UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN MATHÉMATIQUES ET INFORMATIQUE
APPLIQUÉES

PAR
ÉRIC PELLERIN

Méta-apprentissage des algorithmes génétiques

DÉCEMBRE 2005

Résumé

La conception d'un système de méta-apprentissage représente un enjeu important dans le domaine de l'apprentissage machine (*Machine learning*). Le défi est tout aussi important pour les applications de la Défense Nationale surtout utilisées dans le cadre des opérations urbaines, car la robustesse requise par de tels systèmes n'est pas disponible aujourd'hui. Ceci prend toute son importance car le théâtre urbain devient, avec les années, une zone privilégiée de conflits de natures diverses. Cet environnement renferme une multitude de renseignements trop complexes pour être traités sans l'aide d'un système d'apprentissage. La surcharge des informations disponibles aux utilisateurs de systèmes augmente ainsi l'incapacité de ceux-ci à assimiler et à exploiter adéquatement l'information. Cette incapacité a pour effet une réduction de l'efficacité d'actions stratégiques et tactiques des opérations militaires. Le succès des opérations sera directement proportionnel à la rapidité et à l'efficacité avec laquelle la connaissance de la situation de l'ennemi et du milieu urbain sera fournie au commandant.

L'application du méta-apprentissage (MA) dans ces situations complexes du monde réel exige une adaptation continue à un environnement dynamique. Cette adaptation est réalisée grâce aux algorithmes génétiques (AG) qui se sont montrés favorables pour la conception d'un tel système. Un système adaptatif d'optimisation des connaissances en lien direct avec l'environnement est donc préconisé à l'aide d'algorithmes génétiques. Pour garantir le succès d'un tel prototype d'apprentissage, le concept de MA doit être examiné rigoureusement, ce qui implique alors deux possibilités de MA : l'une centrée sur l'AG lui-même, selon sa capacité de maîtriser son propre fonctionnement par MA et la seconde possibilité, par l'utilisation des AG pour réaliser le MA. Dans ce cas l'apprentissage est centré sur les expériences acquises du système en accumulant des méta-connaissances. Ces deux possibilités ont conduit au développement des prototypes suivants :

1. Prototype **APAG** pour **A**uto-adaptation des **P**aramètres de l'**A**lgorithme **G**énétique,

2. Prototype **MAAG** pour **M**éta-**A**pprentissage des **A**lgorithmes **G**énétiques.

Le prototype APAG est essentiellement basé sur le concept de l'autonomie. L'autonomie se reflète par une architecture spéciale au niveau de l'individu de l'algorithme génétique par rapport aux individus des AG traditionnels.

Le prototype MAAG se caractérise par son aspect modulatoire avec une orientation importante sur la notion de méta-connaissances. Le module principal de MAAG est le module *Darwin Brain*, qui renferme le cœur du système d'apprentissage. L'apprentissage est basé sur l'évolution par algorithme génétique, mais aussi en combinaison avec le darwinisme neuronal. Le darwinisme apporte la partie structurale de l'apprentissage qui peut se comparer aux neurones du cerveau humain.

La capacité de l'AG à maîtriser son propre paramétrage est démontrée par les expérimentations effectuées dans ce travail. Les résultats indiquent que les AG qui possèdent un mécanisme d'auto-adaptation ont une performance supérieure par rapport aux AG traditionnels. Le choix du

meilleur jeu de paramètres est le résultat d'une combinaison des paramètres avec d'autres paramètres et ce, en étroite interaction avec le problème et le mécanisme d'évolution des AG.

L'expérimentation préliminaire sur l'apprentissage de vecteurs de renseignement donne des résultats prometteurs et met en perspective la capacité d'apprentissage des algorithmes génétiques. Les résultats d'expérimentation permettent de valider la capacité des AG comme outils d'apprentissage. À la lumière des résultats, l'élaboration d'un système de méta-apprentissage à l'aide des algorithmes génétiques ouvre une voie intéressante pour atteindre le niveau requis de robustesse dans le développement de futurs systèmes militaires appliqués aux opérations urbaines.

Abstract

The design of an efficient system of meta-learning represents a great challenge in the field of machine learning. The challenge is quite as important for the applications of National Defence, especially those used within the framework of the urban operations, because the robustness required by such systems is not available today. This takes all its importance because the urban theatre becomes, with the years, a privileged zone of conflicts of various natures. This environment contains a multitude of too complex information to be treated without the assistance of a learning system. The information overload available to the users of systems increases their incapacity to assimilate the information and to exploit it adequately. This incapacity causes a reduction of the effectiveness of strategic and tactical actions in the military operations. The success of the operations will be directly proportional to the speed and the effectiveness with which the knowledge of the situation of the enemy and the urban environment will be provided to the commander.

The application of meta-learning in these complex situations of the real world requires a continuous adaptation to a dynamic environment. In the adaptation context, to help development of meta-learning algorithms, we suggest to use genetics algorithms like an adaptive system. An adaptive system of optimization of knowledge in direct interaction with the environment is thus recommended using genetic algorithms. To guarantee the success of such a prototype of learning, the concept of meta-learning must then be examined rigorously, which implies two possibilities of meta-learning: one centered on GA itself, according to its capacity to control its own operation by meta-learning and the second possibility, the use of GA's to carry out meta-learning. In this case, the learning is centered on the gained experiences of the system by accumulating meta-knowledge. These two possibilities led to the development of the following prototypes:

1. Prototype **APAG** from French "**A**uto-adaptation des **P**aramètres de l'**A**lgorithme **G**énétique",

2. Prototype **MAAG** from French " **M**éta-**A**pprentissage des **A**lgorithmes **G**énétiques".

Prototype APAG is primarily based on the concept of autonomy. Autonomy is reflected by a special architecture of the individual of the genetic algorithm compared to the traditional individuals of GA's.

Prototype MAAG is characterized by its modulator aspect, with an important orientation on the concept of meta-knowledge. The principal module of MAAG is the *Darwin Brain* module, which contains a center of gravity of the learning system. The learning is based on the evolution by genetic algorithm, but also in combination with the neuronal Darwinism. The Darwinism brings the structural part of the learning, which can be compared with the neurons of the human brain. The capacity of GA to control its own parameter setting is shown by the experiments carried out in this work. The results indicate that GAs that have a mechanism of self-adaptation have a higher performance than traditional GAs.

The choice of the best set of parameters is the result of a combination of the parameters with other parameters and this, in close interaction with the problem and the mechanism of evolution of GAs. The preliminary experimentation on the learning of vectors of information gives promising results and brings out in prospect the capacity of learning related to the genetic algorithms. The results of experimentation validate the capacity of GAs to be tools of learning. According to the results, the development of a system of meta-learning using the genetic algorithms opens an interesting way to reach the necessary level of robustness in the development of future military systems applied to urban operations.

Remerciements

Je désire remercier mon co-directeur Luc Pigeon, scientifique de la défense, section gestion de l'information et des connaissances au RDDC Valcatier, pour sa confiance, pour ses conseils précieux et pour m'avoir permis de réaliser ce travail au sein de la Défense Nationale.

Je remercie également mon directeur de maîtrise, Sylvain Delisle pour sa direction, pour la confiance dont il m'a honoré tout au long de mon projet de maîtrise et pour tout ce qu'il m'a appris.

Je tiens à remercier mes évaluateurs Ofélia Cervantes de l'UDLAP et professeure invitée au DMI ainsi qu'Alain Goupil, professeur au DMI.

Je remercie Chantal Lessard, secrétaire du comité des études de cycles supérieurs au niveau du département de Mathématiques et Informatique, pour sa disponibilité et sa bonne humeur.

De nombreuses autres personnes m'ont aidé dans ce travail, et particulièrement Alain Bergeron pour ses encouragements et ses conseils comme "ombudsman".

Je remercie sincèrement Richard Grenier, responsable de l'unité d'affaire de Québec pour Thales Canada Inc., pour le réel soutien qu'il m'a apporté depuis mon intégration au sein de son équipe. Je remercie par le fait même l'ensemble du personnel de Thales à Québec.

Ma tendre épouse Carmen à qui la clarté du contenu de ce mémoire doit beaucoup. Elle est sans doute l'une des plus soulagées de l'achèvement de ce travail. Mes trois garçons : Louis-David, Rémi et Anthony qui sont une source d'inspiration et de persévérance, je les remercie pour leur patience. Mes deux grandes familles m'ont apporté encouragement et soutien constant, mes beaux-frères tout particulièrement.

Je tiens également à remercier mes amis Tony et Jacques, qui m'ont soutenu de façon non négligeable, soit par leur soutient moral ou par leurs vives discussions autour d'une bière ou d'un orignal.

Au FCAR, pour la bourse en milieu pratique qui m'a permis de poursuivre mes études de maîtrise.

Merci à tous !

Table des Matières

Liste des figures

Liste des tableaux

Liste des abréviations

AG	algorithme génétique
APAG	auto-adaptation des algorithmes génétiques
MA	méta-apprentissage
MAAG	méta-apprentissage des algorithmes génétiques
MC	méta-connaissance
MDG	mesure de la diversité génétique

Chapitre 1
Introduction

Le méta-apprentissage est un concept qui a fait son apparition dans les années'80 et est en fait la capacité d'apprendre à apprendre d'un système. Il est l'un des domaines de recherche en importance dans l'apprentissage machine (*machine learning*). La notion de méta-apprentissage (MA) réfère aux algorithmes de haut niveau qui peuvent apprendre à partir d'algorithmes d'apprentissage. Le MA est complexe à réaliser, mais devient très utile pour la résolution de problèmes complexes. L'application du méta-apprentissage dans des situations complexes du monde réel exige une adaptation continue à de nouveaux besoins. Cette adaptation est le reflet même de notre évolution en tant qu'êtres humains. L'homme a su s'adapter à son environnement pour survivre, et l'adaptation est la principale caractéristique de l'intelligence humaine.

L'apprentissage s'inspire alors des systèmes vivants et de leurs mécanismes spécifiques, pour élaborer de nouveaux modèles conceptuels et pour définir de nouvelles approches plus adaptées aux caractéristiques des systèmes informatiques. L'usage des algorithmes génétiques convient parfaitement pour la réalisation d'un système d'apprentissage car il est un concept basé sur la théorie de l'évolution des espèces et comprend par le fait même des aptitudes d'adaptation très fortes. Dans la conception d'un méta-apprentissage, si le système d'apprentissage échoue à s'exécuter efficacement, on s'attend à ce que le mécanisme s'adapte par lui-même, en tenant compte des expériences précédentes. Pour arriver à ce stade, le système doit avoir une maîtrise parfaite de son propre fonctionnement.

Les systèmes d'apprentissage représentent un enjeu important et ce, pour différents domaines d'application, comme les opérations militaires. En effet, ce projet de maîtrise s'inscrit dans le cadre de travaux effectués pour le RDDC Valcartier. Établi dans la région de Québec depuis plus de soixante ans, le Centre de recherche et Développement pour la Défense Canada (RDDC) est une agence du ministère de la Défense Nationale du Canada. Elle a été créée pour répondre aux besoins scientifiques et technologiques des Forces canadiennes et sa mission est d'assurer la compétence des Forces canadiennes sur les plans opérationnel et scientifique. Le Centre effectue des travaux dans trois domaines dans lesquels son excellence est reconnue : les systèmes optroniques, les systèmes d'information et les systèmes de combat. Des scientifiques et des étudiants de toutes ces disciplines collaborent étroitement à l'exécution de projets de R et D complexes et à l'intégration des technologies. Cette collaboration est essentielle à la mise au point de systèmes modernes et performants et elle est nécessaire pour satisfaire les besoins des armées d'aujourd'hui et pour répondre aux exigences des armées du futur.

Dans le contexte des applications de la Défense Nationale, la robustesse requise par de tels systèmes d'apprentissage n'est pas disponible aujourd'hui. L'utilisation d'un système de méta-apprentissage qui apprend et s'adapte à des situations complexes pour parfaire son apprentissage est un atout considérable. Dans un contexte militaire, des recherches internes tendent à démontrer l'avantage des algorithmes génétiques sur les autres techniques d'apprentissage [Pigeon et *al.*, 2001]. Une recherche sur l'apprentissage des algorithmes génétiques (méta-apprentissage) ouvre une voie intéressante pour atteindre le niveau requis de robustesse dans le développement de futurs systèmes militaires.

Les opérations militaires d'aujourd'hui se déroulent de plus en plus dans ces milieux urbains très complexes. Que se soient des opérations de guerre, humanitaires ou du maintien de la paix, le théâtre de ces opérations est le reflet de l'urbanisation du monde moderne. La complexité des opérations dans une ville se reflète aussi par la gestion des nombreuses sources de renseignements. Le renseignement militaire est l'un des facteurs clefs dans les opérations et forme une base fondamentale aux opérations et une composante importante du processus décisionnel. Le renseignement permet aux commandants de planifier et de diriger les opérations avec succès, d'identifier les objectifs critiques et de gagner des combats décisifs. Le renseignement est la somme de nos connaissances et de notre compréhension de l'environnement dans lequel se déroulent les activités militaires.

La surcharge du renseignement en milieu urbain devient rapidement problématique pour les opérations militaires. Ainsi, les systèmes de commandement et de contrôle associés aux opérations militaires urbaines sont souvent incapables de traiter efficacement l'information et le renseignement en vue de l'aide à la décision. Les militaires doivent composer avec un renseignement qui se fait de plus en plus complexe, volumineux et hétérogène. Le renseignement devient souvent une limitation pour les opérations et réduit significativement l'efficacité d'actions stratégiques et tactiques. Le succès des opérations sera directement proportionnel à la rapidité et à l'efficacité avec laquelle la connaissance de la situation de l'ennemi et du milieu urbain sera fournie au commandant. Le problème de la surcharge des informations et du renseignement peut être pris en charge par un système de méta-apprentissage qui permet d'assimiler et d'exploiter adéquatement l'information.

Les objectifs principaux dans ce travail de recherche découlent de travaux précurseurs réalisés au RDDC Valcartier favorisant l'utilisation des AG comme fondement pour le méta-apprentissage. Les objectifs sont de proposer une démarche pour le MA des algorithmes génétiques et d'appliquer la démarche précédente au secteur du commandement et contrôle dans le cadre des opérations urbaines.

1.1. Les objectifs spécifiques

Dans le développement d'un système d'apprentissage basé sur les algorithmes génétiques, l'apprentissage doit conduire à l'optimisation des relations entre les variables observées. Ces variables ou renseignements proviennent de diverses mesures qui sont prises à partir d'un ensemble de capteurs. Ces capteurs ont des paramètres intrinsèques non modifiables. Chaque renseignement reçoit un poids d'apprentissage afin de déterminer l'importance accordée au capteur. Le traitement de ces données se fait via un objectif d'apprentissage et vise principalement à optimiser un ensemble de poids d'apprentissage afin de valider le meilleur renseignement. Cette optimisation est réalisée grâce aux algorithmes génétiques et elle comporte deux objectifs spécifiques :

1. un apprentissage par AG des poids des renseignements ;

2. un apprentissage des paramètres de l'AG par auto-adaptation.

Pour atteindre les objectifs spécifiques, on doit identifier et définir un individu et son chromosome associé, ce qui correspond à l'unité de base des algorithmes génétiques. Par la suite, on doit établir une population d'individus pour appliquer l'optimisation. Cette optimisation devient à son tour un problème et elle doit être optimisée afin que l'apprentissage soit optimum (méta-apprentissage).

Pour la réalisation des objectifs spécifiques, le développement de prototypes devient indispensable en tant qu'outil de validation. Les objectifs complémentaires sont donc les suivants :

1. Développement d'un prototype de système de méta-apprentissage pour montrer la capacité des algorithmes génétiques d'apprendre à apprendre selon les poids des renseignements ;

2. Développement d'un prototype pour l'amélioration de la performance des algorithmes génétiques par l'apprentissage du réglage des paramètres de l'AG par l'algorithme lui-même (méta-apprentissage).

Ce mémoire repose donc sur la réalisation de deux méthodes de méta-apprentissage bien distinctes. La première méthode fait intervenir un mécanisme d'optimisation de poids d'apprentissage pour conduire à l'acquisition de méta-connaissances. La deuxième méthode passe par la capacité des AG de contrôler leur propre fonctionnement. Une recherche sur l'apprentissage des paramètres de l'algorithme génétique ouvre une voie intéressante pour atteindre le niveau requis de robustesse dans le développement de systèmes intelligents.

