Des AG[1] vers la GA[2]

Sélection darwinienne et systèmes multi-agents

Rapport de 2ème année de thèse

Samuel LANDAU
samuel.landau@lip6.fr

équipe MIRIAD / thème OASIS
LIP6

Directeur de thèse : Alexis DROGOUL

17 septembre 2001

1. algorithmes génétiques
2. génétique d'agents

Table des matières

Chapitre 1

Introduction

Ce rapport présente l'état d'avancement actuel des travaux de recherche dans le cadre de ma thèse. Celle-ci s'inscrit dans le champ des systèmes multi-agents (SMA), de la robotique collective et de la robotique évolutionniste.

1.1 Sujet

Notre objectif est de pouvoir générer automatiquement des comportements adaptatifs et structurés pour des groupes de robots mobiles opérant dans un environnement dynamique et non contrôlé.

Du point de vue des recherches en systèmes multi-agents [48, 42], les collectivités de robots mobiles présentent un intérêt particulier. La collectivité de robots mobiles est un paradigme de systèmes multi-agents dits *situés* [43], dont l'environnement, dynamique, n'est pas complètement spécifiable à l'avance, et dans lesquels le système est obligé de composer avec sa *corporéité* [37, 31]. En effet, pour une collectivité de robots mobiles, la dynamique de l'environnement est pour partie due au groupe de robots lui-même, et elle échappe au contrôle du système et des concepteurs. D'autre part la corporéité des robots et de la collectivité de robots est *implicite* et *physique*. Ces caractères des collectivités de robots mobiles ne sont pas contournables, à moins de simplifier drastiquement l'environnement ou les conditions de réalisation d'une éventuelle tâche à effectuer collectivement. Les contraintes font qu'il est impossible de prévoir avec exactitude quel sera l'état du système à un moment donné, et donc encore moins du comportement que chaque robot devra exprimer à cette occasion : c'est pourquoi les comportements générés doivent être adaptatifs.

Pour obtenir automatiquement ces comportements adaptatifs nous avons choisi d'utiliser des méthodes s'inspirant du modèle de la sélection darwinienne [36], à l'œuvre dans la nature. Ces méthodes sont très employées dans ce qu'il est convenu d'appeler depuis une dizaine d'années la « nouvelle IA », notamment en vie artificielle [81] ou en robotique [102]. Les méthodes de ce type pour l'instant les plus utilisées sont les algorithmes évolutionnistes [21, 64, 52, 113].

Nous avons également choisi une description structurée des comportements qui a l'intérêt, du point de vue des concepteurs, d'être modulaire, plus compréhensible et manipulable. Cela permet de faire des essais, d'altérer une solution existante ou de pouvoir soumettre à la machine une solution partiellement conçue (*bootstrap*). Notre choix s'est porté sur des graphes étiquetés, similaires à des ATN [1] [129], déjà utilisés pour décrire des comportements d'agent [57].

Certaines propriétés nous paraissent très importantes pour la conception automatique par évolution artificielle des architectures contrôlant les robots. Cependant, les méthodes évolutionnistes classiques ne les vérifient pas toutes simultanément. Nous avons réuni ces propriétés en un ensemble de principes de conception de comportements évoluables d'agents (l'Éthogénétique), puis

1. Augmented Transition Network

nous avons défini une nouvelle méthode évolutionniste qui les satisfait. Elle s'appuie sur un mécanisme d'interprétation à l'aide d'une pile. Nous l'avons appliquée à la conception des graphes que nous voulons utiliser.

Enfin, dans le but de manipuler des structures simples tout en conservant la possibilité de voir un comportement complexe exprimé par un robot, il est intéressant d'envisager de contrôler un robot non pas avec un seul agent, mais avec un système multi-agents. Le comportement d'un système multi-agent peut être beaucoup plus riche que celui des agents qui le composent [44, 42]. En termes de génie logiciel, ceci permettrait par exemple une approche componentielle du comportement de chaque robot : chaque agent serait expert pour une partie du comportement, et l'organisation du système coordonnerait et exploiterait en parallèle ces expertises.

Pour expérimenter cette nouvelle approche de comportements d'agents évolutifs et la comparer à d'autres, nous avons conçu le framework SFERES, qui permet d'appliquer des algorithmes évolutionnistes à des systèmes multi-agents. Puis nous avons implémenté une extension à la plate-forme appelée ATNoSFERES qui permet de faire évoluer nos graphes.

1.2 Plan

Dans une première partie (chapitre 2) nous présentons le contexte dans lequel se situent nos recherches, d'où émerge la nécessité de trouver un nouveau modèle de comportements structurés et adaptatifs, obtenus par évolution artificielle. Puis suivent l'établissement des principes d'évolution de comportements pour que ceux-ci exhibent les propriétés voulues (chapitre 3) et la description des modèles élaborés pour instancier et évaluer ces principes, ATNoSFERES et SFERES (chapitre 4). Enfin, nous présentons des expérimentations menées pour vérifier que le modèle satisfait bien aux principes posés et permet de répondre aux contraintes de la robotique collective dans le cadre de laquelle nous la considérons (chapitre 5) et concluons (chapitre 7) après une discussion des principes et modèles proposés (chapitre 6).

Chapitre 2

Contexte

La robotique est amenée à jouer un rôle croissant dans notre quotidien. Celui-ci est déjà en train d'être envahi par les objets « communiquants » (téléphones portables, assistants personnels et ordinateurs de poche, appareils éléctro-ménager « intelligents », ...) et le sera demain par des robots mobiles (aspirateurs autonomes, automobiles sans conducteur, ...) qui devront pouvoir s'adapter à la complexité de notre quotidien [1]. Là où ne se posaient que des problèmes informatiques et techniques liés à la distributions des objets, vont s'ajouter des contraintes et problèmes liés à la mobilité, au corps physique et à l'ancrage [60] des robots dans le monde sur lequel ils pourront agir.

Il apparaît à ce titre deux classes de problèmes (dues au nombre des robots impliqués). La première est l'immersion sociale de tels robots dans notre quotidien, dans un environnement d'humains [43, 108]. Les objets « communiquants » sont déjà plus ou moins bien acceptés (songeons à ces sonneries si agaçantes dans les lieux publics ...) et partie intégrante de notre quotidien, qu'en sera-t-il pour des robots, qui auront plus de capacités d'action sur le monde (à la limite autant que nous) ? La seconde est la socialité des robots (entre eux), ceux-ci pouvant désormais non seulement interagir de façon physique avec leur environnement et nous, mais aussi être amenés à interagir entre eux, de façon continue ou sporadique, et ce pour de multiples applications. Ces interactions peuvent être le fait d'une tâche à exécuter de façon collective (opérations de sauvetage, explorations de milieux difficilement accessibles ou dangereux, ...) ou juste le fait d'une rencontre à fortuite. Le projet MICRobES [46, 109], cadre dans lequel s'inscrit ma thèse, a vocation à étudier ces nouveaux problèmes.

Ces facteurs à prendre en compte, irréductibles, rendent la conception des comportements de groupes de robots en environnement réel encore plus complexe que la conception des comportements d'un seul. Cette complexité rend un contrôle central impossible. Il est donc nécessaire que les robots disposent d'un minimum d'autonomie et que dans ce but ils fassent montre de suffisamment d'adaptativité.

Nous présentons le contexte dans lequel se placent nos travaux en robotique autonome (section 2.1), en robotique collective (section 2.2) et en robotique évolutionniste (section 2.3). Cette synthèse va nous permettre d'expliquer notre choix des approches sélectionnistes pour générer des comportements. Ce choix résulte de ce que la conception directe par modules comportementaux s'avère trop complexe, et de ce que l'apprentissage par renforcement ne suffit pas pour réduire suffisamment la complexité de conception. Nous nous intéressons alors à l'évolution artificielle pour fabriquer automatiquement les architectures d'agents. Une revue des méthodes existantes nous permet de dégager des propriétés nécessaires à l'évolution d'architectures d'agents, et nous constatons qu'aucune des approches existantes n'exhibe simultanément toutes ces propriétés.

À l'aune de ces propriétés, nous érigeons des principes pour l'évolution de comportements d'agents au chapitre suivant (chapitre 3 page 14).

2.1 Robotique autonome en environnement réel

La conception de comportements de robots mobiles autonomes en environnement réel n'est pas chose aisée, et les approches de type ingénieur classiques (planification, etc.) ne suffisent pas pour obtenir des comportements robustes dans ces conditions. Des méthodes réactives plus efficaces ont vu le jour depuis une quinzaine d'années : notamment en effectuant des découpages comportementaux, puis plus récemment des approches évolutionnistes, plus souples et moins contraignantes. Ces dernières ont pour l'instant surtout été appliquées à des robots seuls, nous les évoquerons à la section 2.3.

2.1.1 Un système dynamique

Notre quotidien est un environnement dynamique, riche et bruité pour un robot [25]. À ce titre, programmer un robot mobile et autonome de façon à ce que son comportement soit robuste face aux situations imprévues s'avère difficile [12]. La cause principale des problèmes de conception est que le comportement des robots est une propriété émergente de leur interaction permanente avec l'environnement. Le système robot-environnement peut être décrit comme un système dynamique [1] car à tout moment l'état des perceptions du robot dépend de l'environnement et de ses propres actions passées.
Dès lors, comme l'environnement n'est pas entièrement prévisible et contrôlé, il est difficile de prévoir à l'avance quels comportements vont être exprimés connaissant les règles données au robot, et inversement il est aussi difficile de prédire quelle règles vont produire un comportement donné. Il est aussi intéressant de noter que de ce fait la complexité du comportement exprimé n'est pas liée à celle du robot : un exemple connu est celui des robots de Braitenberg [20], qui bien que simples exhibent des comportement relativement complexes.

2.1.2 Les approches délibératives

FIG. 2.1 – *Schéma général des approches délibératives. Le concepteur doit fournir au planificateur les symboles et les règles pour percevoir et agir.*

Les approches délibératives [101] s'avèrent donc dans ce cadre peu adaptées, car elles nécessitent des environnements structurés et connus, notamment pour avoir des temps de réponse acceptables [12]. La critique chronique adressée à ce type d'approche [2, 24, 1, ...] est l'absence d'ancrage des symboles (*symbol grounding*) manipulés par le système [60] : ceux-ci n'ont souvent aucun lien avec la réalité, ils n'ont pas d'ancrage sensori-moteur. Le système raisonne sur des symboles représentant des abstractions données par le concepteur. C'est un travail d'ingénierie important, et le résultat n'est pas robuste dans un environnement réel. Le schéma général de fonctionnement de tels systèmes est donné figure 2.1.

2.1.3 Les approches réactives et comportementales

D'inspiration biologique (éthologie, psychologie animale), les approches réactives et comportementales (*behavior-based robotics*) [26, 11, 23, 85, ...] se sont développée surtout suite aux travaux de R. Brooks [22]. Les principes, exposés dans [24] mettent l'accent sur la *situation* du robots dans le monde (on n'utilise plus de symboles et de descriptions abstraites), la *corporéité* des robots qui leur permet d'expérimenter directement le monde, et *l'émergence* du comportement comme fruit de l'interaction du robot avec le monde.

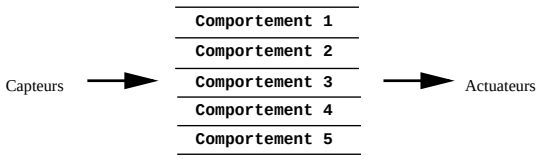

FIG. 2.2 – *Schéma général des approches comportementales (d'après [102]). Les modules comportementaux sont séparés et conçus par le concepteur.*

Ces approches ascendantes sont modulaires : on définit des modules comportementaux simples qui peuvent agir de façon coopérative [11] ou compétitive [22]. Le comportement global du robot émerge au travers des interactions entre ces comportements et l'environnement. L'environnement joue donc ici un rôle central.

La conception avec ce type d'approche se fait par essais et erreurs : le concepteur modifie les modules et augmente progressivement leur nombre, tout en continuant à tester le comportement global résultant dans l'environnement.

Cette approche de la conception de robots en environnement réel fonctionne mieux que la planification. Cependant, elle repose sur l'intuition du concepteur qui doit séparer et spécifier les modules, et requiert une attention soutenue du concepteur lors des expérimentations. De l'aveu même de ceux qui ont adopté ce schéma de décomposition comportementale, cette séparation est un problème crucial [24]. La séparation et la spécification des modules sont rendues difficiles par l'aspect émergent du comportement. Les interactions mêmes entre modules rendent difficile la conception incrémentale puisqu'elles ajoutent de l'incertitude sur le comportement actuel du robot dans son environnement.

Pour pallier à ces difficultés d'ajustement, des techniques d'apprentissage par renforcement ont été utilisées avec succès [67, 86, 25, 88, 112]. Cependant, pour ce faire, il faut au préalable décider quelles structures vont être organisées par l'apprentissage, et il faut aussi pouvoir appliquer la technique voulue (chacune a plus ou moins de prérequis). L'apprentissage par renforcement n'est pas aussi flexible que peuvent l'être les méthodes évolutionnistes pour concevoir des comportements de robots (*cf.* [102] chapitre 1). Ces dernières permettent avec un minimum d'hypothèses de construire automatiquement des architectures, résolvant du même coup le problème du découpage et de la conception des modules comportementaux. Nous aborderons les méthodes évolutionnistes dans la section 2.3.

2.2 Robotique collective

2.2.1 Un surcroît de complexité

Un groupe de robots peut permettre de résoudre des problèmes où un seul robot ne suffirait pas (soulever, déplacer à plusieurs, ...). La redondance peut par ailleurs conférer une plus grande robustesse au système multi-robots. Un certain nombre de tâches collectives ont déjà été étudiées [29, 47, 14, 90, 92, 16, 13], de la navigation en formation [17] à l'exploration collective, ou à la fusion de capteurs. Ces tâches ajoutent les problèmes de distribution (distribution physiques des robots, distribution du contrôle, distribution des perceptions et des actions) à ceux qui se posent déjà avec un robot seul. La complexité de conception tient aussi au fait que le comportement collectif est bien souvent plus lié aux interactions entre les robots qu'à leurs comportements individuels, et que leurs actions sont interdépendantes.

Passer d'un robot seul à un groupe de robots se traduit donc par un surcroît de complexité à tous les niveaux, en plus de ceux qui apparaissent avec le groupe (problématiques sociales). De fait, les comportements collectifs effectivement étudiés sont souvent des comportements « simples » (fourragement, poussée d'objets, manipulations coopératives, ...), ou tout du moins dans un

environnement suffisamment contrôlé pour rendre la tâche moins complexe.

Dans tous les cas, l'environnement est rarement réel : la lumière est soigneusement contrôlée si les robots utilisent des caméras, l'environnement électromagnétique s'ils ont des compas, etc. C'est d'autant plus vrai lorsque la tâche collective est complexe, par exemple jouer au football [45]. Lors de la RoboCup [1], les variations de chaque paramètre (puissance lumineuse, couleur de l'éclairage, taille des robots, forme du terrain, etc.) font l'objet d'âpres marchandages, et une équipe de robots peut ne plus fonctionner du tout suite à la faible variation d'un seul des paramètres ...

2.2.2 Apprentissage multi-robot

Pour pallier à une partie de la complexité de conception, l'apprentissage a aussi été utilisé dans le cas multi-robots [91, 104, 123, 94, 121, 93], mais cela n'a pas permis de s'intéresser à des comportements collectifs plus complexes, ni d'opérer indifféremment en environnement réel. Aux problèmes rencontrés avec un robot seul (*cf.* section 2.1) s'ajoutent des problèmes d'apprentissage multi-agents, notamment comment distribuer le renforcement sur l'ensemble des robots (*credit assignment*). À nouveau, les méthodes évolutionnistes que nous abordons ci-après vont apporter une réponse plus flexible à ces problèmes de conception.

2.3 Robotique évolutionniste

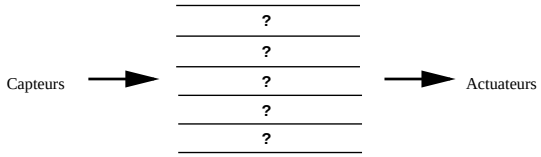

FIG. 2.3 – *Schéma général des approches en robotique évolutionniste (d'après [102]). La séparation et la conception des modules comportementaux se fait automatiquement, sans intervention du concepteur.*

La robotique évolutionniste [96, 103, 61, 102] s'inspire elle aussi de modèles biologiques pour définir les architectures des robots. Le paradigme est celui de la sélection naturelle [36]. Cette méthode non supervisée est moins contraignante que les apprentissages par renforcement classiques car elle permet notamment de générer entièrement l'architecture du robot, pas seulement de la paramétrer (*cf.* section 2.3.4).

À noter que cette approche concerne pour l'instant surtout des robots seuls et qu'il existe encore peu d'applications de la robotique évolutionniste à des collectivités de robots, et encore moins pour des groupes de robots plongés dans un environnement réel. Des applications dans un environnement plus contrôlé existent par exemple pour la RoboCup [1] [3, 83], au nombre desquelles il faut aussi compter notre équipe pour le LIP6 en 1999 [76].

Une courte introduction au paradigme de la sélection naturelle (section 2.3.1), aux algorithmes évolutionnistes (section 2.3.2) puis aux algorithmes classiques (section 2.3.3) nous permettra d'expliquer en quoi cette solution pour la conception de comportements d'agents nous paraît la plus indiquée. Puis nous nous intéresserons plus particulièrement aux méthodes d'évolution de structures exécutables (section 2.3.4). Ces dernières sont utilisées comme architectures d'agents, en particulier en robotique évolutionniste.

Cette synthèse nous permettra de dégager les propriétés que nous pensons nécessaires pour faire évoluer des architectures d'agent (dans notre cas pour la robotique collective). Ces propriétés seront érigées en principes de l'Éthogénétique au chapitre suivant (section 3.1 page 14).

1. *cf.* `http://www.robocup.org`, [70, 69]

2.3.1 Le paradigme de la sélection naturelle

La sélection naturelle est un mécanisme de création d'ordre à partir du désordre, qui repose sur des variations aléatoire qu'une sélection va filtrer pour ne retenir que celles qui seront qualifiées alors de plus adaptées (à la pression sélective). Ce mécanisme de hasard-sélection, sans finalité, opère simultanément à plusieurs niveaux, de celui des constituants les plus intimes des cellules [75] à celui des organismes [36].

Ce mécanisme simple a inspiré les méthodes d'apprentissage par évolution artificielle. Comme les variations sont aléatoires, ces méthodes ne nécessitent pas d'introduire de finalité donc de connaissance sur le problème à résoudre dans les opérateurs de recherche. Cette propriété en fait une bonne alternative lorsqu'aucune méthode déterministe d'optimisation n'est utilisable ou connue. Pour accélérer la recherche, il demeure cependant possible d'informer l'algorithme en biaisant les opérateurs de variation, selon ce qu'on sait du problème, mais la généricité des opérateurs est alors perdue. Notons qu'il faut tout de même le plus souvent introduire de la connaissance dans le mécanisme de sélection – même si ce n'est pas toujours nécessaire. Ce peu d'hypothèses faites sur les populations de solutions à trouver a pour conséquence que les algorithmes évolutionnistes peuvent souvent être appliqués là où d'autres méthodes classiques d'apprentissage (recuit simulé, remontée de gradient, TD-learning...) sont inutilisables ou trop peu efficaces à cause de contraintes sur l'espace de recherche. La remontée de gradient, par exemple, impose l'utilisation de fonctions continûment dérivables.

Les aspects parallèles (plusieurs solutions sont recherchées de façon simultanée) et aléatoire (variations produites par les opérateurs) de la recherche permettent d'éviter les minima locaux du paysage de l'espace de recherche, par la combinaison de solutions éloignées ou par une variation suffisamment grande.

Cependant, la généricité et la souplesse d'utilisation ont parfois pour conséquence le fait que la convergence n'est pas assurée. Si le temps de calcul est trop limité, on obtient des solutions satisfaisantes mais pas toujours optimales.

2.3.2 Principe des algorithmes évolutionnistes

L'appellation générique *algorithme évolutionniste* désigne des systèmes informatiques de résolution de problèmes [21, 64] qui s'inspirent des mécanismes adaptatifs de la sélection naturelle [36]. Différents algorithmes ont été proposés, les principaux ayant été conçus presque simultanément et indépendamment dans les années 1960 : les algorithmes génétiques, les stratégies évolutionnistes, la programmation évolutionniste et, plus récemment, la programmation génétique.

