Une représentation hiérarchique de comportements agents pour l'apprentissage progressif et continu

François Suro, Jacques Ferber, Tiberiu Stratulat, Fabien Michel

Une représentation hiérarchique de comportements agents pour l'apprentissage progressif et continu

François Suro suro@lirmm.fr
Jacques Ferber ferber@lirmm.fr
Tiberiu Stratulat stratulat@lirmm.fr
Fabien Michel fmichel@lirmm.fr

LIRMM, Université de Montpellier, CNRS, Montpellier, France

Résumé

Dans la perspective d'un développement ouvert et continu il est cruciale qu'un comportement final souhaité à un instant donné puisse être un élément pour la création future de comportements plus complexes, dont le but ne peut être anticipé. Nous proposons MIND (Modular Influence Network Design), une architecture de contrôle pour agent artificiel conçue pour apprendre des comportements à partir d'un Curriculum établi par un instructeur humain. MIND encapsule les comportements dans des modules et les combine en une hiérarchie reflétant la nature modulaire et hiérarchique des tâches complexes, et permet la préservation et la réutilisation des compétences acquises.

Mots-clés : *Robotique développementale, Apprentissage agent, Architecture modulaire, Architecture hiérarchique, Apprentissage par Curriculum*

Abstract

In open ended and continuous development the desired behaviour at a given time can be an element of future behaviours of greater complexity, the purpose of which cannot be anticipated. MIND (Modular Influence Network Design) is a control architecture for artificial agents designed to learn behaviours from a curriculum developed by a human instructor by encapsulation into modules. These skill modules are combined into a hierarchy that perform behaviours of greater complexity while allowing the preservation and reuse of acquired skills.

Keywords: *Developmental robotics, Agent learning, Modular architecture, Hierarchical architecture, Curriculum learning*

1 Introduction

La théorie de Jean Piaget [13][14] sur le développement cognitif chez l'homme comme processus dynamique de schémas de coordination en plusieurs étapes (depuis les schémas sensorimoteurs jusqu'aux opérations de niveau abstrait) est un point de départ pour toute recherche sur la robotique épigénétique [9]. L'apprentissage se fait progressivement par l'interaction entre l'enfant et son environnement, des tâches plus complexes s'ajoutant à des tâches plus simples. Cette idée a influencé le domaine de la robotique développementale, où il est fondamental que la complexité des connaissances et des compétences acquises par un agent artificiel augmente progressivement [11]. Selon Piaget, la complexité de l'apprentissage peut évoluer selon trois axes :

1. La complexité de l'environnement peut croître progressivement, du bac à sable aux situations réelles.
2. Les motivations peuvent être de plus en plus complexes, de saisir un objet à la construction d'une maison.
3. Les compétences et leur structuration en comportements doivent se développer pour faire face à l'augmentation de la complexité de l'environnement et des motivations.

Depuis les premiers pionniers de la Renaissance qui ont identifié des zones du cerveau humain liées à des fonctions spécifiques jusqu'aux travaux plus récents sur la coordination entre ces zones, la recherche montre que les systèmes biologiques utilisent une approche modulaire [6]. Différentes zones du cerveau sont dédiées à des processus spécifiques et organisées en modules pour accomplir des tâches. Les travaux sur le cortex visuel des primates visant à créer un pont entre la biologie et la vision par ordinateur ont mis en évidence la structure hiérarchique du cerveau et la nécessité d'une conception hiérarchique en vision par ordinateur [8].

Les systèmes de vision par ordinateur tirent parti de structures hiérarchiques pour identifier des objets dans un monde structuré hiérarchiquement, dont objets peuvent naturellement être divisés en parties et sous-parties, caractéristiques complexes et caractéristiques simples [8]. De même, un système de représentation de comportements bénéficierait lui aussi d'une structure hiérarchique pour décrire des comportements complexes qui peuvent être naturellement divisés en parties et sous-parties, à la manière d'un algorithme.

Dans cet article, nous visons à fournir une représentation des connaissances pour les agents artificiels apprenant d'un instructeur à travers un Curriculum, ou cursus, soigneusement conçu. Nous définissons un Curriculum comme un ensemble de tâches indépendantes à accomplir, chaque tâche ayant un environnement, un but et des mécanismes de récompense ou de motivation. Les différentes tâches d'un Curriculum sont ordonnées en fonction de leur complexité croissante, ces dernières pouvant dépendre des

connaissances acquises lors des tâches précédentes. Cet ensemble de tâches couvrira divers aspects d'un comportement désiré, et leur maîtrise conduira à la maîtrise de ce comportement. Dans la perspective d'un développement continu, nous attachons une importance cruciale au fait que le comportement final actuel souhaité sera un élément pour la création future d'un, voir de plusieurs, comportements plus complexes, dont le but ne peut être prévu. En se basant sur des principes de conception modulaire, nous proposons l'architecture Modular Influence Network Design (MIND), une architecture adaptée à l'apprentissage progressif et par Curriculum. Cette architecture reflète les propriétés modulaires du Curriculum dans la représentation des connaissances en créant des modules qui peuvent encapsuler de nombreuses formes de structures d'apprentissage (réseaux neuronaux, approximateur de fonctions, etc.) et permet à chaque module de se coordonner avec les autres pour accomplir une tâche complexe. Cette architecture permettra d'accumuler continuellement de nouvelles compétences et d'utiliser la coordination des compétences antérieures pour maîtriser rapidement de nouvelles tâches d'une complexité croissante.

Après un bref état de l'art à la section 2 nous présenterons l'architecture MIND dans la section 3, puis nous décrirons dans la section 4 le protocole et les résultats des expériences que nous avons menées afin d'évaluer l'architecture. Avant de conclure, nous discuterons des limites et perspectives de MIND dans la section 5.

2 Contexte

2.1 Ontologie

Une *tâche* est un objectif à accomplir pour un agent situé dans un environnement. Si la tâche est destinée à entraîner un agent, elle doit comprendre un système de motivation, un signal de feedback ou une fonction de fitness.

