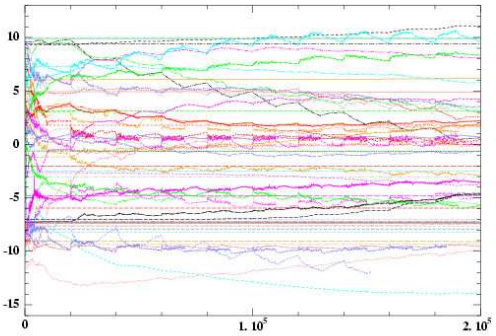
Epoque 10000 | Epoque 30000 | Epoque 90000

FIG. A.12 – Essai 6

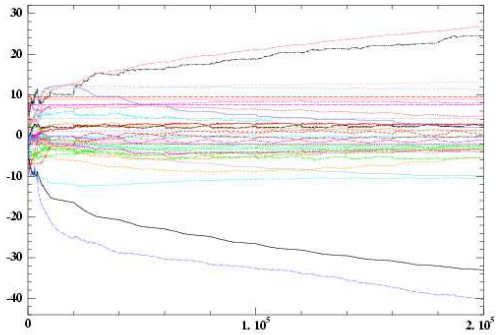
A.7 essai 7



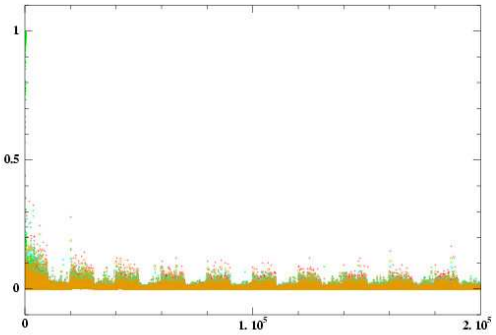
Les poids du réseau à la fin



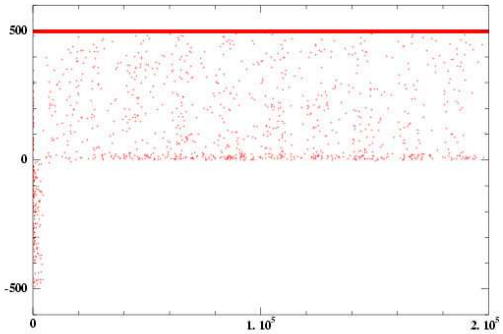
Detail des poids à travers les époques du réseau d'anticipation



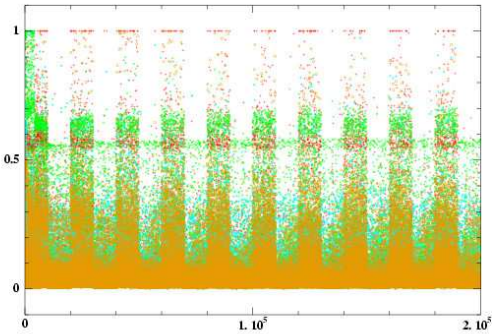
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

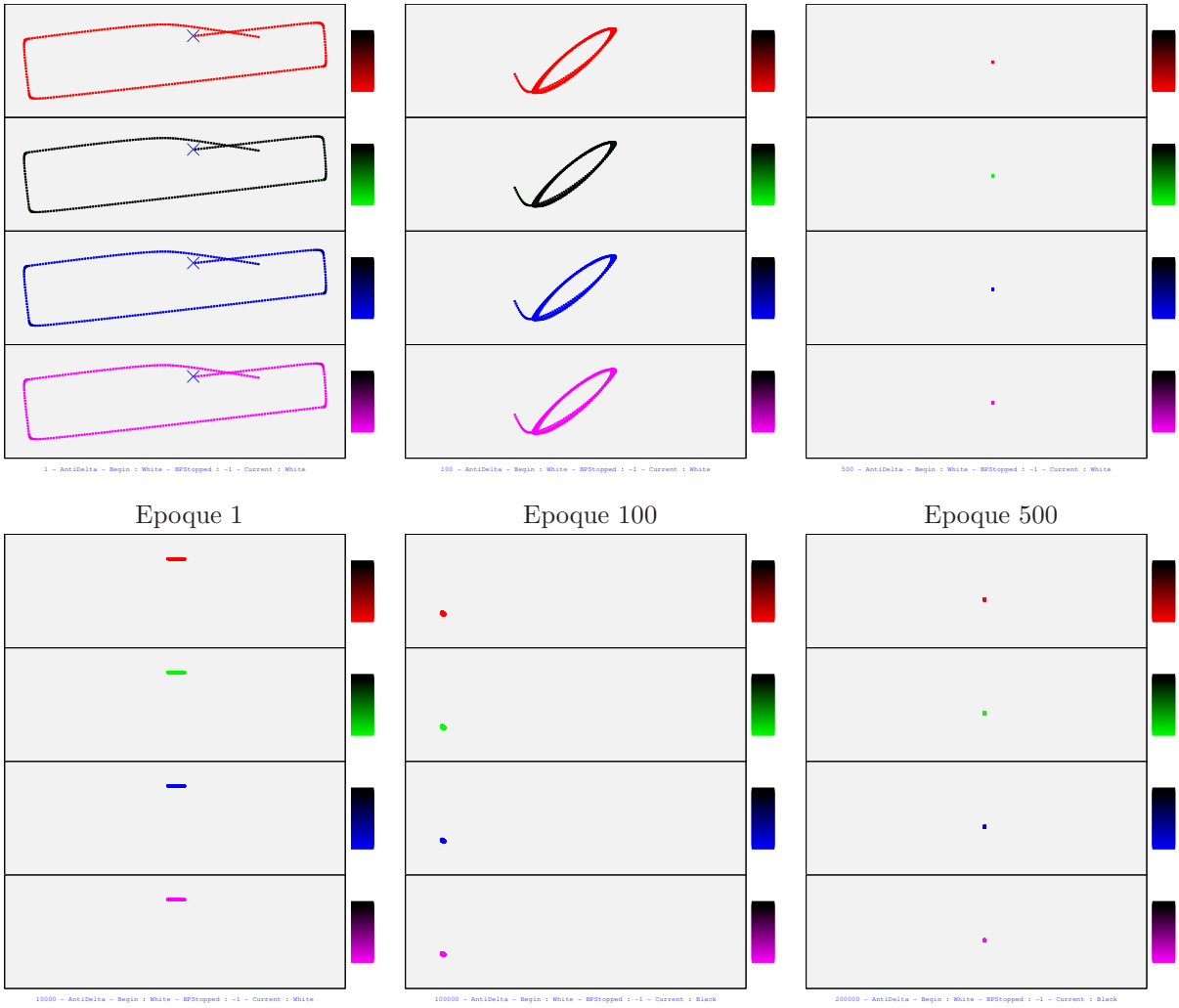


Evolution de la fitness à travers les époques



Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. A.13 – Essai 7

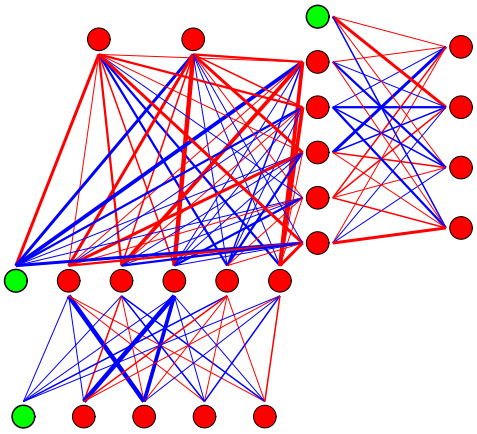


Epoque 1 | Epoque 100 | Epoque 500

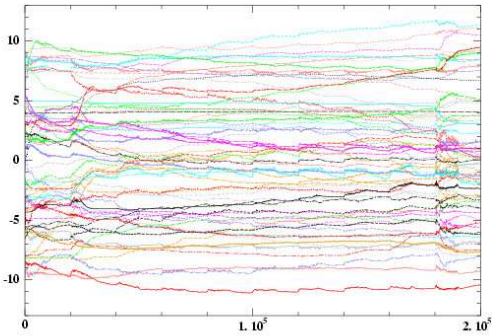
Epoque 10000 | Epoque 100000 | Epoque 200000

FIG. A.14 – Essai 7

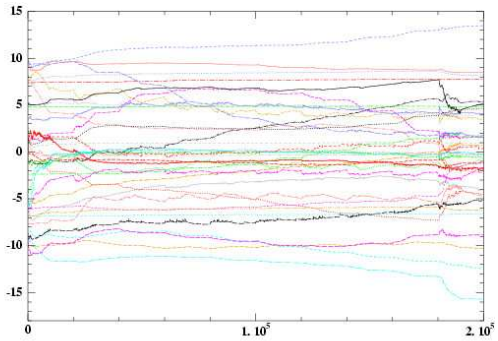
A.8 essai 8



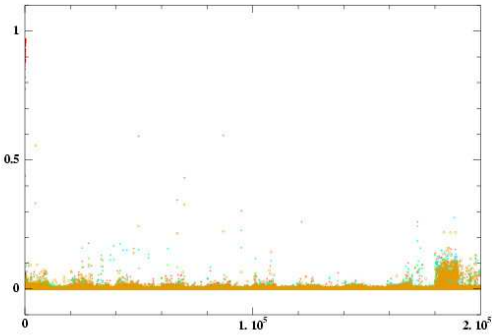
Les poids du réseau à la fin



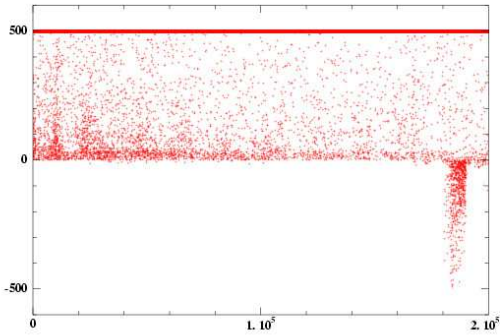
Detail des poids à travers les époques du réseau d'anticipation



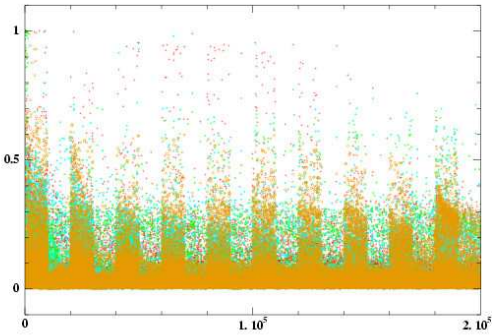
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

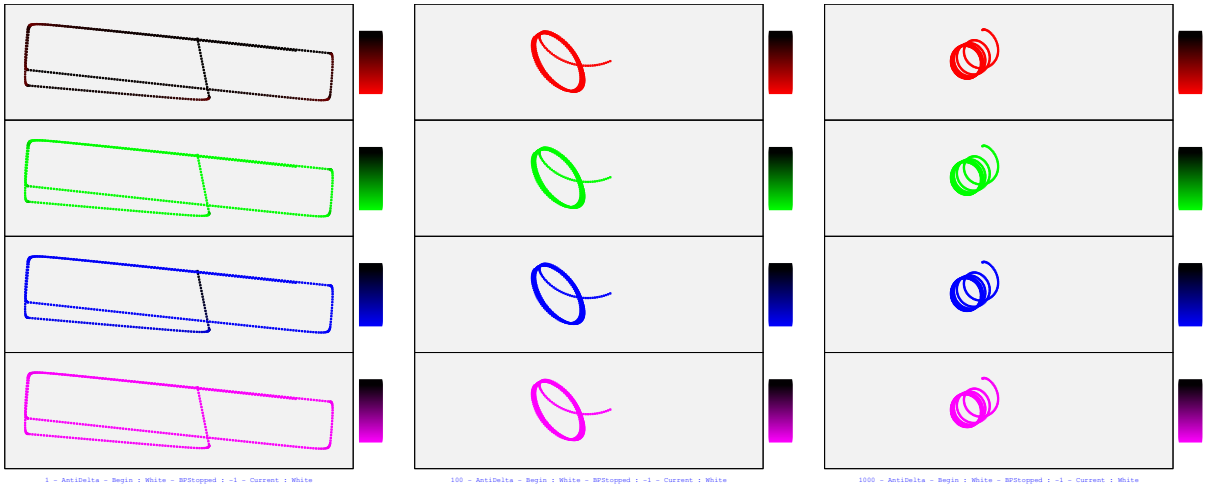


Evolution de la fitness à travers les époques

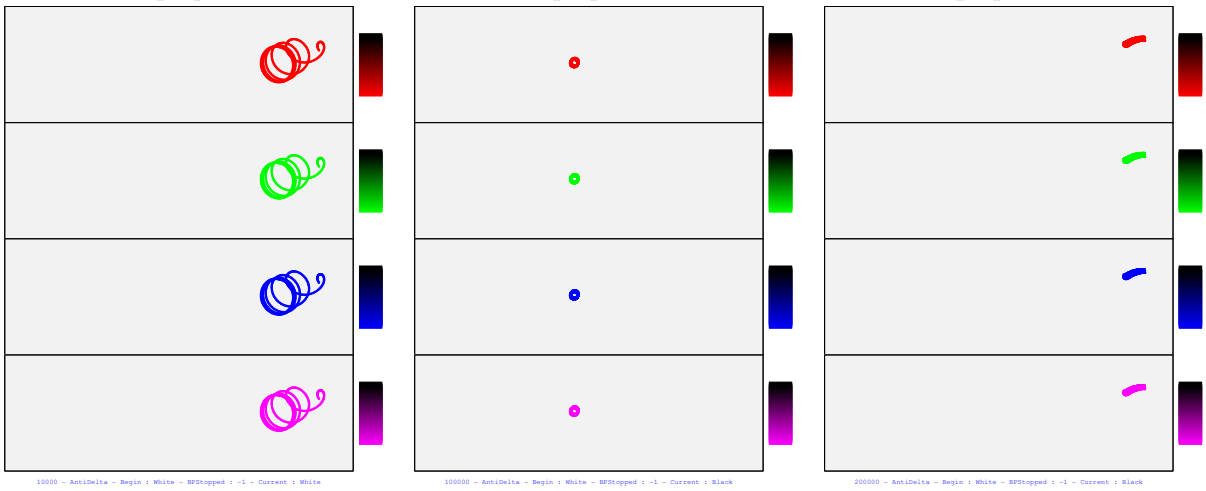


Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. A.15 – Essai 8



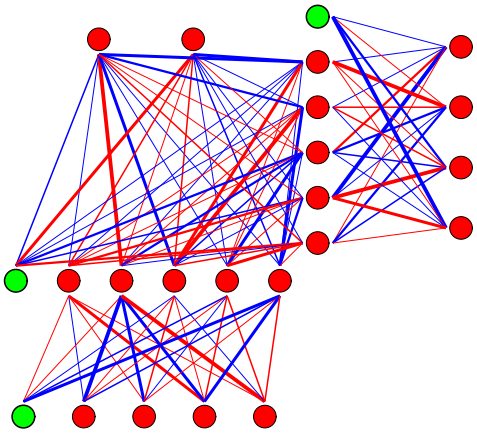
Epoque 1 | Epoque 100 | Epoque 1000



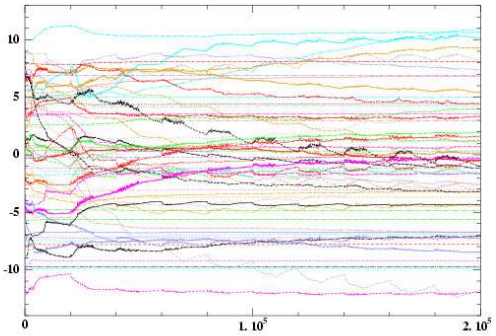
Epoque 10000 | Epoque 100000 | Epoque 200000

FIG. A.16 – Essai 8

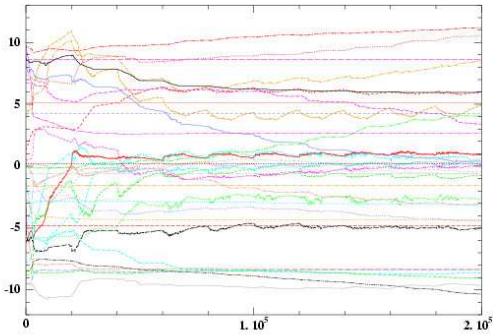
A.9 essai 9



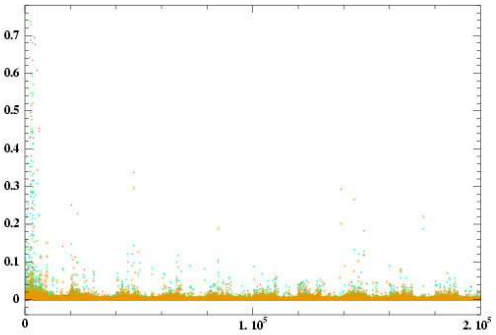
Les poids du réseau à la fin



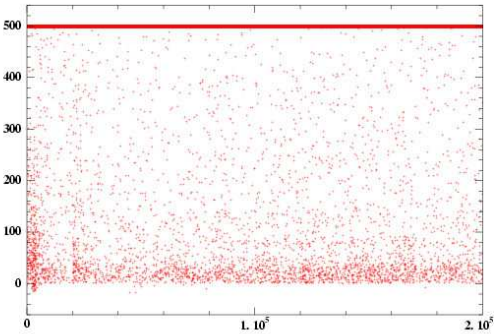
Detail des poids à travers les époques du réseau d'anticipation



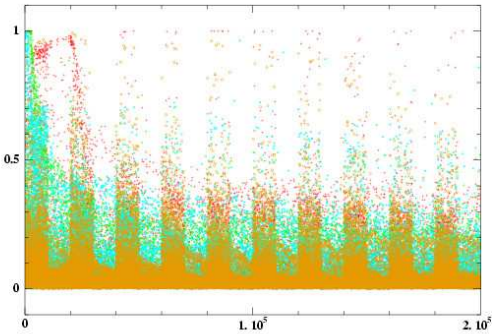
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