Le principe général des algorithmes évolutionnistes est de tester en parallèle différentes solutions potentielles, appelées par la suite *individus*, à un problème posé, puis de retenir les plus efficaces et, à partir de ces solutions, d'en générer de nouvelles en les combinant de façon à améliorer progressivement leurs performances. La base conceptuelle commune aux algorithmes évolutionnistes réside donc en la simulation de l'évolution de générations successives de codes génétiques ou *génomes*, qui composent les individus, via des processus de *sélection* et de *reproduction*. Une première génération d'individus est tirée au sort. Chaque individu est testé et une note - valeur sélective ou *fitness*[2] - lui est attribuée en conséquence. Les individus à forte valeur sélective auront une probabilité plus forte de se "reproduire", exploitant ainsi l'information disponible sur le degré d'adaptation de l'individu. Une seconde génération est créée en appliquant des opérateurs génétiques sur les structures des individus parents, comme des *recombinaisons* – échanges de matériel génétique entre individus – ou des *mutations* – transformations aléatoires d'un partie du génome. Chaque individu est de nouveau testé et le processus est reconduit de génération en génération jusqu'à l'obtention d'individus performants pour la tâche donnée. La figure 2.4 illustre le principe de fonctionnement d'un algorithme évolutionniste.

Bien que simpliste d'un point de vue biologique, ces algorithmes sont suffisamment performants pour fournir des mécanismes de recherche adaptative robustes et puissants.

2. fitness : mesure de l'adaptation d'un individu à son environnement ; mesure de performance de l'individu dans des problèmes d'optimisation.

```
t := 0
initialiser_population P(t)
évaluer P(t)
tant que non critère d'arrêt faire
        t := t + 1
        P'(t) := sélectionner_parents(P(t))
        croiser(P'(t))
        muter(P'(t))
        évaluer(P'(t))
        P(t+1) := sélectionner_survivants(P(t), P'(t))
```

FIG. 2.4 – *Principe de fonctionnement d'un algorithme évolutionniste*

L'utilisation de l'évolution artificielle comme méthode d'apprentissage multi-agent apporte de plus une réponse au problème du *credit assignment*. Ce problème se pose lors de l'apprentissages par plusieurs agents d'un système : lesquels faut-il récompenser ou punir, et dans quelles proportions, au vu de ce qu'a produit le système entier ? Une réponse possible avec l'évolution artificielle est de choisir un type d'individu de l'algorithme évolutionniste qui représente les caractéristiques du *groupe* d'agents que l'on veut faire évoluer [62], et non pas d'un agent pris individuellement – auquel cas il faudrait par exemple autant d'individus à évaluer qu'il y a de clones d'agents dans le système. Un individu de l'algorithme encode alors des caractéristiques de tous les agents du groupe à la fois, et ce qui est sélectionné est le comportement du groupe plutôt qu'une somme de comportements individuels. Le *credit assignment* est alors réglé de façon indirecte et implicite.

Pour plus de détails sur les algorithmes évolutionnistes et des comparaisons des mérites de chacune des approches, consulter par exemple [63, 18, 98, 125, 15].

2.3.3 Les principaux algorithmes

De nombreuses variantes d'algorithmes évolutionnistes existent. Elles différent principalement sur la stratégie employée pour sélectionner les individus à conserver, sur le codage des solutions ainsi que sur les opérateurs génétiques utilisés.

Les *algorithmes génétiques* [65, 39, 54, 66] utilisent un codage de type chaîne binaire, alors que les *stratégies évolutionnistes* [113, 114, 115] utilisent un vecteur de flottants. La *programmation évolutionniste* [52, 50] permet de faire évoluer des structures quelconques (machines à états finis, réseaux de neurones,etc.) et la *programmation génétique* [72, 99] construit directement des programmes informatiques sous forme d'arbres.

	algorithme génétique	stratégie évolutionniste	programmation évolutionniste	programmation génétique
individu	chaîne de bit	tableau	quelconque	arbre
sélection	probabiliste [a]	élitiste [b]	élitiste	probabiliste
croisement	oui	non	non	oui
mutation	oui	oui	oui	non

[a] la probabilité d'être sélectionné est fonction (croissante) de la fitness
[b] la sélection se fait sur la valeur de la fitness : les « meilleurs » sont choisis

FIG. 2.5 – *Caractéristiques des principaux algorithmes évolutionnistes*

La figure 2.5 récapitule les caractéristiques de chacun des principaux algorithmes évolutionnistes – dans leur version « orthodoxe ». Beaucoup d'échanges ont eu lieu entre ces techniques, par exemple l'ajout de paramètres dans le génome qui permettent de rendre plus adaptatifs les opérateurs de recherche au fil des générations (un taux de mutation, ...).

2.3.4 Évolution de structures

La construction automatique de structures [7] constitue l'un des courants les plus importants de l'informatique évolutionniste. Celles-ci peuvent être par exemple un circuit [73], une machine à états finis [52], un arbre représentant un programme [72], ou un réseau de neurones [130]. La plupart du temps, ce sont des structures *exécutables* [9] dont on évalue le comportement, via celui de l'agent ou de l'animat [127, 128, 96] qui s'en sert comme architecture.

Il est possible de distinguer deux façon de faire évoluer des structures, selon le codage utilisé. La représentation évoluée peut être soit utilisée directement (codage direct) soit indirectement (codage indirect). Nos donnerons des exemples. Ces deux façons de faire ont chacune des propriétés intéressants qui ne sont pas incompatibles, pourtant aucune méthode existante n'a toutes ces propriétés simultanément. Nous allons présenter les différentes approches d'évolution de structure pour mieux illustrer ces propriétés, qui nous serviront comme principes aux chapitres suivants pour la construction de nos propres méthodes d'évolution de structures (*cf.* section 3.2).

Codages directs

Les codages directs sont les premiers à avoir été utilisés. Deux catégories peuvent être distinguées. La première consiste en un codage de paramètres pour une structure existante. Elle permet de régler finement la structure évoluée, mais pas de la générer ex nihilo. La seconde manipule la représentation de la structure elle-même, sans faire de distinction entre la représentation qui est l'objet de manipulation génétiques et la structure qui est utilisée effectivement. Les structures sont alors complètement générées, mais le concepteur doit faire face à beaucoup de contraintes pour les opérateurs génétiques.

Codage de paramètres pour une structure existante Le réglage de paramètres pour structures existantes consiste en une optimisation d'un ensemble de paramètres [15], dont la position dans la représentation génétique et la signification pour la structure évoluées sont bien précises. Les algorithmes évolutionnistes les plus utilisés à cet effet sont les algorithmes génétiques et les stratégies évolutionnistes (*cf.* section 2.3.3). Le réglage de paramètres est beaucoup utilisé pour faire évoluer des réseaux de neurones [97, 126, 55, ...] (pour plus de détails consulter [130]). Cette méthode donne de bons résultats comparés à ceux d'autres méthodes. Mais comme la structure est donnée, l'avantage n'est pas décisif. Le réglage de paramètres manque de plasticité. Pour la suite nous nous référerons à ce défaut en disant que qu'un encodage qui déplore ce défaut manque de *pouvoir d'expression.*

Manipulation de la représentation de la structure elle-même L'évolution directe de structures manipule les structures directement lors des opérations génétiques et aussi lors de leur évaluation. Cette méthode est le fait de la programmation évolutionniste et de la programmation génétique (*cf.* section 2.3.3). Des structures très différentes ont été évoluées de cette façon : des machines à états finis [52], des réseaux de neurones [10, 4], des graphes [117], des arbres [72] ... Cette méthode présente l'avantage de pouvoir générer des structures ex nihilo. Cependant, la conception des opérateurs génétiques s'avère délicate. Les contraintes principales auxquelles ils faut faire face sont alors :

- la *syntaxe*, c'est-à-dire s'assurer de la consistance des structures produites. Une génération aléatoires, une mutation ou le croisement de structures doivent produire des structures valides.
- la *sémantique*, c'est-à-dire s'assurer que le comportement des structures des enfants produits par reproduction soit proche de celui des structures des parents. Ceci afin que la recherche qui s'appuie sur des variations aléatoires ne soit pas pour autant une marche au hasard. Cette contrainte est issue de la propriété de *continuité* entre le génotype et son évaluation (que celle-ci soit ou non directe). Les encodages qui assurent cette propriété de continuité sont optimaux au regard des algorithmes évolutionnistes.

- la *complétude*, c'est-à-dire s'assurer que l'on peut explorer tout l'espace de rechercher sans exclure des portions qui pourraient éventuellement contenir de meilleures solutions. Cette contrainte peut se réduire en pratique à ne pas trop biaiser les opérateurs de recherche, pour la même raison.

Ces trois problèmes ont été l'objet de beaucoup de travaux [72, 99, 5, 51, 8, ...]. Les difficultés reposent ici en partie sur la division du génotype en parties qui ont des propriétés différentes (inhérentes à la structure). Les modifications induites par les opérateurs génétiques doivent donc prendre en compte ces différents rôles tout en respectant au mieux les contraintes.

Codages indirects

Les codages indirects nécessitent un processus d'expression plus complexe pour produire puis évaluer la structure [7]. L'indirection a beaucoup d'avantages potentiels, dont surtout :

- la possibilité de créer la structure de façon *adaptative* : la structure peut ne pas être une interprétation linéaire de la représentation génétique. Cette possibilité permet de résoudre les problèmes syntaxiques évoqués précédemment (*cf.* section 2.3.4).
- la *compacité* du code génétique : la structure peut-être bien plus grande que sa représentation génétique.

Comme pour les codages directs, nous pouvons distinguer deux catégories principales de génération de structures. La première règle des paramètres, dont une partie sert à décrire la structure. Cette méthode a des défauts que nous détaillons ci-après. La seconde utilise une structure de description dont l'expression sert à construire la structure finale. C'est cette dernière qui est évaluée. Cette approche permet de générer n'importe quelle structure et de disposer de représentation génétiques très compactes, mais le concepteur doit faire face au même type de difficultés évoquées précédemment, avec un niveau d'indirection.

Réglage de paramètres de description de la structure L'utilisation de paramètres de *description* garde la qualité de pouvoir régler finement la structure évoluée (comme pour le codage direct). Elle y ajoute la possibilité de générer une structure ex nihilo. Cependant, même si la structure n'est pas donnée à l'avance, la façon dont sont positionnés les paramètres et leurs types sont très significatifs : la construction de la structure se fait selon un plan bien précis. Il apparaît un manque de flexibilité, puisqu'il faut déjà avoir un certain nombre de connaissances fixes sur la façon d'interpréter le génotype. Ces connaissances devront être prises en compte lors de la conception des opérateurs génétiques, notamment pour assurer la correction syntaxique, ce qui nous éloigne des variations aléatoires « aveugles » sur lesquelles nous voulons nous appuyer de façon exclusive (*cf.* section 2.3.1).

Structure de description pour générer la structure évaluée La seconde méthode, celle utilisant une structure de description à interpréter pour générer la structure à évaluer, constitue l'approche la plus sophistiquée pour faire évoluer des structures. Elle est souvent désignée sous le nom de *développement*, par analogie avec le modèle biologique du développement. Le développement a surtout été utilisé pour fabriquer des structures en réseau, par exemple des circuits électriques [73], ou des réseaux de neurones artificiels (*cf.* [130] section 3.2.2 pour une synthèse). Le modèle d'expression peut opérer par réécritures successives, par exemple des règles de réécritures grammaticales (L-systems [82]) dans [19], des règles de réécritures matricielles [68], ou il peut opérer en exécutant un programme sous forme d'arbre de développement. Ce dernier peut être contraint syntaxiquement [71] ou non [56, 84]. La structure manipulée génétiquement et celle qui est évaluée peuvent ainsi être de nature complètement différente. Les tailles des deux structures ne sont pas corrélées : ce mode d'expression indirect permet de générer des structures dont la taille est de plusieurs ordres de grandeur celle de la structure manipulée génétiquement. Enfin, les complexités des structures ne sont pas non plus corrélées : un génotype simple pourra être la source de structures complexes.

Cependant, la structure manipulée génétiquement doit toujours faire face aux trois contraintes exprimées précédemment, même si de façon indirecte cette fois-ci :

- son interprétation doit produire des structures valides (contrainte syntaxique),
- le comportement des structures obtenues par reproduction (ce qui implique de petites variations) doit être aussi proche que possible de celui des parents (contrainte sémantique, propriété de continuité)
- aucune portion de l'espace de recherche susceptible de contenir des solutions intéressantes ne devrait être exclue (contrainte de complétude).

La contrainte syntaxique est cette fois-ci simple à respecter, il suffit de songer à l'utilisation des structures grammaticales par exemple. En revanche respecter les contraintes sémantique et de complétude est plus complexe, justement à cause de l'indirection. Il est d'autant plus difficile de prévoir si les petites variations vont se traduire par de petits changements dans le comportement de la structure évaluée que la complexité et la taille de ces structures ne sont pas a priori corrélées. Il en va de même pour l'exploration de l'espace de recherche.

Chapitre 3

Principes

Les approches comportementales donnent de bons résultats pour la conception de comportements de robots, mais posent de gros problèmes de conceptions. Les approches évolutionnistes paraissent donc les mieux adaptées pour la conception de comportements de robots, puisqu'elles permettent de concevoir automatiquement des architectures, et de faire les découpages comportementaux de façon nécessaire pour le robot (et non pas avec le biais de l'observateur-concepteur).

Parmi les approches utilisées pour la conception de structures chacune a des avantages, mais aucune ne les présente simultanément. Certaines ont une propriété de *continuité*, propriété optimisant la recherche avec des algorithmes évolutionnistes, et d'autres ont un fort *pouvoir expressif*, permettant de réellement *construire* des structures à partir de rien.

D'autre part, un problème important identifié dans les approches existantes lors de la conception des opérateurs génétiques est de satisfaire aux trois contraintes de syntaxe, sémantique et complétude exprimées précédemment.

Nous allons dans un premier temps expliciter et ériger en principes pour des approches de comportements d'agents évoluables les propriétés de continuité et de pouvoir expressif. Ceci est l'objet de la section 3.1 dédiée à l'Éthogénétique. Ensuite, nous présentons un nouveau principe d'évolution de structures, supposé satisfaire aux principes de l'Éthogénétique. Il s'agit d'un encodage basé sur l'expression de gènes à l'aide d'une pile (section 3.2). Cette approche a de plus le mérite de permettre d'évacuer de façon simple et élégante les difficultés de conception des opérateurs génétiques.

Note : les concepts développés dans ce chapitre sont en partie le fruit d'un travail commun avec Sébastien Picault, thésard de l'équipe Miriad.

3.1 L'Éthogénétique

L'Éthogénétique (*Ethogenetics* [111]) suggère des principes généraux pour la conception de modèles d'expression génétique appliqués à la conception de comportements d'agent évolutifs. Le moyen retenu est de construire des structures décrivant les comportements d'agents à partir d'un substrat non signifiant pour se délivrer des difficultés rencontrées par les autres approches (*cf.* section 2.3.4). Les deux propriétés que nous avons précédemment dégagées des méthodes existantes pour faire évoluer des structures sont ici érigées en principes. Un modèle d'expression génétique, pour satisfaire aux conditions de l'Éthogénétique, doit exhiber une propriété de continuité, et avoir un (fort) pouvoir expressif.

3.1.1 Continuité

Nous utilisons des opérateurs de variations aléatoires et sans finalité. Ceux-ci opèrent généralement par des variations de petite ampleur sur la structure génétique. Les variations sont peu

susceptibles d'amener chacune des améliorations au comportement exprimé par l'agent. Par conséquent, ces variations de faible amplitude du génotype doivent se traduire, *le plus souvent*, par des changements de faible amplitude dans le comportement exprimé. Si cette contrainte n'est pas respectée, le processus sélectionniste devient une marche au hasard et l'on ne peut pas espérer obtenir des individus de plus en plus adaptés au fil des générations. Voici donc la continuité que nous recherchons définie plus précisément : la continuité est appliquée ici à la fonction qui exprime le génotype en structure décrivant le comportement et ce sur la quasi totalité du domaine des génotypes.

Avec cette propriété, le processus darwinien de hasard-sélection [75] peut générer de l'ordre à partir de désordre [41], sans finalité et sans information des opérateur de recherche. Le processus est auto-catalytique par *adaptation cumulative* des individus au fil des générations.

Du point de vue des algorithmes évolutionnistes, il a été montré que les fonctions d'expression du génotype (le « paysage de fitness ») qui sont continues sont optimales. C'est rarement le cas en pratique, où il faut se contenter de fonctions presque partout continues.

Afin d'obtenir cette propriété de continuité, nous suggérons d'utiliser des fonctions d'expression du génotypes en comportement qui soient indépendantes du substrat génétique. Dans les modèles évoqués précédemment permettant de générer des structures ex nihilo (*cf.* section 2.3.4), la fonction d'expression n'est justement pas indépendante de la structure génétique. La hiérarchisation de cette structure et les interdépendances entre des parties peut-être éloignées font que l'expression de parties distantes du génotype sont susceptibles d'avoir souvent un fort impact sur l'expression d'autres. Par exemple en programmation génétique, l'utilisation d'une structure d'arbre fait que les variations opérées à la racine ou au feuilles n'ont pas du tout le même impact, l'expression de ces dernières étant complètement subordonnée à l'expression de ces premières. Des petites variations sur des nœuds proches de la racines se traduiront presque systématiquement par un changement important du comportement de l'arbre lui même. Que cette structure soit utilisée directement ou pas, il y a toutes les chances pour que l'impact sur la structure évaluée soit grand.

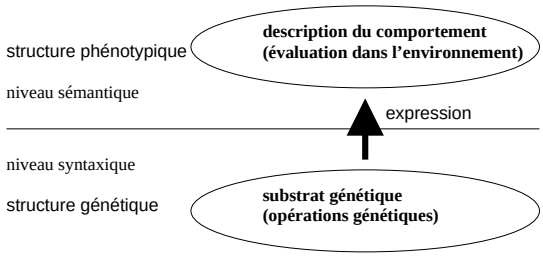

FIG. 3.1 – *Le substrat génétique (structure génotypique) est exprimé en une description du comportement (structure phénotypique). Nous préconisons une séparation entre les niveaux syntaxique et sémantique.*

En d'autres termes, nous préconisons lors de la conception de cette fonction de séparer le niveau « syntaxique » (celui de la structure génétique), du niveau « sémantique » (celui de la structure qui va donner lieu au comportement) (*cf.* figure 3.1). S'il n'y a pas de séparation, les problèmes évoqués précédemment découlent de ce que le niveau sémantique contraint trop le niveau syntaxique.

3.1.2 Pouvoir expressif

Notre objectif est de produire de façon automatique des comportements d'agent. Pour ce faire il est souhaitable de ne pas avoir à donner de limites à l'espace de recherche, d'autant cet espace de recherche n'est pas connu à l'avance. Nous souhaitons pouvoir obtenir des structures décrivant le comportement de complexité arbitraire ; la sélection se chargera de déterminer le niveau de complexité nécessaire par le filtrage lors de l'évaluation du comportement. Le modèle candidat d'évolution de comportement d'agent doit donc permettre d'obtenir des structures de description

des comportements de complexité au moins égale à celle des approches précédemment évoquées (réseaux de neurones, (arbres de) programmes, etc.).

Ce fort pouvoir expressif devrait être celui d'une structure compréhensible et modulaire. Ces pré-requis ont plusieurs avantages :

- pouvoir *expliquer* le comportement exprimé,
- pouvoir réaliser, modifier ou tester manuellement des solutions, voire par la suite les soumettre à l'évolution pour les améliorer encore (*bootstrap*) ou tester leur stabilité,
- pouvoir construire de façon incrémentale la structure, en exploitant sa modularité.

Tout ceci peut s'avérer délicat avec des structures pourtant très efficaces comme les réseaux de neurones.

3.2 Expression de gènes à l'aide d'une pile

Les modèles existants ne satisfont pas simultanément les principes Éthogénétiques. D'une part, les encodages directs permettent un accès simple à la structure, et facilitent donc les manipulations humaines. Cependant, ils ne vérifient pas simultanément les propriétés de continuité et de pouvoir expressif. D'autre part, les encodages indirects sont plus flexibles et plastiques. Cependant dans les modèles existants l'indirection rend problématique la conception des opérateurs génétiques pour respecter les principes de quasi-continuité et de pouvoir d'expression.

Le principe d'expression de gènes que nous présentons ci-après est un modèle d'encodage indirect, construit dans le but de satisfaire les principes Éthogénétiques. Il s'agit d'un modèle d'interprétation à l'aide d'une pile (*Stack-Based Gene Expression* [79]).