Un *Curriculum*, ou cursus, est une série de tâches d'entraînement, ou leçons, indépendantes et d'une complexité croissante, couvrant les différents aspects d'un comportement souhaité.

Un *comportement* est l'expression d'une compétence, d'une capacité, dont le but est d'accomplir une action en rapport avec la tâche.

Un *skill* ou capacité, est un élément mémorisé, acquis par expérience ou entraînement, qui s'exprime sous forme d'un comportement.

Un *Base skill* ou skill sensori-moteur, est un skill qui associe directement les capteurs avec les actionneurs, et représente le niveau le plus bas de décision, les réflexes. Le fait qu'un base skill contrôle directement les actionneurs le met en concurrence avec les autres base skills souhaitant contrôler les mêmes actionneurs.

Un *Complex skill*, réalise la coordination de skills, ou délégation à d'autres skills, dans le but de représenter un comportement plus complexe que ses sous-skills.

2.2 Apprentissage par Curriculum et Architectures

L'apprentissage par Curriculum [2] est une méthode d'apprentissage progressive étudiée à ce jour dans le domaine du machine learning comme moyen d'accélérer l'apprentissage ou de résoudre des problèmes d'apprentissage qui sont autrement impossibles. L'apprentissage nécessite un Feedback, soit de l'environnement, soit d'une entité de supervision, mais lorsque la tâche et l'environnement d'apprentissage sont trop complexes, le signal de Feedback peut se révéler trop complexe pour permettre l'apprentissage. Par exemple, considérons le cas de l'accumulation de différentes sources de récompense pour différentes actions, où il ne serait pas possible de déterminer quelle action devrait être récompensée.

L'apprentissage par Curriculum subdivise une tâche d'apprentissage en sous-tâches différentes mais complémentaires (Source tasks) à apprendre séparément, et a été appliqué avec succès aux problèmes de robotique et de jeu vidéo [10]. Ici, la compétence est mémorisée par un approximateur de fonction unique grâce à l'apprentissage par transfert des différentes tâches sources [12].

L'apprentissage par Curriculum a également été appliqué dans un contexte plus abstrait pour enseigner à un réseau neuronal à réaliser l'approximation d'une fonction [7]. Dans ce travail, l'idée est d'entrainer le réseau sur une fonction cible simplifiée, puis de faire évoluer cette fonction cible jusqu'à ce qu'elle corresponde à la fonction désirée. Bien qu'il s'agisse d'une méthode d'apprentissage progressif intéressante, l'utilisation du terme Curriculum est discutable. Cette méthode d'apprentissage ajoute simplement plus de complexité à la même tâche d'apprentissage au lieu d'être une collection de tâches d'apprentissage complémentaires. Dans ce travail, la connaissance est également représentée comme un module unique, le réseau neuronal.

Bien que ces méthodes permettent d'apprendre par des moyens progressifs, leur portée est limitée à la tâche spécifique à accomplir. Ils ne couvrent pas l'apprentissage continu, ouvert et cumulatif d'une variété de tâches, un défi au coeur de la robotique dévelopementale qui se caractérise par l'acquisition progressive de

compétences et de connaissances [11].

Afin de pouvoir assimiler le Curriculum sous forme de modules de compétence indépendants l'architecture utilisée doit pouvoir coordonner des modules dont les sorties sont concurrentes.

Des travaux antérieurs dans le domaine des systèmes de commande de robots tels que l'architecture Subsomption [3] offrent des solutions pour alterner l'utilisation de différentes compétences selon le contexte. D'autres, comme AuRA [1] ou Sat-Alt [16], utilisent un mécanisme de composition vectorielle, permettant de choisir entre alterner ou combiner les actions de compétences distinctes. L'échelle de ces architectures reste limitée et elles ne sont pas destinées à des hiérarchies profondes.

2.3 Travaux connexes

Les travaux qui nous inspirent le plus, et que nous espérons développer, sont les techniques de Robot Shaping de Dorigo et Colombetti [4][5]. Ici, les comportements s'apprennent par Curriculum et sont représentés comme des compétences indépendantes. Ces compétences sont ensuite combinées pour obtenir des comportements de plus haut niveau. Plusieurs architectures sont discutées, des hiérarchies monolithiques aux hiérarchies multi-niveaux. Des méthodes d'apprentissage et de récompense adaptées aux agents artificiels, pertinentes dans le cadre de notre travail, sont également proposées. Nous présentons ici quelques points ou les Hiérarchies de Robot Shaping (*RSH*) et MIND diffèrent et expliquons comment et pourquoi MIND représente une amélioration par rapport aux *RSH*.

En *RSH*, les compétences de niveau inférieur sont les seules à avoir accès aux données des capteurs et à envoyer des demandes d'intervention aux compétences de niveau supérieur (coordinateurs). En se basant uniquement sur ces demandes, la compétence de niveau supérieur choisit quelles sous-compétences doivent être coordonnées et comment. Dans MIND, les compétences de niveau supérieur (Complex skills) basent leur décision de coordination sur l'environnement et le contexte en accédant directement aux données des capteurs, y compris celles des capteurs qui ne sont pas utilisés par les sous-compétences. Nous pensons qu'il s'agit là d'un concept important, car lors d'une prise de décision, l'information sur le *pourquoi*, le *quand* et le *si* sont souvent occultés pas le *comment*, ce qui conduit à la même réponse lorsque des détails subtils dans le contexte exigent une approche totalement différente.

Dans MIND, la sortie de toutes les compétences(Skills) est un vecteur de nombres réels entre 0 et 1 au lieu des chaînes binaires de *RSH*. Bien que cela ait un coût computationel élevé, cela permet une certaine finesse dans les commandes de sortie. La méthode d'Influence dépend également de cette sortie en nombre réel pour servir un rôle similaire à celui des opérateurs de combinaison du coordinateur dans la méthode *RSH*. L'utilisation de la méthode d'Influence, à la base de MIND, élimine le besoin de définir un ensemble fixe d'opérateurs de combinaison et permet à la place d'apprendre l'opération de combinaison.

