Thèse de doctorat de l'Université Paris VI
Spécialité : Informatique

présentée par
Samuel Landau

pour obtenir le titre de
Docteur de l'Université Paris VI

Des AG[1] vers la GA[2]

Sélection Darwinienne et Systèmes Multi-Agents

soutenue le 17 janvier 2003 devant le jury composé de :

M.	Jacques Pitrat,	Dir. de Recherche émérite (CNRS/LIP6, Paris)	(Président)
M.	Paul Bourgine,	Enseignant-chercheur (CREA/École Polytechnique)	(Rapporteur)
M.	Philippe Preux,	Professeur (Univ. du littoral, Calais)	(Rapporteur)
M.	Jean-Jules Brault,	Professeur agrégé (École Polytechnique, Montréal)	(Examinateur)
M.	Jean-Yves Jaffray,	Professeur (Univ. Paris VI)	(Examinateur)
M.	Sébastien Picault,	Maître de Conférences (Univ. de Lille I)	(Examinateur)
M.	Alexis Drogoul,	Professeur (Univ. Paris VI)	(Directeur)

1. algorithmes génétiques
2. génétique d'agents

Document formaté avec LaTeX, compilation du 13 février 2003

Remerciements

Je tiens tout d'abord à exprimer mes vifs remerciements aux chercheurs et professeurs qui ont accepté de participer à ce jury :

- à Paul Bourgine, qui a accepté le rôle de rapporteur sur cette thèse dans les circonstances que l'on sait ;
- à Philippe Preux, dont le rapport et les remarques pertinentes m'ont permis de clarifier mon propos ;
- à Jean-Yves Jaffray, qui avait déjà porté un regard extérieur sur mon travail qui n'en était qu'à sa deuxième année. Ce sont ses commentaires d'alors qui m'ont permis de mieux étayer mon travail par des comparaisons avec ces autres systèmes aux noms si cryptiques, qu'il a pointés avec humour lors de la soutenance ;
- à Jean-Jules Brault, qui m'a fait l'honneur de venir de l'Ouest pour ma soutenance, pour goûter la célèbre ontophylogenèse parisienne, version intelligence artificielle ;
- et à Jacques Pitrat, enfin, dont les remarques surprenantes et les questions décalées de prime abord, ont ouvert la discussion vers de nouvelles réflexions sur les modèles que j'ai présentés.

Cette thèse n'aurait pu se dérouler sans la confiance totale dont m'a crédité Alexis Drogoul, qui, après m'avoir initié aux problématiques de robotique distribuée lors d'un stage de maîtrise, puis, en DEA IARFA, à l'univers des systèmes multi-agents, ne s'est pas laissé désarçonner par un étudiant au sommeil alors intempestif. Sous sa direction, l'autonomie dont j'ai joui m'a permis d'explorer sans entraves, et m'a permis de trouver, sous les agents, le darwinisme.

Lors du DEA, Jean-Arcady Meyer et Agnès Guillot m'ont fait prendre conscience de mon statut précaire d'animat opportuniste sujet à sélection darwinienne. Nos échanges fréquents sont indissociables de mon cheminement, et même si je ne suis toujours pas parvenu à déterminer quelle architecture me contrôle, la flamme darwinienne ne m'a plus quitté.

Jean-Pierre Briot, Jean-François Perrot et Jean-Daniel Zucker ont toujours été présents, accordant aide et écoute sans compter.

Pendant toute la durée de ma thèse, Alain Cardon a été un interlocuteur privilégié et une référence pour ma recherche. Depuis Royaumont jusqu'à

Washington en passant par Orlando, j'ai toujours pu compter sur son soutien actif et inlassable. Ses réflexions et visions, parfois ésotériques pour qui ne connait pas par cœur les œuvres complètes de ses philosophes et mathématiciens de référence, sont cependant toujours une source inépuisable d'inspiration.

L'ambiance chaleureuse qui règne parmi les doctorants a rendu agréable les échanges et travaux au laboratoire, d'autant que j'ai cumulé en étant entre deux équipes ; merci en particulier à Angelica Muñoz, Diane Vanbergue, Nicolas Bredèche, David Filliat, Pierre Gérard, Benoît Girard, Stéphane Gourichon, Louis Hugues, Frédéric Kaplan, François Sempé, Thomas Meurisse, Frédéric Peschanski et David Servat.

Au-delà de nos travaux de thèse respectifs, je partage avec Stéphane Doncieux des rêves et des projets sferiques et dronatiques, sans que nous sachions vraiment s'ils seront réalisables un jour.

Collaborer et interagir avec Olivier Sigaud, à l'origine de la grande unification « ATNoSFERES-LCS », m'a ouvert de nouvelles pistes dont beaucoup sont encore à explorer. Son regard critique et ses travaux avancés sur les apports et le sens de la simulation d'agents adaptatifs ont nourri ma réflexion ; nous nous sommes en quelques sorte rejoints sur une hétérodoxie commune, Olivier étant aux animats ce que je suis aux systèmes mutli-agents.

Ma coopération étroite avec Sébastien Picault, dont l'ouverture et la culture n'ont d'égales que sa gourmandise intellectuelle, a été un milieu fertile qui a permis l'éclosion de la plupart des idées sous-jacentes à ma thèse. Nos discussions intarissables sont un plaisir sans cesse renouvelé.

Merci à mes parents et à mon frère de m'avoir toujours encouragé et soutenu, d'avoir cru en moi malgré mes difficultés à leur faire comprendre quoique ce soit de mes travaux.

Enfin merci à Yaël, patiente et compréhensive, qui m'a épaulé et aidé de tout son amour.

Résumé

La conception de comportements de robots mobiles autonomes est une tâche difficile. Les différentes approches existantes ne sont pas entièrement satisfaisantes, soit parce qu'elles ne permettent pas de faire face aux difficultés qui se présentent dans les environnements réels, soit parce que leur mise en œuvre se révèle trop complexe. Pour des raisons de distributions fonctionnelle et temporelle des comportements d'un robot, nous proposons d'employer un mécanisme de contrôle qui soit un système multi-agents, et de le concevoir à l'aide de techniques évolutionnistes. La conception de systèmes multi-agents par des méthodes inspirées de la sélection darwinienne constitue donc la problématique centrale de cette thèse. Dans ce but, nous proposons d'abord un cadre méthodologique pour l'évolution d'architectures d'agents (l'Éthogénétique) et un principe de construction de structures pour décrire leur comportement respectant ce cadre (*Stack-Based Gene Expression*). Puis, nous instancions un modèle (ATNoSFERES) qui applique ce principe d'évolution de structures à l'évolution de graphes. Ce modèle est d'abord testé pour un agent seul, et comparé à d'autres architectures d'agent, puis employé dans un modèle multi-agent de contrôle de robot (ATNoCells), conçu pour permettre l'enseignement par démonstration d'un comportement à un robot par un tuteur humain. Nous discutons enfin des propriétés de notre approche, aussi bien en tant qu'algorithme évolutionniste, que comme base de méthodes génériques de conception de SMA adaptatifs.

Mots-clés: conceptions de systèmes multi-agents, évolution artificielle de structures, sélection darwinienne, robotique évolutionniste

Abstract

The design of autonomous mobile robots behaviors is a difficult task. The different existing approaches are not fully satisfactory, either because they do not allow to cope with the difficulties in real-world environments, either because their implementation reveal to be too complex. The functional and temporal distribution of the behaviors of a robot lead us to use a control mechanism which is a multi-agent system, et to design it thanks to evolutionary techniques. The design of multi-agent systems by methods inspired by Darwinian selection is thus the main problematic of this thesis. With this goal in mind, we first propose a methodological frame for evolving agents architectures (Ethogenetics) and a principle for building structures to describe their behavior, that complies with this frame (Stack-Based Gene Expression). Then, we instantiate a model (ATNoSFERES) which applies this principle to the evolution of graphs. This model is first tested for a lonely agent, and compared to other agents architectures, then it is used in a multi-agent model for controlling a robot (ATNoCells), designed to allow the teaching by demonstration of a behavior to a robot by a human tutor. We finally discuss the properties of our approach as an evolutionary algorithm, as well as a basis for generic methods to design adaptive multi-agent systems.

Keywords: design of multi-agent systems, artificial evolution of structures, Darwinian selection, evolutionary robotics

Table des matières

Table des figures

Liste des tableaux

Introduction

Cette thèse s'inscrit dans le champ des systèmes multi-agents (SMA), de l'évolution artificielle, de la vie artificielle, et de la robotique évolutionniste.

Sujet

Du point de vue des recherches en systèmes multi-agents (Ferber, 1995; Drogoul, 1993), les architectures de contrôle de robots mobiles présentent un intérêt particulier. Une telle architecture qui prendrait la forme d'un système multi-agent est un paradigme de systèmes multi-agents dits *situés* (Drogoul, 2000), dont l'environnement, dynamique, n'est pas complètement spécifiable à l'avance, et dans lequel le système est obligé de composer avec sa *corporéité* (Dautenhahn, 1999; Cardon, 2001), qui s'inscrit dans celle du robot et non pas celle d'un environnement simulé, contrôlable dans ses moindre détails. La corporéité du robot présente cette caractéristique d'être *implicite* et *physique*, donnée a priori et non-modifiable, rendant le problème du contrôle d'autant plus ardu. Dans ce contexte, le système multi-agent et son concepteur sont confrontés non seulement à l'environnement du robot, mais aussi aux caractéristiques propres au robot (perceptions bruitées, actionneurs imparfaits). Ces caractères des robots mobiles ne sont pas contournables, à moins de réduire

le problème en simplifiant drastiquement l'environnement ou les conditions de réalisation des éventuelles tâches à effectuer.

Dans l'autre sens, du point de vue d'un concepteur de comportements pour robot, un système multi-agent utilisé comme architecture (Connell, 1989; Kaebling, 1987) apporte modularité, robustesse et la possibilité de distribuer causalement le comportement du robot, ce qui permet de le rendre plus intelligible, apport précieux car la conception de comportements de robots mobiles n'est pas une activité facile du fait des contraintes sus-nommées. Au-delà de la complexité des tâches, il faut aussi considérer leur nombre et leurs interactions, notamment leurs conflits : les systèmes multi-agents se présentent alors naturellement dans cette perspective. Les contraintes font qu'il est impossible de prévoir avec exactitude quel sera l'état du robot à un moment donné, et donc encore moins le comportement qu'il devra exprimer à cette occasion : c'est pourquoi le système multi-agent qui génère les comportements doit être adaptatif (Cardon, 1998; Cardon, 1999).

Pour obtenir automatiquement ces comportements adaptatifs nous avons choisi d'utiliser des méthodes s'inspirant du modèle de la sélection darwinienne (Darwin, 1859), 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 (Langton, 1988) ou en robotique (Nolfi and Floreano, 2000). Les méthodes de ce type les plus utilisées pour l'instant sont les algorithmes évolutionnistes (Bremermann, 1962; Holland, 1962; Fogel et al., 1966; Rechenberg, 1973).

Nous avons également choisi une description structurée des comportements des agents qui composent le système, qui a l'intérêt, du point de vue des concepteurs, d'être modulaire, compréhensible et facilement manipulable. Cela permet de faire des essais, d'altérer une solution existante ou de pouvoir

soumettre à la machine une solution partielle dans l'espoir qu'elle trouve une meilleure solution plus rapidement : c'est le principe de « l'amorçage >(*bootstrap* (Pitrat, 1990)). Notre choix s'est porté sur des graphes étiquetés, similaires à des ATN[1] (Woods, 1970), utilisés comme descriptions d'automates finis non-déterministes.

« 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 (Picault and Landau, 2001b)), puis nous avons défini une nouvelle méthode évolutionniste qui les satisfait (Landau and Picault, 2002b). 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 (Landau et al., 2001b; Landau and Picault, 2002a; Picault, 2001).

« Cette thèse envisage le contrôle d'un robot non pas au moyen d'un seul agent, mais au moyen d'un système multi-agents, ce qui permet de manipuler des structures simples tout en conservant la possibilité de voir un comportement complexe exprimé par un robot. Le comportement d'un système multi-agent peut en effet être beaucoup plus riche que celui des agents qui le composent (Drogoul and Dubreuil, 1992; Drogoul, 1993). 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.

1. Augmented Transition Network

« Pour expérimenter cette nouvelle approche de conception de comportements d'agents évolutifs et la comparer à d'autres, nous avons conçu le framework SFERES (Landau et al., 2001a; Landau et al., 2002a), qui permet d'appliquer des algorithmes évolutionnistes à des systèmes multi-agents. Puis nous avons implémenté une extension (ATNoSFERES (Landau et al., 2001b; Picault, 2001)) qui permet de faire évoluer nos graphes.

« Plan

Dans une première partie (chapitre 1) 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. Nous y décrivons aussi les principes d'évolution de comportements pour que ceux-ci exhibent les propriétés voulues. Puis suivent l'établissement des principes que nous avons émis afin de faire évoluer des comportements d'agents en respectant les propriétés voulues (chapitre 2), et la description des modèles et implémentations élaborés pour instancier et évaluer ces principes, SFERES et ATNoSFERES (chapitre 3). Dans une quatrième partie (chapitre 4) nous présentons des expérimentations menées pour vérifier que le modèle ATNoSFERES satisfait bien aux principes posés et le comparer à des approches voisines ayant vocation à s'attaquer à des problèmes de même type. On verra notamment que ATNoSFERES se classe parmi les meilleures méthodes évolutionnistes existantes. Après une cinquième et dernière partie (chapitre 5) où nous discutons sur les principes et modèles proposés, ainsi que sur ce que les expériences ont permis de valider et d'émettre comme nouvelles conjectures, nous concluons sur les perspectives ouvertes par ce travail de thèse (conclusion, page 167).

Chapitre 1

Contexte

La robotique mobile est amenée à jouer un rôle croissant dans notre quotidien. Celui-ci est déjà envahi par les objets « communicants » (téléphones portables, assistants personnels et ordinateurs de poche, appareils électroménager « intelligents », ...) et commence à l'être par des robots mobiles (aspirateurs autonomes, automobiles sans conducteur, robots de surveillance, robots de sauvetage...). Ceux-ci se verront confier des tâches de plus en plus complexes dans des environnement de moins en moins contrôlés et contrôlables. Il leur sera donc nécessaire d'avoir de bonnes capacités d'adaptation, en particulier pour pouvoir faire face à la complexité de notre quotidien (Agre, 1988). Là où ne se posaient que des problèmes informatiques et techniques liés à la distribution des objets, vont s'ajouter des contraintes et problèmes liés à la mobilité, au corps physique et à l'ancrage (Harnad, 1990) 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 (Drogoul, 2000; Picault and Drogoul, 2000a). Les objets « communiquants » sont déjà plus ou moins bien acceptés (songeons à ces sonneries de

téléphones portables si agaçantes dans les lieux publics) et partie intégrante de notre vie, qu'en sera-t-il pour des robots, qui auront plus de capacités d'action sur le monde ? La seconde est la conception du comportement des robots, 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 fortuite. Le projet MICRobES[1] (Drogoul and Picault, 1999; Picault and Drogoul, 2000b), 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 robots en environnement réel bien plus complexe que la conception de systèmes logiciels. L'impossibilité de modéliser à l'avance tous les évènements et d'avoir un contrôle sur eux (par exemple réagir à temps pour ne pas risquer de blesser une personne, sans savoir comment elle va se comporter ou quels vont être ses réflexes de protection) plaide pour un contrôle du robot qui soit décentralisé pour mieux séparer les fonctions, et asynchrone pour pouvoir réagir immédiatement aux évènements. Il est nécessaire que les robots disposent d'un minimum d'autonomie et que dans ce but ils fassent montre de suffisamment d'adaptativité. Nous nous proposons d'adopter l'approche par systèmes multi-agents adaptatifs, qui permet en plus de rendre compte causalement de la modularité et des interactions entre modules.

Nous présentons le contexte dans lequel se placent nos travaux en robotique autonome (section 1.1), en évolution artificielle (section 1.2) et plus

1. *cf.* `http://miriad.lip6.fr/microbes/`

particulièrement en évolution artificielle de structures (section 1.3). Cette synthèse va nous permettre d'expliquer notre choix des systèmes multi-agents comme architectures pour les robots, et celui des approches sélectionnistes pour générer les comportements adaptatifs de ces systèmes. Ce choix résulte de ce que la conception directe des comportements de robots par modules comportementaux (vision fonctionnelle, distribuée et asynchrone) s'avère trop complexe, et de ce que l'apprentissage par renforcement (Sutton and Barto, 1998) ne suffit pas pour réduire suffisamment la complexité de conception. Nous nous intéressons alors à l'évolution artificielle pour fabriquer automatiquement des architectures d'agents. La revue des méthodes évolutionnistes existantes nous permet de dégager des propriétés nécessaires à l'évolution d'architectures d'agents que sont principalement d'une part la *(quasi) continuité génotype-phénotype* et d'autre part le *pouvoir expressif*, et nous constatons qu'aucune des approches existantes ne présente simultanément ces deux propriétés.

Enfin, un récapitulatif résume ce chapitre (section 1.4), avant d'aborder au chapitre suivant nos directions de recherche et les moyens théoriques que nous nous donnons pour les atteindre. Ces principes et leur implémentation constituent ce travail de thèse, et la première étape de notre travail de recherche.

1.1 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 (notamment la planification (Nilsson, 1969; Fikes and Nilsson, 1971)) ne suffisent pas pour obtenir des comportements robustes dans les environne-

ments non contrôlés (Agre and Chapman, 1990). Des méthodes réactives plus efficaces ont vu le jour depuis une quinzaine d'années : notamment en effectuant des découpages comportementaux (Brooks, 1986), puis plus récemment des approches dites évolutionnistes (Nolfi and Floreano, 2000), plus souples et moins contraignantes. Ces dernières ont pour l'instant surtout été appliquées à des robots solitaires, et pour des architectures dont la structure est le plus souvent figée. Nous les évoquerons à la section 1.1.4.

1.1.1 Un système dynamique

Notre quotidien est un environnement dynamique, riche et bruité pour un robot (Brooks, 1991b). 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 (Arkin, 1998). 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 (Agre, 1988), 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 ni 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 forcément liée à celle du robot : un exemple minimaliste connu est celui des robots de Braitenberg (Braitenberg, 1986), qui, bien que simples, exhibent des comportement relativement complexes.

1.1.2 Les approches délibératives

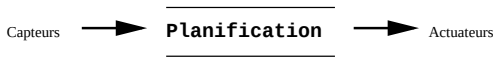


FIG. 1.1 – *Schéma général des approches délibératives. Le concepteur doit fournir au planificateur les symboles et les règles pour percevoir et agir.*

Les approches délibératives (Nilsson, 1969), pour sophistiquées qu'elles soient (Chatila and Laumont, 1985), s'avèrent dans ce cadre peu adaptées – le schéma de fonctionnement de tels systèmes est donné figure 1.1.

Ces systèmes raisonnent sur des symboles représentant des abstractions données par le concepteur. Concevoir de tels systèmes représente un travail d'ingénierie important et méticuleux, qui nécessite une modélisation extensive et fine du monde. La critique chronique adressée à ce type d'approche (Agre and Chapman, 1990; Brooks, 1991a; Agre, 1988; Connell, 1989) concerne l'absence d'ancrage des symboles (*symbol grounding* (Harnad, 1990)) manipulés par le système, ou plus exactement le fait que cet ancrage *doit être* réalisé a priori par le concepteur. Les symboles peuvent alors n'avoir aucun lien avec la réalité, ou leur choix s'avérer mal adapté. On dira que ces symboles n'ont pas *d'ancrage sensori-moteur*, car ils ne sont pas le fruit d'une construction, de l'interaction entre le robot et son environnement.

Pour pallier au problème du temps de traitement prohibitif des informations en provenance des capteurs du robot, des travaux préconisent de prétraiter hors-ligne des modèles en procédures perceptuelles spécifiques (Ghallab, 1999). Cependant cette méthode est inutilisable dans un environnement réel qui est par nature instable et changeant, notamment car la plupart des compilations de connaissances et de modèles du monde dans de tels systèmes reposent sur les perceptions visuelles. Or, l'environnement lumineux a une

influence considérable sur la perception des couleurs par une caméra vidéo. Il faut donc pour utiliser de tels systèmes imposer un environnement complètement contrôlé (sans lumière naturelle surtout, donc un environnement complètement clos), et même sans changement apparent de l'environnement, la perception des couleurs par une caméra n'est généralement pas constante (le problème de la constance de la couleur est un problème très complexe et fait l'objet d'importants travaux (Brainard and Freeman, 1997)).

Par ailleurs, les capacités sophistiquées de perception mises en jeu, notamment en vision, sont coûteuses et complexes à mettre en œuvre. Sans compter que tout jeu de capteurs robotiques est sujet à un fort bruit et nécessite un recalage régulier. Tout ceci interdit par avance l'utilisation de tels systèmes dans des environnements généraux et non-structurés.

Des modèles de planification hybrides, plus souples, incluant une partie comportementale ou réactive sont aussi explorés, par exemple des modèles de planification réactive ou des architectures liant des modules de planification rationnelle et de comportements réactifs (comme InteRRaP (Müller and Pischel, 1993)). Il n'est alors plus nécessaire d'avoir une connaissance a priori de toutes les séquences possibles des situations et des actions à exécuter. Le système dispose d'un ensemble – non structuré, donc – prédéterminé de règles conditions-actions (comme les Règles de Contrôle Situées (Drummond, 1989)) qui font qu'en théorie, il est possible de faire face à tous les évènements exogènes ainsi qu'à des effets incertains ou des conditions initiales inconnues. En effet, en théorie toujours, le concepteur peut fournir au système une règle de réaction pour toute situation rencontrée, que les circonstances qui y mènent soient ou non envisagées. Même dans ce cas plus souple des systèmes de planification hybrides, il n'en demeure donc pas moins

que la conception reste délicate et très laborieuse, puisque l'environnement n'est pas contrôlé.

La nécessité de faire évoluer le robot dans des environnements structurés et connus, notamment pour avoir des temps de réponse acceptables (Arkin, 1998) et le travail nécessaire à une description exhaustive des situations qui peuvent être rencontrées rendent donc ces approches peu adaptées et robustes dans un environnement réel.

1.1.3 Les approches réactives et comportementales

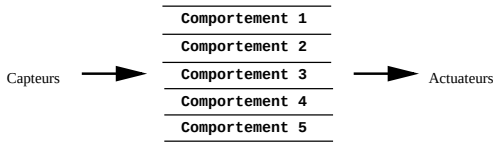


FIG. 1.2 – *Schéma général des approches comportementales (d'après (Nolfi and Floreano, 2000)). Les modules comportementaux sont séparés et définis par le concepteur.*

D'inspiration biologique (éthologie et psychologie animale), les approches réactives et comportementales (*behavior-based robotics* (Brooks, 1999; Arkin, 1998; Brooks, 1990; Maes, 1990)) se sont développées surtout suite aux travaux de R. Brooks (Brooks, 1986). Les principes, exposés dans (Brooks, 1991a), mettent l'accent sur la *situation* du robot dans le monde (on ne raisonne plus sur des symboles et des 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. Brooks résume cette vision minimaliste (qui préconise l'absence de modèle

du monde) dans cette devise célèbre : *"The world is its best own model"*[2] (Brooks, 1991a).

Ces approches sont ascendantes et modulaires : on définit des modules comportementaux simples qui peuvent agir de façon coopérative (Arkin, 1987) ou compétitive (Brooks, 1986). 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 se fait principalement 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.

Ces méthodes de conception fonctionnent mieux pour des robots en environnement réel que la planification sur des tâches telles que la manipulation et la collecte d'objets (Connell, 1989), la navigation en évitant les obstacles (Brooks, 1986), l'exploration, la localisation, la cartographie (Filliat, 2001) ou la navigation en formation d'un groupe de robots, et des combinaisons simultanées de ces comportements. Cependant, elle repose sur l'intuition du concepteur qui doit séparer et spécifier les modules, et requiert de sa part une attention soutenue lors des expérimentations. De l'aveu même de ceux qui ont adopté ce schéma de décomposition comportementale, cette séparation est un problème crucial (Brooks, 1991a). La figure 1.3 représente l'architecture de subsomption pour un robot muni de 3 « compétences » de base : éviter tout contact avec des objets (mobiles ou immobiles), évoluer sans but (*wander*) en évitant des obstacles et « explorer » le monde en repérant des endroits accessibles et en s'y rendant. Le robot exécute deux tâches concurrentes : la première par défaut est d'évoluer au hasard, la seconde est de se rendre dans un endroit désigné lorsque celui-ci apparaît accessible. La figure laisse pré-

2. « le monde est son propre meilleur modèle »

sager de la difficulté à concevoir des comportements plus riches, comportant plus de tâches.

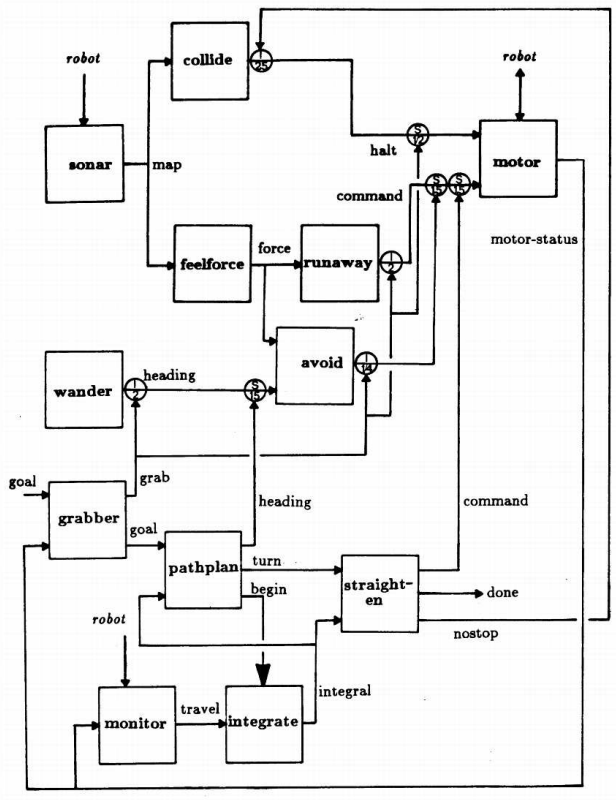


FIG. 1.3 – *Exemple d'architecture de subsomption (extrait de (Brooks, 1986)). Le robot évolue au hasard tant qu'il ne détermine pas d'endroit en direction duquel aller, le tout en évitant les obstacles.*

La séparation et la spécification des modules sont rendues difficiles par l'aspect émergent du comportement. Les interactions mêmes entre modules rendent difficile une approche incrémentale de la conception puisqu'elles ajoutent de l'incertitude sur le comportement actuel du robot dans son environnement. Pour pallier ces difficultés d'ajustement, des techniques d'apprentissage par renforcement ont été utilisées avec succès (Kaebling, 1990; Maes and Brooks, 1990; Brooks, 1991b; Mahadevan and Connell, 1992; Ram

et al., 1994). 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). De manière générale, l'apprentissage par renforcement n'est pas aussi flexible que peuvent l'être les méthodes évolutionnistes pour concevoir des comportements de robots (*cf.* (Nolfi and Floreano, 2000) 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 abordons l'évolution artificielle à la section 1.2, et l'évolution de structures à la section 1.3.

1.1.4 Robotique évolutionniste

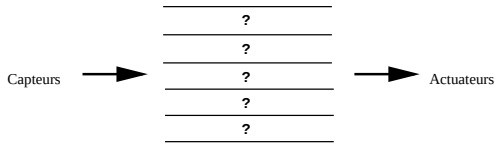


FIG. 1.4 – *Schéma général des approches en robotique évolutionniste (d'après (Nolfi and Floreano, 2000)). La séparation et la conception des modules comportementaux se fait automatiquement, sans intervention du concepteur.*

La robotique évolutionniste (Meyer and Wilson, 1991; Nolfi et al., 1994; Harvey et al., 1997; Nolfi and Floreano, 2000) s'inspire elle aussi de modèles biologiques pour définir des architectures de robots. Le paradigme est celui de la sélection darwinienne (Darwin, 1859), que nous allons présenter plus en détail ci-après (section 1.2). Cette méthode non supervisée est moins contraignante que les apprentissages par renforcement classiques car elle permet notamment de générer entièrement l'architecture du robot, et pas seulement de la paramétrer (*cf.* section 1.3).

Il ne faudrait pas croire que les comportements ainsi obtenus se cantonnent à des arcs réflexes, car l'on peut tout à fait faire évoluer des architectures ayant des états internes. Certaines expériences spectaculaires et minimalistes (Floreano and Urzelai, 2000a) parviennent par exemple à faire apprendre à un petit robot, de façon non-supervisée et en ligne, certaines séquences de tâches sans avoir à lui montrer quoi que ce soit. Dans ce cas, on a fait évoluer les paramètres d'apprentissage d'une architecture (apprentissage hebbien d'un réseau de neurones artificiel en l'occurrence) afin que le robot puisse apprendre *en ligne* et *sans supervision ni démonstration.*

1.2 Évolution artificielle

Une courte introduction au paradigme de la sélection darwinienne (section 1.2.1), aux algorithmes évolutionnistes (section 1.2.2) puis aux algorithmes classiques (section 1.2.3) nous permet d'expliquer en quoi l'évolution artificielle pour la conception de comportements d'agents nous paraît la méthode la plus indiquée (section 1.2.4), alors que nous insistons sur les limites de leur principale instanciation, les algorithmes évolutionnistes (section 1.2.5). Nous considérons en effet qu'un « retour aux sources » de la sélection darwinienne est nécessaire (section 1.2.6), les algorithmes évolutionnistes classiques n'en ayant, à notre sens, que capté la forme sans en retenir le fond. Ceci peut précisément être fait dans le cadre des systèmes multi-agents et cette proposition sous-tend ce travail de thèse.

Une fois ces points établis, le premier problème concret qui se pose est celui de décider comment faire évoluer les architectures d'agents. Nous dégageons les propriétés que nous pensons nécessaires pour faire évoluer des architectures d'agent à la section suivante (section 1.3, page 29). Ces propriétés

seront érigées en principes de l'Éthogénétique au chapitre suivant (section 2.1 page 42).

1.2.1 Le paradigme de la sélection darwinienne

La théorie darwinienne de l'évolution des espèces s'appuie sur le fait qu'un environnement naturel donné fournit une quantité de ressources limitée, donc ne peut assurer la survie d'un nombre infini d'individus. Par suite, ne peuvent survivre et se reproduire que ceux qui sont les plus adaptés à leur environnement (à un moment donné) : c'est le principe de la sélection. La seconde observation est la ressemblance entre enfants et parents (l'hérédité) : des traits communs sont transmis d'une génération à l'autre, parmi lesquels ceux qui ont permis aux parents d'être adaptés à leur environnement. Mais les enfants ne sont pas des copies conformes des parents : il y a variation. Du fait de ces variations, certains individus seront plus adaptés que d'autres et par suite favorisés par la sélection. L'hypothèse sous-jacente à la théorie darwinienne est l'indépendance entre les causes des variations et celles de la sélection : les variations sont en ce sens « aléatoires », sans rapport avec les contraintes qui vont déterminer le degré d'adaptation d'un individu. Darwin avait en effet noté que les éleveurs, qui pratiquent une sélection artificielle en croisant les animaux pour obtenir certains caractères, ne parviennent pas à susciter des variations correspondant aux critères désirés : ils ne peuvent que choisir des traits intéressants portés par certains individus parmi des variations aléatoires. L'extrapolation à une sélection naturelle dans laquelle les contraintes de l'environnement favorisent la propagation de certains caractères aux dépens d'autres constitue l'idée centrale de la théorie darwinienne (Darwin, 1859). Le processus ainsi produit peut être vu alors comme un algorithme très général capable de produire de l'ordre, de la forme, des comportements

de façon graduelle à partir de désordre et d'aléatoire (Dennett, 1995). Ce mécanisme de hasard-sélection, sans finalité, opérerait simultanément à tous les niveaux, de celui des constituants les plus intimes des cellules (Kupiec and Sonigo, 2000) à celui des organismes (Darwin, 1859) et des groupes, voire des idées (*mèmes* (Dawkins, 1976)), *y compris* à des niveaux *meta* (Dennett, 1995).

Ce mécanisme simple a été repris en vie artificielle, et est source d'inspiration pour des méthodes d'optimisation et d'apprentissage automatique, dont, surtout, les algorithmes évolutionnistes.

1.2.2 Principe des algorithmes évolutionnistes

L'appellation générique *algorithme évolutionniste* désigne des systèmes informatiques de résolution de problèmes (Bremermann, 1962; Holland, 1962) qui s'inspirent des mécanismes adaptatifs de la sélection darwinienne. 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, dans les années 1980, la programmation génétique. Nous les présentons sommairement à la section suivante (section 1.2.3).