L'interprétation à l'aide d'une pile est un modèle d'expression simple, déjà employé en informatique pour des langages comme PostScript ou Forth. L'utilisation d'une pile permet de n'avoir que des interactions locales entre instructions via la pile. Ainsi la plupart des modifications du génotype n'ont qu'un impact local sur la structure construite. Cette propriété permet de respecter le principe de quasi-continuité : des petites modifications de la chaîne vont se traduire le plus souvent par des petites modifications de la structure.

Par ailleurs nous devons choisir un langage de construction pour la structure. Celui-ci devrait plutôt être déclaratif, de façon à éviter les problèmes de hiérarchie dans le génotype. Le langage doit aussi contribuer à la localité en évitant d'avoir des atomes qui modifient la structure trop profondément. Nous proposons donc d'utiliser un langage de construction de structure à grain fin. Enfin, le langage doit permettre de décrire n'importe quelle structure du type que l'on a choisi de faire évoluer, et ce quelle que soit sa complexité. Ainsi la propriété de pouvoir d'expression est respectée.

Avant de décrire du modèle (section 3.2.3), nous allons présenter rapidement des travaux apparentés (section 3.2.1) et le modèle biologique qui est notre source d'inspiration (section 3.2.2).

3.2.1 Travaux apparentés

Perkis [105] a le premier utilisé avec succès un modèle de programmation génétique à l'aide d'un pile (Stack-Based Genetic Programming). Il ne l'a cependant pas utilisé pour construire des structures, seulement pour exécuter un programme. Cette approche s'est avérée compétitive par rapport à la programmation génétique classique.

Stoffel et Spector [120] ont repris cette approche en la parallélisant (HiGP). Ils l'ont ensuite enrichie en ajoutant dans le langage de pile des opérateurs permettant d'opérer sur le flux d'instruction en train d'être exécuté [119, 118]. De cette façon l'expression d'un programme peut être auto-modifiante et adaptative. Ces améliorations ont accéléré la convergence de façon significative, cependant eux non plus n'ont pas utilisé ce modèle à pile pour produire des structures.

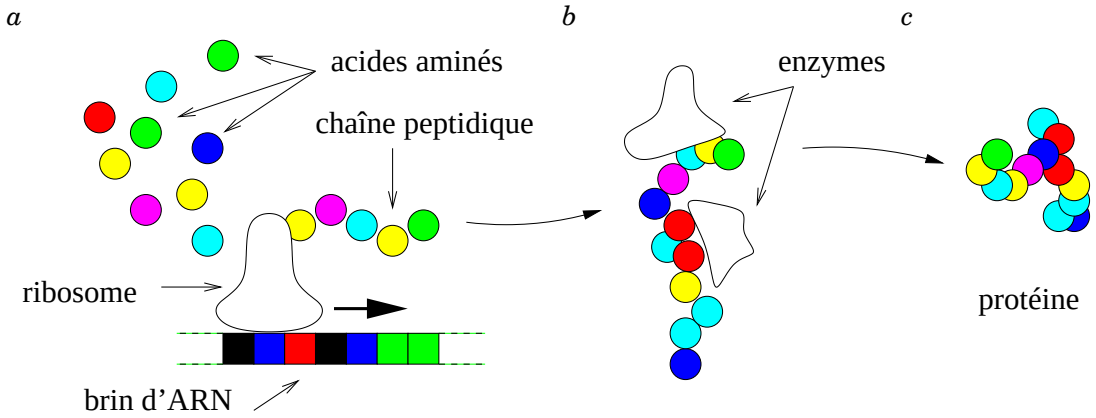

FIG. 3.2 – *Synthèse d'un protéine. a. le ribosome (une enzyme) lit une séquence d'acides nucléiques (brin d'ARN) et assemble des acides aminés en une chaîne polypeptidique ; b. la chaîne polypeptidique se replie, avec le concours d'enzymes... ; c. ...en la protéine finale*

3.2.2 Le modèle biologique

Le modèle biologique qui est la source d'inspiration des approches évolutionnistes de construction de structures est la synthèse de protéines dans la cellule. La figure 3.2 en résume les étapes principales.

Il s'agit d'un modèle à encodage indirect, puisque la représentation génétique n'est pas sous forme de protéine. Les modèles en informatique évolutionniste n'ont pas nécessairement respecté ce principe, comme nous l'avons précédemment détaillé section 2.3.4.

D'autre part, le modèle biologique semble bien séparer la « syntaxe » génétique de la « sémantique » protéique. En effet, le comportement de la protéine n'est pas tant déterminé par sa composition en acides aminés que par sa *forme*. Or la séquence d'acides nucléiques ne spécifie que la séquence d'acides aminés. La forme, donc la comportement de la protéine, est déterminée pour partie par des interactions locales et spontanées entre acides aminés, et aussi pour autre partie par des enzymes présentes dans la cellules qui vont intervenir lors du repliement de la chaîne peptidique. Les enzymes peuvent n'opérer que sélectivement sur certains sites de la chaîne. La formation d'une protéine est donc un processus complexe, qui dépend autant de l'environnement dans lequel il a lieu que de ses données initiales (la chaîne peptidique).

3.2.3 Description

De la même façon que dans le modèle biologique, nous allons procéder en deux étapes : une traduction puis une interprétation. Comme la représentation génétique ne constituera qu'un support syntaxique dénué de toute signification, notre choix s'est porté sur une chaîne de bit. La chaîne de bit présente les avantages d'être simple, flexible et déjà très utilisée par les algorithmes génétiques. Lors du processus d'expression elle est parcourue par la « tête de lecture » du traducteur, comme le ribosome parcourt le brin d'ARN.

Le processus d'expression de la chaîne a lieu comme suit :

1. le traducteur associe à chaque *codon* (n-uplet de bit) un lexème.
2. l'interprète reçoit le flux de lexèmes. Ceux-ci sont successivement empilés et/ou une action qui est associée à chacun est exécutée.
3. lorsque le flux de lexèmes est tari, l'interprète dépile la structure finale dont on veut évaluer le comportement.

La figure 3.3 résume le processus d'expression génétique.

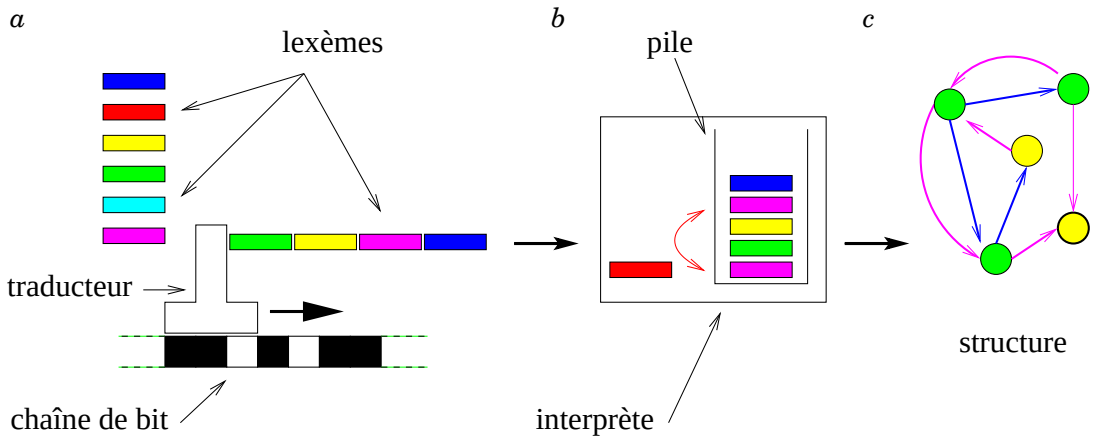

FIG. 3.3 – *Expression de gènes à l'aide d'une pile. a. la traducteur lit une chaîne de bit et produit un flux de lexèmes ; b. les léxèmes sont envoyés à un interprète, qui peut empiler et/ou effectuer une action liée au lexème ; c. la structure finale est dépilée*

L'analogie avec le modèle biologique est simple (*cf.* figure 3.2). Comme pour la protéine, le comportement exprimé par la structure dépend principalement de sa morphologie, plus que de chacune de ses composantes prises isolément. La morphogénèse de la structure est aussi le fruit d'interactions locales – entre les lexèmes, via la pile principalement. L'interprète à pile constitue le cadre et le catalyseur dans lequel les lexèmes s'expriment pour former la structure. L'interprète joue ainsi un rôle similaire à celui des enzymes qui agissent sur la chaîne peptidique en cours de repliement.

La traduction

Elle s'appuie sur un code génétique, c'est-à-dire une fonction :

$$\mathcal{G} : \{0,1\}^n \longrightarrow \mathcal{T} \qquad (|\mathcal{T}| \leq 2^n)$$

où $\mathcal{T}$ est un ensemble de lexèmes, et n est la taille de nos codons. Il y a deux catégories de lexèmes :

- les instructions pour le langage à pile, nous les appellerons « lexèmes de pile ». Ils comptent par exemple le lexème `swap`, dont l'action est d'échanger les deux éléments du sommet de la pile.
- et les instructions propres à la construction de la structure, nous les appellerons « lexèmes de structure ». Par exemple pour une structure de graphe, nous pouvons imaginer un lexème `connect` qui crée un arc reliant deux nœuds du graphe.

Notons que le code génétique peut être redondant (surjection), en associant plusieurs codons au même lexème.

L'interprétation

L'interprète est générique, il est le même quelle que soit la structure décrite ou le langage à pile utilisé.

Chapitre 4

Modèles

Les principes Éthogénétiques fournissent un cadre pour la conception par évolution artificielle de structures décrivant des comportements d'agents. L'expression à l'aide d'une pile est une proposition de codage pour concevoir des modèles vérifiant ces principes. Comme annoncé nous appliquons ce principe à la conception de graphes étiquetés, similaires à des ATN [1]. Cette instanciation s'est faite dans le modèle ATNoSFERES que nous allons présenter ci-après (section 4.1).

Afin de pouvoir évaluer ce modèle sur plusieurs applications et de pouvoir le comparer à d'autres modèles sur une même application, il est intéressant de disposer d'une plate-forme commune d'évaluation. Il s'agit ici de pouvoir comparer différents algorithmes et formalismes évolutionnistes sur une même instance d'une tâche multi-agent, ou de pouvoir appliquer une même technique à plusieurs instances différentes. SFERES, notre framework et environnement de développement a été conçu dans ce but. Nous décrivons SFERES section 4.2.

Note : les modèles présentés dans ce chapitre ont été développés en coopération avec Sébastien Picault, thésard de l'équipe Miriad (pour ATNoSFERES) et avec Stéphane Doncieux, thésard de l'équipe AnimatLab (pour SFERES).

4.1 ATNoSFERES

ATNoSFERES [80] a été conçu pour concevoir automatiquement des comportements d'agents évoluables, en respectant les principes de l'Éthogénétique. Nous y décrivons le comportement d'un agent à l'aide d'un graphe orienté et étiqueté, similaire à un ATN. Les ATN sont des graphes qui ont d'abord été utilisés dans le cadre du traitement automatique du langage [129]. Ils ont aussi été utilisés pour représenter des comportements d'agents [57].

Nous allons d'abord décrire comment le graphe est utilisé pour décrire un comportement (section 4.1.1). Puis nous décrivons comment il est produit à partir d'une chaîne bit (section 4.1.2), en suivant le modèle expliqué au chapitre précédent.

4.1.1 De l'ATN au comportement

À chaque classe d'agents est associé un ensemble de lexèmes correspondant d'une part à des *actions* pouvant être exécutées par les agents de cette classe (primitives comportementales), d'autre part à des *conditions* portant sur l'environnement des agents et pouvant restreindre ou orienter le choix des actions [2].

Le graphe de comportement d'un agent comporte systématiquement un nœud de départ (appelé par la suite `Start`) et un nœud final (`End`). Les autres nœuds sont reliés par des arcs qui peuvent

1. Augmented Transition Network
2. Sur les graphes donnés en exemple, les actions seront notées `uneAction!` et les conditions uneCondition?.

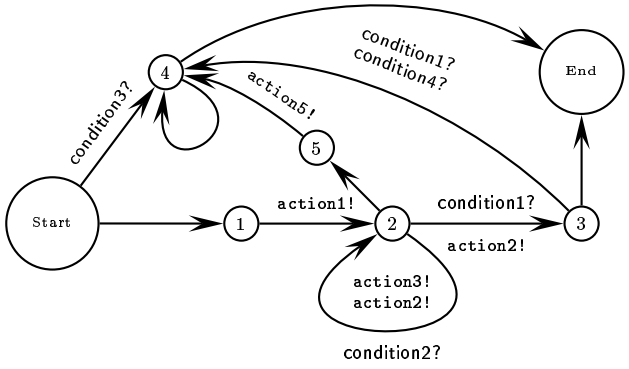

FIG. 4.1 – *Un exemple d'ATN.*

être étiquetés par un ensemble de conditions et par une séquence d'actions (voir l'exemple donné dans la figure 4.1).

Tout agent est initialisé dans l'état `Start`. Ses actions sont déterminées de la façon suivante :

1. Sélectionner, parmi les arcs issus du nœud courant, ceux qui sont *franchissables*, c'est-à-dire sans conditions ou dont les conditions sont vérifiées *simultanément*.
2. Choisir un de ces arcs aléatoirement.
3. Le franchir (se placer sur le nœud auquel il aboutit) en exécutant les éventuelles actions associées (dans l'ordre).

L'agent cesse d'agir lorsque le parcours de son graphe parvient au nœud `End`.

4.1.2 De la chaîne de bit à l'ATN

La construction du graphe se fait selon le principe expliqué au chapitre précédent (*cf.* section 3.2). Il ne nous reste donc qu'à expliciter ce qui doit être instancié pour faire évoluer un graphe. Un exemple simple est détaillé ensuite.

Le traducteur

Le traducteur est complètement défini par le code génétique qu'il utilise. Comme dit précédemment il y a deux type de lexèmes :

- les *lexèmes de pile*, qui sont utilisés pour manipuler la pile lors de la construction de la structure.
- les *lexèmes de structure*, qui incluent :
  - les *lexèmes d'ATN*, comme ceux utilisés pour créer les nœuds ou pour les connecter
  - les *lexèmes d'agent*, comme les conditions et actions, qui sont utilisés pour étiqueter les arcs des ATN.

L'interprète

L'interprète construit le graphe à partir du flux de lexèmes. Il les lit successivement depuis le traducteur et exécute une éventuelle actions associée à chacun (*cf.* le tableau 4.1) :

- chaque lexème de pile exécute une opération spécifique sur la pile.
- les lexèmes de condition et d'action sont juste empilés.
- le lexème de création de nœud ne fait qu'empiler un nouveau nœud.
- les lexèmes de connexion, selon le cas, connectent un nœud interne à un autre nœud interne, au nœud `Start` ou au nœud `End`, en l'étiquetant avec l'ensemble des conditions et la liste

token	(état initial de la pile)	$\longrightarrow$	(pile résultante)
dup	$(x\ y\ ...)$	$\longrightarrow$	$(x\ x\ y\ ...)$
del	$(x\ y\ ...)$	$\longrightarrow$	$(y\ ...)$
dupNode	$(x\ y\ N_i\ z\ ...)$	$\longrightarrow$	$(N_i\ x\ y\ N_i\ z\ ...)$
delNode	$(x\ N_i\ y\ N_i\ z\ ...)$	$\longrightarrow$	$(x\ y\ N_i\ z\ ...)$
popRoll	$(x\ y\ ...\ z)$	$\longrightarrow$	$(y\ ...\ z\ x)$
pushRoll	$(x\ ...\ y\ z)$	$\longrightarrow$	$(z\ x\ ...\ y)$
swap	$(x\ y\ ...)$	$\longrightarrow$	$(y\ x\ ...)$
node	$(x\ ...)$	$\longrightarrow$	$(N_i\ x\ ...)$ [a]
startConnect	(c1? c2? c1? x a2! a1!y N_i...)	$\longrightarrow$	$(x\ y\ N_i...)$ [b]
endConnect	(c1? c2? c1? x a2! a1!y N_i...)	$\longrightarrow$	$(x\ y\ N_i...)$ [c]
connect	(c1? c2? x N_i y c1? z a2! a1! t N_j u ...)	$\longrightarrow$	$(x\ N_i\ y\ z\ t\ N_j\ u\ ...)$ [d]
condition?	$(x\ ...)$	$\longrightarrow$	(condition? x ...)
`action!`	$(x\ ...)$	$\longrightarrow$	(`action!` x ...)

TAB. 4.1 – *Le langage de construction d'ATN.*

[a] crée un nœud N_i
[b] connecte le nœud N_i à `Start` en étiquetant l'arc avec (c1? & c2?) et (a1!, a2!)
[c] connecte le nœud N_i à `End` en étiquetant l'arc avec (c1? & c2?) et (a1!, a2!)
[d] connecte N_j à N_i en étiquetant l'arc avec (c1?& c2?) et {a1!,a2!}

des actions rencontrées en remontant la pile jusqu'au dernier nœud nécessaire à la connexion (les nœuds `Start` et `End` sont implicites et ne sont pas empilés).

Classiquement pour les langages à pile, si une instruction ne peut pas être exécutée faute d'opérandes dans la pile en particulier, l'instruction est ignorée.

Enfin, quand le flux de lexèmes est tari, l'interprète termine la construction de l'ATN (ceci peut être vu comme l'action d'un lexème virtuel de terminaison, toujours en fin de flux. Ceci n'est pas contradictoire avec la généricité de l'interprète). La terminaison consiste en premier lieu à traiter les conditions et actions encore présentes dans la pile comme des connexions *implicites*, ce qui implique la création de nouveau arcs. En second lieu, la consistance de l'ATN est vérifiée. Il s'agit d'une part de connecter le nœuds `Start` à tous les nœuds n'ayant pas d'arcs entrants ou pas d'autres arcs entrants que provenant d'eux-mêmes, et d'autre part de connecter à `End` les nœuds n'ayant pas d'arcs sortant.

Notons que le langage est assez déclaratif (seulement localement impératif) et qu'il ne requiert pas d'explicitation détaillée de la structure.

Exemple

Pour l'exemple de fonctionnement *cf.* la figure 4.2 page 22.

4.2 SFERES

SFERES[3] [77, 78] est un framework et une plate-forme[4] d'évolution artificielle et de simulation multi-agent.

SFERES est constitué de deux parties. La première est la partie évolutionniste et représente les algorithmes évolutionnistes tels que présentés section 2.3.2. La seconde est la partie simulation et représente des simulations multi-agent. Les deux parties sont liées par l'évaluation des individus de l'algorithme évolutionniste dans la simulation. L'évaluation porte sur le comportement des agents simulés. Ce comportement est décrit plus ou moins directement selon les cas dans le génome sur lequel travaille l'évolution.

3. `http://miriad.lip6.fr/SFERES/`
4. codée en C++ sous Linux, prochainement distribuée sous licence libre.