L'utilisation d'un seul type de sortie pour toutes les commandes (influence, information capteur ou commande moteur) permet toute combinaison entre les éléments de l'architecture. *RSH* fait une distinction claire entre les compétences de base (dont les sorties sont des commandes de moteur) et les coordinateurs (dont les sorties sont des commandes de coordination). Dans MIND, nous faisons également la différence entre le Complex skill de plus haut niveau, les Complex skill subordonnés et les Base skill, mais cette distinction n'est que conceptuelle. Il n'y a pas de limite stricte qui empêcherait un Complex skill d'influencer plusieurs Base skills et en même temps d'envoyer des commandes motrices à plusieurs actionneurs.

3 Modular Influence Network Design

MIND (Modular Influence Network Design) est une hiérarchie de modules capables de coordonner des comportements (*Skills*) distincts pour accomplir une tâche complexe. Grâce à cette modularité nous souhaitons obtenir les propriétés suivantes :

1. **Encapsulation** : les comportements génériques seront encapsulés et combinés avec d'autres. On conserve la possibilité d'être la source de plusieurs nouveaux comportements, accélérant ainsi l'apprentissage de tâches différentes ayant une base commune. Cette propriété permet aussi d'identifier et de modifier le comportement localement.
2. **Méthode générique** : MIND a peu de contraintes sur la méthode de décision utilisée par chaque module, réseaux neuronaux et procédures de programmation peuvent être coordonnés par le mécanisme d'influence.
3. **Flexibilité** : les modules de comportement peuvent être remplacés soit par un nouveau module, soit par une hiérarchie de modules favorisant une évolution constante du système. Grâce à son mécanisme de

coordination générique, MIND facilite l'ajout de nouveaux capteurs et actionneurs dans un système déjà existant sans perdre les comportements acquis précédemment.

3.1 Base skill, complex skill, et influence

Considérons un agent dont les informations sensorielles et les commandes motrices sont représentées comme un vecteur de réels, normalisés entre 0 et 1. On peut créer un module qui encapsule une fonction $f(x)$ qui prend en entrée le vecteur $V_I = [I_1, I_2, ..., I_n]$ et fournit en sortie le vecteur $V_O = [O_1, O_2, ..., O_2, ..., O_m]$. Cette fonction peut être implémentée comme une procédure de programmation, ou peut être un approximateur de fonction, un réseau neuronal, ou tout autre type de fonction qui associe deux vecteurs de nombres réels. Nous appellerons un tel module un *skill*, et le module dont le vecteur de sortie est utilisé directement comme commande moteur un *base skill*.

$$V_O = f(V_I) \quad (1)$$

Il est possible de créer un unique base skill qui utilise toutes les entrées sensorielles et toutes les sorties motrices de l'agent pour apprendre à accomplir une tâche, même relativement complexe, chaque leçon du Curriculum étant mémorisée dans la même structure unique. Ce skill est donc la somme de toutes les différentes expériences, sans qu'il soit possible de différencier ce qui a été enseigné.

Au lieu d'accomplir une tâche complexe au moyen d'un seul skill, nous accomplirons de nombreuses sous-tâches, potentiellement concurrentes, avec des base skills distincts. Chaque base skill n'associera que les entrées et sorties nécessaires à l'exécution de la tâche qui lui est associée. Pour accomplir la tâche complexe, nous allons ensuite créer un *complex skill* qui va coordonner plusieurs compétences de base, que nous appelons ses *subskills* (Fig.1). Un *complex skill* coordonne ses *subskills* en envoyant un signal appelé *influence* qui détermine le poids (influence) qu'un subskill aura sur l'action résultante, ce qui revient à déléguer à une ou plusieurs sous-capacités la résolution de la tâche courante.

Un Complex skill, comme n'importe quel skill, encapsule une fonction qui prend en entrée un vecteur de valeurs issues des capteurs V_I et sort un vecteur de nombres réels V_O, dont la sortie est dirigée vers ses subskills. Ce vecteur de sortie est appelé le vecteur d'influence V_{Infl} = $[Infl_1, Infl_2, ..., Infl_m]$, et ses éléments $Infl_x$ sont appelés *influences*.
Un complex skill peut avoir d'autres complex skills comme subskills, créant des hiérarchies

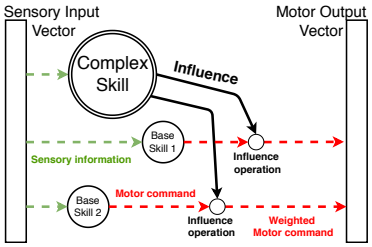

Fig. 1 – Un *complex skill* influençant deux *base skills*

de skills. Au sommet de la hiérarchie se trouve le *Master skill* dont la seule particularité est de recevoir une influence constante de 1.0, une impulsion permettant de lancer le mécanisme de calcul [15]. Cette hiérarchie de skills forme

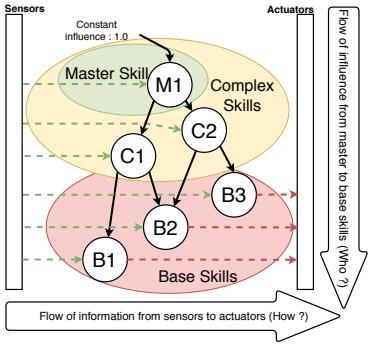

Fig. 2 – Une hiérarchie de skills, le *master skill* influence des *complex skills* qui à leur tour influencent des *base skills*

un graphe orienté acyclique (Fig.2). L'influence s'exerce le long d'un axe vertical, du master skill jusqu'aux base skills, et détermine qui est responsable de l'action résultante. L'information des capteurs est disponible pour tous les skills de la hiérarchie et les commandes des moteurs sont envoyées par les base skills vers les actionneurs formant un flux d'information horizontal, ce flux détermine comment l'action résultante va être exécutée.