Suivant la réalisation des deux méthodes de méta-apprentissage à l'aide des algorithmes génétiques, un troisième objectif spécifique vient en complément dans ce mémoire. Cet objectif de recherche pose un regard plus approfondi sur une approche hypothétique plus près du modèle de la nature. Précisément, l'objectif est d'améliorer notre compréhension des processus naturels d'adaptation et de concevoir des systèmes artificiels possédant des propriétés similaires aux systèmes naturels.

1.2. L'organisation du mémoire

Le contenu du mémoire est réparti comme suit. Le chapitre 2 situe ce projet par rapport au contexte militaire et fait état de la problématique de recherche dans ce même contexte. Le chapitre 3 présente une revue de la littérature sur le méta-apprentissage et les différentes techniques d'apprentissage utilisées. Une attention particulière sera accordée pour décrire le principe des algorithmes génétiques. Le chapitre 4 présente une rétrospective de la classification des différentes approches de paramétrage des algorithmes génétiques ainsi qu'un survol des concepts théoriques qui définissent la méthode de méta-apprentissage envisagée. Ces éléments de littérature constituent les outils essentiels qui serviront à concevoir les prototypes d'apprentissage à l'aide des AG. Le chapitre 5 aborde une description des méthodes proposées dans le chapitre 4. Ce chapitre montre précisément l'architecture, la structure de données ainsi que le fonctionnement des prototypes de méta-apprentissage. Le chapitre 6 contient les modèles expérimentaux utilisés pour valider les objectifs de recherche. Le chapitre 7 présente et analyse les résultats d'implantation d'un système d'auto-adaptation des paramètres de l'algorithme génétique ainsi que les résultats d'un prototype de méta-apprentissage à l'aide des AG. En conclusion, le chapitre 8 résume les principaux résultats obtenus, fait état des limitations des deux approches employées et termine avec une suggestion pour des recherches futures.

Chapitre 2
Problématique

Ce chapitre a pour but de bien situer le projet de recherche dans la réalisation d'un prototypage de système d'apprentissage dans le cadre des opérations urbaines associées au secteur du commandement et contrôle militaires.

Les opérations militaires en milieu urbain ont pris naissance avec l'empire d'Akkad dans l'histoire de la Mésopotamie [Pigeon et Van Chestein, 2004]. Dès 2218 av. J.-C.), une épitaphe du roi Naram-sin (roi d'Akkad de 2254 à 2218 av. J.-C.) fait mention du théâtre urbain comme lieu privilégié de combat. Plusieurs faits historiques relatent aussi l'environnement urbain comme zone privilégiée de conflits de natures diverses. Les villes sont donc un endroit de prédilection pour toute organisation criminelle ou terroriste qui profite d'un ratio attaquant/défendeur pouvant atteindre 27 :1 [U.S. Army, 2000]. Les villes sont un objectif militaire de premier plan, même dans les pays du tiers monde, parce qu'elles regroupent de plus en plus les populations, la richesse, les médias, la politique, les transports, les industries et la culture [Pigeon, 2002a]. Ainsi, une meilleure compréhension de tous ces éléments, qui constituent une ville, permettra de mieux contrôler les opérations en milieu urbain.

2.1. Le milieu urbain

Le milieu urbain pourrait se définir comme un regroupement permanent mais dynamique de populations humaines dans un espace localisé. Selon le nombre d'individus composant la population, cet espace peut s'agrandir dans le temps. Par exemple, en 2003, le nombre d'individus vivant dans les milieux urbains a atteint les 3 milliards et, selon les prévisions (voir figure 2.1), devrait atteindre 5 milliards d'ici 2030.

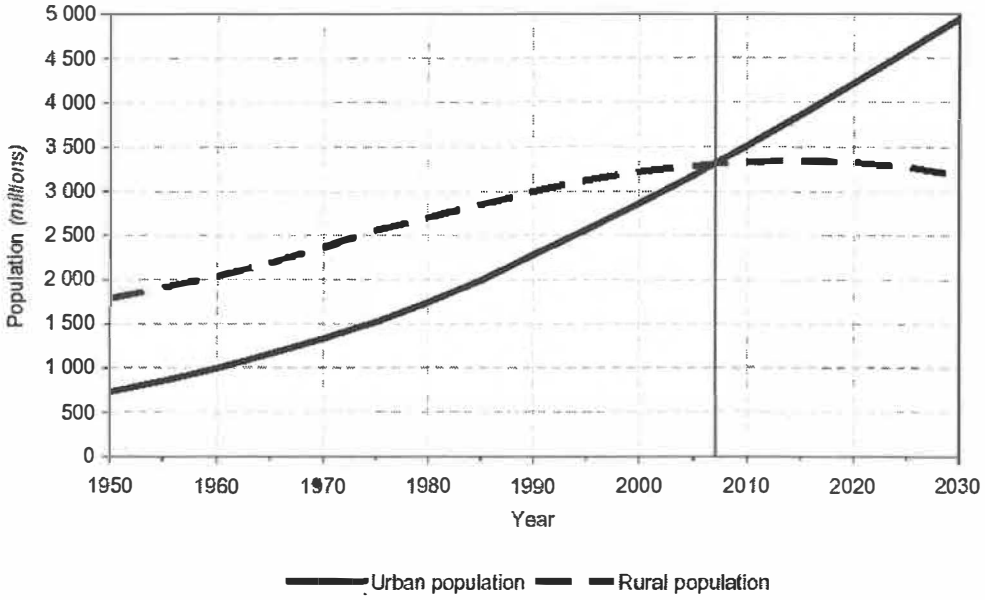


Figure 2.1. Graphique de la croissance de la population mondiale
(Tiré de World Urbanization Prospects : The 2003 Revision, United Nations 2004)

Presque toute la croissance démographique prévue au niveau mondial au cours des 30 prochaines années sera concentrée dans les milieux urbains. Ainsi, pour la première fois dans l'histoire de l'homme moderne, la population urbaine sera égale à la population rurale en 2007. Tandis que 30% de la population mondiale vivait dans les zones urbaines en 1950, cette proportion était de 47% en 2000 et devrait atteindre 82% en 2030 [United Nations Population Division, 2004].

Le milieu urbain est un système dynamique et complexe qui est en constante évolution dans le temps et dans l'espace. Cette dynamique résulte de l'interaction de plusieurs processus interdépendants qui gravitent dans ce monde urbain (Figure 2.2).

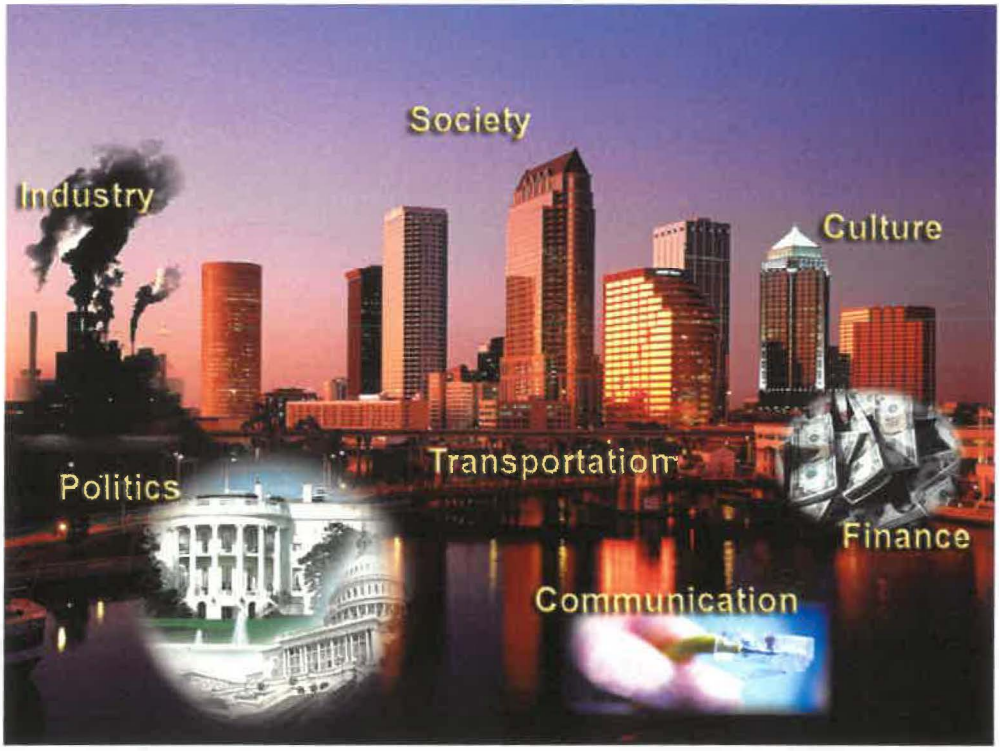


Figure 2.2. Les diverses sources de processus en milieux urbain

Ces processus se présentent sous diverses formes comme :

- processus économiques (finances) ;
- processus culturels ;
- processus de mobilité (transports) ;
- processus de gestion (politique) ;
- processus de communication ;

- processus de production (industries) ;
- processus démographiques (population) ;
- processus d'urbanisation.

À travers ces différents processus, on perçoit donc la ville comme un système complexe. C'est-à-dire que la ville est composée de nombreux processus différents réagissant entre eux sans règles précises et adaptant en permanence leurs comportements en fonction de leurs environnements. Ainsi, une meilleure compréhension de tous ces processus, qui constituent une ville, permettra de mieux contrôler les opérations militaires en milieu urbain.

2.2. La complexité des opérations urbaines

Les opérations militaires d'aujourd'hui se déroulent de plus en plus dans ces milieux urbains très complexes. Que ce soit des opérations de guerre, humanitaires ou du maintien de la paix, le théâtre de ces opérations est le reflet de l'urbanisation du monde moderne.

Voici quelques exemples courants qui illustrent la complexité actuelle des opérations dans un milieu urbain :

- Mogadishu en Somalie, 3 octobre 1993 : durant cette bataille, qui dura moins de 48 heures, 18 soldats américains ont été tués et plus de 77 ont été blessés [Legault, 2000]. Cette opération est connue sous le nom de la chute du faucon noir. Une centaine de marines américains sont déposés par hélicoptère au coeur de Mogadishu, la capitale de la Somalie, pour y capturer deux hauts lieutenants du général Mohamed Farrah Aidid. Au cours de l'opération, deux hélicoptères (Blackhawks), sont abattus et les quelques marines se retrouvent seuls confrontés à un large contingent de Somaliens fortement armés.

- Grozny, capitale de la Tchétchénie en 1999 : les Tchétchènes rendirent difficile l'avancée des Russes dans les rues de Grozny. Une compagnie tchétchène de 300 hommes pouvait tenir tête à une brigade russe de 5000 hommes. Les Forces russes, même mieux préparées qu'en 1995, étaient toujours faibles en tactique urbaine. Les troupes tchétchènes prirent le temps de préparer le terrain avec de nombreux points d'embuscades. Les Tchétchènes bloquèrent les portes et les fenêtres des premiers étages, rendant impossible le simple déplacement dans un immeuble. Pendant que les Russes essayaient d'entrer dans les immeubles, ils devinrent la cible des snipers tchétchènes positionnés sur les toits et dans les étages supérieurs. Selon les rapports, les soldats tchétchènes étaient divisés en sous groupes de 25 hommes, lesquels

étaient subdivisés en trois petits groupes de huit hommes chacun. Ils essayaient de rester près des troupes russes pour les faire sortir dans les rues, selon le chef des troupes de défense de Grozny, le général Aslanbek Ismailov. Les Tchétchènes avaient deux lignes de défense, avec les hommes les moins entraînés à l'avant. Les snipers occupaient les toits et les étages supérieurs des bâtiments, contrôlant les voies menant à des intersections spécifiques [Lieutenant-colonel Timothy, 2000].

- Québec, 20 avril 2001: lors du Sommet des Amériques, le conflit entre manifestants et policiers prend une ampleur surprenante devenant l'une des plus importantes opérations de sécurité de l'histoire du Canada. Misant sur une forte campagne d'intimidation, les forces de sécurité se sont révélées impuissantes à maîtriser quelques manifestants. Malgré les 6500 policiers, incluant 1500 affectés à l'anti-émeute, les 1200 militaires et une dépense d'au moins 70 millions $ en mesures de sécurité, il n'aura fallu que quelques manifestants bien déterminés pour faire tomber un pan complet de la clôture sur la rue René Lévesque. Devant tant de désordre public, la cérémonie officielle d'ouverture du Sommet des Amériques, a dû être retardée et d'autres activités prévues ont été annulées. Les nuages de gaz lacrymogènes ont enveloppé la haute-ville, empoisonnant les dignitaires et obligeant les autorités à verrouiller et sceller les édifices où se déroulaient les travaux du Sommet, de même qu'à interdire l'accès au périmètre à tout véhicule. [Tiré du journal « Cette Semaine », 2001]

- New York, 11 septembre 2001 : environ 80 personnes armées de quatre avions commerciaux ont pris part à l'une des opérations terroristes les plus tragiques qui a plongé en quelques heures les États-Unis dans l'horreur. Ils ont réussi à déstabiliser un pays et même une bonne partie du reste du globe. Au total, quatre avions de ligne ont été détournés par des terroristes et dirigés sur des cibles hautement symboliques, comme le World Trade Center, symbole de la puissance économique des États-Unis, et le Pentagone, siège de l'état-major américain.

- Washington, automne 2002 : un seul tireur d'élite suffit pour terroriser une ville entière et mobiliser plusieurs corps policiers, services fédéraux et agents du FBI. Le tueur choisissant ses victimes apparemment au hasard dans les lieux publics, a assassiné dix personnes et blessé trois autres à Washington et dans les états voisins du Maryland et de Virginie.

Pendant longtemps, les armées ont mené des sièges contre les villes mais livré bataille à la campagne. Aujourd'hui le champ de bataille a changé de lieu, c'est la ville. La ville devient avec le temps la zone de confrontation préférée de certains groupes. Mais cette ville comprend une multitude d'obstacles qui complexifient les opérations en milieux urbains. Voici quelques exemples illustrés dans la figure 2.3 de contraintes que les militaires rencontrent au niveau de l'environnement urbain :

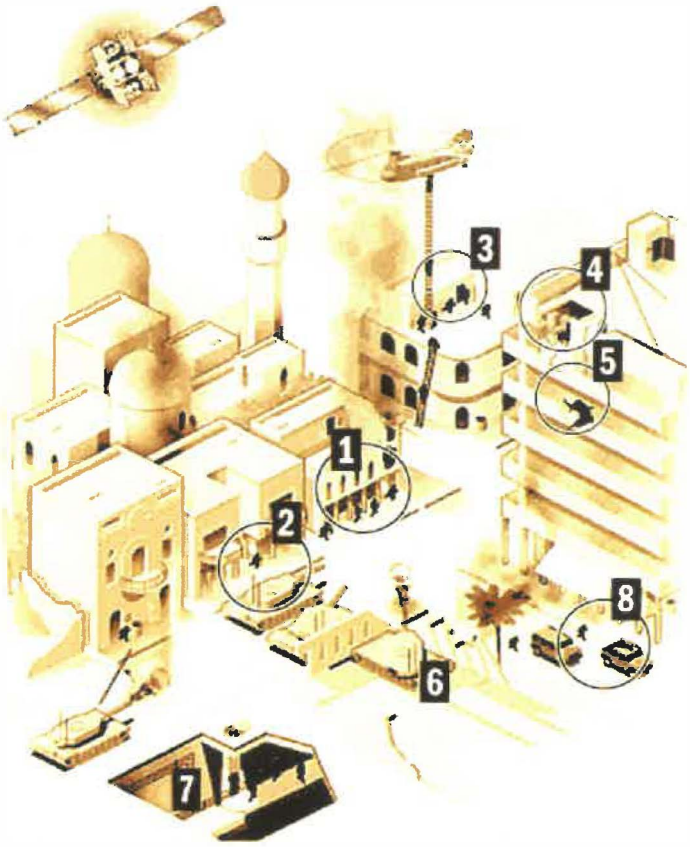


Figure 2.3. Complexité des opérations militaires en milieu urbain
(Adapté de The Age [The Age, 2004])

1. Le déplacement des troupes est souvent compliqué par la proximité des bâtiments et le déploiement sur un seul côté de la rue implique une couverture plus marquée pour le côté opposé.