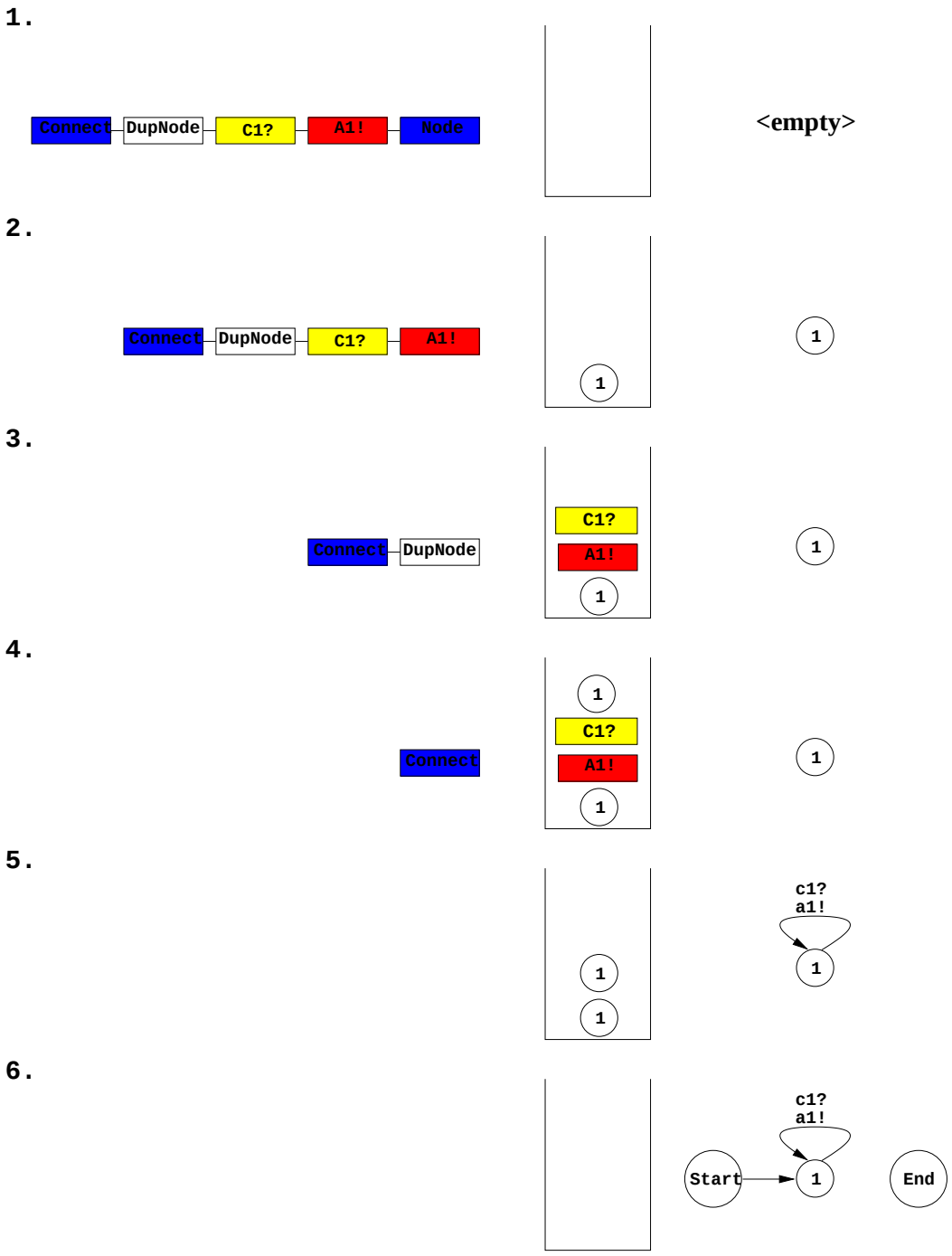

FIG. 4.2 – *Exemple de construction d'un ATN à l'aide d'une pile. 1. état initial : pas de structure et pile vide ; 2. un premier nœud est créé ; 3. les conditions et actions sont empilées ; 4. dupNode empile une copie du premier nœud rencontré en remontant la pile ; 5. le nœud 1 est connecté à lui-même ; 6. état final, après une connexion implicite au nœud* `Start`.

L'originalité de SFERES réside dans le couplage de la partie évolution artificielle et de la partie simulateur multi-agent pour évaluer les individus, ce que n'offrent pas la plupart des bibliothèques d'algorithmes évolutionnistes [124, 122, ...]. Ce couplage est décrit section 4.2.3. Grâce à ce couplage, expérimenter plusieurs technique d'apprentissage évolutionnistes sur une même tâche multi-agent ou employer une même technique sur plusieurs tâches peut être fait sans modifications conceptuelles ni réécriture de code.

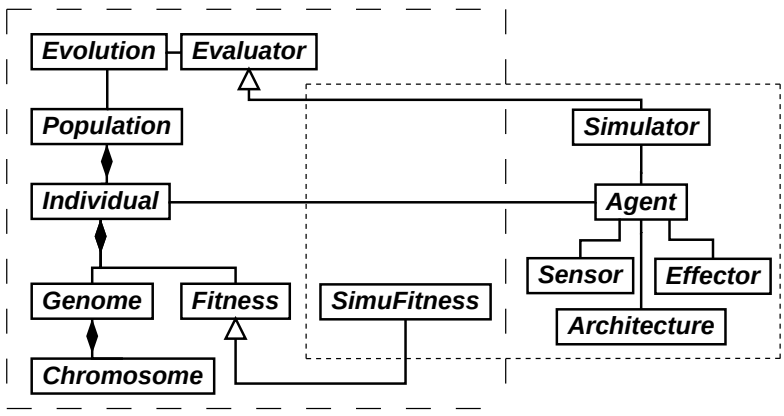

FIG. 4.3 – *Diagramme UML des classes de SFERES. Encadrée de tirets, la partie évolutionniste, et encadrée de pointillés, la partie simulation.*

La figure 4.3 rassemble les classes abstraites du framework en les regroupant par parties. La relation entre les classes `Agent` et `Individual`, la classe `SimuFitness` et l'héritage de `Evaluator` par `Simulator` constituent le lien entre les deux parties du simulateur (*cf.* section 4.2.3).

Nous allons d'abord décrire la partie évolutionniste du framework (section 4.2.1), puis la partie simulation (section 4.2.2). Après la description du couplage entre ces deux parties (section 4.2.3), nous décrivons les « hot spots » du framework, c'est-à-dire que nous détaillons ce qui doit être dérivé en fonction de que l'on veut faire (section 4.2.4). Enfin, nous donnons un exemple d'utilisation (section 4.2.5) et concluons sur notre utilisation de SFERES pour ce qui nous intéresse ici (section 4.2.6).

4.2.1 Partie évolutionniste

La figure 4.4 rassemble les classes de la partie évolutionniste du framework. Ces classes peuvent être regroupées en deux parties : le moteur d'évolution et l'individu.

Moteur d'évolution

Le moteur d'évolution (cadre en tirets sur la figure 4.4), regroupe les classes gérant le déroulement de l'algorithme évolutionniste.

La class `Evolution` gère l'aspect technique des calculs. La répartition des calculs sur différentes machines, le choix des statistiques enregistrées pendant le déroulement de l'algorithme font partie de ses attributions.

La classe `Population` contient l'ensemble des individus et prend en charge la sélection de l'algorithme évolutionniste (*cf.* figure 2.4). La sélection est tout à fait indépendante du codage des solutions et des opérateurs génétiques. La classe `Population` a pour tâche de générer une nouvelle population à partir de la population courante et de mettre à jour les informations statistiques (performance du meilleur individu, moyenne...), dont la class `Evolution` gère la sauvegarde.

Au cours de la génération de nouveaux individus, la classe `Population` a besoin de les évaluer. Elle envoie alors un message à la classe `Evaluator`, qui les met à jour.

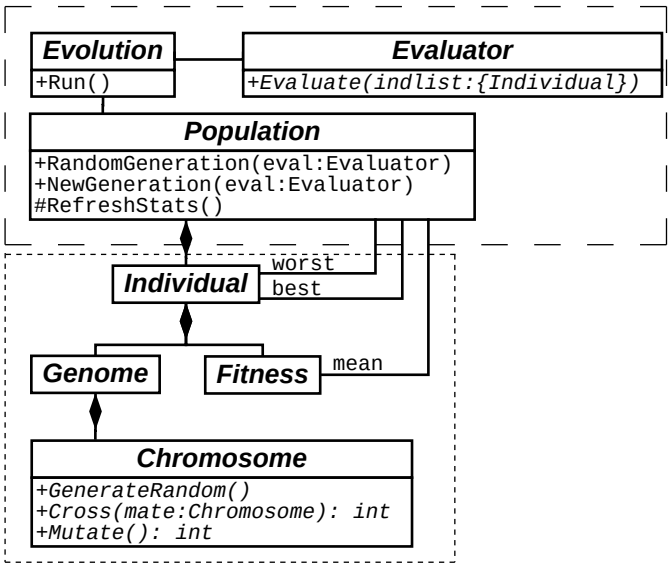

FIG. 4.4 – *Diagramme UML des classes de la partie évolutionniste de SFERES. Encadré de tirets, le moteur d'évolution, et encadré de pointillés, l'individu.*

Individu

L'individu contient toutes les informations relatives à une solution potentielle : l'ensemble de structures et de paramètres composant la solution (classe `Genome`) ainsi que les performances de cette solution sur le problème posé (class `Fitness`).

Un génome (classe `Genome`) est composé de chromosomes (classe `Chromosome`), qui contiennent le code génétique à proprement parler. La classe `Chromosome` contient également toutes les méthodes de manipulation de l'information génétique : principalement la génération aléatoire, le croisement et la mutation. Ces méthodes sont utilisées par la population au moment de la génération d'un nouvel individu. Leur implémentation est fortement liée à un codage génétique particulier.

La fonction d'évaluation, qui va définir la pression sélective exercée sur les individus, est contenue dans la classe `Fitness`. Du point de vue de `Population`, cette classe permet d'ordonner les individus en vue de leur sélection. Cette classe est le principal biais introduit par le concepteur pour diriger l'évolution vers les solutions souhaitées.

Principe de fonctionnement

Lors du lancement d'une expérience, `Evolution` exécute la méthode `Run()` qui fait appel à la méthode `NewGeneration()` de `Population`. Cette méthode génère de nouveaux individus et fait appel pour cela à `GenerateRandom()` lors de la première génération, puis à `Cross()` et `Mutate()` de la classe `Chromosome`. `Population` évalue alors la performance des individus nouvellement générés par un appel à la méthode `Evaluate()` de `Evaluator`. Le résultat de cette évaluation, stocké dans `Fitness`, servira à sélectionner les individus les plus performants lors du prochain `NewGeneration()`. À la fin de `NewGeneration()`, `Population` met à jour les différentes statistiques avec `RefreshStats()` avant de rendre la main à `Evolution`.

4.2.2 Partie simulation

La figure 4.5 rassemble les classes de la partie simulation du framework. Ici encore, on peut subdiviser les différents éléments en deux parties : le moteur de simulation et l'agent. Précisons d'emblée que, dans une implémentation particulière, le moteur de simulation et les agents, abstractions utiles au fonctionnement du framework, peuvent ne constituer qu'une interface avec un

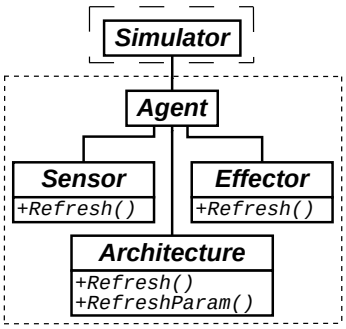

FIG. 4.5 – *Diagramme UML des classes de la partie simulation de SFERES. Encadré de tirets, le moteur de simulation, et encadré de pointillés, l'agent.*

simulateur existant [59, 58] ou des robots.

Moteur de simulation

Le moteur de simulation (classe `Simulator`) prend en charge la gestion de l'environnement des agents et le déroulement des simulations. `Simulator` est une classe abstraite dérivée d'`Evaluator`, permettant d'effectuer les évaluations des individus dans la simulation choisie. Il transmet les `Individu` aux `Agent` qui vont servir à évaluer leurs performances et il met à jour `SimuFitness`, classe dérivée de `Fitness` à laquelle on a ajouté des méthodes liées à l'évaluation du comportement d'un agent. Ces méthodes servent à évaluer le comportement d'un `Agent` au cours du temps, nous y reviendrons à la section 4.2.3).

Agent

Les interactions de l'`Agent` avec son environnement se font par le biais de capteurs (classe `Sensor`) et effecteurs (classe `Effector`), et l'information qui circule depuis les capteurs et vers les effecteurs est traitée par des `Architecture`. Ces classes forment à l'intérieur de l'agent un réseau dont la taille et la structure ne sont pas imposées et peuvent être commandées par un `Chromosome` (*cf.* section 4.2.3). Les `Chromosome` peuvent aussi définir la structure interne de chacun de ces éléments.

Principe de fonctionnement

`Simulator` est appelé par le biais de sa méthode `Evaluate()` depuis `Population`. Les `Agent` ont accès à un `Individu` et peuvent utiliser son information génétique pour définir leur structure ou leurs paramètres. Au cours de l'évaluation, `Simulator` met à jour les `SimuFitness`.

4.2.3 Couplage entre les parties évolutionniste et simulation

La figure 4.6 représente la classe `SimuFitness`, les héritages et relations sur lesquels repose le couplage entre les parties évolutionniste et simulation de SFERES.

`Simulator`, la classe de base de la partie simulation du framework, hérite d'`Evaluator`. Elle dispose donc de la méthode permettant à `Population` de demander l'évaluation d'individus. Dans le cas de `Simulator`, cette méthode, `Evaluate()`, a la particularité de nécessiter une classe dérivée de `Fitness`, `SimuFitness`, pour effectuer ces évaluations.

`SimuFitness` a en effet des méthodes d'interface propres au calcul des performances d'un agent au sein de la simulation. Celles-ci sont nécessaires car à la différence du cas d'une optimisation de fonction, où le calcul de la valeur de la fonction au point défini par le génome est suffisant, lors de l'évolution de systèmes multi-agent, plusieurs critères sont à prendre en compte. Il ne suffit pas d'observer un agent à un instant donné pour avoir une idée de ses performances, il est nécessaire

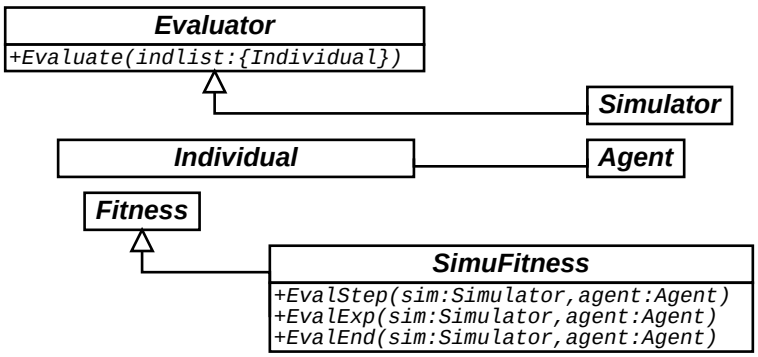

FIG. 4.6 – *Diagramme des classes du couplage entre les parties évolutionniste et simulation de SFERES*

de pouvoir le suivre lors de ses interactions avec son environnement et les agents qui l'entourent. Le calcul de `SimuFitness` se fait donc pas à pas avec la méthode `EvalStep()`. Il peut ensuite être affiné à la fin d'une expérience et à la fin de l'évaluation avec les méthodes `EvalExp()` et `EvalEnd()`.

Enfin, `Agent` peut être lié au plus à un `Individu` qui est évalué. Ce lien permet de définir tout ou partie de l'agent à partir de l'information génétique contenue dans `Genome`. L'agent, les capteurs, les effecteurs et les architectures de contrôle peuvent ainsi être définies par les `Chromosome`, aussi bien dans leur nombre, leurs connexions que dans leur structure interne. Aucune contrainte n'est imposée quant à la relation entre un `Agent` et un `Chromosome`. Les possibilités sont très variées, aussi une interface imposée restreindrait la portée générale du framework.

4.2.4 Dérivations de classes

L'implémentation de nouvelles expériences ou de variantes d'expériences existantes se fait simplement par dérivation des classes concernées tout en conservant les autres.

L'étude d'un nouvel algorithme de sélection ne nécessite que la réimplémentation de la classe `Population`. Il peut ensuite facilement être comparé aux stratégies utilisées précedemment en le faisant fonctionner sur des expériences déjà étudiées.

Le changement de simulation nécessite l'implémentation des classes liées à l'environnement simulé, c'est-à-dire `Simulator`, `Agent`, `Sensor` et `Effector`. Les `Architecture` utilisées pour contrôler les `Agent` n'étant pas connectées directement à l'environnement simulé, elles n'ont pas à être redéfinies.

La résolution d'un problème différent dans un environnement déjà implémenté ne nécessite que de changer la pression sélective appliquée aux individus. Il suffit donc d'implémenter une seule nouvelle classe: `SimuFitness`, toutes les autres classes étant identiques.

Le changement de l'architecture de contrôle de l'agent requiert d'implémenter la nouvelle classe `Architecture`, ainsi que, généralement, la classe qui sert à la définition de sa structure ou de ses paramètres, à savoir le `Chromosome` associé.

Enfin, pour étudier un nouveau codage génétique, il faut implémenter un nouveau `Chromosome` ainsi que, généralement, des classes qui vont l'utiliser (`Architecture` le plus souvent).

4.2.5 Exemple : expérience proie-prédateur

Les agents considérés ici seront des robots à roues. Un premier robot, la proie, dispose uniquement de capteur de proximités et elle est capable de se mouvoir rapidement. Un deuxième robot, le prédateur, dispose, en plus de capteurs de proximité, d'une caméra pour détecter la proie, mais il se déplace moins vite qu'elle. Les robots se déplacent dans une arène de taille donnée. Cette expérience est celle décrite dans [49]. On considère que lorsque le prédateur touche la proie, il la capture.

`Simulator` contiendra les équations de mise à jour des coordonnées avec gestion des collisions. La partie `Agent` sera identique pour les deux robots. Elle correspondra à un robot à roues et contiendra ses différentes coordonnées. Les capteurs (classe `Sensor`) utilisés seront de deux types : les capteurs de proximité et la caméra linéaire qui enverra à l'architecture de contrôle, qui lui est reliée, ce qui est dans son champ de vision. Les effecteurs (classe `Effector`) seront les moteurs. Chaque agent en aura deux, un par roue et les moteurs de la proie seront plus "puissants" que les moteurs du prédateur. L'architecture de contrôle utilisée pour relier capteurs à effecteurs sera, par exemple, un réseau de neurones de type feed-forward avec trois neurones sur la couche cachée.

La `Fitness` de la proie pourra être le temps pendant lequel la proie a survécu, pour le prédateur, ce sera l'opposé du temps qu'il a mis à attraper la proie (valeur à maximiser).

L'information génétique contiendra les poids des connexions des neurones. Mettons que l'on utilise pour la partie évolution un algorithme génétique. `Chromosome` sera donc une chaîne binaire. `Population` doit prendre en compte deux ensembles de solutions potentielles : celle des proies et celle des prédateurs. De plus, pour l'évaluation, elle doit choisir les adversaires qui vont s'affronter.

L'évaluation consistera à laisser les deux agents se déplacer dans l'arène jusqu'à ce que le prédateur capture la proie ou jusqu'à ce qu'un temps limite soit dépassé.

4.2.6 Conclusion sur SFERES

SFERES a été utilisé pour diverses simulations (proies-prédateurs, dirigeable), pour diverses techniques évolutionnistes (stratégies évolutionnistes, algorithmes génétiques, programmation génétique en cours d'implémentation) et diverses architectures d'agent (réseaux de neurones, PID, arbres de programmes).

Un simulateur de SFERES est en cours de dérivation, pour venir s'interfacer indifféremment avec les robots de MICRobES ou le simulateur multi-robots pour les robots de MICRobES. Cette interface se fait grâce au protocole fourni avec les robots.

Enfin, ATNoSFERES a été codé avec la plate-forme SFERES, nous permettant ainsi de l'évaluer sur diverses tâches et sur une même tâche de facilement pouvoir comparer avec d'autres techniques évolutionnistes et architectures.

Chapitre 5

Expérimentations

Le modèle ATNoSFERES a été implémenté sous la plate-forme SFERES. Des expériences préliminaires ont été menées pour bien vérifier l'adéquation du modèle implémenté avec la formulation théorique qui lui a donné naissance. C'est-à-dire qu'il s'agit de vérifier que les principes Éthogénétiques et les propriétés prédites pour l'expression à l'aide d'un pile sont bien observées expérimentalement sur cette instance. D'autres expériences en cours ont pour but de vérifier si ce modèle, répondant aux attentes théoriques, permet d'obtenir des solutions simples, et permet d'attaquer des problèmes distribués.

Ces expériences préliminaires (section 5.1) et les expériences en cours (section 5.2) sont décrites ci-après. Chaque expérience a été conçue pour répondre à une question bien précise :

- le modèle est-il fonctionnel? (section 5.1.1)
- le modèle peut-il générer des solutions compactes/simples à un problème relativement complexe? (section 5.2.1)
- le modèle fonctionne-t-il pour un SMA, permet-il de recréer une dynamique d'écosystème? (section 5.2.2)

5.1 Expériences préliminaires

5.1.1 Fonctionnalité du modèle

Cette expérience en simulation, très simple, a d'abord eu pour but de tester la possibilité effective de construction de comportements adaptés à l'environnement dans un cadre classique (opérations génétiques effectuées par le système, évaluation explicite et individuelle de l'adéquation des agents à leur milieu, etc.).

Protocole expérimental

L'environnement est un espace discret contenant une lampe qui change aléatoirement de couleur (de vert à rouge et vice-versa) à chaque pas de temps (avec une probabilité $p = 0.05$).

On définit un agent `Pieton` disposant des lexèmes de condition et d'action décrits dans le tableau 5.1. Chaque agent `Pieton` jouera le rôle d'un individu de l'algorithme évolutionniste. Le comportement souhaité consiste à aller à droite lorsque la lampe est verte, et à gauche quand elle est rouge.