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'indivi-

```
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. 1.5 – *Principe de fonctionnement d'un algorithme évolutionniste*

dus est tirée au sort. Chaque individu est testé et une note – valeur sélective ou *fitness*[3] – 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 1.5 illustre le principe de fonctionnement d'un algorithme évolutionniste.

Bien que simplistes du point de vue de la métaphore biologique (nous en discutons section 1.2.5), ces algorithmes sont cependant suffisamment performants pour fournir des mécanismes de recherche adaptative robustes et puissants, en particulier pour des problèmes d'optimisation numérique (Goldberg, 1989; Bäck and Schwefel, 1993; Schoenauer, 1997).

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

Pour des présentations plus détaillées des algorithmes évolutionnistes et des comparaisons entre les différents types, on pourra consulter (Heitkötter and Beasley, 1998; Beasley et al., 1993; Mitchell and Taylor, 1999; Whitley, 1994).

1.2.3 Les principaux algorithmes

De nombreuses variantes d'algorithmes évolutionnistes existent. Elles diffèrent 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* (Holland, 1975a; De Jong, 1975; Goldberg, 1989; Holland, 1996) utilisent un codage de type chaîne binaire, alors que les *stratégies évolutionnistes* (Rechenberg, 1973; Schwefel, 1975; Schwefel, 1995) utilisent un vecteur de flottants, et une matrice triangulaire, elle aussi sujette aux mutations, qui code les corrélations entre les paramètres du problème contenus dans le vecteur. Ces corrélations sont ensuite utilisées lors des mutations des paramètres. Les stratégies évolutionnistes ont donc une approche *meta* de la mutation, et on parle *d'opérateurs génétiques adaptatifs*, à tort dans ce cas précis puisque les opérateurs génétiques sont figés et que seuls leurs paramètres changent.

La *programmation évolutionniste* (Fogel et al., 1966; Fogel, 1992) permet de faire évoluer directement des structures quelconques (machines à états finis, réseaux de neurones, etc.) et la *programmation génétique* (Koza, 1992; Montana, 1995) construit directement des expressions ou fonctions sous la forme de leur arbre de syntaxe abstraite. La programmation génétique est issue des algorithmes génétiques, mais s'en est rapidement distinguée pour devenir une méthode reconnue au même titre que les autres.

	algorithme génétique	stratégie évolutionniste	programmation évolutionniste	programmation génétique
individu	chaîne de bits	vecteur et matrice	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. 1.6 – *Caractéristiques des principaux algorithmes évolutionnistes*

La figure 1.6 récapitule les caractéristiques de chacun des principaux algorithmes évolutionnistes dans leur version « orthodoxe ». Beaucoup d'échanges ont eu lieu entre ces techniques, par exemple les opérateurs génétiques adaptatifs des stratégies évolutionnistes et de la programmation évolutionniste se sont d'abord assez naturellement hybridés, « naturellement » car ils reposent sur la même idée de faire porter par l'individu un ensemble de paramètres qui vont influencer les mutations respectives de ses paramètres ou de sa structure. Puis ils ont essaimé vers les algorithmes génétiques.

1.2.4 Qualités des algorithmes évolutionnistes

Les algorithmes évolutionnistes peuvent fournir des solutions à des problèmes n'ayant pas de solution analytique ou algorithmique, et ce avec moins de contraintes que les autres méthodes d'apprentissage et d'optimisation. Surtout, il permettent de *construire* des comportements et non pas juste de les paramétrer, et sont des méthodes d'apprentissage non-supervisées (ce dont nous avons besoin).

Du point de vue de l'optimisation, les algorithmes évolutionnistes sont une méthode globale et stochastique d'ordre zéro-ième (*i.e.* ne requérant que les valeurs de la fonction à optimiser) qui peuvent trouver l'optimum global de

fonctions très chahutées (Schoenauer, 1997). Ce nombre réduit d'hypothèses sur ce qu'on cherche implique que les algorithmes évolutionnistes peuvent souvent être appliqués là où d'autres méthodes classiques d'apprentissage sont inutilisables ou trop peu efficaces. L'aspect parallèle (à chaque étape plusieurs solutions sont prises en compte simultanément) et stochastique de la recherche permet d'éviter les minima locaux du paysage de l'espace de recherche, par la combinaison de solutions éloignées ou par une variation suffisamment grande. D'autre part, l'algorithme peut être interrompu à tout moment pour donner une solution utilisable même si elle est sous-optimale. Les *algorithmes de gradient*, par contre, ne peuvent s'appliquer qu'à des fonctions dérivables et dans un espace de recherche continu, pour peu que la fonction soit suffisamment régulière (c'est rarement le cas des comportements d'agents et de robots) ou que l'on ait suffisamment de connaissances a priori pour bien placer le point de départ de la recherche (et c'est à nouveau rarement le cas) (Mitchell, 1997). De plus, en tant que méthode de recherche locale, elle peut s'enfermer dans un optimum local. De même, les méthodes du type *Hill-Climbing*, bien qu'applicables à toute fonction à tout espace, ne garantissant qu'une convergence monotone vers un maximum local (Mitchell, 1997). Du fait de la localité de la méthode, le choix du point de départ de la recherche est déterminant.

Du point de vue comportemental, les apprentissages par renforcement (Sutton and Barto, 1998), comme en particulier le *TD-learning* (Sutton, 1988), sont très utilisés et efficaces, mais nécessitent un temps et un espace d'états et d'actions discrets. De plus, ces approches perdent leur efficacité si l'espace d'états grandit trop (*curse of dimensionality* (Sutton and Barto, 1998)), ce qui risque d'être le cas pour des robots mobiles.

Pour accélérer la recherche, il est possible d'informer les algorithmes évolutionnistes en les hybridant avec des algorithmes de recherche locale, notamment en biaisant les opérateurs de variation, selon ce qu'on sait du problème. Cependant la généricité de ces opérateurs est alors perdue.

Pour le cas multi-agent, l'utilisation de l'évolution artificielle comme méthode d'apprentissage permet de plus d'apporter une réponse au problème du *credit assignment* (Minsky, 1963). Ce problème est celui de décider, lorsque plusieurs parties d'un système sont actives au même moment, lesquelles de ces parties, actives à un pas de temps donné, ont contribué à accomplir un but désiré au(x) pas de temps suivant(s). Ce problème se pose donc lors de l'apprentissage pour un système multi-agent. Quels agents faut-il récompenser ou punir, et dans quelles proportions, au vu de ce qu'a produit le système ? Une réponse possible avec les algorithmes évolutionnistes est de choisir un type d'individu de l'algorithme qui représente les caractéristiques du *groupe* d'agents que l'on veut faire évoluer (Haynes et al., 1995), et non pas d'un agent pris individuellement. Un individu de l'algorithme code 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.

1.2.5 Limites des algorithmes évolutionnistes

Un outil complexe et coûteux

Il est significatif, malgré leur grande souplesse, que la plupart des travaux sur et avec les algorithmes évolutionnistes consistent à trouver des moyens pour accélérer la recherche (très coûteuse en temps et en espace), en particulier en trouvant des méthodes ou méthodologies pour biaiser les opérateurs

génétiques, et, partant, la politique de reproduction. Ils aboutissent à des meta-heuristiques, des politiques qui deviennent très complexes, dont ils sont incapables de bien mesurer les conséquences. Finalement, ils en sont réduits à procéder par tâtonnements, et accumuler des connaissances réutilisables dans de nouveaux contextes est très difficile. Toute personne familière des algorithmes évolutionniste en a fait la douloureuse expérience : « *ça* » *ne marche vraiment pas tout seul.* À témoin, les nombreuses « recettes de cuisine » que s'échangent les chercheurs du domaine pour tel ou tel problème, prophétisant que tel jeu de paramètres combiné avec telle politique reproductive constitue la meilleure solution. Assez paradoxalement, alors que « l'algorithme » de la sélection naturelle (section 1.2.1) est si simple au départ, ce qui en a été retenu dans les algorithmes évolutionnistes en informatique et leur utilisation pratique tend au contraire vers des abîmes de complexité. Peut-être est-ce ironiquement pour mieux illustrer un exemple de sélection naturelle au niveau meta : la véritable sélection naturelle à l'œuvre aurait lieu dans l'esprit du concepteur, lors des multiples essais et choix de conception pour son algorithme évolutionniste...

Le Paradigme perdu

Peut-être qu'une des causes réside dans ce que les algorithmes évolutionnistes se sont orientés vers une réduction du paradigme de la sélection naturelle. Alors que la sélection naturelle émerge de la distribution des individus qui vivent, se reproduisent et meurent, les algorithmes évolutionnistes sont au contraire un modèle centralisé, concentré principalement sur une modélisation phylogénétique d'individus le plus souvent isolés, sans environnement commun. Les travaux incluant aussi une modélisation du développement de ces individus lors de leur existence (*ontogenèse*) par exemple dans (Kod-

jabachian, 1998) ou l'apprentissage pendant l'existence des individus (*épigenèse*), par exemple dans (Floreano and Urzelai, 2000b), sont encore très minoritaires. Le modèle des algorithmes évolutionnistes semble éloigné du modèle biologique de la sélection naturelle au moins en ce que pour ce dernier, tout a lieu en même temps. Les individus se développent, apprennent éventuellement, et l'ensemble évolue. Des théories avancées en biologie moléculaire modélisent même ces trois processus (ontogénique, épigénique et phylogénique) comme plusieurs aspects, à des échelles de temps, d'espace et de formes différentes, d'un même processus primordial, basé sur des variations aléatoires/sélections, que Jean-Jacques KUPIEC appelle « ontophylogenèse » (Kupiec and Sonigo, 2000).

Cependant le plus frappant est que le *moteur* de la sélection naturelle est absent des algorithmes évolutionnistes : le fait que les individus soient *tous présents*, en interaction, dans *un même environnement*, qui conditionne leur survie et est en retour modifié par leur présence, induisant de fait une *compétition sur les ressources*, qui ne sont pas illimitées. En effet, dans les algorithmes évolutionnistes, les individus n'évoluent pas dans un même environnement de façon simultanée. Ils sont en général évalués séparément et n'ont aucune interaction, sauf éventuellement via la partie de l'algorithme dédiée à la reproduction, moment où ont lieu des mélanges de matériel génétique. L'évaluation séparée nous fait perdre une grande partie de la dynamique de la sélection darwinienne. Dans les algorithmes évolutionnistes, la sélection est un mécanisme *statique* opéré sur une population d'individus évalués séparément ; il n'y a pas de notion de temps, uniquement de succession des générations.

D'autre part, la partie reproductive de l'algorithme préfigure une version simplifiée, codifiée de la « compétition », adaptative ou sexuelle selon les cas.

La politique suivie est choisie a priori par le concepteur, et s'il arrive que des paramètres de son déroulement puissent être sujets à variation (*cf.* par exemple le cas des opérateurs génétiques adaptatifs, section 1.2.3), nous ne connaissons qu'un travail très récent où celle-ci ait pu être modifiée le long de l'exécution de l'algorithme (Spector and Robinson, 2002a). Or, le comportement reproductif fait partie de la panoplie des comportements des individus. Il n'y a donc pas de raison pour qu'il ne soit pas aussi systématiquement sujet à variation et sélection, d'autant plus qu'il joue justement un rôle central dans l'évolution. Les individus réputés les plus « aptes » ne sont-ils pas, a posteriori, justement ceux dont la lignée a fait montre de la meilleure capacité à se reproduire et à s'approprier les ressources indispensables à leur survie ?

Le problème de la fitness

Dans les algorithmes évolutionnistes, cette « aptitude » est mesurée par la fitness (*cf.* section 1.2.2). Déterminer la méthode de calcul d'une fitness est donc un problème-clé lors de l'utilisation de ces algorithmes. Il s'agit d'enfermer dans sa formule et dans son protocole d'évaluation la connaissance qualitative que l'on a de la solution recherchée, ainsi bien souvent qu'un biais pour l'atteindre plus sûrement et plus rapidement, dans un souci de « guidage » de l'évolution. La politique de reproduction imposant le plus souvent aux fitness d'être toutes comparables entre elles, nous pouvons souvent les réduire *in fine* à un nombre flottant. L'évaluation du comportement d'un agent, déployé dans le temps et les conditions environnementales imparties, est alors résumée à un flottant, une sorte de mesure de sa valeur, immanente et atemporelle, faisant abstraction de l'environnement.

Hormis les considérations sur l'absence de dynamique de l'ensemble des individus dans un même environnement exprimées précédemment, il se pose alors aussi un problème similaire à celui de l'apprentissage par renforcement : l'affectation de crédit temporel (*temporal credit assignment* (Sutton, 1984)). Il s'agit de décider comment récompenser ou punir à tel moment les prises de décision, l'objectif étant par itération de déterminer les meilleures, alors que l'on ne connaît pas toutes les conséquences futures des actions au moment où celle-ci sont exécutées. L'utilisation d'une fitness dans les algorithmes évolutionnistes correspond dans ce cadre à une distribution de récompense retardée : plus grande est la fitness, plus l'individu aura de chance de voir son héritage représenté à la génération suivante, sélectionnant ainsi ses caractères comportementaux. C'est en ce sens que déterminer une fitness est un exercice subtil, car elle doit permettre de mettre en avant les comportements exprimant des actions les plus proches possibles d'un optimum dans certaines conditions, alors que l'on n'a qu'une connaissance qualitative et globale du comportement.

Par ailleurs, les conditions d'évaluations sont de plus précisément définies et rarement exhaustives dès qu'il s'agit de mesurer un comportement. Précisément définies, car tous les individus doivent demeurer comparables afin de pouvoir appliquer la politique de reproduction. Rarement exhaustives, car comme le comportement d'un agent est le fruit d'une interaction permanente avec un environnement qui change, il n'est pas envisageable d'expérimenter toutes les conditions. Il faudra donc estimer à l'avance, alors que l'on n'a qu'une connaissance qualitative de la solution, la quantité de temps allouée à l'agent pour faire ses preuves et se débattre, et un échantillon de conditions expérimentales censées recouvrir au mieux les conditions les plus courantes auxquelles le système final devra faire face, suffisamment pour débruiter les

comportements qui ne prendront pas nécessairement des décisions déterministes ou pour pouvoir mieux généraliser des situations requérant la même action, mais pas trop pour ne pas allonger encore plus un algorithme évolutionniste par nature lourd en calculs.

1.2.6 Retour aux sources

Les algorithmes évolutionnistes ne retiennent qu'une vue partielle et déformée de la sélection darwinienne. Ce sont essentiellement des modèles statiques et centralisés de la phylogenèse d'un groupe d'individus « autistes », chacun dans son univers, alors qu'au contraire la sélection darwinienne est un processus dynamique, distribué, mettant en jeu des individus en interaction et en conflit pour des ressources limitées dans un environnement commun. En conséquence, les travaux mettant en jeu les algorithmes évolutionnistes sont assez concentrés sur la façon de récupérer ce qui a été perdu lors de la réduction du paradigme ; en particulier, réintroduire la dynamique d'un comportement d'agent s'exprimant tout le long de son existence, dans le calcul de la fitness, dont la formule à déterminer pour un problème donné constitue déjà en soi un travail de recherche à mener en parallèle avec l'étude du problème.

Nous pensons que ce surcroît de travail et de complexité peut être évité en « revenant aux sources » de la sélection darwinienne, en utilisant des modèles qui soient moins réducteurs que les algorithmes évolutionnistes. De plus, de façon analogue à celle des algorithmes d'apprentissage par renforcement qui exploitent toute l'information disponible pendant le déploiement du comportement des agents, nous espérons que revenir à un modèle sélectionniste dynamique permettra d'améliorer grandement les temps de convergence vers les solutions recherchées. En effet, pour l'instant, les algorithmes

évolutionnistes ne tiennent généralement pas la comparaison devant les algorithmes d'apprentissage par renforcement pour une tâche donnée, puisqu'ils exploitent beaucoup moins bien l'information disponible le long de l'existence de l'agent (Sutton and Barto, 1998).

Pour ce qui est des gains en vitesse de convergence, des modèles de systèmes multi-agent employant de la *génétique d'agents* (Cardon and Vacher, 2000), plus proches du paradigme de la sélection darwinienne que les algorithmes évolutionnistes, ont déjà été utilisés avec succès pour résoudre des problèmes multi-objectifs complexes et dynamiques (Cardon et al., 1999), et ce de façon beaucoup plus rapide que par des algorithmes évolutionnistes classiques.

Ce qui inspire notre travail de recherche est justement qu'il est possible d'envisager ce « retour aux sources » grâce aux modèles multi-agents. À l'instar des prémisses de la sélection naturelle, un système multi-agent en développement (*i.e.* dont les agents naissent, vivent, se reproduisent, et meurent) est un modèle dynamique, distribué, et les agents sont en interaction dans un environnement commun. Avec deux hypothèses supplémentaires nous pouvons envisager un modèle informatique du paradigme de la sélection darwinienne moins réducteur que les algorithmes évolutionnistes.

La première hypothèse est qu'il y ait des ressources qui conditionnent la survie des agents. Or, la façon dont ces ressources conditionnent la survie des agents est complètement dépendante du phénomène modélisé. L'étudier de façon générique apparaît être une tâche très complexe, trop peut-être : il faut déjà être capable de trouver des critères pertinents pour les caractériser.

La seconde hypothèse est de disposer d'un support structuré de description du comportement des agents (pour pouvoir l'appréhender, l'analyser,

le modifier) et d'un support héréditaire, à partir duquel la description du comportement serait obtenue. Il ne nous paraît a priori pas indispensable d'envisager le cas où le support héréditaire ne décrirait pas toute la structure (en particulier si nous envisageons un modèle incluant l'ontogenèse, la phase d'apprentissage ayant la charge de compléter la construction de la structure incomplète). Dans une telle situation, la description du modèle doit être refaite avec un grain plus fin pour aussi rendre compte de la dynamique de hasard-sélection à l'œuvre à l'intérieur même des agents. On procède ainsi en biologie pour expliquer, par exemple, l'apprentissage via le développement des neurones (où un mécanisme de hasard-sélection serait à l'œuvre (Edelman, 1992)) alors que le génome de l'individu ne peut porter en lui toutes les informations nécessaires à son élaboration.

Nous avons la conviction qu'il est presque toujours possible de modéliser ce que nous voulons avec ces deux seules hypothèses, pour peu que l'on puisse ajuster correctement l'échelle d'observation du phénomène à modéliser. Par ailleurs, la seconde hypothèse, au contraire de la première, peut, elle, être traitée de façon générique, puisqu'elle est indépendante des agents. Dans la section suivante nous faisons donc une revue critique des principales méthodes pour faire évoluer des structures, en ne perdant pas de vue notre objectif qui est de les utiliser pour décrire des comportements d'agent et de les obtenir à partir d'un substrat héréditaire.

1.3 Évolution de structures

La construction automatique de structures (Angeline, 1995) constitue l'un des courants les plus importants de l'informatique évolutionniste. Celles-ci peuvent par exemple être un circuit (Koza et al., 1996a), une machine à

états finis (Fogel et al., 1966), un arbre représentant un programme (Koza, 1992), ou un réseau de neurones (Yao, 1999). La plupart du temps, ce sont des structures *exécutables* (Angeline, 1998) dont on évalue le comportement, via celui de l'agent ou de l'animat (Wilson, 1987; Wilson, 1991; Meyer and Wilson, 1991) qui s'en sert comme architecture.

Il est possible de distinguer deux façons de faire évoluer des structures, selon le codage utilisé. La représentation génétique, celle sur laquelle seront appliquées les manipulations, peut être utilisée soit directement (codage direct, section 1.3.1) soit indirectement (codage indirect, section 1.3.2) – nous donnerons des exemples. Ces deux façons de faire ont des propriétés intéressantes qui ne sont pas incompatibles entre elles (pouvoir expressif, quasi continuité génotype-phénotype), mais aucune méthode existante ne présente ces propriétés simultanément. Nous allons présenter les différentes approches d'évolution de structures 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 2.2).

1.3.1 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 manipulations génétiques et la structure qui est utilisée effectivement. Les structures sont alors complètement générées, mais le concepteur doit faire face à beaucoup de contraintes (syntaxiques et surtout sémantiques) lors de la conception des opérateurs génétiques.

Codage paramétrique pour une structure existante

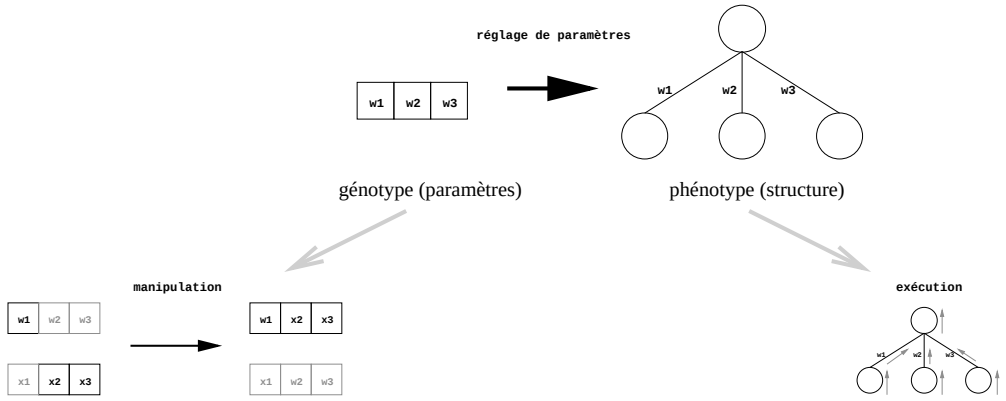


FIG. 1.7 – *Codage direct de paramètres pour une structure prédéterminée, ici les pondérations d'un perceptron (Rosenblatt, 1962).*

Le réglage de paramètres pour structures existantes consiste en une optimisation d'un ensemble de paramètres (Bäck and Schwefel, 1993), dont la position dans la représentation génétique et la signification pour la structure évoluée sont bien précises. Les algorithmes évolutionnistes les plus utilisés à cet effet sont les algorithmes génétiques et les stratégies évolutionnistes (*cf.* section 1.2.3). Le réglage de paramètres est beaucoup utilisé pour faire évoluer des réseaux de neurones (Miller et al., 1989; Whitley et al., 1990; Greenwood, 1997) (pour plus de détails consulter (Yao, 1999)). Cette méthode donne de bons résultats comparés à ceux d'autres méthodes d'apprentissage. Cependant, la nécessité de fournir a priori la structure est une contrainte trop forte, alors que d'autres méthodes permettent de ne pas avoir à le faire. Pour la suite nous nous référerons à ce défaut en disant qu'un tel codage manque de *pouvoir d'expression*.

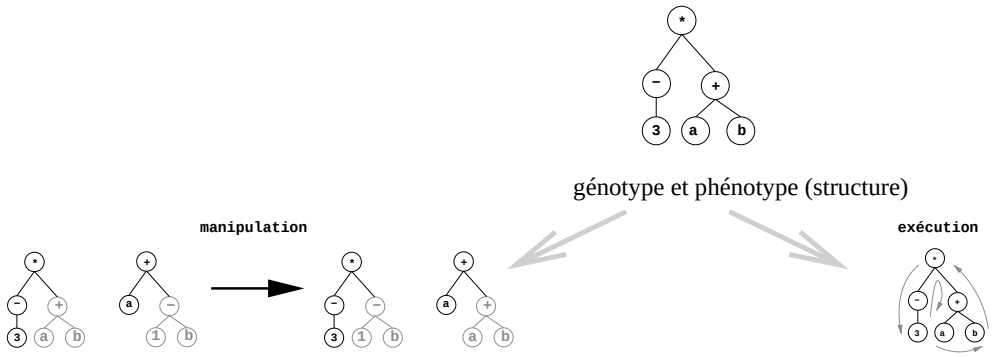


FIG. 1.8 – *Codage direct d'une structure, ici un arbre de dérivation pour la programmation génétique (Koza, 1992).*

Codage direct de la structure

L'évolution directe de structures manipule celles-ci directement lors des opérations génétiques et aussi lors de leur évaluation. Cette méthode est à la base de la programmation évolutionniste et de la programmation génétique (*cf.* section 1.2.3). Des structures très différentes ont été évoluées de cette façon : des machines à états finis (Fogel et al., 1966), des réseaux de neurones (Angeline et al., 1994b; Angeline et al., 1994a), des graphes (Sims, 1995), des arbres (Koza, 1992). 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 il faut faire face concernent :

« – la *syntaxe*, c'est-à-dire la possibilité de s'assurer de la consistance des structures produites. Une génération aléatoire, une mutation, le croisement de structures ou toute autre manipulation envisagée doivent produire des structures valides ;

« – la *sémantique*, c'est-à-dire s'assurer que de faibles variations de la représentation génétique se traduisent autant que possible par de faibles

variations de la structure, donc du comportement. Par suite, le comportement des structures-enfants produites par reproduction sera le plus souvent proche de celui des structures-parents. Il s'agit d'une propriété de *quasi continuité* entre le génotype et le comportement de la structure décodée qui lui est liée. Cette propriété, bien que non précédemment mentionnée explicitement comme nécessaire au fonctionnement des approches évolutionnistes, est pourtant souvent implicite. C'est la base de l'hérédité. Sans elle, la recherche qui s'appuie sur des variations aléatoires pourraient revenir à une marche au hasard. En rendant impossible l'approche graduelle d'une bonne solution, elle rendrait le mécanisme de sélection souvent inefficace. De fait, les codages qui ont une propriété de continuité sont optimaux au regard des algorithmes évolutionnistes (Mitchell, 1996) ;

« – la *complétude*, c'est-à-dire s'assurer que l'on peut explorer tout l'espace de recherche 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.

Ces trois problèmes font l'objet de beaucoup de travaux (Koza, 1992; Montana, 1995; Angeline, 1994a; Fogel and Stayton, 1994; Angeline, 1996) et constituent un domaine de recherche en soi dans le champ de l'informatique évolutionniste. Les difficultés reposent ici en partie sur la division du génotype en parties qui ont des propriétés différentes (inhérentes à la structure), le plus souvent une hiérarchie. Les modifications induites par les opérateurs génétiques doivent donc prendre en compte ces différents rôles tout en respectant au mieux les contraintes ; par exemple, en programmation génétique, le croisement de deux arbres (*cf.* figure 1.8) échange complètement deux sous-arbres, ce qui fait que pour une variation génotypique de même type, plus le

point de croisement est proche de la racine, plus la variation phénotypique risque d'être forte. Cet exemple fait bien ressortir que c'est la contrainte sémantique qui pose ici le plus problème, la propriété de quasi continuité faisant défaut.

1.3.2 Codages indirects