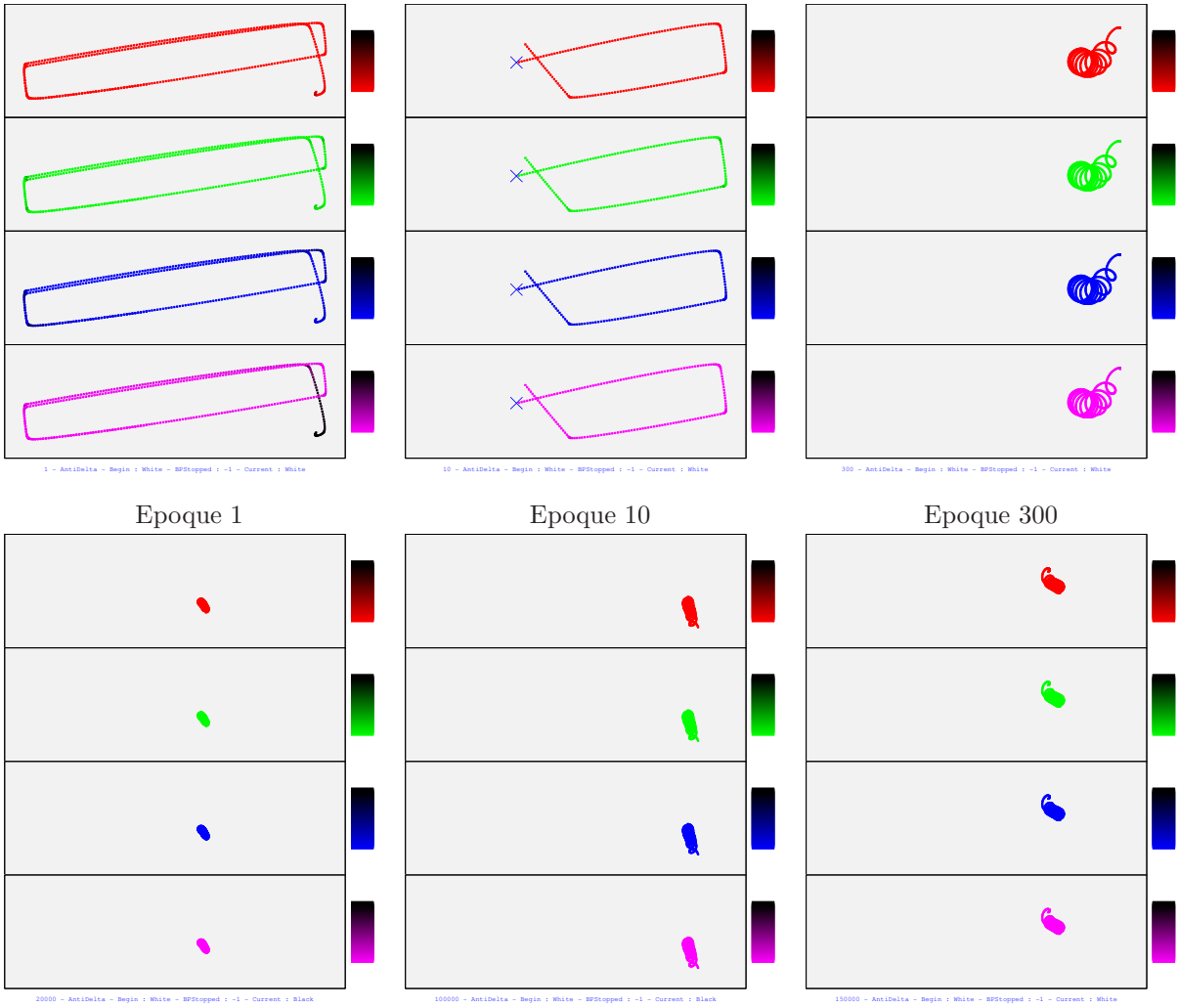


Evolution de la fitness à travers les époques



Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. A.17 – Essai 9

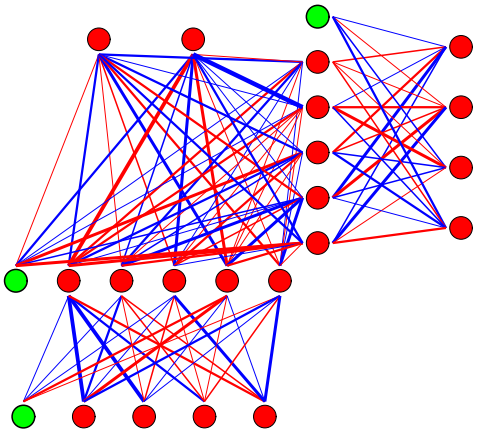


Epoque 1 | Epoque 10 | Epoque 300

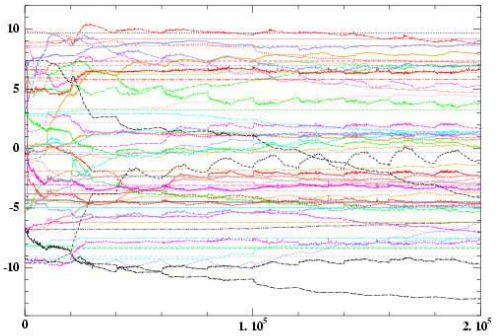
Epoque 20000 | Epoque 100000 | Epoque 150000

FIG. A.18 – Essai 9

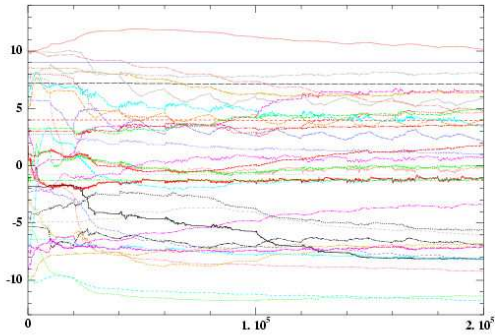
A.10 essai 10



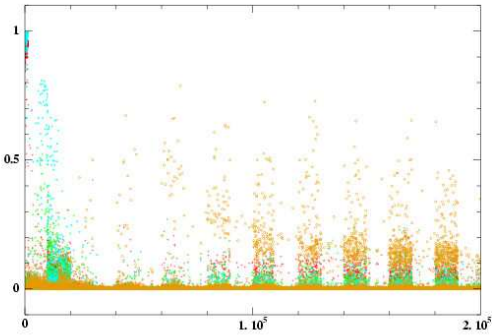
Les poids du réseau à la fin



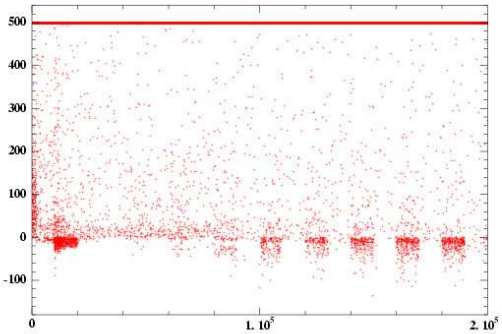
Detail des poids à travers les époques du réseau d'anticipation



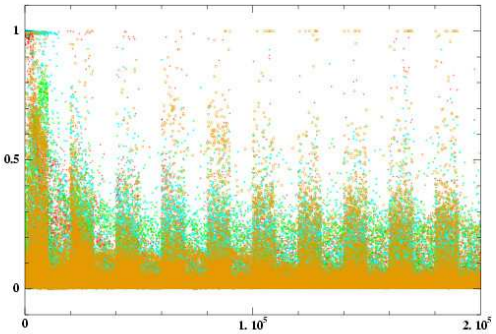
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

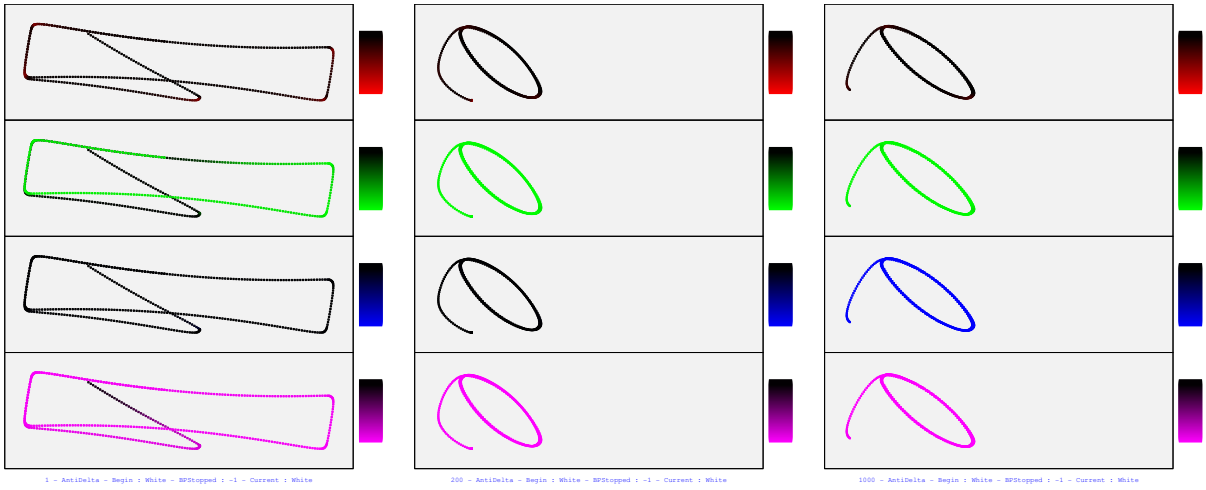


Evolution de la fitness à travers les époques

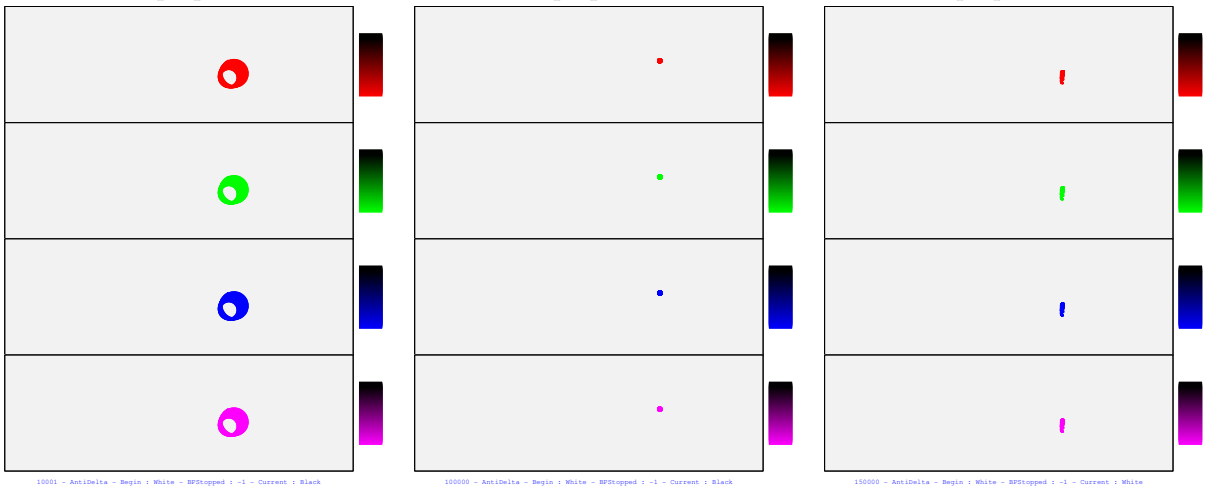


Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. A.19 – Essai 10



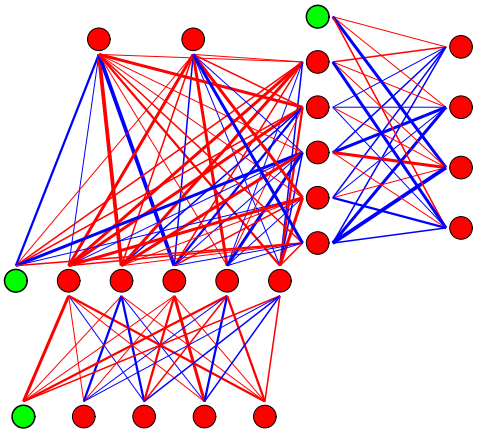
Epoque 1 | Epoque 200 | Epoque 1000



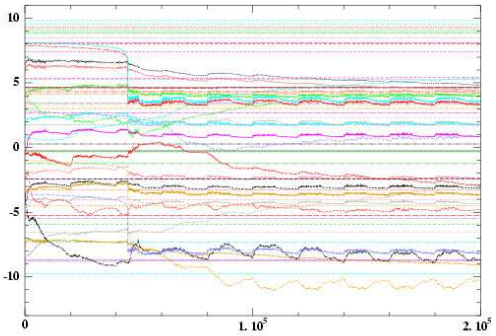
Epoque 10001 | Epoque 100000 | Epoque 150000

FIG. A.20 – Essai 10

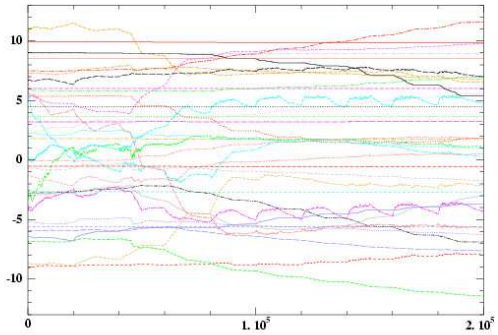
A.11 essai 11



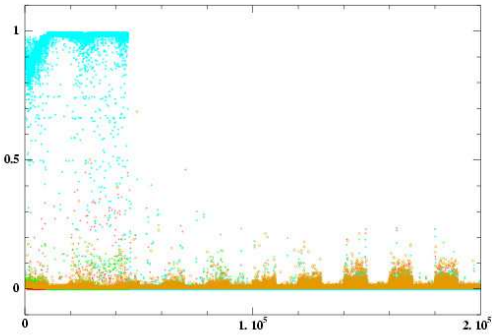
Les poids du réseau à la fin



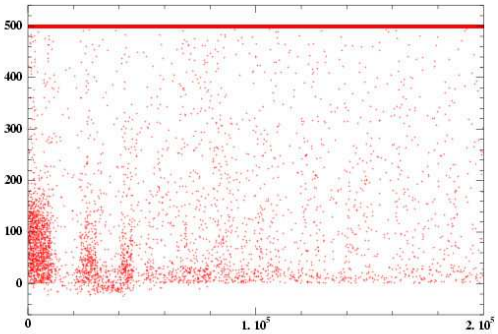
Detail des poids à travers les époques du réseau d'anticipation



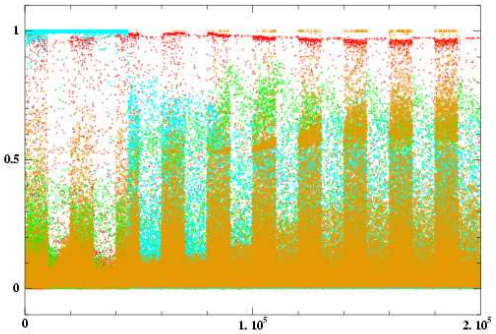
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

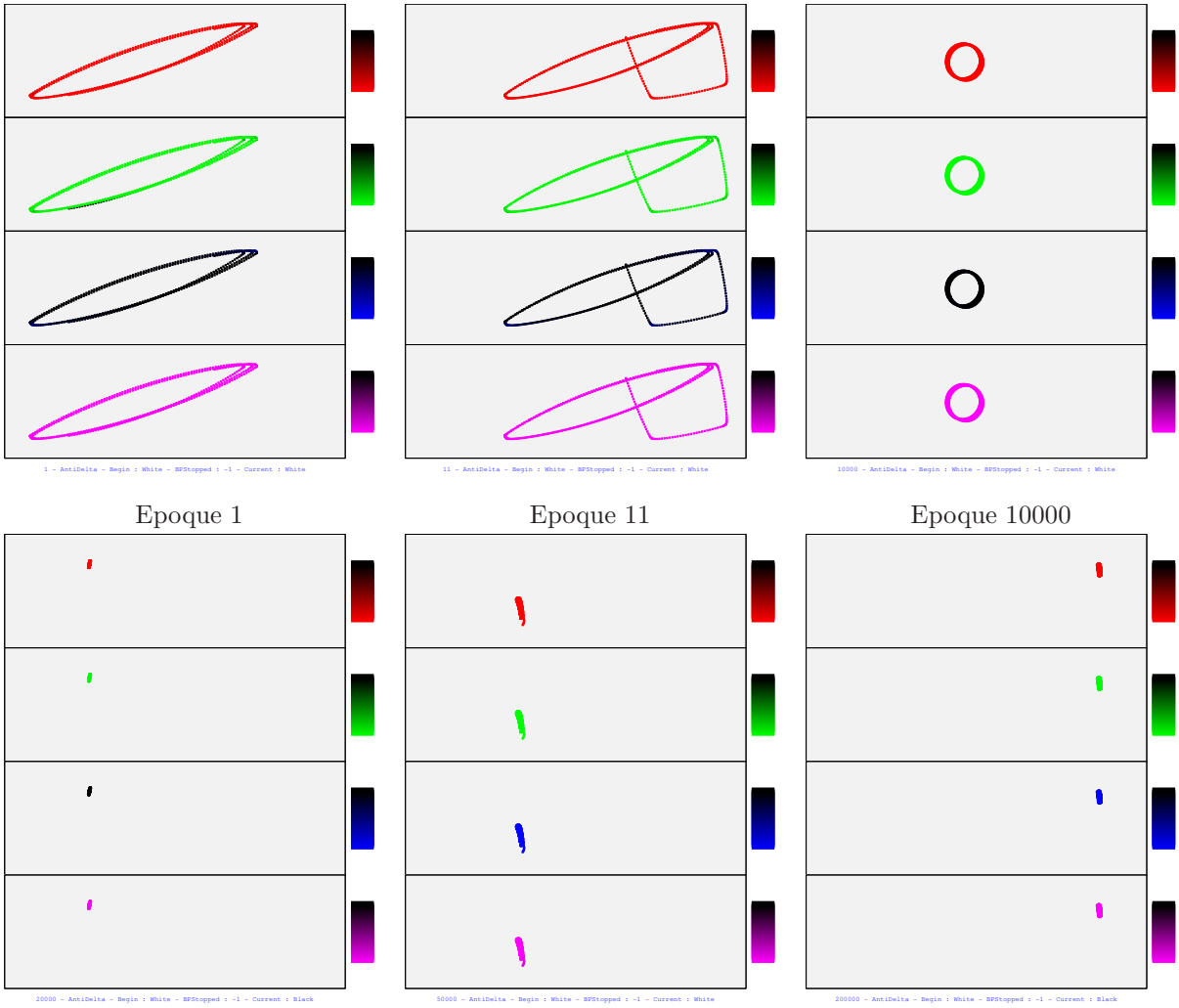


Evolution de la fitness à travers les époques



Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. A.21 – Essai 11

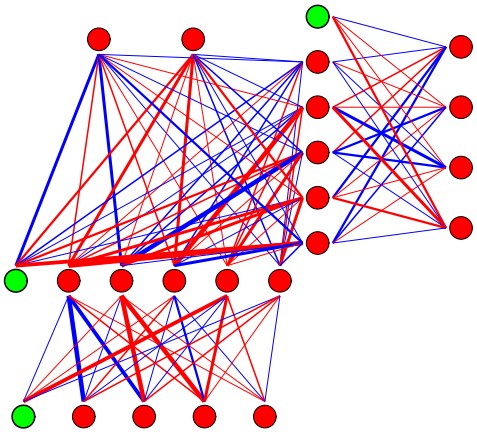


Epoque 1 | Epoque 11 | Epoque 10000

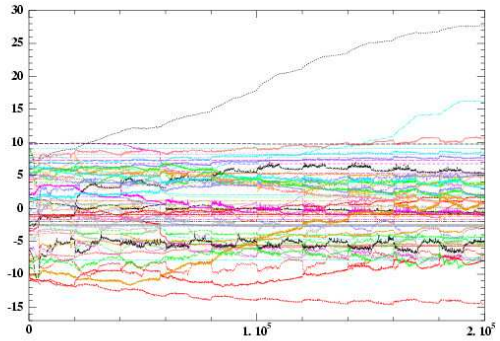
Epoque 20000 | Epoque 100000 | Epoque 200000