3.2 Utilisation de l'influence pour déterminer les commandes moteurs

A partir du master skill, chaque complex skill calcule son vecteur de sortie V_O et multiplie chaque élément par la somme des influences reçues, formant le vecteur d'influence V_{Infl}. Le skill envoie alors chaque élément $Infl$ de V_{Infl} au subskill correspondant (Fig.3).

$$V_{Infl} = V_O * \sum_{x=1}^{n} Infl_x \quad (2)$$

avec V_{Infl} le vecteur d'influence vers les subskills, $V_O = f(V_I)$ le vecteur de sortie de la fonction interne du skill, et $\sum_{x=1}^{n} Infl_x$ la somme de toutes les influences reçues par le skill (également noté $\Sigma Infl$).

Un base skill, comme un complex skill, calcule

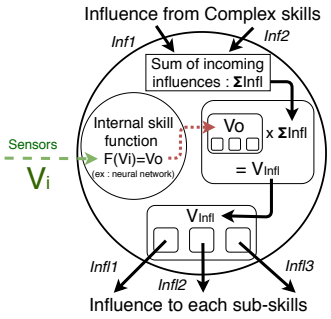

Fig. 3 – Architecture interne d'un skill

son vecteur de sortie et multiplie chaque élément par la somme des influences qu'il a reçues (Eq.2), formant le vecteur de commandes moteur $V_{Com} = [Com_1, Com_2, ..., Com_m]$. Le base skill envoie alors chaque élément Com_x du vecteur de commandes moteur au module moteur correspondant, ainsi que la somme des influences reçues par lui même $\Sigma Infl$.

Chaque module moteur calcule ensuite la commande correspondant à son actionneur sous la forme d'une somme pondérée normalisée :

$$M = \frac{\sum_{x=1}^{n} Com_x}{\sum_{x=1}^{n} \Sigma Infl_x} \quad (3)$$

avec M la commande de moteur résultante, x l'index de la compétence de base qui envoie une commande de moteur, Com_x la commande de moteur pondérée pour ce module moteur depuis le base skill x, $\Sigma Infl_x$ la somme des influences du base skill x.

4 Expérimentation

Pour évaluer l'architecture proposée, nous avons réalisé une série d'expériences dans un simulateur avec un agent réactif, un robot, utilisant un algorithme génétique et des réseaux neuronaux. La première expérience consiste à apprendre la tâche d'aller chercher un objet et de le ramener dans une zone spécifique. Nous avons ensuite modifié la tâche pour inclure une contrainte de consommation d'énergie.

4.1 Le robot

Le robot utilisé dans notre simulation est composé de deux moteurs, un pour chaque roue, et d'une pince pour saisir l'objet cible. Il dispose également de 18 capteurs, détecteurs d'obstacles, orientation de la cible, indicateur de charge de batterie. Le logiciel du robot, au moyen de ce que nous appelons un module capteur, est capable de calculer la dérivée de n'importe quelle entrée de capteur et de l'utiliser comme un capteur virtuel. Il peut également générer une onde sinusoïdale qui peut être utilisée à la manière d'un capteur (pour, par exemple, résoudre des situations de deadlock réactif).

La commande du moteur, transmise par le module moteur à son actionneur sous la forme d'un nombre réel compris entre 0 et 1, est interprétée comme un pourcentage de la capacité de l'actionneur (soit puissance, vitesse, position angulaire, etc.). Dans le cas des roues, 1 correspond à la pleine vitesse avant, 0 à la pleine vitesse arrière et 0,5 à l'arrêt. Dans le cas de la pince, le nombre correspond à son état (traitée comme binaire) : au-dessus de 0,5 fermé, au-dessous de 0,5 ouvert.

4.2 Fonctions internes des skills et algorithme d'apprentissage

Les skills que nous avons utilisées mettent en oeuvre un perceptron multicouche comme fonction interne, dont la couche d'entrée correspond au vecteur d'entrée de la compétence et la couche de sortie à son vecteur de sortie. Selon la compétence, son réseau neuronal utilisera 2 ou 3 couches cachées de N_{HIDDEN} neurones (Eq.4).

$$N_{HIDDEN} = Max(N_{Vi}, N_{Vo}) + 2 \quad (4)$$

N_{Vi} le nombre de neurones de la couche d'entrée et N_{Vo} le nombre de neurones de la couche de sortie

La fonction de transfert des couches du réseau neuronal est sigmoïde, à l'exception de la dernière couche qui utilise une fonction de transfert linéaire qui est fixée entre 0 et 1.

Cette configuration du perceptron est suffisamment générique pour couvrir la plupart des types de compétences, au prix d'un coût supplémentaire pour l'apprentissage génétique.

Contrairement à une approche non hiérarchique qui utilise une compétence unique avec un réseau neuronal unique qui relie tous les capteurs et actionneurs, dans l'approche hiérarchique, chaque compétence a son propre réseau neuronal utilisant uniquement les actionneurs, capteurs ou sous-capacités appropriés.

Pour apprendre les tâches, nous avons utilisé un algorithme génétique simple. Les poids des connexions du réseau neuronal sont ordonnés dans un vecteur, qui correspond au génome de l'individu du point de vue de l'algorithme génétique. Le génome sera croisé et muté en fonction du résultat de la fonction de fitness (d'adaptation) que l'environnement fournit en retour (Fig.4). Nous avons choisi d'utiliser un algorithme génétique pour sa simplicité, ses propriétés exploratoires et sa bonne performance avec récompenses différées. Par nature, il n'est pas nécessaire de lui fournir une description de

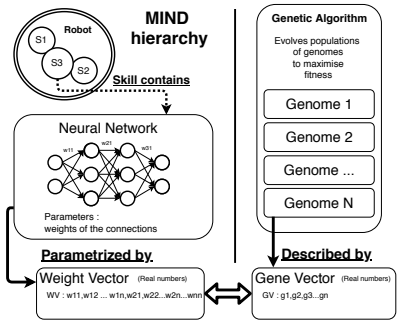

Fig. 4 – Relation entre un skill et l'algorithme génétique

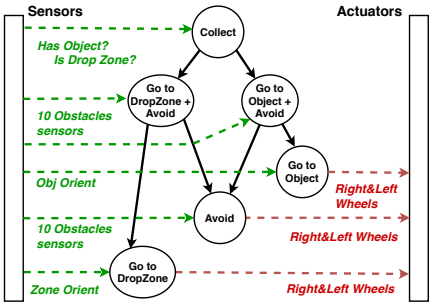

Fig. 5 – La hiérarchie Collect, relation entre les skills, les capteurs et actionneurs

la façon d'atteindre un but, contrairement aux méthodes qui utilisent la rétropropagation qui nécessite un set de couples d'entrées-sorties, mais seulement un moyen de mesurer si la performance d'un individu est meilleure ou pire que celle des autres individus sur une période d'évaluation. Contrairement à d'autres algorithmes d'apprentissage par renforcement où le signal de récompense modifie le comportement, la qualité du comportement est jugée à la fin du cycle de vie de l'individu. Cela permet à l'individu d'obtenir des récompenses négatives s'il en résulte une récompense positive supérieure par la suite, contournant ainsi la question de la récompense différée.