Les codages directs sont à plusieurs titres insatisfaisants, soit parce que leur pouvoir d'expression est trop faible (codage de paramètres pour une structure existante), soit parce qu'ils ne respectent pas la propriété de quasi continuité (codages directs de structures). Il est alors apparu nécessaire d'aborder une approche indirecte, qui procède à la *construction* de la structure. Le codage est dans ce cas un langage de construction de structure et le processus d'expression pour produire puis évaluer la structure est plus complexe (Angeline, 1995). Ces approches, souvent désignées par le terme générique de *développement*, par analogie avec le modèle biologique du développement, sont les plus sophistiquées pour faire évoluer des structures. Le développement a surtout été utilisé pour fabriquer des structures en réseau, par exemple des circuits électriques (Koza et al., 1996a), ou des réseaux de neurones artificiels (*cf.* (Yao, 1999) section 3.2.2 pour une synthèse). Le modèle d'expression peut opérer par réécritures successives, par exemple des règles de réécritures sur des chaînes ($\mathcal{L}$-*systems* (Lindenmayer, 1968)) dans (Boers and Kuiper, 1992), des règles de réécritures matricielles (Kitano, 1990), ou alors il peut opérer en exécutant des règles représentées sous la forme d'un arbre de dérivation plus ou moins contraint (Kodjabachian and Meyer, 1998; Gruau, 1994; Luke and Spector, 1996). La représentation génétique et celle qui est évaluée peuvent ainsi être de nature complètement différente, et leurs tailles ne sont pas corrélées : il est possible de générer des structures dont la taille

est de plusieurs ordres de grandeur celle de la représentation génétique, ce qui est notamment le cas avec les règles de réécriture des $\mathcal{L}$-systems. Cet effet grossissant combiné à des atomes intrinsèquement discret fait que ces derniers systèmes ne respectent pas la propriété de quasi continuité, puisque de petites variations dans la représentation génétique se traduiront le plus souvent par des variations discontinues notables de la structure.

Grâce à l'utilisation d'un langage, il est possible de générer une structure ex nihilo, de pouvoir mieux apprécier et contrôler quelle partie de l'espace des structures est balayée, et de quelle façon. La grammaire permet de garantir facilement la correction syntaxique des structures. La question est donc de savoir quel langage de construction employer.
Ceux actuellement utilisés sont basés sur des grammaire sans contexte (*context-free* (Chomsky, 1962)) ou des règles de réécriture du type $\mathcal{L}$-systems. Bien que ces approches soient de natures différentes (les règles des grammaires de CHOMSKY sont appliquées itérativement (Chomsky and Miller, 1963), alors que les règles de réécriture des $\mathcal{L}$-systems sont appliquées simultanément (Lindenmayer, 1968)), elles sont comparables, les $\mathcal{L}$-systems sans contexte étant moins contraintes que les grammaires sans contexte au sens de CHOMSKY (Savitch, 1975; Walker, 1974).

Comme dit plus haut, la plupart des $\mathcal{L}$-systems n'ont pas la propriété de quasi continuité. C'est aussi le cas pour les langages sans contexte, précisément à cause de la structure grammaticale. Nous allons expliquer pourquoi en décrivant les problèmes rencontrés pour les deux opérations génétiques courantes (mutation et croisement). La figure 1.9 page 40 va nous aider à illustrer ce propos : il s'agit du cas particulier d'un langage d'arbres sans contexte. Les arbres qui sont évolués sont des représentations arborescentes de programmes exécutés par des neurones artificiels, ce processus de « déve-

loppement » étant illustré dans la sous-figure (b). La sous-figure (a) montre tous les objets informatiques en jeu : la grammaire d'arbre, un exemple de programme, et la vue générale du processus. À gauche, la situation initiale du réseau avec en son centre deux cellules germinales qui vont chacune exécuter leur programme ; à droite le réseau après exécution des programmes.

Lors d'une mutation d'une partie du génome (qui est un mot du langage), il faut le plus souvent opérer d'autres mutations pour rester conforme à la grammaire, car les règles grammaticales induisent une dépendance hiérarchique entre certaines parties du mot. Il s'agit du même problème que pour la programmation génétique, et ce n'est pas un hasard si l'on considère que l'on peut représenter sous forme arborescente la dérivation du mot à partir de la grammaire donnée (arbre de dérivation). Dans l'exemple de la figure 1.9, la mutation d'un nœud interne (en un autre nœud interne produit par la même règle grammaticale) imposerait le plus souvent de muter tous les nœuds en-dessous, car les règles seraient différentes. Pour prendre un exemple familier en informatique, c'est comme si lorsqu'on mutait une expression, on était obligé d'en muter d'autres parties pour que la sous-expression modifiée reste syntaxiquement correcte (par exemple, si l'on a transformé un opérateur en un autre avec une arité différente, il faut muter la partie de l'expression correspondant aux opérandes pour que leur nombre change).

Lors d'un croisement de deux génomes, peut se produire une situation analogue. Pour que la syntaxe soit correcte *a priori*, il n'est possible de croiser deux génomes qu'à certains endroits, et sur des segments de longueurs déterminés par la grammaire et dépendants de l'endroit du croisement, car le croisement ne peut se faire que sur deux segments qui dérivent de la même règle grammaticale. Pour reprendre l'analogie avec la programmation génétique,

si l'on considère l'arbre de dérivation, avoir une syntaxe grammaticalement correcte a priori impose le couple des nœuds (pour chaque arbre parent) à partir duquel l'échange des sous-arbres enracinés en ces nœuds peut se faire. La plupart des croisements imposeront donc un important changement structurel. Dans l'exemple de la figure 1.9, la sous-figure (c) illustre le croisement qui doit échanger des sous-arbres enracinés en une même règle grammaticale, ici celle de membre gauche Level1. À nouveau, pour prendre un exemple familier en informatique, c'est comme si, lorsqu'on croise deux expressions, on se forçait à n'échanger que des sous-expressions complètes (correctement parenthésées si elles sont représentées sous forme complètement parenthésée). Et pour que la syntaxe soit correcte *a posteriori*, il faut faire une analyse syntaxique après avoir échangé du matériel (de façon quelconque) entre les deux génomes. Ceci est rendu possible dans le cas général car une des propriétés des grammaires sans contexte est de pouvoir être analysées syntaxiquement en temps linéaire. Cependant l'analyse syntaxique risque le plus souvent de rejeter les fruits du croisement, rendant cette approche fortement inefficace.

Les fortes dépendances hiérarchiques de certaines parties de la représentation génétique à d'autres rendent donc d'autant plus difficiles les petites variations de la représentation génétique que les mutations ou les croisements sont appliqués à des endroits du génome dérivés d'une règles proche de l'axiome. C'est le même problème que pour le codage direct de la structure.

De plus, ces dépendances impliquent que la conception des opérateurs génétiques doit être faite en fonction de la grammaire pour assurer la correction syntaxique, ce qui nous éloigne des variations aléatoires « aveugles » sur lesquelles nous voulons nous appuyer de façon exclusive (*cf.* section 1.2.1).

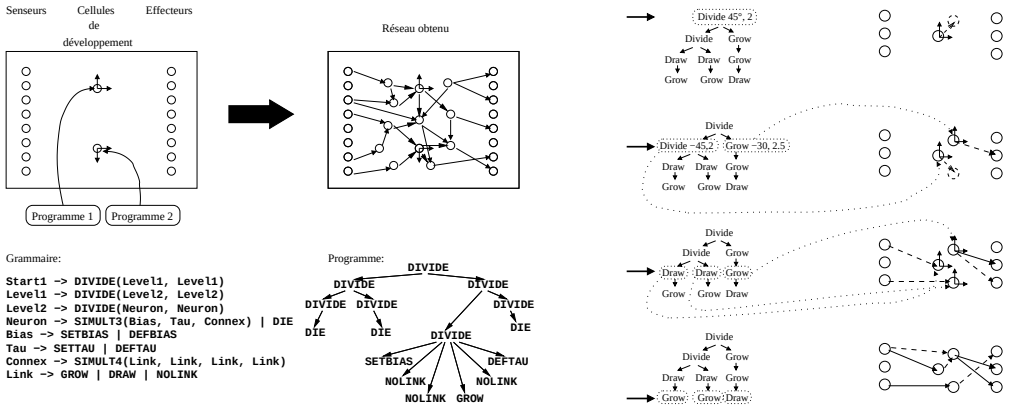
1.4 Récapitulatif

La robotique mobile autonome en environnement réel pose des problèmes difficiles de conception pour les comportements des robots. Les approches délibératives puis comportementales ont été proposées, mais soient s'avèrent peu adaptées au problème, soit exigent trop de travail de la part du concepteur. Appliquer l'évolution artificielle à la conception de comportements de robots mobiles s'avère dès lors très intéressant car le travail de conception complexe pour créer les modules comportementaux et gérer leurs conflits est alors automatisé. Les méthodes d'évolution artificielle les plus utilisées dans cette optique sont les algorithmes évolutionnistes, qui sont une réduction du paradigme de la sélection darwinienne. Alors que la sélection darwinienne peut être vue comme un « algorithme » très simple, l'emploi des algorithmes évolutionnistes ne s'avère, lui, pas simple du tout. En ne retenant que l'aspect phylogénétique de la sélection darwinienne, les algorithmes évolutionnistes créent le problème de l'évaluation des individus (la fitness) et perdent ce qui constitue le moteur de la sélection darwinienne.

Par ailleurs, nous désirions utiliser une architecture composée d'un système multi-agent, afin de mieux gérer la distribution, la concurrence et l'asynchronisme des tâches. Or, l'idée qui guide nos travaux de recherche est que justement grâce aux modèles multi-agents, il est possible d'opérer un « retour aux sources » de la sélection darwinienne, pour peu que ce soient des modèles de systèmes multi-agents en développement, que la survie des agents soit conditionnée par les ressources dont ils arrivent à prendre le contrôle, et que l'on dispose d'un « bon » codage génétique de structures décrivant les comportements des agents. Une revue critique des principales méthodes existantes a permis de constater qu'aucune ne présentait simultanément les

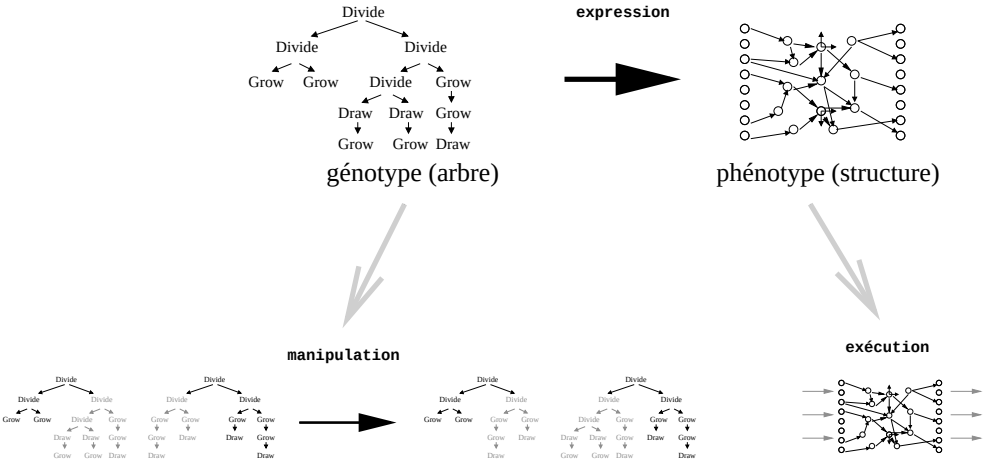
propriétés principales d'un tel codage (pouvoir d'expression et quasi continuité).

Compte tenu de ceci, au chapitre suivant, nous allons donc présenter plus précisément comment le développement de systèmes multi-agents peut être envisagé pour constituer un système multi-agent adaptatif qui puisse contrôler un robot, et les principes que nous nous donnons pour obtenir un codage génétique ayant les propriétés voulues.



(a) vue générale

(b) quelques étapes du développement



(c) programme et réseau de neurones

FIG. 1.9 – *Codage indirect d'une structure, ici un réseau de neurones via un programme respectant une grammaire d'arbre context-free (d'après (Kodjabachian, 1998)). Les mutations génèrent un sous-arbre enraciné au point de mutation, respectant la grammaire, et les croisement se font en des nœuds grammaticalement équivalents, échangeant ainsi des branches des arbres croisés.*

Chapitre 2

Principes [1]

Les approches comportementales donnent de bons résultats pour la conception de comportements de robots, mais posent de gros problèmes de conception. Les approches évolutionnistes paraissent donc les mieux adaptées pour la conception de comportements de robots. Nous les choisissons pour construire une architecture multi-agent, qui présente l'avantage d'être à la fois modulaire, distribuée, dynamique et asynchrone, répondant ainsi aux besoins de la robotique mobile dans un environnement non contrôlé. Ces approches évolutionnistes permettent de concevoir automatiquement les structures tenant lieu d'architecture des agents, et l'évolution de l'ensemble du système de faire les découpages comportementaux de façon nécessaire, incrémentale si besoin, pour le robot (et non pas avec le biais de l'observateur-concepteur).

Parmi les approches utilisées pour la conception de structures, certaines ont une propriété de *quasi continuité*, propriété présente le plus souvent implicitement et qui permet aux méthodes sélectionnistes de fonctionner efficacement, sans biais dans les opérateurs génétiques, et d'autres ont un fort

1. Les concepts développés dans ce chapitre sont en partie le fruit d'un travail commun avec Sébastien Picault, alors doctorant de l'équipe Miriad (Picault, 2001).

pouvoir expressif, permettant de réellement *construire* des structures à partir de rien, mais aucune ne présente ces deux propriétés simultanément.

Nous allons dans un premier temps mieux expliciter et ériger en principes pour des approches de comportements d'agents évoluables les propriétés de quasi continuité et de pouvoir expressif. Ceci est l'objet de la section 2.1 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 codage basé sur l'expression de gènes à l'aide d'une pile (section 2.2). Cette approche a le mérite de permettre d'évacuer de façon simple et élégante les difficultés de conception des opérateurs génétiques, ce qui va nous permettre par la suite de mieux nous concentrer exclusivement sur l'évolution des comportements.

2.1 L'Éthogénétique

L'Éthogénétique (*Ethogenetics* (Picault and Landau, 2001b)) 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 1.3). Les deux principales 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 présenter une propriété de quasi continuité, et avoir un (fort) pouvoir expressif.

2.1.1 Quasi 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 quasi continuité que nous recherchons définie plus précisément : il s'agit de la continuité de 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 (Kupiec and Sonigo, 2000) peut générer de l'ordre à partir de désordre (Dennett, 1995), sans biais téléologique et sans informer explicitement les opérateurs de variation, c'est-à-dire d'exploration. 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 qui sont continues sont optimales (Mitchell, 1996), le « paysage de fitness » y étant continu en fonction des génotypes. Ce n'est pas toujours strictement le cas en pratique, mais il est cependant possible de se contenter de fonctions presque partout continues.

Afin d'obtenir cette propriété de quasi continuité, nous suggérons d'utiliser des fonctions d'expression du génotype en comportement qui ne reposent pas sur une représentation génétique structurée. Dans les modèles évoqués précédemment permettant de générer des structures ex nihilo (*cf.* section 1.3),

ce n'était pas le cas. L'utilisation de grammaires context-free ou de règles de réécriture type $\mathcal{L}$-systems induit une hiérarchie et des relations fortes entre les parties de la représentation génétique. Par ailleurs, il est souvent difficile, dans le cas des $\mathcal{L}$-systems, d'estimer l'ampleur d'une variation même infime du génome, à la seule vue de ses dépendances structurelles (les règles de réécriture).

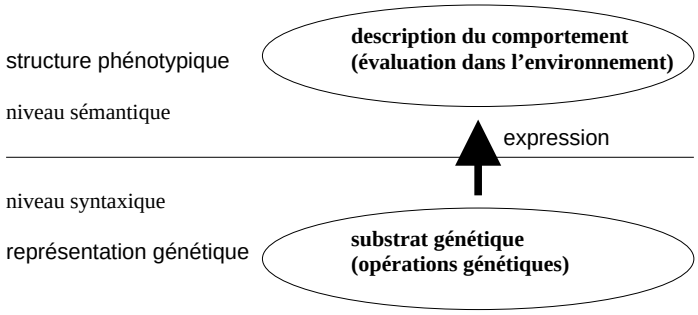


FIG. 2.1 – *Séparation syntaxe-sémantique. Le substrat génétique (représentation génétique) est exprimé en une description du comportement (structure phénotypique). Nous préconisons une séparation entre ces niveaux, correspondant aux niveaux syntaxique et sémantique.*

La hiérarchie et les interdépendances entre parties éventuellement éloignées de la représentation génétique font que l'expression de parties distantes du génotype sont susceptibles d'avoir un fort impact sur l'expression d'autres. Par exemple, lors de l'utilisation d'une grammaire context-free, l'utilisation explicite d'une représentation arborescente (l'arbre de dérivation) fait que les variations opérées à la racine ou aux feuilles n'ont pas du tout le même impact, l'expression de ces dernières étant complètement subordonnée à l'expression des premières. Des variations minimes sur des nœuds proches de la racine se traduiront presque systématiquement par un changement important du comportement de l'ensemble. Il y a donc toutes les chances pour que l'impact sur la structure évaluée soit grand.

En d'autres termes, nous préconisons lors de la conception de cette fonction de séparer le niveau « syntaxique » (celui de la représentation génétique), du niveau « sémantique » (celui de la structure qui va donner lieu au comportement) (*cf.* figure 2.1). S'il n'y a pas de séparation, le niveau sémantique contraint trop le niveau syntaxique, ce qui conduit aux problèmes évoqués précédemment.

2.1.2 Pouvoir expressif

Notre objectif est de produire de façon automatique des comportements d'agents. Pour ce faire, il est souhaitable de ne pas avoir à donner de limites à l'espace de recherche. Nous souhaitons en effet pouvoir obtenir des structures (décrivant le comportement) de complexité arbitraire, la sélection se chargeant de déterminer le niveau de complexité nécessaire par 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, automates finis, S-expressions, etc.).

Ce fort pouvoir expressif devrait être celui d'une structure compréhensible et modulaire, afin de :

« – 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 (*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.

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

Les modèles existants ne satisfont pas simultanément les principes de l'Éthogénétique. D'une part, les codages 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 codages indirects sont plus flexibles et plastiques, mais d'autres problèmes surtout liés à la quasi continuité se posent, qui rendent complexes la conception des opérateurs génétiques.

Le principe d'expression de gènes que nous présentons ci-après est un modèle de codage indirect, construit dans le but de satisfaire les principes de l'Éthogénétique. Il s'agit d'un modèle d'interprétation à l'aide d'une pile (*Stack-Based Gene Expression* (Landau and Picault, 2002b)).

L'interprétation à l'aide d'une pile est un modèle d'expression simple, déjà employé en informatique pour des langages comme PostScript (Systems, 1985) ou Forth (Moore and Leach, 1970). 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 (que nous avions formulé ainsi : « de faibles variations de la représentation génétique devraient se traduire le plus souvent par de faibles variation 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. 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 le principe adopté (section 2.2.2), nous allons présenter rapidement quelques travaux apparentés (section 2.2.1). Après quoi, nous évoquerons le modèle biologique qui est notre source d'inspiration (section 2.2.3).

2.2.1 Travaux apparentés

Il faut noter qu'il y a très peu de modèles en informatique évolutionniste qui mettent en jeu un interprète à pile : il y en a à notre connaissance deux, et le second s'inspire du premier. De plus aucun ne se sert du langage pour construire une structure. Le langage est toujours utilisé directement comme description d'un comportement.

PERKIS et *Stack-Based Genetic Programming*

C'est PERKIS qui a le premier utilisé avec succès un modèle de programmation génétique à l'aide d'une pile (*Stack-Based Genetic Programming* (Perkis, 1994)). Il ne l'a cependant pas utilisé pour construire des structures, seulement pour exécuter un programme. Un vecteur contenant les instructions du langage à pile est exécuté par l'interpréteur. Les opérateurs génétiques utilisés sont des plus classiques : des croisements à un point, similaires à ceux des algorithmes génétiques, et des mutations qui pour chaque cellule du vecteur transforment avec une certaine (faible) probabilité l'instruction en une autre. La souplesse inhérente au langages à piles fait qu'il n'y a pas besoin de se préoccuper plus avant des conséquences des manipulations génétiques,

puisque n'importe quel vecteur en entrée peut être accepté, pour peu que l'on ignore les erreurs levées par les instructions qui ne peuvent pas s'exécuter – principalement faute d'opérandes dans la pile. Cette approche s'est avérée compétitive par rapport à la programmation génétique classique sur le terrain où celle-ci excelle et constitue la référence : la construction d'expressions mathématiques résolvant un problème donné (régression et approximation de fonction). Il faut tout de même souligner que même à performances moindres, il y aurait encore pour le concepteur-utilisateur un gain qu'il est impossible de négliger, lié à la facilité d'appréhension et donc d'utilisation de la méthode. Il s'agit du fait qu'il n'a pas été nécessaire de développer des trésors d'imagination pour *défaire ce qui a été fait*, c'est-à-dire concevoir et évaluer des opérateurs génétiques spécifiquement biaisés (« *défaire* »), si ce n'est pour la tâche donnée, du moins pour pallier les contraintes de dépendances structurelles propres à la programmation génétique, que nous avons évoquées au chapitre précédent (dépendances structurelles explicitement introduites dans le modèle, donc « *ce qui a été fait* »). Les opérateurs génétiques utilisés sont *complètement génériques* et il n'est nul besoin de les spécialiser. La seule limite posée est la longueur des vecteurs, qui pourrait trop grandir du fait que tous les vecteurs de la population initiale ne sont pas de longueur identique.

Spector et *HiGP*, puis *Push*, *PushGP* et *PushPop*

Stoffel et Spector ont repris cette approche en la parallélisant dans *HiGP* (Stoffel and Spector, 1996). Ils l'ont ensuite enrichie en ajoutant dans le langage à pile des opérateurs permettant d'opérer sur le flux d'instruction en train d'être exécuté (Spector and Stoffel, 1996b; Spector and Stoffel, 1996a). Ils ont donc en particulier introduit le mécanisme de la récursion. Pour éviter les possibles problèmes de bouclage infinie, ils ont arbitraire-

ment limité le nombre d'instructions exécutables par leur interprète. L'ajout de ces instructions rend l'expression d'un programme potentiellement auto-modifiante. Ils ont observé grâce à elles une accélération significative de la convergence.

Plus récemment, SPECTOR s'est encore plus éloigné du modèle initial en élaborant des modèles assez complexes avec plusieurs piles typées (basé sur le langage *Push* (Spector, 2001; Crawford-Marks and Spector, 2002)). L'une des piles a la particularité de ne contenir que du code, alors manipulable comme une donnée. Il faut noter, cependant, qu'à cause du parenthésage qui sert à délimiter les blocs, ce langage n'a pas la propriété de localité que nous recherchons.

Dans *PushGP*, SPECTOR fait évoluer des programmes écrits en Push avec un algorithme évolutionniste classique. À cause de la non localité les opérateurs génétiques doivent tenir compte de la grammaire du langage. En fait, ces opérateurs sont du même type qu'en programmation génétique. La seule différence est celle de la représentation génétique : au lieu de muter un sous-arbre ou d'en échanger deux lors d'un croisement, il faut ici tenir compte du parenthésage et muter une sous-expression ou croiser des sous-expressions.

Enfin, dernièrement SPECTOR s'est servi de Push dans *PushPop*. PushPop est un modèle informatique sélectionniste alternatif intéressant. Il s'agit d'une variation du principe des algorithmes évolutionnistes. Classiquement, les individus sont évalués séparément chacun dans leur environnement et se voient crédités d'une fitness qui servira à opérer la sélection. Ce qui est original est que *chaque individu a la charge de se reproduire*, il n'y a pas d'opérateurs génétiques externes aux individus (Spector and Robinson, 2002b; Spector and Robinson, 2002a). Chacun a, pour ce faire, accès à son code propre et à celui des autres individus, et une pile supplémentaire a été introduite pour

contenir le code de l'enfant produit par l'exécution du programme. Le code qui a été accumulé dans cette pile au cours de l'exécution du programme constitue le code de l'enfant. Si le programme d'un individu ne lui permet pas de se reproduire alors il n'aura pas de descendance ; si la population baisse en-deçà d'un certain seuil elle est complétée par des individus générés aléatoirement. La fitness devient partiellement implicite, en ce sens que la capacité à se reproduire n'a pas besoin d'être mesurée explicitement. En effet, si justement elle fait défaut, ce caractère ne sera pas héréditaire par l'absence de descendance.

Cependant, dans notre perspective, ces langages (HiGP et Push) ne sont pas utilisés pour produire des structures. De plus, pour Push, le langage de pile n'est pas local et nécessite de respecter des contraintes syntaxiques aussi contraignantes que celles de la programmation génétique. Ce langage a perdu la souplesse de manipulation que HiGP avait, l'interprète à pile ne pouvant plus accepter n'importe quel programme donné en entrée et un contrôle syntaxique devenant nécessaire.

Enfin, dans PushPop, la reproduction est tout de même explicitement contrainte syntaxiquement et sémantiquement pour éviter un problème de convergence prématurée : les enfants identiques aux parents sont rejetés, et les individus identiques sont retirés de la population pour qu'il n'en reste qu'un exemplaire unique.

2.2.2 Description du principe

Dans le modèle que nous proposons, le décodage du génome est exécuté en deux étapes : une traduction puis une interprétation – l'évaluation séquentielle de l'action associée à chacun des lexèmes. Comme la représentation génétique ne constitue qu'un support syntaxique dénué de toute signification,

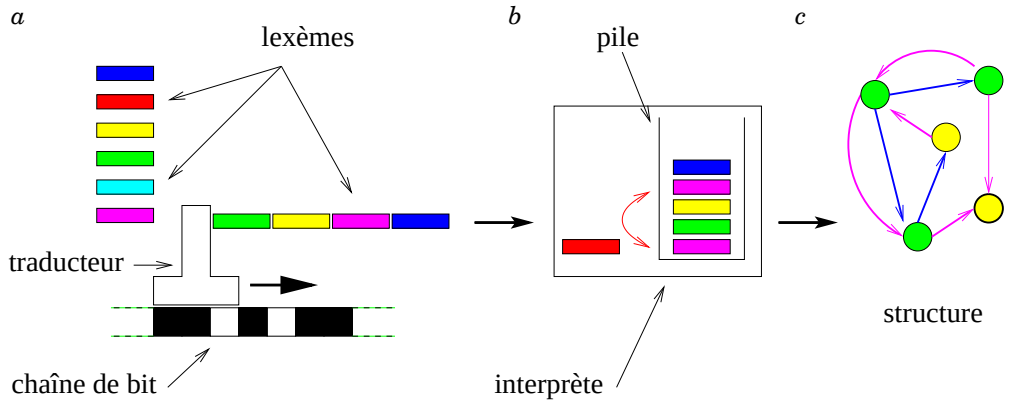


FIG. 2.2 – *Expression de gènes à l'aide d'une pile. a. la traducteur lit une chaîne de bits et produit un flux de lexèmes ; b. les lexèmes sont envoyés à un interprète, qui peut empiler et/ou effectuer une action liée au lexème ; c. la structure finale est dépilée*

notre choix s'est porté sur une chaîne de bits. La chaîne de bits présente le triple avantage 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. 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 l'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 2.2 résume le processus d'expression génétique.

La traduction

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

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

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

« – les instructions pour le langage à pile, que nous 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, que nous 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 présents dans la pile.

Notons que s'il y a plus de codons (qui sont en nombre puissance de 2) que de lexèmes, le code génétique peut être redondant (surjection), en associant plusieurs codons au même lexème. Ceci nous permet de jouer éventuellement sur les probabilités de tirage aléatoire de chacun des lexèmes. Par défaut, nous ferons toujours en sorte que les codes génétiques soient les plus équiprobables et « symétriques » (*i.e.* même probabilité de tirage aléatoire pour des lexèmes ayant une sémantique ou une action exactement opposée, par exemple deux lexèmes de pile exprimant deux actions opposées, tels que `pushRoll` et `popRoll`, présentés plus avant dans le tableau 3.1 page 71).

L'interprétation et le langage

L'interprète est générique, il est le même quelle que soit la structure décrite ou le langage à pile utilisé. En effet, le langage repose sur le compor-

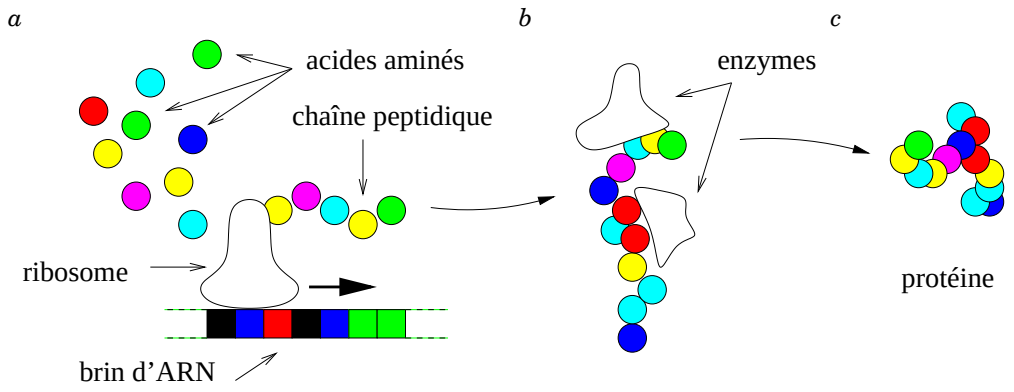


FIG. 2.3 – *Synthèse d'un protéine. (a) le ribosome (une enzyme) lit une séquence d'acides nucléiques (brin d'ARN) et assemble des acides aminés en une chaîne polypeptidique ; (b) la chaîne polypeptidique se replie, avec le concours d'enzymes... ; (c) ...en la protéine finale*

tement des lexèmes choisis, qui savent utiliser la pile. Classiquement, pour les langages à pile, si une instruction ne peut pas être exécutée – faute d'opérandes dans la pile en particulier – alors elle peut être ignorée, ou plus exactement l'erreur renvoyée suite à l'impossibilité d'exécuter l'instruction est ignorée. Cette propriété rend le langage très souple, et permet à l'interprète de reconnaître n'importe quel flux de lexèmes – et par conséquent n'importe quelle chaîne de bits donnée en entrée au système.

2.2.3 Le modèle biologique

Le modèle biologique qui est la source d'inspiration de notre approche évolutionniste de construction de structures est la synthèse de protéines dans la cellule (la figure 2.3 en résume les étapes principales). Nous avons fait ce choix à la fois pour des raisons « historiques » (Picault et al., 1997), parce que c'est à ce premier niveau d'organisation (des molécules) qu'ont lieu les phénomènes les plus fondamentaux de l'ontophylogenèse et surtout