FIG. A.22 – Essai 11

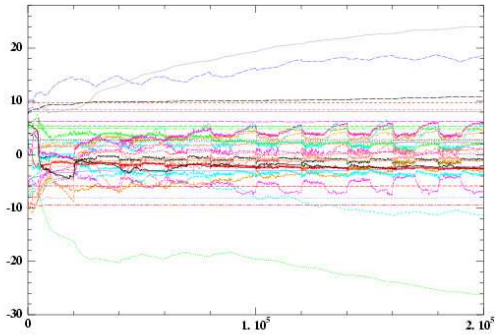
A.12 essai 12



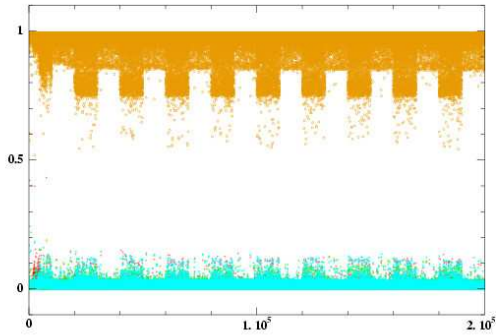
Les poids du réseau à la fin



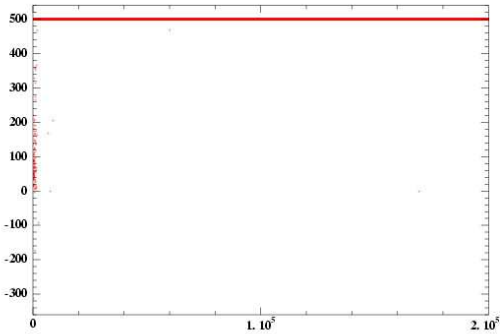
Detail des poids à travers les époques du réseau d'anticipation



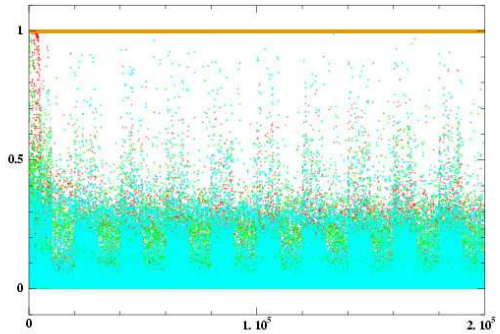
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

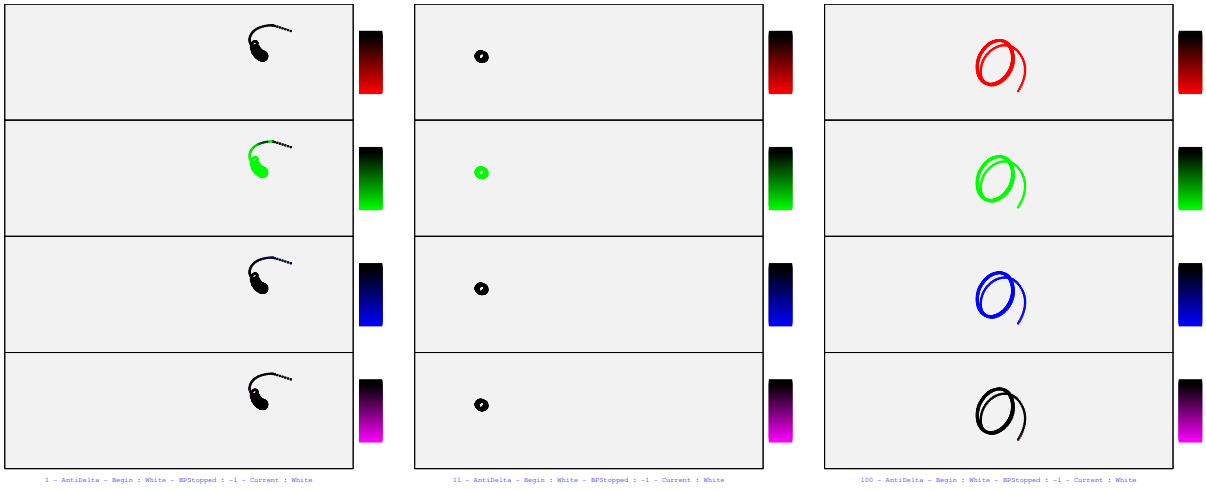


Evolution de la fitness à travers les époques

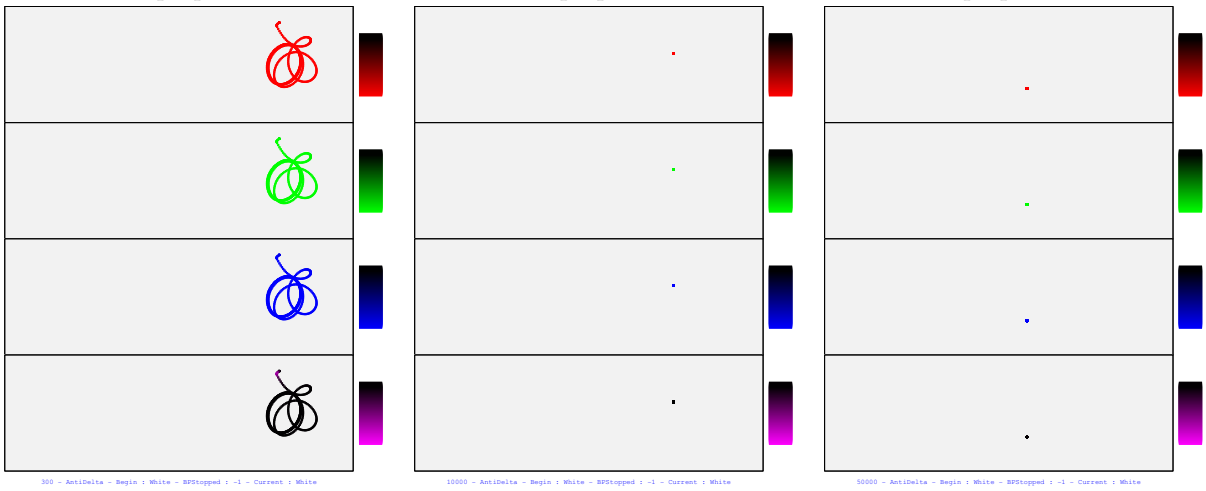


Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. A.23 – Essai 12



Epoque 1 | Epoque 11 | Epoque 100



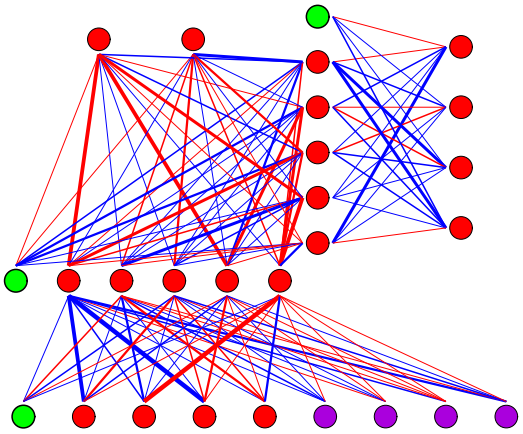
Epoque 300 | Epoque 10000 | Epoque 50000

FIG. A.24 – Essai 12

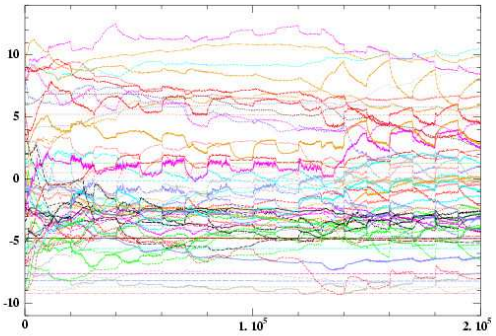
Annexe B

Les résultats détaillés avec l'architecture AAA avec les DELTA en plus

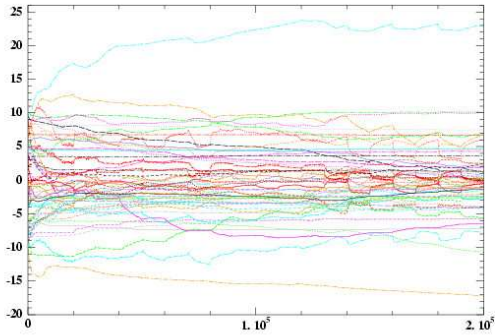
B.1 essai 1



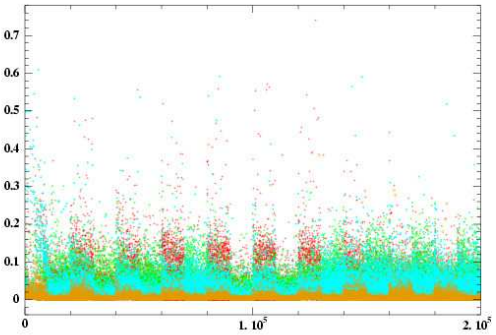
Les poids du réseau à la fin



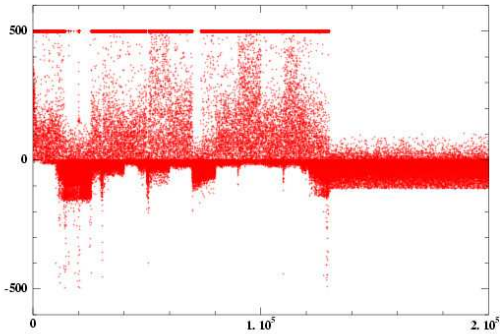
Detail des poids à travers les époques du réseau d'anticipation



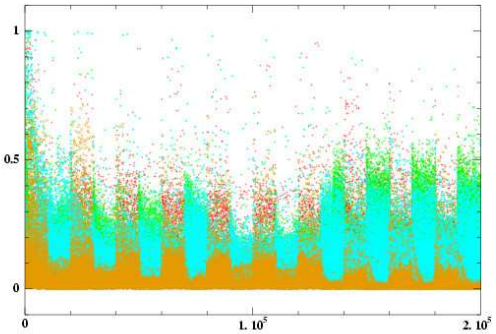
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

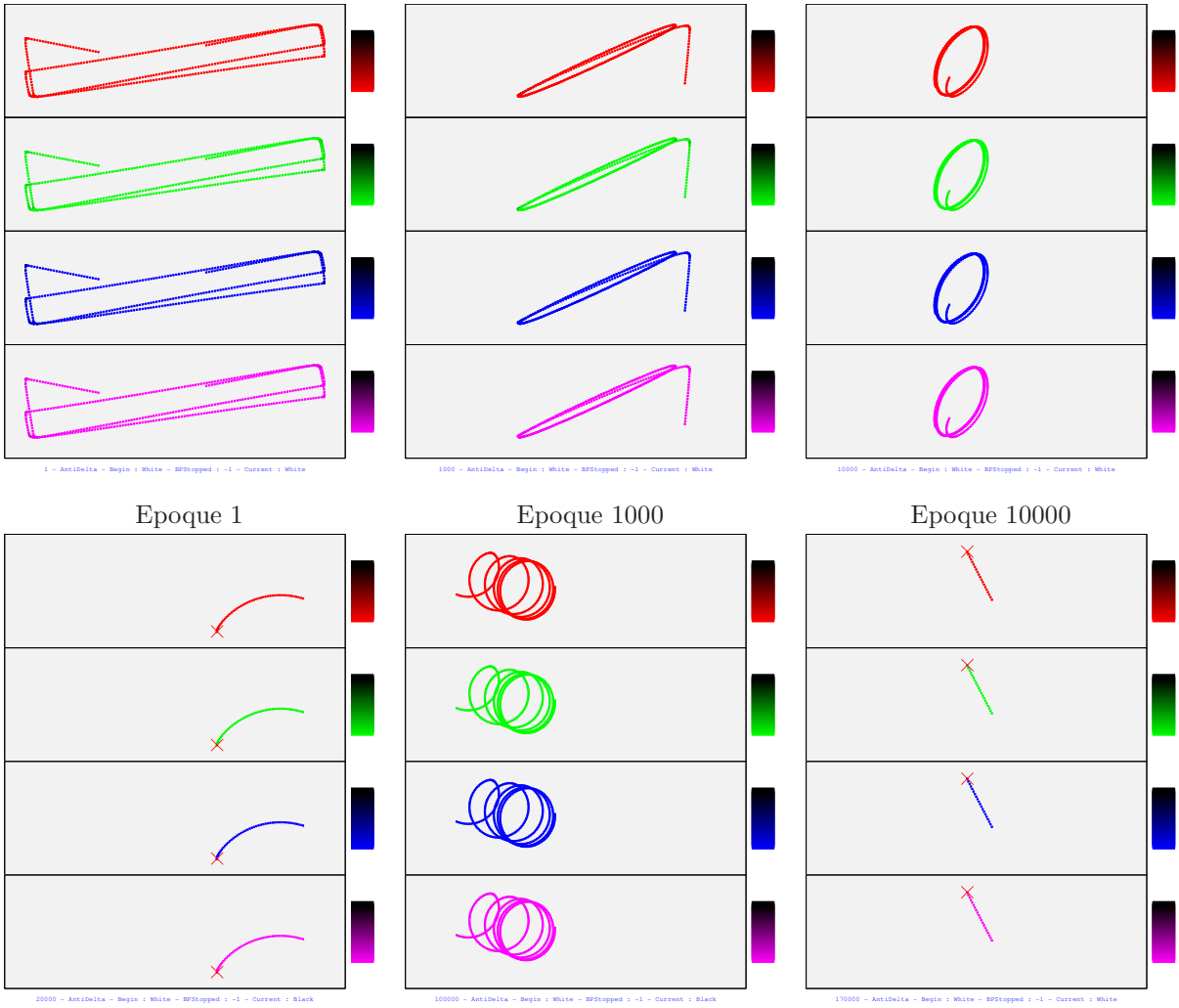


Evolution de la fitness à travers les époques



Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. B.1 – Essai 1

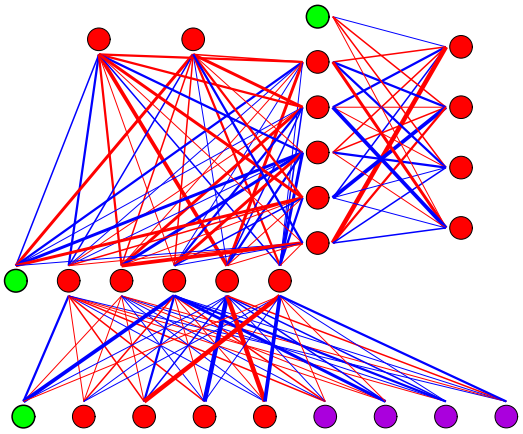


Epoque 1 | Epoque 1000 | Epoque 10000

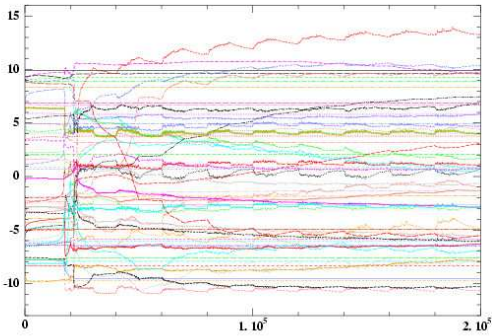
Epoque 20000 | Epoque 100000 | Epoque 170000

FIG. B.2 – Essai 1

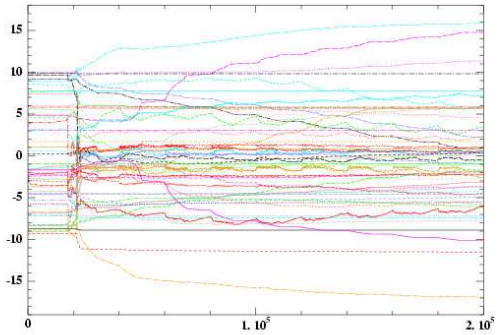
B.2 essai 2



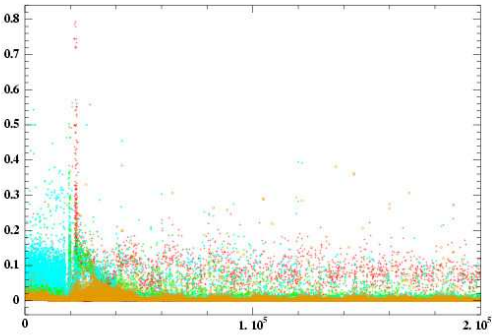
Les poids du réseau à la fin



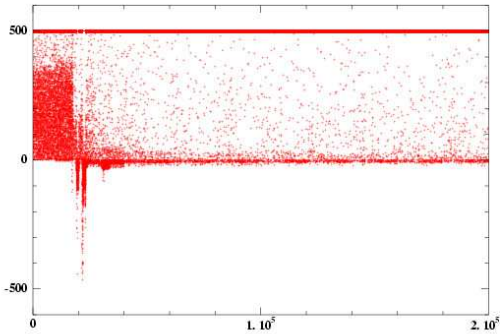
Detail des poids à travers les époques du réseau d'anticipation



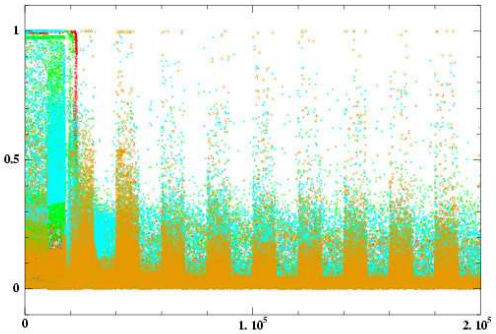
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