L'un des inconvénients majeurs de l'algorithme génétique est son coût élevé en ressources, en particulier l'évaluation de la fonction de fitness qui consiste à observer le comportement de l'individu, c'est-à-dire un agent, dans un environnement simulé pendant une période suffisamment longue. Toutefois, comme l'évaluation de chaque agent est totalement indépendante, elle peut être exécutée en parallèle. Ceci permet l'utilisation de solutions de calcul haute performance (HPC, mésocentres).

4.3 Scénarios

Les comportements appris présentés dans les scénarios suivants se basent sur la hiérarchie initiale de collecte présentée dans le scénario 1. Afin d'établir la fiabilité de la méthode en dépit de de la nature stochastique des algorithmes génétiques, chaque comportement du scénario 1 à été appris en 10 tentatives indépendantes. Les résultats numériques dépendants des fonctions de récompenses utilisées donnent une indication sur l'évolution et peuvent être utilisés a titre de comparaison, toutefois l'évaluation et l'interprétation d'un comportement complexe repose sur l'observation. Les vidéos des résultats sont disponibles à l'adresse suivante :

`www.lirmm.fr/~suro/Works.html`

Scenario 1 : Collect. Dans ce scénario, l'objectif est d'apprendre au robot une tâche de ramassage qui consiste à ramasser un objet et à le ramener dans une zone de dépôt sans entrer en collision avec des obstacles.

Nous voulons démontrer ici que l'architecture MIND est capable d'organiser des comportements simples en comportements complexes et qu'elle est capable de le faire de manière fiable même en utilisant un processus d'apprentissage génétique simple.

Nous avons choisi de diviser la tâche complexe en un Curriculum de cinq sous-tâches, dont trois sont des tâches de base : atteindre l'objet dans un environnement vide (GoToObject), atteindre la zone de dépôt dans un environnement vide (GoToDropZone), et éviter les collisions en se déplaçant dans un environnement avec des obstacles (Avoid). Sur ces tâches de base, nous construisons deux tâches complexes de niveau supérieur : atteindre l'objet en évitant les collisions dans un environnement avec des obstacles (GoToObject+Avoid), et atteindre la zone de dépôt en évitant les collisions dans un environnement avec des obstacles (GoToDropZone+Avoid).

Pour créer une hiérarchie initiale, il existe de nombreuses façons d'organiser les différentes sous-compétences, qui sont toutes valides, du moment qu'elles soient capables d'assimiler le Curriculum.

Apprendre la tâche Collect est un défi intéressant. Nous pouvons voir dans ses sous-tâches que le fait de se rendre à l'objet et de se rendre à la zone de dépôt utilisent les fonctions motrices d'une manière complètement opposée et devra être utilisée de façon séquentielle et exclusive. Au contraire, la tâche Avoid serait mieux utilisée comme une composition de vecteurs, en modifiant la trajectoire définie pour éviter un obstacle en douceur.

Une tentative sur dix des skills GoToObject et GoToDropZone n'atteignent pas le résultat optimal dans le nombre de générations données (Fig.6). Il est intéressant de noter que les deux

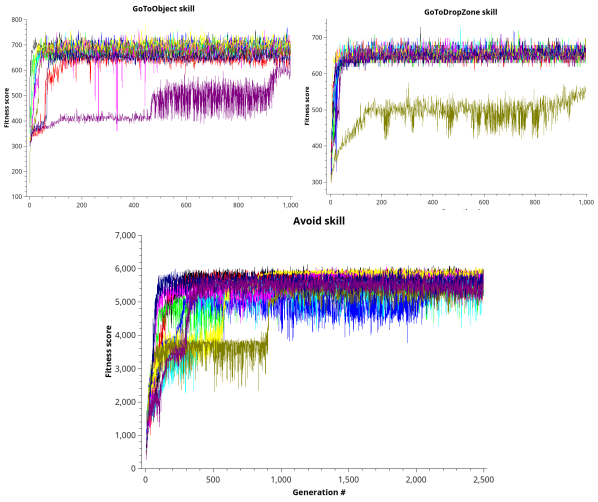

Fig. 6 – Base skills, 10 tentatives distinctes. GoToObject et GoToDropZone : 1000 générations. Avoid : 2500 générations.

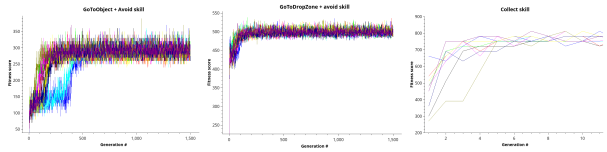

Fig. 7 – Complex skills GoToObject + Avoid, GoToDropZone + Avoid et Master skill Collect : 10 tentatives distinctes, 1500 générations.

tentatives échouées continuent de s'améliorer à chaque génération, ce qui est positif dans le contexte de notre travail visant le développement continu d'agents artificiels. La complexité des tâches peut être difficile à évaluer, mais il est facile de comprendre que le réseau neuronal affecté au skill GoToDropZone, qui utilise deux entrées, aura beaucoup moins de connexions, et donc moins de poids à ajuster, que le réseau neuronal affecté au skill Avoid, qui utilise plus de 10 entrées. Les graphiques de la figure 6 montrent comment cette différence de complexité affecte l'apprentissage. La plupart des tentatives des skills GoToDropZone et GoToObject atteignent leur valeur maximale en 100 générations, alors que la plupart des tentatives du skill Avoid continuent à évoluer après 500 générations. En se basant sur les meilleures base skills, nous apprenons les complex skills GoToObject+Avoid et GoToDropZone+Avoid. Le skill GoToObject+Avoid semble un peu plus difficile à apprendre que GoToDropZone+Avoid, ce qui s'explique par le fait que l'objet est plus petit que la dropZone et nécessite plus de précision (la "pince" doit être alignée avec l'objet pour le saisir). L'apprentissage des deux skills réussit pour chacune de leurs 10 tentatives respectives, dans le nombre de générations données.