Le code génétique associé à la classe `Pieton` comporte les 11 lexèmes de l'interpréteur et les 8 lexèmes d'action/condition de `Pieton`. Il faut donc au minimum 32 codons (soit des codons de 5 bits). Dans les expériences présentées ici, un code génétique de 32 codons a été construit automatiquement à partir de la liste de tous les lexèmes (sans souci particulier de robustesse vis-à-vis des mutations). Chaque agent dispose par ailleurs d'un chromosome aléatoire, et d'une taille aléatoire.

Actions		Conditions	
`N!`	aucune action		
`R!`	aller à droite	`g?`	vrai si la lampe est verte
`L!`	aller à gauche	`r?`	vrai si la lampe est rouge
`U!`	aller en haut	`rand?`	vrai avec une probabilité
`D!`	aller en bas		$p = 0.5$

TAB. 5.1 – *Lexèmes d'actions et de conditions d'un* `Pieton`.

On applique une sélection darwinienne sur une population composée de ces individus en les évaluant à tour de rôle dans l'environnement défini ci-dessus. Un agent est placé seul dans son environnement pendant 100 pas de temps et noté selon les critères suivants : +1 point si le mouvement est conforme à ce que l'on recherche, -1 point s'il est inversé, 0 dans les autres cas. Seul le premier *mouvement* réalisé pour un pas de temps est noté (toutes les actions sauf `N!` comptent comme un mouvement). Pour pouvoir donner une note représentative de la *fitness*, on procède à 10 notations indépendantes (ce qui permet de tenir compte d'un éventuel polyéthisme) et on retient leur moyenne.

À chaque génération, la population comporte 100 agents. On calcule leur *fitness*, en fonction de laquelle on leur attribue une probabilité de reproduction et de décès. 30 nouveaux agents sont produits par croisement des chromosomes, puis 30 agents de l'ancienne population sont éliminés (la population reste donc constante).

Avant de calculer la *fitness* des individus d'une génération donnée, on leur fait subir des mutations. Pour ce faire, deux stratégies ont été envisagées :

1. tous les individus de la population subissent une mutation aléatoire de leur chromosome : les bits de la chaîne sont inversés avec une probabilité r (ici $r = 1$ %);
2. tous les individus subissent soit une mutation aléatoire (de taux r), soit (dans p % des cas) une insertion ou une délétion aléatoire d'un codon dans leur chromosome.

Résultats et analyse

Les agents étant initialisés sur le nœud `Start`, aucune action n'est effectuée durant le premier pas de temps : le parcours de l'ATN commence avec le choix d'un premier nœud. La plus haute valeur que peut prendre la *fitness* d'un agent au cours d'un test de 100 pas de temps est donc 99 (soit 99 mouvements corrects).

La figure 5.1 montre l'évolution moyenne de la fitness, calculée sur 10 expériences, dans la première situation (mutations uniquement acec un taux $r = 1$ %). La figure 5.2 présente l'évolution moyenne de la fitness dans le second cas, lorsque $p = 20$ % des individus subissent des insertions ou délétions aléatoires de codons (au lieu de mutations classiques).

Dans quelques cas (notamment avec la première stratégie), aucune bonne solution n'a été trouvée avant les 500 générations qui limitent les expériences.

Bien que le problème à résoudre soit ici extrêmement simple, ces résultats valident les principes de l'Éthogénétique et le modèle d'expression via une pile pour la construction de comportements évolutifs à partir d'un substrat génétique de granularité faible.

Le modèle choisi révèle un certain nombre de propriétés intéressantes. En particulier, il est possible d'effectuer des opérations d'insertion ou de délétion de codons sans nuire à la découverte d'une solution. Cela n'est bien entendu possible qu'au moyen d'un codage non positionnel, ce qui est en grande partie le cas dans ATNoSFERES où les instructions sont en partie déclaratives. Nous allons nous intéresser plus particulièrement à la spécificité de ce modèle en nous livrant à une analyse plus fine des comportements construits par la dynamique sélectionniste.

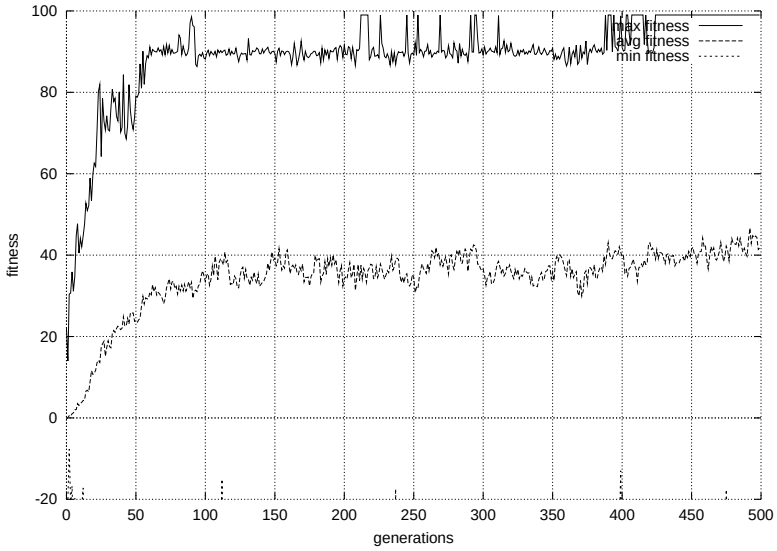

FIG. 5.1 – *Évolution moyenne de la fitness (mutations ponctuelles seulement, r = 1 %). Chromosomes initiaux de 50 à 100 bits.*

Optimalité et adaptation

L'ATN décrit par la figure 5.3 est une solution « optimale » au problème au sens où il permet d'obtenir les 99 points (*fitness* maximale) avec la structure la plus simple : un seul nœud, deux arcs (si la lampe est verte, aller à droite ; si elle est rouge, aller à gauche). Pour produire cet ATN, il suffit théoriquement de 35 bits seulement (en faisant un usage maximal du mécanisme de connexion implicite), par exemple pour coder les 7 lexèmes suivants :

node, g?, R!, *dupObject*, L!, r?, *dupObject*

Mais dans cette solution l'ordre et la nature des lexèmes jouent un rôle crucial ; elle est donc particulièrement vulnérable aux mutations.

On peut donc raisonnablement penser que les individus vont avoir besoin, pour développer des comportements stables vis-à-vis des mutations, de chromosomes un peu plus longs que la solution « ingénieur ».

Toutefois, un problème nouveau peut alors se poser. Le surcroît de matériel génétique disponible peut fort bien se révéler « nocif », c'est-à-dire donner naissance à des nœuds, des connexions supplémentaires qui peuvent *dégrader* le comportement en constituant une sorte d'« excroissance comportementale » parasite. Autrement dit, l'adéquation du comportement de l'agent à son milieu peut être rendu plus difficile par un allongement des chromosome.

Qu'en est-il exactement ? On peut formuler un début de réponse au vu des figures 5.4 à 5.7, réalisées alternativement avec les stratégies mutations seules ou insertions-délétions, et pour des tailles de chromosomes initialement comprises entre 200 et 300 bits, puis entre 500 et 700 (soit 20 fois la taille d'un chromosome produit par ingénierie).

D'une part, les résultats s'améliorent nettement lorsqu'augmente le nombre de bits dans le génome, et ce en dépit des difficultés induites par ce surplus d'information génétique. La combinatoire résultant de l'accroissement des chromosomes est donc régulée, biaisée, en faveur de comportements adaptés. Nous verrons comment.

D'autre part, on voit se réduire la différence entre les deux stratégies de variation. Plus, non seulement les insertions et délétions ne sont pas léthales pour la population, mais on peut même dire

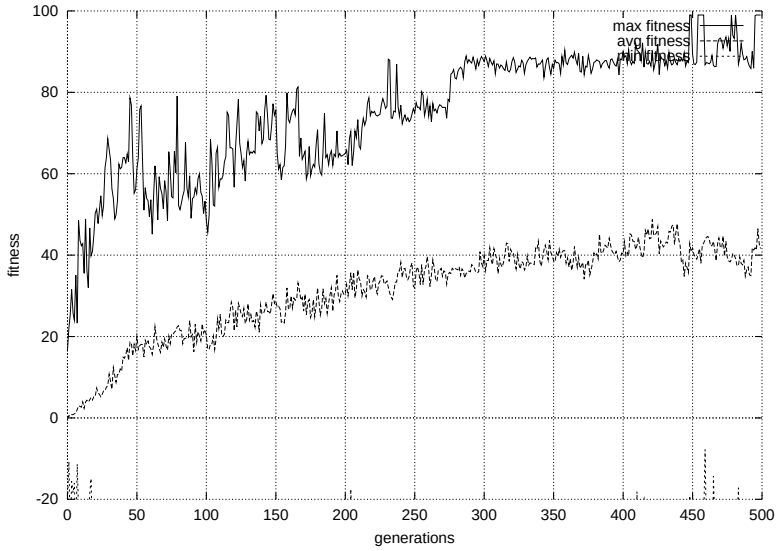

FIG. 5.2 – *Évolution moyenne de la fitness (insertions et délétions aléatoires de codons, $r = 1$ %, $p = 20$ %). Chromosomes initiaux de 50 à 100 bits.*

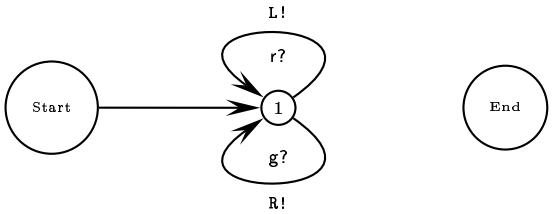

FIG. 5.3 – *L'ATN « optimal » (donnant la fitness maximale avec la structure la plus simple).*

qu'elles sont favorables puisqu'en fait, on observe aussi que le nombre de cas de non-convergence diminue dans le cas des insertions-délétions.

En fait, lorsqu'on augmente la taille d'un chromosome, les individus sont soumis à une double pression sélective : celle issue de leur environnement « fonctionnel » bien sûr (la fonction de *fitness*), mais également une évaluation implicite de la *robustesse* de leur chromosome, c'est-à-dire de la manière dont leur comportement est *construit*.

Nous pouvons apporter deux arguments dans ce sens. Le premier consiste à observer ce qui se passe expérience par expérience, en particulier dans les cas d'insertions et de délétions, comme par exemple sur la figure 5.8 : certains individus ont trouvé assez rapidement un comportement adéquat, mais celui-ci ne s'est pas maintenu dans la population.

Le second vient de l'examen qualitatif des comportements adéquats produits. L'analyse des résultats expérimentaux montre que deux stratégies sont mises en œuvre pour produire un comportement adéquat (donnant la *fitness* maximale). La première consiste à construire un ATN simple (proche de l'ATN « optimal » de la figure 5.3), en retardant, lors de l'expression génétique, la création des nœuds, c'est-à-dire en utilisant des lexèmes sans effet (par exemple en plaçant au début de la séquence de lexèmes des instructions de manipulation de la liste qui sont ignorées, la liste étant vide). Ainsi, un des agents possède un chromosome de 207 bits, codant les lexèmes

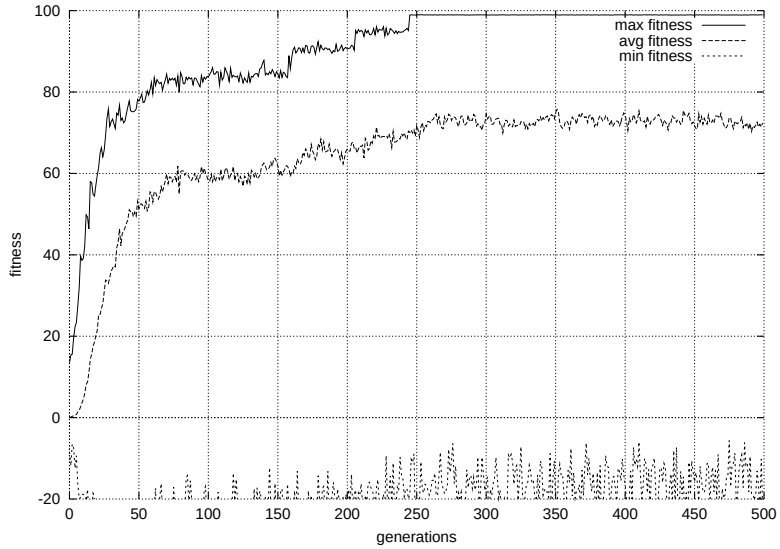

FIG. 5.4 – *Évolution moyenne de la fitness (mutations ponctuelles seulement, $r = 1$ %). Chromosomes initiaux de 200 à 300 bits (moyenne sur 20 expériences).*

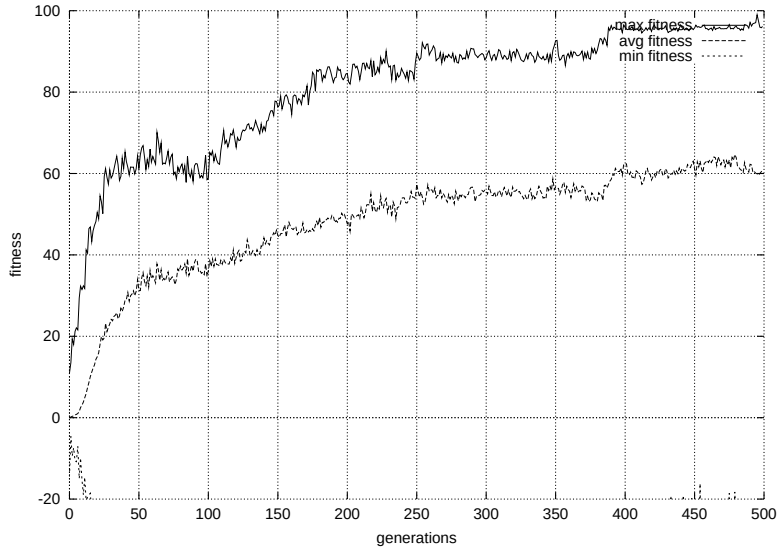

FIG. 5.5 – *Évolution moyenne de la fitness (insertions et délétions aléatoires de codons, $r = 1$ %, $p = 20$ %). Chromosomes initiaux de 200 à 300 bits (moyenne sur 20 expériences).*

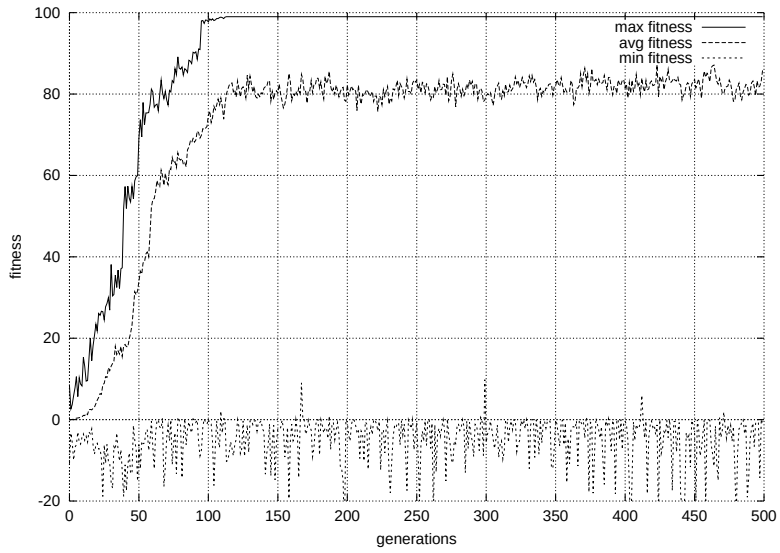

FIG. 5.6 – *Évolution moyenne de la fitness (mutations ponctuelles seulement, $r = 1$ %). Chromosomes initiaux de 500 à 700 bits (moyenne sur 10 expériences).*

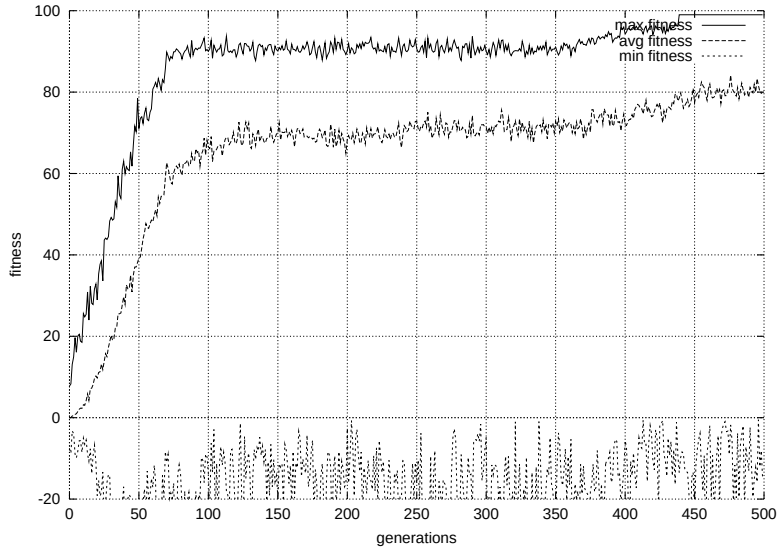

FIG. 5.7 – *Évolution moyenne de la fitness (insertions et délétions aléatoires de codons, $r = 1$ %, $p = 20$ %). Chromosomes initiaux de 500 à 700 bits (moyenne sur 10 expériences).*

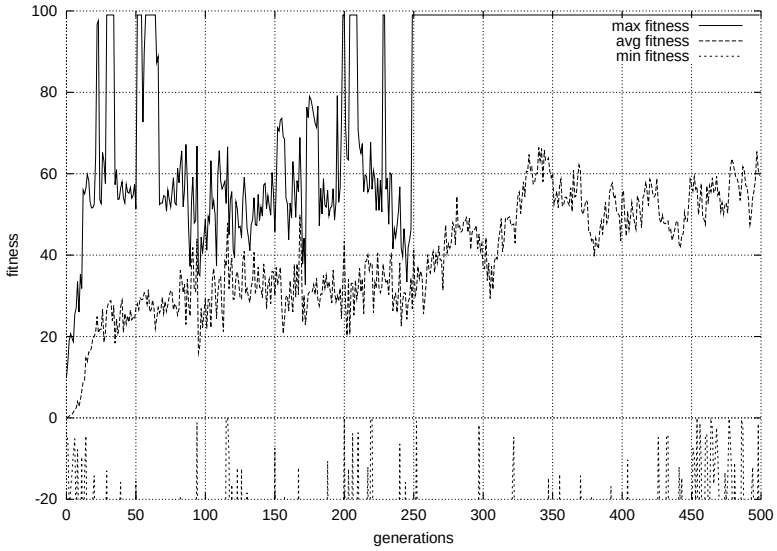

FIG. 5.8 – *Évolution de la fitness (insertions et délétions aléatoires de codons, $r = 1$ %, $p = 20$ %). Chromosomes initiaux de 200 à 300 bits.*

suivants :

> `L!`, *dup*, *swap*, *popRoll*, *popRoll*, *forget*, *popRoll*, *swap*, *forget*, *recall*, *recall*, `R!`, `L!`, *recall*, `rand?`, `L!`, *forget*, *startNode*, `R!`, *dup*, `g?`, *dupObject*, *popRoll*, *pushRoll*, `L!`, *forget*, *pushRoll*, `L!`, `r?`, *startNode*, *forget*, *dup*, *dupObject*, `r?`, `R!`, *forget*, *dup*, `R!`, *dup*, `g?`, *popRoll*

Ces lexèmes produisent un ATN presque identique à celui de la figure 5.3, avec un arc étiqueté (`r?`, `L!L!R!R!`) et l'autre (`g?`, `R!R!`). C'est un exemple intéressant de l'utilisation des propriétés du langage de construction de l'ATN pour aboutir à une structure simple *malgré* la complexité potentielle du graphe.

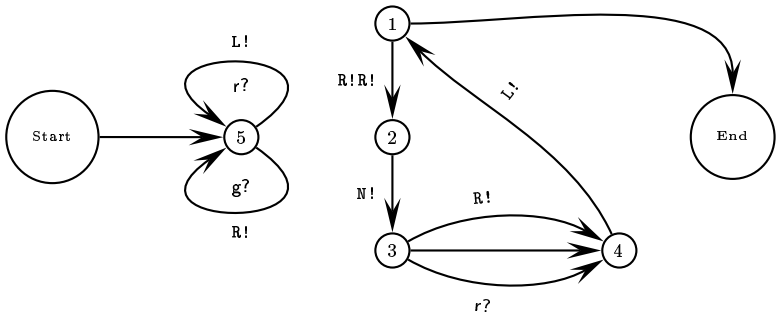

FIG. 5.9 – *Un ATN construit par sélection naturelle, donnant un comportement adéquat. Cet ATN a été construit dans une expérience avec insertions-délétions (chromosomes initiaux de 200 à 300 bits).*

Les agents utilisant la seconde stratégie construisent un ATN relativement complexe mais dont seule une petite partie est réellement utilisée ; pour cela, plusieurs solutions sont envisageables : le graphe peut compter plusieurs composantes connexes (comme celui de la figure 5.9), ou les arcs partant des nœuds « utiles » vers des nœuds superflus sont étiquetés par des conditions irréalisables (telles que la conjonction {`g?`,`r?`}). Cet ATN conduit donc exactement au même comportement

que la solution « optimale », mais avec cette fois une utilisation des propriétés du graphe lui-même.