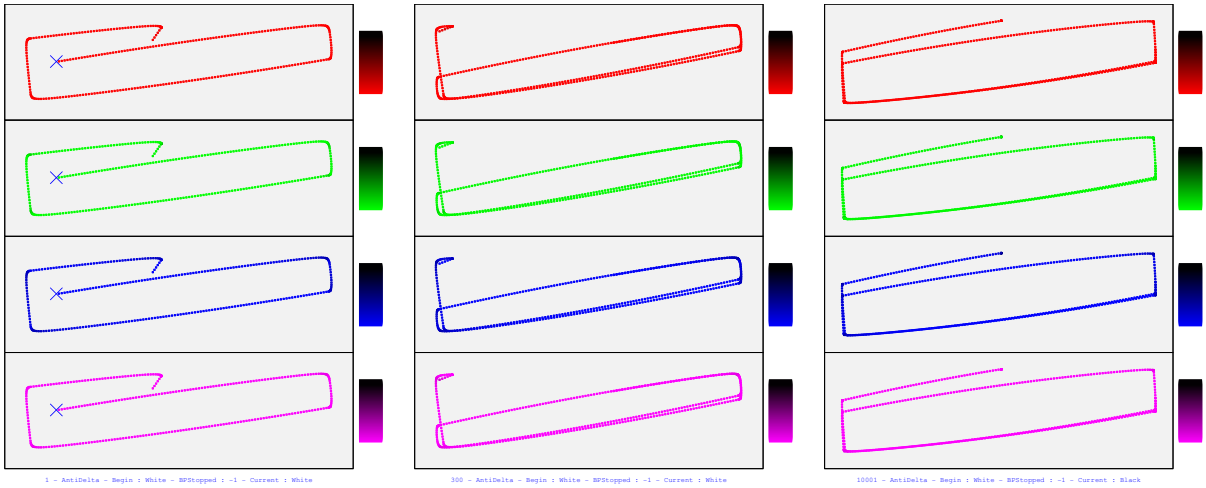


Evolution de la fitness à travers les époques

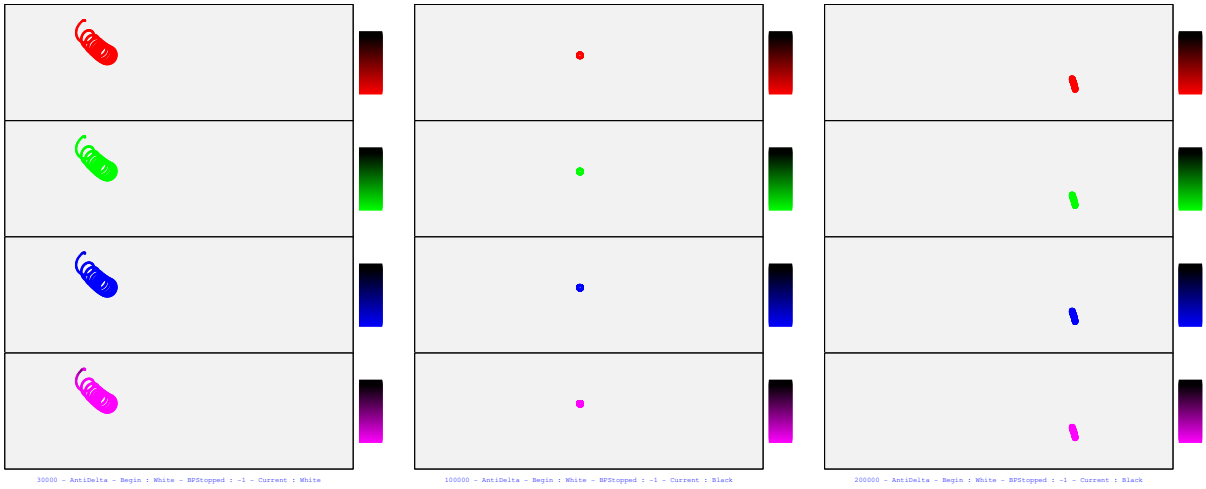


Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. B.3 – Essai 2



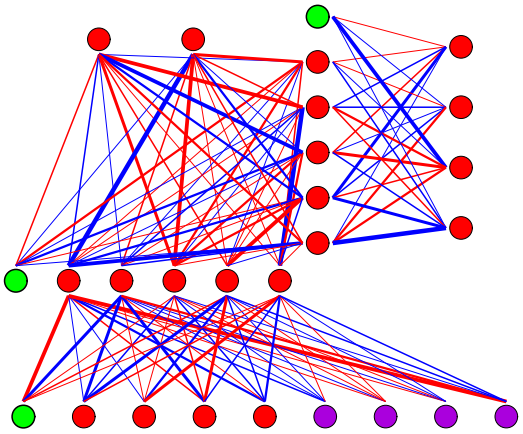
Epoque 1 | Epoque 300 | Epoque 10001



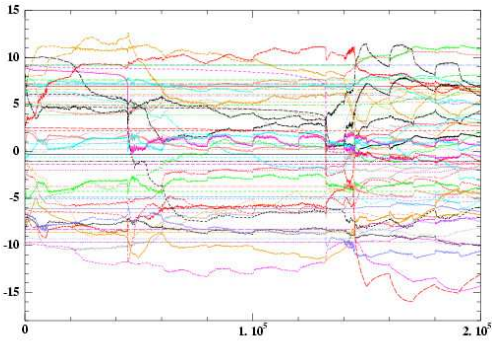
Epoque 30000 | Epoque 100000 | Epoque 200000

FIG. B.4 – Essai 2

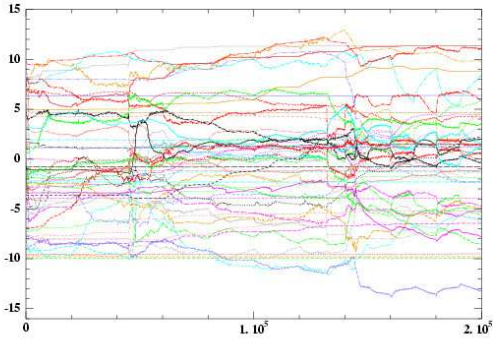
B.3 essai 3



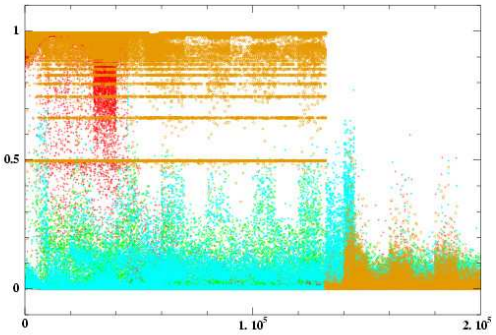
Les poids du réseau à la fin



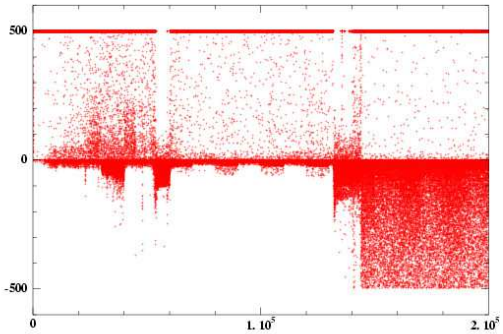
Detail des poids à travers les époques du réseau d'anticipation



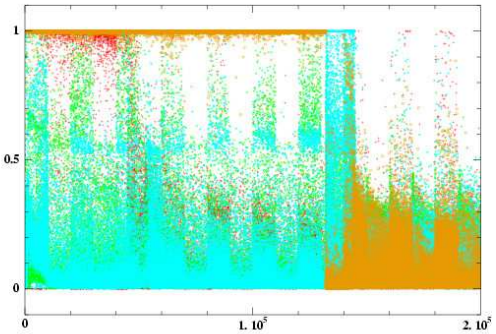
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

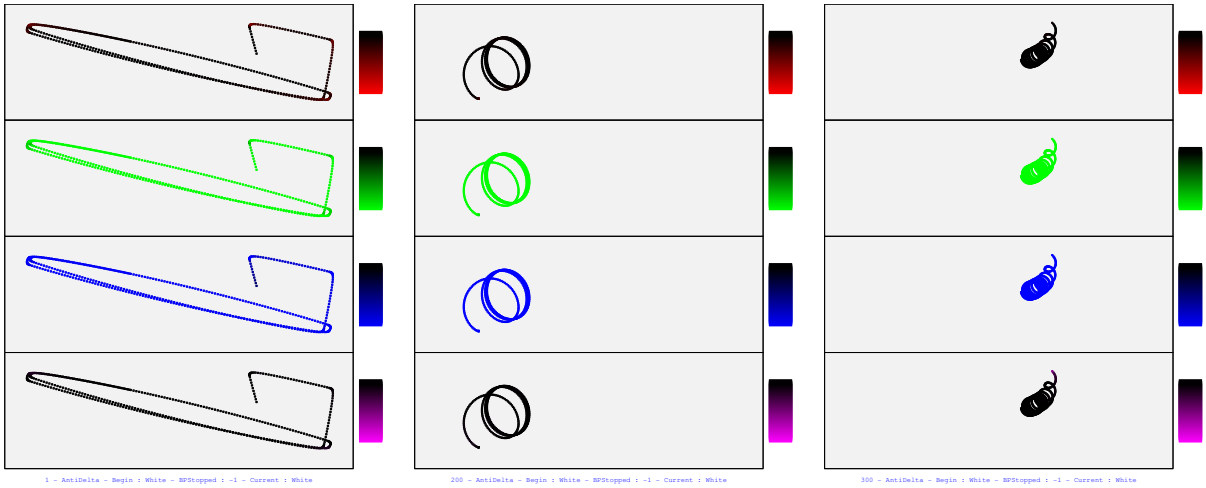


Evolution de la fitness à travers les époques

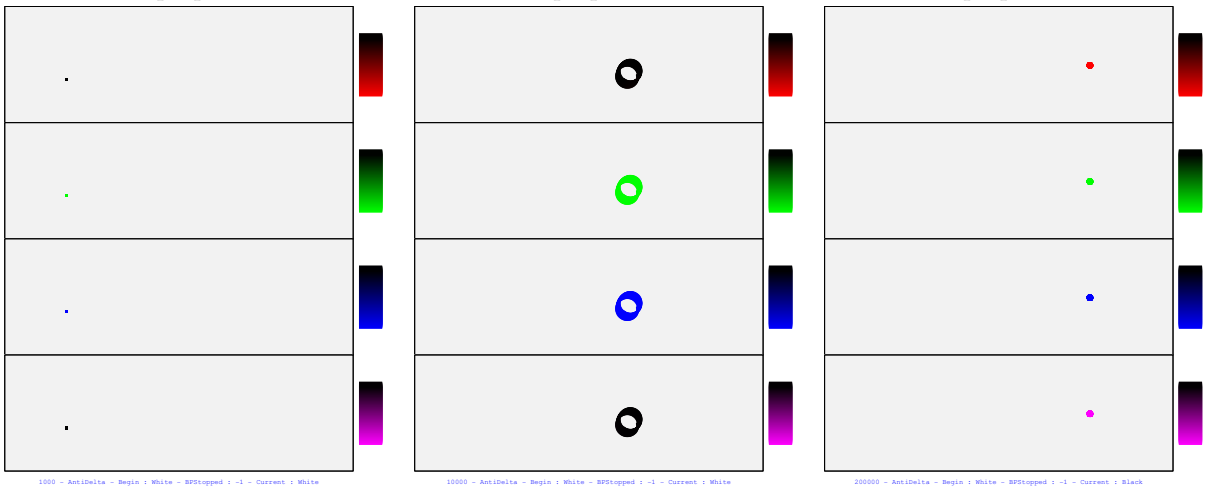


Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. B.5 – Essai 3



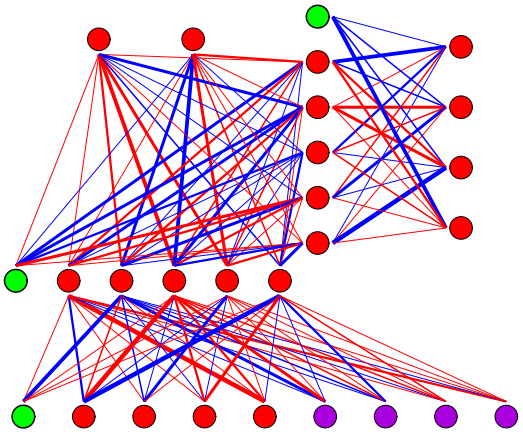
Epoque 1 | Epoque 200 | Epoque 300



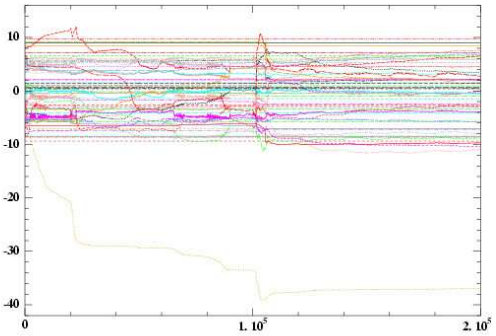
Epoque 1000 | Epoque 10000 | Epoque 200000

FIG. B.6 – Essai 3

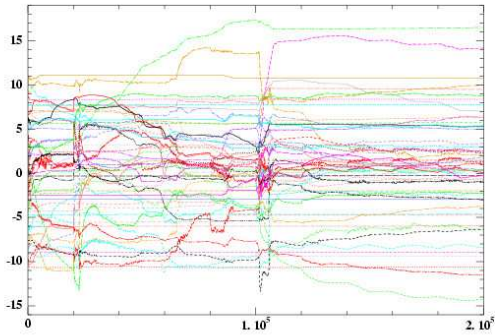
B.4 essai 4



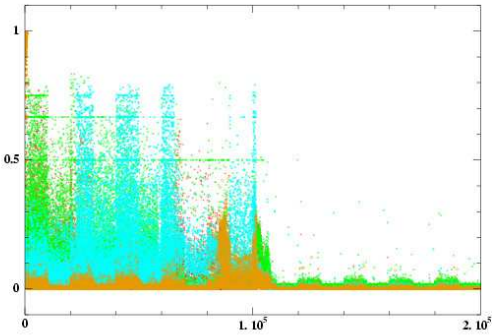
Les poids du réseau à la fin



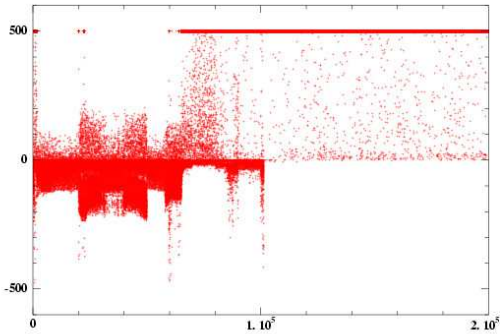
Detail des poids à travers les époques du réseau d'anticipation



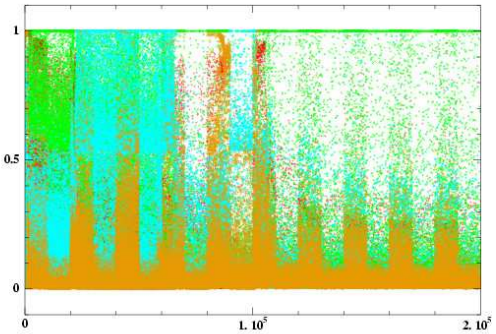
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

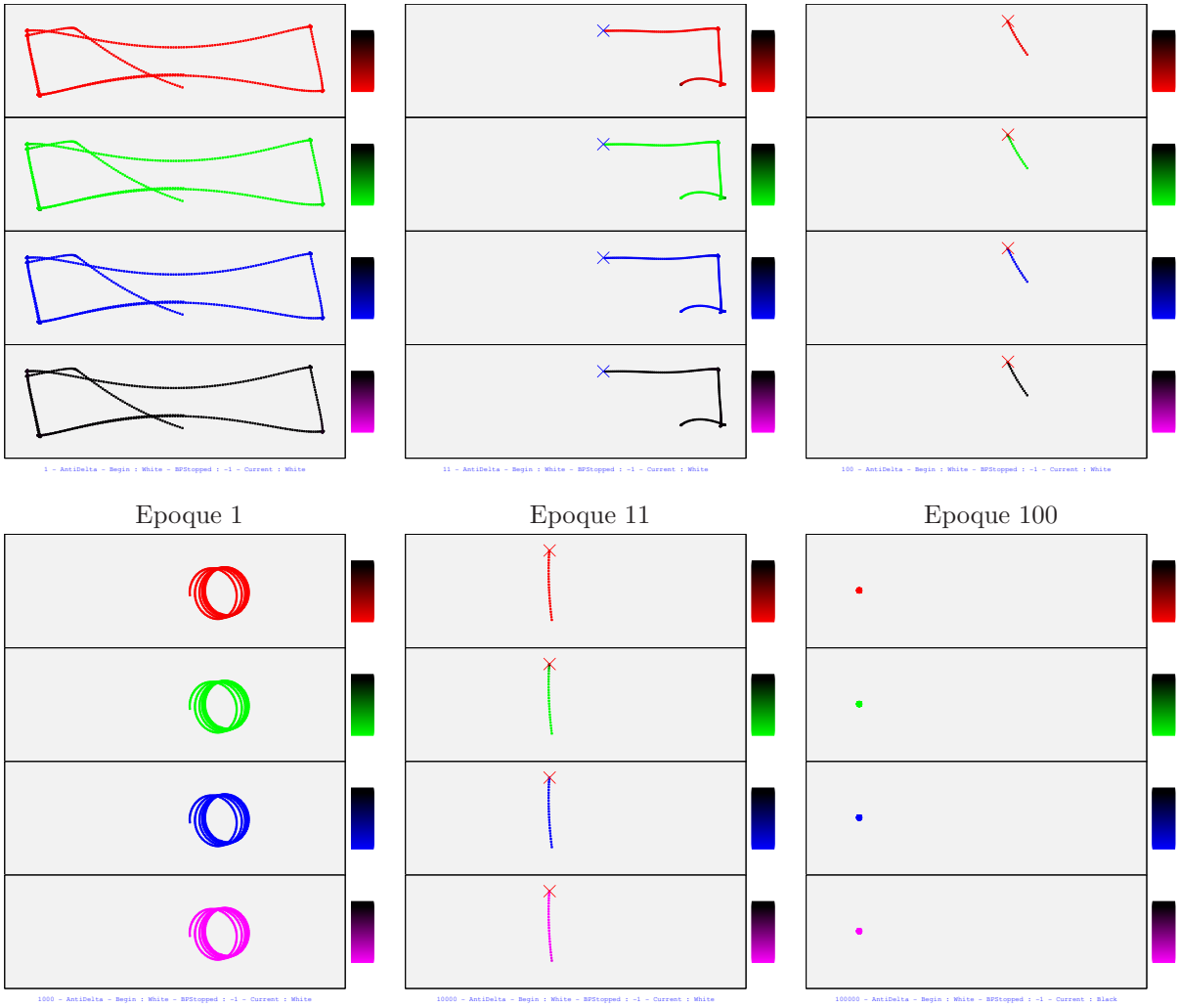


Evolution de la fitness à travers les époques



Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. B.7 – Essai 4

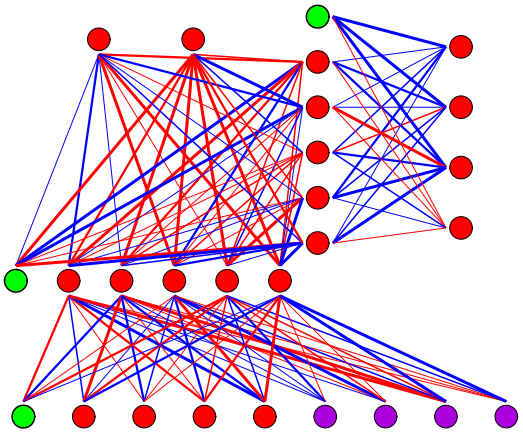


Epoque 1 | Epoque 11 | Epoque 100

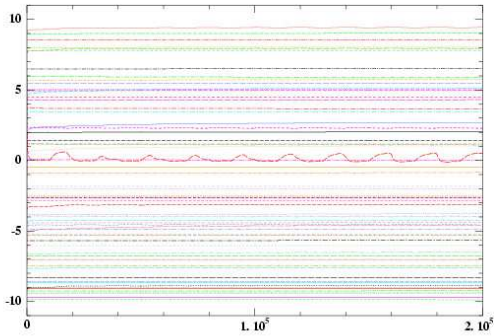
Epoque 1000 | Epoque 10000 | Epoque 100000