parce qu'une caractéristique de la production des protéines est de produire directement une structure, caractérisée par sa *forme*, à l'origine des propriétés de la protéine.

Il s'agit d'un modèle à codage indirect, puisque la représentation génétique n'est pas une protéine – donc les génotypes et phénotypes sont structurellement distincts. À noter que les modèles en informatique évolutionniste n'ont pas nécessairement respecté ce principe, comme nous l'avons précédemment détaillé section 1.3.

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 le 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 cellule, 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 la donnée initiale, la chaîne peptidique.

Notre système est une réduction du modèle biologique, notamment car il n'y a pas d'interventions extérieures lors de la formation de la structure. Il en conserve tout de même certains caractères. 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 morphogenèse 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 analogue, mais centralisé, à celui des enzymes qui agissent sur la chaîne peptidique en cours de repliement.

Chapitre 3

Modèle et implémentations

Afin de pouvoir évaluer notre modèle sur plusieurs applications et de pouvoir le comparer à d'autres sur une même application, il nous est apparu nécessaire 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[1]. Nous le décrivons section 3.1.

Les principes de l'Éthogénétique 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[2], que nous utiliserons comme des automates finis et non déterministes. Cette

1. SFERES a été développé en coopération avec Stéphane Doncieux, doctorant de l'équipe AnimatLab. Paul Guyot, alors stagiaire de maîtrise de l'équipe Miriad, a participé à son implémentation.
2. Augmented Transition Network

instanciation s'est faite dans le modèle ATNoSFERES[3] que nous présentons section 3.2. Pour nos expériences, ATNoSFERES a tout naturellement été implémenté dans SFERES.

3.1 SFERES

SFERES[4] (Landau et al., 2001a; Landau et al., 2002a) est un framework et une plate-forme[5] d'évolution artificielle et de simulation multi-agent.

SFERES est constitué de deux parties. La première est la partie évolutionniste qui sert de support aux algorithmes évolutionnistes comme ceux présentés section 1.2.2. La seconde est la partie simulation et permet de développer et déployer des simulations multi-agents. 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.

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 (Whitley, 1989; Wall, 2000). Ce couplage est décrit section 3.1.3. 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.

3. ATNoSFERES a été conçu en collaboration avec Sébastien PICAULT, alors doctorant de l'équipe Miriad.

4. *cf.* `http://miriad.lip6.fr/SFERES/`

5. codée en langage C++ sous Linux/EGCS, prochainement distribuée sous licence libre.

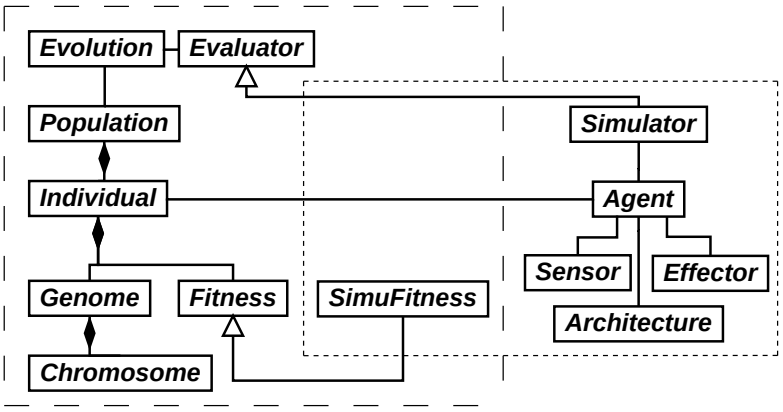


FIG. 3.1 – *Diagramme UML des classes de SFERES. Encadrée de tirets, la partie évolutionniste, et encadrée de pointillés, la partie simulation.*

La figure 3.1 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 3.1.3).

Nous allons d'abord décrire la partie évolutionniste du framework (section 3.1.1), puis la partie simulation (section 3.1.2). Après la description du couplage entre ces deux parties (section 3.1.3), nous décrivons les points d'ouvertures du framework (*hot spots* (Schmid, 1997)), c'est-à-dire que nous détaillons ce qui doit être dérivé en fonction de que l'on veut faire (section 3.1.4). Enfin, nous concluons sur notre utilisation de SFERES pour ce qui nous intéresse ici (section 3.1.5).

3.1.1 Partie évolutionniste

La figure 3.2 rassemble les classes de la partie évolutionniste du framework. Ces classes peuvent être regroupées en deux parties : le moteur d'évolution et l'individu.

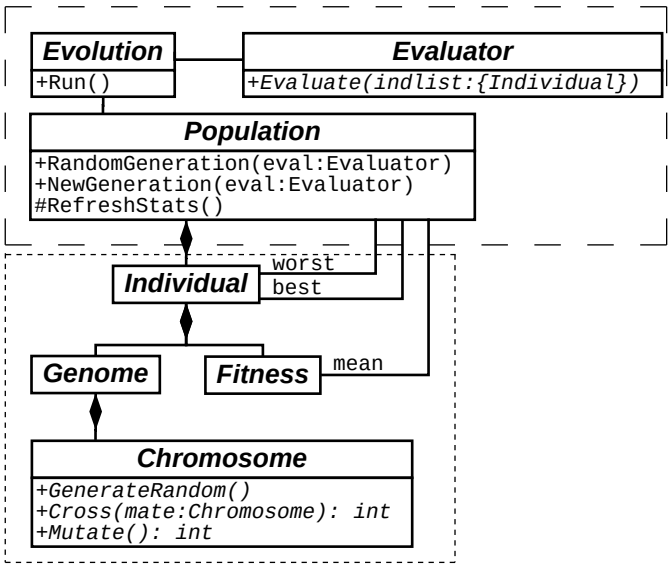


FIG. 3.2 – *Diagramme UML des classes de la partie évolutionniste de SFERES. Encadré de tirets, le moteur d'évolution, et encadré de pointillés, l'individu.*

Moteur d'évolution

Le moteur d'évolution (cadre en tirets sur la figure 3.2), regroupe les classes gérant le déroulement de l'algorithme évolutionniste.

La classe `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 l'opération de sélection de l'algorithme évolutionniste (*cf.* figure 1.5). 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 classe `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.

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 (classe `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`.

3.1.2 Partie simulation

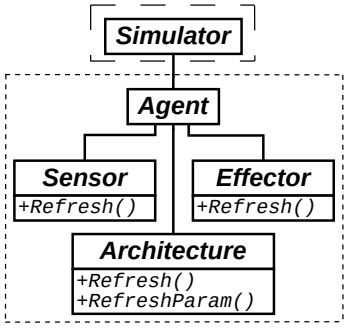


FIG. 3.3 – *Diagramme UML des classes de la partie simulation de SFERES. Encadré de tirets, le moteur de simulation, et encadré de pointillés, l'agent.*

La figure 3.3 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 simulateur existant (Gutknecht and Ferber, 1997) 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 3.1.3.

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 3.1.3). 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`.

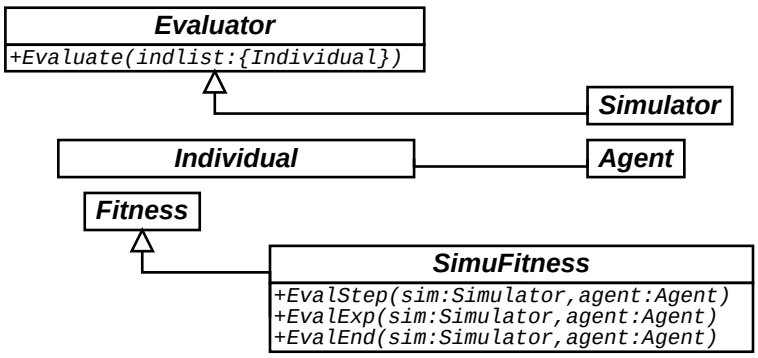


FIG. 3.4 – *Diagramme des classes du couplage entre les parties évolutionniste et simulation de SFERES*

3.1.3 Couplage entre les parties évolutionniste et simulation

La figure 3.4 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 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 définis 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, car une interface imposée restreindrait la portée générale du framework.

3.1.4 Dérivations de classes

L'implémentation de nouvelles expériences ou de variantes d'expériences existantes se fait simplement par dérivation des classes concernées.

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 précédemment 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, certaines des classes qui vont l'utiliser (`Architecture` le plus souvent).

3.1.5 Conclusion sur SFERES

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

Une dérivation du simulateur a été développée pour s'interfacer avec les robots de MICRobES, pour nos propres expérimentations. Cette interface se fait grâce au protocole Saphira fourni avec les robots. D'autre part, la plate-forme a servi de support pour l'implémentation d'ATNoSFERES, notre modèle d'évolution d'ATN appliquant le principe de l'expression de gènes à l'aide d'une pile (*cf.* section 2.2).

3.2 ATNoSFERES

ATNoSFERES[6] (Landau et al., 2001b) 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

6. *cf.* `http://miriad.lip6.fr/ATNoSFERES/`

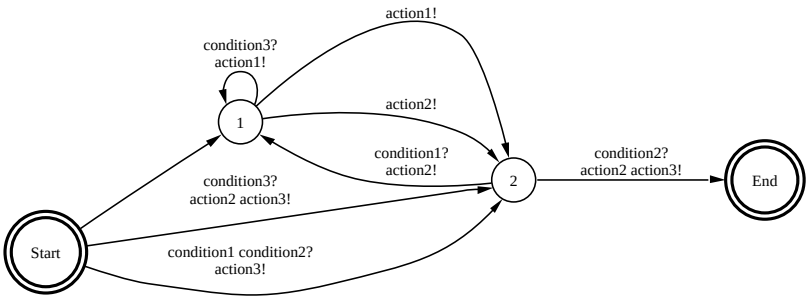


FIG. 3.5 – *Un exemple d'ATN.*

qui ont d'abord été utilisés dans le cadre du traitement automatique du langage (Woods, 1970; Pitrat, 1985) ; ils ont aussi par la suite été utilisés pour modéliser des interactions et représenter des comportements (Winograd and Flores, 1986). Pour nous, l'ATN est une description du comportement, un automate fini et non déterministe.

Nous allons d'abord décrire comment le graphe est utilisé pour décrire un comportement (section 3.2.1). Puis nous décrivons comment il est produit à partir d'une chaîne de bits (section 3.2.2), en suivant le modèle expliqué au chapitre précédent. Enfin, nous décrivons les dérivations de classes qui ont été nécessaires pour implémenter ATNoSFERES dans le framework SFERES (section 3.2.3).

3.2.1 De l'ATN au comportement

À un type d'agent dont on veut décrire le comportement avec l'automate est associé un ensemble de lexèmes correspondant d'une part à des *actions* qui peuvent être exécutées par cet agent, et d'autre part à des *conditions* qui portent sur des perceptions de l'environnement et qui vont permettre les

prises de décision. Sur les graphes donnés en exemple, les actions sont notées avec un point d'exclamation (par exemple `uneAction!`) et les conditions avec un point d'interrogation (par exemple `uneCondition?`).

Le graphe de comportement d'un agent comporte systématiquement un nœud de départ (appelé par la suite `Start`) et un nœud final (`End`). Les nœuds sont reliés par des arcs qui peuvent être étiquetés par un ensemble de conditions et par une séquence d'actions (voir l'exemple donné dans la figure 3.5).

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`.

3.2.2 De la chaîne de bits à l'ATN

La construction du graphe se fait selon le principe expliqué au chapitre précédent (*cf.* section 2.2). Il ne nous reste donc qu'à expliciter ce qui est particulier et nécessaire au codage pour construire les graphes. Un exemple simple de construction est détaillé ensuite pas-à-pas.

Le traducteur

Le traducteur est complètement défini par le code génétique qu'il utilise. Comme dit section 2.2.2 il y a deux types 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 l'éventuelle action associée à chacun (*cf.* le tableau 3.1 page 71) :

« – 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 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).

Ces instructions de construction sont des primitives classiques pour construire un graphe. Rappelons que si une instruction ne peut pas être exécutée, elle est ignorée (*cf.* section 2.2.2).

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 et 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 nouveaux arcs. En second lieu, des connexions implicites particulières sont faites : d'une part, connecter le nœud `Start` à tous les nœuds n'ayant pas d'arc entrant ou pas d'autres arcs entrants que ceux provenant d'eux-mêmes, et d'autre part, connecter à `End` tous les nœuds n'ayant pas d'arcs sortant.

Exemple

Pour un exemple de fonctionnement *cf.* la figure 3.6 page 72.

lexème	**action résultante**
lexèmes de pile	manipule la pile
noop	ne fait rien [a]
swap	échange les deux lexèmes au sommet de la pile
dup	empile une copie du premier lexème d'agent rencontré
del	retire de la pile le premier lexème d'agent rencontré
dupNode	empile une copie du premier nœud rencontré
delNode	retire de la pile le premier nœud rencontré [b]
popRoll	dépile un lexème et le met en fond de pile
pushRoll	retire le lexème de fond de pile et l'empile
lexèmes de structure	crée des constituants de la structure
lexèmes d'ATN	crée de nouveaux nœuds ou connexions entre nœuds
node	crée un nouveau nœud et l'empile
connect	crée un arc du premier au second nœud rencontrés dans la pile [c] ; l'étiquette avec l'ensemble des conditions et la liste des actions présents jusqu'au second nœud ; retire de la pile les actions et conditions présentes jusqu'au second nœud
connectStart	crée un arc de `Start` au premier nœud rencontré dans la pile ; l'étiquette avec l'ensemble des conditions et la liste des actions présents jusqu'au nœud ; retire de la pile les actions et conditions présentes jusqu'au nœud
connectEnd	crée un arc du premier nœud rencontré dans la pile à `End` ; l'étiquette avec l'ensemble des conditions et la liste des actions présents jusqu'au nœud ; retire de la pile les actions et conditions présentes jusqu'au nœud
lexèmes d'agent	lexèmes d'action et de condition, spécifiques au type d'agent
`condition?`	empiler la condition
`action!`	empiler l'action

TAB. 3.1 – *Le langage de construction d'ATN.*

[a] sauf mention explicite, ce lexème n'est pas représenté dans nos codes génétiques

[b] il peut s'agir d'une copie ; dans ce cas d'autres copies du nœud demeurent dans la pile

[c] il peut s'agir de deux copies du même nœud ; dans ce cas l'arc sera une boucle sur le nœud

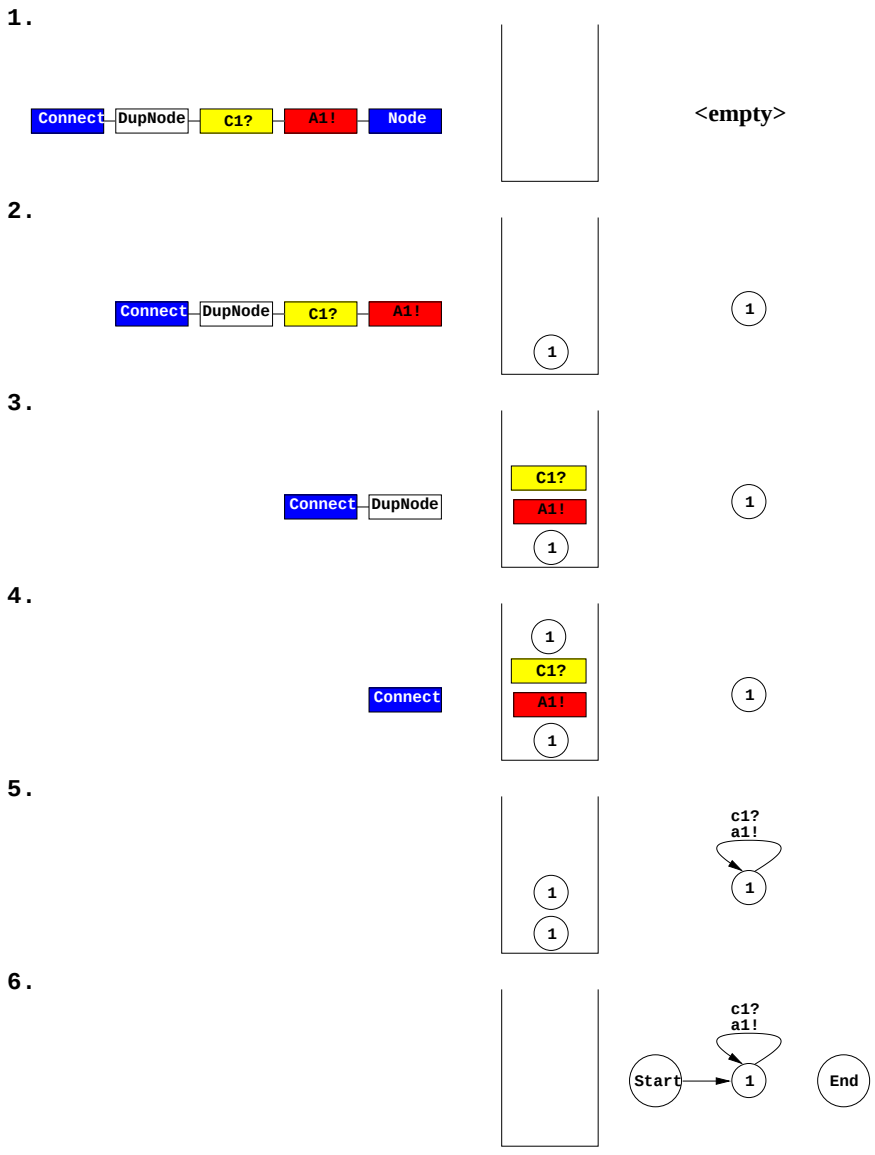


FIG. 3.6 – *Exemple de construction d'un ATN à l'aide d'une pile. 1. état initial : pas de structure et pile vide ; 2. un premier nœud est créé ; 3. les conditions et actions sont empilées ; 4. DupNode empile une copie du premier nœud rencontré en remontant la pile ; 5. le nœud 1 est connecté à lui-même ; 6. état final, après une connexion implicite au nœud* `Start` *car le nœud n'a pas d'arcs entrants autres que bouclés. La présence de cet arc bouclé fait que le nœud n'est pas connecté implicitement à* `End`

3.2.3 Implémentation d'ATNoSFERES en SFERES

L'implémentation d'ATNoSFERES en SFERES a été une tâche assez simple. Comme ATNoSFERES est essentiellement un codage génétique, les classes de SFERES à dériver sont le `Chromosome` et l'`Architecture`. L'implémentation d'ATNoSFERES a alors consisté à établir le lien entre ces classes dérivées. La figure 3.7 contient les principales classes : l'architecture dérivée `ATNArchitecture` contenant l'ATN (détaillée figure 3.8), le chromosome dérivé `BitStringChromosome` contenant la chaîne de bits (détaillé figure 3.9), le traducteur et l'interprète qui ne dérivent d'aucune classe de SFERES (tous deux sont détaillés figure 3.10).

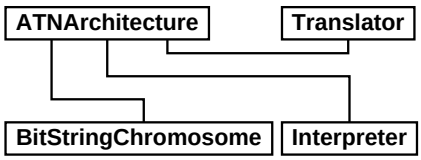


FIG. 3.7 – *Diagramme UML des classes principales d'ATNoSFERES.*

C'est l'architecture qui orchestre la fabrication de son graphe en étant associée au chromosome qui contient l'information à décoder, au traducteur qui connaît le bon code génétique et à l'interprète qui le produit en se servant du traducteur sur la chaîne de bits. L'implémentation d'ATNoSFERES peut être scindée en trois parties. La première rassemble ce qui concerne l'architecture, la deuxième ce qui concerne le chromosome, et la dernière l'interprétation.

Architecture

L'architecture (classe `ATNArchitecture`) est composée d'un graphe (classe `ATNGraph`) représentant l'ATN, lui-même composé d'au moins deux nœuds (classe `ATNNode`) – ces deux nœuds minimaux représentent bien sûr `Start` et `End`. Les nœuds sont connectés par des arcs (classe `ATNEdge`), chaque arc

ayant une origine et une destination. Tout arc porte l'ensemble des conditions (classe `ConditionToken`) et la liste des actions (classe `ActionToken`) qui lui sont dévolus. Lors de son utilisation, l'architecture garde une trace du nœud et de l'arc courants.

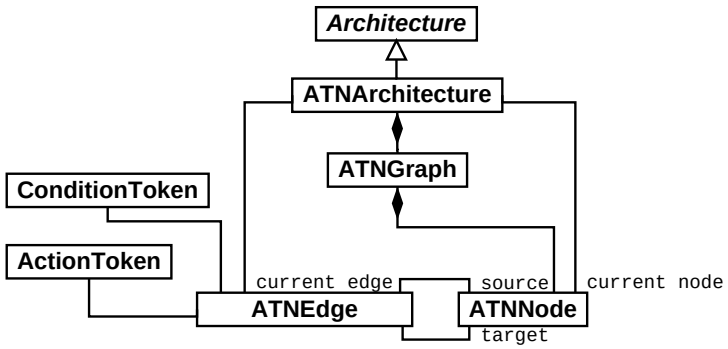


FIG. 3.8 – *Diagramme UML des classes de la partie architecture d'ATNoSFERES. La hiérarchie de classes dans laquelle s'inscrivent* `ConditionToken` *et* `ActionToken` *est représentée figure 3.10.*

Chromosome

Le chromosome (classe `BitStringChromosome`) contient la chaîne de bits. Il est capable de se générer au hasard (`GenerateRandom()`), de faire un croisement à deux points avec un semblable (`Cross()`) ou de muter (`Mutate()`). En particulier, cette mutation se fait en inversant chaque bit avec une probabilité `MutationRate` et en ajoutant ou retirant (avec une même probabilité) un paquet de `InsDelLength` bits avec une probabilité `InsDelRate`. La taille du paquet de bits sera dans nos expériences celle d'un codon pour l'expérience en cours.

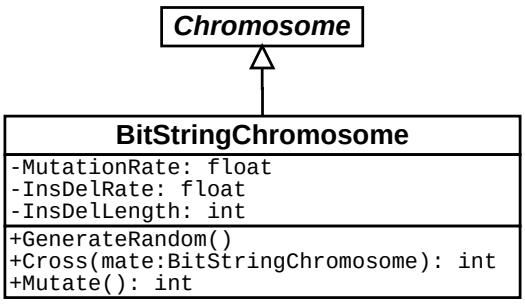


FIG. 3.9 – *Diagramme UML de la classe de chromosome d'ATNoSFERES.*

Interprétation

La partie interprétation inclut l'interprète (classe `Interpreter`), sa pile (classe `Stack`) à lexèmes (classe `Token`), le traducteur (classe `Translator`) caractérisé par son code génétique (classe `GeneticCode`). Le code génétique associe à un paquet de bits de taille donnée, un lexème. La hiérarchie de classes des lexèmes est telle que détaillée section 3.2.2. Tout lexème sait s'empiler (`Push()`). Les lexèmes de piles (classe `StackToken`) comme *swap* (classe `SwapToken`) par exemple, exécutent à cette occasion une action, et sont aussitôt éliminés. Les lexèmes de structure (classe `StructureToken`) se partagent entre les lexèmes d'ATN (classe `ATNToken`) comme par exemple la classe `NodeToken`, et les lexèmes d'agent (classe `AgentToken`). Un `NodeToken` a la particularité d'être associé à un `ATNNode`, un nœud du graphe. Un même `ATNNode` peut être associé à plusieurs `NodeTokens`, si le lexème `DupNode` a été exécuté par l'interprète. Pour finir, les lexèmes d'agent savent s'empiler, et n'exécutent aucune autre action à cette occasion. Ils sont eux-mêmes soit de type `ConditionToken`, soit de type `ActionToken`, comme par exemple respectivement la classe `FreeNorthConditionToken` et la classe `GoNorthActionToken`. Un lexème de condition doit pouvoir fournir la valeur de la condition qu'il représente, cette fonction étant assurée par la méthode `Test()` qui de façon générale retourne un flottant dans [0,1]. De façon ana-

logue, une action est effectuée lors de l'invocation de sa méthode `Do()`, cette action pouvant échouer ou réussir et retournant donc un booléen.

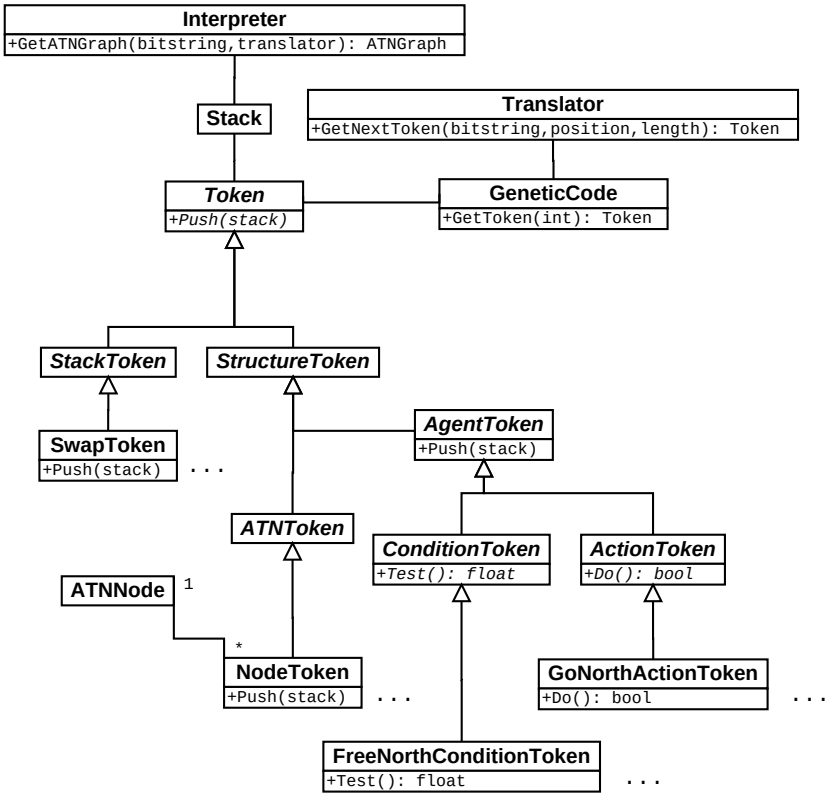


FIG. 3.10 – *Diagramme UML des classes de la partie interprétation d'ATNoSFERES.*

Chapitre 4

Expérimentations

Comme nous l'avons vu dans le chapitre précédent, le modèle ATNoSFERES a été implémenté à l'aide de la plate-forme SFERES. Des expériences préliminaires ont été réalisées pour vérifier l'adéquation du modèle implémenté avec la formulation théorique qui lui a donné naissance. Il s'agissait de vérifier que les principes de l'Éthogénétique et les propriétés prédites pour l'expression à l'aide d'une pile sont bien observées expérimentalement sur cette instance. D'autres expériences ont eu pour but de vérifier si ce modèle, répondant aux attentes théoriques, permettait d'obtenir des solutions simples, d'attaquer des problèmes markoviens ou d'être utilisé effectivement dans le cadre distribué d'un système multi-agents (expérience en cours). Certaines expériences ont également servi à comparer ATNoSFERES à une famille d'architecture d'agent très utilisée en apprentissage par renforcement, les systèmes de classeurs. La dernière expérimentation, ATNoCells, aborde tout à la fois les problématiques qui sont à l'origine des travaux et de la thèse défendue dans ce mémoire : le contrôle, par un système multi-agent en développement, d'un

robot en environnement réel. Chacune des expériences, décrites ci-après, a été conçue pour répondre à une question bien précise :

« – le modèle est-il fonctionnel? (section 4.1)

« – le modèle peut-il générer des solutions compactes/simples à un problème classique de la littérature? (section 4.2)

« – le modèle permet-il d'attaquer des problèmes non-markoviens, et comment se compare-t-il aux systèmes de classeurs[1] ? (section 4.3)

« – le modèle fonctionne-t-il pour une population d'agent en développement, permet-il de recréer une dynamique d'écosystème et de contrôler collectivement un robot[2]? (section 4.4)

4.1 Fonctionnalité du modèle

4.1.1 Introduction

Cette expérience en simulation, très simple, a eu pour but de tester la possibilité effective de construction de comportements adaptés à l'environnement dans un cadre classique d'algorithme évolutionniste : opérations génétiques effectuées par le système, évaluation explicite et individuelle de l'adéquation des agents à leur milieu, etc. Il s'est agi de commencer à « jouer » avec notre modèle et de faire des premiers tests en se focalisant uniquement sur le modèle d'expression génétique, pour une tâche simpliste.

1. Ces travaux sur les problèmes non-markoviens sont le fruit d'une collaboration de Sébastien Picault et moi-même avec Pierre Gérard, doctorant de l'équipe AnimatLab, et Olivier Sigaud, Maître de Conférences de l'équipe AnimatLab, spécialistes des systèmes de classeurs, et notamment leur application à l'apprentissage dans des environnement non-markoviens (Gérard, 2002).

2. Ces expérimentations avec un robot n'auraient pas été possibles sans le concours de Louis Hugues, doctorant de l'équipe Miriad. L'expertise développée lors de son doctorat m'a amené à concevoir ATNoCells, qui peut être vu comme une extension du modèle d'apprentissage par démonstration qui fait l'objet de sa propre thèse (Hugues, 2002).

4.1.2 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 4.1. 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.

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. 4.1 – *Lexèmes d'action et de condition d'un* `Pieton`.

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 bit). 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, d'une taille elle-même aléatoire.

On applique un algorithme évolutionniste 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.

4.1.3 Résultats

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 4.1 montre l'évolution moyenne de la fitness, calculée sur 10 expériences, dans la première situation (mutations uniquement avec un taux r = 1 %). La figure 4.2 présente l'évolution moyenne de la fitness dans le second

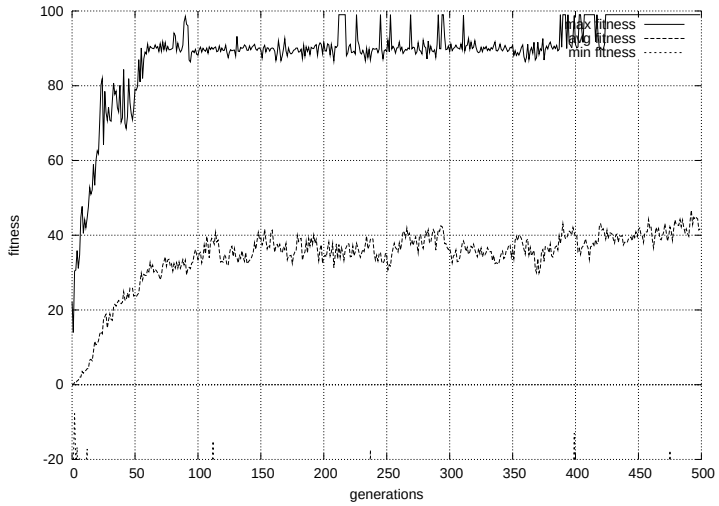


FIG. 4.1 – *Évolution moyenne de la fitness (mutations ponctuelles seulement, r = 1 %). Chromosomes initiaux de 50 à 100 bits.*

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.

4.1.4 Analyses

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

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 le cas dans ATNoSFERES où les instructions sont en partie déclaratives. Nous

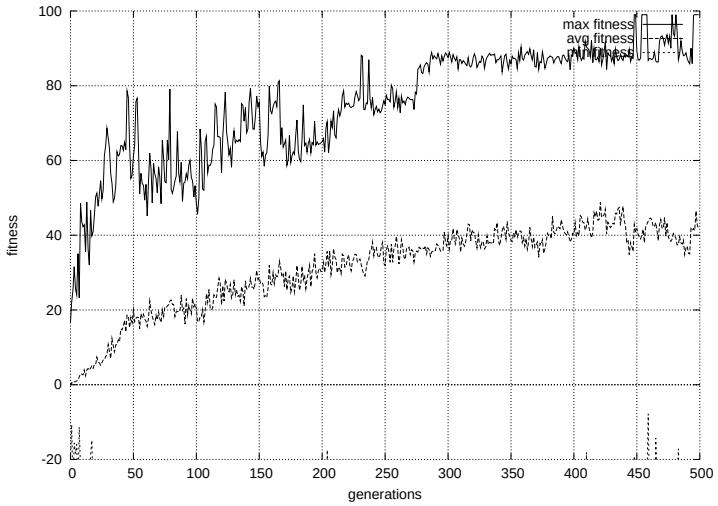


FIG. 4.2 – *Évolution moyenne de la fitness (insertions et délétions aléatoires de codons, r = 1 %, p = 20 %). Chromosomes initiaux de 50 à 100 bit.*

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.

L'ATN décrit par la figure 4.3 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 ».

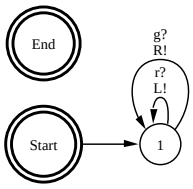


FIG. 4.3 – *L'ATN « optimal » (donnant la fitness maximale avec la structure la plus simple).*

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 chromosomes.

Qu'en est-il exactement ? On peut formuler un début de réponse au vu des figures 4.4 à 4.7, réalisées alternativement avec les stratégies mutations seules ou insertions-délétions, et pour des tailles de chromosomes initialement comprises entre 200 et 300 bit, puis entre 500 et 700 (soit 20 fois la taille d'un chromosome produit manuellement).

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. Non seulement les insertions et délétions ne semblent pas létales pour la population, mais il semble même qu'elles lui sont bénéfiques puisqu'on

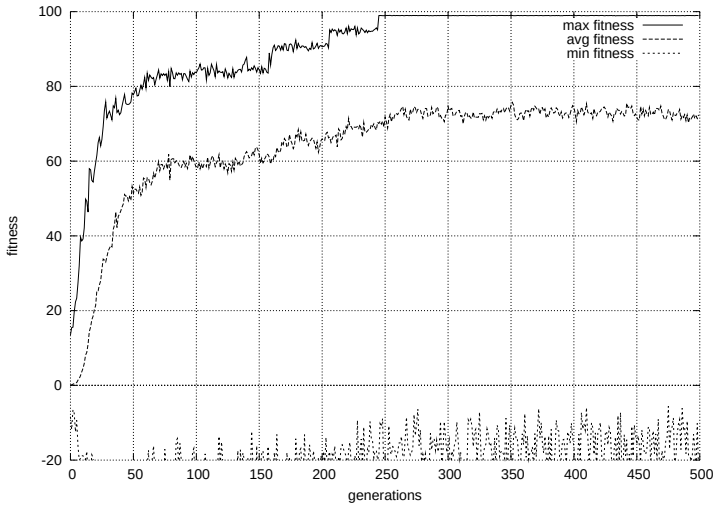


FIG. 4.4 – *Évolution moyenne de la fitness (mutations ponctuelles seulement, r = 1 %). Chromosomes initiaux de 200 à 300 bits (moyenne sur 20 expériences).*

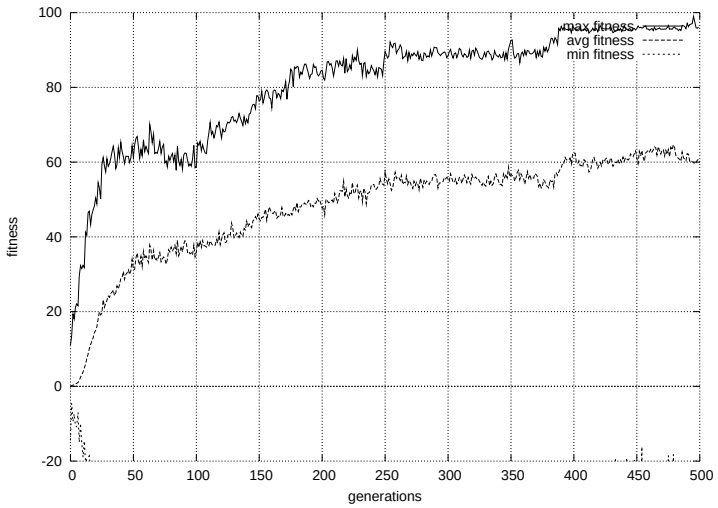


FIG. 4.5 – *Évolution moyenne de la fitness (insertions et délétions aléatoires de codons, r = 1 %, p = 20 %). Chromosomes initiaux de 200 à 300 bits (moyenne sur 20 expériences).*

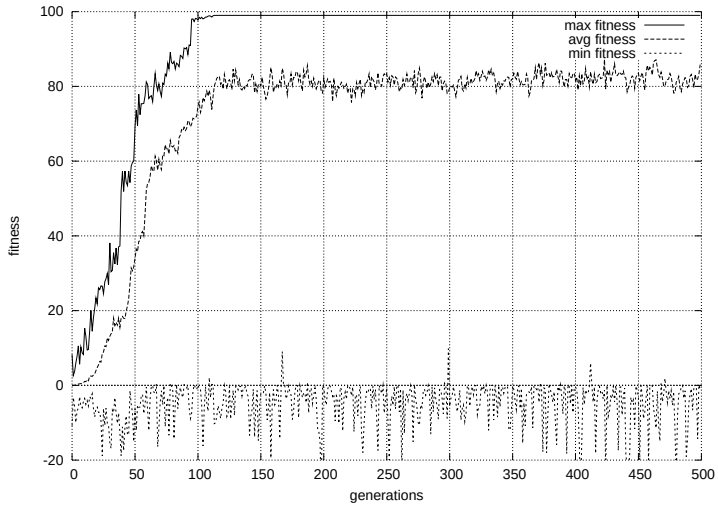


FIG. 4.6 – *Évolution moyenne de la fitness (mutations ponctuelles seulement, r = 1 %). Chromosomes initiaux de 500 à 700 bits (moyenne sur 10 expériences).*

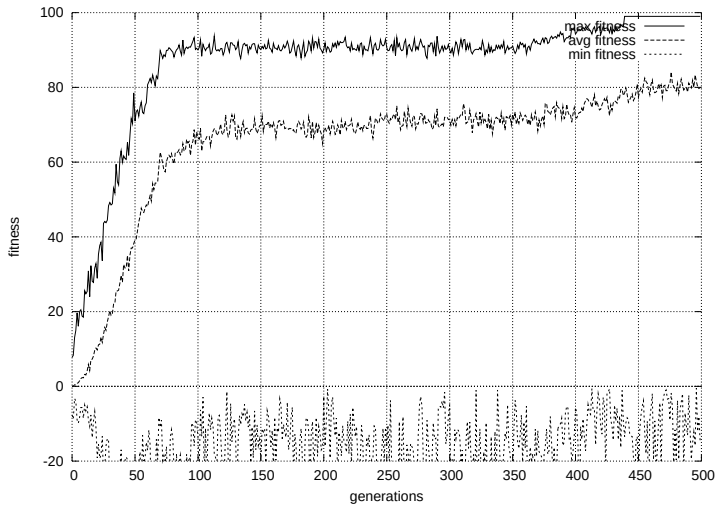


FIG. 4.7 – *Évolution moyenne de la fitness (insertions et délétions aléatoires de codons, r = 1 %, p = 20 %). Chromosomes initiaux de 500 à 700 bits (moyenne sur 10 expériences).*

observe que le nombre de cas de non-convergence diminue dans le cas des insertions-délétions.

Lorsque la taille d'un chromosome augmente, les individus peuvent être vus comme soumis à une double pression sélective : celle issue de leur environnement « fonctionnel » (la fonction de *fitness*), doublée d'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 4.8 : certains individus ont trouvé assez rapidement un comportement adéquat, mais celui-ci ne se maintient pas dans la population.

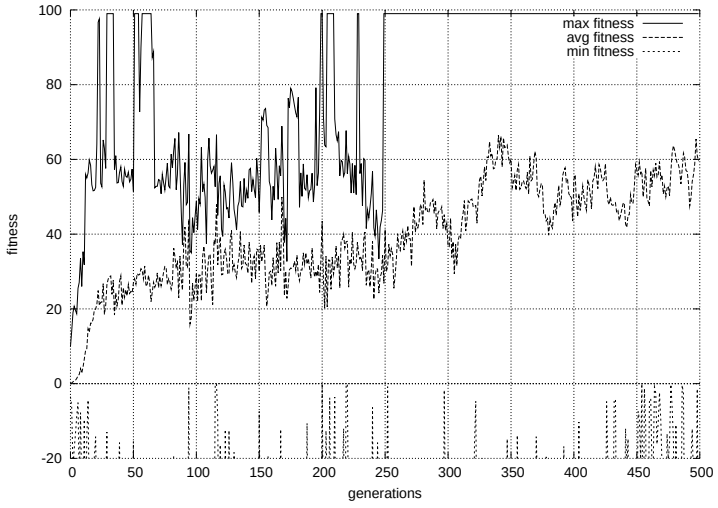


FIG. 4.8 – *Évolution de la fitness (insertions et délétions aléatoires de codons, r = 1 %, p = 20 %). Chromosomes initiaux de 200 à 300 bit.*

Le second vient de l'examen qualitatif des comportements produits. L'analyse des résultats expérimentaux montre que deux stratégies sont mises en œuvre pour produire un comportement adéquat (donnant la *fitness* maxi-

male). La première consiste à construire un ATN simple (proche de l'ATN « optimal » de la figure 4.3), 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 pile qui sont ignorées, la pile étant vide).

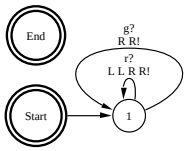


FIG. 4.9 – *Graphe d'un individu exprimant le bon comportement, et dont le début du chromosome contient uniquement des instructions de manipulation de pile ignorée.*

Ainsi, un des agents possède un chromosome de 207 bit, qui une fois décodé produit l'ATN représenté figure 4.9, presque identique à l'ATN optimal (*cf.* figure 4.3). 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.

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 4.10), 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. Ces opéra-

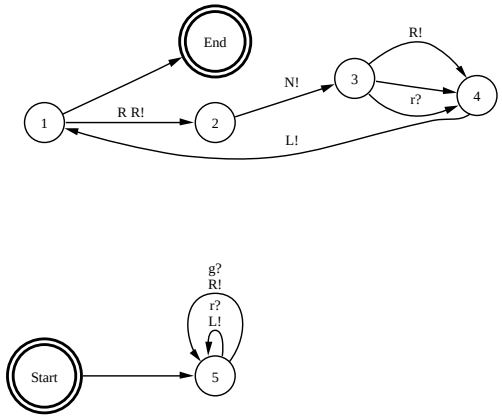


FIG. 4.10 – *Un ATN construit par évolution, exprimant le bon comportement. Cet ATN a été construit dans une expérience avec insertions-délétions (chromosomes initiaux de 200 à 300 bit).*

tions 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 4.11 à 4.13). Il semble, pour les populations à chromosomes longs, qu'on obtienne une stabilisation autour de 650 bits (voir notamment l'expérience de la figure 4.14).

En conclusion de ces expériences, Nous pouvons constater, d'une part, que le modèle est fonctionnel. D'autre part, en nous penchant exclusivement sur le modèle d'expression génétique, il nous a permis, de part ses particularités, d'observer des propriétés intéressantes. Pour obtenir une structure exprimant un comportement correct et contraindre sa complexité, ou bien pour se prémunir contre des manipulations génétiques systématiques, le modèle dispose de plusieurs moyens : parties de chromosome effectivement non codantes ou parties de la structure inutilisées. Ce sont là des conséquences du principe de séparation entre syntaxe et sémantique, qui ici nous permettent d'observer

des effets des pressions sélectives différentes suivant qu'ils s'appliquent au génotype ou au phénotype.

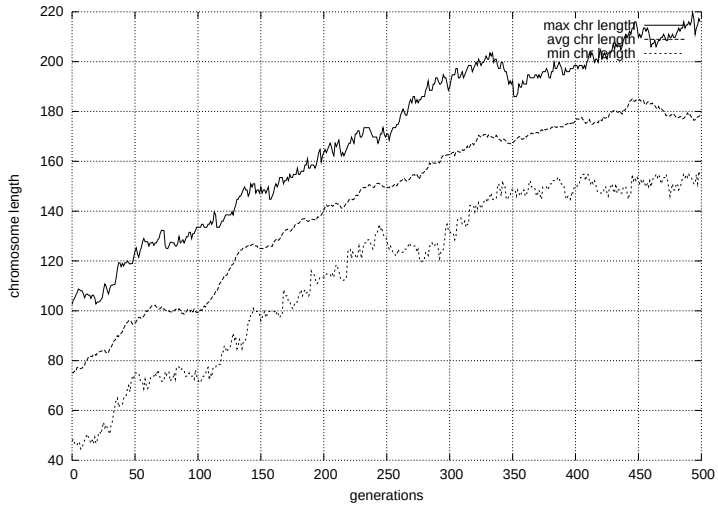


FIG. 4.11 – *Évolution moyenne de la taille des chromosomes (insertions et délétions aléatoires de codons, $r = 1$ %, $p = 20$ %). Chromosomes initiaux de 50 à 100 bits (moyenne sur 10 expériences).*

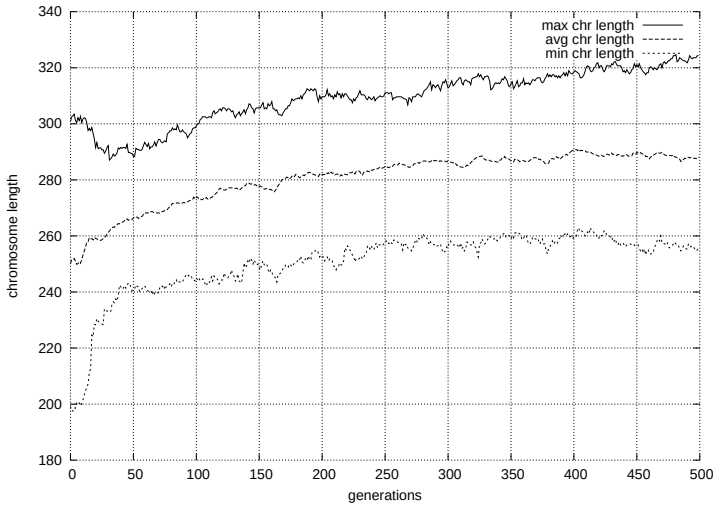


FIG. 4.12 – *Évolution moyenne de la taille des chromosomes (insertions et délétions aléatoires de codons, r = 1 %, p = 20 %). Chromosomes initiaux de 200 à 300 bits (moyenne sur 20 expériences).*

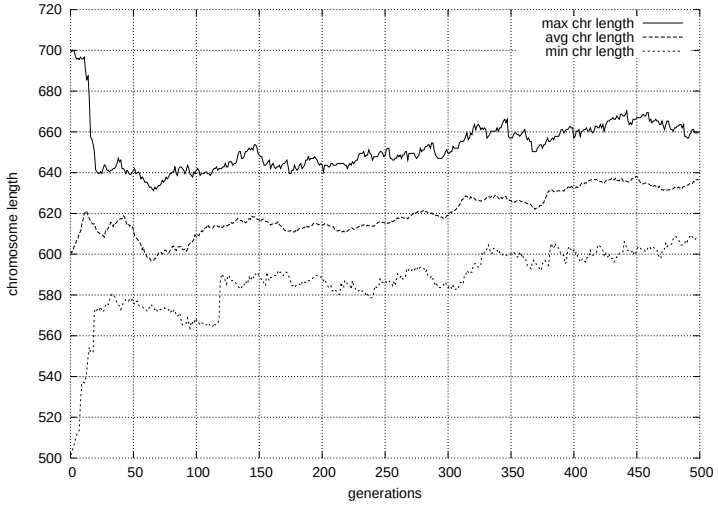


FIG. 4.13 – *Évolution moyenne de la taille des chromosomes (insertions et délétions aléatoires de codons, r = 1 %, p = 20 %). Chromosomes initiaux de 500 à 700 bits (moyenne sur 10 expériences).*

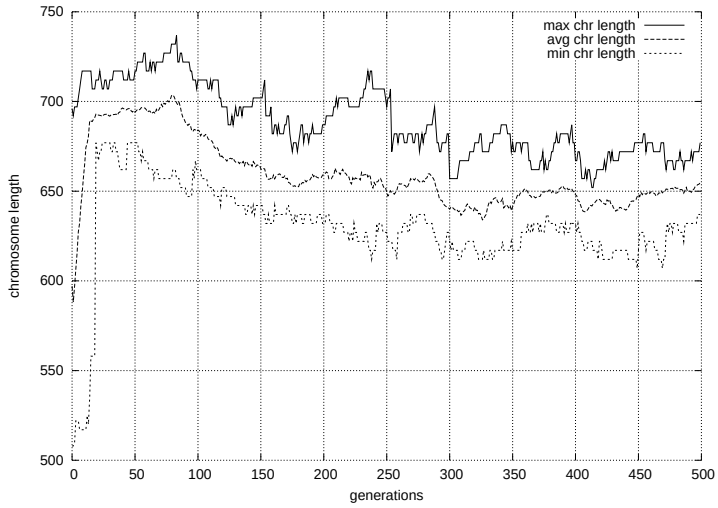


FIG. 4.14 – *Évolution de la taille des chromosomes au cours d'une expérience, montrant un cas de diminution (insertions et délétions aléatoires de codons, r = 1 %, p = 20 %). Chromosomes initiaux de 500 à 700 bits (moyenne sur 10 expériences).*

4.2 Parcimonie du modèle

4.2.1 Introduction

Cette expérience en simulation a eu pour objectif de tester la capacité du modèle à générer des comportements les plus simples possibles au vu de la tâche à accomplir. Ce principe de parcimonie est sous-jacent à nos recherches sur la conception de comportements d'agents (Drogoul, 2000). Il nous est paru intéressant de vérifier si notre modèle vérifiait aussi ce principe.