On comprend mieux, ainsi, le rôle positif des insertions et délétions aléatoires de codons. Ces opérations ne sont pas en elles-mêmes plus nuisibles que de simples mutations, puisque notre langage de construction d'ATN ne dépend que faiblement et localement de l'ordre des instructions. Mais en outre, elles peuvent même *faciliter* l'adaptation aux pressions sélectives. En effet, si l'on s'intéresse à l'évolution de la taille des chromosomes *au cours des expériences*, on constate une augmentation (autrement dit, il existe un biais sélectif en faveur des chromosomes plus longs), qui est d'autant plus nette que les chromosomes de départ sont courts (cf. figures 5.10 à 5.12). Il semble, pour les populations à chromosomes longs, qu'on obtienne une stabilisation autour de 650 bits (voir notamment l'expérience de la figure 5.13).

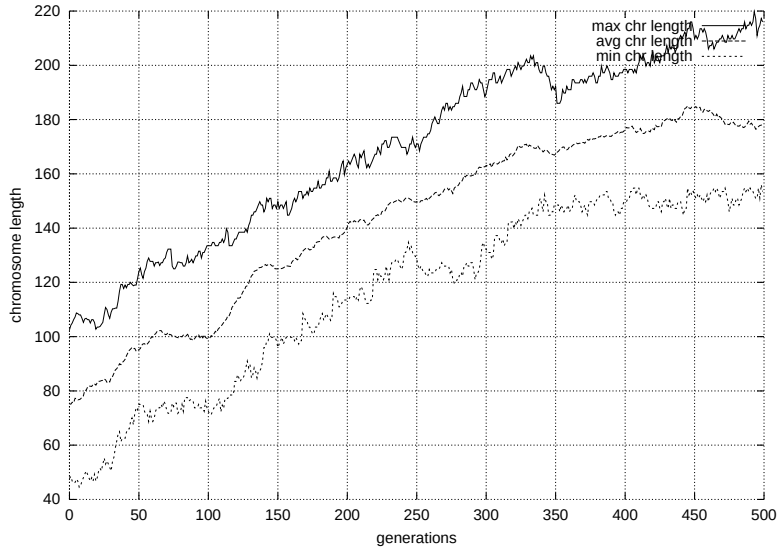

FIG. 5.10 – *Évolution moyenne de la taille des chromosomes (insertions et délétions aléatoires de codons, $r = 1$ %, $p = 20$ %). Chromosomes initiaux de 50 à 100 bits (moyenne sur 10 expériences).*

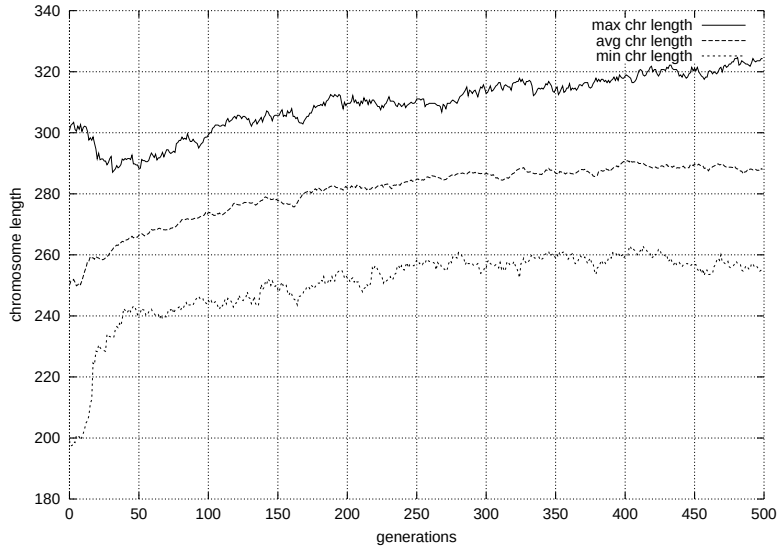

FIG. 5.11 – *Évolution moyenne de la taille des chromosomes (insertions et délétions aléatoires de codons, $r = 1$ %, $p = 20$ %). Chromosomes initiaux de 200 à 300 bits (moyenne sur 20 expériences).*

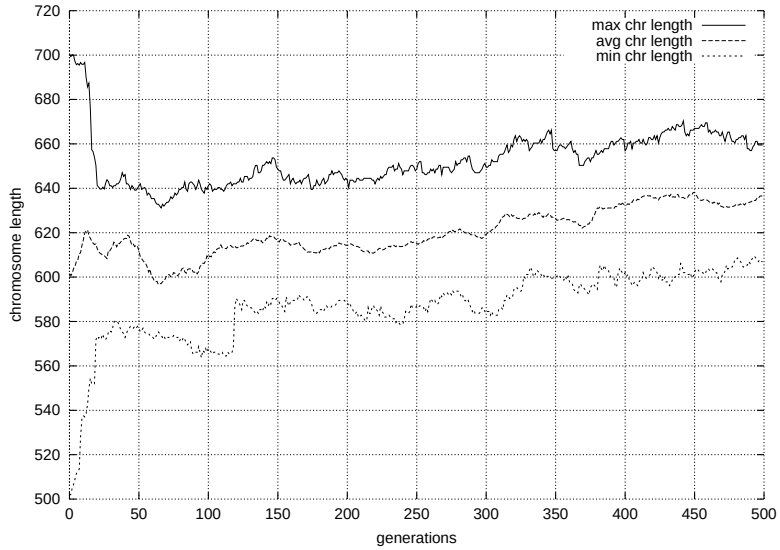

FIG. 5.12 – *Évolution moyenne de la taille des chromosomes (insertions et délétions aléatoires de codons, $r = 1$ %, $p = 20$ %). Chromosomes initiaux de 500 à 700 bits (moyenne sur 10 expériences).*

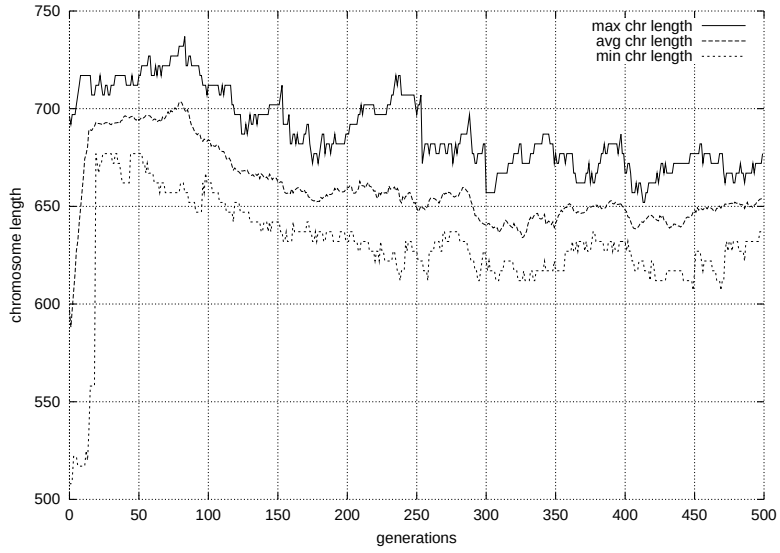

FIG. 5.13 – *Évolution de la taille des chromosomes au cours d'une expérience, montrant un cas de diminution (insertions et délétions aléatoires de codons, r = 1 %, p = 20 %). Chromosomes initiaux de 500 à 700 bits (moyenne sur 10 expériences).*

5.2 Expériences en cours

5.2.1 Parcimonie du modèle

Cette expérience en simulation a pour objectif de tester la capacité du modèle à générer des comportements les plus simples possibles au vu de la tâche à accomplir, même si celle-ci est relativement complexe. Ce principe de parcimonie guide nos recherches lors de la conception de comportements pour des agents [43]. Il nous paraît essentiel de vérifier que, de la même façon qu'il serait à l'œuvre dans le processus de la sélection naturelle, il le soit aussi dans notre modèle. S'il devait échouer, cela montrerait que nous avons involontairement introduit un biais (gênant) dans notre modèle.

Protocole expérimental

L'environnement est un labyrinthe à cases. On définit un agent `Rat` disposant des lexèmes de condition et d'action décrits dans le tableau 5.2. Le comportement souhaité consiste à sortir du labyrinthe en partant de l'entrée, le plus rapidement possible. À chaque pas de temps le `Rat` peut effectuer une des actions parmi : avancer d'une case, tourner sur place d'un quart de tour vers la gauche ou tourner d'un quart de tour vers la droite. Les conditions sont des tests sur les 4 cases nord, sud, est et ouest autour de celle où se trouver l'agent. Dans toutes les expériences, le `Rat` est initialement placé face à une paroi pour le forcer à devoir prendre au moins décision pour gagner des points.

De même que pour l'expérience du `Piéton`, il faut au moins 32 codons ; la création du code génétique et des chromosomes des individus initiaux sont faits dans les mêmes conditions que précédemment.

Actions		Conditions	
`A!`	aller en avant	`a?`	vrai si la case en avant est libre
`R!`	tourner d'un quart de tour vers la droite	`r?`	vrai si la case à droite est libre
`L!`	tourner d'un quart de tour vers la gauche	`l?`	vrai si la case à gauche est libre
		`b?`	vrai si la case en arrière est libre

TAB. 5.2 – *Lexèmes d'actions et de conditions d'un* `Rat`.

Pour toutes les expériences, un temps limité est donné, strictement supérieur au nombre de pas de temps minimal pour sortir du labyrinthe. Une évaluation se termine lorsque le `Rat` a atteint la case de sortie, ou lorsque le temps imparti est écoulé. Les évaluations sont faites chacune sur 3 essais dont les notes sont ajoutées, afin de moyenner les comportements exprimé pour débruiter le non-déterministe.

La fitness est calculée en fin d'évaluation, à l'aide d'un gradient décroissant g diffusé à partir de la sortie. Sa valeur y est alors la distance de Manhattan entre la sortie et l'entrée du labyrinthe. Les obstacles (parois) ne sont pas pris en compte. De ce fait, le plus souvent, le chemin depuis l'entrée vers la sortie du labyrinthe traverse beaucoup de minima et maxima locaux du gradient. Le calcul de la fitness f se fait comme suit :

$$f = g(p) + t$$

où p est la position finale et $t \geq 0$ le temps imparti restant.

Cette fitness présente l'intérêt de ne pas être une fonction strictement croissante de la proximité à la sortie, ainsi le problème bien que simple n'est pas non plus trivial. Selon la forme et la taille du labyrinthe on peut même le compliquer à loisir, en introduisant volontairement beaucoup de minima et maxima locaux dans le paysage de fitness.

FIG. 5.14 – *Petit labyrinthe. L'entrée est en haut.*

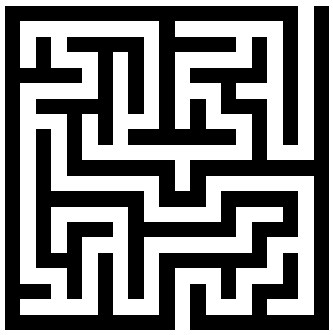

FIG. 5.15 – *Labyrinthe moyen. L'entrée est en haut.*

Plusieurs expériences sont menées :
- en faisant varier la taille du labyrinthe (pour l'instant 4×4, *cf.* figure 5.14 et 19×19, *cf.* figure 5.15),
- en faisant varier la stratégie de remplacement d'une génération (stratégie « élitiste » qui conserve tels quels les parents, ou alors une stratégie qui mute systématiquement tous les individus, similaire à celle qui a été utilisée pour le `Piéton`),
- en faisant varier les taux de mutation et d'insertion-délétion. À la différence des expériences préliminaires, ces deux opérateurs sont systématiquement appliqués successivement,
- en faisant varier la taille des individus initiaux, générés aléatoirement (400, 600 et 800 bit).

Premiers résultats

Avec le petit labyrinthe, des solutions optimales au sens du parcours du labyrinthe sont trouvées presque immédiatement quelle que soit la stratégie : le `Rat` atteint la sortie en un minimum de pas de temps. Ceci se retrouve sur quelques dizaines d'expériences, avec des graines aléatoires initiales différentes, et pour divers taux de mutation et insertion-délétion (chacun variant dans une fourchette entre 0.1% et 5%). Seules quelques expériences n'ont pas convergé, lorsque les taux étaient trop forts et que la stratégie mutait tous les individus.

Des solutions originales comme pour le piéton ont été observées : utilisation d'instructions ignorées car nécessitant des arguments sur la pile avant de construire la solution décrivant le parcours optimal, ou solution construisant un sous-graphe qui ne peut pas être atteint.

On observe que presque toutes les expériences voient la taille du chromosomes se réduire à un même minimum de 125 bit si on laisse les calculs continuer bien que la fitness maximale soit atteinte. La solution, très compacte, exploite des propriétés de la construction du graphe et de son utilisation (en particulier, si aucun arc n'est éligible, l'action précédemment exécutée l'est à nouveau bien qu'aucun changement d'action n'ait été décidé). Ce qui est remarquable est qu'il n'y a plus de critère *explicite* de sélection, puisque tous les meilleurs individus ont tous la fitness maximale. La sélection est *implicite* et se fait sur la structure du chromosome lui-même, en particulier sa longueur, comme nous l'avions déjà observé dans les expériences avec le `Piéton`. Ceci est certainement le fait d'un jeu statistique sur les croisements de parties utiles (des modules du graphe) couplé à l'opérateur de mutation/insertion-délétion qui peut modifier la taille du chromosome.

Avec le labyrinthe moyen, dans un premier temps, les expériences ne convergent pas vers un résultat satisfaisant : le `Rat` reste coincé dans un minimum local au bout du premier couloir. Dans un second temps, l'ajout de conditions supplémentaires (*cf.* tableau 5.3) a permis de résoudre ce problème. En effet, grâce aux négations des conditions il devient possible de discriminer des situations.

Avec ces conditions supplémentaires, nous avons obtenu des individus capables de sortir du labyrinthe plus rapidement que le temps imparti, en moyenne ils ne nécessitent que 20 pas de

Conditions supplémentaires	
$\bar{a}?$	vrai si la case en avant est occupée
$\bar{r}?$	vrai si la case à droite est occupée
$\bar{l}?$	vrai si la case à gauche est occupée
$\bar{b}?$	vrai si la case en arrière est occupée

TAB. 5.3 – *Lexèmes de conditions supplémentaires d'un* `Rat`.

temps en plus des 155 nécessaires. Le comportement trouvé correspond à un parcours « main gauche » du labyrinthe, les individus ne prennent pas les longues voies sans issue sur la droite. L'agent entre dans les petits cul-de-sac sur la gauche, et arrivé au fond il est capable de faire demi-tour. Certaines solutions que nous avons obtenues ne sont pas déterministes, la probabilité y est non nulle qu'à peine sorti l'agent retourne dans le cul-de-sac. C'est dans ces petits culs-de-sac que les meilleurs individus perdent les 20 pas de temps en plus du parcours optimal.

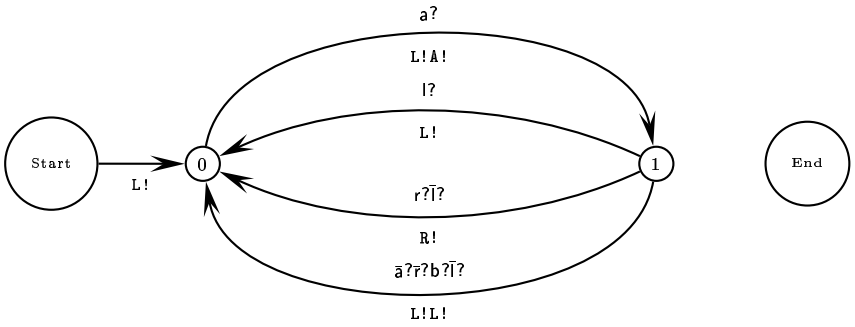

FIG. 5.16 – *Un ATN construit par sélection naturelle, donnant un comportement presque optimal. Lorsque l'on franchit un arc, c'est la dernière action de la liste qui est exécutée. En l'absence de franchissement d'arc (aucun arc n'étant éligible), c'est la dernière action décidée qui est effectuée à nouveau.*

D'autre part, les graphes obtenus sont très petits (*cf.* par exemple la figure 5.16). Il n'y a pas plus de deux ou trois nœuds, et la sortie est rarement atteignable. Enfin, les stratégies et les tailles de départ donnent des solutions similaires dans le comportement mais génétiquement très différentes. Lorsque l'on applique systématiquement la mutation/insertion-délétion, la taille des chromosomes augmente et diminue cycliquement d'un facteur 2 au fil des générations, alors que lorsque les parents sont conservés intacts, la taille des chromosomes finit par converger vers un minimum à chaque fois différent, de l'ordre de 220 bit.

Avec l'exemple typique de la figure 5.16, on comprend aisément la stratégie appliquée. Avant de l'expliquer, remarquons qu'elle est déterministe, il n'y a pas d'arcs partants d'un même nœud qui pourraient être simultanément éligibles. Le non-déterminisme a cependant contribué, au fil des générations, à trouver la solution, en apportant plus de liberté au comportement.

Détail du comportement

L'arc entre Start et 0 est nécessaire car au départ le `Rat` est orienté vers la gauche, face à la paroi (*cf.* figure 5.15), pour le forcer à « prendre une première décision ». L'agent se trouve alors orienté vers le bas, face au long couloir. La stratégie se décode comme suit :

- en 0 : si la case en avant est libre, alors avancer (sinon, implicitement, répéter la dernière action décidée et ne pas franchir l'arc. Ce comportement implicite est exploité par exemple pour faire demi-tour au fond des culs-de-sac).
- en 1 :
  - si la case à gauche est libre alors tourner à gauche (ce qui revient à emprunter un couloir

à gauche dès qu'il s'en présente un)
- si la seule case libre est derrière, alors tourner à gauche (c'est ce qui permet de se sortir des culs-de-sac).
- si la case à droite est libre, mais pas à gauche, alors tourner à droite (c'est ce qui permet de franchir les coudes vers la droite, et de faire les bons choix lorsqu'il y a une alternative entre continuer tout droit et prendre un couloir à droite).
- sinon, à nouveau, implicitement répéter la dernière action (ce qui permet par exemple d'avancer dans les couloirs).

Premières analyses

Les expériences sur le labyrinthe de taille moyenne sont plus significatives. Grâce à elles nous sommes optimistes quant à la capacité effective du modèle à générer des comportements aussi simples que possibles. Dans le cas du labyrinthe de taille moyenne utilisé, les individus exploitent le biais du labyrinthe pour les choix vers la gauche qui excluent les longs cul-de-sac. Nous n'avons pas pu obtenir de comportement parfaitement optimal (temps de parcours minimal, sans « hésitation » dans les petits culs-de-sac sur la gauche). Ceci est peut-être du au fait que le gain est trop faible pour le coût génétique qu'il représente (mieux spécialiser la discrimination des arcs, voire ajouter un nœud).

L'hypothèse émise dans les expériences préliminaires sur l'évolution de la taille du chromosome semble à nouveau confirmée. La pression s'exerce de façon explicite sur le comportement par la fitness, mais s'exercerait aussi de façon implicite par la longueur de la chaîne de bit et la diffusion dans la population de morceaux de chaîne codant pour des modules utiles à la résolution de la tâche. Cette pression implicite sur la dimension de la chaîne concourrait à simplifier le comportement produit par le graphe.

D'autres expériences en cours se font d'une part sur des labyrinthes ayant une autre forme, et d'autre part en évaluant l'individu sur plusieurs labyrinthes différents afin d'observer si le `Rat` peut trouver une solution commune – vraissemblablement sous-optimale, mais efficace tout de même.

5.2.2 Distribution du modèle

Cette expérience en simulation a pour objectif de tester la capacité du modèle à générer des comportements pour un ensemble d'agents en interaction, et de vérifier qu'il est possible de recréer une dynamique d'écosystème.

Cet ensemble d'expériences est réalisé dans le cadre d'un stage de maîtrise par Julien Poudade.