2. Déplacement conjoint des troupes et des chars d'assaut pour compléter leur champ visuel restreint par les bâtiments. Les troupes à pied dirigent les mouvements du char, assignant les cibles à atteindre.

3. L'entrée par les toits des édifices peut s'avérer efficace pour déloger un sniper mais ce sont des endroits très propices pour les pièges.

4. Les édifices très hauts fournissent un bon endroit pour la surveillance et les communications, mais ils sont parfois difficiles d'accès.

5. La présence de tireurs embusqués peut être très dommageable et généralement ils sont difficiles à neutraliser.

6. Des chars d'assaut et des troupes ennemies peuvent se cacher à l'intérieur des bâtiments, procurant une protection supplémentaire et la possibilité d'une frappe surprise.

7. L'environnement souterrain assure un environnement caché pour le déplacement des troupes ennemies.

8. Les véhicules légers armés ennemis deviennent vite un facteur de contrainte pour la recherche de chemin à l'intérieur d'une ville, grâce à leurs déplacements rapides.

La complexité des opérations dans une ville se reflète aussi par la gestion des nombreuses sources de renseignements. Le renseignement militaire est l'un des facteurs clefs dans les opérations car il améliore la connaissance que le commandant a de la situation. Ainsi, le commandant peut engager ses ressources au bon moment et au bon endroit sur le champ de bataille grâce à l'apport de renseignements.

2.3. Le renseignement militaire

Le renseignement militaire est un regroupement d'informations provenant de données militaires et non militaires qui englobe les divers processus (politique, économique, social et autres). Le renseignement fait partie intégrante des opérations militaires et joue le rôle de fonction coordonnatrice centrale du ISTAR. ISTAR signifie renseignement, surveillance, acquisition d'objectifs et reconnaissance. ISTAR est en fait un système de systèmes qui collecte des informations par observation des soldats et par de nombreux capteurs électroniques. Cette information est ensuite transmise aux analystes du renseignement pour la transformer en renseignement puis l'acheminer vers le commandant et à son état-major afin de formuler les actions stratégiques et tactiques des opérations militaires.

Le rôle d'ISTAR consiste à relier le processus de renseignement à la surveillance, à l'acquisition d'objectifs et à la reconnaissance en vue d'améliorer la connaissance que le commandant a de la situation. Ainsi, le commandant peut engager ses ressources de manœuvre et d'attaque exactement au bon moment et au bon endroit sur le champ de bataille.

Une distinction entre le renseignement et l'information est de mise ici pour bien différencier ces deux représentations. L'information peut se définir de la façon suivante selon le manuel de campagne de la Défense Nationale : *Donnée non traitée, de toute nature, qui peut être utilisée pour l'élaboration du renseignement.* Le renseignement, tel qu'il est défini dans le glossaire OTAN est : *Le résultat de l'exploitation des renseignements bruts concernant les nations étrangères, les forces armées ennemies ou pouvant le devenir, les zones où des opérations sont effectivement menées ou pourraient l'être. Le terme s'applique aussi aux activités d'élaboration du renseignement et aux organismes qui s'y consacrent* [Canada National Defence, 2001].

Le renseignement forme une base fondamentale aux opérations et une composante importante du processus décisionnel. Le renseignement permet aux commandants de planifier et de diriger les opérations avec succès, d'identifier les objectifs critiques et de gagner des combats décisifs. Le renseignement est la somme de nos connaissances et de notre compréhension de l'environnement dans lequel se déroulent les activités militaires. Le renseignement militaire est classifié selon les différents types de sources de renseignements tels que (Figure 2.4.) :

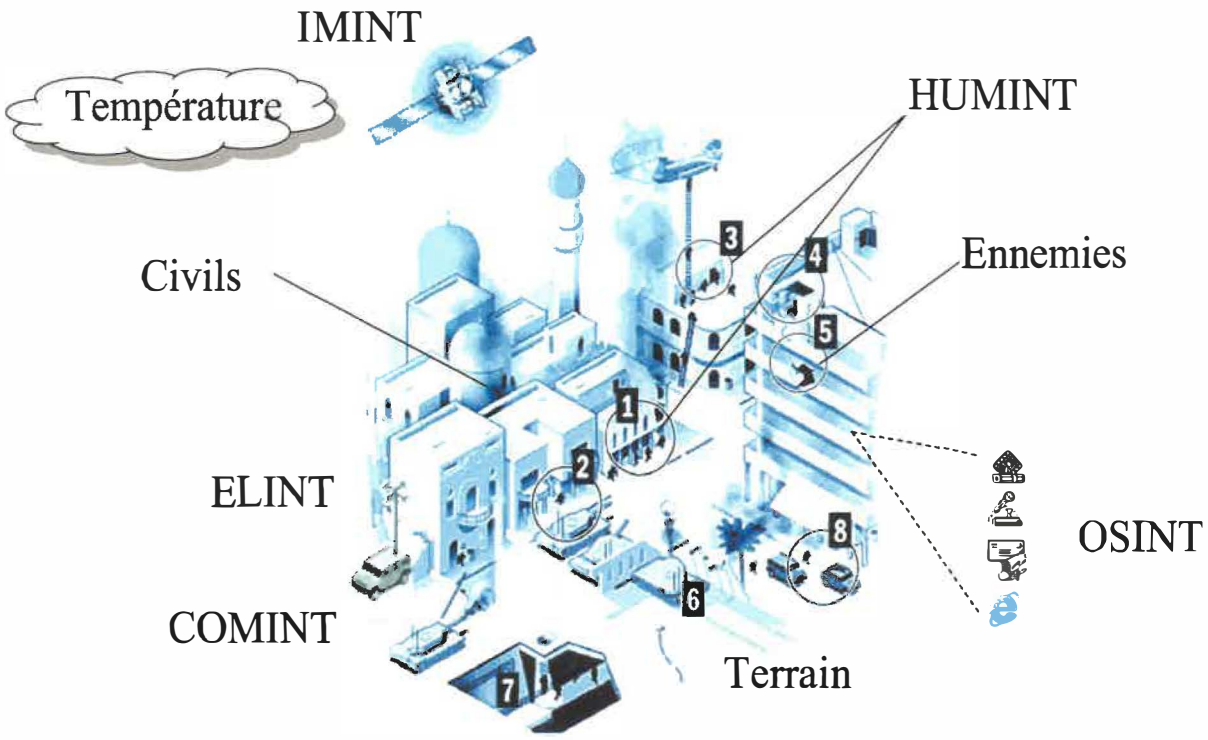


Figure 2.4. Représentation des différents types de sources de renseignements (Adapté de The Age [The Age, 2004])

- Le renseignement d'origine électromagnétique (SIGINT), qui inclut le renseignement de communications (COMINT) et le renseignement électronique (ELINT).

- Le renseignement par imagerie (IMINT). Renseignement obtenu à partir d'images acquises par des capteurs photographiques, radars, électro-optiques, infrarouges, thermiques et multi-spectraux.

- Le renseignement dit mesures et signatures (MASINT). Renseignement de type scientifique et technique qui découle de l'analyse qualitative de données métriques, angulaires, spatiales et temporelles. Regroupe des mesures dans le domaine acoustique et des radiations.

- Le renseignement de sources ouvertes (OSINT). C'est la catégorie de renseignement découlant des sources ouvertes comme les médias, la presse, les radios et l'Internet.

- Le renseignement humain (HUMINT). Renseignement découlant de renseignements bruts recueillis et fournis par une source humaine.

Lorsque les analystes du renseignement reçoivent ces divers types de données ou informations, elles font l'objet d'une vérification afin de les transformer en renseignements. Cette étape se fait en associant l'information à d'autres faits afin de réaliser une interprétation de celle-ci pour découvrir les causes, les circonstances ainsi que la signification possible. Ce processus de conversion des données et de l'information en renseignement est expliqué de façon plus détaillée dans la prochaine section qui traite du cycle du renseignement.

2.4. Le cycle du renseignement

Le cycle du renseignement est une séquence d'opérations par lesquelles les renseignements bruts sont obtenus, regroupés et rendus exploitables. Les renseignements bruts, ou informations, sont les données non traitées, de toute nature, qui peuvent être utilisées pour l'élaboration du renseignement.

Le cycle du renseignement suit une séquence d'activités incluant quatre phases : diffusion, orientation, recherche et exploitation (Figure 2.5).

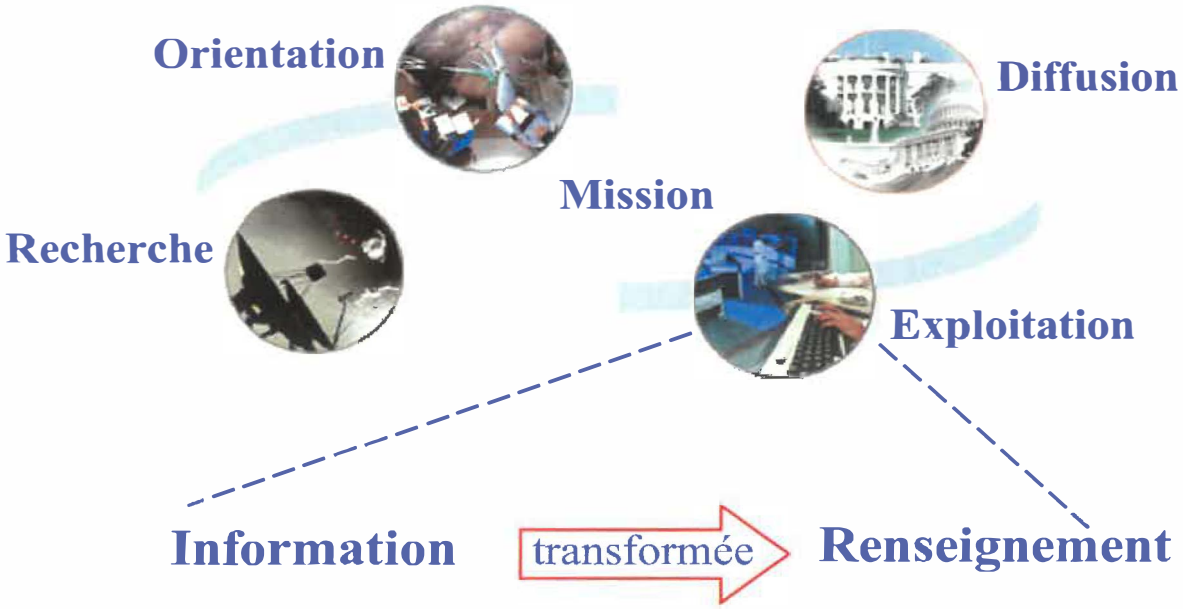


Figure 2.5. Représentation des étapes du cycle du renseignement (Adapté du manuel de campagne de la Défense Nationale)

Il est important de noter que le cycle n'a pas vraiment de phase de départ. En même temps que l'information est recueillie, l'information préalablement obtenue est exploitée, le renseignement est diffusé et une nouvelle orientation est prise.

La phase de l'orientation repose sur la définition des besoins en information. Dans cette phase, il s'agit d'acquérir une bonne connaissance de la stratégie militaire à appliquer pour dégager les grandes lignes de développement pour préparer la recherche d'informations.

La phase de recherche se concentre sur la collecte des informations. Elle s'appuie sur les diverses sources et organismes de recherche pour recueillir de l'information, tant électronique qu'humaine (HUMINT, SIGINT, etc.). Par la suite, l'information est transmise à l'état major du renseignement qui a fait la demande d'information.

La phase d'exploitation se veut une phase critique car c'est à ce niveau du cycle que l'information est transformée en renseignement. C'est à ce stade que des analystes interviennent pour valider les informations. La prochaine section décrira plus en détail de cette phase.

La phase de diffusion est essentiellement une phase d'envoi du renseignement. Le renseignement n'a de valeur que s'il parvient à temps et sous la bonne forme au commandant ou à l'état major pour la bonne conduite des actions et des décisions.

2.5. Exploitation

L'exploitation est un processus de regroupement des informations de même nature afin de réaliser l'évaluation de la fiabilité de la source et de la crédibilité de l'information. Ces évaluations indiquent le degré de confiance que l'on peut avoir pour le renseignement.

Le tableau 2.1 montre la cote d'évaluation de l'information. Celle-ci utilise une cotation par les lettres allant de A à F pour indiquer le niveau de fiabilité d'une source, et une cotation de chiffres de 1 à 6, afin de représenter le degré de crédibilité de l'information.

Fiabilité de la source		**Crédibilité de l'information**	
A	Entièrement fiable	**1**	Confirmée par d'autres sources
B	Habituellement fiable	**2**	Probablement vraie
C	Assez fiable	**3**	Possiblement vraie
D	Peu fiable	**4**	Douteuse
E	Non fiable	**5**	Improbable
F	La fiabilité ne peut être évaluée	**6**	La vérité de l'information ne peut être évaluée

Tableau 2.1. Système d'évaluation de la fiabilité des sources et de la crédibilité de l'information (Tiré du manuel de campagne de la Défense Nationale)

La fiabilité et la crédibilité sont considérées comme étant indépendantes l'une de l'autre. Par exemple, on attribue à une information reçue significative d'une source assez fiable et jugée probablement vraie la cote C2. Mais si la source de cette même information ne peut être évaluée, alors la cote est F2. Par la suite, une étape d'analyse permet de faire ressortir les faits significatifs ce qui conduit à un processus de déduction par rapport aux faits. Les déductions sont ordonnées de la plus probable à la moins probable. Un jugement doit alors être porté quant à la pertinence de ce classement. Cependant, la capacité des experts humains à effectuer

efficacement de telles évaluations peut être sérieusement affectée par des situations stressantes et la surcharge de l'information.

2.6. La problématique dans la phase de l'exploitation

Pendant le processus de la phase d'exploitation, plusieurs facteurs problématiques peuvent mener à une interprétation incorrecte de l'information en renseignement. L'un de ces facteurs est inévitablement la complexité du terrain urbain. En effet, celui-ci entraîne assurément un apport important en terme de quantité d'informations à traiter. De plus, certaines sources possibles d'information peuvent êtres ignorées ou mal exploitées. L'information elle-même peut être une source problématique de par sa grande complexité. L'information se retrouve sous plusieurs états possibles. Elle peut être :

- vraie ou fausse ;
- précise ou imprécise ;
- confirmée ou non confirmée ;
- complète ou inachevée ;
- nouvelle ou corroborée ;
- pertinente ou non ;
- positive ou négative.

Un autre facteur problématique est l'intervention humaine comme source d'information. L'information HUMINT est la plus complexe des sources de renseignement et elle est l'une des plus importantes sources dans les opérations urbaines
Le HUMINT en soi est une source très problématique pour beaucoup de raisons comme le manque de clarté et sa grande vulnérabilité au mensonge.

2.7. Conclusion

Le milieu urbain est un environnement dynamique et complexe qui regorge d'informations de toutes sortes et qui rendent plus difficiles l'utilisation de systèmes militaires. Ces systèmes reposent en grande partie sur cet ensemble massif de données provenant de différentes sources de renseignements. La surcharge des informations disponibles aux utilisateurs de systèmes diminue ainsi la capacité de ceux-ci à assimiler et à exploiter adéquatement l'information. Ce qui a pour effet une réduction de l'efficacité d'actions stratégiques et tactiques des opérations militaires. Le succès des opérations sera directement proportionnel à la rapidité et à l'efficacité avec laquelle la connaissance de la situation de l'ennemi et du milieu urbain sera fournie au commandant.

Chapitre 3
Revue de la littérature

La conception d'un système de méta-apprentissage (MA) est complexe à réaliser surtout si celui-ci repose sur une panoplie de techniques. En effet, les diverses techniques qui constituent le MA ne sont pas toutes compatibles avec la réalisation d'un système d'apprentissage adaptatif des connaissances en lien direct avec l'environnement. En contrepartie, un système avec une voie adaptative comme solution ne peut être réalisé dans une optique purement probabiliste ou à l'aide de nombreux cas d'apprentissage. Il faut appliquer une approche d'optimisation rapide et efficace aux moindres changements de l'environnement. Dans ce sens, une solution préconisant les algorithmes génétiques comme base du MA semble envisageable. Cependant, plusieurs travaux de recherche ont déjà été effectués sur le MA.