4.2.2 Protocole expérimental

L'environnement est un labyrinthe à cases. On définit un agent disposant des lexèmes de condition et d'action décrits dans le tableau 4.2. Le comportement souhaité consiste à sortir du labyrinthe en partant de l'entrée, le plus rapidement possible. À chaque pas de temps, l'agent 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 la position de l'agent. Comme l'action arrêt n'existe pas et que l'action par défaut consiste à avancer, dans toutes les expériences, l'agent est initialement placé face à une paroi pour le forcer à prendre une première décision.

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.

Pour toutes les expériences, un temps limité est donné, strictement supérieur au nombre minimal de pas de temps permettant de sortir du labyrinthe. Une évaluation se termine lorsque l'agent a atteint la case de sortie, ou lorsque le temps imparti est écoulé. Les évaluations sont faites chacune sur 3 essais

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

TAB. 4.2 – *Lexèmes d'action et de condition pour la $2^{ème}$ expérimentation.*

dont les notes sont ajoutées, afin de moyenner les comportements exprimés et tenir compte du « bruit » comportemental dû au non-déterminisme des automates.

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 :

$$f = g(p) + t \tag{4.1}$$

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. 4.15 – *Petit labyrinthe. L'entrée est en haut.*

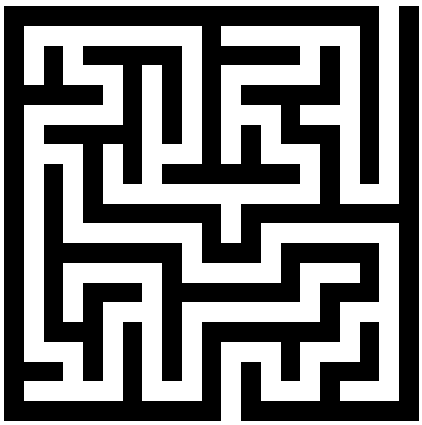


FIG. 4.16 – *Labyrinthe moyen. L'entrée est en haut.*

Plusieurs expériences ont été réalisées :

« – en faisant varier la taille du labyrinthe (pour l'instant 4×4, *cf.* figure 4.15 et 19×19, *cf.* figure 4.16) ;

« – 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`, *cf.* section 4.1) ;

« – en faisant varier les taux de mutation et d'insertion-délétion. Ces deux opérateurs sont systématiquement appliqués ensemble et successivement, les expériences préliminaires nous ayant convaincu de l'utilité des opérations d'insertion-délétion ;

« – en faisant varier la taille des individus initiaux générés aléatoirement (400, 600 et 800 bits).

4.2.3 Résultats

Avec le petit labyrinthe, des solutions optimales au sens du parcours du labyrinthe sont trouvées presque immédiatement quelle que soit la stratégie : l'agent 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 chromosome se réduire à un même minimum de 125 bits 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, à ce moment, 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, nous avons vérifié que les expériences ne convergent pas vers un résultat satisfaisant en retirant les conditions négative (*cf.* tableau 4.2) : l'agent reste coincé dans un minimum local au bout du premier couloir. En effet, grâce aux conditions négatives, il est possible de distinguer des situations ambiguës, ce qui est ici nécessaire (la question des situations ambiguës est plus spécifiquement l'objet des expériences suivantes, décrites section 4.3).

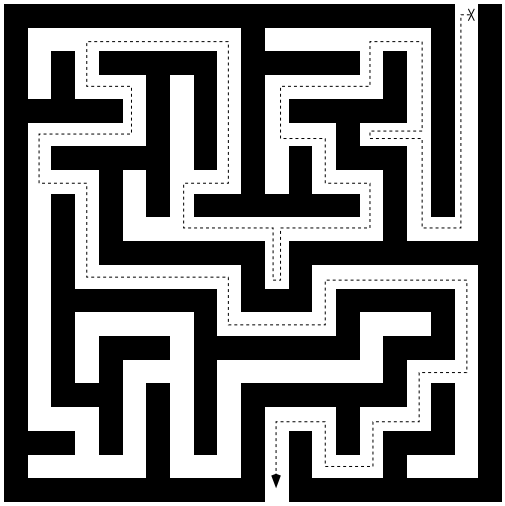


FIG. 4.17 – *Exemple de parcours observé. Le parcours des deux culs-de-sac sur la gauche du parcours rendent cependant cette solution légèrement sous-optimale.*

Dans un second temps, avec toutes les conditions, nous avons obtenu des individus capables de sortir du labyrinthe plus rapidement que le temps imparti. En moyenne, ils n'utilisent que 20 pas de temps en plus des 155 optimaux. Le comportement trouvé ressemble à un parcours « main gauche » du labyrinthe, les agents ne prenant 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 est capable de faire demi-tour. Certaines solutions que nous avons obtenues ne sont cependant pas déterministes, la probabilité existant qu'à

peine sorti, l'agent retourne dans le cul-de-sac. C'est dans ces culs-de-sac que les meilleurs individus perdent les 20 pas de temps en plus du parcours optimal. La figure 4.17 illustre le parcours observé pour l'un des meilleures individus, dont le graphe est représenté figure 4.18.

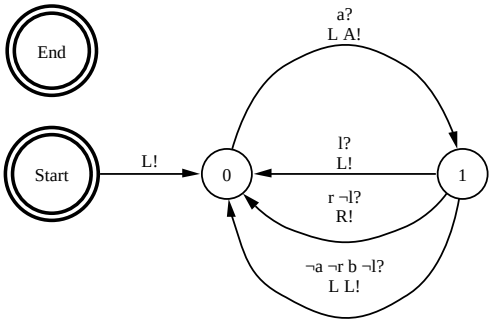


FIG. 4.18 – *Un ATN construit par évolution, donnant un comportement presque optimal. Le parcours du labyrinthe est illustré figure 4.17. Lorsque l'on franchit un arc, c'est la dernière action de la liste qui est exécutée. En l'absence de franchissement d'arc (aucun arc n'étant éligible), c'est la dernière action décidée qui est effectuée à nouveau.*

D'autre part, les graphes obtenus sont très compacts (*cf.* par exemple la figure 4.18). Il n'y a pas plus de deux ou trois nœuds, et `End` 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, autour de 220 bits.

Avec l'exemple typique de la figure 4.18, on comprend aisément la stratégie appliquée. Avant de l'expliquer, remarquons qu'elle est déterministe, car

il n'y a pas d'arcs partants d'un même nœud qui pourraient être simultanément éligibles. Le non-déterminisme éventuel de certains individus lors de l'évolution a cependant contribué, au fil des générations, à trouver la solution, en apportant plus de liberté à leur comportement, leur permettant de faire des essais.

Détail du comportement : l'arc entre Start et 0 est nécessaire car au départ l'agent est orienté vers la gauche, face à la paroi (*cf.* figure 4.16), 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 (*cf.* l'illustration figure 4.17) :

« – 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).

4.2.4 Analyses

Les expériences sur le labyrinthe de taille moyenne sont plus significatives que celles sur le petit labyrinthe. Grâce à elles, nous pouvons démontrer 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 dû au fait que le gain est trop faible au regard du coût génétique qu'il représente (mieux spécialiser les conditions portées par les 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'exerce aussi de façon implicite par la longueur de la chaîne de bits 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 concourt certainement à simplifier le comportement produit par le graphe.

4.3 Problèmes non-markoviens et comparaison avec les systèmes de classeurs

4.3.1 Introduction

Dans les environnements réels, il peut arriver qu'un agent perçoive la même situation en différents endroits, dont certains requièrent des actions distinctes au regard de la tâche à accomplir. Il se pose alors un problème d'alias

perceptuel (*perceptual aliazing*). Dans de telles conditions, l'environnement est dit *non-markovien*, et l'agent ne peut plus s'appuyer exclusivement sur ses perceptions du moment pour prendre la meilleure décision. Pour nous qui voulons appliquer ATNoSFERES à la description de comportements d'agents pour contrôler un robot en environnement réel, il est donc apparu nécessaire de tester le modèle sur un problème de ce type. Nous en avons choisi un qui a déjà été utilisé pour étudier des systèmes de classeurs (type d'architecture d'agent qui a la capacité d'apprendre par renforcement). Comme nous allons le montrer formellement, ATNoSFERES est en effet équivalent à un système de classeurs doté d'états internes.

Les systèmes de classeur (LCS)

Un système de classeur (*Learning Classifier System*, LCS (Holland, 1975b)) est une architecture d'agent constituée d'un ensemble de règles – ou classeurs – décrivant quelle action exécuter au vu des conditions. Les conditions sont des attributs représentant des propriétés observables de l'environnement. Un cas particulier de condition est le symbole de généralisation, représenté par le caractère "#", qui peut être mis en correspondance avec n'importe quelle valeur d'attribut, permettant ainsi de généraliser sur une condition. Comme dans le cadre de l'apprentissage par renforcement (Sutton and Barto, 1998), les buts du système sont définis par des récompenses scalaires fournies par l'environnement. Le système doit apprendre les classeurs, qui définissent le comportement comme montré figure 4.19.

Il y a deux principales familles de systèmes de classeurs, les *Pittsburg* et les *Michigan*. Pour les *Pittsburg*, une population de systèmes de classeurs est évoluée à l'aide d'un algorithme génétique. Les individus sont ici des systèmes de classeurs complets, chacun évalué à l'aide d'une fonction de fitness. Pour

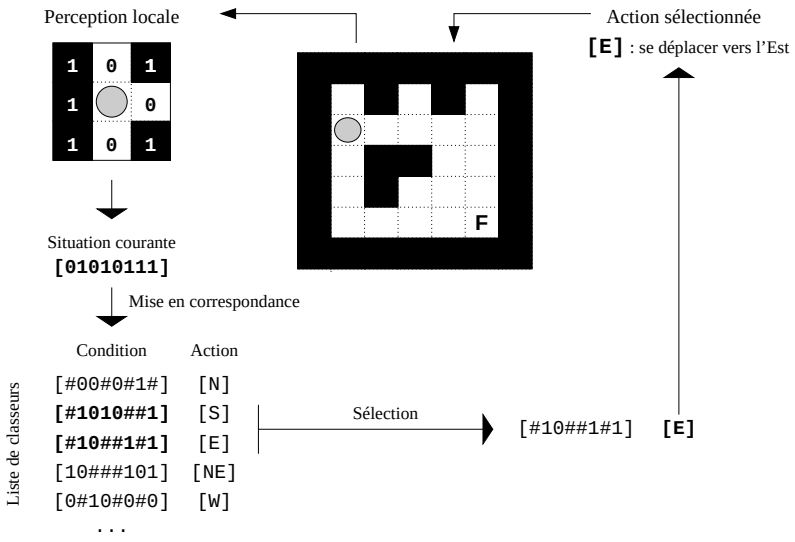


FIG. 4.19 – *Boucle sensori-motrice impliquant un système de classeurs. L'agent perçoit la présence/absence (resp. 1/0) de bloc dans chacune des cases qui l'entourent (dans le sens des aiguilles d'une montre, à partir de la case Nord). Par conséquent à partir de sa position actuelle, l'agent perçoit [01010111]. Le LCS choisit d'abord les classeurs qui correspondent à la situation. Ensuite, parmi ceux-ci il en choisit un et l'action correspondante est exécutée.*

les *Michigan*, par contre, c'est la liste des classeurs d'un système qui est évoluée par un algorithme génétique : les individus sont les classeurs individuels. Dans ce cas, l'algorithme génétique est utilisé pour faire de l'apprentissage en ligne : les classeurs sont améliorés durant la vie de l'agent.

Bien que les systèmes de classeurs soient le plus souvent utilisés pour résoudre des problèmes markoviens – le système le plus employé est alors XCS (Wilson, 1995) – il y a eu plusieurs tentatives d'applications à des problèmes non-markoviens, par exemple (Lanzi, 1998; Tomlinson and Bull, 2000). Dans ce cadre, des états internes ont été ajoutés au couple (*conditions*,*actions*) des classeurs (Lanzi, 1998; Lanzi and Wilson, 2000; Wilson, 1995). Ces états internes fournissent de l'information supplémentaire pour pouvoir choisir la meilleure action quand l'agent se trouve en situation d'alias perceptuel.

XCSM (Lanzi, 1998) est une extension de XCS, auquel des états internes ont été ajoutés – M est l'initiale de *memory*. XCSM gère un registre mémoire interne de plusieurs bits. Un classeur est alors représenté par un quadruplet (*conditions externes*,*conditions internes*,*actions externes*,*actions internes*). Les conditions et actions externes se rapportent aux perceptions et actions sur l'environnement, et les conditions et actions internes se rapportent aux perceptions et actions sur le registre. Pour pouvoir être choisi par le LCS, un classeur doit désormais voir correspondre à la fois ses conditions internes et externes à la perception de l'environnement et de son registre. De façon analogue, l'exécution se fait en exécutant à la fois les actions externes et internes, en considérant pour chaque action interne le cas particulier symbolisé par "#", qui signifie qu'il ne faut pas modifier le bit du registre qui lui est associé. La figure 4.20 résume le fonctionnement d'un système de type XCSM.

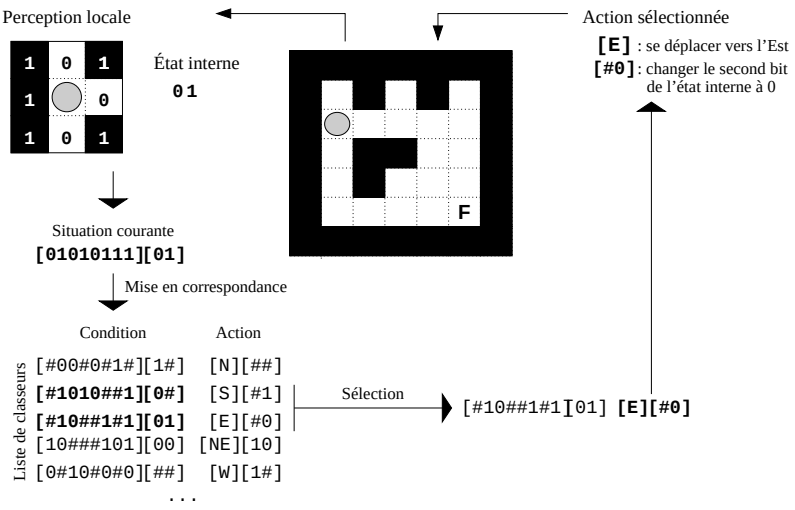


FIG. 4.20 – *Boucle sensori-motrice impliquant un système de classeurs avec registre mémoire. L'agent perçoit la présence/absence (resp. 0/1) de bloc dans chacune des cases qui l'entourent ainsi que l'état du registre. Par conséquent à partir de sa position actuelle, l'agent perçoit [01010111][01]. Le LCS choisit d'abord les classeurs qui correspondent à la situation. Ensuite, parmi ceux-ci, il en choisit un et les actions externe et interne correspondantes sont exécutées.*

Équivalence formelle entre XCSM et ATNoSFERES

Un ATN du type de ceux utilisés comme support de description de comportements dans ATNoSFERES peut être traduit en une liste de classeurs et réciproquement. Cette équivalence repose sur plusieurs identifications entre certaines parties des deux représentations :

« – les nœuds de l'ATN jouent le rôle d'états internes – permettant à ATNoSFERES de faire face aux alias perceptuels. Ils peuvent donc en particulier être numérotés en base 2, comme le registre mémoire de XCSM, ce sur quoi nous pouvons nous appuyer pour faire la traduction d'un formalisme à l'autre ;

« – les arcs de l'ATN portent des informations qui peuvent être traduites dans un formalisme de classeur : l'origine de l'arc est une condition

interne, et son extrémité une action interne ; les conditions portées par l'arc correspondent aux conditions externes du classeur, et les actions portées par l'arc aux actions externes du classeur.

La figure 4.21 résume ces identifications.

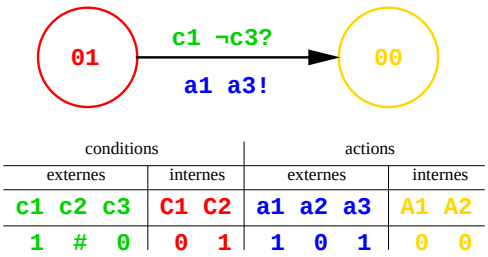


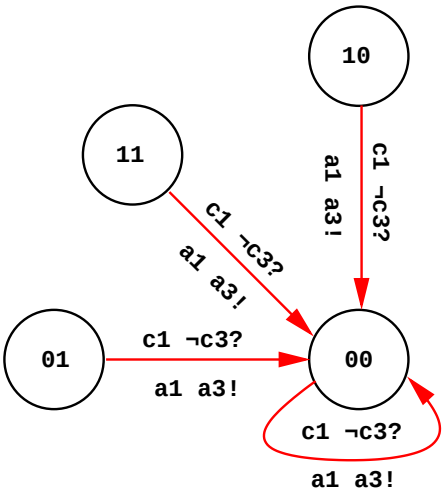
FIG. 4.21 – *Équivalence entre un arc d'ATN et ses extrémités, d'une part, et un classeur, d'autre part.*

La seule différence notable est que, dans le cas général, un arc peut porter une *séquence* d'actions, ce qui n'est pas le cas pour les classeurs que nous considérons. Pour nos besoins, nous restreignons donc le maximum d'actions à une seule par arc – en fait la dernière de la liste prend le pas sur les autres.

Il y a encore deux différences en pratique :

« – la première est qu'un arc peut aussi porter une liste vide d'actions. Dans ce cas nous décidons qu'une action aléatoire sera exécutée lors du parcours de l'arc ;

« – la seconde est qu'avec les lexèmes de construction dont nous disposons actuellement, ATNoSFERES ne peut pas explicitement généraliser sur son état interne, alors que XCSM peut le faire avec un classeur dont les conditions internes sont toutes à "#". Il demeure cependant possible de traduire une représentation d'un formalisme dans l'autre comme le montre la figure 4.22.



condition					actions				
externe			interne		externe			interne	
c1	c2	c3	C1	C2	a1	a2	a3	A1	A2
1	#	0	#	#	1	0	1	0	0

FIG. 4.22 – *Généralisation avec XCSM et ATNoSFERES.*

4.3.2 Protocole expérimental

Le problème sur lequel porte l'expérience est, comme le précédent, un problème de labyrinthe, présentant des situations d'alias perceptuel. Il a été utilisé dans (Lanzi, 1998) pour analyser les capacités de XCSM en environnement non-markovien. L'environnement `Maze10` est présenté sur la figure 4.23. Cette fois-ci, l'agent doit trouver la sortie (signalée par un F sur la figure) le plus rapidement possible au départ de *chaque* autre case ; les mouvements possibles sont les déplacements dans chacune des 8 cases autour de la position actuelle de l'agent (si la case cible est libre) et enfin l'agent peut percevoir la présence ou l'absence d'une case libre parmi les 8 qui l'entourent.

Il y a donc 8 actions et 16 conditions booléennes, décrites dans le tableau 4.3. Pour pouvoir coder ces 24 conditions et actions, plus les 7 lexèmes

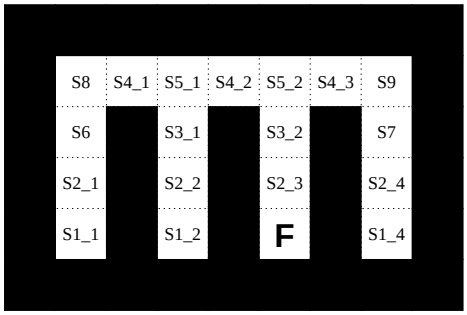


FIG. 4.23 – *L'environnement* `Maze10`. `F` *représente le but à atteindre (*food*) depuis toute case du labyrinthe ; quelques cases ne sont pas ambiguës (S_i), mais parmi les autres (S_{i_j}) la même situation perceptuelle peut requérir soit une action similaire soit une action différente (par exemple aller vers le nord en $S_{2\_\{1,2,4\}}$ mais aller au sud en S_{2_3}).*

Actions		**Conditions**	
`N!`	aller au nord	`N?`/¬`N?`	vrai si la case au nord est libre/occupée
`NE!`	aller au nord-est	`NE?`/¬`NE?`	vrai si la case au nord-est est libre/occupée
`E!`	aller à l'est	`E?`/¬`E?`	vrai si la case à l'est est libre/occupée
`SE!`	aller au sud-est	`SE?`/¬`SE?`	vrai si la case au sud-est est libre/occupée
`S!`	aller au sud	`S?`/¬`S?`	vrai si la case au sud est libre/occupée
`SW!`	aller au sud-ouest	`SW?`/¬`SW?`	vrai si la case au sud-ouest est libre/occupée
`W!`	aller à l'ouest	`W?`/¬`W?`	vrai si la case à l'ouest est libre/occupée
`NW!`	aller au nord-ouest	`NW?`/¬`NW?`	vrai si la case au nord-ouest est libre/occupée

TAB. 4.3 – *Lexèmes d'action et de condition pour l'environnement* `Maze10`.

de pile et les 4 lexèmes de création de nœud/connexion (*cf.* tableau 3.1 page 71), ce qui nous fait 35 lexèmes distincts au total, il faut donc au moins 6 bits par lexème, soit un nombre de codons de $2^6 = 64$. Le code génétique sera donc redondant, certains lexèmes étant représentés plusieurs fois.

Les expériences portent sur une population de 300 individus (initialisés au hasard pour la première génération) évoluée sur 10 000 générations. Les géniteurs de la génération suivante sont les 20% d'individus de la population courante ayant les plus hautes fitness. Lors de la reproduction, chaque bit de la chaîne-individu est muté avec une probabilité de 1%, et chaque

codon a 0,5% de chances d'être soit effacé (délétion) soit précédé par un nouveau codon généré aléatoirement (insertion). La politique de sélection des géniteurs est calculée en fonction du rang : l'individu ayant la n-ième fitness (dans l'ordre décroissant) a une probabilité d'être tiré au sort $c = \sqrt[60]{2}$ fois plus grande que l'individu de rang $n+1$. Le coefficient c a été déterminé de façon à ce que l'individu de plus haut rang (le 1^{er} géniteur) ait une probabilité d'être choisi égale au double de celle de l'individu de plus bas rang (le $60^{\text{ème}}$ géniteur).

Chaque agent dispose de 20 pas de temps au départ de chaque case et chaque case de départ est testée 4 fois pour débruiter le comportement qui peut être non déterministe du fait de la sélection aléatoire parmi les arcs éligibles à partir d'un nœud. À chaque pas de temps une action est exécutée : si aucun arc n'est éligible ou si l'arc élu ne porte pas d'action, une action est tirée au hasard. Si l'action n'est pas exécutable (notamment aller dans une case occupée), l'action est ignorée et l'agent perd un pas de temps.

La fitness totale est calculée en additionnant toutes les fitness $F_{(i,j)}$ obtenues par case de départ i, et pour chacun des 4 essais j, calculées selon la formule 4.2.

$$F_{(i,j)} = D_{(i,j)} - K_{(i,j)} + B + 2 * R_{(i,j)} \tag{4.2}$$

où : $F_{(i,j)}$ est la fitness de l'agent pour la case de départ i, à l'essai j ; $D_{(i,j)}$ est le nombre de case découvertes ; $K_{(i,j)}$ est le nombre de pas de temps passés sur des cases déjà connues ; B est un bonus quand la sortie est atteinte (30 points) ; $R_{(i,j)}$ est le nombre de pas de temps restant si la sortie a été trouvée dans le temps imparti ($R_{(i,j)} < 19$).

La fitness a été conçue de façon à inciter l'agent à explorer (termes $D_{(i,j)}$ et $K_{(i,j)}$) et à atteindre au plus vite la sortie (grâce au coefficient de $R_{(i,j)}$,

les pas de temps restants rapportent plus que la découverte de nouvelles cases). La fitness d'un agent pour lequel la contrainte d'observation partielle serait levée et qui prendrait la meilleure décision à chaque pas de temps (*cf.* figure 4.24) serait alors de 4 712. Comme ce n'est pas le cas, c'est une borne de la meilleure fitness qu'une agent puisse obtenir dans le conditions de l'expérience.

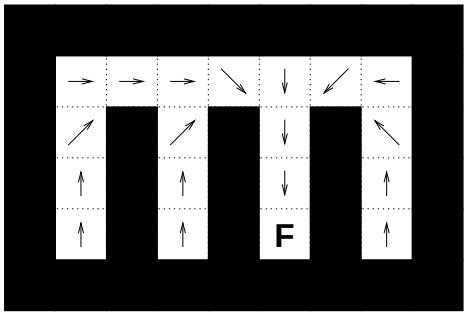


FIG. 4.24 – *Décisions optimales dans l'environnement* `Maze10`*. La fitness (théorique) associée serait alors de 4 712 suivant la formule 4.2 page précédente.*

Nous avons répété les expériences pour des tailles initiales de chromosomes (populations homogènes) allant de 10 à 130 codons, tous les 10 codons – donc des tailles allant de 60 à 780 bits par incrément de 60. Ces tailles sont sujettes à variation comme dans les expériences précédentes, du fait de l'opérateur génétique d'insertion/délétion.

Lors de nos expériences (Landau et al., 2002b; Landau et al., 2002c), nous avons été amenés à émettre des hypothèses de recherche sur la nécessité de pouvoir généraliser sur les états internes pour résoudre des problèmes non-markoviens. Pour le cas particulier de Maze10, la réponse s'est avérée être négative, comme les résultats des expériences décrits ci-après le montrent. Nous avons pour aborder cette question, introduit deux nouveaux lexèmes,

lexème	**action résultante**
lexèmes de structure	crée des constituants de la structure
lexèmes d'ATN	crée de nouveau nœuds ou connexions entre nœuds
connectSelf	crée un arc bouclé sur le premier nœud rencontré dans la pile ; l'étiquette avec l'ensemble des conditions et la liste des actions présents jusqu'au nœud ; retire de la pile les actions et conditions présentes jusqu'au nœud
connectDefaultSelf	crée un arc bouclé sur chacun des nœuds dans la pile ; étiquette chaque arc avec l'ensemble des conditions et la liste des actions présents jusqu'au premier nœud ; retire de la pile les actions et conditions présentes jusqu'au premier nœud

TAB. 4.4 – *Lexèmes d'ATN supplémentaires (les autres lexèmes sont décrits dans dans le tableau 3.1 page 71).*

`connectDefaultSelf` et `connectSelf` (*cf.* tableau 4.4 page 109) dont voici les descriptions :

« – le lexème `connectDefaultSelf` permet de créer un même arc bouclé sur tous les nœuds alors présents dans la pile, en une seule instruction. Comme la pile peut contenir entre aucun et tous les nœuds du graphe final au moment où `connectDefaultSelf` est interprété, l'arc bouclé ne sera pas nécessairement présent sur tous les nœuds. Ce lexème a été introduit car il permet d'opérer explicitement une généralisation partielle sur l'état interne : pour une même condition, depuis un certain nombre d'états, l'action associée sera la même, sans changement d'état ;

« – le lexème `connectSelf` permet, lui, de créer un arc bouclé sur un nœud en une seule instruction, alors que sans il aurait été nécessaire d'en avoir en moyenne entre une et deux. En effet, pour faire de même sans cette instruction supplémentaire, il faudrait deux copies successives du même nœud dans la pile (donc un lexème `dupNode`) et il faudrait soit les connecter explicitement par l'action d'un lexème `connect`, ce qui

fait deux instructions, soit implicitement à la fin de la construction du graphe, s'il demeurait des conditions ou actions entre les deux copies présentes dans la pile, ce qui fait une instruction (mais ce second cas de figure paraît moins probable). Ce lexème, qui ne permet pas de généralisation explicite, a été introduit suite aux expériences avec le lexème `connectDefaultSelf` et aux observations faites de son utilisation.

Nous avons voulu tester l'apport de chacun de ces lexèmes séparément. Pour ce faire, trois codes génétiques ont été employés dans les expériences. Ces trois codes génétiques ne différaient que d'un codon. Le premier code associait à ce codon le lexème `noop`, le deuxième `connectSelf` et le troisième `connectDefaultSelf`. Le premier code sert de code génétique témoin ; les conditions expérimentales pour les 3 codes ne différent que du seul lexème associé à ce codon.

4.3.3 Résultats

Nous détaillons les comportements des meilleurs individus trouvés pour chacun des 3 codes génétiques ; comme ils sont semblables, un résumé suit avec une illustration représentant la stratégie adoptée. Puis, nous montrons dans un même graphique les fitness moyennes et la distributions des fitness pour chaque code génétique et chaque taille initiale des chromosomes.

Avec le lexème `noop`

Le graphe du meilleur individu obtenu avec le lexème `noop` est présenté figure 4.25. Le comportement est déterministe et l'agent sort systématiquement du labyrinthe. Le graphe comporte 2 nœuds internes (2 états), et La

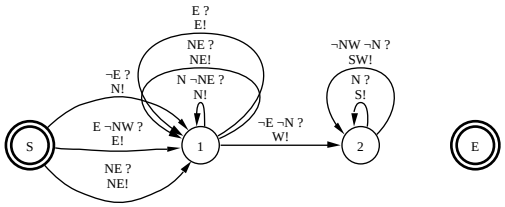


FIG. 4.25 – *Graphe du meilleur individu obtenu avec le lexème* `noop`.

fitness de cet individu est de 4 466. La sortie du labyrinthe se déroule comme suit :

« – depuis Start vers l'état 1,

  « – si l'agent est dans une colonne alors il se déplace vers le nord ;

  « – s'il est sur la dernière case d'une colonne (par exemple S_6 pour la première colonne, *cf.* figure 4.23 page 106) alors il se déplace en biais (vers le nord-est) dans la rangée (par exemple en S_{4_1} depuis S_6) ;

  « – s'il est dans la rangée alors il se déplace vers l'est ;

  « – s'il est dans le coin droit S_9, il perd un pas de temps cas il ne peut se déplacer vers le nord ;

« – dans l'état 1,

  « – si l'agent est dans une colonne alors il la parcourt vers le nord jusqu'à sa dernière case ;

  « – s'il est sur la dernière case d'une colonne alors il se déplace en biais (vers le nord-est) dans la rangée ;

  « – s'il est dans la rangée alors il se déplace vers l'est jusqu'au coin droit S_9 ;

« – s'il est dans le coin droit S_9, reconnu de façon unique par les conditions « ¬E ¬N ? », alors il se déplace à l'ouest en S_{4_3} et passe dans l'état 2 ;

« – dans l'état 2

« – si l'agent est en S_{4_3} (reconnu par la condition « ¬NW ¬N ? ») alors il se déplace en biais vers sud-ouest en S_{3_2} ;

« – si l'agent est dans la colonne contenant la sortie, alors il se déplace vers le sud pour l'atteindre.

Avec le lexème `connectDefaultSelf`

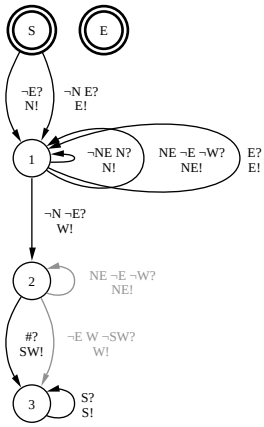


FIG. 4.26 – *Graphe du meilleur individu obtenu avec le lexème `connectDefaultSelf`. Deux arcs portant des conditions contradictoires (et donc jamais éligibles) n'ont pas été représentés. Les arcs en gris ne sont jamais éligibles (cf. la description du comportement).*

Le graphe du meilleur individu obtenu avec le lexème `connectDefaultSelf` est présenté figure 4.26. Comme avec le lexème `noop`, le comportement est déterministe et l'agent sort systématiquement du labyrinthe. Le graphe comporte 3 nœuds internes (3 états), et la fitness de cet individu est de 4 452. Le

comportement est assez semblable à celui qui a été détaillé précédemment, et se déroule comme suit :

« – depuis Start vers l'état 1,

  « – si l'agent est dans une colonne alors il se déplace d'une case vers le nord ;

  « – si l'agent est dans la rangée il se déplace d'une case vers l'est ;

  « – si l'agent est en S_9 il perd un pas de temps (il ne peut pas se déplacer vers le nord) ;

« – dans l'état 1,

  « – si l'agent est dans une colonne alors il la parcourt vers le nord jusqu'à sa dernière case ;

  « – s'il est sur la dernière case d'une colonne alors il se déplace en biais (vers le nord-est) dans la rangée ;

  « – s'il est dans la rangée alors il se déplace vers l'est jusqu'au coin droit S_9 ;

  « – s'il est dans le coin droit S_9, reconnu de façon unique par les conditions « ¬N ¬E ? », alors il se déplace à l'ouest en S_{4_3} et passe dans l'état 2 ;

« – dans l'état 2,

  « – l'agent se déplace en biais vers le sud-ouest, vers S_{3_2}, et passe dans l'état 3 ;

  « – les arcs représentés en gris sur la figure 4.26 sont toujours inéligibles donc inutiles puisque arrivé à l'état 2, l'agent est nécessairement en S_{4_3} ;

« – dans l'état 3,

  « – l'agent se déplace dans la colonne vers la sortie, au sud.

Avec le lexème `connectSelf`

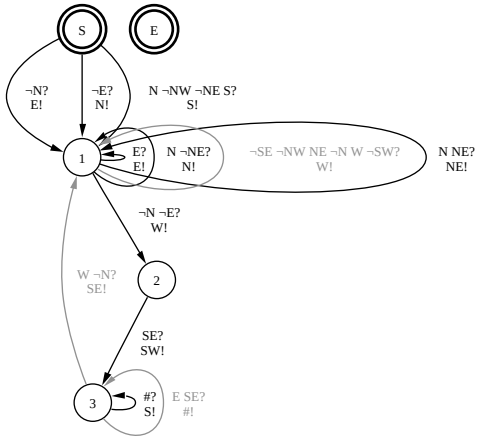


FIG. 4.27 – *Graphe du meilleur individu obtenu avec le lexème* `connectSelf`*. Trois arcs portant des conditions contradictoires (et donc jamais éligibles) n'ont pas été représentés. Les arcs en gris ne sont jamais éligibles (cf. le descriptif du comportement)*

Le graphe du meilleur individu obtenu avec le lexème `connectSelf` est présenté figure 4.27. Cette fois-ci le comportement n'est pas tout-à-fait déterministe, mais l'agent sort tout de même systématiquement du labyrinthe. Le graphe comporte 3 nœuds internes (3 états), et la fitness de cet individu est de 4 496 en moyenne. Le comportement à nouveau est assez semblable à ceux qui ont été détaillés précédemment, voici son déroulement :