Enfin, le master skill Collect, ayant un réseau très simple à entraîner, réussit son apprentissage pour chacune de ses 10 tentatives en moins de 10 générations (Fig.7).

Scenario 2 : apprendre avec des subskills sub-optimaux, rééducation et apprentissage dans un contexte plus général. Après que le scénario 1 ait réussi à établir une hiérarchie capable d'accomplir la tâche Collect, nous avons observé que le skill Avoid (évitement) était la cause principale d'échec pour l'agent et une perte de performance pour la hiérarchie. En nous inspirant de la citation suivante, nous avons expérimenté avec une autre stratégie d'apprentissage.

> *Les architectures hiérarchiques sont particulièrement sensibles à la stratégie de construction (Shaping Policy) ; en effet, il semble raisonnable que les modules de coordination soient façonnés après que les modules inférieurs aient appris à produire les comportements simples. Les expériences (...) montrent qu'en fait, on obtient de bons résultats en formant les modules les plus bas, puis en les "gelant" pour ensuite en former les coordinateurs, et enfin en laissant tous les éléments libres de continuer à apprendre ensemble.*
>
> *Dorigo et Colombetti*[4]

Dans un premier temps, le processus d'apprentissage génétique utilisait 1000 générations pour chaque skill. Le skill Collect résultant était bien capable d'accomplir la tâche de collection, mais échouait encore régulièrement à cause de collisions. Il n'est pas surprenant, du fait de son plus grand réseau neuronal qui coordonne plus de 10 capteurs, que le skill Avoid ait besoin de plus de ressources que les autres skills aux réseaux plus petits pour être entraîné correctement.

Nous avons recommencé l'entrainement Avoid avec 2500 générations ainsi que toutes les compétences de niveau supérieur. Avec 2500 générations, le score de fitness final du skill Avoid a augmenté de 10% par rapport au skill Avoid entraînée avec 1000 générations. Le skill Collect basé sur le nouveau skill Avoid a vu son score final augmenter de 14%, ce qui indique fortement que le skill Avoid est le facteur limitant dans la performance globale de la hiérarchie.

Même en allouant plus de ressources au skill Avoid (plus du double), nous n'avons pas été en mesure d'améliorer significativement la performance globale. Cela nous amène à nous interroger sur la qualité de la tâche d'évitement dans le Curriculum.

Les choix faits dans les éléments de la fonction de récompense sont certainement la cause de la mauvaise qualité de l'apprentissage. La nature du processus d'évolution est d'exploiter toutes

les règles mise en place pour obtenir le meilleur score, sans soucis de l'esprit de ces règles et des raisons qui nous on conduit a les établir. Par exemple, au cours d'expériences préliminaires, la seule récompense pour le skill Avoid que nous ayons donné était une récompense négative que l'individu recevait lorsqu'il heurtait un obstacle, de sorte que l'individu ayant obtenu le meilleur score ne bougeait tout simplement pas. Chacune des règles ajoutées par la suite a été à son tour exploitée jusqu'à ce que nous trouvions un ensemble de règles qui nous a permis d'obtenir un comportement proche de ce que nous souhaitions. Ceci illustre le problème de la définition de la fonction de fitness, qui peut être aussi difficile que de simplement programmer le comportement que l'on souhaite.

Nous avions besoin d'un skill Avoid pour construire la hiérarchie, mais maintenant que chaque skill a son propre rôle défini dans la hiérarchie, au lieu d'essayer d'améliorer la qualité d'Avoid par lui-même, nous allons l'entrainer dans le contexte final de la tâche de collecte, en tant qu'élément de la hiérarchie.

Pour ce faire, nous avons utilisé la hiérarchie déjà entraînée du scénario 1 en utilisant le skill Collect comme master skill. Nous avons utilisé l'environnement final de collecte, et le signal de récompense (la fonction de fitness) provenait de l'accomplissement de la tâche de collecte. Dans ce contexte, nous avons recommencé l'entraînement du skill Avoid (en ignorant la fonction de fitness définie à la main pour la tâche Avoid en faveur de la fonction de fitness de la tâche Collect). En n'utilisant que 1500 générations pour entraîner Avoid avec cette méthode, le score de fitness final du skill Collect s'est amélioré de 30%.

Ce résultat suggère qu'il pourrait y avoir d'autres stratégies d'apprentissage à considérer lorsqu'on entraîne une hiérarchie au-delà d'un simple déroulement du Curriculum, des tâches de base aux tâches complexes. Revenir sur les tâches précédentes pour améliorer les subskills dans le contexte final, une fois que tout le Curriculum a été sommairement enseigné, peut donner de meilleurs résultats que d'essayer de maximiser chaque subskill avant de passer au suivant.

L'entraînement progressif initial de la hiérarchie est toujours d'une grande importance, même si chaque compétence n'a pas besoin d'être entraînée à la perfection, cette première étape du Curriculum est celle où chaque skill se spécialisera dans son rôle au sein de la hiérarchie. Au cours d'expériences précédentes non détaillées dans le présent article, nous avons tenté d'entraîner la hiérarchie de collecte

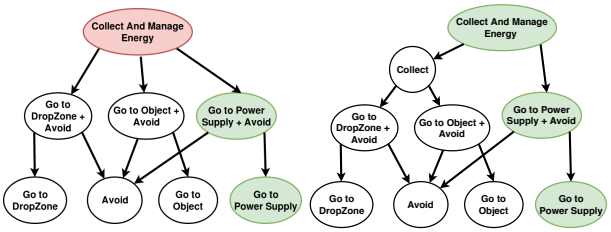

Fig. 8 – Collecte + gestion d'énergie, à gauche ré-entrainée, à droite encapsulée

en apprenant toutes les compétences en même temps. Dans ces conditions la plupart des branches de la hiérarchie restait inactives et un seul skill a tenté d'apprendre la tâche complète.