Protocole expérimental

L'environnement est un monde continu et torique. On définit un ensemble d'agents `Proie` et `Prédateur`. Les membres de chaque espèce ont peuvent se nourrir et se reproduire entre eux lorsqu'ils ont un certain niveau d'énergie – atteint par l'alimentation, et dépensé lors des déplacements et de la reproduction. Tous sont dotés de capacité de perception visuelles élémentaires.

Les travaux en cours n'ont pas encore donné de résultats présentables sommairement.

Chapitre 6

Discussion

6.1 Éthogénétique

L'Éthogénétique est mise à l'épreuve par les modèles évalués qui s'y conforment. Il s'agit d'un ensemble de principes pour la construction automatique de comportements d'agents évoluables. Sa principale conséquence est d'ordre méthodologique [107], notamment pour la conception de systèmes multi-agents dans ce que Kupiec et Sonigo ont appelé un processus « d'ontophylogénèse » [75].

En effet, l'Éthogénétique prend tout son sens considérée dans un contexte multi-agent. Lorsque l'objectif n'est pas de reproduire explicitement une organisation précise pour un système mais de concevoir un comportement collectif adaptatif, prédéfinir une organisation est une contrainte qui peut gêner l'adaptation du système à son environnement. C'est d'autant plus probable que le système devra se réorganiser pendant son fonctionnement pour maintenir son adaptation à son environnement ou à la tâche collective. Dans un tel contexte, l'organisation n'est pas une qualité préexistante du système, mais plutôt l'expression d'un processus dynamique permanent [30]. Nous préférons donc considérer l'organisation du système comme une propriété émergente reflétant l'équilibre dans un écosystème [110] (coordination, coopérations entre agents, compétition pour des ressources...).

Le cycle de conception d'un système se décompose alors en tester les comportements d'agents dans *l'écosystème* formé par le système multi-agent et évaluer le comportement adaptatif du système vu comme un *organisme* se développant dans son environnement. L'Éthogénétique a été forgée avec l'espoir de donner un cadre théorique sélectionniste satisfaisant pour ce cycle de conception.

6.2 Expression de gènes à l'aide d'une pile

Nous allons discuter de ce modèle d'évolution de structures selon les 3 axes abordés section 2.3.4 : la syntaxe (section 6.2.1), la sémantique (section 6.2.2) et la complétude (section 6.2.3).

6.2.1 Syntaxe

Représentation dynamique par une chaîne

Comme la plupart des approches pour l'évolution de structures, l'expression par pile utilise comme génotype une représentation dynamique [5, 6]. C'est une propriété nécessaire, puisque comme nous ne connaissons rien par avance de la forme structure la plus adaptée, nous ne connaissons pas non plus la longueur de la chaîne nécessaire à sa fabrication.

Cependant, alors que dans les autres modèles la taille de la représentation est limitée par le concepteur (et non uniquement par la mémoire disponible lors du calcul), par exemple la profondeur des arbres en programmation génétique [72], ou la longueur de la chaîne pour les diverses approches de programmation génétique par pile [105, 120], nous n'avons pas eu besoin d'ajouter

de telles contraintes à notre modèle. Nous avons observé des convergences (ou des fluctuations périodiques) dans des limites finies raisonnables, donc nous n'avons pas eu besoin de contraindre la taille maximale de la représentation génétique.

Nous sommes en train d'étudier de plus près cette propriété. Nous pensons que c'est lié à la propriété de localité, et qu'un équilibre est trouvé entre une tendance à l'allongement et une tendance au raccourcissement. Notre hypothèse actuelle est que :

- si la longueur de la chaîne est trop courte, l'impact d'une mutation ou d'une insertion-délétion est statistiquement plus grand. Dans ces conditions le génotype ne contient pas beaucoup de « code poubelle » pour se protéger des variations. Par analogie avec le modèle biologique, « code poubelle » désigne ici les morceaux de chaîne qui ne seront pas pris en compte dans le phénotype, par exemple des instructions ignorées, ou construisant une partie inutilisée de la structure.
- si la longueur de chaîne est trop longue, il devient d'autant plus difficile pour l'opérateur de croisement de faire des échanges de matériel génétique qui s'avèrent phénotypiquement efficaces. Chaque sous-chaîne échangée est susceptible de porter une partie conséquente de la structure finale (grâce à la propriété de localité), puisque les sous-chaînes échangées deviennent statistiquement plus longues et donc de plus en plus indépendantes et signifiantes après décodage. Ainsi donc, au fur et à mesure de l'allongement des chaînes, le croisement deviendrait statistiquement un opérateur destructeur et inefficace car il juxtaposerait des sous-parties de structure incompatibles.

Cet ajustement automatique de la longueur des chaînes agirait comme une contrainte implicite sur les structures produites. Nous n'aurons donc vraisemblablement pas besoin d'introduire de biais pour ce faire dans les opérateurs de recherche, comme c'est le fait de la plupart des autres approches évolutionnistes.

Séparation entre la syntaxe et la sémantique

Comme la syntaxe du génotype n'est pas intimement liée à la sémantique de la structure, le modèle d'expression à pile est encore moins contraint que par exemple les encodages indirects basés sur une grammaire, ou que la programmation génétique. Avec le modèle d'expression à pile il n'est pas nécessaire de contraindre la syntaxe comme dans d'autres approches (par exemple le principe de clôture [72] ou au contraire un typage fort [99] en programmation génétique. Ceci autorise une plus grande flexibilité pour les opérateurs génétiques. La chaîne de bit peut être manipulée par un processus « aveugle » sans avoir à considérer son interprétation et nous n'avons pas besoin d'opérateurs spécifiquement liés au type de structure qui est évolué.

De plus, comme la chaîne de bit n'a pas de signification et que n'importe quelle chaîne peut être interprétée en une structure valide, n'importe quel opérateur génétique importé des algorithmes génétiques peut être appliqué sans restriction. Nous avons même utilisé un opérateur d'insertion-délétion de codons qui à notre connaissance n'a jamais été utilisé pour des algorithmes génétiques. Cet opérateur en particulier permet de gérer les changement de dimension de la chaîne de bit de façon plus douce que les croisements.

Redondance

La description des différentes parties de la structure n'est pas dépendante de la position dans la chaîne de bit, plusieurs combinaisons de sous-chaînes peuvent produire la même structure. Ce qui parait plus important sont les positions et distances *relatives* des lexèmes impliqués dans la construction d'une partie de la structure, car les lexèmes interagissent localement via la pile. De plus, l'interprétation de certains lexèmes peuvent ne pas prendre en compte l'ordre des lexèmes avec lesquels ils interagissent dans la pile. Par exemple dans ATNoSFERES, le lexèmes `connect` ne pose de contrainte ni sur l'ordre des conditions utilisées pour étiqueter un arc (c'est un *ensemble* de conditions), ni sur l'ordre relatif entre les conditions et actions rencontrées sur la pile (les premières sont mise dans un ensemble, les secondes dans une liste). Cette redondance dans l'interprétation et le fait de pouvoir obtenir une même structure à partir de plein de chaînes différentes augmentent

le champ des possibles du modèle et donc sa flexibilité et son efficacité (puisque le processus sélectionniste s'appuie sur le hasard et donc les statistiques).

Il y a un autre type de redondance, celle du code génétique. En fonction du nombre de lexème disponibles, le code génétique peut être plus ou moins redondant (le code génétique est une surjection). Si nécessaire, il peut être conçu de façon à offrir plus de résistance aux mutations (en choisissant de façon adéquate les codons) ou de façon à encourager les instructions de construction de structure (en augmentant la proportion de codons de manipulation de pile ou de structure).

6.2.2 Sémantique

Lien comportemental fort entre les parents et leur descendance

Nous avons supposé que nous utiliserions un langage de construction de structure à granularité fine. La localité des actions des atomes du langage et leur granularité fine pour la construction de la structure assurent que les manipulations locales ont un petit impact la plupart du temps, donc les structures de la descendance sont le plus souvent produites à partir de faible variations de leurs parents. Ce lien fort entre le comportement exprimé par les parents et celui exprimé par leur descendance est une propriété importante, c'est elle qui induit la quasi-continuité de l'encodage. Comme nous l'avons écrit précédemment, cette quasi-continuité adoucit le paysage de fitness et facilite la convergence vers des structures adaptées à la pression sélective.

Adaptation cumulative

Grâce aux petites variations du génotype et de la quasi-continuité comportementale entre parents et descendants, l'adaptation des individus au fil des générations croît de façon cumulative par de faibles variations. Cette continuité depuis les variations jusqu'aux comportements exprimés est pour nous un axe principal de conception par évolution artificielle. La continuité permet de guider la recherche de solutions adaptées en s'appuyant sur de l'aléatoire, mais sans pour autant faire de la marche au hasard. Le processus ne requiert dès lors ni biais ni finalité.

6.2.3 Complétude

Pouvoir d'expression

Le langage de construction de graphe que nous proposons pour ATNoSFERES permet par exemple de construire un graphe *quelconque* du type que nous avons choisi. À partir d'un graphe il est toujours possible de déterminer au moins une chaîne de bit qui permet de le construire. Nous pouvons donc explorer tout l'espace des structures.

Notre approche permet de plus de régler finement les structures. Pour répondre à l'une des problématiques posées dans [7], nous avons là un bon candidat pour le compromis entre le pouvoir d'expression et la finesse du réglage des structures construites.

Biais limité des opérateurs de recherche

Comme tout opérateur génétique est indépendant de la structure construite, en utilisant notre modèle simple et uniforme d'interprétation, nous nous assurons que nous pouvons introduire aussi peu de biais que possible dans les opérateurs de recherche : l'exploration de l'espace des structures peut se faire avec des opérateurs classiques, génériques ou rares.

Il est cependant nécessaire de déterminer le code génétique – qui influence les probabilités d'occurrence de chaque lexème. Cela n'a pour l'instant pas semblé crucial dans nos expériences, mais ce point paraît suffisamment important pour être étudié, puisque c'est la façon principale d'introduire du biais de façon explicite pour un langage donné.

6.3 ATNoSFERES

ATNoSFERES est un instanciation du modèle d'expression à pile pour des graphes similaires à des ATN. Le choix d'automates décrits par des graphes a été motivé par le fait que c'est une structure des plus classiques en informatique, compréhensible par tous. Ceci permet de concevoir ou tenter d'améliorer manuellement des solutions, puis de les soumettre à nouveau à la machine.

Le non-déterminisme introduit certes du bruit lors des évaluations. C'est de toute façon souvent le cas lorsque l'on évalue un comportement d'agent situé (le comportement est le résultat d'un couplage de l'agent plongé dans son environnement), et toutes les évaluations en robotique évolutionniste notamment se font sur plusieurs tests d'un même génome.

L'utilisation non-déterministe nous a paru être un plus. Elle est intéressante parce qu'elle confère plus de liberté au comportement qui, dans la même situation, peut ne pas s'exprimer de la même façon. Des solutions intermédiaires lors de nos expériences montrent que le processus d'évolution exploite beaucoup cette propriété : certains arcs d'un nœud vers un autre sont ainsi représentées plusieurs fois presque à l'identique, ce qui augmente en leur faveur le biais statistique au moment de l'élection de l'arc parmi ceux éligibles. Ainsi au fil des générations les individus « apprennent » à choisir statistiquement une voie plutôt qu'une autre. Lorsque les solutions s'améliorent globalement, et deviennent donc plus homogènes pour ce choix, la contrainte implicite sur la taille du génome qui semble à l'œuvre pourrait réduire le nombre d'arcs qui se recoupent pour ces graphes, n'en retenant finalement qu'un seul.

D'autre part, ATNoSFERES permet de décrire tant des comportements réactifs que des comportements cognitifs. Il n'est pas nécessaire de faire d'hypothèse sur le comportement que l'on désire obtenir, les solutions trouvées sont simples (pas forcément explicitement optimales), et ce de manière suffisante. Un comportement purement réactif est modélisé par un graphe à nœud unique (en dehors de Start et End) qui n'a d'arcs que vers lui-même. Un comportement cognitif est modélisé par un graphe à plusieurs états, c'est-à-dire contenant plusieurs nœuds connexes. Dans l'exemple du labyrinthe de taille moyenne par exemple, les solutions sont cognitives, et reflètent qu'il est par exemple nécessaire pour l'agent de savoir quels couloirs emprunter sur la droite, alors que certains mènent à une longue voie sans issue.

6.4 Expériences prévues

Les résultats des expériences déjà menées ont confirmé les prédictions faite lors de l'établissement de nos principes. Les expériences en cours sont tout aussi encourageantes pour ce qui est des principes, mais il semblerait que techniquement quelques ajustements doivent être apportés à leur instanciation, au modèle ATNoSFERES. Ces réglages seront validés à l'aide des expériences en cours et de celles prévues, et ne dérogent en rien aux principes Éthogénétique, les modifications ne portent que sur la façon d'utiliser le graphe.

Les expériences que nous prévoyons pour la dernière année de la thèse vont nous permettre de tester ATNoSFERES sur des problème individuels et collectifs avec des robots. Il s'agira de déterminer si :

- le modèle est utilisable pour un robot autonome. Nous expérimenterons l'apprentissage et la composition de plusieurs comportements.
- le modèle est utilisable pour une collectivité de robots autonomes. Nous expérimenterons l'apprentissage d'une tâche collective de déploiement spatial, vraisemblablement de la navigation en formation.

Chapitre 7

Conclusion

Nous résumons la démarche, puis inscrivons nos travaux dans la perspective de la génétique d'agents. Cette perspective est un retournement par rapport aux approches classique en informatique évolutionniste. Au lieu de ne considérer que la population d'individus (en l'occurrence un individu est le système multi-agent vu comme un organisme), et de centraliser l'évolution de cette population comme dans les algorithmes évolutionnistes classiques, nous laissons chaque agent libre dans le système qui évolue alors de lui-même. Nos travaux ont vocation à s'inscrire dans cette ligne.

7.1 Une approche sélectionniste pour les SMA situés

Nous nous intéressons aux collectivités de robots en environnement réel car c'est un exemple de système multi-agent situé, et que c'est un problème riche en questions scientifiques, notamment pour ce qui a trait aux systèmes dynamiques et complexes. Les problèmes importants de conception nous ont fait percevoir l'intérêt des approches évolutionnistes pour aborder ce type de problèmes, mais les approches existantes ne nous paraissent pas satisfaisantes à tous points de vue. Suite à une synthèse des défauts et qualités de chacune, nous avons déterminé un ensemble de principes à respecter pour concevoir automatiquement des comportements d'agents évoluables, l'Éthogénétique. L'élaboration d'un nouvel encodage génétique basé sur un interprète à pile nous a permis de dessiner une nouvelle approche pour l'évolution de structures, qui s'inscrit dans les principe Éthogénétiques. Puis nous avons instancié cette approche en un modèle appliqué à l'évolution de graphes étiquetés, ATNoSFERES. C'est en menant des expériences avec ce modèle que nous testons la pertinence et la faisabilité de l'Éthogénétique.

7.2 Vers la génétique d'agents

Ces travaux et les outils conceptuels et pratiques forgés pour un besoin multi-agent spécifique (les systèmes multi-agent situés) s'inscrivent dans un cadre plus large de recherche en génétique d'agents [34]. La génétique d'agent est un modèle multi-agent de sélection darwinienne appliqué aux agents d'un système. Contrairement aux algorithmes évolutionnistes classiques – où un moteur d'évolution centralise les opérations génétiques, ordonnance les évaluations et gère les sélections – il s'agit d'une approche décentralisée et inscrite dans le temps. Chaque agent porte son propre code génétique et est amené à se reproduire ou non avec d'autres agents. L'évaluation et la sélection dans un tel système sont aussi distribuées, elles sont implicites et/ou déléguée à d'autres agents dont c'est le rôle. Ce type d'approche permet donc d'exploiter la dynamique du système en fonctionnement lors de la recherche de solutions, alors qu'une approche centralisée impose une vision beaucoup plus pauvre des possibilités du système. En effet, dans une approche centralisée les comportements des agents sont donnés définitivement pour chaque évaluation, et il faut être capable de calculer l'évaluation du système tout entier.

La génétique d'agents s'est déjà avérée beaucoup plus performante que les algorithmes génétiques classiques pour des résolutions de problèmes multi-objectifs complexes [33]. Cette approche permettrait de modéliser des systèmes complexes à l'aide de systèmes massivement agents par exemple [32, 30].

L'Éthogénétique qui a été conçue pour être appliquée à des systèmes multi-agent situés a donc vocation toute trouvée à servir de cadre théorique pour l'évolution de comportements d'agents en génétique d'agents, et l'expression génétique à pile fournit un langage génétique – la chaîne de bit – aux agents quels qu'ils soient, ce langage pouvant être interprété en des structures puis des comportements de complexité arbitraire comme nous l'avons montré. Un langage commun en génétique d'agents est une façon de s'affranchir des difficultés que pose la décentralisation des reproductions d'agents, et donc des croisements de génotypes. Nous disposons donc d'une sorte de langage génétique universel de la même façon que le vivant a massivement plébiscité la génétique à base d'acides nucléiques.

Calendrier

septembre-décembre 2001 :	expériences en cours et prévues avec un robot seul
	comparaisons expérimentales avec la programmation génétique
18, 19 et 20 octobre :	démonstration éventuelle à la journée « Portes Ouvertes »
janvier-mars 2002 :	expériences prévues avec plusieurs robots
avril-septembre 2002 :	rédaction de la thèse
octobre 2002 :	soutenance de la thèse

Publications

- **Microb III : Prise de décision pour une équipe de robots-footballeurs.**
  S. Landau
  Mémoire de DEA, DEA IARFA, Université Pierre et Marie Curie, Paris, 1999.
- **ATNoSFERES : a Model for Evolutive Agent Behaviors**
  S. Landau, S. Picault et A. Drogoul
  Proc. of AISB'01 symposium on Adaptive Agents and Multi-agent systems, pages 95–99, mars 2001.
- **SFERES : un framework pour la conception de systèmes multi-agents adaptatifs**
  S. Landau, S. Doncieux, A. Drogoul et J.-A. Meyer
  Rapport technique LIP6, 2001.
- **Ethogenetics and the Evolutionary Design of Agents Behaviors**
  S. Picault et S. Landau
  Proc. of the 5th World Multi-Conference on Systemics, Cybernetics and Informatics, vol. iii, pages 528–533, juillet 2001.
- **Modeling Adaptive Multi-Agent Systems Inspired by Developmental Biology**
  S. Landau et S. Picault
  Proc. of the Workshop on Adaptability and Embodiment using Multi-Agent Systems of ACAI'01, pages 238–246, juillet 2001.
- **Ethogenetics: an Evolutionary Approach to Agents Organization**
  S. Picault et S. Landau
  Actes du colloque ALCAA, septembre 2001.
- **Developing Agents Populations with Ethogenetics (article accepté)**
  S. Landau et S. Picault
  Proc. of the Workshop on Radical Agent Concepts, septembre 2001.
- **SFERES : un framework pour la conception de systèmes multi-agents adaptatifs (article soumis)**
  S. Landau, S. Doncieux, A. Drogoul et J.-A. Meyer
  Technique et Science Informatiques, Hermes, 2001.
- **Stack-Based Gene Expression (article soumis)**
  S. Landau et S. Picault
  Genetic Programming and Evolvable Machines, Kluwer, 2001.
- **L'Éthogénétique - Une approche darwinienne du comportement individuel et collectif d'agents (en préparation)**
  S. Picault et S. Landau
  Rapport technique LIP6, 2001.

Bibliographie

[1] AGRE, P. E. *The Dynamic Structure of Everyday Life.* PhD thesis, MIT Artificial Intelligence Laboratory, 1988.

[2] AGRE, P. E., AND CHAPMAN, D. *What Are Plans for?* In Maes [85], 1990, pp. 17–34.

[3] ANDRE, D., AND TELLER, A. Evolving Team Darwin United. In *RoboCup-98: Robot Soccer World Cup II* (Paris, France, july 1999), M. Asada and H. Kitano, Eds., vol. 1604 of *LNCS*, Springer Verlag, pp. 346–351.

[4] ANGELINE, P., SAUNDERS, G., AND POLLACK, J. Complete Induction of Recurrent Neural Networks. In Sebald and Fogel [116], pp. 1–8.

[5] ANGELINE, P. J. Genetic Programming: A Current Snapshot. In Sebald and Fogel [116], pp. 224–232.

[6] ANGELINE, P. J. Genetic Programming and Emergent Intelligence. In *Advances in Genetic Programming*, K. E. Kinnear, Jr., Ed. MIT Press, 1994, pp. 75–98.