« – depuis Start vers l'état 1,

  « – si l'agent est dans l'une des cases $S_{2\_\{1,2,3,4\}}$ (*cf.* figure 4.23 page 106), alors il se déplace soit vers le nord, soit vers le sud, avec la même probabilité ;

  « – si l'agent est dans une colonne, à l'exclusion de l'une des cases $S_{2\_\{1,2,3,4\}}$ sus-mentionnées, alors il se déplace vers le nord ;

  « – si l'agent est dans la rangée il se déplace d'une case vers l'est ;

« – si l'agent est en S_9 il perd un pas de temps (il ne peut pas se déplacer vers le nord) ;

« – dans l'état 1,

« – si l'agent est dans une colonne alors il la parcourt vers le nord jusqu'à sa dernière case ;

« – s'il est sur la dernière case d'une colonne alors il se déplace en biais (vers le nord-est) dans la rangée ;

« – s'il est dans la rangée alors il se déplace vers l'est jusqu'au coin droit S_9 ;

« – s'il est dans le coin droit S_9, reconnu de façon unique par les conditions « ¬N ¬E ? », alors il se déplace à l'ouest en S_{4_3} et passe dans l'état 2 ;

« – l'arc bouclé sur l'état 2 représenté en gris sur la figure 4.27 ne reconnaît aucune situation existante dans le labyrinthe, et n'est donc jamais éligible ;

« – dans l'état 2,

« – l'agent se déplace en biais vers le sud-ouest, vers S_{3_2}, et passe dans l'état 3 ;

« – dans l'état 3,

« – l'agent se déplace dans la colonne vers la sortie, au sud.

« – les arcs ayant pour origine le nœud 3 et représentés en gris sur la figure 4.27 sont toujours inéligibles donc inutiles puisque arrivé à l'état 3, l'agent est nécessairement dans une colonne ;

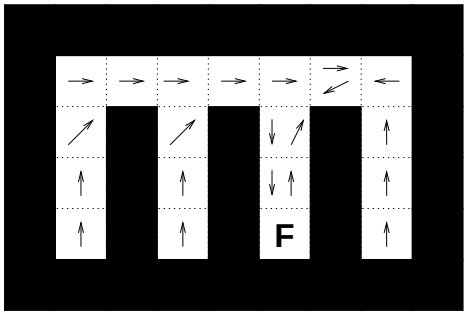


FIG. 4.28 – *Stratégie adoptée par les meilleurs individus pour les 3 codes génétiques testés dans l'environnement* `Maze10`*. Les cases portant deux flèches sont parcourues vers l'est (resp. le nord ou le nord-est) uniquement lorsque l'agent vient de l'ouest (resp. du sud), et en biais vers le sud-ouest (resp. droit vers le sud) sinon.*

Stratégie adoptée

Pour résumer, les comportements des meilleurs individus pour les 3 codes génétiques sont des petites variations du comportement représenté figure 4.28 (à comparer au comportement optimal et hors d'atteinte présenté figure 4.24 page 108). La moyenne sur toutes les cases de départ du nombre de pas de temps pour atteindre la sortie est pour cette stratégie de 7,77 ; la même moyenne pour le cas optimal idéal est de 4,55.

Moyennes et distributions des fitness

La figure 4.29 présente les moyennes des fitness (sur 10 expériences) pour chacun des 3 code génétique et pour chaque taille de chromosome initiale. En moyenne, le code génétique incluant le lexème `connectSelf` permet souvent d'obtenir de meilleurs résultats que les autres. Le code génétique incluant le lexème `connectDefaultSelf` donne des résultats intermédiaires.

Les figures 4.30 montrant les distributions des fitness sont plus parlantes : le code génétique incluant `connectSelf` donne beaucoup plus souvent que les

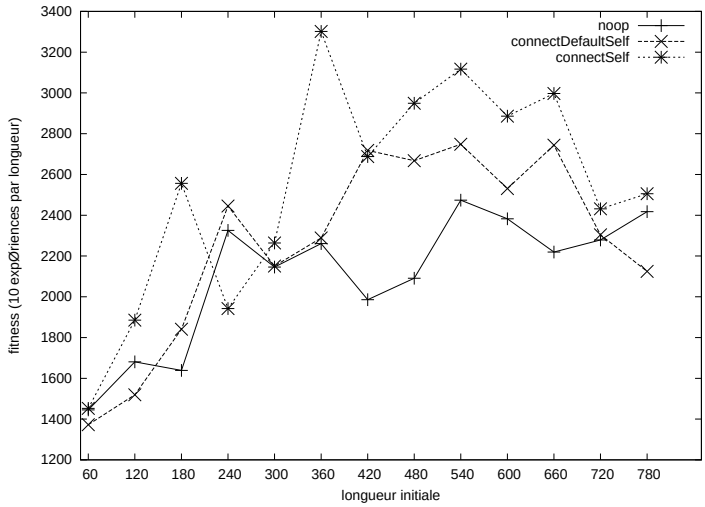
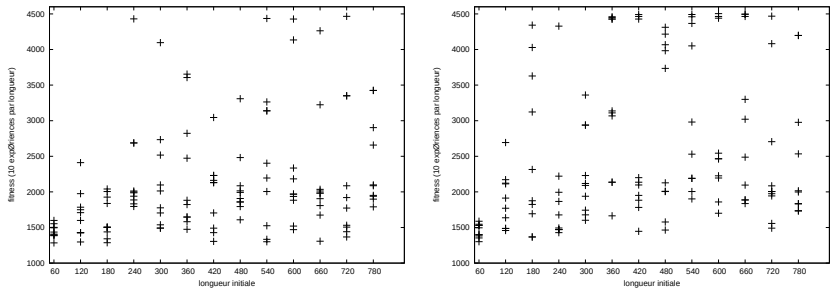


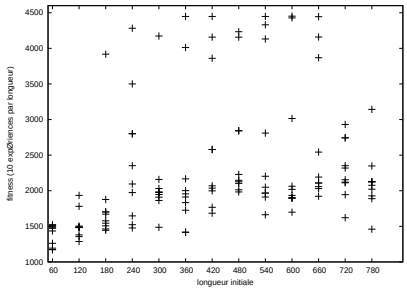
FIG. 4.29 – *Fitness moyennes dans l'environnement* `Maze10` *en fonction des tailles initiales des chromosomes et des lexèmes utilisés (*`noop`*,* `connectSelf`*,* `connectDefaultSelf`*)*

autres de très bon résultats (fitness supérieure à 4 000), alors que l'écart-type est plus petit pour les deux autres codes génétiques, leurs distributions étant concentrées autour de leurs fitness moyennes respectives.



(a) avec le lexème `noop`

(b) avec le lexème `connectSelf`



(c) avec le lexème `connectDefaultSelf`

FIG. 4.30 – *Distribution des fitness dans l'environnement* `Maze10` *en fonction des tailles initiales des chromosomes pour chacun des 3 lexèmes* `noop`, `connectSelf` *et* `connectDefaultSelf`

4.3.4 Analyses

ATNoSFERES permet de résoudre le problème non-markovien posé par l'environnement Maze10, mais pas de façon optimale. La difficulté de la tâche réside dans les multiples cases aux perceptions identiques, mais requérant des actions distinctes ou, dans la succession des actions, l'enregistrement d'une information qui servira plus tard à lever une ambiguïté.

Sur la stratégie adoptée

La stratégie adoptée est assez bonne puisque l'agent sort toujours. Elle est aussi très simple mais sous-optimale – les fitness des meilleurs individus sont proches et en-deçà de 4 500, soit à 4,5% de l'optimum idéal. Les agents se servent exclusivement du coin S_9 pour lever les ambiguïtés sur les cases présentant un alias perceptuel, quelle que soit la case de départ. La stratégie pourrait par exemple être améliorée si pour les cases de la colonne de gauche ils se servaient aussi du coin opposé S_8 qui ne présente lui non plus aucune ambiguïté. De même, lors de la remontée de la deuxième colonne, la situation est ambiguë, puisque l'on confond S_{5_1} avec S_{5_2} (*cf.* la figure 4.23 page 106). Cependant, venant de la deuxième colonne, l'ambiguïté, une fois arrivé en S_{5_2} est levée, puisqu'on ne peut pas rencontrer la même case deux fois. Cette sous-optimalité est peut-être en partie imputable au choix de la fonction de fitness (*cf.* page 107). Celle-ci, bien qu'améliorant la vitesse de résolution du problème, laisse peut-être une part trop grande à l'exploration. D'autre part, mieux résoudre le problème impliquerait de créer plusieurs nœuds et arcs supplémentaires (par exemple pour appliquer les amélioration proposées ci-dessus). Le changement structurel induit pourrait s'avérer trop coûteux pour le gain espéré, ou alors l'impossibilité de le faire par étapes constituerait une barrière trop difficile à franchir.

Différences entre les trois codes génétiques

Il faut remarquer que tous les graphes comportent peu d'états (2 ou 3), et ne présentent pas de composantes connexes inatteignables, comme cela avait été le cas dans des expériences précédentes. Les conditions expérimentales étaient alors différentes, puisque tous les individus étaient mutés – même les parents – à chaque génération.

Une particularité de la solution obtenue avec le lexème `connectSelf` est son petit indéterminisme si la case de départ est perçue comme celle qui est juste au-dessus de la sortie. Dans ce cas, il y a une chance sur deux pour que l'agent se déplace vers le nord, et autant vers le sud. Ce comportement est opportuniste. Si l'agent est effectivement juste au-dessus de la sortie et qu'il se déplace vers le sud, la fitness sera alors de 69 points. S'il se déplace vers le nord, il devra effectuer le comportement consistant à aller reconnaître le coin S_9 et revoir toutes les cases qui l'en séparent avant de pouvoir sortir ; la fitness est alors de 54 points. L'espérance du gain est donc positive, puisque dans les 3 autres cas où il n'est pas au-dessus de la sortie, la perte n'est que de 4 points ($+1$ point de découverte, -1 point de case revue, -2 points de temps qui comptent double).

Les meilleures solutions présentent beaucoup d'arcs bouclés. Il apparaît donc naturel que le code contenant `connectSelf`, qui facilite la création d'arcs bouclés, donne plus souvent et en moyenne de meilleurs résultats. Pour une raison analogue le code contenant `connectDefaultSelf` donne des résultats sensiblement meilleurs que ceux obtenus avec les expériences témoin : l'analyse des individus ayant obtenu de bonnes fitness (supérieures à 4 000) avec le code contenant `connectDefaultSelf` montre que ce lexème a été soit rejeté, car la pile ne contenait pas de nœud au moment où il était interprété, soit utilisé pour ne créer qu'un seul arc bouclé, car la pile ne conte-

nait qu'un seul nœud à ce moment-là. La capacité de généralisation partielle sur l'état interne n'a pas du tout été exploitée ici. Il ne s'agit pas d'un dysfonctionnement du lexème. Dans d'autres conditions expérimentales où ce lexème apportait un gain adaptatif crucial, sa capacité de généralisation a été très utilisée (Landau et al., 2002c). Nous en concluons que la généralisation sur l'état interne ne constitue pas un gain adaptatif significatif pour l'environnement Maze10 .

Comparaison avec XCSM

Sur ce problème, nous avons comparé ATNoSFERES aux meilleurs systèmes de classeurs disposant d'un registre, donc d'états internes. ATNoSFERES a obtenu de meilleurs résultats que XCSM, puisque le nombre de pas de temps moyen pour atteindre la sortie obtenu est inférieur (7,77 contre un peu plus de 15 pour XCSM), et ce sans heuristiques particulières sur la gestion des états internes tant pour l'un que pour l'autre, nonobstant la nature différente des états internes dans les deux formalismes. LANZI a étendu XCSM (Lanzi, 1998) en ajoutant une nouvelle technique d'exploration hiérarchique des classeurs, et en modifiant la politique de mise-à-jour des actions internes. Le système modifié par l'ajout de ces heuristiques gérant explicitement l'état interne a été baptisé XCSMH – H est l'initiale de *hierarchical*. Il permet d'obtenir de bien meilleurs résultats que XCSM pour l'environnement Maze10, avec un nombre de pas de temps moyen pour atteindre la sortie de 6,12. Ce score est aussi meilleur que celui d'ATNoSFERES, qui était de 7,77. La différence n'est cependant pas si grande, en prenant en considération que le temps moyen en levant la contrainte de perception locale est de 4,55 (l'optimum avec les contraintes doit donc se trouver entre 4,55 et 6,12) et surtout que la bonne résolution du problème *repose* sur la finesse de l'exploitation

des états internes : ATNoSFERES, qui ne dispose pas d'heuristique particulière pour la gestion des états internes est donc fortement désavantagé.

Un avantage important d'ATNoSFERES sur XCSM et XCSMH est que le graphe résultant de l'évolution est beaucoup plus facile à comprendre. Cette qualité n'est pas juste le fait d'une représentation graphique. XCSM et XCSMH produisent une liste de classeurs de taille constante, dans lesquels le nombre des conditions externes et la taille du registre doivent être fixés à l'avance. Par conséquent, il y a généralement plus de classeurs et plus d'états internes que nécessaire. Par contraste, ATNoSFERES construit un graphe dont le nombre de nœuds, d'arcs et d'étiquettes sur les arcs ne sont pas fixés à l'avance. La possibilité reste ouverte de construire un contrôleur minimal pour résoudre le problème ; or, dans toutes nos expérience nous avons observé que les solutions proposées par ATNoSFERES, bien que sous-optimales, étaient toujours économes en nombre d'états internes utilisés.

Un autre différence importante est que, avec XCSM, la séquence des états internes de l'agent durant un cycle de fonctionnement n'est pas définie explicitement et doit être reconstituée à la main après un suivi attentif. Au contraire, cette séquence est parfaitement claire quand on lit un ATN. De plus cette séquence est très stable dans ATNoSFERES alors que ce n'est pas le cas avec XCSM : les bit du registre oscillent tout le temps et peuvent être difficilement associés à un contexte ou une situation donnée.

Pour ce qui concerne la généralisation sur les conditions externes, il n'y a pas dans ATNoSFERES de pression sélective explicite vers la généralité, alors que la production de classeurs généraux est inhérente aux systèmes de classeurs. Nous n'obtenons par conséquent pas de règles générales et les conditions peuvent être inutilement trop spécifiques (par exemple pour la meilleure solution contenant le lexème `connectSelf`, représenté figure 4.27

page 114, l'arc bouclé portant la condition « N NE? » pourrait se contenter de « NE? » pour identifier la case du haut d'une colonne).

4.4 ATNoCells : contrôle d'un robot par un SMA en développement

Les expériences précédentes nous ont permis de vérifier, pour un agent en simulation, l'adéquation d'ATNoSFERES pour générer une architecture d'agent évolutive. Dans cette expérience nous allons nous placer dans le cadre d'un système multi-agent en développement et dans celui de la robotique en environnement réel. Nous abordons ainsi toutes les problématiques qui nous ont *in fine* amenés à nous intéresser à l'évolution de structures et à concevoir l'Éthogénétique et ATNoSFERES : le contrôle d'un robot mobile en environnement réel par un système multi-agent adaptatif, et la conception automatique de ce système par un processus inspiré de la sélection darwinienne qui soit moins contraignant que les algorithmes évolutionnistes classiques.

4.4.1 Introduction

Micro Population for Learning

Cette expérience se présente comme une extension aux travaux de thèse de Louis Hugues (Hugues, 2002). Pour faire face aux difficultés de conception de comportements de robots mobiles en environnement réel, il a en effet adopté une approche ascendante et interactive basée sur l'apprentissage par démonstration. Cette approche devait présenter des propriétés qui la rendent utilisable par n'importe qui, n'importe où. En particulier, le robot ne devait avoir aucune connaissance a priori sur l'environnement ou la tâche à exécuter,

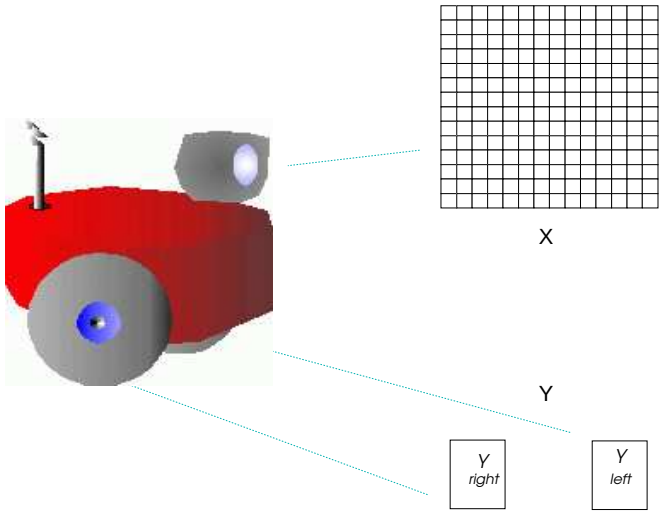


FIG. 4.31 – *Le robot élémentaire considéré dans le modèle.*

et le démonstrateur ne devait posséder aucune connaissance en robotique pour enseigner une tâche. L'apprentissage doit se faire uniquement à partir des perceptions du robot, en environnement réel, et n'est tributaire d'aucune modélisation de cet environnement. L'apprentissage des percepts nécessaire se fait par démonstration, et ne nécessite pas de connaissance a priori sur la tâche visée.

Louis HUGUES a proposé le modèle d'apprentissage MPL (Micro Population for Learning), dans le but de produire des comportements synthétiques à partir des démonstrations réalisées par un tuteur humain. Dans un souci de généricité, le « robot élémentaire » considéré pour le modèle est doté d'un senseur visuel (caméra vidéo 2D) et de deux effecteurs, deux roues motrices contrôlables de façon indépendante (*cf.* figure 4.31).

$$c_i \equiv (x_i, y_i, w_i) \tag{4.3}$$

$$x_i = mp_1^i \wedge mp_2^i \wedge \cdots \wedge mp_p^i \tag{4.4}$$

FIG. 4.32 – *Une cellule du modèle MPL.*

MPL est basé sur une population de cellules c_i (*cf.* équation 4.3 et figure 4.32), qui associent une conjonction x_i de micro-percepts visuels[3] mp_j^i (*cf.* équation 4.4) à une action a_i sur les effecteurs (pondérée par un gain w_i). Une cellule est donc une règle locale, ayant pour condition une conjonction de micro-percepts et pour action une proposition d'action sur les effecteurs. La figure 4.33 présente une vue générale du modèle MPL.

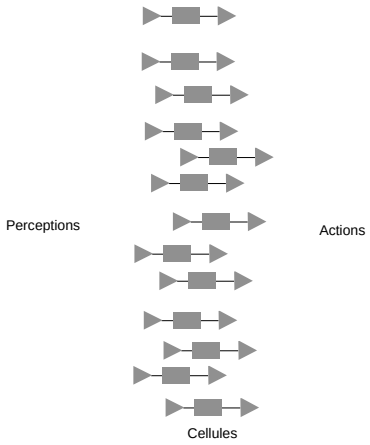
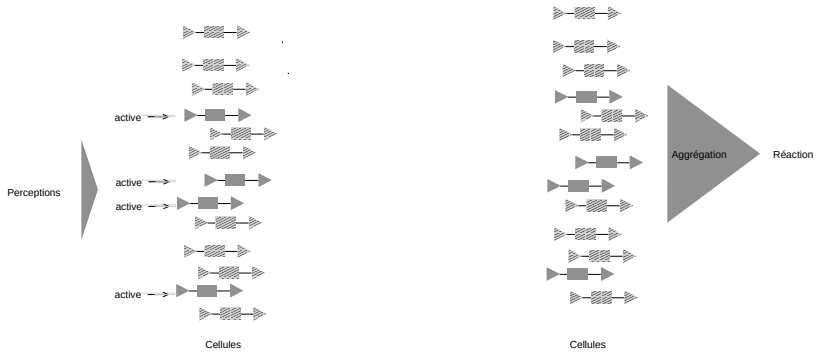


FIG. 4.33 – *Vue générale du modèle MPL. La population de cellules relient perceptions et actions du robot.*

Lorsque l'ensemble des micro-percepts d'une cellule est perçu, la cellule est activée. Pour chaque image capturée par la caméra du robot, un ensemble

3. un micro-percept est une petite partie fixe du champ perceptuel, par exemple pour un micro-percept visuel, un pixel d'une couleur donnée à une position fixée

de cellules sont activées (*cf.* figure 4.34 (a)), et toutes les actions de ces cellules sont agrégées pour former l'action à exécuter par le robot pour cette perception (*cf.* figure 4.34 (b)).



(a) perception (b) action

FIG. 4.34 – *Fonctionnement général du modèle MPL. (a) les perceptions activent des cellules ; (b) les cellules actives proposent chacune une action, et l'ensemble est agrégé pour déterminer l'action courante du robot*

La première instance du modèle, MPL-t (modèle taxique) impose aux micro-percepts d'une même cellule d'être *tous situés à une même position.* L'ensemble des cellules pertinentes pour les démonstrations effectuées par l'utilisateur sont déterminées après un traitement linéaire sur les données, appliqué sur les fichiers regroupant les captures des images et actions associées. Cette première version du modèle a vite posé deux problèmes. Le premier est le nombre croissant du nombre de micro-percepts à traiter, et par conséquent la grande taille de la population de cellules à engendrer. Le second est que les micro-percepts sont considérés indépendamment les uns des autres, ce qui peut produire des comportements incorrects, notamment en cas de suite d'alias perceptuels : une séquence de micro-percepts identiques pour une cellule, mais requérant des actions différentes. Pour faire face à ces

problèmes, Louis HUGUES a alors proposé une seconde version de son système, MPL-s (modèle à échantillon). Grâce à une observation statistique, il a en effet constaté que peu de corrélations entre micro-percepts situés à des positions distinctes étaient nécessaires pour identifier avec plus de 99% de probabilité une image parmi celles d'un ensemble de démonstrations faites au robot. Il a alors proposé de lever la contrainte de MPL-t sur la position identique des micro-percepts d'une cellule. Chaque cellule pouvait alors avec plus de probabilités permettre de distinguer une image dans une démonstration. La phase de détermination des cellules s'appuyait alors sur un traitement statistique et stochastique, dont on pouvait contrôler le coût en temps de calcul et en espace mémoire. Ce second système s'est avéré bien plus générique et plus robuste que le premier, mais le problème de l'absence de notion de temps dans le système demeurait : les comportements appris étaient uniquement réactifs. Le système apprenais à réagir à des perceptions visuelles, mais n'avait pas de moyen de distinguer une situation d'alias perceptuel.

Pour dépasser cette contrainte, nous proposons de remplacer les cellules, qui sont des micro-règles, par des ATN, qui sont équivalents à des ensembles de règles doté d'états internes (*cf.* section 4.3.1).

ATNoCells : remplacer les cellules de MPL-t par des ATN

ATNoCells est notre premier modèle de système multi-agent en développement s'appuyant sur ATNoSFERES. Le rôle du système est de déterminer collectivement, en fonction des perceptions visuelles du robot, l'action à exécuter par chaque roue. Comme les modèles MPL, ATNoCells est conçu pour permettre d'enseigner un comportement au robot grâce aux démonstrations d'un tuteur, sans connaissances a priori sur l'environnement ou la tâche à

exécuter. ATNoCells intègre aussi dès sa conception la possibilité de récompenser ou punir le robot lorsqu'il réalise son comportement.

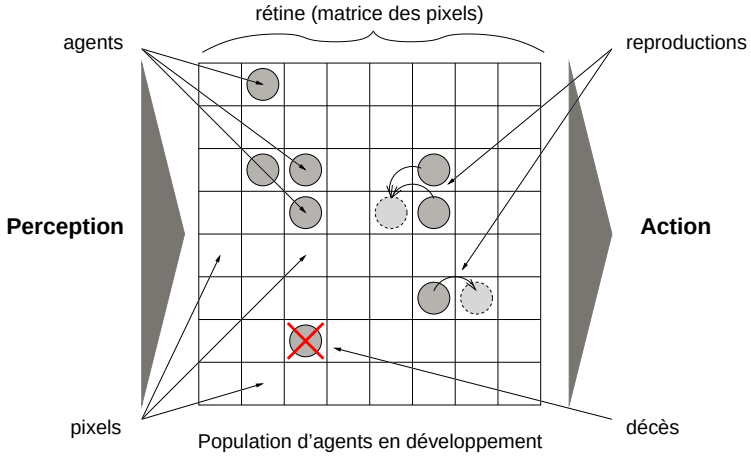


FIG. 4.35 – *ATNoCells, vue d'ensemble de l'environnement des agents. Un agent perçoit uniquement la couleur du pixel sur lequel il se trouve. Les agents peuvent se reproduire par croisement, ou par division*

ATNoCells substitue aux cellules du modèle MPL-t des agents dont les comportements sont décrits par des ATN, et spécifie leur organisation au sein du système. Le système multi-agent vient prendre la place de la population de cellules de MPL. La figure 4.35 l'illustre et est à comparer à la figure 4.33, page 125. L'environnement des agents est spatialisé et discret, chaque case correspondant à un pixel de l'image perçue par le robot. Comme les cellules de MPL-t, les perceptions de chaque agent sont restreintes à celles du pixel associé à sa position, mais par contraste les agents ne peuvent percevoir ici que la couleur du pixel (les micro-percepts de MPL peuvent compter d'autres informations que la couleur).

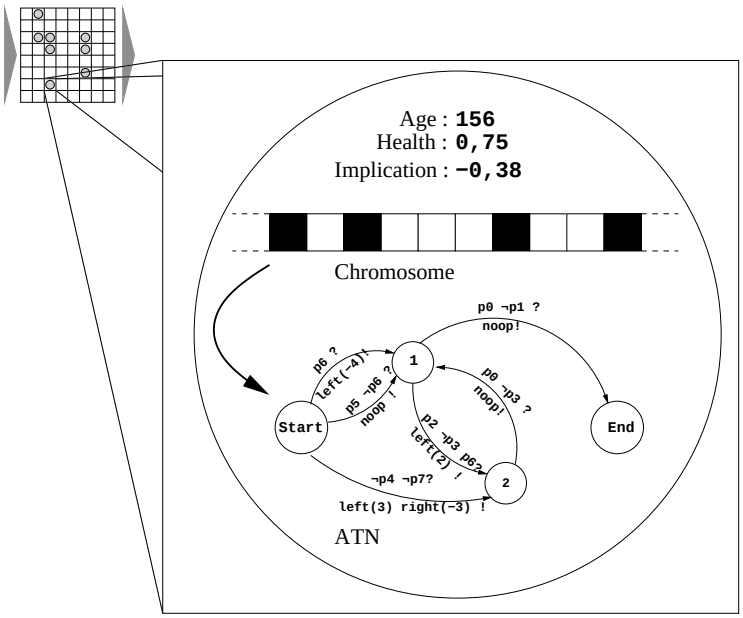


FIG. 4.36 – *ATNoCells, détail d'un agent. Un agent est constitué d'un chromosome (chaîne de bits) et d'un ATN décrivant son comportement, construit à partir du chromosome. Un agent a aussi un état de santé (nombre flottant dans* [0,1]*), un âge (nombre entier) et un taux d'implication (nombre flottant dans* [−1,1]*). La descriptions de l'utilisation de l'ATN est détaillée figure 4.37.*

Description des agents d'ATNoCells

Le comportement de chaque agent est décrit par son ATN (*cf.* figure 4.36). Celui-ci est obtenu par expression d'un chaîne de bits qui constitue le chromosome de l'agent. Chaque agent a aussi plusieurs attributs : un âge, qui correspond au nombre de pas de temps depuis la création de l'agent ; un état de santé, qui est comme nous le verrons ci-après une probabilité (son domaine est donc l'intervalle [0,1]) ; et un nombre appelé implication, dans l'intervalle [−1,1]. Comme préalable nécessaire à l'explication de la raison d'être de ces attributs, nous allons décrire le modus operandi des agent et celui du système en général.

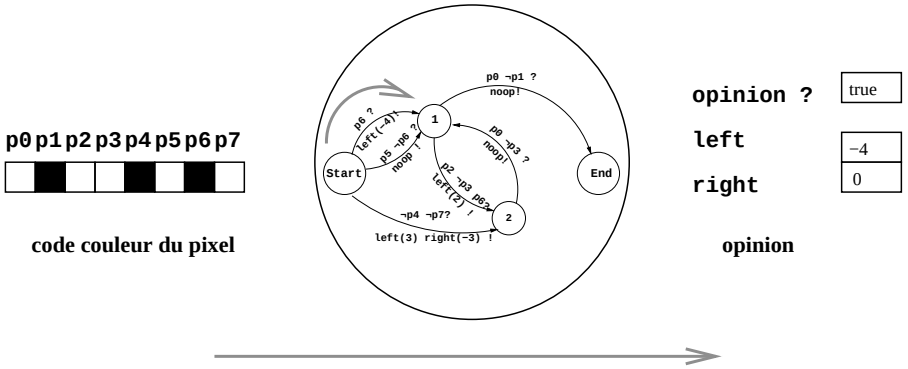


FIG. 4.37 – *ATNoCells, détail du fonctionnement d'un agent. Les conditions de l'ATN portent sur les 8 bits du code couleur du pixel (les bits sont nommés p0,...,p7) et les actions sont soit de n'avoir aucune opinion (`noop !`) soit d'avoir une opinion pour les effecteurs gauche (`left(n) !`) et/ou droit (`right(n) !`). La valuation `n` de l'opinion pour chaque effecteur dépend du nombre d'occurrences du lexème correspondant sur l'arc. Au cas où l'opinion porterait sur un seul effecteur, l'opinion pour l'autre effecteur est considérée égale à 0 (arrêt).*

Un agent ne perçoit que la couleur du pixel sur lequel il est « posé », et la valeur de cette couleur est codée dans un entier sur 8 bits (*cf.* figure 4.37), chaque bit étant nommé *p0*,...,*p7*. Par conséquent les conditions de l'ATN de l'agent sont au nombre de 16, une par bit et par valeur booléenne de celui-ci (*cf.* tableau 4.5). Par exemple, les deux conditions associées à p0 sont `p0?` et `¬p0?`.

Pour ce qui est des actions, l'agent peut soit n'avoir aucune opinion (action `noop!`), soit l'exprimer, en votant pour une vitesse de roue gauche (action valuée `left(n)!`) et/ou une vitesse de roue droite (action valuée `right(n)!`). S'il ne vote explicitement que pour un seul effecteur, nous considérons qu'il vote implicitement pour l'autre avec une valuation de 0, correspondant à l'arrêt. La valuation des éventuelles actions `left(n)!` et `right(n)!` portées par un arc dépend des occurrences de ces actions dans la liste portée par l'arc. Les actions ayant trait à l'effecteur gauche sont réunies en une seule en addi-

Actions		Conditions	
`left(+/-1)!`	incrémenter/décrémenter le vote à gauche	`p0?`/$\neg$`p0?`	vrai si le 1$^{\text{er}}$ bit est à vrai/faux
`left(+/-2)!`	incrémenter/décrémenter de 2 le vote à gauche	`p1?`/$\neg$`p1?`	vrai si le 2$^{\text{ème}}$ bit est à vrai/faux
`left(+/-5)!`	incrémenter/décrémenter de 5 le vote à gauche	`p2?`/$\neg$`p2?`	vrai si le 3$^{\text{ème}}$ bit est à vrai/faux
`left(+/-10)!`	incrémenter/décrémenter de 10 le vote à gauche	`p3?`/$\neg$`p3?`	vrai si le 4$^{\text{ème}}$ bit est à vrai/faux
`right(+/-1)!`	incrémenter/décrémenter le vote à droite	`p4?`/$\neg$`p4?`	vrai si le 5$^{\text{ème}}$ bit est à vrai/faux
`right(+/-2)!`	incrémenter/décrémenter de 2 le vote à droite	`p5?`/$\neg$`p5?`	vrai si le 6$^{\text{ème}}$ bit est à vrai/faux
`right(+/-5)!`	incrémenter/décrémenter de 5 le vote à droite	`p6?`/$\neg$`p6?`	vrai si le 7$^{\text{ème}}$ bit est à vrai/faux
`right(+/-10)!`	incrémenter/décrémenter de 10 le vote à droite	`p7?`/$\neg$`p7?`	vrai si le 8$^{\text{ème}}$ bit est à vrai/faux
`noop!`	ne pas voter/ne pas avoir d'opinion		

TAB. 4.5 – *Lexèmes d'action et de condition pour ATNoCells.*

tionnant les valuations des mêmes actions présentes. Il en va de même pour l'action droite. Ainsi contrairement aux expériences précédentes, la taille variable de la liste des actions est ici importante et exploitée. C'est elle qui permet de définir *extensivement* les valuations des actions de vote pour l'un ou l'autre effecteur. Afin de rendre un peu plus uniformes les probabilités d'obtenir des valuations entières allant de -20 à +20[4], nous utiliserons dans le code génétique des lexèmes valués de valeurs indiquées dans le tableau 4.5. Sans cela, en utilisant uniquement des lexèmes valués avec +1 et -1, nous obtiendrions une distribution de probabilité des valuations tendant vers une loi loi normale centrée en 0[5] avec un écart-type petit, donc une distribution de probabilités trop déséquilibrée en faveur de 0. En utilisant plusieurs valuations (symétriques par rapport à 0) dans les lexèmes, nous augmentons

4. intervalle sur lequel sont segmentées les vitesses des roues

5. puisque cette façon extensive d'obtenir des entiers en additionnant ou retranchant l'unité de façon aléatoires est un problème binomial.

l'écart-type de la distribution et « rééquilibrons » un peu les probabilités pour les différentes valuations.

Enfin, l'action par défaut de l'automate en cas d'absence d'arc éligible au nœud courant, ou en cas d'absence d'étiquettes d'actions sur l'arc élu, est de ne pas avoir d'opinion (action `noop!`).

Fonctionnement général d'ATNoCells

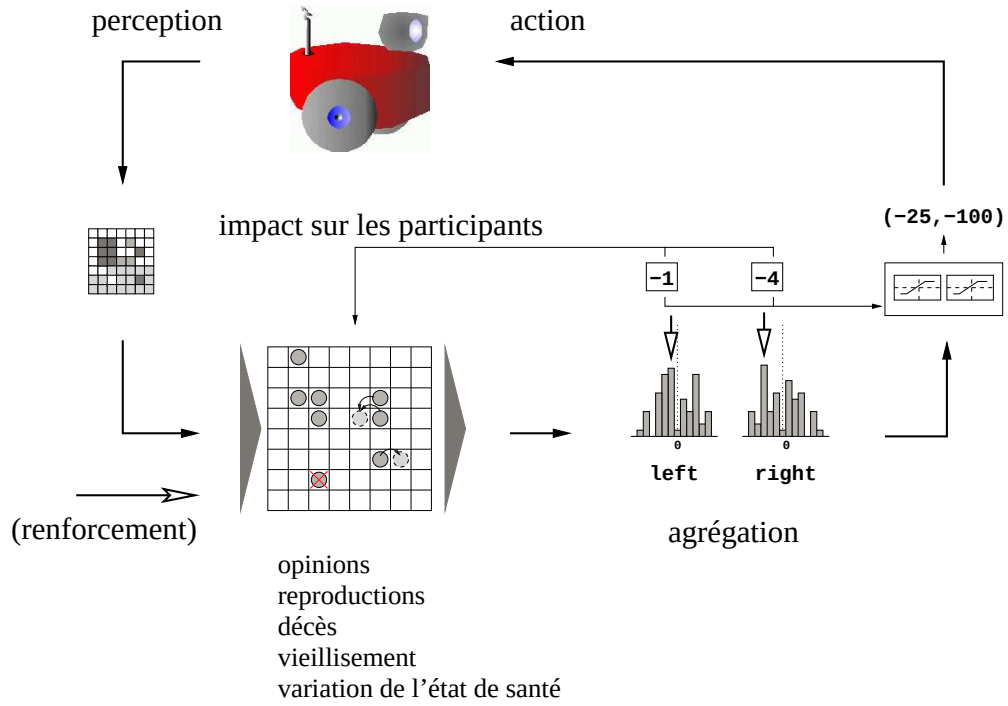


FIG. 4.38 – *ATNoCells, fonctionnement général. En fonction de leurs perceptions, certains agents ont une opinion (la politique de mise-à-jour est décrite figure 4.39). Toutes les opinions vont être agrégées (ici par un vote majoritaire sur chaque effecteur) afin de prendre une décision. En retour, cette décision va permettre de déterminer, parmi les agents qui y ont contribué, s'ils se sont* positivement ou négativement impliqués *(cf. texte).*

La figure 4.38 présente une vue d'ensemble du fonctionnement du système. La perception du robot (lors de l'exploitation du comportement ou de

sa démonstration, *en ligne*) ou son enregistrement (lors de la phase d'apprentissage, *hors ligne*) permet de positionner les couleurs des pixels dans l'environnement des agents. Certains ont alors une opinion qu'ils expriment, et pendant ce pas de temps des agents se reproduisent, d'autres meurent, et ceux qui restent vivants vieillissent. L'état de santé d'un agent peut varier si un renforcement est alors appliqué au système (qui le répercute sur les agents), et cette variation est aussi fonction du taux d'implication de l'agent. Divers types de renforcements peuvent être appliqués, qui sont détaillés dans la section dédiée aux renforcements (page 136).

Les opinions exprimées (indifféremment appelées *votes*) sont ensuite agrégées par le système pour déterminer l'action à entreprendre pour le robot. La décision qui l'emporte est alors traduite en un couple d'ordres aux moteurs, en utilisant une fonction de gain sigmoïdale pour éviter les dépassements de capacité aux valeurs extrêmes. Cette décision sert aussi à déterminer en retour quels agents ayant donné leur opinion ont été *positivement ou négativement impliqués*[6]. L'attribut *implication* des agents (*cf.* figure 4.36) est alors modifié en conséquence ; comme un agent ne peut pas avoir donné plus souvent son opinion que son âge (nombre de pas de temps depuis sa création), l'implication, qui est le rapport de l'équation 4.5, est un nombre de $[-1,1]$.

$$implication(age) = \frac{implications_{+}(age) - implications_{-}(age)}{age} \qquad (4.5)$$

L'implication est utilisée pour calculer le renforcement pour chaque agent lorsque l'ensemble du système reçoit un renforcement (la politique de renforcement est décrite figure 4.40).

6. positivement si leur vote est gagnant, négativement sinon.