Scenario 3 : Collect et gestion d'énergie, la modularité de MIND. Nous souhaitons montrer que l'architecture MIND est adaptée au développement ouvert par l'ajout de nouvelles compétences qui étendent les capacités d'une hiérarchie déjà formée. Nous ajoutons à la compétence de collecte initiale la consommation d'énergie et la nécessité de recharger la batterie du robot. Un capteur qui indique le niveau de la batterie et des capteurs qui donnent des informations sur la direction de la source d'alimentation sont ajoutés au robot. Ce scénario ajoute au scénario précédent la tâche de base d'aller à la source d'alimentation pour se recharger et la tâche complexe d'aller à la source d'alimentation sans entrer en collision avec les obstacles, en réutilisant le skill Avoid.

Pour intégrer un nouveau skill dans une hiérarchie pré-existante, il y a deux façons possibles de procéder :

1. modifier le complex skill existant pour qu'il intègre le nouveau skill, et la ré-entraîner pour l'adapter (Variante ré-entraînée).
2. créer un nouveau complex skill qui coordonne le nouveau skill avec le complex skill existant. Le nouveau complex skill sera entraîné et le complex skill pré-existant ne sera pas modifié (Variante encapsulée).

L'évaluation pour ce scénario est la même que celle du scénario 1 avec l'ajout de la gestion de l'énergie. La batterie du robot se décharge de manière constante à moins que le robot ne se trouve dans le zone de recharge, auquel cas la batterie se recharge rapidement. L'évaluation est exécutée pendant une période donnée ou jusqu'à ce que le robot échoue (collision ou batterie vide). Chaque fois que le robot ramène l'objet dans la zone de dépôt, il marque un point et un nouvel objet est placé au hasard dans l'environnement.

Aucune indication de ce qui constitue un niveau de batterie faible n'a été donnée au robot, seulement le niveau de batterie actuel. Le robot

a déterminé par lui-même le niveau de batterie pour lequel il devait abandonner la tâche en cours et se diriger vers la source d'énergie. Cette décision se base sur sa propre expérience avec de la vitesse de décharge de sa batterie et la taille et complexité de l'environnement.

Les deux variantes de la hiérarchie (Fig.8), ré-entrainée et encapsulée, obtiennent des scores comparables montrant que les deux sont valables. Il faut noter que la variante encapsulée préserve le skill original Collect, la rendant disponible pour de futures combinaisons, ce qui est une des propriétés que nous souhaitions obtenir en proposant MIND (sec. 3).

5 Perspectives

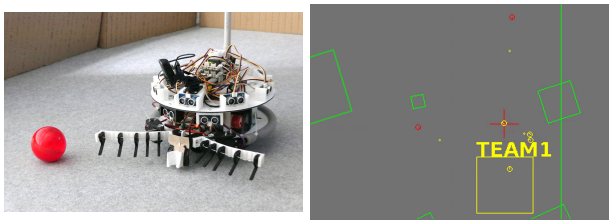

Fig. 9 – à gauche, Notre robot hi-tech, à droite, Capture de drapeau avec élimination

5.1 MIND sur un robot physique

Étant donné que la hiérarchie MIND apprise en simulation est entièrement réactive, nous n'avons pas anticipé de problème pour l'utiliser dans une application réelle. Nous avons conçu une tâche simple en utilisant les deux base skills GoToObject et Avoid, et le complex skill (dans ce cas aussi le master skill) GoToObject+Avoid.

La seule difficulté que nous avons rencontrée est le délai d'acquisition des capteurs et de réponse du moteur, très probablement à cause du matériel amateur utilisé. Néanmoins, en définissant des zones mortes pour les capteurs appropriées et une portée maximale, notre robot produit le comportement attendu, et ce en utilisant la hiérarchie et les compétences acquises dans une simulation qui n'a été calibrée d'aucune façon pour s'adapter au robot ou à l'environnement du monde réel.

Notre robot utilise un équipement sensori-moteur issu d'un kit de robotique amateur, afin de représenter le capteur d'orientation de l'objet cible nous avons utilisé un bras pan-tilt et une webcam pour tracker une balle lumineuse ("l'objet"). La position du bras panoramique est convertie en une information de capteur reproduisant le capteur d'orientation.

5.2 Applications multi-agents

Nous étudions actuellement l'apprentissage social en utilisant des hiérarchies MIND sur une tâche de capture de drapeau compétitive. Deux équipes de 10 agents doivent capturer le drapeau adverse et le ramener à leur base tout en protégeant le leur. Les agents ont à leur disposition, en plus de capteurs et d'actionneurs appropriés, un système d'émetteur et de récepteur de signaux. La technique d'apprentissage reste génétique, les agents de la même équipe utilisent une instance de la même hiérarchie de skills, mais le score de fitness est évalué collectivement. Les skills de bas niveau restent essentiellement sensori-moteurs, mais plus on monte dans la hiérarchie plus l'aspect social est pris en compte. Pour accomplir la tâche finale, gagner une partie de capture de drapeau, une organisation stratégique, et donc sociale, est nécessaire. Les agents sont capables de réagir à leur contexte respectif, leur position relative aux drapeaux, aux ennemis, on obtient une répartition des tâches, mais pour obtenir une répartition en rôles spécifiques, il est nécessaire de leur fournir une identité, dans notre cas une variable mémoire (constante dans ce cas). Pour le moment nous avons une preuve de concept programmée, mais nous espérons à terme être capable d'apprendre une répartition des rôles à la manière du niveau de batterie critique dans le scénario 3, par émergence.