3.1. Méta-apprentissage

Le méta-apprentissage (MA) est un concept qui a fait son apparition dans les années 80. Il peut être décrit comme la capacité d'un système d'apprendre à apprendre. Par définition, le but du MA est de construire un système d'apprentissage auto-adaptatif qui s'adapte constamment à son environnement. Ce système améliore dynamiquement son fonctionnement en adaptant sa stratégie d'apprentissage en accord avec les expériences acquises, ou les méta-connaissances accumulées. Cette conception du MA est généralement la même pour plusieurs auteurs [Schmidhuber, 1987], [Schmidhuber, 1996], [Vilalta et Drissi, 2001]. Cependant, il s'avère que l'application de cette définition varie significativement d'un chercheur à l'autre. Les différents aspects et définitions du MA peuvent prendre plusieurs formes, mais la question qui revient toujours est la suivante : comment exploiter les connaissances de l'apprentissage pour améliorer les performances des algorithmes d'apprentissage ?

Dans un premier temps, il est important de mentionner les bases de la théorie de l'apprentissage pour comprendre ce qu'est vraiment le MA. Par la suite, une rétrospective des différents aspects du MA sera abordée selon divers auteurs et leur utilisation d'un seul ou de plusieurs algorithmes d'apprentissage. Puis, un survol de divers outils d'apprentissage sera traité mais avec une note plus spécifique sur les algorithmes génétiques. Finalement, une étude plus approfondie du MA impliquant l'utilisation des algorithmes génétiques terminera ce chapitre.

3.2. L'apprentissage en théorie

L'apprentissage est l'une des caractéristiques de l'intelligence humaine et fait partie du processus cognitif qu'est la mémoire. Cette mémoire représente la faculté de conserver des informations qui sont constituées par des expériences, des connaissances et elle permet de se souvenir des apprentissages antérieurs. Apprendre signifie donc intégrer de nouvelles informations en vue de s'en servir dans le futur. Lorsqu'une nouvelle information apparaît, le système peut la classifier, faire une prédiction à partir d'exemples, ou la généraliser à de nouvelles situations. Un modèle peut ainsi être défini par un schéma sur l'apprentissage d'un système en interaction avec son environnement (voir figure 3.1).

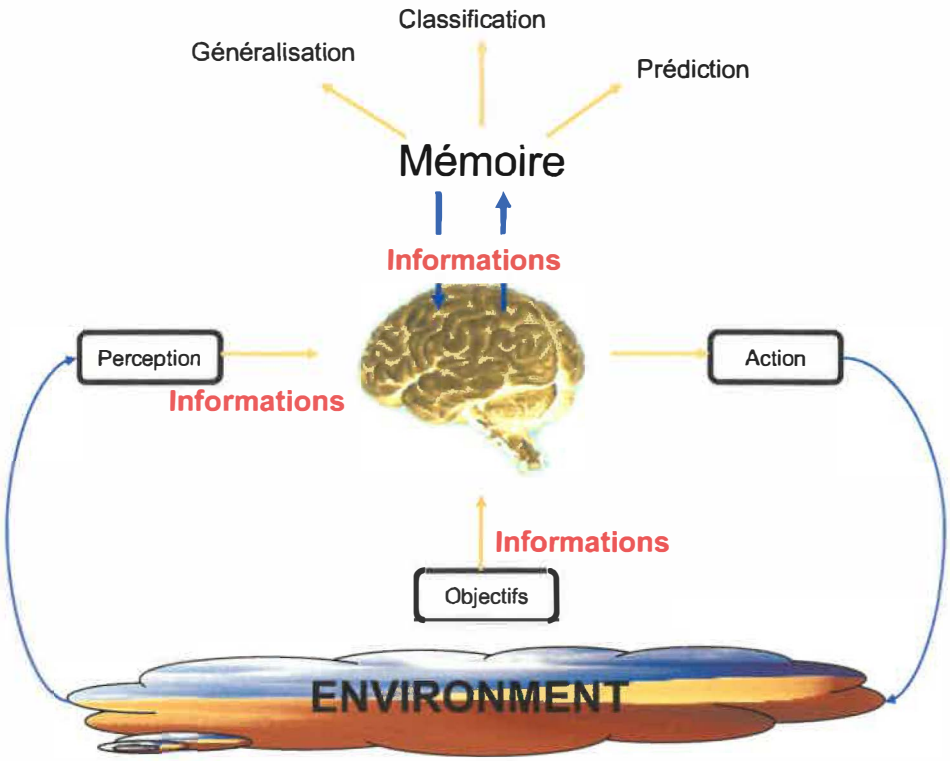


Figure 3.1. Schématisation de l'apprentissage

Pour que le système porte une action, il doit, à partir de l'information, extraire les relations entre les variables observées pour dégager des exemples d'apprentissage (ces exemples sont

couramment appelés *training sets*). C'est dans cet ensemble de solutions (figure 3.2) que l'apprentissage prend son importance pour en trouver la solution satisfaisante.

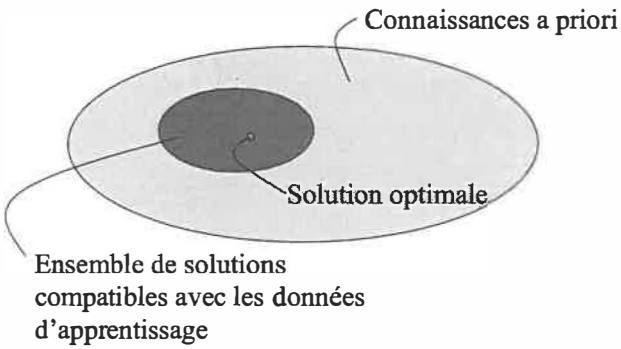


Figure 3.2. Apprentissage dans un ensemble de solutions
(Adapté des algorithmes d'apprentissage de [Bengio, 1997])

D'une manière plus théorique, un algorithme d'apprentissage est un algorithme qui a pour fonction d'apprendre à effectuer une tâche à partir d'un ensemble *S* de données. Cet ensemble contient un nombre fini *m* d'exemples d'apprentissage de la tâche à apprendre. Un algorithme d'apprentissage *A* apprend à partir d'un ensemble S d'exemples d'apprentissage comme suit :

$$S = \{\mathbf{z}_i\}_{i=1}^{m} = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_m\}$$

On suppose généralement que ces exemples sont tirés de manière indépendante de la même distribution inconnue P(Z). C'est l'hypothèse que les données sont identiquement et indépendamment distribuées.

Chaque exemple z_i est constitué d'un "objet" d'entrée X_i et d'une valeur de sortie y_i

$$\mathbf{z}_i = (\mathbf{x}_i, y_i)$$

Dans presque tous les cas, la valeur de sortie y_i de chaque exemple est une valeur numérique (réelle, entière, ou booléenne).

Souvent l'objet d'entrée de chaque exemple est un n-tuplet de valeurs numériques. Un tuplet peut se définir de la même manière qu'une liste soit un ensemble constant de valeurs dans un ordre déterminé.

$$\mathbf{x}_i = (x_1, x_2, \ldots, x_n)_i$$

Dans ce cas, le nombre n de valeurs d'entrée est le même pour chaque X_i qui est alors un vecteur de dimension *n*. Chaque exemple est généré par un environnement dont la structure est inconnue de la part de l'algorithme d'apprentissage.

Alors chaque exemple $z = (x,y)$ est la réalisation d'une variable aléatoire $Z = (X,Y)$.

L'objectif de l'algorithme d'apprentissage est de construire une "bonne" fonction *h*(x), appelée « l'hypothèse », à partir de l'observation d'un échantillon d'exemples *S* :

$$S = \{\mathbf{z}_i\}_{i=1}^{m} = \{\mathbf{z}_1, \ldots, \mathbf{z}_m\} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$$

Idéalement, nous aurions $h(x) = y$ pour tout exemple (x,*y*) non observé par l'algorithme. C'est-à-dire, *h*(x) est une "bonne" fonction si et seulement si elle prédit bien la valeur de sortie des exemples à venir.

Donc, l'entrée d'un algorithme d'apprentissage *A* est un échantillon d'exemples *S* et la sortie de *A* est une fonction appelée l'hypothèse *h*.

On écrit donc parfois: $A(S) = h$.
$h(x)$ subit une perte sur (x,y) lorsque $h(x) \neq y$. Cette perte est caractérisée par une fonction de perte l (h(x),y) non négative. Une fonction de perte doit normalement satisfaire:

$$l(y',y) \begin{cases} = 0 & \text{lorsque} \quad y' = y \\ > 0 & \text{lorsque} \quad y' \neq y \end{cases}$$

Typiquement pour $y' = h(x)$.

On peut alors décomposer l'erreur d'apprentissage en 3 éléments selon la figure 3.3. Dans cette figure, on traite l'ensemble des solutions possibles h, soit l'ensemble des valeurs y de sortie possibles. Ces sorties sont en fait représentées par une fonction de l'objet d'entrée x qui prédit une valeur de sortie (soit *f* une fonction de *x* vers *y*, *x* étant un objet d'entrée et *y* une valeur de sortie).

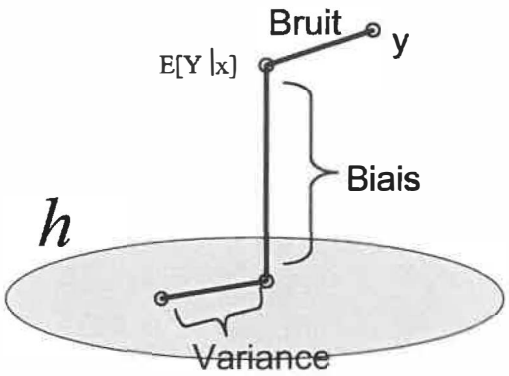


Figure 3.3. Représentation de l'erreur d'apprentissage
(Adapté des algorithmes d'apprentissage de [Bengio, 1997])

- **L'erreur de biais**

Le biais est l'écart entre la meilleure solution d'un algorithme d'apprentissage et la solution optimale. Plus l'ensemble de fonctions est grand, plus le biais sera petit. L'ensemble de fonctions h ne contient pas nécessairement la solution optimale, car l'espérance de la fonction choisie sur tous les ensembles d'apprentissage ne sera pas la fonction globalement optimale.

- **L'erreur de la variance**

La variance est la distance moyenne entre la fonction trouvée et la meilleure fonction qui aurait pu être trouvée. Plus l'ensemble de fonctions est grand, plus la variance pourra être grande.

- **L'erreur du bruit**

C'est l'erreur reliée au fait que la solution optimale peut être fausse. Dans toutes les données, nous retrouvons du bruit car elles sont imparfaites. Ces imperfections proviennent des erreurs

de mesure, d'une étiquette incorrecte, etc. Si, pour chaque valeur de X, il y a plusieurs valeurs de Y possibles, alors il n'existe pas de solution parfaite car la relation entre X et Y n'est pas définie par une seul fonction.

L'erreur due au bruit est difficilement contrôlable. Par contre, le couple biais-variance peut être contrôlable par les divers algorithmes d'apprentissage. Par exemple, l'application de la décomposition biais-variance dans l'utilisation de méta-algorithmes d'apprentissage apporte une réduction de la variance. Cet aspect des algorithmes d'apprentissage, ainsi que plusieurs autres points de vue seront énumérés dans la prochaine section du chapitre.
L'apprentissage peut être représenté sous différentes formes et se divise comme suit :

- **Apprentissage supervisé**

  Le système apprend à partir d'exemples pour lesquels on connaît la sortie ou l'action désirées. C'est-à-dire que les exemples fournis au système sont déjà classés.

- **Apprentissage non supervisé**

  C'est un apprentissage par lequel le système utilise des exemples sans fournir de retour explicite. C'est-à-dire que le système ne reçoit aucune information de l'environnement lui indiquant quelles devraient être ses sorties ou même si celles-ci sont correctes. Le système doit donc découvrir par lui-même les corrélations existant afin de déterminer la sortie à effectuer.

- **Apprentissage par renforcement**

  L'apprentissage se fait au moyen d'une évaluation par pénalité / récompense sans avoir besoin de spécifier comment la tâche doit être remplie. Le système apprend alors grâce à des comportements bons ou mauvais.

- **Apprentissage par apprentissage**

  C'est le type d'apprentissage qui réfère au MA, soit apprendre des connaissances qui ont été acquises par le système via les expériences antérieures. Donc, c'est le processus d'exploitation de la connaissance de l'apprentissage.

C'est au niveau du dernier point d'apprentissage que repose en grande partie le fondement des deux prototypes présentés dans les chapitres suivants. Par conséquent, la prochaine section élabore plus précisément sur différentes visions que partagent certains auteurs sur ce qu'est le MA.

3.3. Différents points de vue du méta-apprentissage

La description des divers aspects du MA suit, en général, l'article décrit par [Vilata et Drissi, 2002] qui font une bonne revue de la littérature dans le domaine de l'apprentissage machine (au sens *Machine learning*). L'apprentissage machine est l'étude des algorithmes qui améliorent automatiquement le système par l'apprentissage via les expériences acquises. La notion de méta-apprentissage réfère aux algorithmes de MA qui peuvent apprendre de meilleurs algorithmes d'apprentissage. Le MA est complexe à réaliser mais devient très utile dans le domaine de l'apprentissage machine.

- **Méta-algorithmes d'apprentissage**

Un méta-algorithme d'apprentissage combine les prédictions d'un système d'apprentissage de base. Pour ce faire, on entraîne séparément *n* modèles de manière à ce qu'ils soient le plus indépendants possibles l'un de l'autre et on combine leurs réponses. Les méthodes de combinaison les plus communes sont le *boosting*, le *bagging* et le *stacking*.

La méthode du *bagging* a comme principe de moyenner les prédictions de plusieurs modèles indépendants permettant ainsi de réduire la variance et donc de réduire l'erreur de prédiction.

La méthode du *boosting* est similaire à celle du *bagging* pour la création d'une famille de modèles qui sont ensuite agrégés par une moyenne pondérée des estimations. Elle diffère au niveau de la construction des modèles car chaque modèle est une version adaptative du précédent en donnant plus de poids, lors de l'estimation suivante, aux observations mal ajustées ou mal prédites.

Le principe du *stacking* a été introduit par [Wolpert, 1992]. Dans cette dernière, on entraîne de manière séquentielle plusieurs modèles, où le n-ième modèle est entraîné en tenant compte de la performance des modèles précédents. Par exemple, on entraîne un premier modèle, puis un second qui cherche à prédire l'erreur de généralisation faite par le premier sur chaque exemple d'apprentissage. On peut faire cela grâce à la méthode de la validation croisée.

- **Sélection dynamique de biais**

Ce type de MA se concentre sur l'évaluation et le choix de biais pour fournir un état approprié à l'algorithme d'apprentissage. Le choix dynamique de biais procède en modifiant l'espace d'hypothèse pour couvrir le plus grand nombre de tâches possibles.

- **Méta-règles associant la tâche avec la performance de l'algorithme**

Le but ici est de déterminer, dans certaines conditions, quel algorithme d'apprentissage on doit appliquer afin de bien exécuter les tâches actuelles. Il est alors souhaitable de tenir compte de plusieurs des critères des tâches en essayant d'identifier un algorithme approprié.

- **Transfert inductif**

Le transfert inductif améliore l'apprentissage pour une tâche en employant l'information contenue dans d'autres tâches relatives. Pendant que l'expérience s'accumule, l'apprentissage pour chaque tâche peut aider d'autres tâches à effectuer un meilleur apprentissage.

- **Système de classificateur**

C'est un système d'apprentissage par renforcement et par algorithmes génétiques qui est placé dans une situation d'interaction avec un environnement. Le système apprend à identifier des règles selon la réponse donnée par l'environnement. Ces règles se renforcent ou s'affaiblissent en fonction de leur efficacité, selon un système de récompense. Le système juge la qualité des connaissances en fonction de leur degré de contribution à la résolution du problème en interaction avec l'environnement. L'algorithme génétique va être utilisé pour modifier ces règles et il existe, dans les systèmes de classificateurs, deux approches pour l'application d'un AG :

  - **Approche de Pitt**

    Dans l'approche de Pitt, introduite par De Jong, on considère toute la base de règles comme un seul individu. Il faut donc raisonner sur une population de bases de règles pour sélectionner la meilleure population de règles.

  - **Approche de Michigan**

    Holland a développé un modèle cognitif dit approche de Michigan. Chaque règle y est considérée comme un individu, la population est constituée par l'ensemble des règles, et l'AG va donner les meilleurs de ces individus, soit les règles.