[7] ANGELINE, P. J. Morphogenic Evolutionary Computations: Introduction, Issues and Example. In *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming* (Cambridge, Massachusetts, 1995), A Bradford Book, MIT Press, pp. 387–401.

[8] ANGELINE, P. J. An Investigation into the Sensitivity of Genetic Programming to the Frequency of Leaf Selection During Subtree Crossover. In Koza et al. [74], pp. 21–29.

[9] ANGELINE, P. J. A Historical Perspective on the Evolution of Executable Structures. *Fundamenta Informaticae 36*, 1-4 (1998), 179–195.

[10] ANGELINE, P. J., SAUNDERS, G. M., AND POLLACK, J. P. An Evolutionary Algorithm that Constructs Recurrent Neural Networks. *IEEE Transactions on Neural Networks 5*, 1 (January 1994), 54–65.

[11] ARKIN, R. C. Motor Schema Based Navigation for a Mobile Robot: An Approach to Programming by Behavior. pp. 264–271.

[12] ARKIN, R. C. *Behavior-Based Robotics.* MIT Press, Cambridge, Massachusetts, 1998.

[13] ARKIN, R. C., AND BALCH, T. *Cooperative Multiagent Robotic Systems.* 1997.

[14] ARKIN, R. C., AND BEKEY, G. A., Eds. *Robot Colonies.* Kluwer, 1997.

[15] BÄCK, T., AND SCHWEFEL, H.-P. An Overview of Evolutionary Algorithms for Parameter Optimization. *Evolutionary Computation 1*, 1 (1993), 1–23.

[16] BALCH, T., AND ARKIN, R. C. Communication in reactive multiagent robotic systems. In *Autonomous Robots*, vol. 1. 1994, pp. 1–25.

[17] BALCH, T., AND ARKIN, R. C. Motor Schema-Based Formation Control for Multiagent Robot Teams. In *Proc. of the First International Conference on Multiagent Systems (ICMAS'95)* (San Francisco, CA, 1995).

[18] BEASLEY, D., BULL, D. R., AND MARTIN, R. R. An overview of genetic algorithms: Part 1, fundamentals. *University Computing 15*, 2 (1993), 58–69.

[19] BOERS, E. J., AND KUIPER, H. Biological Metaphors and the Design of Modular Artificial Neural Networks. Master's thesis, Dep. CS and Exp. and theor. psych., Leiden University, Netherlands, 1992.

[20] Braitenberg, V. *Vehicles: Experiments in Synthetic Psychology.* MIT Press, 1986.

[21] Bremermann, H. J. Optimization through Evolution and Recombination. In *Self organizing Systems*, Yovits, Jacobi, and Goldstein, Eds. Pergamon Press, Oxford, 1962, pp. 93–106.

[22] Brooks, R. A. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation RA-2*, 1 (1986), 14–23.

[23] Brooks, R. A. *Elephants Don't Play Chess.* In Maes [85], 1990, pp. 3–15.

[24] Brooks, R. A. Intelligence without Reason. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI'91)* (1991), J. Mylopoulos and R. Reiter, Eds., Morgan Kaufmann, pp. 569–595.

[25] Brooks, R. A. The Role of Learning in Autonomous Robots. In *COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers* (1991).

[26] Brooks, R. A. *Cambrian Intelligence.* MIT Press, Cambridge, Massachusetts, 1999.

[27] Brooks, R. A., and Maes, P., Eds. *Proc. of the Artificial Life IV Conference* (Cambridge, Massachusetts, 1994), vol. 4, MIT Press.

[28] Callaos, N., Esquivel, S., and Burge, J., Eds. *Proceedings of the 5th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI'01)* (2001), vol. iii.

[29] Cao, Y. U., Fukunuga, A. S., and Kahng, A. B. Cooperative Mobile Robotics: Antecedents and Directions. In *Autonomous Robots* [14], pp. 7–27.

[30] Cardon, A. *Conscience artificielle & systèmes adaptatifs.* Eyrolles, Paris, 1999.

[31] Cardon, A. The Approaches of the Concept of Embodiment for an Autonomous Robot. Towards Consciousness of its Body. In Mařík et al. [89], pp. 218–229.

[32] Cardon, A. Étude de la conception et du comportement d'une organisation d'agents massive. Article en préparation, 2001.

[33] Cardon, A., Galinho, T., and Vacher, J.-P. A Multi-Objective Genetic Algorithm in Job Shop Scheduling Problem to Refine an Agents' Architecture. In *Proceedings of EUROGEN'99* (Jyväskylä, Finland, 1999), K. Miettinen, M. M. Mäkelä, P. Neittaanmäki, and J. Periaux, Eds., University of Jyváskylä.

[34] Cardon, A., and Vacher, J.-P. Genetic Algorithm using Multi-Objective in a Multi-Agent System. In *Robotics and Autonomous Systems*, no. 33. 2000, pp. 179–190.

[35] *Proceedings of the International Conference on Distributed Autonomous Robotic Systems (DARS'2000)* (2000).

[36] Darwin, C. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life.* 1859.

[37] Dautenhahn, K. Embodiment and Interaction in Socially Intelligent Life-Like Agents. In Nehaniv [100], pp. 105–147.

[38] Dautenhahn, K., Ed. *Socially Intelligent Agents: the Human in the Loop (SIA'2000)* (2000), vol. FS–00–04 of *AAAI Fall Symposium Series*, AAAI Press.

[39] De Jong, K. A. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, Dept. of Computer and Communication Sciences, University of Michigan, 1975.

[40] Demazeau, Y., and Werner, E., Eds. *Decentralized AI 3* (Amsterdam, 1992), Elsevier (North-Holland).

[41] Dennett, D. C. *Darwin's Dangerous Idea. Evolution and the Meanings of Life.* Penguin Books Ltd., London, 1996.

[42] Drogoul, A. *De la Simulation Multi-Agents à la Résolution Collective de Problèmes.* Thèse de doctorat, Université Paris VI, 1993.

[43] Drogoul, A. *Systèmes multi-agents situés.* Dossier d'habilitation à diriger les recherches, Université Paris VI, Paris, 2000.

[44] Drogoul, A., and Dubreuil, C. Eco-Problem-Solving Model: Results of the N-Puzzle. In Demazeau and Werner [40], pp. 283–295.

[45] Drogoul, A., Landau, S., and Muñoz, A. RoboCup : un benchmak pour l'IAD? Article en préparation, 2001.

[46] Drogoul, A., and Picault, S. MICRobES: vers des collectivités de robots socialement situées. In Gleizes and Marcenac [53].

[47] Drogoul, A., Tambe, M., and Fukuda, T., Eds. *Proceedings of the First International Workshop on Collective Robotics (CRW'98)* (1998), Springer Verlag, Lectures Notes in AI 1456.

[48] Ferber, J. *Les systèmes multi-agents: Vers une intelligence collective.* InterEditions, Paris, 1995.

[49] Floreano, D., Nolfi, S., and Mondada, F. Competitive Co-Evolutionnary Robotics: From Theory to Practice. In Pfeifer et al. [106].

[50] Fogel, D. B. *Evolving Artificial Intelligence.* PhD thesis, University of California, San Diego, 1992.

[51] Fogel, D. B., and Stayton, L. C. On the Effectiveness of Crossover in Simulated Evolutionary Optimization. In *Biosystems*, no. 32. 1994, pp. 171–182.

[52] Fogel, L. J., Owens, A. J., and Walsh, M. J. *Artificial Intelligence through Simulated Evolution.* John Wiley & Sons, 1966.

[53] Gleizes, M.-P., and Marcenac, P., Eds. *Ingénierie des systèmes multi-agents. Actes des JFIADSMA'99* (Paris, 1999), Hermès.

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

[55] Greenwood, G. W. Training Partially Recurrent Neural Networks Using Evolutionary Strategies. In *IEEE Trans. on Speech and Audio Processing*, vol. 5. 1997, pp. 192–104.

[56] Gruau, F. *Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm.* Thèse de doctorat, ENS Lyon – Université Lyon I, 1994.

[57] Guessoum, Z. *Un environnement opérationnel de conception et de réalisation de systèmes multi-agents.* Thèse de doctorat, Université Paris VI, 1996.

[58] Guessoum, Z. Dima: Une plate-forme multi-agents en smalltalk. *Revue Objet 3*, 4 (1998), 393–410.

[59] Gutknecht, O., and Ferber, J. Madkit: Organizing heterogeneity with groups in a platform for multiple multi-agent systems. Tech. Rep. RR97188, LIRMM, december 1997.

[60] Harnad, S. The Symbol Grounding Problem. In *Physica D*, vol. 42. 1990, pp. 335–346.

[61] Harvey, I., Husbands, P., Cliff, D., Thompson, A., and Jakobi, N. Evolutionary Robotics: the Sussex Approach. *Robotics and Autonomous Systems 20*, 2-4 (1997), 205–224.

[62] Haynes, T., Sen, S., Schoenefeld, D., and Wainwright, R. Evolving a Team. In *Working Notes for the AAAI Symposium on Genetic Programming* (Cambridge, MA, november 1995), E. V. Siegel and J. R. Koza, Eds., AAAI.

[63] Heitkötter, J., and Beasley, D. The Hitch-Hiker's Guide to Evolutionnary Computation (FAQ for comp.ai.genetic), december 1998.

[64] Holland, J. H. Outline for a Logical Theory of Adaptive Systems. *Journal of the ACM 9*, 3 (july 1962), 297–314.

[65] Holland, J. H. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* Ann Arbor: University of Michigan Press, 1975.

[66] Holland, J. H. *Hidden Order: How Adaptation Builds Complexity.* Addison-Wesley, 1996.

[67] Kaelbling, L. P. *Learning in embedded systems.* PhD thesis, Stanford University, 1990.

[68] Kitano, H. Designing Neural Networks Using Genetic Algorithms with Graph Generation System. In *Complex Systems*, vol. 4. 1990, pp. 461–476.

[69] Kitano, H., and Asada, M. RoboCup: a Challenge Problem for AI. *AI Magazine 18*, 1 (1997).

[70] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and awa, E.-I. O. RoboCup: The Robot World Cup Initiative. In *Proc. of IJCAI–95 Workshop* (Menlo Park, CA, 1995), AAAI Press, pp. 41–47.

[71] KODJABACHIAN, J., AND MEYER, J.-A. Evolution and Development of Neural Controllers for Locomotion, Gradient-Following, and Obstacle-Avoidance in Artificial Insects. *IEEE Transactions on Neural Networks 9* (1998), 796–812.

[72] KOZA, J. R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, Massachusetts, 1992.

[73] KOZA, J. R., BENNETT III, F. H., ANDRE, D., AND KEANE, M. A. Automated Design of Both the Topology and Sizing of Analog Electrical Circuits Using Genetic Programming. In *Artificial Intelligence in Design'96* (1996), J. S. Gero and F. Sudweeks, Eds., pp. 151–170.

[74] KOZA, J. R., GOLDBERG, D. E., FOGEL, D. B., AND RIOLO, R. L., Eds. *Genetic Programming 1996: Proceedings of the First Annual Conference* (Stanford University, CA, 1996), MIT Press.

[75] KUPIEC, J.-J., AND SONIGO, P. *Ni Dieu ni gène. Pour une autre théorie de l'hérédité.* Science ouverte. Editions du Seuil, Paris, 2000.

[76] LANDAU, S. Microb III: Prise de décision pour une équipe de robots-footballeurs. Master's thesis, DEA IARFA, Université Pierre et Marie Curie, Paris, 1999.

[77] LANDAU, S., DONCIEUX, S., DROGOUL, A., AND MEYER, J.-A. SFERES. Rapport technique, LIP6, Paris, 2001.

[78] LANDAU, S., DONCIEUX, S., DROGOUL, A., AND MEYER, J.-A. SFERES, un framework pour la conception de systèmes multi-agents adaptatifs. *Technique et Science Informatique* (2001). Article soumis.

[79] LANDAU, S., AND PICAULT, S. Stack-Based Gene Expression. An Approach Based on Ethogenetics Principles. *Genetic Programming and Evolvable Machines* (2001). Article soumis.

[80] LANDAU, S., PICAULT, S., AND DROGOUL, A. ATNoSFERES: a Model for Evolutive Agent Behaviors. In *Proceedings of the AISB'01 Symposium on Adaptive Agents and Multi-Agent Systems* (2001).

[81] LANGTON, C., Ed. *Artificial Life* (London, 1988), vol. 1, Addison-Wesley.

[82] LINDENMAYER, A. Mathematical Models for Cellular Interaction in Development, parts I and II. *Journal of theoretical biology 18* (1968).

[83] LUKE, S. Genetic Programming Produced Competitive Soccer Softbot Teams for RoboCup97. In *Proc. of the Third Annual Conference on Genetic Programming* (University of Wisconsin, Madison, Wisconsin, USA, 22-25 1998), J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, Eds., Morgan Kaufmann, pp. 214–222.

[84] LUKE, S., AND SPECTOR, L. Evolving Graphs and Networks with Edge Encoding: Preliminary Report. In Koza [74], pp. 117–124.

[85] MAES, P., Ed. *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back.* MIT Press, Cambridge, MA, 1990.

[86] MAES, P., AND BROOKS, R. A. Learning to coordinate behaviors. In *Proc. of AAAI-90* (Boston, MA, 1990), vol. 2, AAAI Press/The MIT Press, pp. 796–802.

[87] MAES, P., MATARIĆ, M. J., MEYER, J.-A., POLLACK, J., AND WILSON, S. W., Eds. *From Animals to Animats 4. Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour (SAB'96)* (Cambridge, Massachusetts, 1996), MIT Press.

[88] MAHADEVAN, S., AND CONNELL, J. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55 (1992), 311–365.

[89] MAŘÍK, V., ŠTĚPÁNKOVÁ, O., KRAUTWURMOVÁ, H., AND BRIOT, J.-P., Eds. *Proceedings of the Workshop on Adaptability and Embodiment Using Multi-Agent Systems (AEMAS'2001)* (Prague, 2001), Czech Technical University.

[90] MATARIĆ, M. Designing Emergent Behaviors: From Local Interactions to Collective Intelligence. In Meyer et al. [95], pp. 432–441.

[91] MATARIĆ, M. Learning in Multi-Robot Systems. In *Workshop on Adaptation and Learning in Multi-Agent Systems, 14th International Joint Conference on Artificial Intelligence (IJCAI'95)* (1995), AAAI Press.

[92] MATARIĆ, M. J. Issues and Approaches in the Design of Collective Autonomous Agents. *Robotics and Autonomous Systems 16* (december 1995), 321–331.

[93] MATARIĆ, M. J. Learning in multi-robot systems. In Weiß and Sen [123], pp. 152–163.

[94] MATATIĆ, M. J. Reinforcemnt Learning in the Multi-Robot Domain. In *Autonomous Robots* [14], pp. 73–83.

[95] MEYER, J.-A., ROITBLAT, H. L., AND WILSON, S. W., Eds. *From Animals to Animats 2. Proceedings of the 2nd International Conference on Simulation of Adaptive Behavior* (Cambridge, Massachusetts, 1993), MIT Press.

[96] MEYER, J.-A., AND WILSON, S. W., Eds. *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior* (1991), MIT Press.

[97] MILLER, G. F., TODD, P. M., AND HEGDE, S. U. Designing Neural Networks using Genetic Algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms* (1989), J. D. Schaffer, Ed., Morgan Kaufmann, pp. 65–80.

[98] MITCHELL, M., AND TAYLOR, C. E. Evolutionary Computation: An Overview. *Annual Review of Ecology and Systematics 20* (1999), 593–616.

[99] MONTANA, D. J. Strongly Typed Genetic Programming. In *Evolutionnary Computation*, vol. 3. 1995.

[100] NEHANIV, C. L., Ed. *Computation for Metaphors, Analogy and Agent* (1999), vol. 1562 of *Lectures Notes in Artificial Intelligence*, Springer Verlag.

[101] NILSSON, N. J. A Mobile Automaton: An Application of Artificial Intelligence Techniques. In *Proceedings of the International Joint Conference on Artificial Intelligence* (Washington, D. C., 1969), pp. 509–520.

[102] NOLFI, S., AND FLOREANO, D. *Evolutionary Robotics. The Biology, Intelligence, and Technology of Self-organizing Machines.* MIT Press, Cambridge, Massachusetts, 2000.

[103] NOLFI, S., FLOREANO, D., MIGLINO, O., AND MONDADA, F. How To Evolve Autonomous Robots: Different Approaches In Evolutionary Robotics. In Brooks and Maes [27].

[104] PARKER, L. E. *Heterogeneous Multi-Robot Cooperation.* PhD thesis, MIT, EECS Dept., 1994.

[105] PERKIS, T. Stack-Based Genetic Programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence* (Orlando, FL, 1994), vol. 1, IEEE Press, pp. 148–153.

[106] PFEIFER, R., BLUMBERG, B., MEYER, J.-A., AND WILSON, S. W., Eds. *From Animals to Animats 5. Proceedings of the 5th International Conference on Simulation of Adaptive Behavior (SAB'98)* (Cambridge, Massachusetts, 1998).

[107] PICAULT, S. *Modèles de comportements sociaux pour les colelctivités d'agents et de robots.* Thèse de doctorat, Université Paris VI, 2001.

[108] PICAULT, S., AND DROGOUL, A. The MICRobES Project, an Experimental Approach towards "Open Collective Robotics". In DARS [35].

[109] PICAULT, S., AND DROGOUL, A. Robots as a Social Species: the MICRobES Project. In Dautenhahn [38], pp. 139–141.

[110] PICAULT, S., AND LANDAU, S. Ethogenetics: an Evolutionary Approach to Agents Organization. In *Actes du Colloque ALCAA (Agents Logiciels, Coopération, Apprentissage et Activité Humaine* (2001), IUT de Bayonne, Université de Pau.

[111] PICAULT, S., AND LANDAU, S. Ethogenetics and the Evolutionary Design of Agent Behaviors. In Callaos et al. [28], pp. 528–533.

[112] RAM, A., ARKIN, R., BOONE, G., AND PEARCE, M. Using genetic algorithms to learn reactive control parameters for autonomous robotic navigation. *Adaptive Behavior 2*, 3 (1994), 277–304.

[113] Rechenberg, I. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann–Holzboog, Stuttgart, 1973.

[114] Schwefel, H.-P. *Evolutionsstrategie und numerische Optimierung.* Dr.-Ing. Thesis, Technical University of Berlin, Department of Process Engineering, 1975.

[115] Schwefel, H.-P. *Evolution and Optimum Seeking.* John Wiley and Sons, Inc., 1995.

[116] Sebald, A. V., and Fogel, L. J., Eds. *Proceedings of the Third Annual Conference on Evolutionary Programming* (1994), Evolutionary Programming Society, World Scientific.

[117] Sims, K. Evolving 3D Morphology and Behavior by Competition. *Artificial Life 1*, 1 (1995), 353–372.

[118] Spector, L., and Stoffel, K. Automatic generation of adaptive programs. In Maes et al. [87].

[119] Spector, L., and Stoffel, K. Ontogenetic programming. In Koza et al. [74], pp. 394–399.

[120] Stoffel, K., and Spector, L. High-performance, parallel, stack-based genetic programming. In Koza et al. [74], pp. 224–229.

[121] Tan, M. Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents. In *Proc. of the Tenth International Conference on Machine Learning* (Amherst, MA, 1993), pp. 330–337.

[122] Wall, M. Galib. `http://lancet.mit.edu/ga/`, 2000.

[123] Weiss, G., and Sen, S., Eds. *Adaptation and Learning in Multi–Agent Systems.* Lecture Notes in Artificial Intelligence. Springer Verlag, Berlin, 1996.

[124] Whitley, D. the genitor genetic algorithm. `http://www.cs.colostate.edu/~genitor/`, 1989.

[125] Whitley, D. A Genetic Algorithm Tutorial. *Statistics and Computing 4* (1994), 65–85.

[126] Whitley, D., Starkweather, T., and Bogart, C. Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity. In *Parallel Computing*, vol. 3. august 1990, pp. 347–361.

[127] Wilson, S. W. Classifier Systems and the Animat Problem. *Machine Learning 2*, 3 (1987), 199–228.

[128] Wilson, S. W. The Animat Path to AI. In Meyer and Wilson [96], pp. 15–21.

[129] Woods, W. A. Transition Networks Grammars for Natural Language Analysis. *Communications of the Association for the Computational Machinery 13*, 10 (1970), 591–606.

[130] Yao, X. Evolving Artificial Neural Networks. *PIEEE: Proceedings of the IEEE 87* (1999).