FIG. B.8 – Essai 4

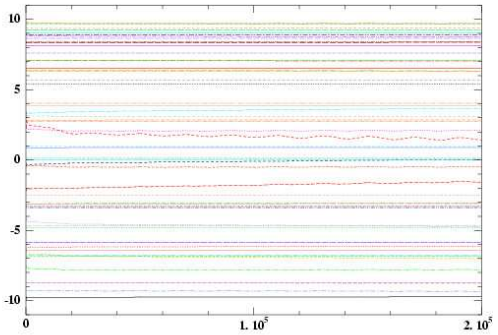
B.5 essai 5



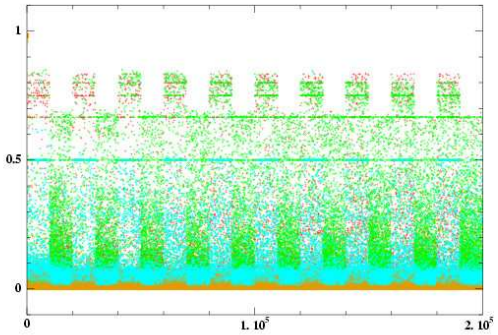
Les poids du réseau à la fin



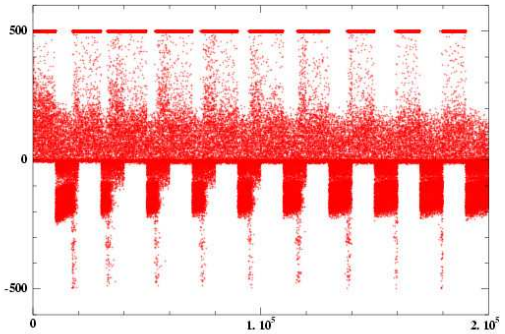
Detail des poids à travers les époques du réseau d'anticipation



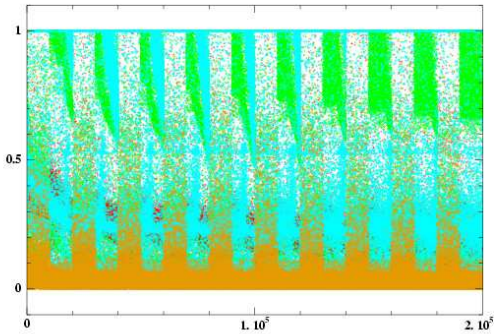
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

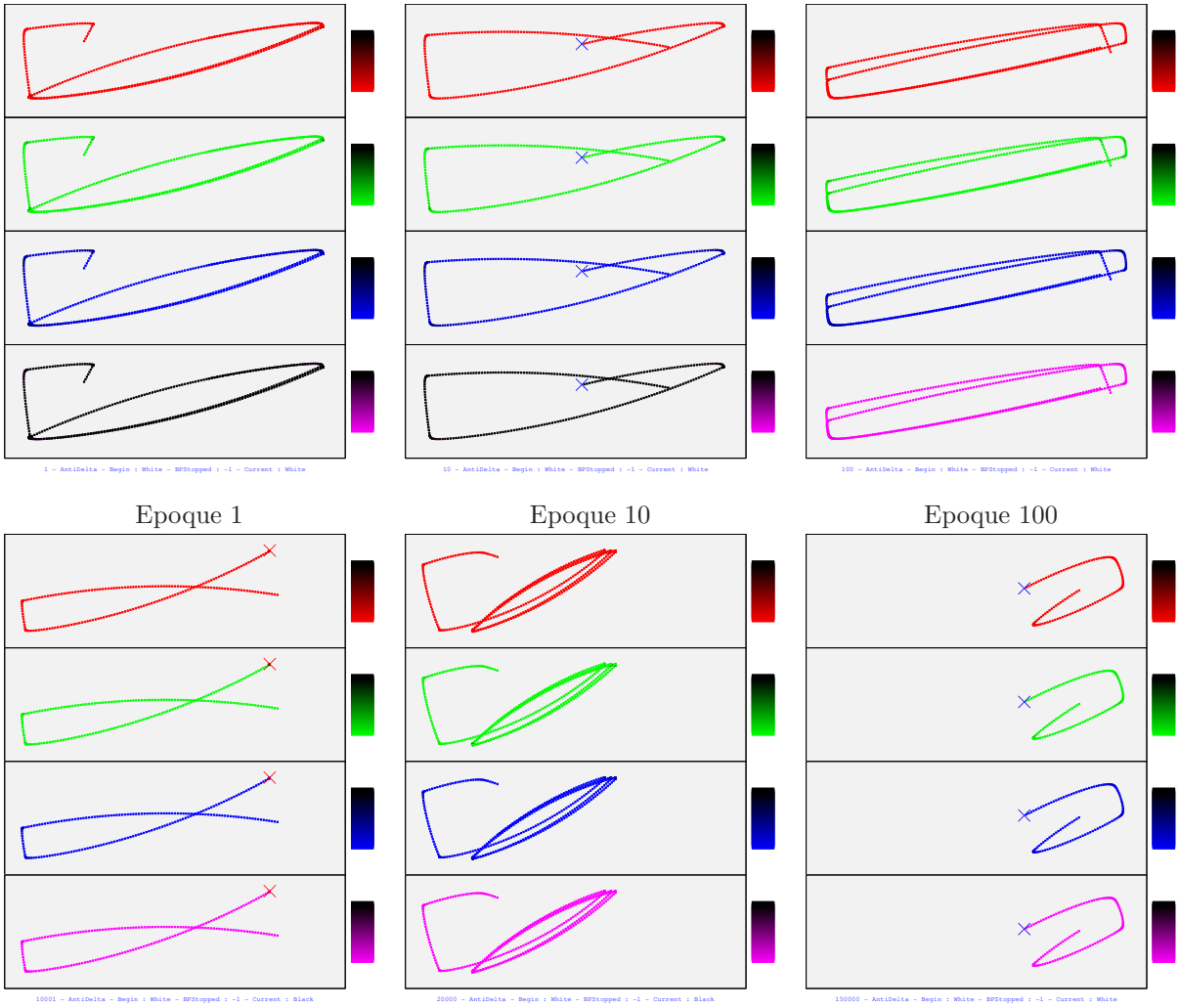


Evolution de la fitness à travers les époques



Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. B.9 – Essai 5

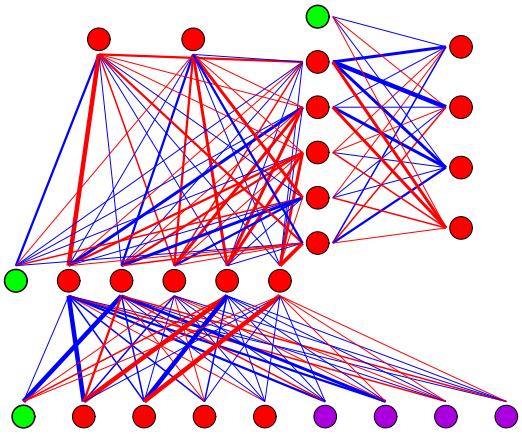


Epoque 1 | Epoque 10 | Epoque 100

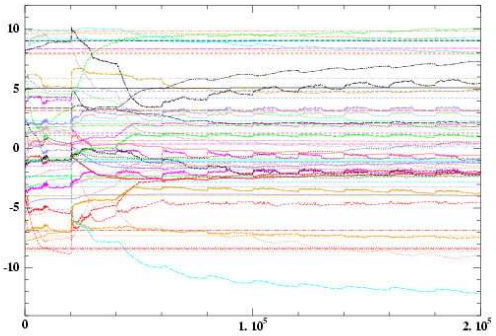
Epoque 10000 | Epoque 20000 | Epoque 150000

FIG. B.10 – Essai 5

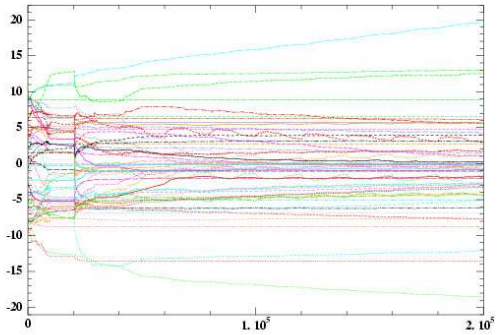
B.6 essai 6



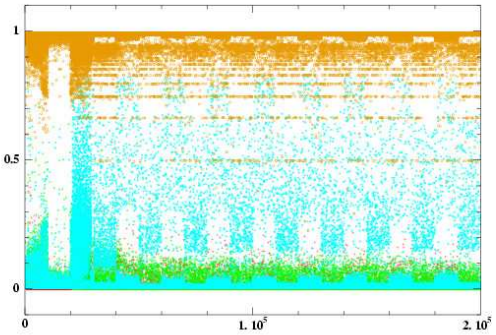
Les poids du réseau à la fin



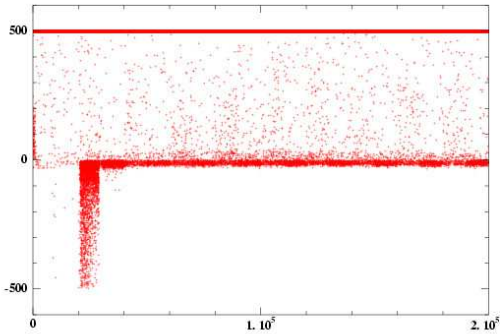
Detail des poids à travers les époques du réseau d'anticipation



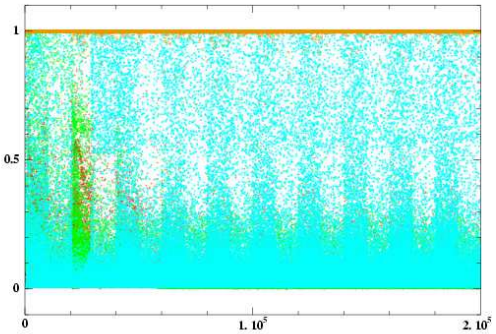
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

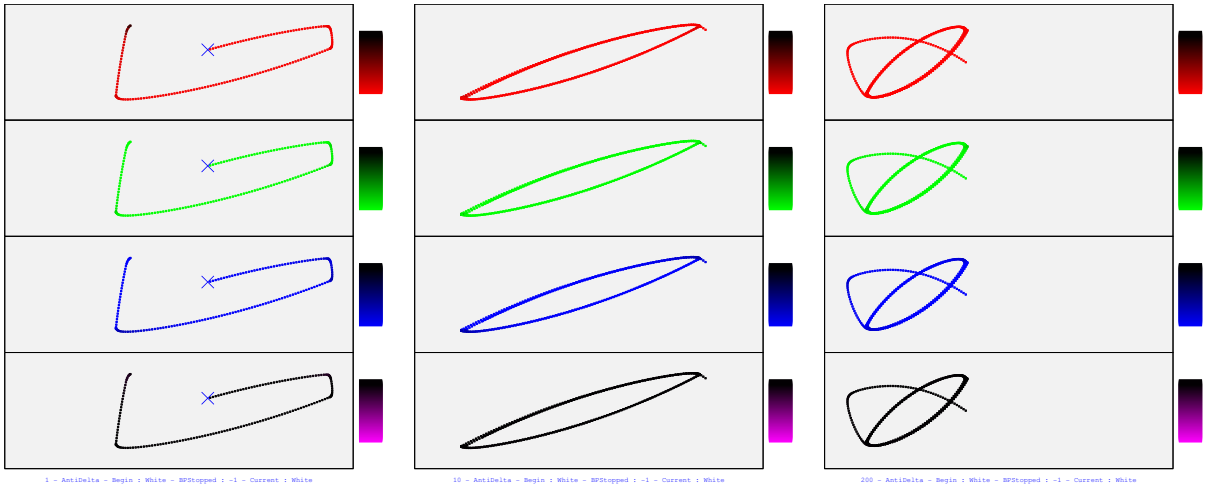


Evolution de la fitness à travers les époques

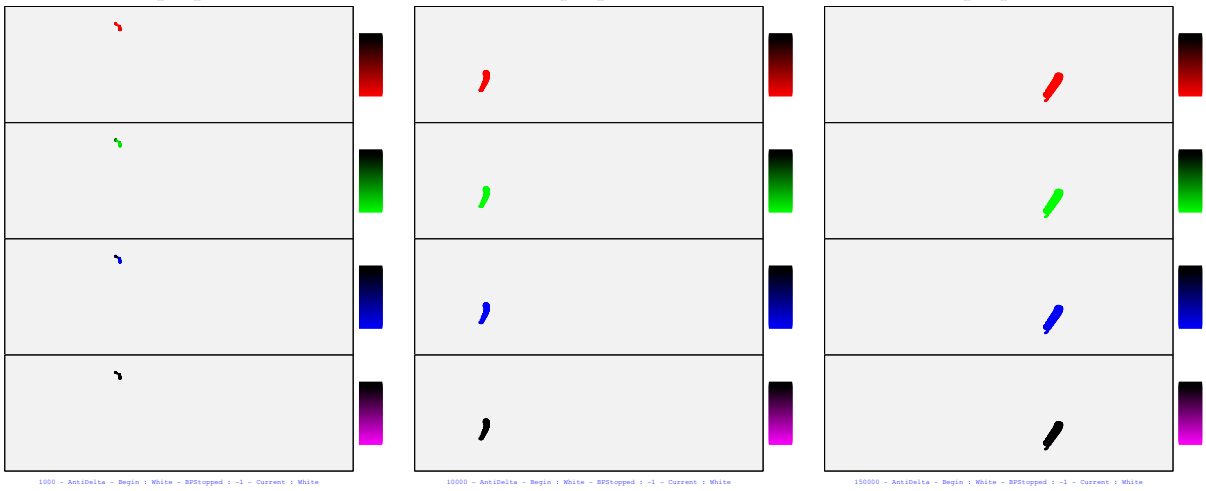


Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. B.11 – Essai 6



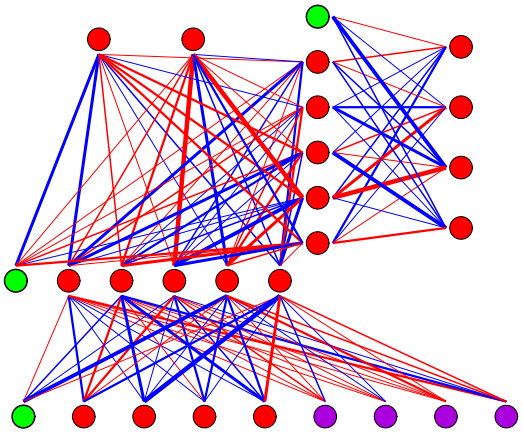
Epoque 1 | Epoque 10 | Epoque 200



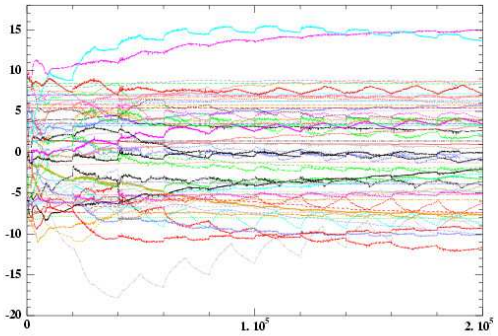
Epoque 1000 | Epoque 10000 | Epoque 150000

FIG. B.12 – Essai 6

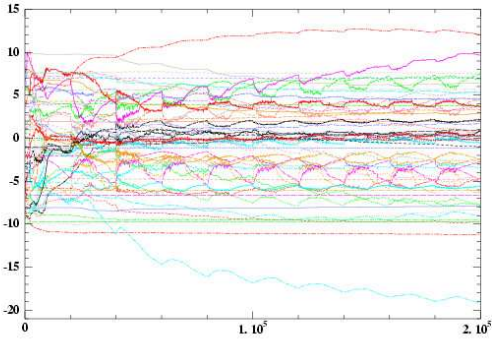
B.7 essai 7



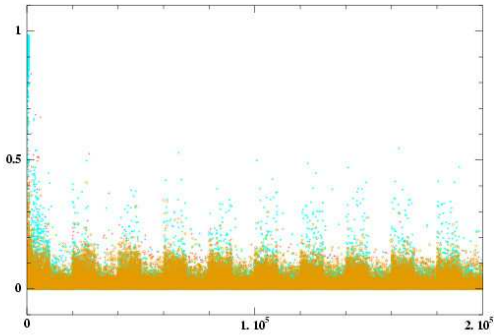
Les poids du réseau à la fin



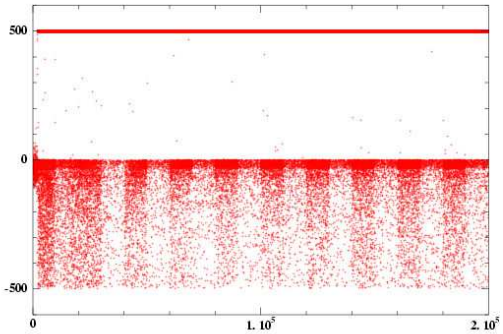
Detail des poids à travers les époques du réseau d'anticipation



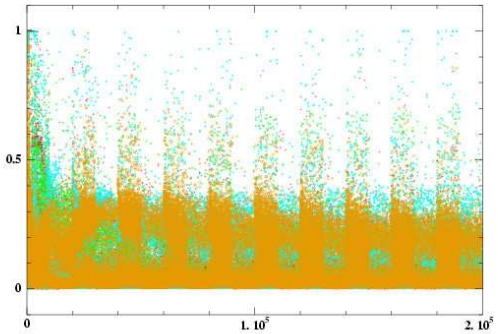
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