- **Raisonnement à base de cas**

Comme le définissent [Aamodt et Plaza, 1994], le raisonnement à base de cas est une approche capable d'utiliser la connaissance spécifique de problèmes (cas) s'étant déjà produits dans le passé pour répondre à de nouveaux problèmes. Un nouveau problème est résolu en cherchant un cas passé similaire, et en réutilisant sa solution dans le problème présent. Cette définition montre que le raisonnement à base de cas est un algorithme très naturel : en tant qu'être humain, lorsqu'un problème se pose, l'idée première est de rechercher si ce problème (ou un cas similaire) s'est déjà produit et, le cas échéant, reprendre la solution passée pour essayer de l'adapter au problème présent.

- **Agents intelligents**

Par définition, un agent est une entité située, réelle ou virtuelle, agissant dans un environnement, capable de le percevoir, d'agir sur celui-ci et d'interagir avec les différents composants l'entourant. Selon [Wooldridge et Jennings, 1995] un agent intelligent devrait posséder les différentes caractéristiques suivantes :

  - **Réactif**

    Un agent est capable de percevoir son environnement via des senseurs et d'agir sur son environnement via des effecteurs. La réponse à l'environnement se fait d'une façon favorable aux changements qui y arrivent pour satisfaire leurs objectifs de conception.

  - **Proactif**

    Un agent est capable de prendre de l'initiative pour atteindre son but ou effectuer des tâches et d'adopter les comportements appropriés.

  - **Social**

    Un agent est capable d'interagir, de communiquer avec les autres agents et de coopérer pour résoudre des problèmes ou effectuer des tâches.

De cette façon, l'interaction parmi différents agents intelligents peut être vue comme un MA et peut aider à résoudre des problèmes NP-complet d'une manière à obtenir des solutions satisfaisantes qui ne peuvent pas êtres obtenues avec certains algorithmes qui n'ont pas la capacité de communication.

3.4. Différents outils d'apprentissage

Le développement du système de MA passe par l'utilisation d'outils ou de techniques d'apprentissage que l'on retrouve dans le domaine de l'intelligence artificielle. Dans ce projet de recherche, l'utilisation des algorithmes génétiques favorise la famille des métaheuristiques comme outils de base. Une métaheuristique est constituée d'un ensemble de concepts fondamentaux (par exemple, la liste tabou et les mécanismes d'intensification et de diversification pour la métaheuristique tabou), qui permettent d'aider à la conception de méthodes heuristiques pour un problème d'optimisation [Hao et *al.*, 1999]. On peut ainsi établir une liste d'outils sommaire selon les méthodes suivantes :

- Méthodes évolutives :
    - **les algorithmes génétiques** ;
    - la programmation génétique ;
    - les algorithmes de colonies de fourmis.
- Méthodes de voisinage :
    - **le Recuit simulé** ;
    - la recherche tabou ;
    - la méthode de bruitage.
- Méthodes hybrides :
    - **la logique floue** ;
    - **les réseaux de neurones.**

Les modèles hybrides méritent une attention particulière même s'ils ne font pas partie de la famille des algorithmes d'optimisation. En effet, plusieurs travaux de recherche portent sur les systèmes hybrides comme outils d'apprentissage. Les sections suivantes décrivent ici les techniques soulignées dans l'énumération ci-haut. Le choix de ces quatre techniques, par rapport à toutes celles qui sont disponibles aujourd'hui, est justifié part un souci d'alléger le présent mémoire et motivé par le choix de techniques comportant un modèle reposant sur un fondement biologique.

3.4.1. Réseaux de neurones

Le fondement des réseaux de neurones provient d'une hypothèse proposée par de nombreux biologistes : pour recréer le comportement intelligent du cerveau, il faut s'appuyer sur son architecture, en fait, tenter de l'imiter. De ce fait, la base du réseau de neurones est le simple neurone biologique lui-même. Il est alors perçu comme un mini processeur qui traite les informations qu'il reçoit pour produire une information unique. Pour former le réseau de neurones, il y aura une architecture de connexion des mini processeurs fonctionnant en parallèle.

Le neurone (figure 3.4) est une cellule différenciée appartenant au système nerveux, qui assure le contrôle de toutes les fonctions de l'organisme. La structure du neurone comprend trois parties :

- un corps cellulaire qui renferme le noyau ;
- une ou plusieurs dendrites (prolongements du corps cellulaire) qui reçoivent des signaux, les convertissent en impulsions électriques et conduisent cet influx nerveux en direction du corps cellulaire ;
- un axone qui conduit l'influx du corps cellulaire vers les boutons terminaux synaptiques.

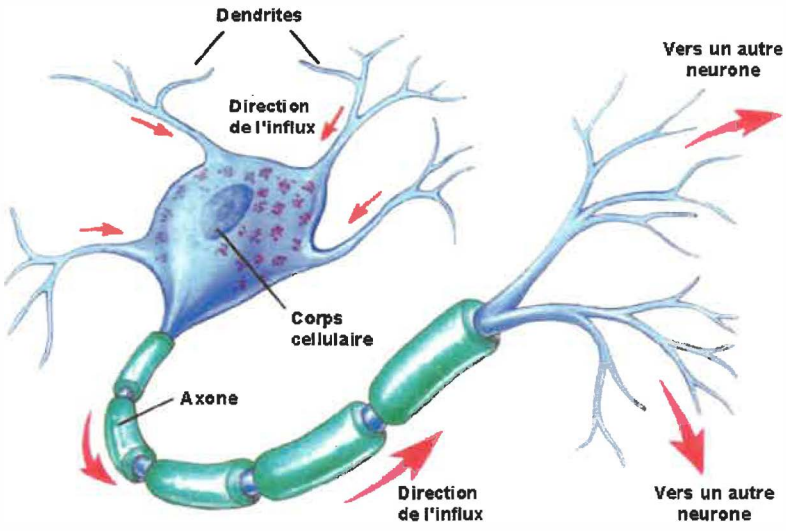


Figure 3.4. Structure d'un neurone biologique.
(Tiré de McGraw-Hill Companies www.mhhe.com/socscience/intro)

La communication entre un neurone et une autre cellule se produit au niveau d'une synapse (figure 3.5). Il existe une synapse toutes les fois qu'une des ramifications d'un axone entre en contact avec une dendrite. L'extrémité terminale d'un axone est un bouton synaptique qui possède de nombreuses vésicules synaptiques remplies de neurotransmetteurs. Les neurotransmetteurs sont, en fait, une substance chimique élaborée à l'extrémité d'un axone et qui permet la transmission de l'influx à travers la synapse. Notons que la transmission du signal (influx) est unidirectionnelle.

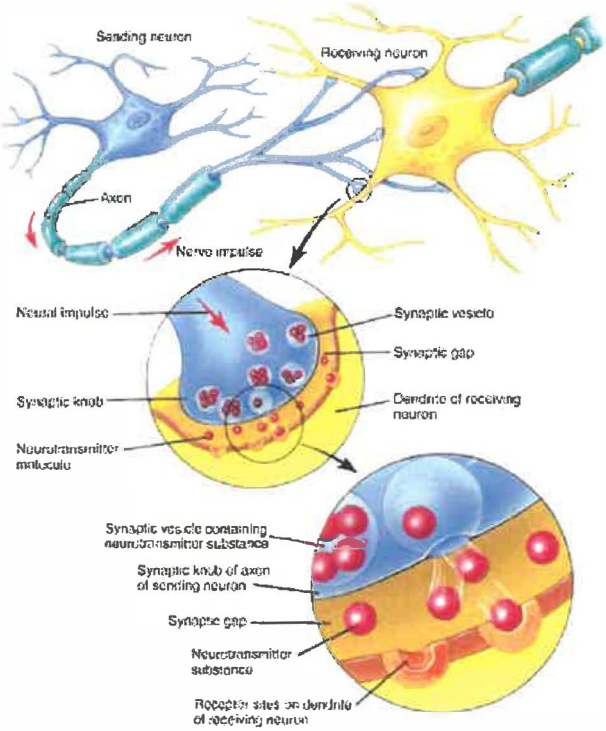


Figure 3.5. Représentation d'une synapse et des neurotransmetteurs.
(Tiré de McGraw-Hill Companies ww.mhhe.com/socscience/intro)

Une synapse se compose de la membrane présynaptique, de la fente synaptique et de la membrane postsynaptique polarisée car elle est négative du côté interne et positive du côté externe. Lorsque l'influx entre en contact avec les vésicules, celles-ci viennent se fusionner avec la membrane présynaptique pour décharger leurs neurotransmetteurs dans la fente ; ceux-ci parcourent la fente et viennent dépolariser la membrane postsynaptique. Cette dépolarisation déclenche un influx au niveau du neurone suivant.

Le neurone artificiel est une simplification des observations du fonctionnement du neurone biologique et conduit au modèle illustré dans la figure 3.6 suivante :

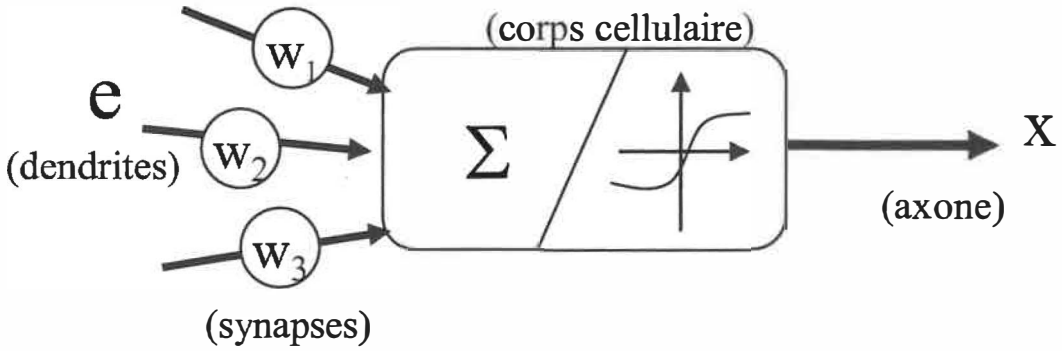


Figure 3.6. Modèle de Mac Culloch et Pitts du neurone artificiel
(Adapté des réseaux de neurones de [Amat et Yahiaoui, 1996])

Par analogie au neurone biologique, les dendrites sont des vecteurs d'entrée qui sont associés à un poids (boutons synaptiques) représentant la force de la connexion. Le corps cellulaire est en fait une fonction d'activation qui conduit à un résultat unique (axone), qui se ramifie ensuite pour alimenter un nombre variable de neurones.

Le fonctionnement des systèmes de neurones, à l'instar du système nerveux biologique, se caractérise par trois principes :

- le parallélisme des opérations de traitement ;
- le caractère collectif et la distribution de l'activité des éléments du réseau ;
- les capacités d'apprentissage à partir d'exemples.

Ils sont limités par plusieurs contraintes, et en particulier par la durée d'apprentissage qui augmente avec le nombre de neurones.

➢ Le fonctionnement

La dynamique des états d'un simple neurone (Perceptron) implique une fonction d'activation qui conduit à une seule valeur de sortie x. La fonction d'activation est la résultante de deux fonctions :

- Fonction de combinaison

  Les vecteurs d'entrée *e1*, *e2* et *e3* associés au neurone sont pondérés par les poids *W1*, *W2* et *W3* (voir figure 3.8 à la page 32). Cette fonction rassemble toutes les entrées en une seule valeur en faisant la somme pondérée des entrées.

$$a = \Sigma (w_i . \ e_i)$$

- Fonction de transfert

Cette fonction calcule la valeur de l'état du neurone par comparaison à une valeur de seuil *S*. Elle implique des fonctions de transfert qui sont généralement continues et la plus courante est la sigmoïde. La figure 3.7 montre différents types de fonctions de transfert.

Le premier modèle suit une fonction d'activation à seuil simple (binaire : 0=inactif, 1=actif). Le déclenchement de l'activité intervient si la somme des excitations dépasse un certain seuil propre au neurone.

Le second modèle à fonction linéaire par morceaux provient du modèle binaire, qu'on a perfectionné en ajoutant un intervalle. Un neurone est d'autant plus actif qu'il est excité (fonction monotone croissante) et un neurone ne peut être actif au-delà d'une certaine valeur (fonction bornée).

Le troisième modèle suit une fonction sigmoïde. Une fonction est sigmoïde lorsqu'elle est monotone croissante, dérivable et bornée. Ce type de fonction combine l'avantage de l'effet de seuil et de la dérivabilité. Les fonctions sigmoïdes sont donc très utilisées [Touzet, 1992].

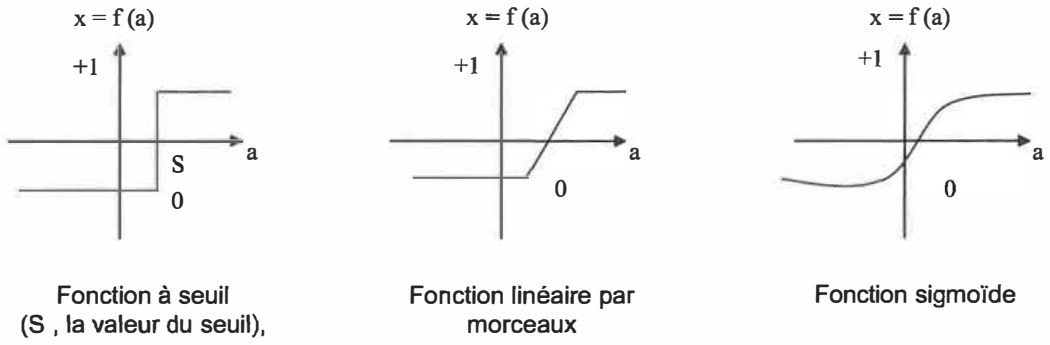


Figure 3.7. Représentation des trois types de fonctions de transfert (Tiré des réseaux de neurones artificiels de [Touzet, 1992])

Le fonctionnement d'un neurone peut alors se représenter dans la figure 3.8 qui suit :

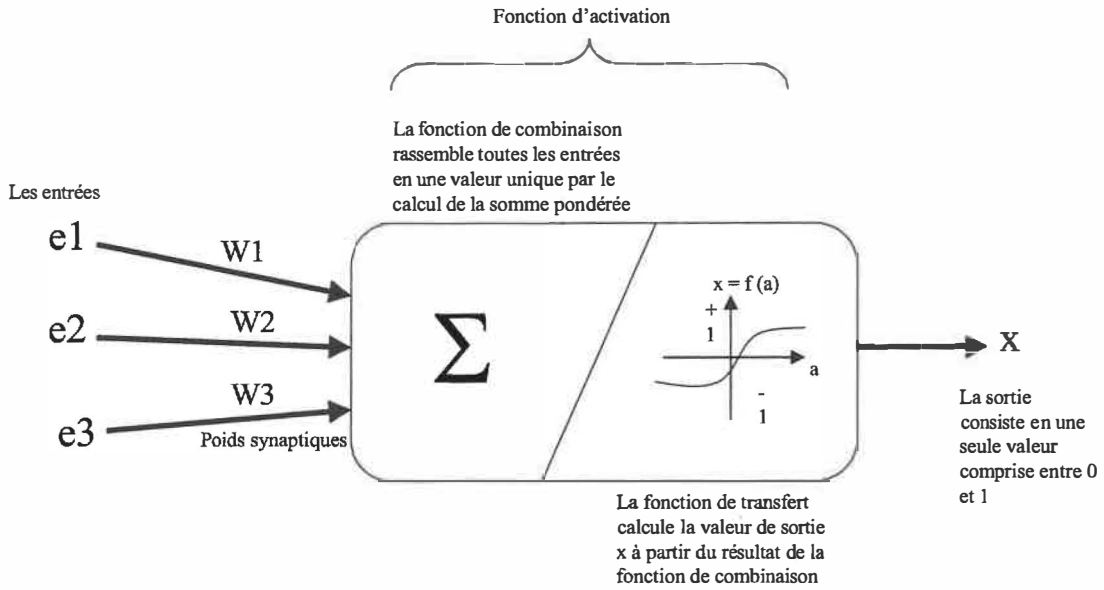


Figure 3.8. Schéma du fonctionnement d'un simple neurone (Perceptron) (Adapté des réseaux de neurones de [Amat et Yahiaoui, 1996])

- **Les limites**

Un simple neurone fonctionne en deux étapes pour réaliser une sortie X. Dans un premier temps, une sommation pondérée des entrées est effectuée. Dans un second temps, la valeur déterminée par la fonction de combinaison est transférée en sortie via une valeur de seuil. Puisque la sortie comporte une fonction de seuillage, elle ne peut prendre que deux valeurs. Ce fonctionnement est alors limité car le neurone simple ne sait calculer qu'une séparation linéaire. En fait, si le problème est non linéairement séparable, un simple neurone ne peut traiter ce cas.