5.3 Limites

Nos expériences ont montré la capacité de MIND à gérer de petites hiérarchies. Toutefois, on peut se demander si MIND reste applicable pour de grandes hiérarchies, en particulier pour des hiérarchies profondes où le transfert successif d'influence pourrait donner lieu à une forme de bruit sur les commandes motrices due à l'imprécision des réseaux de neurones. Si cette préoccupation est justifiée et qu'un entraînement plus fin des fonctions internes est coûteux et peu pratique, nous envisagerons des moyens de réduire le bruit dans les grands réseaux tels que des filtres ou des couches seuils.

6 Conclusion

Les résultats positifs obtenus sur la tâche Collecte sont encourageants, l'architecture MIND a su transformer le Curriculum en une hiérarchie de skills avec à la fois coordination et exclusion mutuelle de skills. MIND a été en mesure de traiter les tâches sensori-motrices de bas niveau ainsi que les opérations complexes d'influence des skills, sans fournir aucune information a priori sur la nature de la tâche.

Même si nous n'avons discuté que de skills utilisant des perceptrons multicouches comme fonctions interne en raison de notre focalisation sur le problème d'apprentissage, l'encapsulation de la fonction interne dans un skill peut donner à

toute fonction la capacité de s'intégrer dans une hiérarchie MIND. Du moment qu'il est possible d'obtenir des vecteurs d'entrées et de sorties sous forme de nombres réels, n'importe quel type de fonction peut être utilisé. Par exemple, la fonction contrôlant la "pince" du robot dans notre expérience est une procédure Java triviale qui a été intégrée à un skill.

D'autres auteurs[10][7] ont déjà établi les avantages de l'apprentissage progressif, en soumettant l'entité apprenante à une augmentation progressive de la complexité de la tâche ou en apprenant un Curriculum de tâches complémentaires visant à accomplir une tâche complexe. Les résultats positifs que nous avons présentés dans cet article en ré-entraînant les subskills de la hiérarchie semblent indiquer qu'il est intéressant de repasser plusieurs fois sur les leçons du Curriculum plutôt que d'essayer d'obtenir des résultats parfaits sur les tâches de base avant de passer aux plus complexes. Nous utilisons cette technique, en conjonction avec la capacité de certains algorithmes d'apprentissage de reprendre l'entraînement à partir d'un état sauvegardé, pour optimiser les hiérarchies MIND.

Au moment de la rédaction de cet article, plusieurs améliorations sont prévues pour la mise en oeuvre de l'architecture MIND, notamment l'utilisation de l'algorithme NEAT[17] pour remplacer notre algorithme génétique naïf. Non seulement cela améliorera grandement les performances, mais aussi éliminera complètement la nécessité de définir la topologie du réseau neuronal de la fonction interne des skills.

L'utilisation d'une méthode de communication standard entre tous les modules ouvre un certain nombre de possibilités, telles que l'utilisation de l'état précédent d'un actionneur de la même manière que nous utilisons les données du capteur. Il permet également de créer un certain nombre de modules de mémoire qui peuvent stocker la sortie d'un skill de la même manière qu'un actionneur reçoit une commande motrice, et peuvent fournir leur valeur à d'autres skills comme le ferait un module capteur. Grâce à ces modules mémoires se conformant au système d'influence MIND nous espérons permettre à l'agent d'évoluer au-delà du simple comportement réactif. Nous étudions actuellement un moyen d'enseigner une représentation de la connaissance avec la contrainte de ne pas franchir la barrière de l'intériorité de l'agent.

Notre prochaine étape vers la création d'un agent apprenant autonome est la création automatique de nouveaux skills. Pour une nouvelle leçon donnée par l'instructeur, quelles sont les entrées de capteurs pertinentes ? Quelles sorties d'actionneur ou subskills l'agent doit-il utiliser ? Quel langage et quelles stratégies dans la formulation d'une leçon conviendrait à l'agent tout en étant naturel et simple à définir pour l'instructeur humain ?

Références

[1] R. C. Arkin and T. Balch. Aura : Principles and practice in review. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2-3) :175–189, 1997.

[2] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 41–48. ACM, 2009.

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

[4] M. Dorigo and M. Colombetti. Robot shaping : Developing autonomous agents through learning. *Artificial intelligence*, 71(2) :321–370, 1994.

[5] M. Dorigo and M. Colombetti. *Robot shaping : an experiment in behavior engineering*. MIT press, 1998.

[6] D. J. Felleman and D. C. Van Essen. Distributed hierarchical processing in the primate cerebral cortex. *Cerebral cortex*, 1(1) :1–47, 1991.

[7] Ç. Gülçehre, M. Moczulski, F. Visin, and Y. Bengio. Mollifying networks. *CoRR*, abs/1608.04980, 2016.

[8] N. Kruger, P. Janssen, S. Kalkan, M. Lappe, A. Leonardis, J. Piater, A. J. Rodriguez-Sanchez, and L. Wiskott. Deep hierarchies in the primate visual cortex : What can we learn for computer vision ? *IEEE transactions on pattern analysis and machine intelligence*, 35(8) :1847–1871, 2013.

[9] M. Lungarella, G. Metta, R. Pfeifer, and G. Sandini. Developmental robotics : a survey. *Connection science*, 15(4) :151–190, 2003.

[10] S. Narvekar, J. Sinapov, M. Leonetti, and P. Stone. Source task creation for curriculum learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 566–574. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[11] P.-Y. Oudeyer. Developmental robotics. In *Encyclopedia of the Sciences of Learning*, pages 969–972. Springer, 2012.

[12] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10) :1345–1359, 2010.

[13] J. Piaget. *The construction of reality in the child.* Basic Books, New York, 1954.

[14] J. Piaget and E. Duckworth. Genetic epistemology. *American Behavioral Scientist*, 13(3) :459–480, 1970.

[15] A. Schopenhauer. *Die Welt als Wille und Vorstellung*. Brockhaus, Leipzig, 1819.

[16] O. Simonin and J. Ferber. Modeling self satisfaction and altruism to handle action selection and reactive cooperation. In *Proceedings of the 6th International Conference on the Simulation of Adaptive Behavior*, volume 2, pages 314–323, 2000.

[17] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2) :99–127, 2002.