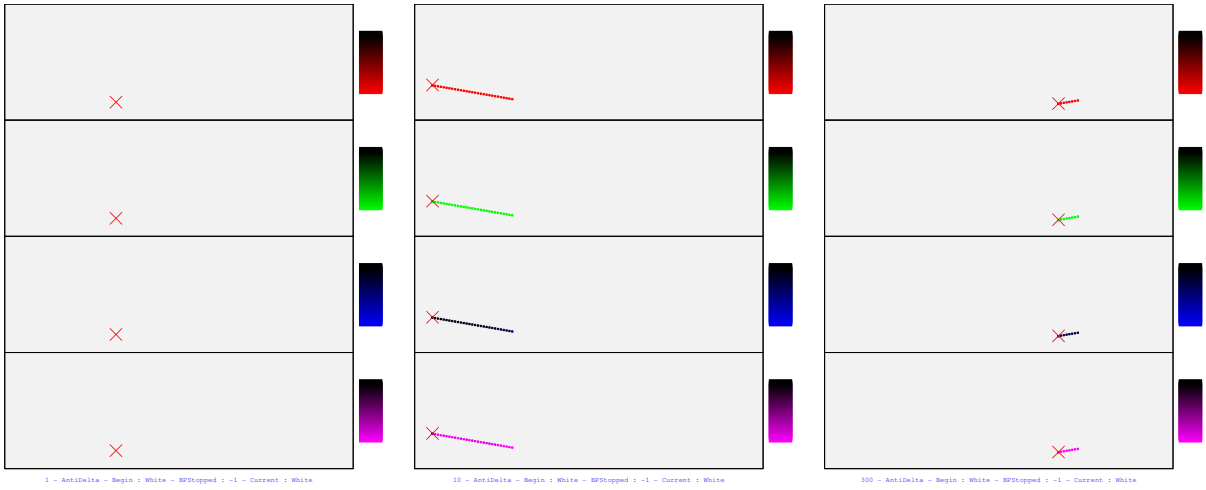


Evolution de la fitness à travers les époques

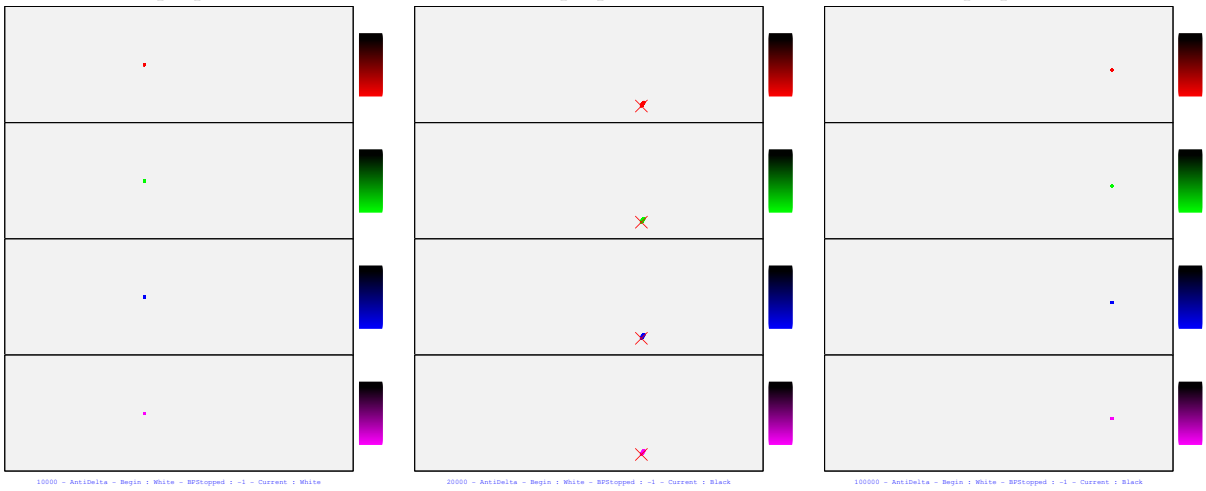


Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. B.13 – Essai 7



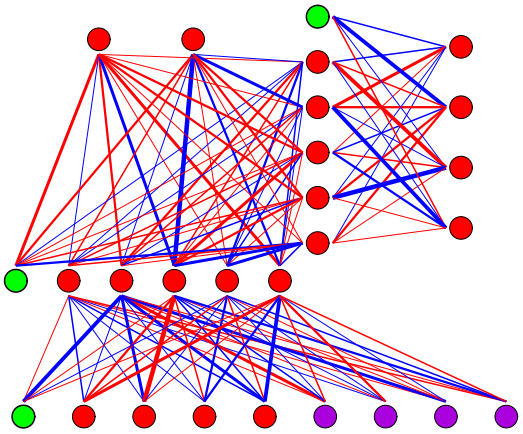
Epoque 1 | Epoque 10 | Epoque 300



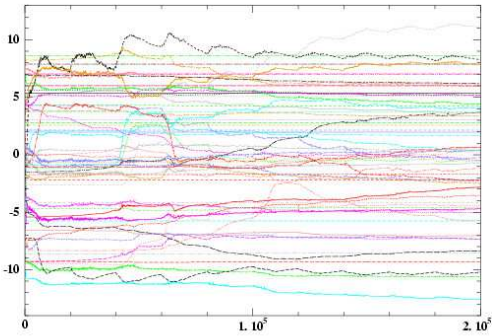
Epoque 10000 | Epoque 20000 | Epoque 100000

FIG. B.14 – Essai 7

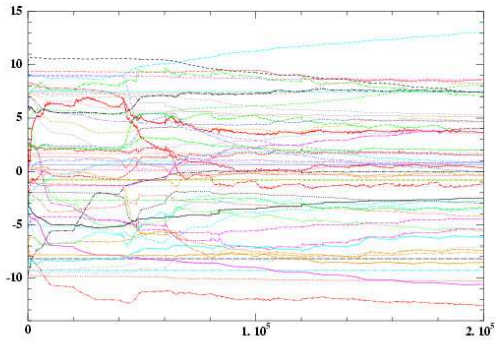
B.8 essai 8



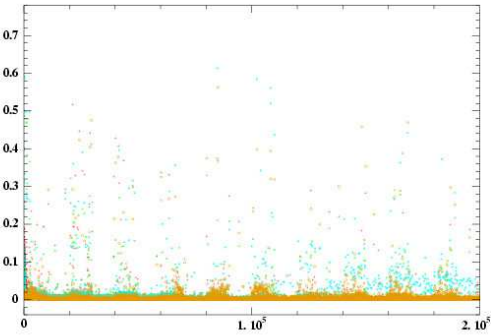
Les poids du réseau à la fin



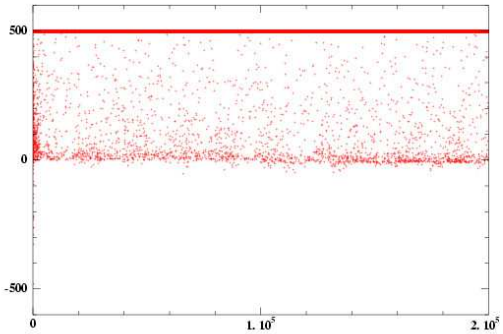
Detail des poids à travers les époques du réseau d'anticipation



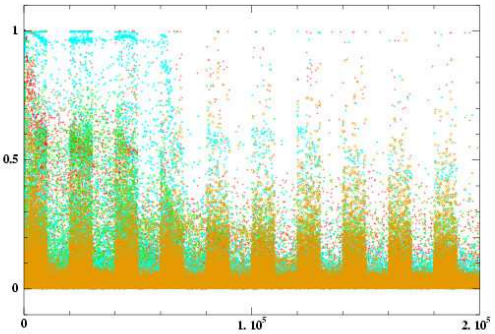
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

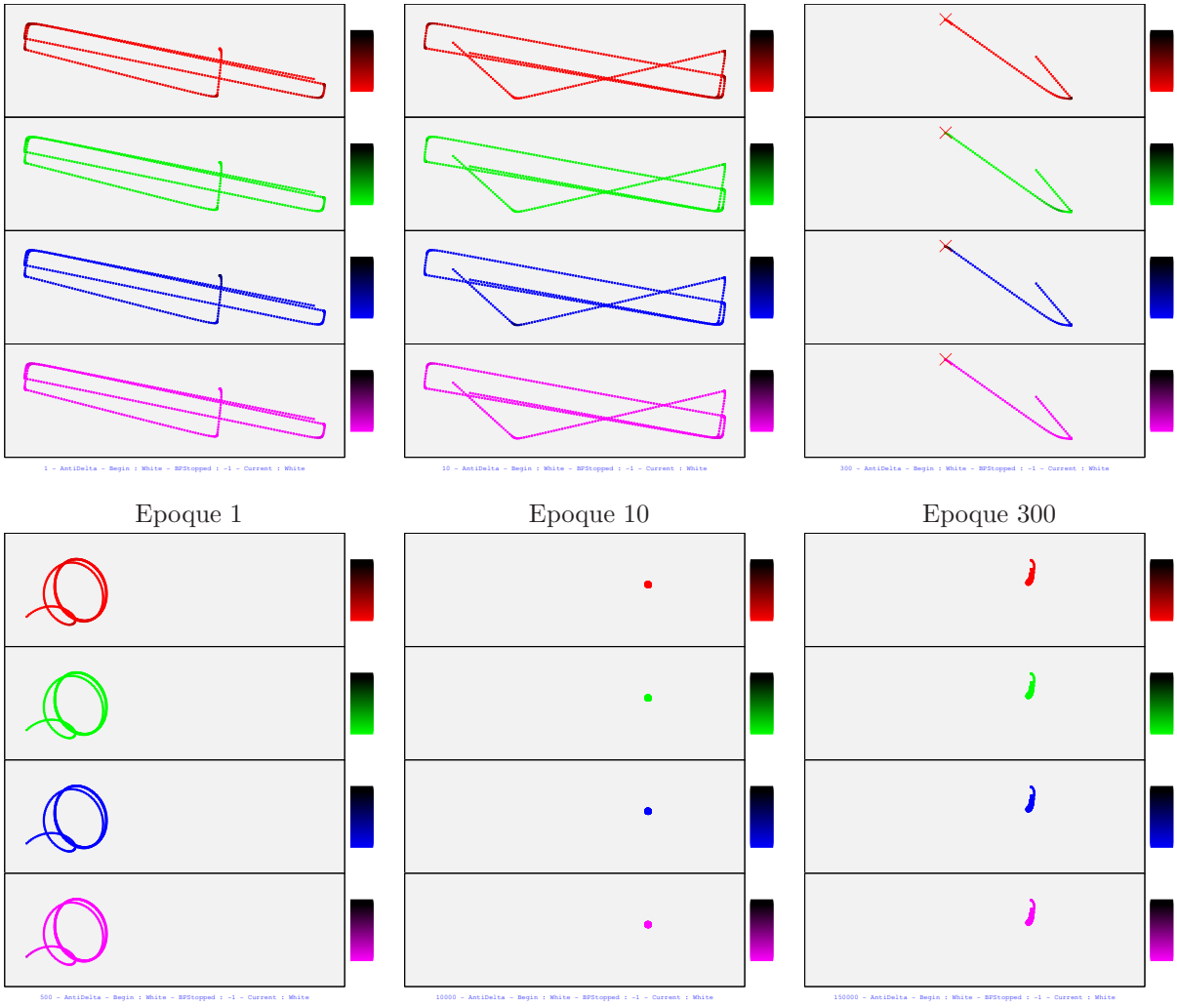


Evolution de la fitness à travers les époques



Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. B.15 – Essai 8

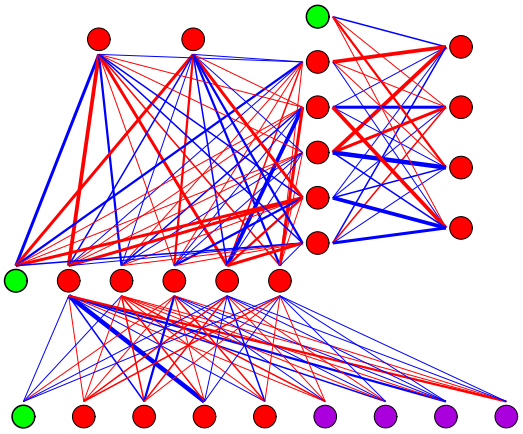


Epoque 1 | Epoque 10 | Epoque 300

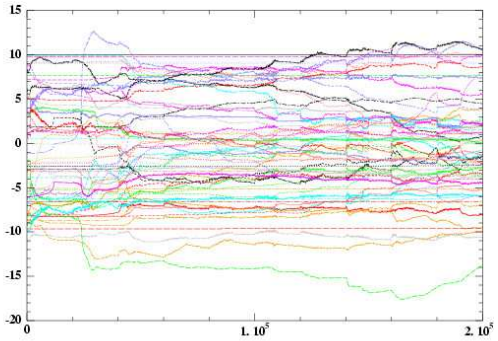
Epoque 500 | Epoque 10000 | Epoque 150000

FIG. B.16 – Essai 8

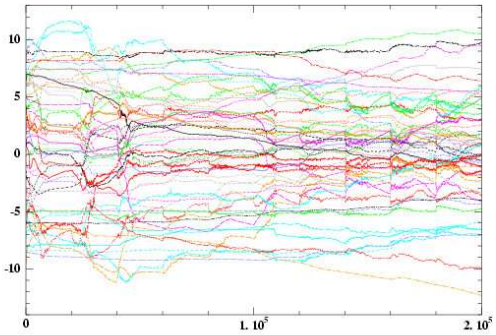
B.9 essai 9



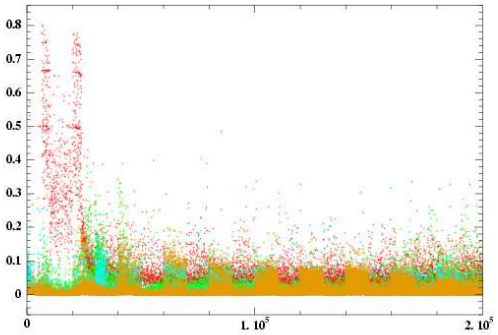
Les poids du réseau à la fin



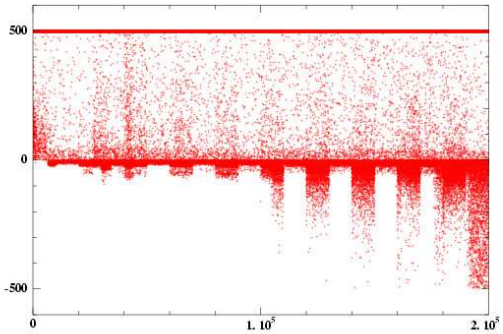
Detail des poids à travers les époques du réseau d'anticipation



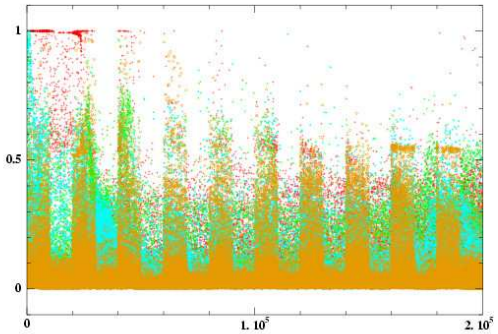
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

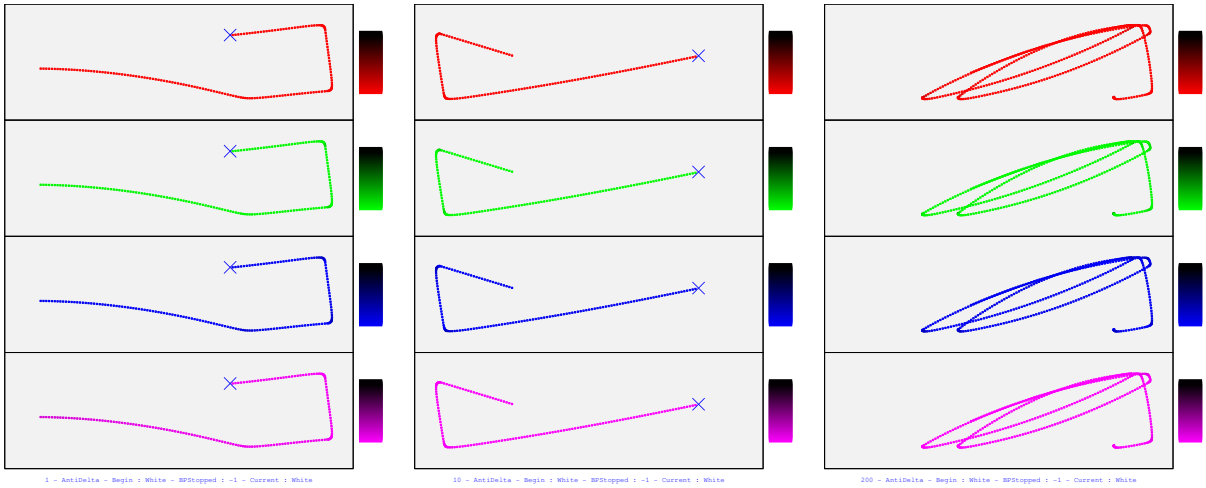


Evolution de la fitness à travers les époques

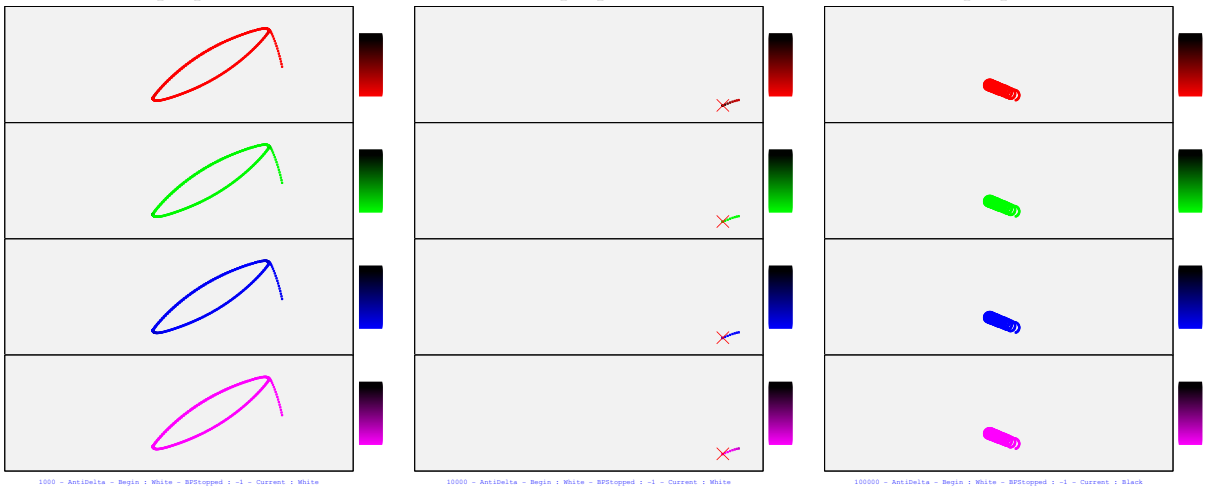


Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. B.17 – Essai 9



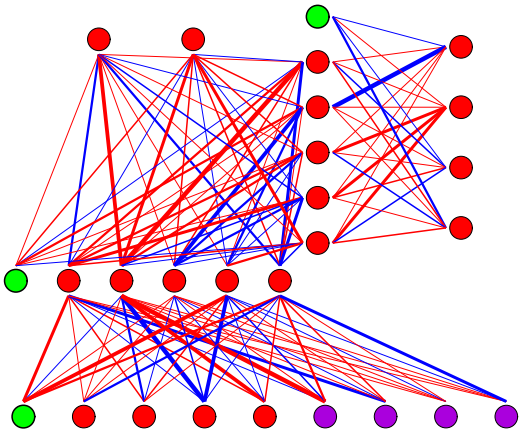
Epoque 1 | Epoque 10 | Epoque 200



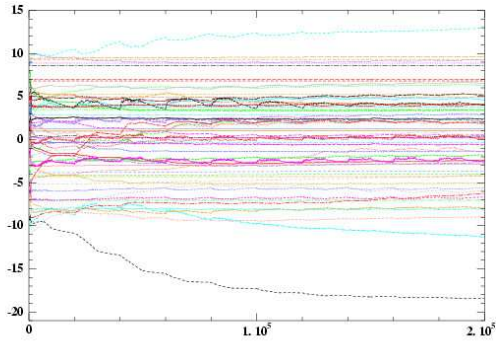
Epoque 1000 | Epoque 10000 | Epoque 100000