La résolution d'un problème non linéairement séparable est réalisée grâce à un ensemble de neurones simples. Par contre un réseau de neurones doit, pour son apprentissage, utiliser plusieurs séries d'exemples pour résoudre un problème donné. L'utilisation fréquente de plusieurs séries d'exemples devient vite un facteur limitatif à l'utilisation d'un réseau de neurones.

3.4.2. Logique floue

La logique floue a été introduite en 1965 à Berkeley dans les travaux de Lotfi Zadeh avec la théorie des sous-ensembles flous, puis en 1978 avec la théorie des possibilités. La théorie des possibilités a été présentée pour généraliser le concept de probabilité, en s'appuyant sur des sous-ensembles flous.

Le concept de logique floue vient de la constatation que la variable booléenne, qui ne peut prendre que deux valeurs (vrai ou faux) est mal adaptée à la représentation de la plupart des phénomènes courants. Alors que la logique classique considère qu'une proposition est soit vraie soit fausse, la logique floue distingue une infinité de valeurs de vérité (entre 0 et 1). Elle permet de représenter des données imprécises et incertaines, et donc de rendre compte de concepts de la vie courante qui ne sont pas traités par les approches binaires classiques.

La logique floue prend en considération les données imprécises. Prenons l'exemple où on veut déterminer si 15°C est une température chaude ou froide. En logique classique, on introduit une valeur dite de seuil. Au-dessus de cette valeur, la température sera chaude et au-dessous, elle sera froide. Une erreur de 1% sur la valeur du seuil peut provoquer une erreur de 100% sur la décision. La logique floue permet de caractériser une appartenance graduelle à un sous-ensemble ou sous-ensemble flou. Cela signifie que la théorie des ensembles flous permet d'éviter le seuillage et ainsi de combiner plusieurs critères sémantiquement contradictoires mais numériquement non exclusifs.

Dans la figure 3.9 suivante, il n'y a pas exclusion mutuelle entre les deux ensembles flous froid et chaud. T1 est une température qui appartient à l'ensemble flou froid à 60% et à l'ensemble flou chaud à 20%.

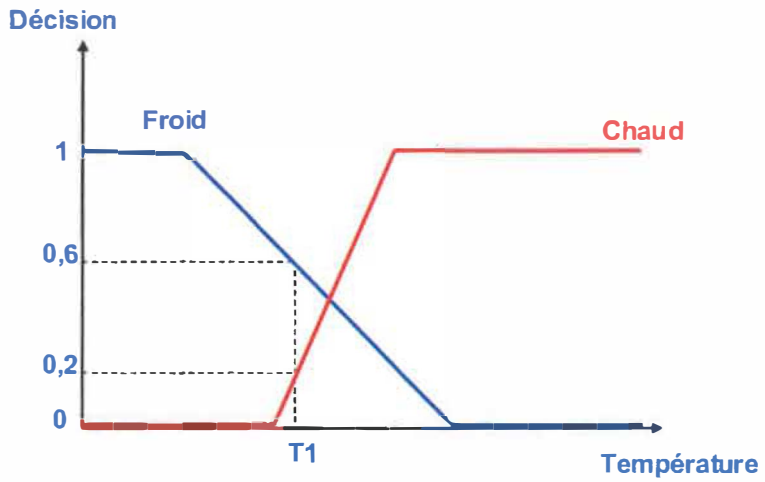


Figure 3.9. Graphique du degré d'appartenance aux sous-ensembles flous (Adapté de la logique floue de [Amat et Yahiaoui, 1996])

La logique floue prend aussi en compte les données incertaines. Reprenons l'exemple de la température, mais avec une mesure perturbée par une interaction physique inconnue qui rend incertaine la mesure du capteur. On peut représenter le taux d'incertitude (ligne verte) sur cette mesure sur la figure 3.10. Lorsque l'incertitude vaut 0, la possibilité qu'une température d'environ 15°C soit froide est différente de la possibilité pour que cette même température soit chaude. Lorsque l'incertitude se rapproche de 1, alors ces deux possibilités sont toutes deux égales à 1 et dans ce cas tout est possible.

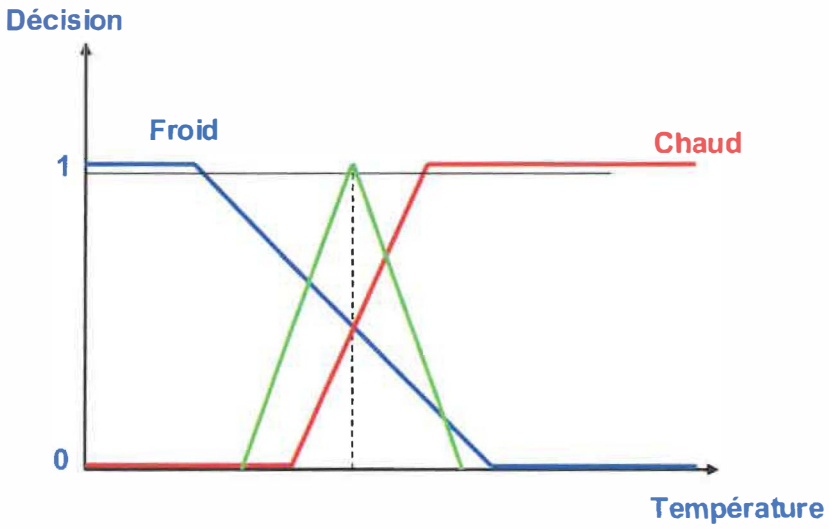


Figure 3.10. Graphique représentant les incertitudes dans la logique floue (Adapté de la logique floue de [Amat et Yahiaoui, 1996])

➢ Fonctionnement

Le fonctionnement de la logique floue passe par trois phases, soit :

- la théorie des ensembles flous (fuzzification)
- base de règles floues et inférence floue
- défuzzification

Fuzzification

La fuzzification permet la conversion des données en entrée en degré d'appartenance par l'intermédiaire de fonctions d'appartenance. Prenons un ensemble flou A, défini sur un univers de discours U (ensemble d'éléments discrets ou continus) par sa fonction d'appartenance μ_A. La grandeur $\mu_A(x)$ définit le degré d'appartenance de l'élément x à l'ensemble A.

$$\mu_A : U \rightarrow [0,1]$$

$$A = \{ (x, \mu_A(x)) \mid x \in U \}$$

Comme dans le cas des ensembles classiques, les ensembles flous possèdent certaines propriétés et mathématiquement, il existe une différence essentielle :

Probabilités : $p(A \cap \bar{A}) = p(\Phi) = 0$ et $p(A * \bar{A}) = p(U) = 1$

Logique floue : $p(A \cap \bar{A})$ pas nécessairement égal $p(\Phi)$ et $p(A * \bar{A})$ pas nécessairement égal $p(U)$

Donc, l'intersection d'un ensemble flou et de son complément n'est pas vide et l'union d'un ensemble flou et de son complément ne donne pas l'univers du discours U.

Règles de bases

Les règles peuvent mettre en jeu plusieurs variables dans leurs conditions et leurs conclusions et la détermination du degré d'appartenance de chacune des conditions des règles se fait par inférence.

L'application de la logique floue repose sur un ensemble de règles du type (**si** variable **alors** condition) établies en général de manière empirique. Par exemple : **Si** la température est faible **alors** chauffer plus.

On peut évidemment, comme en logique classique, utiliser des opérateurs **ET**, **OU** et **NON**. La définition des opérateurs se fait à partir des fonctions d'appartenance aux ensembles flous correspondant. on considère le degré d'appartenance **maximum** parmi les conditions d'entrée.

Lorsqu'on utilise un **ET**, seul le degré d'appartenance minimum parmi les conditions d'entrée est pris en compte. $\mu_A(x) \cap \mu_B(x) = min\ (\mu_A(x), \mu_B(x))$

De même, pour un **OU**, on ne tient compte que du degré d'appartenance maximum parmi les conditions d'entrée. $\mu_A(x) \cup \mu_B(x) = max\ (\mu_A(x), \mu_B(x))$

Le **NON** se traduit par un complément ou une négation. $non\ (\mu_A(x)) = \mu_{\bar{A}}(x) = 1 - \mu_A(x)$

On peut aussi utiliser des opérateurs de précision du type **ENVIRON**, **EXACTEMENT**, etc. En appliquant les règles, on obtient le degré d'appartenance des variables de sortie à chaque état.

Défuzzification

L'objectif de la défuzzification est de transformer un ensemble flou en une valeur de commande.

Soit *C* un ensemble flou, et *defuzz* l'opérateur de défuzzification :

$z_u = defuzz(C)$, est une valeur précise.

Le problème de la defuzzification est d'obtenir une valeur précise à donner aux variables de sortie à partir des degrés d'appartenance.

- **Les limites**

La logique floue s'avère très simple à mettre en œuvre par rapport aux techniques traditionnelles basées sur des modèles mathématiques. Ainsi, un système basé sur la logique floue peut être plus efficace mais le caractère empirique de ces règles est par ailleurs un inconvénient, car ces règles ne sont pas précises et peuvent être une source d'erreurs. De plus, il n'existe pas toujours de modèle mathématique pour toutes les situations, et s'il existe, ce modèle peut être très coûteux en temps de calcul ou en espace mémoire. La logique floue a beau avoir une apparence intuitive, son utilisation demande tout de même une certaine expérience.

3.4.3. Recuits simulés

Certains auteurs vont jusqu'à dire du recuit simulé qu'il est l'un des plus vieux algorithmes d'apprentissage et un des premiers à utiliser une stratégie pour éviter le piège des minimums locaux. L'algorithme est inspiré du principe de la thermodynamique et a été proposé par [Kirkpatrick et *al.*, 1983].

Le principe de la thermodynamique repose sur les notions de chaleur et de température et peut se définir comme la science de tous les phénomènes qui dépendent de la température et de ses changements. Ce principe est utilisé en métallurgie pour améliorer la qualité d'un solide par la cherche d'un état d'énergie minimale (correspond à une structure stable du solide). En partant d'une haute température à laquelle le solide est devenu liquide, la phase de refroidissement conduit la matière liquide à retrouver sa forme solide par une diminution progressive de la température. Chaque température est maintenue jusqu'à ce que la matière trouve un équilibre thermodynamique. Quand la température tend vers zéro, seules les transitions d'un état à un état d'énergie plus faible sont possibles.

Le principe de base du recuit simulé est alors emprunté à la physique du solide : on chauffe un morceau de métal, puis on le laisse refroidir lentement. Si le refroidissement est trop brusque, les atomes peuvent s'éloigner de leur état d'équilibre et causer une imperfection au morceau de

métal. Le refroidissement du métal est donc lent et régulier pour permettre aux atomes de se stabiliser peu à peu dans une position d'énergie minimale.

L'algorithme de recuit simulé consiste donc en une recherche aléatoire de l'espace d'état, à favoriser les descentes, mais sans interdire tout à fait les remontées. Cependant, une augmentation du niveau d'énergie permet de sortir des minima locaux (figure 3.11). L'algorithme montre qu'on tend très rapidement vers un minimum local proche de la meilleure solution.

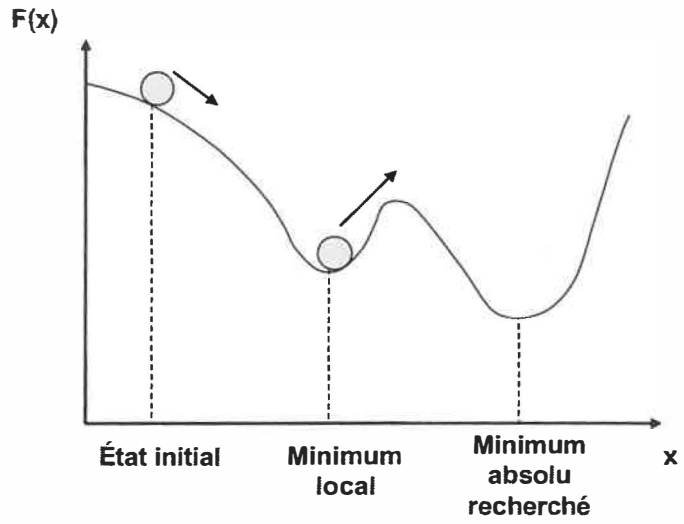


Figure 3.11. Graphique de la recherche dans l'espace d'état avec le recuit simulé
(Adapté du récuit simulé de [Amat et Yahiaoui, 1996])

- **Le fonctionnement**

Par analogie avec la physique, on fait intervenir un paramètre T (température) qui décroît lentement avec le nombre d'itérations et qui est associé à une probabilité qui est une fonction croissante de T. Au début de l'algorithme, le nombre d'itérations est faible et la température est élevée, ce qui permet d'éviter les minimums locaux. Tout au long de l'algorithme, la température va décroître, ni trop vite pour ne pas rester bloquée autour d'un minimum local, ni trop lentement pour un temps de traitement raisonnable. On représente à la figure 3.12 de la page 39 le pseudo code de l'algorithme du recuit simulé.

1. [Initialisation] Génération aléatoire d'une solution s , T ← valeur initiale
2. [Tant que] Génération aléatoire d'une solution s' voisine de s jusqu'au critère d'arrêt
    3. [Condition] Si $f(s') < f(s)$ alors s ← s'
        4. [Sinon] faire s ← s' avec une probabilité $e^{\frac{f(s)-f(s')}{T}}$
    5. [Mise à jour] Diminuer T
6. [Fin tant que] Aller en 2 ou Stop

Figure 3.12. Pseudo code de l'algorithme du recuit simulé

Dans l'algorithme de recuit simulé, le critère de Metropolis est utilisé et permet de décider si un nouvel état généré présente une variation de coût acceptable. Il permet aussi de sortir des minima locaux quand le critère d'arrêt n'est pas encore atteint. Entre chaque passage d'un état u à un état v, on calcule la variation de la fonction de coût :

$$\Delta g = g(v) - g(u)$$

La transformation est acceptée selon la probabilité :

$$p(u,v) = e^{\frac{-\Delta g}{T}}$$

Si $\Delta g \leq 0$
Alors le nouvel état doit être accepté et la probabilité est maximale et vaut 1.

Si $\Delta g > 0$
Alors on compare $p(u,v)$ à un nombre aléatoire r [0,1[.

Si $r < p(u,v)$ l'état v est accepté,
sinon il est rejeté et l'on essaie un autre état.

Au début de l'algorithme, la probabilité p(u,v) est proche de 1 et presque toutes les variations Δg sont acceptables. Lorsque la température diminue, les remontées sont de plus en plus difficiles et très peu de variations peuvent être acceptées.

- **Les limites**

Lorsqu'il y a trop de minima locaux, le nombre d'itérations devient prohibitif. Par conséquent, l'algorithme peut s'arrêter sur une solution non-optimale. Cependant, cet algorithme a la propriété d'exploration aléatoire locale, au voisinage d'un point donné. Il est utile pour une recherche locale rapide mais moins adapté pour une recherche globale.

De plus l'algorithme converge bien vers une solution optimale, mais la convergence peut être coûteuse en temps. En effet, par le principe d'un refroidissement lent, l'algorithme devient moins rapide s'il est comparé à d'autres algorithmes d'apprentissage. Il est à noter qu'il est difficile de trouver les bons coefficients à appliquer pour obtenir une convergence efficace car le recuit simulé est très dépendant de la structure de voisinage et du nombre de paramètres qui gèrent la recherche. En plus du problème des minima locaux, le recuit simulé ne considère qu'une seule solution à la fois et ne construit pas la forme générale de l'espace de recherche.

3.4.4. Algorithmes génétiques

Les algorithmes génétiques (AG) sont des algorithmes d'optimisation basés sur la théorie de l'évolution des espèces de Charles Darwin [Darwin, 1980]. Les premier travaux de John Holland remontent aux années 1960 et ont trouvé un premier aboutissement en 1975 avec la publication de *Adaptation in Natural and Artificial Systems* [Holand, 1975]. C'est cependant l'ouvrage de David Goldberg qui a largement contribué à développer les algorithmes génétiques [Goldberg, 1989].