```
pour toutes les cases faire
      si case vide
      alors si au moins un voisin pubère
            alors opérer une reproduction (croisement ou division) avec une
probabilité p_repr
            sinon génération aléatoire avec une probabilité p_gen
      sinon si marqué mourant
            alors si pubère
                  alors si au moins un voisin pubère
                        alors reproduction par croisement avec une probabilité
de 0.5, reproduction par division sinon
                  sinon reproduction par division
            sinon marquer mourant avec une probabilité de (1 − health)
            mettre à jour
```

FIG. 4.39 – *Algorithme de reproduction et de mise-à-jour dans ATNoCells. Cet algorithme est appliqué à chaque pas de temps.*

La reproduction et les mises-à-jour

Des agents se reproduisent, meurent et se mettent à jour à chaque pas de temps. La reproduction peut se faire soit par croisement (le chromosome de l'agent-enfant est alors le croisement des chromosomes de deux parents) soit par division (le chromosome de l'agent-enfant est alors une variation[7] de celui de son parent unique). La politique de reproduction et de mises-à-jour est décrite figure 4.39. Un point important est que tous les agents n'ont pas la possibilité de se reproduire. Ils doivent pour cela avoir au moins un certain âge, fixé arbitrairement et à l'avance : c'est ce que nous avons assez naturellement qualifié de *puberté* dans la description de l'algorithme. Cette contrainte paraissait nécessaire pour réguler la reproduction (pour éviter des phénomènes « cancéreux » de croissance incontrôlée d'agents inaptes). Elle fait pleinement partie de la pression sélective sur les agents, en favorisant

7. composition d'une mutation qui inverse les bits avec une certaine probabilité, et de l'opérateur d'insertion-délétion dont nous avons déjà parlé section 4.2.2

les individus plus aptes : le présupposé est que la plupart des agents trop mal adaptés mourront avant d'atteindre l'âge de se reproduire. Cet âge de la puberté est donc à choisir avec soin, au regard de la durée des expériences auxquelles seront soumis le robot et le système. D'autre part, le décès des agents intervient pour partie de façon aléatoire. L'attribut *health* des agents (*cf.* figure 4.36) est considéré comme une probabilité de survivre au pas de temps suivant. À chaque pas de temps, pour chaque agent vivant, un tirage au sort détermine si la « santé » de l'agent est assez bonne pour survivre.

Outre l'âge de la *puberté*, deux autres paramètres fixes interviennent dans l'algorithme : la probabilité p_{repr} d'essayer la reproduction d'un éventuel voisinage (par croisement ou par division avec une égale probabilité) d'une case vide pour la remplir, et la probabilité p_{gen} de générer un individu aléatoirement dans cette même case vide, si la reproduction n'a pas eu lieu.

Les motivations de la politique de reproduction sont d'ordre « spatio-temporelles ». Ni le système, ni les agents n'ont de notions de forme ou d'objet observés sur la rétine (*cf.* figure 4.35). Or, les images que perçoit la caméra du robot ne sont nullement aléatoires[8]. Il faut s'attendre à observer des régularités locales, spatiales et temporelles, sur les couleurs des pixels (taches de couleurs se déplaçant, se déformant, etc.). En particulier, il est probable qu'une couleur détectée à une certaine position, le soit aussi autour de cette position, peut-être légèrement modifiée, plus tard. Il nous paraissait par suite inconcevable que cette notion de localité spatiale et temporelle ne se retrouve pas *dans la politique de reproduction*, puisque *les agents ne se déplacent pas*. Les agents restent immobiles, mais leurs informations géné-

8. même si le bruit sur les caméras numériques grand public que nous utilisons n'est pas négligeable

tiques se déplacent, s'altèrent, s'hybrident au fil du temps, et avec eux, les comportements des agents qu'ils décrivent.

Des considérations d'ordre plus bio-mimétiques viennent en seconde position seulement, même si la première source d'inspiration du modèle est effectivement celle-là. En termes imagés, nous pouvons nous représenter la rétine comme un milieu de culture, une sorte de boîte de Pétri, où une colonie d'animaux monocellulaires prospérerait... Cependant la métaphore s'arrête là. Nous ne saurions dire quels animaux monocellulaires pourraient indifféremment se reproduire par division cellulaire ou par croisement de matériels génétiques, et surtout, avec quelle notion biologique établir une analogie avec la réponse du système, ou avec la pression sélective induite par le tuteur désireux de voir le robot exhiber un comportement. L'application directe de la sélection darwinienne en informatique ne nécessite pas obligatoirement une *caution biologique* et le concept nous paraît suffisamment universel pour se détacher du substrat biologique.

Nous allons maintenant décrire les divers renforcements qui permettent de façonner le système, et qui sont justement le vecteur explicite principal de la pression sélective dans notre modèle.

Les renforcements

ATNoCells est un modèle qui emploie plusieurs sources de renforcement. Ces derniers ne sont pas appliqués avec la même fréquence, n'ont pas la même origine ni les mêmes buts, mais l'application de chacun d'entre eux modifie l'attribut *health*, évoqué dans la section précédente pour son rôle dans la politique de reproduction et de mise-à-jour des agents. De plus toutes les punitions sont appliquées avec la même formule 4.6 (où seul un coefficient change). Il en va de même pour les récompenses (*cf.* formule 4.7). Ces for-