FIG. B.18 – Essai 9

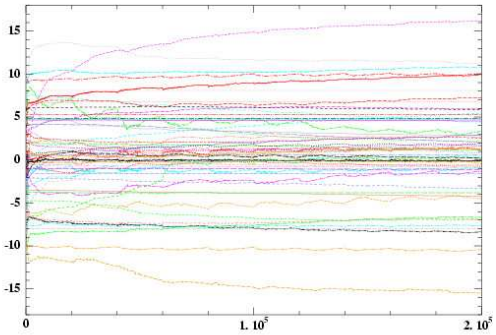
B.10 essai 10



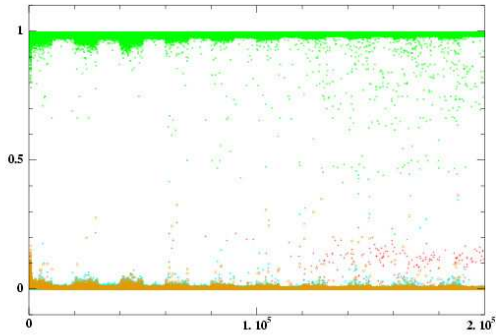
Les poids du réseau à la fin



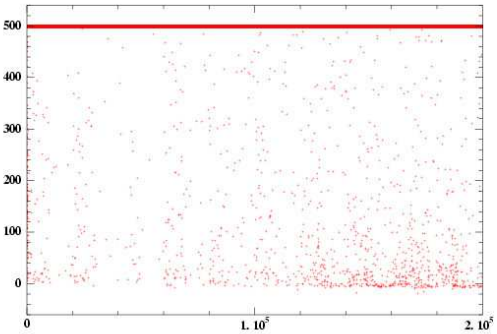
Detail des poids à travers les époques du réseau d'anticipation



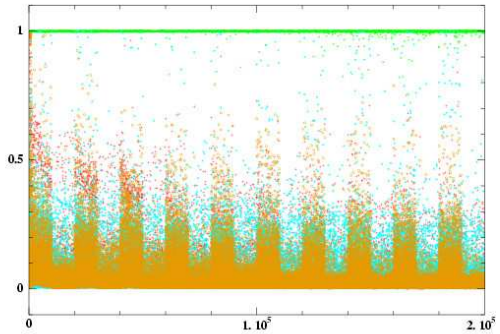
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

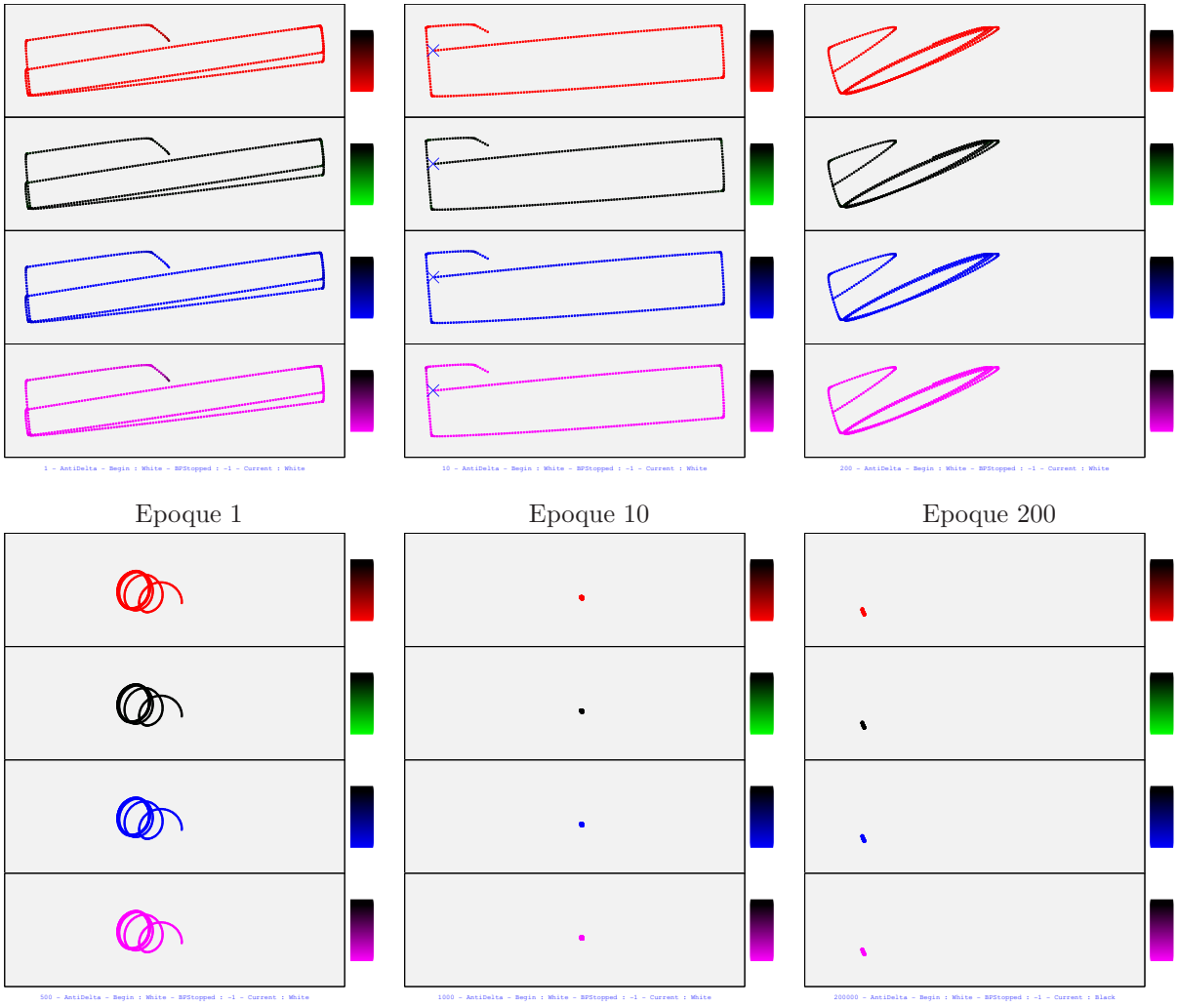


Evolution de la fitness à travers les époques



Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. B.19 – Essai 10

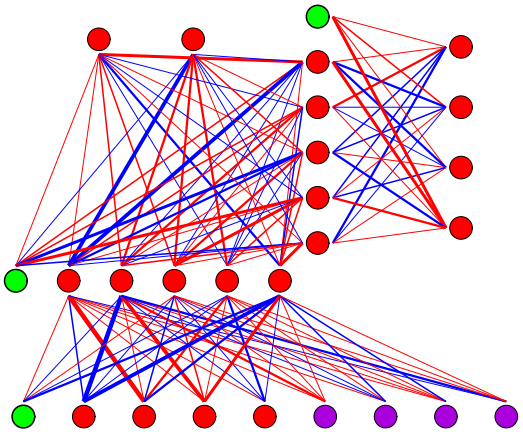


Epoque 1 | Epoque 10 | Epoque 200

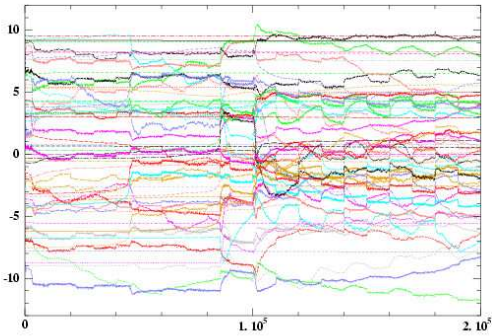
Epoque 500 | Epoque 1000 | Epoque 200000

FIG. B.20 – Essai 10

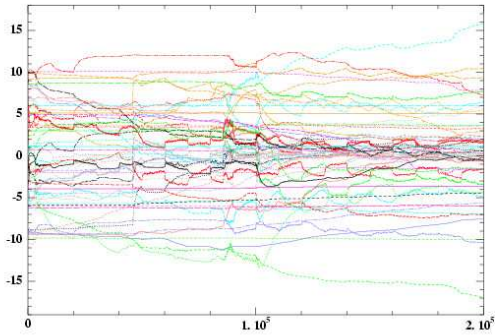
B.11 essai 11



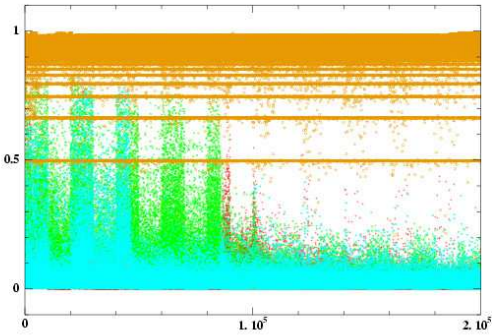
Les poids du réseau à la fin



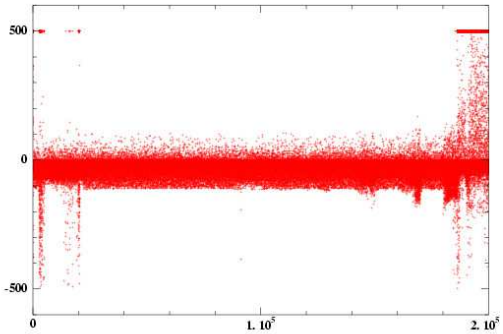
Detail des poids à travers les époques du réseau d'anticipation



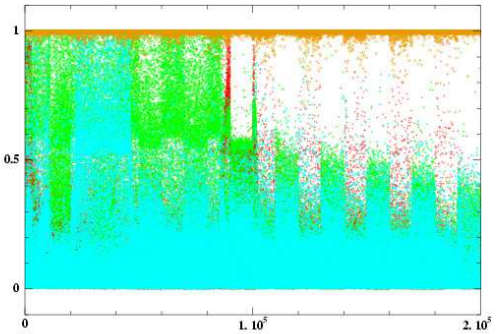
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

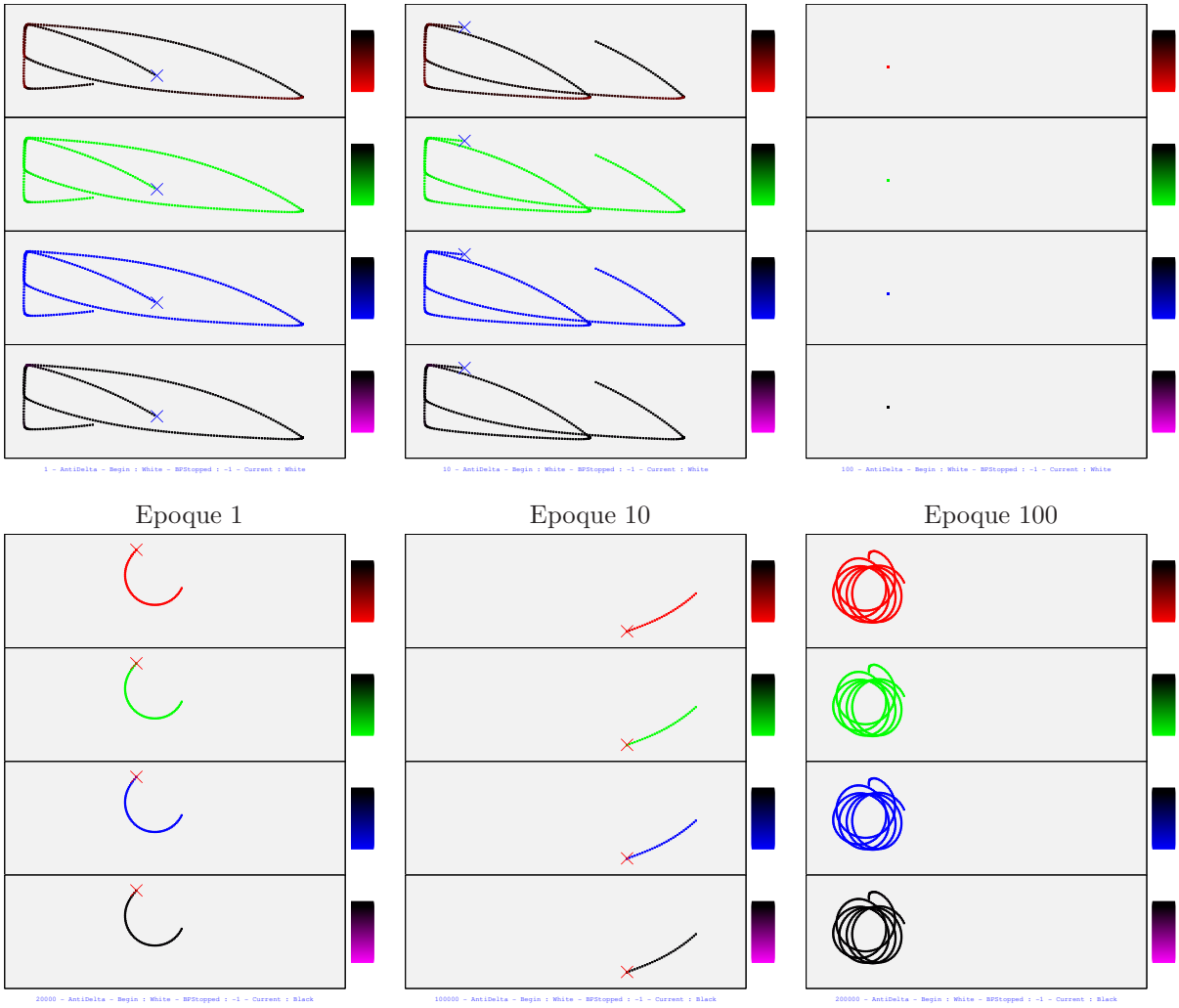


Evolution de la fitness à travers les époques



Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. B.21 – Essai 11

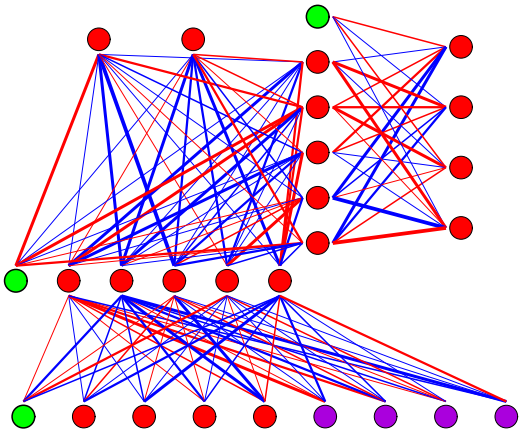


Epoque 1 | Epoque 10 | Epoque 100

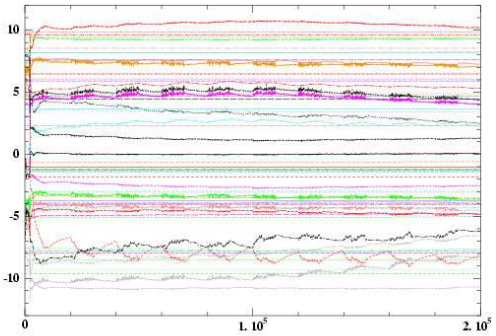
Epoque 20000 | Epoque 100000 | Epoque 200000

FIG. B.22 – Essai 11

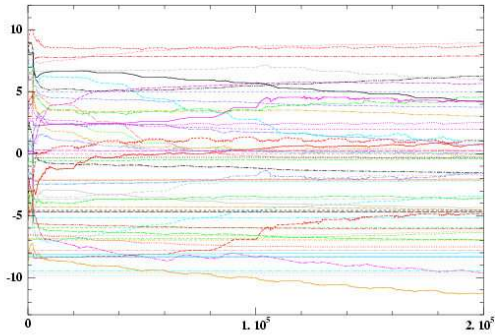
B.12 essai 12



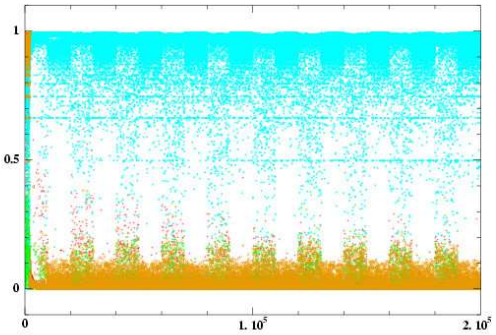
Les poids du réseau à la fin



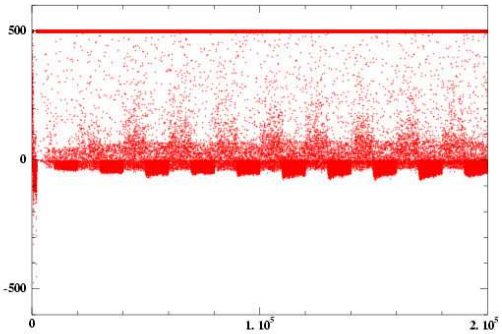
Detail des poids à travers les époques du réseau d'anticipation



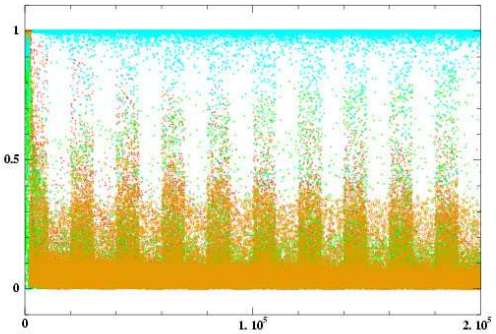
Detail des poids à travers les époques du réseau contrôleur



Erreurs d'anticipation moyennes ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