Un algorithme génétique a pour but de rechercher un extrema d'une fonction définie sur un espace de données. L'algorithme repose sur les points suivants :

- un principe de codage de l'élément de population : la qualité du codage des données conditionne le succès de la recherche de l'optimum. On retrouve deux formes de codage : le codage binaire et le codage réel.

- un mécanisme de génération de la population initiale non homogène. Le choix de la population conditionne la rapidité de la convergence vers l'optimum.

- une fonction à optimiser qui retourne une fonction d'adaptation (*fitness*) de l'individu.

- un mécanisme de sélection des individus permettant d'identifier statistiquement les meilleurs individus d'une population et d'éliminer les moins bons.

- des opérateurs permettant de diversifier la population au cours des générations et d'explorer l'espace des autres solutions possibles. L'opérateur de croisement (*crossing-over*) recompose les gènes d'individus existant dans la population. L'opérateur de mutation a pour but de garantir l'exploitation de l'espace de solutions.

- des paramètres de dimensionnement : taille de la population, critères d'arrêt, probabilité d'application des opérateurs génétiques.

➢ **Fonctionnement**

Un algorithme génétique est défini par un cycle de population et fait intervenir trois facteurs importants : *fitness*, *crossing-over*, mutation (figure 3.13). Un cycle représente le passage d'une population à la génération suivante soit l'évolution génétique d'une population.

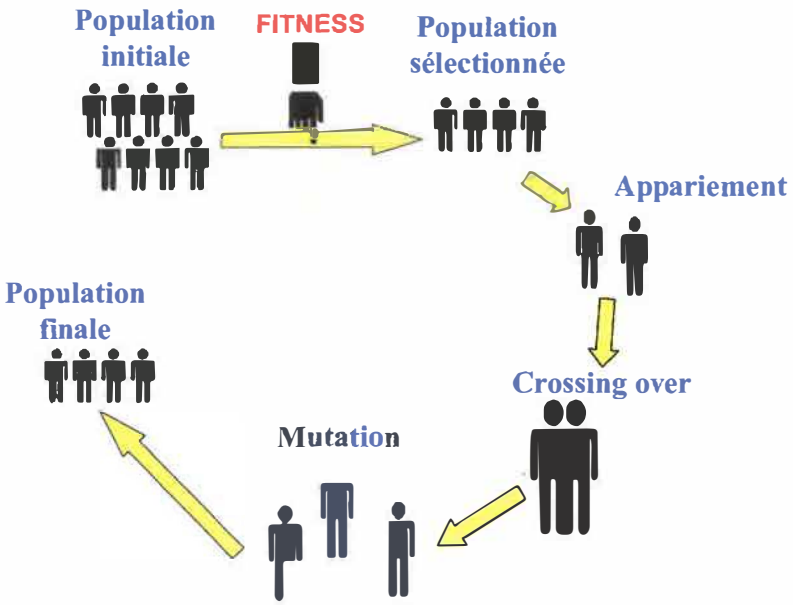


Figure 3.13. Illustration d'un cycle d'un algorithme génétique

Le fonctionnement de l'algorithme est représenté par le pseudo code de la figure 3.14 suivante :

a. **[Initiation]** Générer aléatoirement une population de *n* individus (les solutions du problème)
b. **[Fitness]** Evaluer la fonction d'adaptation *f(x)* de chaque individu *x* dans la population
c. **[Nouvelle population]** Créer la nouvelle population en répétant les étapes suivantes jusqu'à ce que la population soit complète
   d. **[Sélection]** Sélectionner deux parents à partir de la population selon la valeur d'adaptation (La meilleure valeur a plus de chance d'être sélectionnée)
   e. **[Crossing-over]** Appliquer l'opérateur de croisement sur les parents avec la probabilité de croisement associée pour donner des enfants. S'il n'y a pas de croisement, les enfants sont la copie identique des parents
   f. **[Mutation]** Appliquer l'opérateur de mutation sur les enfants avec la probabilité de mutation associée
   g. **[Accepter]** Placer les enfants dans la nouvelle population
h. **[Remplacer]** Utiliser la population générée pour exécuter l'algorithme
i. **[Test]** Si la condition d'arrêt est satisfaite, **stop,** et retourner la meilleure solution
j. **[Boucle]** Aller à l'étape **b**

Figure 3.14. Pseudo code de l'algorithme génétique

Pour comprendre en détail les étapes précédentes, le pseudo code est repris selon un modèle décrit par [Holland, 1975]. Ce modèle comporte une population *P* de *n* individus (solutions) où *n* est un entier fixé. Chaque solution s'exprime sous la forme d'une chaîne binaire de longueur *l* et l'AG cherche à trouver la chaîne binaire maximisant *f*. Pour bien illustrer le modèle, l'exemple du problème *OneMax* [Schaffer et Eshelman, 1991] est utilisé. Ce problème a été largement utilisé avec les algorithmes génétiques. Le problème *OneMax* est un problème simple qui consiste à maximiser le nombre de bits à 1 dans une solution donnée. Ainsi, si on considère que :

P = population
n = 5 individus
$G(P_1,P_2,P_3,.....P_n)$ = génération
l = 6 bits (chaîne binaire)
s = 000000 (solution au point d'abscisse 0)
s = 111111 (solution au point d'abscisse 63)

La fonction d'adaptation *f(s)* est définie comme le nombre de bits de valeur 1 dans une solution donnée. Dans cet exemple, la recherche de la fonction d'adaptation est assez simpliste, mais en général, la recherche de cette fonction n'est pas une tâche triviale (voir la section limitation). Alors,

$f(000000) = 0$ (valeur d'adaptation pour l'individu au point d'abscisse 0)
$f(111111) = 6$ (valeur d'adaptation pour l'individu au point d'abscisse 63)

La solution optimale cherchée est $s^* = 111111$ et la fonction f est représentée dans la figure 3.15 suivante :

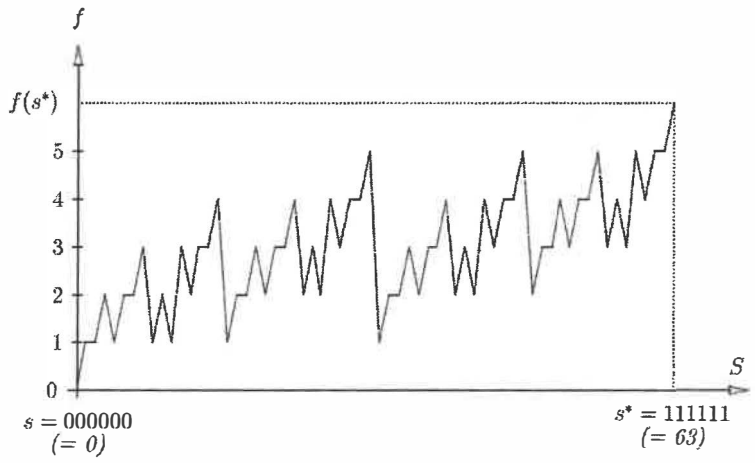


Figure 3.15. Graphique des solutions possibles de la fonction f

a. **[Initiation]**

La première étape est la génération aléatoire d'une population P_1 de n individus. Ces individus représentent les s solutions au problème. La population ne représente alors qu'une infime partie de l'ensemble des solutions possibles.

b. **[Fitness]**

Une évaluation de la fonction d'adaptation *f(s)* de chacun des individus est effectuée et elle est représentée dans le tableau 3.1.

P_1	
s	*f(s)*
011011	4
100101	3
001010	2
001101	3
110100	3

Tableau 3.1. Représentation de la première génération P_1

c. **[Nouvelle population]**

Création d'une nouvelle génération P(t + 1) en fonction de la population p(t). Cette itération de création se poursuit tant que les nouvelles solutions de p(t + 1) ne satisferont pas la fonction d'adaptation.

d. **[Sélection]**

La sélection va choisir une première population intermédiaire P_s de n solutions à partir de *P(t)*. La première population intermédiaire P_s est sélectionnée en effectuant n tirages aléatoires de solutions de *P(t)* où chaque solution s_i de *P(t)* a la probabilité suivante d'être sélectionnée :

$$P_{select}(s_i) = \frac{f(s_i)}{\sum_{j=1}^{n} f(s_j)}$$

Selon l'exemple ci-haut, l'application de la sélection sur la population P_1 donne :

P_1		
s	$f(s)$	$P_{sélect}(s)$
011011	4	27%
100101	3	20%
001010	2	13%
001101	3	20%
110100	3	20%

Tableau 3.2. Représentation de la première génération P_1 avec l'application de la sélection

Par la suite, un tirage aléatoire de la population intermédiaire P_s est réalisé. La sélection favorise les meilleurs individus (011011) au détriment des mauvais (001010). Le tableau 3.3 représente la population intermédiaire.

P_1	
S	$f(s)$
011011	4
100101	3
001010	2
001101	3
110100	3

P_s	
s	$f(s)$
011011	4
100101	3
011011	4
001101	3
110100	3

Tableau 3.3. Représentation de la population intermédiaire Ps_1

e. **[Crossing-over]**

Le croisement fait l'échange d'informations entre deux solutions sélectionnées de P_s pour former la population P_c de n solutions. Pour sélectionner des couples, la population P_s est parcourue et chaque solution a une probabilité pc d'être sélectionnée pour le croisement.
Après sélection, les couples choisis échangent de l'information (bits) selon l'opérateur de croisement. Dans ce cas, l'opérateur a un point de coupure avec une probabilité de croisement $pc = 0{,}8$. Les enfants ainsi engendrés remplacent leurs parents pour former la population P_c.

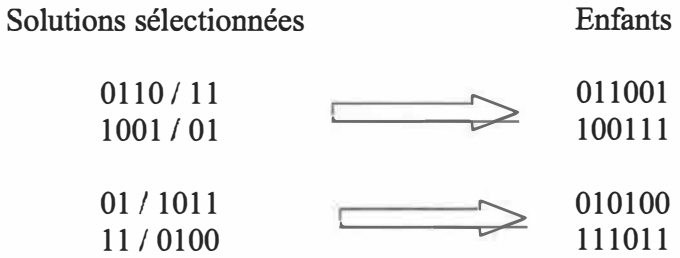


Figure 3.16. Représentation du croisement des solutions sélectionnées pour produire les enfants

P_s	
s	*f(s)*
011011	4
100101	3
011011	4
001101	3
110100	3

P_c	
s	*f(s)*
011001	?
100111	?
010100	?
001101	3
111011	?

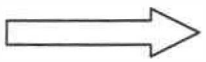
Tableau 3.4. Représentation de la population croisée *Pc*

On remarque que l'évaluation de la fonction d'adaptation est réalisée après les étapes de croisement et de mutation.

f. **[Mutation]**

L'opérateur de mutation modifie aléatoirement, selon une probabilité associée *pm*, la population pour donner la population P_m. Dans ce cas, la population P_c est parcourue et chaque bit de la solution a une probabilité $pm = 0.05$ d'avoir une mutation (inversion du bit).

P_c	
s	*f(s)*
011001	3
100111	4
010100	2
001101	3
111011	3



P_m	
s	*f(s)*
111001	?
100101	?
010101	?
001101	3
111011	3

Tableau 3.5. Représentation de la population mutante *Pm*

g. **[Accepte, Remplace, Test]**

Après acceptation des modifications, les enfants forment la population *P(t+1)*. Ainsi la nouvelle génération P_2 du tableau 3.6 peut recommencer le cycle, à moins que la condition d'arrêt soit satisfaite. Plusieurs conditions peuvent être utilisées, comme le dépassement d'un certain nombre de générations, ce qui évite de tomber dans une boucle sans fin. L'algorithme peut aussi s'arrêter lorsque la qualité de la solution trouvée dépasse un seuil donné.

P_2	
s	*f(s)*
111001	4
100101	3
010101	3
001101	3
111011	5

Tableau 3.6. Représentation de la deuxième génération P_2

- **Les limites**

Les difficultés de mise en œuvre sont le choix d'un codage du problème, la définition de la fonction d'adaptation et le paramétrage.

Le choix d'un codage binaire permet de créer des opérateurs de croisement et de mutation simples. Pour des problèmes d'optimisation de grande dimension, la structure du problème peut être altérée. Les algorithmes génétiques utilisant un codage réel [Goldberg, 1990] évitent ce genre de problème et permet l'optimisation de problème de grande dimension.

La recherche d'une fonction d'adaptation appropriée est très importante pour obtenir un bon fonctionnement de l'AG. Cette fonction représente l'environnement du problème, c'est-à-dire qu'elle permet d'attribuer une valeur à la solution présentée par le génotype (espace de recherche) en fonction du problème à résoudre. La fonction d'adaptation que l'on recherche dépend des critères que l'on veut maximiser ou minimiser. Elle peut devenir complexe si l'on calcule uniquement sa valeur approchée. Elle devrait attribuer des valeurs très différentes aux individus afin de faciliter la sélection. En effet, s'il apparaît des solutions invalides, la fonction d'adaptation doit pouvoir attribuer une valeur pour bien guider le processus de sélection.

Certains processus de sélection sont sensibles aux écarts de la fonction d'adaptation et dans certains cas, on remarque qu'un individu fort risque d'être reproduit trop souvent et ainsi d'éliminer les individus les plus faibles. On obtient dans ce cas une population homogène contenant un seul type d'individu. Le *scaling* et le *sharing* [Hekanaho, 1996], [Goldberg, 1987] peuvent modifier les valeurs d'adaptation afin de réduire ou d'amplifier les écarts entre les individus. L'objectif est de répartir, sur chaque sommet de la fonction à optimiser, un nombre d'individus proportionnel à la fonction d'adaptation associée à ce sommet.

Des améliorations au niveau des opérateurs apparaissent dans certains travaux comme le croisement restreint [Goldberg, 1987] et le contrôle adaptatif de l'opérateur de croisement [Sebag, 1994]. Dans les travaux de Sebag portant sur le contrôle adaptatif, ils associent aux AG un apprentissage inductif dans le but de faire apprendre aux AG les schémas de croisements indésirables en fonction des individus.

3.5. Comparaison des outils d'apprentissage

La comparaison entre les différentes méthodes de métaheuristiques est très rare dans la littérature. Selon la conception du problème et le mode de fonctionnement des algorithmes, on retrouve quelques études de comparaison de performance pour les réseaux de neurones et le recuit simulé en comparaison avec les algorithmes génétiques.

Une étude de [Hao et *al.*, 1999] sur la comparaison des principales métaheuristiques (tableau 3.7), montre leur grande efficacité pour fournir des solutions approchées de bonne qualité pour un grand nombre de problèmes d'optimisation classiques et d'applications réelles de grande taille. C'est pourquoi l'étude de ces méthodes est actuellement en plein développement. Les critères de comparaison retenus sont la simplicité de la méthode, la facilité d'adaptation au problème, la possibilité d'intégrer des connaissances spécifiques du problème, la qualité des meilleures solutions trouvées et la rapidité en terme de temps de calcul nécessaire pour trouver une solution satisfaisante.

	Descente	Recuit simulé	tabou	AG	Hybrides
Simplicité	0	-	-	-	- -
Facilité d'adaptation	0	0	-	-	-
Connaissance	0	0	+	+	+
Qualité	0	+	++	+	++ (+++)
Rapidité	0	-	-	- -	- -

Tableau 3.7. Comparaison générale des principales métaheuristiques (Tiré de Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes, [Hao et al., 1999])

En général, les études de comparaison de performance d'algorithme avec les algorithmes génétiques montrent une meilleure performance et une certaine robustesse à résoudre les problèmes avec l'emploi des AG ainsi que de l'utilisation de méthodes hybrides. Il est important de noter que la qualité des solutions trouvées par une méthode peut être très variable selon l'implémentation réalisée.

Le tableau 3.8 est une proposition de résumé qui montre certains éléments de comparaison des caractéristiques entre les algorithmes mentionnés dans les sections ci-haut.