```
agréger [a] les opinions
/* gestion de l'implication */
pour tous les agents faire
      si opinion
      alors si critère d'implication [b]
            alors augmenter l'implication
            sinon diminuer l'implication
      sinon distribuer une punition π0
/* gestion du renforcement court terme */
si agrégation supervisée [c]
alors si décision correcte
      alors pour tous les agents faire
            si opinion
            alors si critère de participation
                  alors distribuer une récompense ρ1
                  sinon distribuer une punition π1
      sinon pour tous les agents faire
            si opinion
            alors si critère de participation
                  alors distribuer une punition π1
                  sinon distribuer une récompense ρ1
/* gestion du renforcement long terme */
si renforcement [d]
alors pour tous les agents faire
      si récompense
      alors si implication strictement positive
            alors distribuer une récompense ρ2
            sinon si implication strictement négative
                  alors distribuer une punition π2
      sinon si implication strictement positive
            alors distribuer une punition π2
            sinon si implication strictement négative
                  alors distribuer une récompense ρ2
```

[a] par un vote majoritaire ou proportionnel.

[b] vote identique à la décision agrégée si vote majoritaire ; vote dont la distance à la décision agrégée est inférieur à l'écart-type si vote proportionnel.

[c] ordre éventuel du tuteur (en ligne) ou réponse attendue (hors ligne).

[d] récompense ou punition du tuteur (en ligne) ou taux de bonnes réponses du système (hors ligne).

FIG. 4.40 – *Algorithme implémentant la politique du renforcement dans ATNoCells. Cet algorithme est appliqué à chaque pas de temps. Se référer au texte pour les descriptions des diverses classes de récompenses et punitions (π_0, ρ_1, etc.).*

mules sont des opérateurs classiques en logique floue (*t-norme et t-conorme probabilistes* (Bouchon-Meunier, 1994)).

$$\begin{aligned} health(t+1) &= health(t) * (1 - \text{coeff}) \\ (health(t)_{\forall t \in \mathbb{N}}, \text{coeff}) &\in [0,1]^2 \end{aligned} \tag{4.6}$$

$$\begin{aligned} health(t+1) &= health(t) + \text{coeff} - (health(t) * \text{coeff}) \\ (health(t)_{\forall t \in \mathbb{N}}, \text{coeff}) &\in [0,1]^2 \end{aligned} \tag{4.7}$$

Les punitions font baisser *health* (tout en le maintenant positif), alors que les récompenses le font augmenter (tout en le maintenant inférieur à 1). Les punitions et récompenses ont par ce biais une influence sur la viabilité d'un agent : plus il est puni, moins il a de chances de survivre, et inversement s'il est récompensé.

La source de renforcement la plus fréquemment appliquée est la punition π_0. Elle est applicable, à chaque pas de temps, à tous les agents qui n'ont pas d'opinion. C'est une punition immédiate et peu coûteuse, de coefficient fixé.

Ensuite viennent la récompense ρ_1 et la punition π_1 associée. Elles sont applicables aux pas de temps où le comportement du robot est supervisé, que ce soit par le tuteur qui le corrige ou le dirige, ou par un enregistrement des perceptions et des réponses attendues. Ce sont des renforcements immédiats et un peu plus coûteux ou bénéfiques que π_0. Le coefficient est ici aussi fixé, plus grand que celui de π_0.

Enfin viennent la récompense ρ_2 et la punition π_2 associée. Elles sont applicables à la discrétion du tuteur. Ce sont des renforcements retardés et plus coûteux ou bénéfiques que π_0 et ρ_1/π_1. Le coefficient est cette fois-ci

variable et déterminé par le tuteur au moment où il décide de l'appliquer. Ce coefficient est modulé par un facteur de $\sqrt[n]{|implication|}$. Ainsi plus l'implication est importante en valeur absolue, plus la récompense ou la punition totale est forte. L'utilisation d'une racine n-ième ($n \in \mathbb{N}/\{0,1\}$) permet de n'avoir pas une relation linéaire et de favoriser un petit peu plus les petits participants – pour les encourager à participer davantage.

4.4.2 Protocole expérimental

Ce système est le plus complexe que nous ayons eu à réaliser, bien plus que ceux mis en œuvre pour les expérimentations précédentes. La réalisation des expérimentations avec ATNoCells a nécessité d'opérer la liaison entre SFERES d'une part, et l'interface utilisateur, le protocole d'exploitation des séquences et le protocole de communication avec le robot programmés par Louis HUGUES pour ses propres travaux de thèse. Ce couplage s'est avéré plus long et plus complexe à réaliser que prévu de prime abord, et n'a pour l'instant été réalisé que partiellement. La cause de ces difficultés de couplage est que les programmes de Louis HUGUES n'ont pas été prévus pour être utilisés dans un cadre général. Ils sont centrés sur les traitement à opérer par chacune des cellules des modèles MPL, alors qu'ATNoSFERES fait intervenir le système multi-agent comme un tout.

Les expérimentations se déroulent en plusieurs temps (Hugues, 2002) :

1. d'abord une phase de démonstration, où le tuteur pilote le robot pour réaliser des enregistrements des couples (*perceptions*,*actions*) ;
2. ensuite une phase d'apprentissage hors ligne, où les expériences enregistrées sont soumises au système, dans un ordre aléatoire. Les renforcements sont alors appliqués de la façon décrite dans l'algorithme de la figure 4.40 ;

3. enfin vient la phase de réalisation du comportement par le robot. Le système est connecté au robot, et les renforcements sont appliqués dans le contexte en ligne. Le tuteur peut alors ne pas intervenir et observer passivement le robot, appliquer un renforcement long terme ($\rho 2$ ou $\pi 2$), ou corriger le robot en prenant les commandes, ce qui appliquera implicitement des renforcement court terme ($\rho 1$ et $\pi 1$) sur les agents qui se sont exprimés lors des prises de décision du système.

Il serait en théorie envisageable de n'avoir pas besoin des deux premières phases, et de faire les démonstrations en même temps que le système apprend, et de les exploiter immédiatement. C'est ce que nous espérons pouvoir faire à terme, cette façon de faire étant la plus commode et la plus simple pour l'utilisateur. Cependant, si le nombre des démonstrations nécessaires pour obtenir le comportement voulu est trop grand, ce protocole d'apprentissage par démonstration ne sera pas utilisable. Si une dizaine de démonstrations successives paraissent raisonnables, une centaine lasseraient l'utilisateur humain le plus motivé. N'ayant aucun moyen d'évaluer a priori le nombre des soumissions d'exemples nécessaire, nous procédons comme décrit ci-avant.

Par ailleurs, la complexité bien plus grande d'ATNoCells par rapport aux expérimentations précédentes – qui ne reposent que sur un seul agent dans un environnement simulé – imposent des outils d'analyse plus complexes que nous n'avons pas eu le temps de développer. Les analyses de nos résultats préliminaires sont donc incomplètes. Elle repose sur des statistiques démographiques comme la taille moyenne de la population, le taux de natalité, le taux de mortalité, l'espérance de vie et la taille moyenne des chromosomes. Néanmoins, il est tout de même possible de tirer des premières conclusions grâce à ces indicateurs.

À fins d'évaluation logicielle autant que de vérification préalable de la fonctionnalité du modèle proposé, notre première expérimentation consiste à apprendre une tâche très simple, sans même essayer de généraliser sur les expériences acquises (avec des contextes environnementaux différents notamment). Cette tâche consiste à enseigner au robot à l'arrêt à accélérer puis avancer à vitesse constante vers un panonceau STOP, et l'apprentissage hors-ligne est fait sur une seule démonstration.

FIG. 4.41 – *Vue d'ensemble de l'environnement de la tâche slalom.*

La seconde tâche est plus longue et plus complexe, et nous avons enregistré 4 démonstrations pour permettre au système de généraliser. Cette tâche consiste en un déplacement dans un couloir du laboratoire, un slalom autour de 3 repères colorés (*cf.* figure 4.41), pour s'arrêter devant un quatrième (juste au-dessus du robot dans l'image).

Pour ce qui est des paramètres, les taux de mutations et d'insertion-délétion sont restés identiques à ceux des expériences précédentes, et nous avons défini comme intervalle de tailles initiales pour la générations aléatoire des chaînes de bits un intervalle de [10,100] (en codons), plutôt que d'avoir à fixer une taille unique pour tous les individus. Les dimensions de la rétine

sont 40 pixels de large pour 30 de haut, et la vitesse des effecteurs du robot est divisée en 41 segments en tout (20 en arrière, le 0 pour l'arrêt, et 20 en avant). Pour chaque série de démonstrations, l'âge de la puberté a été pris égal à environ le quart du nombre d'images de la séquence la plus longue. D'autre part, nous n'avons pas beaucoup fait varier les divers paramètres du système. Le taux de reproduction était d'environ 60%, le taux de génération aléatoire d'environ 10%, les coefficients pour les différentes récompenses et punitions étaient déterminés de façon à ce qu'un individu n'ayant pas d'opinion pendant toute la durée d'une séquence ait une santé de 0,5, et qu'un individu qui se tromperait jusqu'à l'âge de la puberté ait une santé de 0,75 (3 chances sur 4 de survivre au pas de temps suivant).

Par ailleurs, deux types d'agrégations ont été utilisées. La première est un vote majoritaire : le couple ($left$,$right$) plébiscité est adopté par le système, et sont considérés comme impliqués dans cette décision les agents dont la décision est exactement la même. La seconde est un vote proportionnel : la décision adoptée est pour chaque effecteur, la moyenne des votes associés à cet effecteur, et les impliqués sont les individus dont le vote sur chaque effecteur est à une distance inférieure à l'écart-type de la moyenne des votes.

Enfin, l'action par défaut en cas d'absence d'arc éligible était de se suicider, pour forcer tous les agents à toujours prendre en considération la situation courante (quitte à ce que ce soit avec un arc sans conditions).

4.4.3 Résultats préliminaires

Première expérimentation

La première expérimentation avait pour objectif de vérifier que le modèle est fonctionnel. Elle était faite sur une seule démonstration comptant 72

images (*cf.* figure 4.42). Le robot à l'arrêt, doit accélérer puis avancer à vitesse constante vers un panonceau marqué « stop ».

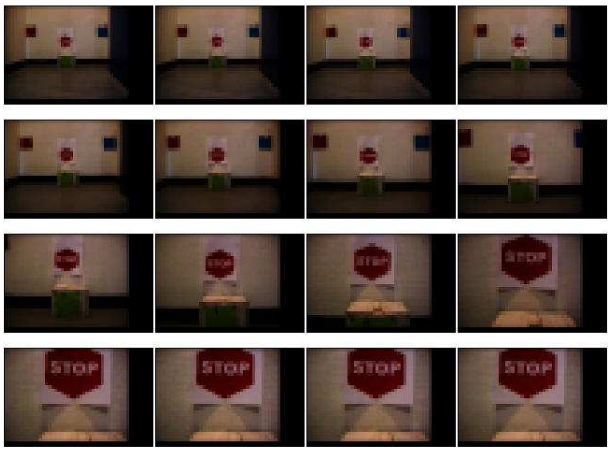


FIG. 4.42 – *Quelques images tirées de la démonstration GoStop, qui compte 72 images en tout.*

Nous avons constaté que très rapidement le système parvient à répondre correctement au moins 90% du temps. Avant d'atteindre ce taux de bonnes réponses, le taux oscille autour de 4%, sans dépasser 5%. La transition se fait en une ou deux *générations*[9], où le taux passe par une valeur intermédiaire pour atteindre les 90%. Après avoir atteint cette valeur une première fois, le taux oscille entre 90% et 92%. Ce seuil de 90% est atteint en moyenne en une douzaine de générations avec le vote majoritaire, une trentaine avec le vote proportionnel. Certaines expériences avec vote majoritaire l'atteignent en seulement 3 ou 4 générations, ce qui est assez surprenant car très rapide par rapport aux temps de convergence des algorithmes évolutionnistes clas-

9. le sens de ce mot est différent de celui qu'il a dans le contexte des algorithmes évolutionnistes. ATNoCells est un modèle de développement continu, et la notion de génération recouvre ici une période de durée arbitraire en pas de temps. Pour chaque séquence de démonstration au système, nous l'avons prise égale à la durée totale en nombre de pas de la séquence. Par exemple pour la première tâche qui ne compte qu'une démonstration, la durée d'une génération est de 72 pas de temps.

siques que nous avons employés lors des expérimentations précédentes avec ATNoSFERES.

Les attributs de la population changent beaucoup au moment de l'explosion du taux de réussite. Les changements qui sont observés sont beaucoup plus rapides dans le cas du vote majoritaire. Avant d'atteindre 90% de réussite, la taille moyenne de la population oscille entre 300 et 400 individus[10], il meurt et naît entre 5 500 et 6 500 individus par génération ; la moyenne d'âge est d'environ 5 avec un écart-type d'en moyenne environ 6, l'espérance de vie est elle aussi d'environ 5 avec un écart-type d'environ 3 ; la taille moyenne des chromosomes est alors dans toutes les expériences, de façon étonnante, de 34 ou 35 codons (et non pas $\frac{10+100}{2} = 55$ comme l'on aurait pu s'y attendre) avec un écart-type qui oscille entre 14 et 15. Après avoir atteint 90% de réussite, la taille de la population augmente jusqu'à environ 1 000 individus dans le cas majoritaire, 1 100 dans le cas proportionnel. Il meurt et naît de moins en moins d'individus, mais l'ordre de grandeur reste le même : entre 4 500 et 5 500 individus par génération pour le cas majoritaire, et un peu moins, entre 3 500 et 5 000 pour le cas proportionnel. La moyenne d'âge augmente assez régulièrement jusqu'à environ 25, et l'espérance de vie augmente aussi mais plus lentement, jusqu'à 12 environ, dans les deux cas. La taille moyenne des chromosomes augmente régulièrement et lentement (en moyenne de 1 codon par génération) dans le cas majoritaire, alors que dans le cas proportionnel la plupart du temps la taille moyenne diminue jusqu'à environ 20 (le minimum observé a été de 15 codons de moyenne). Dans tous les cas, l'écart-type, lui, chute entre 4 et 7.

10. pour un maximum de $40 * 30 = 1\ 200$

Deuxième expérimentation

Cette expérimentation devait servir à vérifier que le système permettait d'enseigner une tâche composée au robot, c'est-à-dire une tâche qui nécessite de prendre une suite de décisions différentes dans des contextes différents (les virages du slalom autour de bornes de couleurs différentes). Les expériences sont en cours. Le système ne converge pour l'instant jamais ; les expériences sont faites sur 4 démonstrations différentes, de longueurs comprises entre 200 et 270 images (*cf.* figure 4.41).

Pour le cas majoritaire, les taux de réussite oscillent entre 5% et 15%. La taille de la population reste d'environ 380 individus, pour entre 7 000 et 7 500 naissances et décès par génération. La moyenne d'âge est assez stable pour chaque expérience, entre 5 et 10, pour une espérance de vie d'environ 5. Les chromosomes ont une longueur moyenne de 55, avec un écart-type de 25.

Pour le cas proportionnel, les taux de réussite oscillent entre 0% et 10%. La taille de la population varie aussi de quelques centaines à presque un millier d'individus. La moyenne d'âge peut atteindre 40 pour une espérance de vie d'une quinzaine de pas de temps. Les chromosomes ont une longueur moyenne toujours inférieure à 55 et d'écart-type ici aussi de 25 environ.

4.4.4 Analyses préliminaires

Première expérimentation

De la même façon que la première expérimentation avec ATNoSFERES a servi à vérifier la fonctionnalité du modèle et à commencer à entrevoir les propriétés et capacités d'ATNoSFERES pour un agent seul (*cf.* section 4.1), cette première expérimentation avec ATNoCells nous a permis de réaliser que

le degré de complexité supérieur du cadre multi-agent nécessite des outils de visualisation et d'analyse adaptés. Être capable de décrire le comportement d'un agent ne suffit plus, ce que nous observons est la résultante du comportement collectif des agents, dont certains disparaissent et d'autres apparaissent à chaque pas de temps. Démunis d'outil de visualisation adapté, nous éprouvons des difficultés à expliquer en détail le fonctionnement du système, même sur cette expérience très simple. Nous constatons tout de même que le modèle est fonctionnel, puisque nous parvenons à obtenir des systèmes qui répondent correctement 90% du temps sur cette tâche très simple ne nécessitant pas de généralisation.

Une analyse est tout de même possible. Il ne faut pas s'étonner de la capacité du système à converger vers une solution stable, puisque l'apprentissage se fait sur un seul exemple. Le taux de 90% de réussite est légèrement supérieur au nombre d'étapes dans la séquence où le robot avance à vitesse constante. Il suffit donc qu'une coalition d'agents boucle sur cette même proposition, sans rien percevoir, pour qu'ils atteignent l'âge minimum pour se reproduire et que le génome se répande rapidement dans la population. L'accroissement de la taille de la population, de l'âge moyen, de l'espérance de vie ainsi que la décroissance des deux écarts-types associés et celui de la longueur des chromosomes s'expliquent dès lors aisément, par une homogénéisation de la population, qui n'est cependant jamais acquise puisque tout individu est en sursis, même si son comportement est parfait. Dans cette optique de diffusion des gènes, sur cette expérience, la différence entre le vote majoritaire et le vote proportionnel semble juste affecter les vitesses de propagation sur la rétine. Il semble toutefois d'après les expériences que plus la transition de phase a lieu tôt pour le cas proportionnel, plus les variations démographiques seront rapides.

Deuxième expérimentation

Tout semble indiquer que, dans nos résultats préliminaires, la population reste très aléatoire dans le cas majoritaire, un peu moins dans le cas proportionnel, mais sans que cela se traduise par une meilleure performance. Dans le cas proportionnel, les agents peuvent en effet s'en sortir à meilleur compte même sans collectivement donner des réponses satisfaisantes : par exemple s'ils donnent des réponses très variées de façon à ce que l'écart-type des votes soit très grand. Cela pourrait expliquer les moyennes d'âges élevées parfois.

En stabilisant et en homogénéisant la population pour former une base d'amorçage (diminution du taux de génération aléatoire, augmentation du taux de reproduction et diminution de l'âge de la puberté), nous avons observé dans le cas proportionnel, et pour certains jeux de paramètres, des taux de réussite temporaires allant jusqu'à 60%. Cette amélioration sensible s'accompagne d'une augmentation de la taille de la population et de sa moyenne d'âge. Il est clair que modifier les paramètres de façon à ralentir encore la multiplication permettrait d'obtenir des gains significatifs, mais sans outils d'analyse et de visualisation, il paraît difficile de déterminer autrement que par une marche au hasard comment les modifier. De plus, comme nous le verrons dans la discussion, ce n'est sans doute pas de ce coté (paramétrage manuel) que nous devons chercher les conditions pour générer des systèmes totalement autonomes.

Conclusions sur les expérimentations en cours

Le modèle est particulièrement dynamique et instable. La première expérimentation, très simple, permet cependant de faire converger le système vers un équilibre stable donnant de bonnes réponses. La seconde ne le permet

apparemment pas avec les paramètres que nous avons essayés, mais, grâce à quelques expériences où nous avons tenté de « ralentir » le système, la réponse fonctionnelle qu'il fournit semble correcte.

Nous devons maintenant terminer le développement logiciel afin de pouvoir passer en phase de réalisation (par le robot) pour la première expérimentation. Pour la deuxième et toutes celles que nous envisageons, il est indispensable de développer des outils de visualisation et d'analyse pour pouvoir mieux les interpréter (et les diriger).

En particulier, nous avons planifié une expérimentation GoStopAndBack qui illustre la capacité du système à gérer les alias perceptuels. Le comportement simple envisagé est une extension simple de la première expérimentation : enseigner au robot à aller vers le panonceau STOP, à s'arrêter à une certaine distance de ce panonceau puis à reculer.

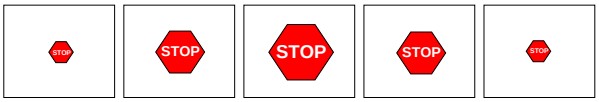


FIG. 4.43 – *Schémas des perceptions du robot dans une expérience avec alias perceptuel. Dans cette expérience, le robot devrait apprendre à s'approcher d'un panonceau STOP jusqu'à une certaine distance, puis à s'en éloigner en reculant.*

Ce comportement, en apparence peu complexe à apprendre, pose cependant un redoutable problème aux architectures ou aux modèles qui n'intègrent ni une gestion du temps, ni celle d'états internes. Au regard des résultats que nous avons obtenus concernant les alias perceptuels (*cf.* section 4.3.2) et ceux, préliminaires, concernant l'évolution de la population d'agents (*cf.* la première expérimentation impliquant ATNoCells), nous pouvons raisonnablement penser que ATNoCells sera à même de fournir des résultats très intéressants, qu'il nous faudra sans doute analyser finement avant de pouvoir conclure.

Chapitre 5

Discussion

5.1 ATNoSFERES

ATNoSFERES est une 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 classique en informatique, de plus facilement compréhensible par tous. Ceci permet de concevoir ou de tenter d'améliorer manuellement des solutions, afin de les soumettre à nouveau à la machine comme amorçage (Pitrat, 1990).

5.1.1 Rôle du non-déterminisme

Si le non-déterminisme des graphes introduit du bruit lors des évaluations, il est de toute façon inévitable lorsque l'on évalue un comportement d'agent situé. Comme le comportement est le résultat du couplage entre l'agent et son environnement, lorsque ce dernier n'est pas contrôlé, que les perceptions de l'agent sont incomplètes ou imprécises (tous les capteurs robotiques le sont), alors chaque expérience sera unique. À noter que toutes les évaluations

des individus, notamment en robotique évolutionniste, se font sur plusieurs évaluations des comportements produits par un même génome.

L'utilisation d'un automate non-déterministe nous a paru être importante. Elle est intéressante en ce qu'elle confère plus de liberté à l'agent qui, dans la même situation, peut ne pas exprimer le même comportement. Des solutions intermédiaires lors de nos expériences avec ATNoSFERES montrent que le processus sélectionniste exploite cette propriété : certains arcs d'un nœud vers un autre sont ainsi représentés 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 qui sont éligibles. Ainsi, au fil des générations, les individus « apprennent » à choisir statistiquement une voie plutôt qu'une autre. Lorsque les solutions s'améliorent globalement, que les individus homogénéisent leur choix entre des arcs qui ne génèrent pas les mêmes actions, des contraintes qui semblent à l'œuvre de façon implicite sur le génome peuvent réduire le nombre d'arcs qui se recoupent pour ces graphes, n'en retenant finalement qu'un seul.

Le non-déterminisme joue donc un rôle important dans l'évolution car cette liberté conférée aux comportements permet d'élargir le champ des possibles, augmentant ainsi les probabilités pour l'évolution de trouver de bonnes solutions.

5.1.2 Un modèle à la fois symbolique et numérique

La description du comportement par un automate non déterministe est une description symbolique qui, comme nous l'avons montré, peut être traduite en terme d'ensemble de règles dotées d'états internes. Cependant, la façon d'obtenir la description des solutions cherchées, comme c'est le cas pour les systèmes de classeurs, se fait avec des méthodes évolutionnistes tra-

ditionnellement considérées comme numériques. C'est d'autant plus vrai pour ATNoSFERES que la représentation génétique manipulée est une chaîne de bits, sans signification ni hiérarchie particulière.

ATNoSFERES représente à nos yeux un bon compromis entre la puissance des méthodes purement numériques et le pouvoir d'expression (et de manipulation par un concepteur humain) des méthodes symboliques.

5.1.3 Adaptation au niveau de complexité du problème

ATNoSFERES permet de décrire tant des comportements réactifs que des comportements plus cognitifs, impliquant une représentation interne en plus des perceptions immédiates de l'environnement. Cependant, il n'est pas nécessaire de faire d'hypothèses sur le niveau de complexité du comportement que l'on désire obtenir avant de lancer le processus d'évolution. Le modèle peut apparemment s'adapter automatiquement à cette complexité, sans aucune heuristique explicite, même pour des problème non-markoviens. Les solutions trouvées sont simples (pas forcément optimales), avec un nombre réduit, mais suffisant, d'états internes. Un comportement purement réactif sera représenté par un graphe à nœud interne unique, qui n'a d'arcs que vers lui-même. Un comportement plus cognitif sera modélisé par un graphe à plusieurs nœuds internes, c'est-à-dire contenant plusieurs nœuds connexes.

De plus, ATNoSFERES a montré des capacités à produire non seulement des comportements simples, mais aussi des graphes de description eux-mêmes assez dépouillés, avec en général seulement deux ou trois nœuds, selon le problème étudié. Il n'a jamais été nécessaire de limiter la taille de la chaînes de bits comme c'est pourtant le cas dans tous les travaux apparentés (*cf.* section 2.2.1). En particulier, les graphes produits lors d'une utilisation sans mutation systématique des individus ne contiennent pas de composante connexe

qui ne soit pas reliée avec le point d'entrée du graphe. Nous pensons que de telles composantes connexes, du fait de la nature répartie du langage de construction, seraient fragiles et difficiles à maintenir héréditairement : ne serait-ce qu'à cause des connexions implicites en fin de construction de graphe qui pourraient rompre l'isolement d'une telle composante connexe, dégradant alors fortement le comportement de l'individu. Un caractère difficilement transmissible, qui représente un gros risque héréditaire, est clairement mis à l'écart dans le processus de sélection que nous utilisons.

Nous songeons à étudier plus avant cette conjecture d'adaptation au niveau de complexité d'un problème non-markovien. Si elle est vérifiée dans suffisamment de cas représentatifs (nous devrons mener beaucoup plus d'expériences), étant donné la simplicité de sa mise en œuvre, ATNoSFERES pourrait alors servir comme outil d'estimation de la complexité d'une tâche dans un environnement non-markovien.

5.2 ATNoCells

ATNoCells n'a dans son implémentation pas encore été suffisamment exploré pour pouvoir tirer des conclusions sur ce sujet. Sa conception même est cependant source de réflexions nouvelles.

5.2.1 Un modèle de sélection darwinienne pour l'apprentissage par démonstration

À notre connaissance, ATNoCells est le seul modèle d'apprentissage par démonstration utilisant un modèle de type sélection darwinienne. D'habitude représentés par les algorithmes évolutionnistes dont nous avons souligné quelques faiblesses dans le contexte de la conception de comportements

d'agents et de systèmes multi-agents (*cf.* section 1.2.5), les modèles s'appuyant sur la sélection darwinienne sont écartés car trop coûteux en temps de développement, en expertise, et en temps de calcul. Les quelques expériences que nous avons menées semblent au contraire suggérer que nos espoirs d'un « retour aux sources » (*cf.* section 1.2.6) de la sélection darwinienne s'accompagne de meilleures performances. S'ils ne sont pas encore vérifiés rigoureusement pour l'instant, ils ne sont en tout cas pas vains.

5.2.2 Alias perceptuels et notion de succession temporelle

ATNoCells décrit un système composé d'automates à états finis, construits selon le modèle ATNoSFERES. Le système est donc armé pour pouvoir faire face aux alias perceptuels, puisque, comme nos expériences l'ont montré, chaque automate en est capable. L'utilisation d'architectures d'agent disposant d'états internes permet aussi d'envisager des successions de tâches quand l'environnement ne suffit pas pour déterminer quelle tâche il faut exécuter. Cette notion de succession temporelle est la conséquence de la possibilité de faire face aux alias perceptuels, et permet d'envisager l'apprentissage implicite d'un modèle de sélection d'action (Guillot and Meyer, 2000).

5.2.3 Apprentissage *pendant* la démonstration

Théoriquement, ATNoCells pourrait permettre à un robot d'apprendre un comportement par démonstration, sans connaissances préalables de l'environnement ou de la tâche, *pendant* que la démonstration est exécutée. Les renforcements appliqués en ligne pendant que le tuteur effectue la démonstration sont exactement les mêmes que ceux appliqués hors-ligne hors démons-

tration. Il n'y a pas de séparation entre un mode supervisé (pour le système), un mode renforcé uniquement (par une appréciation du tuteur), et un mode d'exploitation autonome (sans interaction avec l'utilisateur). C'est suivant les informations que reçoit le système que les renforcements sont appliqués ou pas. Les trois modes sont en quelque sorte simultanés. Cette façon de concevoir un système apprenant nous paraît ergonomique, encore reste-t-il à démontrer son efficacité. Nous n'avons malheureusement pas eu le temps d'explorer plus avant cet aspect expérimentalement.

5.2.4 Un système encore incomplet

Nous pensons qu'en l'état ATNoCells est encore incomplet. Considéré comme un écosystème d'agents, *lorsque le système fonctionne de façon autonome* (*i.e.* sans aucune intervention du tuteur), il n'est en effet constitué que d'un seul niveau, tous les agents étant de même nature. En termes écologiques, le *circuit de rétroaction* formé par ce seul niveau des agents composant le système, agissant directement sur le robot par les effecteurs, et le robot, qui en retour agit sur le système en fournissant les perceptions visuelles, n'est composé que de deux éléments (Thomas and Kaufman, 2001). Le robot peut être considéré comme un circuit positif à cause des variations sur les perceptions (d'autant plus grandes que le robot est en mouvement). Donc, le contrôle du robot par ATNoCells en mode autonome donnera un comportement stable uniquement si le système multi-agent a atteint un équilibre qui lui permet de modérer et réguler le comportement du robot (le rendant assimilable à un circuit négatif). Or, livré à lui-même, ATNoCells pourrait tout-à-fait s'avérer être instable. En effet, *il n'a aucun contrôle explicite de sa propre activité par une sous-partie de lui-même.* Seul fonctionne le mécanisme de reproduction, vie et mort des agents, qui n'est presque pas régulé

(seules les punitions π_0 liées à l'absence d'opinion sont appliquées, *cf.* figure 4.40 page 137) lorsque le système fonctionne de façon autonome.

Un niveau *meta* (Pitrat, 1990; Pitrat, 1995) serait nécessaire à ATNoCells pour qu'il puisse être utilisé de cette façon. Par exemple, l'ajout d'un niveau d'auto-observation par des sous-systèmes du système multi-agents pour contrôler l'organisation de l'ensemble (Cardon, 1999), serait une piste de recherche que nous comptons poursuivre.

En effet, les états d'équilibres que le système peut atteindre par lui-même semblent généralement être des états d'équilibre instables, sauf exceptions (par exemple celui de la première expérimentation, lorsque le système trouve un équilibre dynamique stable en saturant presque l'environnement avec des agents qui doivent probablement avoir le même comportement).

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

5.3.1 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 (Angeline, 1994a; Angeline, 1994b). C'est une propriété nécessaire, puisque comme nous ne connaissons rien par avance de la forme de la 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 (Koza, 1992), ou la longueur de la chaîne pour les diverses approches de programmation génétique par pile (Perkis, 1994; Stoffel and Spector, 1996),

nous n'avons pas eu besoin d'ajouter de telles contraintes à notre modèle. N'ayant pour l'instant pas conçu d'autres modèles suivant ce principe d'expression à l'aide d'une pile, nous ne sommes pas en mesure de déterminer pour quelles parts les propriétés remarquables que nous avons obtenues sont dues au principe ou au modèle de construction d'une structure particulière (un ATN).

Nous pensons cependant que les mérites sont partagés. Pour ce qui est du modèle, nous avons exprimé notre conjecture ci-dessus (en remarquant l'absence de composantes connexes dans les graphes). Pour ce qui est du principe, nous pensons qu'une propriété due à la localité du langage est aussi à l'œuvre, et qu'un équilibre est trouvé, pendant l'évolution, entre une tendance à l'allongement et une tendance au raccourcissement. Notre conjecture pour ce point est que :

« – si la longueur de la chaîne est trop courte, l'impact d'une mutation ou d'une insertion-délétion est statistiquement plus important. Dans ces conditions le génotype ne contient pas beaucoup de « code poubelle » pour se protéger des variations. Par analogie avec le modèle biologique (*junk DNA*, « *ADN poubelle* »), « 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, comme nous les avons surtout observées dans les deux premières expérimentations ;

« – 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, 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.

Ces contraintes sur la longueur des chaînes agiraient ainsi comme une contrainte implicite sur les structures produites, puisque la longueur de la chaîne limite le nombre de nœuds, d'arcs et d'étiquettes dans le graphe construit. Il en irait inversement pour la conjecture que nous avons émise au sujet d'ATNoSFERES dans la section 5.1.3, comme quoi les composantes connexes constitueraient dans certains cas une fragilité structurelle qui en retour viendrait contraindre la chaîne de bits.

5.3.2 Séparation entre syntaxe et 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 codages indirects basés sur une grammaire sans contexte. 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 (Koza, 1992) ou au contraire un typage fort (Montana, 1995) en programmation génétique). Ceci autorise une flexibilité totale pour les opérateurs génétiques. La chaîne de bits peut être manipulée par un processus « aveugle » sans avoir à aucun moment à considérer son interprétation. Nous n'avons par ailleurs pas besoin d'opérateurs spécifiquement liés au type de structure qui est évolué. Nous avons d'ailleurs rapidement adopté un opérateur d'insertion-délétion de paquets de bits qui à notre connaissance a été très rarement utilisé dans les algorithmes génétiques. Cet opérateur permet en

particulier de gérer les changement de dimension de la chaîne de bits de façon très graduelle. C'est donc une cause efficiente de l'adaptation automatique du niveau de complexité de la structure produite, comme évoqué ci-dessus (*cf.* section 5.1.3).

5.3.3 Redondances

La description des différentes parties de la structure n'est pas dépendante de la position dans la chaîne de bits, plusieurs combinaisons de sous-chaînes pouvant produire la même structure. Ce qui paraît 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 peut 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 mises dans un ensemble, les secondes dans une liste).

Un autre type de redondance est présent, celui du code génétique. En fonction du nombre de lexème disponibles, le code génétique peut être plus ou moins redondant. En effet, le code génétique est une fonction surjective (*cf.* page 52). 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).

La première redondance dans l'interprétation et le fait de pouvoir obtenir une même structure à partir d'une multitude de chaînes différentes

Nom	Dispositif de génération	Dispositif de reconnaissance
Langages récursivement énumérables	Grammaires générales *Grammaires de type 0*	Machines de Turing *Automates à 2 piles (pouvant boucler)*
Langages récursifs	?	Machines de Turing *Automates à 2 piles (s'arrêtant toujours)*
Langages contextuels	Grammaires contextuelles *Grammaires de type 1*	Automates bornés linéairement
Langages non contextuels	Grammaires algébriques *Grammaires de type 2*	Automates à pile non déterministes
Langages non contextuels déterministes	?	Automates à pile déterministes
Langages rationnels	Grammaires linéaires droites, expressions régulières *Grammaires de type 3*	Automates finis

TAB. 5.1 – *Classification des grammaires au sens de* CHOMSKY *(Chomsky and Miller, 1963; Stern, 1990)*

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 sur les statistiques). La seconde permet de créer des codes génétiques offrant des propriétés comme la résistance aux mutations, ou la possibilité de jouer sur les probabilités de tirage aléatoire de chaque codon.

5.3.4 Grammaire

Comparé aux autres types de langages utilisés pour faire évoluer des structure (*cf.* section 1.3.2), *ce lui que nous utilisons ne contraint pas la représentation lue* (la chaîne de bits).

Il faut d'abord noter que, comme dans les langages à pile classique, la pile n'en est pas une au sens strict du terme. C'est plutôt une liste. Elle ne joue le rôle de pile que pour la plupart des instructions, qui n'utilisent

directement que son sommet. Il existe cependant des instructions qui procèdent à des accès directs à des cellules autres que le sommet, ce qui, dans un modèle théorique d'automate à pile (Chomsky, 1962), suppose l'existence d'une deuxième pile dans laquelle seront stockées temporairement les cellules sautées lors de l'accès direct. Alors que les langages reconnus par un automate à une seule pile sont justement les langages générés par des grammaires sans contexte (grammaire de type 2 au sens de CHOMSKY, *cf.* tableau 5.1), qui sont très utilisés dans les codages indirects, les langages reconnus par deux piles sont pour leur part plus généraux. Leurs grammaires sont moins contraintes : ce sont au moins des grammaires contextuelles. Ici, le contexte est explicitement spécifié lors de la conception des actions liées aux lexèmes (qui savent comment, où et sur quoi opérer dans la pile).

Cette plus grande souplesse des langages contextuels explique pour partie celle de notre principe de construction de structures à l'aide d'une pile (*cf.* section 2.2).

5.3.5 Adaptation cumulative

Grâce aux petites variations du génotype et à la quasi continuité comportementale entre parents et descendants, l'adaptation des individus au fil des générations croît en moyenne de façon cumulative par de faibles variations. Cette quasi continuité depuis les variations jusqu'aux comportements exprimés est pour nous un axe principal de conception par évolution artificielle. La quasi 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.

5.3.6 Pouvoir d'expression

Le langage de construction de graphes 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 bits 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 (Angeline, 1995), nous avons un bon candidat pour le compromis entre le pouvoir d'expression et la finesse du réglage des structures construites. Au formalisme prêt, nos expériences ont d'ailleurs montré que ATNoSFERES était capable de se comparer, voire de rivaliser, en terme de résultats, avec les meilleurs approches des systèmes de classeurs gérant un registre (*cf.* section 4.3). La résolution du problème qui a permis de les comparer reposait justement sur la finesse de réglage et d'utilisation des états internes.

5.3.7 Biais limité des opérateurs de recherche

Tout opérateur génétique étant indépendant de la structure construite, l'utilisation de notre modèle simple et uniforme d'interprétation nous assure de devoir introduire un minimum de biais dans les opérateurs de recherche. L'exploration de l'espace des structures peut se faire avec des opérateurs classiques, génériques ou plus spécifiques (comme l'insertion/délétion de paquets de bits).

Il est cependant nécessaire de déterminer le code génétique – qui influence les probabilités d'occurrence de chaque lexème – et les atomes du langage utilisés pour construire la structure. Nos expériences ont montré, par exemple,

que l'ajout d'un lexème permettant de créer plus facilement des arcs bouclés améliorait sensiblement les résultats sur un problème où cette capacité procure un avantage adaptatif (*cf.* section 4.3).

5.4 L'Éthogénétique

5.4.1 L'Éthogénétique comme principes de génétique d'agent

Les principes sélectionnistes que nous proposons tentent de fournir un cadre aux besoins spécifiques de la conception automatique de comportements d'agents par des méthodes évolutionnistes. Les propriétés que nous avons identifiées puis érigées en principes de l'Éthogénétique se concentrent sur des aspects propres à la construction par sélection darwinienne de structures, dans la perspective de les employer comme architectures d'agent. Le nouveau principe de codage génétique que nous proposons en vue de satisfaire ces principes permet de plus de délier le substrat génétique de contraintes propres à la structure produite.

C'est donc une réponse adaptée à ce qu'Alain CARDON identifie comme le problème de la *génétique d'agents* dans (Cardon, 1999), l'un des problèmes à résoudre pour pouvoir, selon lui, concevoir des systèmes informatiques dotés d'une forme de « conscience artificielle ». Le cadre est celui de systèmes multi-agents adaptatifs fortement dynamiques, où certaines sous-parties observent les activités et les formes organisationnelles manifestées par d'autres pour en donner une représentation que celles-ci prendront en retour en compte dans le propre activité. L'évolution de tels systèmes, à couplages forts, nécessiterait d'être pour partie construite par un processus sélectionniste s'appuyant sur la reproduction et la mort d'agent. Le problème de la génétique d'agent est dans ces conditions de trouver un support héréditaire pour construire

des structures, à partir de groupes d'agents de nature différente. Il est donc nécessaire de pouvoir donner un support de représentation génétique commun à des agents dont les structures peuvent être de nature différente. Les modèles dérivés du principe de codage génétique que nous proposons s'appuient justement sur une représentation très commune en informatique (la chaîne de bits), et permettent de traduire n'importe quelle chaîne en une structure, pourvu que l'agent dispose d'un langage de construction vérifiant les propriétés que nous avons énoncées.

5.4.2 Conséquences méthodologiques

L'organisation comme observable plutôt que comme hypothèse

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 (Cardon, 1999), dont des représentations constitueraient des formes qui viendraient enrichir les informations dont on dispose sur le système et dont il dispose de lui-même (Cardon, 2003).

Envisager les systèmes multi-agents dans le cadre de la sélection darwinienne ne revient pas juste à employer une méthode d'apprentissage automatique de plus. L'une des principales conséquences est d'ordre méthodologique

(Picault, 2001). Avec ce point de vue ascendant, l'organisation du système est alors vue comme une propriété émergente (Picault and Landau, 2001a), dont les variations ou la stabilité donnent des informations sur le système, sur son équilibre (coordinations, coopérations entre agents, compétition pour des ressources) et non plus, comme c'est souvent le cas pour les approches descendantes en intelligence artificielle distribuée, une donnée statique du système qui limite la taille et la complexité des systèmes que l'on peut concevoir. La spécification explicite de l'organisation s'accompagne souvent du besoin de faire coïncider l'évolution de certains attributs du système avec un modèle qui lui est extérieur, que l'on voudrait mieux comprendre ou sur lequel nous voudrions pouvoir faire des prédictions grâce au système. Or, en spécifiant a priori l'organisation, le système est déjà très contraint. Soit il convient pour le problème posé, soit ce n'est pas le cas et il faut repenser toute l'organisation. Avec une perspective ascendante il est possible de procéder autrement, en laissant toute liberté de manœuvre au système pour se réorganiser (Werner, 1992). Il s'agit de concevoir d'une façon différente, en laissant des organisations se faire et se défaire. L'utilisation de génétique d'agent permet encore plus de souplesse, puisqu'elle permettent de concevoir des parties du système qui observent les organisations et les modifient en retour en modifiant structurellement les comportements des agents impliqués.

La dualité organisme-écosystème

Cette conception ascendante peut reposer sur une conception duale du système multi-agent, autorisant un cycle de conception original, selon que l'on perçoit le système comme un tout ou comme un ensemble d'entités. La première vue, celle d'un système en développement, correspond à celle d'un organisme. Le concepteur sait, au moins qualitativement, ce qu'il at-

tend en partie du système, et appliquera ses connaissances en les modélisant comme des pressions sélectives à appliquer à l'ensemble du système, comme nous l'avons fait par exemple pour ATNoCells avec les renforcements liés à la satisfaction du tuteur. Mais ces connaissances sont insuffisantes pour complètement spécifier le système, dont nous savons qu'il est constitué de parties en interactions persistantes, ce qui le rend difficile à concevoir car instable par nature, indépendamment même de son environnement changeant. En effet, dès qu'il a plus de deux sous-systèmes en interaction persistantes, le système qu'ils composent est mathématiquement *non intégrable* (Poincaré, 1893), et sa trajectoire est chaotique (Prigogine, 1994; Prigogine, 1996). Il nous faut donc alors nous intéresser à la seconde vue, celle d'un ensemble d'entités en interaction. Cette seconde vue est celle d'un écosystème, où nous pouvons décrire des interactions entre les agents. Ceci est rendu possible même si nous ne savons pas quelle forme prendra leur organisation – qui pourra changer – dès lors que ces comportements peuvent évoluer. Cette modélisation des interactions et contraintes locales de sous-systèmes va en retour modifier le comportement global du système, dont nous savons qualitativement ce que nous en attendons. Nous définissons ainsi un cycle de conception s'appuyant sur un processus sélectionniste. 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 (Picault, 2001). L'Éthogénétique a été forgée avec l'espoir de donner un cadre théorique sélectionniste satisfaisant pour ce cycle de conception.

Conclusion

Pour concevoir des systèmes multi-agents à l'aide de méthodes inspirées de sélection darwinienne, les insuffisances des approches existantes nous ont amené à établir le cadre de l'Éthogénétique, et à créer un nouveau type de codage génétique de structures qui le respecte. Ce dernier est basé sur l'interprétation d'une chaîne de bits par un interprète à pile pour construire une structure. Notre modèle, ATNoSFERES, instancie ce codage pour des ATN (des graphes orientés et étiquetés) utilisés comme des automates non-déterministes décrivant des comportements d'agents. Des expériences viennent valider, dans un premier temps, l'utilisation d'ATNoSFERES comme support pour l'évolution de comportements d'agents solitaires et en simulation, et nous ont permis d'observer des propriétés intéressantes du modèle, notamment le fait qu'il n'est pas nécessaire de contraindre la taille de la chaîne de bits, et que le système privilégie la convergence vers de solutions simples et de complexité adaptée à la tâche à résoudre. Dans un deuxième temps, des expériences sur un problème non-markovien difficile montrent que ATNoSFERES peut, en terme de qualité des solutions trouvées, rivaliser avec les meilleurs systèmes de classeurs à états. Dans un troisième et dernier temps, ATNoSFERES est employé dans le modèle ATNoCells. ATNoCells est un modèle de système multi-agent en développement qui contrôle un robot. Il a été conçu pour permettre d'enseigner une tâche à un robot par démons-

tration, sans que le tuteur ait besoin de connaître la robotique, et sans que le robot ait connaissance préalable d'un modèle de l'environnement ni de la tâche à exécuter. Nous n'avons pas eu le temps d'exécuter toutes les expériences voulues avec ATNoCells. Ce modèle demeure néanmoins prometteur, bien que souffrant de certaines insuffisances que nous avons identifiées comme étant un manque de *meta*, que nous modéliserions dans ce contexte multi-agent par de l'auto-observation du système par une partie de lui-même pour le réguler.

Nous comptons poursuivre l'étude d'ATNoSFERES dans le cadre des algorithmes évolutionnistes, notamment ce qui a trait à la possibilité de s'en servir comme outil emprique de mesure de complexité d'un problème non-markovien. Par ailleurs, à l'aide d'outils d'analyse adaptés, nous étudierons plus précisément la dynamique sélectionniste à l'œuvre dans ATNoCells, tout en poursuivant les expérimentations avec des robots.

Bibliographie

Agre, P. E. (1988). *The Dynamic Structure of Everyday Life*. PhD thesis, MIT Artificial Intelligence Laboratory.

Agre, P. E. and Chapman, D. (1990). *What Are Plans for?*, pages 17–34. In (Maes, 1990).

Angeline, P., Saunders, G., and Pollack, J. (1994a). Complete Induction of Recurrent Neural Networks. In (Sebald and Fogel, 1994), pages 1–8.

Angeline, P. J. (1994a). Genetic Programming: A Current Snapshot. In (Sebald and Fogel, 1994), pages 224–232.

Angeline, P. J. (1994b). Genetic Programming and Emergent Intelligence. In Kinnear, Jr., K. E., editor, *Advances in Genetic Programming*, pages 75–98. MIT Press.

Angeline, P. J. (1995). Morphogenic Evolutionary Computations: Introduction, Issues and Example. In *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pages 387–401, Cambridge, MA. A Bradford Book, MIT Press.

Angeline, P. J. (1996). An Investigation into the Sensitivity of Genetic Programming to the Frequency of Leaf Selection During Subtree Crossover. In (Koza et al., 1996b), pages 21–29.

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

Angeline, P. J., Saunders, G. M., and Pollack, J. P. (1994b). An Evolutionary Algorithm that Constructs Recurrent Neural Networks. *IEEE Transactions on Neural Networks*, 5(1):54–65.

Arkin, R. C. (1987). Motor Schema Based Navigation for a Mobile Robot: An Approach to Programming by Behavior. pages 264–271.

Arkin, R. C. (1998). *Behavior-Based Robotics*. MIT Press, Cambridge, MA.

Bäck, T. and Schwefel, H.-P. (1993). An Overview of Evolutionary Algorithms for Parameter Optimization. *Evolutionary Computation*, 1(1):1–23.

Beasley, D., Bull, D. R., and Martin, R. R. (1993). An overview of genetic algorithms: Part 1, fundamentals. *University Computing*, 15(2):58–69.

Boers, E. J. and Kuiper, H. (1992). Biological Metaphors and the Design of Modular Artificial Neural Networks. Master's thesis, Dep. CS and Exp. and theor. psych., Leiden University, Netherlands.

Bouchon-Meunier, B. (1994). *La logique floue*. Que sais-je ? Presses Universitaires de France, Paris.

Brainard, D. H. and Freeman, W. T. (1997). Bayesian color consistancy. *Journal of the Optical Society of America*, 7(14):1393–1411.

Braitenberg, V. (1986). *Vehicles : Experiments in Synthetic Psychology*. MIT Press.

Bremermann, H. J. (1962). Optimization through Evolution and Recombination. In Yovits, Jacobi, and Goldstein, editors, *Self organizing Systems*, pages 93–106. Pergamon Press, Oxford.

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

Brooks, R. A. (1990). *Elephants Don't Play Chess*, pages 3–15. In (Maes, 1990).

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

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

Brooks, R. A. (1999). *Cambrian Intelligence.* MIT Press, Cambridge, MA.

Brooks, R. A. and Maes, P., editors (1994). *Proc. of the Artificial Life IV Conference*, volume 4, Cambridge, MA. MIT Press.

Cardon, A. (1998). Modélisation des Systèmes Adaptatifs par Agents : vers une Analyse-Conception Orientée Agents. Technical report lip6 1998/011, LIP6, Paris.

Cardon, A. (1999). *Conscience artificielle & systèmes adaptatifs.* Eyrolles, Paris.

Cardon, A. (2001). The Approaches of the Concept of Embodiment for an Autonomous Robot. Towards Consciousness of its Body. In (Mařík et al., 2001), pages 218–229.

Cardon, A. (2003). *Modéliser et concevoir une machine pensante.* Automates Intelligents édition. (à paraître).

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

Cardon, A. and Vacher, J.-P. (2000). Genetic Algorithm using Multi-Objective in a Multi-Agent System. In *Robotics and Autonomous Sys-*

tems, number 33, pages 179–190.

Chatila, R. and Laumont, J. (1985). Position referencing and consistent world modelling for mobile robots. pages 138–145.

Chomsky, N. (1962). Context-free grammars and pushdown storage. Quarterly Progress Report 65, MIT Research Laboratory in Electronics.

Chomsky, N. and Miller, G. (1963). *Handbook of Mathematical Psychology 2*, chapter Introduction to the Formal Analysis of Natural Languages, pages 269–321. Wiley and Sons, New York.

Connell, J. H. (1989). *A Colony Architecture for an Artificial Creature*. Thèse de Doctorat, MIT AI Lab.

Crawford-Marks, R. and Spector, L. (2002). Size control via size fair genetic operators in the PushGP genetic programming system. In (Langdon et al., 2002), pages 733–739.

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

Dautenhahn, K. (1999). Embodiment and Interaction in Socially Intelligent Life-Like Agents. In (Nehaniv, 1999), pages 105–147.

Dautenhahn, K., editor (2000). *Socially Intelligent Agents: the Human in the Loop (SIA'2000)*, volume FS–00–04 of *AAAI Fall Symposium Series*, Menlo Park, CA. AAAI Press.

Dawkins, R. (1976). *The Selfish Gene*. Oxford University Press, first edition.

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

Demazeau, Y. and Werner, E., editors (1992). *Decentralized AI 3*, Amsterdam. Elsevier (North-Holland).

Dennett, D. C. (1995). *Darwin's Dangerous Idea. Evolution and the Meanings of Life.* Simon and Schuster, New York.

Drogoul, A. (1993). *De la Simulation Multi-Agents à la Résolution Collective de Problèmes.* Thèse de Doctorat, Université Paris VI.

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

Drogoul, A. and Dubreuil, C. (1992). Eco-Problem-Solving Model: Results of the N-Puzzle. In (Demazeau and Werner, 1992), pages 283–295.

Drogoul, A. and Picault, S. (1999). MICRobES : vers des collectivités de robots socialement situées. In (Gleizes and Marcenac, 1999).

Drummond, M. E. (1989). Situated control rules. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto. Morgan Kaufmann.

Edelman, G. M. (1992). *Biologie de la conscience.* Odile Jacob.

Ferber, J. (1995). *Les systèmes multi-agents : Vers une intelligence collective.* InterEditions, Paris.

Fikes, R. E. and Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 5(2):189–208.

Filliat, D. (2001). *Cartographie et estimation globale de la position pour un robot mobile autonome.* Thèse de Doctorat, Université Paris VI.

Floreano, D. and Urzelai, J. (2000a). *Evolutionary Robotics III*, chapter Evolutionary Robotics: The Next Generation. AAI Books, Ontario.

Floreano, D. and Urzelai, J. (2000b). Evolutionary Robots with on-line self-organization and behavioral fitness. *Neural Networks*, 13:431–443.

Fogel, D. B. (1992). *Evolving Artificial Intelligence.* Thèse de Doctorat, University of California, San Diego.

Fogel, D. B. and Stayton, L. C. (1994). On the Effectiveness of Crossover in Simulated Evolutionary Optimization. In *Biosystems*, number 32, pages 171–182.

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

Ghallab, M. (1999). *Cerveaux et machines*, chapter L'action sensorielle en robotique (18). Sciences. Hermes, Paris.

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

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley.

Gérard, P. (2002). *Systèmes de Classeurs : étude de l'apprentissage latent.* Thèse de Doctorat, Université Paris VI.

Greenwood, G. W. (1997). Training Partially Recurrent Neural Networks Using Evolutionary Strategies. In *IEEE Trans. on Speech and Audio Processing*, volume 5, pages 192–104.

Gruau, F. (1994). *Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm.* Thèse de Doctorat, ENS Lyon – Université Lyon I.

Guillot, A. and Meyer, J.-A. (2000). Chaotic dynamics underlying action selection in mice. *Nonlinear Dynamics, Psychology and Life Sciences*, 4(4).

Gutknecht, O. and Ferber, J. (1997). Madkit: Organizing heterogeneity with groups in a platform for multiple multi-agent systems. Technical Report RR97188, LIRMM.

Harnad, S. (1990). The Symbol Grounding Problem. In *Physica D*, volume 42, pages 335–346.

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

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

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

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

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

Holland, J. H. (1975b). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* University of Michigan Press, Ann Arbor, MI.

Holland, J. H. (1996). *Hidden Order: How Adaptation Builds Complexity.* Addison-Wesley.

Hugues, L. (2002). *Apprentissage de comportements pour un robot autonome.* Thèse de Doctorat, Université Paris VI.

Kaebling, L. P. (1987). *Reasoning About Actions and Plans*, chapter An Architecture for Intelligent Reactive Systems, pages 395–410. Morgan Kaufmann, Palo Alto, CA.

Kaebling, L. P. (1990). *Learning in embedded systems.* Thèse de Doctorat, Stanford University.

Kitano, H. (1990). Designing Neural Networks Using Genetic Algorithms with Graph Generation System. In *Complex Systems*, volume 4, pages

461–476.

Kodjabachian, J. (1998). *Développement et évolution de réseaux de neurones artificiels : Application au contrôle d'un animat hexapode.* Thèse de Doctorat, Université Paris VI.

Kodjabachian, J. and Meyer, J.-A. (1998). Evolution and Development of Neural Controllers for Locomotion, Gradient-Following, and Obstacle-Avoidance in Artificial Insects. *IEEE Transactions on Neural Networks*, 9:796–812.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA.

Koza, J. R., Bennett III, F. H., Andre, D., and Keane, M. A. (1996a). Automated Design of Both the Topology and Sizing of Analog Electrical Circuits Using Genetic Programming. In Gero, J. S. and Sudweeks, F., editors, *Artificial Intelligence in Design'96*, pages 151–170.

Koza, J. R., Goldberg, D. E., Fogel, D. B., and Riolo, R. L., editors (1996b). *Genetic Programming 1996: Proceedings of the First Annual Conference*, Stanford University, CA. MIT Press.

Kupiec, J.-J. and Sonigo, P. (2000). *Ni Dieu ni gène. Pour une autre théorie de l'hérédité.* Science ouverte. Éditions du Seuil, Paris.

Landau, S., Doncieux, S., Drogoul, A., and Meyer, J.-A. (2001a). SFERES. Rapport technique 2001/011, LIP6, Paris.

Landau, S., Doncieux, S., Drogoul, A., and Meyer, J.-A. (2002a). SFERES: un framework pour la conception de systèmes multi-agents adaptatifs. *Technique et Science Informatiques*, 21:427–446.

Landau, S. and Picault, S. (2002a). Modeling Adaptive Multi-Agent Systems Inspired by Developmental Biology. In (Mařík et al., 2001), pages 238–246.

Landau, S. and Picault, S. (2002b). Stack-Based Gene Expression. Rapport Technique LIP6 2002/011, LIP6, Paris.

Landau, S., Picault, S., and Drogoul, A. (2001b). ATNoSFERES: a Model for Evolutive Agent Behaviors. In *Proceedings of the AISB'01 Symposium on Adaptive Agents and Multi-Agent Systems.*

Landau, S., Picault, S., Sigaud, O., and Gérard, P. (2002b). A comparison between ATNoSFERES and XCSM. In Langdon, W. B., Cantú-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M. A., Schultz, A. C., Miller, J. F., Burke, E., and Jonoska, N., editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 926–933, New York. Morgan Kaufmann Publishers.

Landau, S., Picault, S., Sigaud, O., and Gérard, P. (2002c). Further Comparison between ATNoSFERES and XCSM. In Stolzmann, W. et al., editors, *IWLCS-02. Proceedings of the Fifth International Workshop on Learning Classifier Systems*, LNAI, Granada. Springer.

Langdon, W. B., Cantú-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M. A., Schultz, A. C., Miller, J. F., Burke, E., and Jonoska, N., editors (2002). *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, New York. Morgan Kaufmann Publishers.

Langton, C., editor (1988). *Artificial Life*, volume 1, London. Addison-Wesley.

Lanzi, P. L. (1998). An Analysis of the Memory Mechanism of XCSM. In *Proceedings of the Third Genetic Programming Conference.*

Lanzi, P. L. and Wilson, S. W. (2000). Toward optimal classifier system performance in non-markov environments. *Evolutionary Computation*,

8(4):393–418.

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

Luke, S. and Spector, L. (1996). Evolving Graphs and Networks with Edge Encoding: Preliminary Report. In (Koza et al., 1996b), pages 117–124.

Maes, P., editor (1990). *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*. MIT Press, Cambridge, MA.

Maes, P. and Brooks, R. A. (1990). Learning to coordinate behaviors. In *Proc. of AAAI-90*, volume 2, pages 796–802, Boston, MA. AAAI Press/The MIT Press.

Maes, P., Matarić, M. J., Meyer, J.-A., Pollack, J., and Wilson, S. W., editors (1996). *From Animals to Animats 4. Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour (SAB'96)*, Cambridge, MA. MIT Press.

Mahadevan, S. and Connell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, (55):311–365.

Mařík, V., Štěpánková, O., Krautwurmová, H., and Briot, J.-P., editors (2001). *Proceedings of the Workshop on Adaptability and Embodiment Using Multi-Agent Systems (AEMAS'2001)*, Prague. Czech Technical University.

Meyer, J.-A. and Wilson, S. W., editors (1991). *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*. MIT Press.

Miller, G. F., Todd, P. M., and Hegde, S. U. (1989). Designing Neural Networks using Genetic Algorithms. In Schaffer, J. D., editor, *Proceedings of*

the Third International Conference on Genetic Algorithms, pages 65–80. Morgan Kaufmann.

Minsky, M. L. (1963). *Computers and Thought*, chapter Steps Toward Artificial Intelligence. McGraw-Hill, New-York.

Mitchell, M. (1996). *An introduction to genetic algorithms.* MIT Press, Cambridge.

Mitchell, M. and Taylor, C. E. (1999). Evolutionary Computation: An Overview. *Annual Review of Ecology and Systematics*, 20:593–616.

Mitchell, T. (1997). *Machine Learning.* McGraw Hill.

Montana, D. J. (1995). Strongly Typed Genetic Programming. In *Evolutionary Computation*, volume 3.

Moore, C. H. and Leach, G. C. (1970). FORTH - A Language for Interactive Computing. internal publication, Mohasco Industries, Amsterdam NY.

Müller, J. and Pischel, M. (1993). The Agent Architecture InteRRaP: Concept and Application. Technical report rr-93-26, DFKI Saarbrucken.

Nehaniv, C. L., editor (1999). *Computation for Metaphors, Analogy and Agent*, volume 1562 of *Lectures Notes in Artificial Intelligence.* Springer Verlag.

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

Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics. The Biology, Intelligence, and Technology of Self-organizing Machines.* MIT Press, Cambridge, MA.

Nolfi, S., Floreano, D., Miglino, O., and Mondada, F. (1994). How To Evolve Autonomous Robots: Different Approaches In Evolutionary Robotics. In (Brooks and Maes, 1994).

Parker, L. E., editor (2000). *Proceedings of the International Conference on Distributed Autonomous Robotic Systems (DARS'2000).*

Perkis, T. (1994). Stack-Based Genetic Programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 148–153, Orlando, FL. IEEE Press.

Picault, S. (2001). *Modèles de comportements sociaux pour les collectivités d'agents et de robots.* Thèse de Doctorat, Université Paris VI.

Picault, S. and Drogoul, A. (2000a). The MICRobES Project, an Experimental Approach towards "Open Collective Robotics". In (Parker, 2000).

Picault, S. and Drogoul, A. (2000b). Robots as a Social Species: the MICRobES Project. In (Dautenhahn, 2000), pages 139–141.

Picault, S. and Landau, S. (2001a). Ethogenetics: an Evolutionary Approach to Agents Organization. In *Actes du Colloque ALCAA (Agents Logiciels, Coopération, Apprentissage et Activité Humaine.* IUT de Bayonne, Université de Pau.

Picault, S. and Landau, S. (2001b). Ethogenetics and the Evolutionary Design of Agent Behaviors. In Callaos, N., Esquivel, S., and Burge, J., editors, *Proceedings of the 5th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI'01)*, volume III, pages 528–533.

Picault, S., Servat, D., and Kaplan, F. (1997). EDEN : un système évolutif endosémantique. Technical report, École Nationale Supérieure des Télécommunications, Paris.

Pitrat, J. (1985). *Textes, ordinateurs et compréhension.* Eyrolles, Paris.

Pitrat, J. (1990). *Métaconnaissance, futur de l'intelligence artificielle.* Hermès, Paris.

Pitrat, J. (1995). Speaking about and acting upon oneself. Rapport technique laforia 95/29, LAFORIA.

Poincaré, H. (1893). *Les méthodes nouvelles de la mécanique céleste.* Gauthier-Villars, Paris.

Prigogine, I. (1994). *Les lois du chaos.* Flammarion, Paris.

Prigogine, I. (1996). *La fin des certitudes.* Odile Jacob, Paris.

Prigogine, I., editor (2001). *L'homme devant l'incertain.* Odile Jacob, Paris.

Ram, A., Arkin, R., Boone, G., and Pearce, M. (1994). Using genetic algorithms to learn reactive control parameters for autonomous robotic navigation. *Adaptive Behavior*, 2(3):277–304.

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

Rosenblatt, F. (1962). *Principles of Neurodynamics: Percepttons and the Theory of Brain Mechanisms.* Spartan, Washington D.C.

Rozenberg, G. and Salomaa, A., editors (1974). *L Systems, Most of the papers were presented at a conference in Aarhus, Denmark, January 14-25, 1974*, volume 15 of *Lecture Notes in Computer Science.* Springer.

Savitch, W. J. (1975). Some characterizations of Lindenmayer systems in terms of Chomsky-type grammars and stack machines. *Information and Control*, 27(1):37–60.

Schmid, H. A. (1997). Systematic Framework Design by Generalization. *Communications of the ACM*, 40(10):48–51.

Schoenauer, M. (1997). *Evolutionary Computation and Applications at Centre de Mathématiques Appliquées de l'École Polytechnique.* Dossier d'habilitation à diriger les recherches, Université Paris XI – École Polytechnique, Orsay.

Schwefel, H.-P. (1975). *Evolutionsstrategie und numerische Optimierung.* Dr.-Ing. Thesis, Technical University of Berlin, Department of Process En-

gineering.

Schwefel, H.-P. (1995). *Evolution and Optimum Seeking.* John Wiley and Sons, Inc.

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

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

Spector, L. (2001). Autoconstructive evolution: Push, pushGP, and pushpop. In Spector, L., Goodman, E. D., Wu, A., Langdon, W. B., Voigt, H.-M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M. H., and Burke, E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 137–146, San Francisco, California, USA. Morgan Kaufmann.

Spector, L. and Robinson, A. (2002a). Genetic Programming and Autoconstructive Evolution with the Push Programming Language. *Genetic Programming and Evolvable Machines*, 3(1):7–40.

Spector, L. and Robinson, A. (2002b). Multi-type, self-adaptive genetic programming as an agent creation tool. In *Proceedings of the Workshop on Evolutionary Computation for Multi-Agent Systems (ECOMAS 2002).*

Spector, L. and Stoffel, K. (1996a). Automatic generation of adaptive programs. In (Maes et al., 1996).

Spector, L. and Stoffel, K. (1996b). Ontogenetic programming. In (Koza et al., 1996b), pages 394–399.

Stern, J. (1990). *Fondements mathématiques de l'informatique.* McGraw-Hill.

Stoffel, K. and Spector, L. (1996). High-performance, parallel, stack-based genetic programming. In (Koza et al., 1996b), pages 224–229.

Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning.* Thèse de Doctorat, Department of Computer and Information Science, University of Massachusetts.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning, an introduction.* MIT Press, Cambridge, MA.

Systems, A. (1985). *The PostScript Language Reference Manual.* Addison-Wesley, Reading MA.

Thomas, R. and Kaufman, M. (2001). chapter Émergence de comportements complexes à partir de circuits simples. In (Prigogine, 2001).

Tomlinson, A. and Bull, L. (2000). CXCS. In Lanzi, P., Stolzmann, W., and Wilson, S., editors, *Learning Classifier Systems: from Foundations to Applications*, pages 194–208. Springer Verlag, Heidelberg.

Walker, A. (1974). Adult Languages of $\mathcal{L}$-Systems and the Chomsky Hierarchy. In (Rozenberg and Salomaa, 1974), pages 201–215.

Wall, M. (2000). Galib. `http://lancet.mit.edu/ga/`.

Werner, E. (1992). The Design of Multi-Agent Systems. In (Demazeau and Werner, 1992), pages 3–28.

Whitley, D. (1989). the genitor genetic algorithm. `http://www.cs.colostate.edu/~genitor/`.

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

Whitley, D., Starkweather, T., and Bogart, C. (1990). Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity. In *Parallel Computing*, volume 3, pages 347–361.

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

Wilson, S. W. (1991). The Animat Path to AI. In (Meyer and Wilson, 1991), pages 15–21.

Wilson, S. W. (1995). Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175.

Winograd, T. and Flores, F. (1986). *Understanding computers and cognition: a new foundation for design.* Ablex, Norwood, New Jersey.

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

Yao, X. (1999). Evolving Artificial Neural Networks. *Proceedings of the IEEE*, 87.