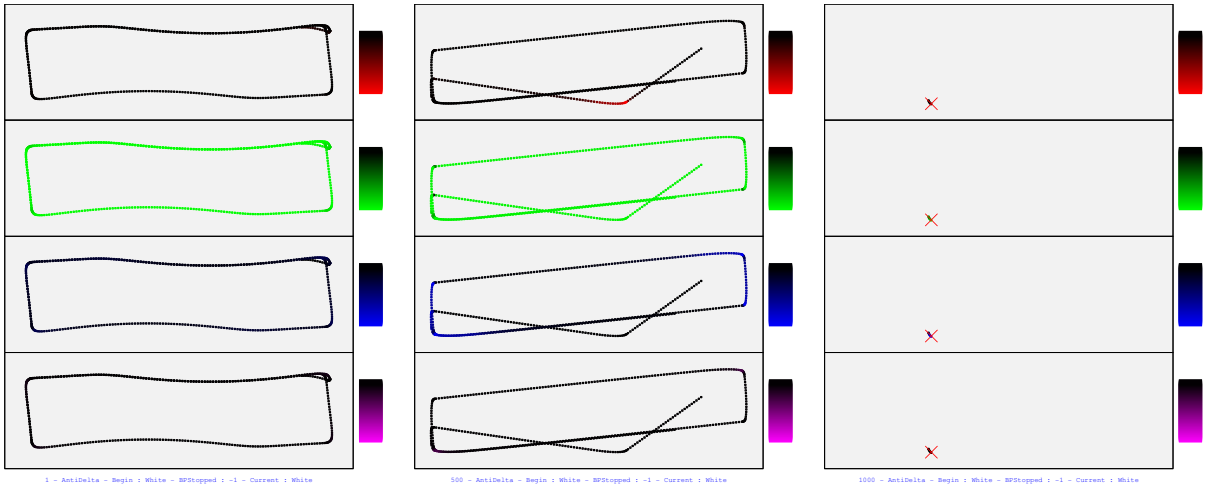


Evolution de la fitness à travers les époques

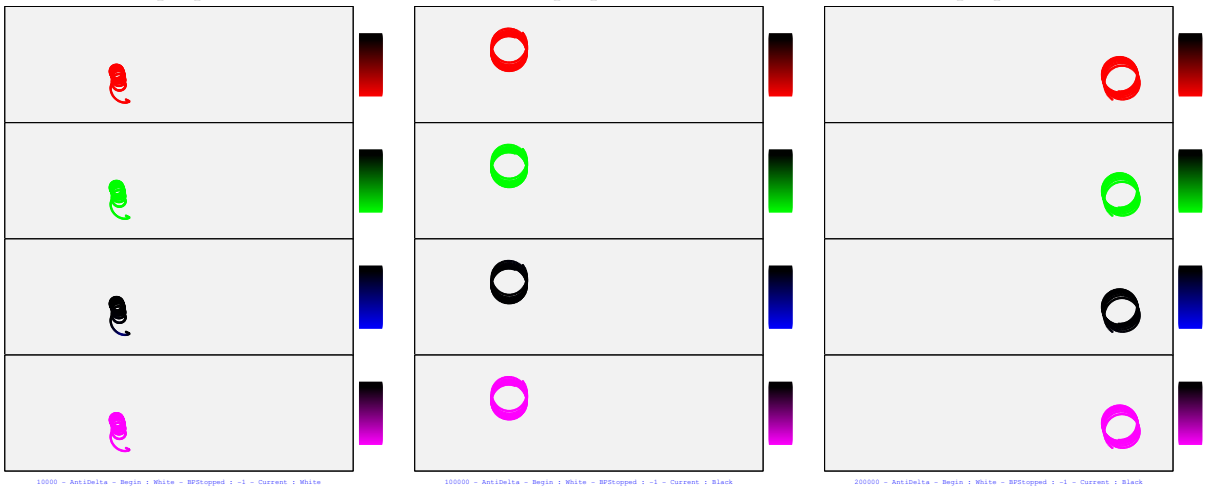


Erreurs d'anticipation maximales ; capteur 0=rouge, 1=vert, 2=bleu, 3=jaune

FIG. B.23 – Essai 12



Epoque 1 | Epoque 500 | Epoque 1000



Epoque 10000 | Epoque 100000 | Epoque 200000

FIG. B.24 – Essai 12

Bibliographie

[Angeline *et al.*, 1993] P. J. Angeline, G. M. Saunders, et J. B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(2) :86–91, 1993.

[Baldwin, 1896] J. M. Baldwin. A new factor in evolution. In *American Naturalist*, pages 441–451, 1896.

[Banzhaf *et al.*, 1998] W. Banzhaf, P. Nordin, R.E. Keller, et F.D. Francone. *Genetic Programming — An Introduction On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann, 1998.

[Barto *et al.*, 1983] A.G. Barto, R.S. Sutton, et C.W. Anderson. Neuronlike elements that can solve difficult learning control problems. In *IEEE Transactions on Systems, Man, and Cybernetics, 13 :835-846. Reprinted in J. A. Anderson and E. Rosenfeld, Neurocomputing : Foundations of Research, MIT Press, Cambridge, MA, 1988.*, 1983.

[Bentley, 1999] P. J. Bentley, editor. *Evolutionary Design by Computers*. Morgan Kaufman Publishers Inc., 1999.

[Blynel et Floreano, 2003] Jexper Blynel et Dario Floreano. Exploring the t-maze : Evolving learning-like robot behaviors using ctrnns. In *Applications of Evolutionary Computing*, pages 593–604. Springer, 2003.

[Brooks, 1986] R. A. Brooks. A robust layered control system for a mobile robot. *Journal of Robotics and Automation*, 2(1) :14–23, 1986.

[Brooks, 1990] Rodney A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6(1&2) :3–15, June 1990.

[Brooks, 1991] R. A. Brooks. How to build complete creatures rather than isolated cognitive simulators. In Kurt VanLehn, editor, *Architectures for Intelligence : The Twenty-second Carnegie Mellon Symposium on Cognition*, pages 225–239. Lawrence Erlbaum Associates, 1991.

[Bäck, 1995] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. New-York :Oxford University Press, 1995.

[Cramer, 1985] N.J. Cramer. A representation for the adaptive generation of simple sequential programs. In *Proceedings of the* 1^{st} *International Conference on Genetic Algorithms*, pages 183–187. Laurence Erlbaum Associates, 1985.

[Cun, 1987] Y. Le Cun. *Modèles connexionnistes de l'apprentissage*. PhD thesis, Université Pierre et Marie Curie, Paris VI, 1987.

[Darwin, 1859] Charles Darwin. *The Origin of Species*. 1859.

[Dawkins, 1988] R. Dawkins. *The blind watchmaker*. Norton, W. W. and Company, 1988.

[Doncieux, 2003] S. Doncieux. *Évolution de contrôleurs neuronaux pour animats volants : méthodologie et applications*. PhD thesis, LIP6/AnimatLab, Université Pierre et Marie Curie, Paris, France, 2003. Spécialité Informatique.

[Dorigo et Bersini, 1994] M. Dorigo et H. Bersini. A comparaison of q-learning and classifier systems. pages 248–255. Cliff et al., 1994.

[Eiben et Smith, 2003] A.E. Eiben et J.E. Smith. *Introduction to Evolutionary Computing*. Springer Verlag, 2003.

[Elman, 1990] J.L. Elman. Finding structure in time. *Cognitive Science*, pages 179–211, 1990.

[Fadda et Schoenauer, 1995] A. Fadda et M. Schoenauer. Evolutionary chromatographic law identification by recurrent neural nets. In J. R. McDonnell, R. G. Reynolds, et D. B. Fogel, editors, *Proceedings of the* 4^{th} *Annual Conference on Evolutionary Programming*, pages 219–235. MIT Press, March 1995.

[Floreano et Mondada, 1994a] Floreano et Mondada. Automatic creation of an autonomous agent : genetic evolution of a neural-network driven robot. In Cliff D., Husbands P., Meyer J., et Wilson S.W., editors, *From animals to animats 3. Proceedings of the Third International Conference on Simulation of Adaptive Behavior*. The MIT Press/Bradford Books, 1994.

[Floreano et Mondada, 1994b] D. Floreano et F. Mondada. Evolution of homing navigation in a real mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics*, 26 :396–407, 1994.

[Floreano et Mondada, 1996] D. Floreano et F. Mondada. Evolution of homing navigation in a real mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics*, pages Part B : Cybernetics, 396–407, 1996.

[Floreano et Urzelai, 2001] D. Floreano et J. Urzelai. Neural morphogenesis, synaptic plasticity, and evolution. *Theory in Biosciences*, 120(3-4) :225–240, 2001.

[Fogel *et al.*, 1966] L. J. Fogel, A. J. Owens, et M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. New York : John Wiley, 1966.

[Fogel, 1995] D. B. Fogel. *Evolutionary Computation. Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, NJ, 1995.

[Godzik *et al.*, 2004] N. Godzik, M. Schoenauer, et M. Sebag. Robustness in the long run : Auto-teaching vs anticipation in evolutionary robotics. In X. Yao et al., editor, *Parallel Problem Solving from Nature VIII*, pages 932–941. LNCS 3242, Springer Verlag, 2004.

[Goldberg, 1989] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, 1989.

[Gruau, 1994] F. Gruau. *Neural Network Synthesis using Cellular encoding and the Genetic Algorithm*. PhD thesis, Ecole Normale Superieure de Lyon, 1994.

[Harvey, 1993] I. Harvey. *The Artificial Evolution of Adaptive Behaviour*. PhD thesis, University of Sussex, 1993.

[Hebb, 1949] D. Hebb. *The Organisation of Behaviour*. John Wiley (New York), 1949.

[Holland et Reitman, 1978] J.H. Holland et J.S. Reitman. Cognitive systems based on adaptive algorithms. In *Pattern directed inference systems*, pages 125–149, 1978.

[Holland, 1975] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[Hornik *et al.*, 1989] K. Hornik, M. Stinchcombe, et H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, pages 359–366, 1989.

[Jacobi, 1997] N. Jacobi. Half-baked, ad-hoc ans noisy : Minimal simulations for evolutionnary robotics. *Fourth European Conference on Artificial Life, ECAL*, pages 348–357, 1997.

[Jacobi, 1998] N. Jacobi. *Minimal Simulations for Evolutionary Robotics*. PhD thesis, University of Sussex, Great Britain, 1998.

[Keijzer *et al.*, 2002] M. Keijzer, J. J. Merelo, G. Romero, et M. Schoenauer. Evolving Objects : a general purpose evolutionary computation library. In P. Collet, E. Lutton, M. Schoenauer, C. Fonlupt, et J.-K. Hao, editors, *Artificial Evolution'01*, pages 229–241. Springer Verlag, LNCS 2310, 2002. URL : http ://eodev.sourceforge.net/.

[Klopf, 1972] A. H. Klopf. Brain function and adaptive systems–a heterostatic theory. In *Technical Report AFCRL-72-0164, Air Force Cambridge Research Laboratories, Bedford, MA. A summary appears in Proceedings of the International Conference on Systems, Man, and Cybernetics, 1974, IEEE Systems*, 1972.

[Klopf, 1975] A. H. Klopf. A comparison of natural and artificial intelligence, 1975.

[Klopf, 1982] A. H. Klopf. The hedonistic neuron : A theory of memory, learning, and intelligence, 1982.

[Kodjabachian et Meyer, 1998] J. Kodjabachian et J.A Meyer. Evolution and development of modular control architectures for 1-d locomotion in six-legged animats. *Connection Science*, 10 :211–237, 1998.

[Koza et al., 1999] J. R. Koza et al. *Genetic Programming III : Automatic Synthesis of Analog Circuits*. MIT Press, Massachussetts, 1999.

[Koza, 1992] J. R. Koza. *Genetic Programming : On the Programming of Computers by means of Natural Evolution*. MIT Press, Massachusetts, 1992.

[Koza, 1994] J. R. Koza. *Genetic Programming II : Automatic Discovery of Reusable Programs*. MIT Press, Massachussetts, 1994.

[Lamarck, 1914] J. B. Lamarck. 1914.

[Lenat et Guha, 1990] D. Lenat et R. V. Guha. Building large knowledge-based systems : Representation and inference in the cyc project. Addison Wesley Publ. Co., 1990.

[Maes, 1989] P. Maes. The dynamics of action selection. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, 1989.

[McCulloch et Pitts, 1943] W. McCulloch et W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(115), 1943.

[Meyer et Guillot, 1991] J.A. Meyer et A. Guillot. Simulation of adaptive behavior in animats : review and prospect. In S. Meyer, J.A. et Wilson, editor, *Proc. First Intl Conf. on Simulation of Adaptive Behavior*. MIT Press, 1991.

[Michie et Chambers, 1968] D. Michie et R. A. Chambers. Boxes : An experiment in adaptive control. In E. Dale et D. Michie, editors, *Machine Intelligence 2*, pages 137–152. Oliver and Boyd, 1968.

[Michie, 1961] D. Michie. Trial and error. pages 129–145. Harmondsworth, 1961.

[Michie, 1963] D. Michie. Experiments on the mechanisation of game learning. characterization of the model and its parameters., 1963.

[Minsky et Papert, 1969] Marvin L. Minsky et Seymour A. Papert. *Perceptrons*. MIT Press, 1969.

[Mondada et Floreano, 1995] F. Mondada et D. Floreano. Evolution of neural control structures : Some experiments on mobile robots. *Robotics and Autonomous Systems*, 16 :183–195, 1995.

[Munos et Moore, 1999] R. Munos et A. Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In *Proc. IJCAI'99*, 1999.

[Munos, 2000] R. Munos. A study of reinforcement learning in the continuous case by the means of viscosity solutions. *Machine Learning Journal*, 40 :265–299, 2000.

[Nguyen et Widrow, 1990] D. Nguyen et B. Widrow. *Neural networks for control*, chapitre The truck backer-upper : an example of self-learning in Neural Networks, pages 287–300. MIT Press, 1990.

[Nolfi *et al.*, 1994a] S. Nolfi, J.L. Elman, et D. Parisi. Learning and evolution in neural networks. *Adaptive Behavior*, 3(1) :5–28, 1994.

[Nolfi *et al.*, 1994b] S. Nolfi, D. Floreano, O. Miglino, et F. Mondada. How to evolve autonomous robots : Different approaches in evolutionary robotics. In R. Brooks et P. Maes, editors, *Artificial Life IV*, pages 190–197. MIT Press/Bradford Books, 1994.

[Nolfi et Floreano, 2000] S. Nolfi et D. Floreano. *Evolutionary Robotics, The Biology, Ingelligence, and Technologie of Self-Organizing Machines*. MIT Press, 2000.

[Nolfi et Parisi, 1993] S. Nolfi et D. Parisi. Auto-teaching : networks that develop their own teaching input. In J. Deneubourg, H. Bersini, S. Goss, G. Nicolis, et R. Dagonnier, editors, *Proc. Second European Conference on Artificial Life*, 1993.

[Nolfi et Parisi, 1997] S. Nolfi et D. Parisi. Learning to adapt to changing environments in evolving neural networks. *Adaptive Behavior*, 5(1) :75–98, 1997.

[O'Regan et Noë, 2001] J. Kevin O'Regan et Alva Noë. A sensorimotor account of vision and visual consciousness. *Behavioral and Brain Sciences*, 24(5), 2001.

[Pavlov, 1927] Pavlov. Conditionned reflexes. Oxford University Press, 1927.

[Rechenberg, 1972] I. Rechenberg. *Evolutionstrategie : Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution*. Fromman-Hozlboog Verlag, Stuttgart, 1972.

[Ronald et Schoenauer, 1994] E. Ronald et M. Schoenauer. Genetic lunar lander : an experiment in accurate neuro-control. In Y. Davidor, H.-P. Schwefel, et R. Manner, editors, *Proceedings of the* 3^{rd} *Conference on Parallel Problems Solving from Nature*, pages 452–461. Springer-Verlag, LNCS 866, 1994.

[Rosen, 1985] R. Rosen. Anticipatory systems. Pergamon Press, 1985.

[Rumelhart *et al.*, 1986] Rumelhart, Hinton, et Williams. Learning internal representations by error propagation. In *Parallel distributed processing - Explorations in the microstructure of cognition*, pages 318–362. MIT Press, 1986.

[Rumelhart et McClelland, 1986] D.E. Rumelhart et J.L. McClelland. *Parallel Distributed Processing*. MIT Press, 1986.

[Schwefel, 1981] H.-P. Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, New-York, 1981. 1995 – 2^{nd} edition.

[Searle, 1980] J. Searle. Minds, brains, and programs. *Behavorial and Brain Sciences*, pages 417–424, 1980.

[Sims, 1994] K. Sims. Evolving virtual creatures. In *SIGGRAPH'94 Proceedings*, pages 15–22. Computer Graphics, Annual Conference Series, July 1994.

[Skinner, 1948] B.F. Skinner. "superstition" in the pigeon. 1948.

[Smith, 1980] S. F. Smith. *A learning system based on genetic adaptive algorithms*. PhD thesis, University of Pittsburgh, 1980.

[Stanley et Miikkulainen, 2002] K. O. Stanley et R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2) :99–127, 2002.

[Stolzmann, 1998] W. Stolzmann. Anticipatory classifier systems. 1998.

[Surry et Radcliffe, 1996] P.D. Surry et N.J. Radcliffe. Inoculation to initialize evolutionary search. In *Evolutionary Computing : AISB workshop*, number 1141 in LNCS, pages 269–285. Springer Verlag, 1996.

[Sutton et Barto, 1998] R.S. Sutton et A. G. Barto. *Reinforcement learning*. MIT Press, 1998.

[Sutton, 1984] R. S. Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts, 1984.

[S.W.Wilson, 1995] S.W.Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2) :149–175, 1995.

[Tani et Kukumura, 94] J. Tani et N. Kukumura. Learning goal-directed sensory-based navigation of a mobile robot. *Neural Networks : The Official Journal of the International Neural Networks Society*, 7(3) :553–564, 94.

[Tesauro et Sejnowski, 1987] G. Tesauro et T.J. Sejnowski. A 'neural' network that learns to play backgammon. In *NIPS*, pages 794–803, 1987.

[Tuci *et al.*, 2002] E. Tuci, I. Harvey, et M. Quinn. Evolving integrated controllers for autonomous learning robots using dynamic neural networks. In B. Hallam et al., editor, *Proc. of SAB'02*. MIT Press, 2002.

[Vapnik, 1998] V. N. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[W. T. Miller *et al.*, 1990] III W. T. Miller, R. S. Sutton, et P. J. Werbos, editors. *Neural networks for control*. MIT Press, 1990.

[Watkins, 1989] C. J. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, 1989.

[Wilson, 2000] S. W. Wilson. Get real ! xcs with continuous-valued inputs. In P. L. Lanzi, W. Stolzmann, et S. W. Wilson, editors, *Learning Classifier Systems : From Foundations to Applications*, number 1813 in LNAI, pages 209–220. Springer Verlag, 2000.

[Yamauchi et Beer, 1994] B.M. Yamauchi et R.D. Beer. Integrating reactive, sequential, and learning behavior using dynamical neural network. In D. Cliff et al., editor, *Proc. SAB'94*. MIT Press, 1994.

[Yao et Liu, 1997] X. Yao et Y. Liu. A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, 8(3) :694–713, 1997.

[Yao, 1999] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9) :1423–1447, 1999.