Caractéristiques	Réseaux de neurones	Algorithmes génétiques	Recuit simulé
Modèle biologique	Système nerveux	Sélection naturelle	Principe de la thermodynamique
Fondement biologique	Apprentissage individuel (cerveau)	Mécanisme d'évolution	Distribution de Boltzmann
Noeud	Neurone	Individu	Configuration
Dynamique d'état	Fonction d'activation (seuil / poids synaptiques)	Selection / Crossing-over / Mutation	Critère de Metropolis / Opérateurs de transformation
Traitement	Parallèle	Parallèle implicite	Séquentiel
Apprentissage et adaptation	Règles d'apprentissage par des exemples	Évolutif	Filtre de hasard
Compréhension du problème	Nécessaire	Non nécessaire	Non nécessaire
Interprétation	Difficile	Facile	Facile
Limitation	Exemples d'apprentissage	Optimisation et paramétrisation	Recherche globale

Tableau 3.8. Comparaison de caractéristiques des métaheuristiques

La particularité des réseaux de neurones est de dégager les relations entre variables par un mécanisme d'apprentissage. La problématique se pose lors du choix de la bonne configuration pour le problème à traiter, car pour un problème donné, il existe plusieurs types d'architecture. De plus, la détermination de la topologie optimale d'un réseau pour un problème quelconque est un problème NP-Complet.

L'algorithme du recuit simulé recherche un optimum local en s'offrant la possibilité de sortir d'une impasse. Mais par rapport au AG, le recuit simulé n'a qu'un seul point de recherche et ne peut échanger des informations entre plusieurs solutions. [Bounsaythip, 1998] montre que les AG sont plus efficaces que le recuit simulé. Et elle fait la conclusion que les AG offrent plus de possibilités d'étendre la méthode vers une stratégie adaptative.

Pour ce qui est de la logique floue, il n'y a pas vraiment de comparaison avec les AG dans la littérature. On remarque beaucoup plus une association des deux méthodes pour former un système hybride [Herrera et Lozano, 1996], [Martin-Bautista et *al.,* 1999]. Plusieurs études favorisent l'émergence de système hybride pour donner une bonne robustesse au système d'apprentissage pour ainsi perfectionner les points faibles d'un algorithme par un ou plusieurs algorithmes d'apprentissage. Par exemple, on retrouve des études de systèmes hybrides par l'utilisation des AG pour optimiser les réseaux de neurones. L'optimisation se fait soit par l'apprentissage de la meilleure architecture du réseau pour traiter le problème [Rovithakis et *al.*, 2003], soit par l'optimisation des meilleures poids d'apprentissage [Maniezzo, 1994],[Montana, 1995].

3.6. Méta-apprentissage avec les algorithmes génétiques

Le MA avec les algorithmes génétiques a fait l'objet d'études depuis les quinze dernières années. [De Jong1988] rapporte que les AG sont plus performants que d'autres techniques d'apprentissage pour la conception de systèmes d'apprentissage. Cette performance est également comparée dans les travaux de [Grenfenstte, 1993] qui présente une rétrospective sur l'application des algorithmes génétiques dans le domaine des algorithmes d'apprentissage. Plusieurs auteurs ont également employé les AG spécifiquement pour l'acquisition de connaissances dans les systèmes de classificateurs [Knight et Sen, 1995], [Fàbrega et Guiu, 1999], [Spears W.M, De Jong, 1990]. [Holmes, 2002] présente un bon exposé sur le sujet dans lequel de nouveaux modèles sont discutés et plusieurs applications intéressantes sont présentées. Des travaux de recherches concernant les systèmes multi-agents proposent aussi d'employer les AG comme système d'apprentissage à l'intérieur des agents intelligents. Cette approche permet aux agents d'être complètement autonomes d'un point de vue génétique [Cadon et *al.*, 2000]. Cette autonomie génétique permet un MA au niveau, par exemple, de la détermination du meilleur jeu d'heuristiques au cœur même des agents.

3.7. Conclusion

Ce chapitre discute du méta-apprentissage et de son rôle comme le processus d'exploitation de la connaissance de l'apprentissage. L'application du MA se fait par l'utilisation d'un seul ou de plusieurs algorithmes d'apprentissage, dont les AG, qui se sont montrés favorables pour la conception d'un tel système. En effet, certains chercheurs favorisent l'utilisation des AG en raison de leurs possibilités pour obtenir rapidement une solution de bonne qualité et ce, assez facilement comparativement à d'autres techniques [Cadon *et al.*, 2000],[Pigeon *et al.*, 2001]. Tout particulièrement, les travaux de [Pigeon *et al.*, 2001] dans un contexte militaire, tendent à démontrer l'avantage des algorithmes génétiques sur les autres techniques d'apprentissage. En effet, ils proposent une approche d'optimisation basée sur les AG d'un système multi-agents de fusion d'information. Ce travail est précurseur à la recherche actuelle et tend à la réalisation d'un système intelligent destiné aux opérations urbaines dans le secteur du commandement et contrôle [Pellerin *et al.*, 2004a], [Pellerin *et al.*, 2004b]. À cette fin, le développement efficace d'un prototype de MA peut être envisagé. Ce prototype implique nécessairement un MA basé sur les AG dans un environnement dynamique.

Chapitre 4
Théories des prototypes proposés

Le chapitre précédant démontre l'utilisation potentielle des algorithmes génétiques pour la réalisation d'un système de méta-apprentissage (MA). En effet, les diverses techniques constituées par le MA ne sont pas toutes compatibles pour un système d'apprentissage évolutif. Une voie adaptative rapide ne peut être envisagée selon une optique probabiliste ou à l'aide de plusieurs cas d'apprentissage. Un système d'optimisation adaptatif des connaissances en lien direct avec l'environnement est donc préconisé à l'aide d'algorithmes génétiques. Pour garantir le succès d'un tel prototype d'apprentissage, les concepts de MA et d'AG doivent être examinés rigoureusement, ce qui implique alors deux perspectives duales l'une de l'autre : MA appliqué aux AG versus les AG appliqués au MA.

4.1. Deux perspectives de méta-apprentissage

Le présent chapitre comportera deux sous-sections importantes impliquant la description de chacune des perspectives de méta-apprentissage (MA). La première section (4.1.1.) fait donc l'élaboration d'un prototype basé sur l'adaptation ou plus spécifiquement sur l'auto-adaptation des algorithmes génétiques vis-à-vis un problème donné. Cette adaptation va se refléter au niveau même de son fonctionnement, soit au niveau de son paramétrage. Pour qu'un algorithme génétique soit en mesure d'optimiser un problème donné, il doit faire l'objet d'un paramétrage. Cette paramétrisation est fortement liée à la nature du problème et affecte directement la solution trouvée.

Pour le deuxième prototype (section 4.1.2.), il y a aussi une implication de l'adaptation, mais elle se concentre sur l'acquisition de MC. Ce prototype fait une analogie avec l'apprentissage du cerveau humain, c'est-à-dire que l'apprentissage est en fait une adaptation ou transformation physique du cerveau. Le cerveau reconstruit ainsi continuellement sa mémoire et cette théorie se nomme le Darwinisme neuronal.

4.1.1. Auto-adaptation des Paramètres de l'Algorithme Génétique (APAG)

L'optimisation des paramètres de l'algorithme génétique est généralement mise au point par essais et erreurs, étant donné l'absence de résultats théoriques disponibles. Le problème qui est soulevé est l'apprentissage du réglage des paramètres de l'AG par l'algorithme lui-même. Cette problématique a fait et fait encore l'objet de nombreuses recherches. Les buts de cette section sont de situer le lecteur sur les approches déjà réalisées et de suggérer un prototype qui semble très prometteur. La prochaine section commence par une introduction sur les principaux paramètres et leurs implications dans l'algorithme génétique. Par la suite, une revue de littérature plus approfondie fera la rétrospective des diverses approches d'apprentissage des paramètres avec quelques exemples de travaux. Pour terminer, une description des principes généraux d'un prototype pour l'auto-adaptation des paramètres sera présentée.

4.1.1.1. Les principaux paramètres

Le mécanisme de base de l'AG dépend du choix de plusieurs paramètres clefs tels que des opérateurs de croisement, des opérateurs de mutation, la probabilité de croisement, la probabilité de mutation, le mécanisme de sélection et la taille de la population. Tous ces paramètres ont un grand impact sur l'exécution et la performance des AG [Gao, 2003], [Gomez et Dasgupta, 2002], [Vasconcelos et *al.*, 2001], [Eiben et *al.*, 1999].

Le paramétrage doit être optimisé pour chaque type de problème traité, ce qui constitue une part importante du travail de l'utilisateur. Ainsi, pour faciliter les efforts de l'utilisateur à trouver le bon paramétrage, certaines études montrent que les paramètres d'un algorithme génétique sont réglés et optimisés par un autre algorithme génétique [Tongchim et Chongstitvatana, 2002], [Mernik et *al.*, 2000], [Lis, 1996], [Pham, 1995], [Freisleben et Hartfelder, 1993], [Grefenstette, 1986], [Mercer et Sampson, 1978].

Dans la pratique, les paramètres des AG sont tout d'abord réglés approximativement selon un standard couramment utilisé pour tester les algorithmes d'optimisation. Effectivement, les études de [De Jong, 1975] ont conduit à la mise au point de standards de paramètres pour les AG. On peut ainsi définir rapidement les paramètres et, par la suite, ajuster les paramètres de façon plus précise, en fonction du problème traité. Ces paramètres standards sont trop généraux et non adaptés à tous les genres de problèmes.

Plusieurs questions se posent alors :

- Quel est le meilleur jeu de paramètres ?

- Quels sont les meilleurs opérateurs génétiques ?

- Quels sont les meilleurs taux associés aux opérateurs génétiques?

Un taux élevé de croisement et un taux bas de mutation pourraient être très bons dans l'exploitation de nouvelles solutions pour les premières générations produites par l'algorithme. Cependant, ces mêmes taux pourraient devenir rapidement défavorables quand l'algorithme est près de la solution optimale, car il n'y a plus d'exploration. Le dilemme exploitation versus exploration demeure très problématique. L'exploitation fait référence à la convergence d'un AG vers une direction de recherche donnée. L'exploitation est guidée par l'opérateur de croisement qui donne une convergence vers une solution unique. Au contraire, l'exploration permet de trouver de nouvelles directions de recherche, qui peuvent contenir de meilleures solutions. L'exploration fait référence à l'opérateur de mutation qui donne une chance à l'AG de trouver mieux s'il converge vers une solution sous-optimale. Exploitation et exploration deviennent vite contradictoires, car si l'algorithme ne fait qu'exploiter les directions qu'il trouve, il convergera trop rapidement vers un minimum local. Au contraire, si il ne fait qu'explorer des directions sans les exploiter, il entraînera la non-convergence de l'algorithme et parcourra l'ensemble de l'espace de recherche. Il faut donc trouver un compromis entre l'exploitation et l'exploration.

Une des solutions potentielles pour déterminer le meilleur ensemble de paramètres réside dans l'utilisation de l'apprentissage. C'est le problème qui est considéré ici : le réglage automatique des paramètres fondé sur l'apprentissage des paramètres d'un AG.

Pour bien comprendre le paramétrage automatique, il faut avoir une bonne compréhension des principaux paramètres que l'on doit régler dans un algorithme génétique. Ces paramètres sont :

- la sélection des individus ;

- la taille de la population ;

- les opérateurs et taux de croisement ;

- les opérateurs et taux de mutation.

Ces paramètres sont repris en détail dans les sections suivantes.

4.1.1.1.1. Sélection des individus

Comment choisir les individus qui ont le plus de chances de générer une solution optimale ?

Le choix d'un individu au détriment d'autres peut avoir comme conséquence de perdre les meilleurs, c'est-à-dire perdre un individu qui pourrait s'améliorer dans les générations suivantes et devenir un bon individu. La sélection se concentre souvent sur un seul aspect des individus et ne tient pas compte de certaines propriétés ou qualités qui peuvent produire un bon individu.

La sélection permet d'identifier statistiquement les meilleurs individus d'une population et d'éliminer partiellement les mauvais. Néanmoins, comme dans la nature, il ne faut pas confondre sélection et élitisme : ce n'est pas parce qu'un individu est bon qu'il survivra nécessairement et de même ce n'est pas parce qu'il est mauvais qu'il doit disparaître. En effet, bien souvent, une espèce «bien adaptée» peut descendre d'un mauvais individu. Il existe différents principes de sélection, dont :

- la *Roulette Wheel selection* ;
- la *Stochastic remainder selection* ;
- la *Stochastic universal selection* ;
- l'élitisme.

Les principes de sélection énumérés ci-dessus risquent de ne reproduire qu'un seul individu. En effet, ces méthodes de sélection sont très sensibles aux écarts de fonction d'adaptation et dans certains cas, un très bon individu risque de se reproduire mieux que les moins adaptés. Cette situation peut même provoquer l'élimination complète des autres individus de la population. On obtient alors une population homogène contenant un seul type d'individu. On peut se retrouver avec un algorithme qui converge prématurément vers un optimum local. De plus, il y a un risque de reproduction des mauvais individus.

Pour éviter ce comportement, d'autres méthodes de sélection ont été développées comme le *ranking,* ainsi que des principes (*scaling*, *sharing*) qui empêchent les individus «forts» d'éliminer totalement les plus «faibles». Le *scaling*, ou mise à l'échelle, modifie la fonction d'adaptation afin de réduire ou d'amplifier artificiellement les écarts entre les individus. De la même façon que le *scaling,* le *sharing* consiste à modifier la fonction d'adaptation utilisée par le processus de sélection. Ainsi l'objectif du *sharing* est de répartir sur chaque sommet de la fonction à optimiser un nombre d'individus proportionnels à la fonction d'adaptation associée à ce sommet. On peut

également modifier le processus de sélection en introduisant des tournois entre parents et enfants, basé sur une technique proche du recuit simulé.

4.1.1.1.2. Taille de la population

Quelle doit être la taille de la population ?

La taille d'une population doit éviter les extrêmes. Par exemple, une population trop petite évoluera probablement vers un optimum local peu intéressant. Ainsi, son espace de recherche ne comportera que très peu de solutions et l'AG ne convergera pas vers une solution optimale. Par opposition, une population trop grande sera inutile car le temps de convergence sera excessif. Le temps excessif est occasionné par le nombre élevé d'évaluations de la fonction d'adaptation à chaque génération. Une population de bonne taille aura pour conséquence de prévenir la convergence prématurée de l'algorithme.

La taille de la population doit être choisie de façon à réaliser un bon compromis entre le temps de calcul et la qualité du résultat. Cependant, il faut tenir compte du fait que cette taille de population dépend des méthodes utilisées (sélection, opérateurs génétiques...), du nombre de variables considérées et de la fonction d'adaptation. Si la fonction à optimiser comporte peu d'optima locaux et un optimum global net, la population nécessaire sera plus petite que dans le cas d'une fonction beaucoup plus compliquée comportant de nombreux optima locaux.

4.1.1.1.3. Opérateurs et taux de croisement

Quel est le meilleur opérateur de croisement et quel est son taux associé?

L'opérateur de croisement réalise la reproduction entre les individus de la population et représente l'étape clef de l'AG. Il peut être effectué de plusieurs manières, mais la plus courante reste le croisement à un point de coupure. Les autres sont :

- le croisement en deux points ;
- le croisement uniforme ;
- le croisement avec des règles (heuristiques) ;
- le croisement arithmétique.

Le choix d'une méthode de croisement est souvent limité par la nature même du problème, sinon c'est à l'utilisateur de faire le bon choix, ce qui n'est pas une tâche simple.

Un taux de croisement élevé peut être bon pour l'exploitation de solutions de même type, surtout vers les dernières générations, pour conserver une bonne convergence vers une solution unique. Par contre, un taux de croisement élevé dans les premières générations peut engendrer soit une convergence prématurée de l'algorithme vers un minimum local ou retarder la convergence, car de bons individus risquent d'être croisés trop rapidement par rapport à l'amélioration que la sélection peut apporter. Un taux faible appliqué tout au long de l'algorithme risque fort de donner une convergence prématurée à cause du faible taux d'échange entre les individus.

Si la probabilité de croisement est de 100%, alors toute la population participe au croisement. Par contre, si elle est de 0%, la nouvelle génération au complet est la copie exacte des individus de l'ancienne population. Ceci ne signifie pas que la nouvelle génération est forcément identique. En effet, la nouvelle population n'a pas seulement été modifiée par l'opérateur de croisement, mais également par l'opérateur de mutation et de sélection, ce qui occasionne des changements dans la nouvelle population.

4.1.1.1.4. Opérateurs et taux de mutation

Quel est le meilleur opérateur de mutation et quel est son taux associé?

L'opérateur de mutation a comme objectif de modifier de façon aléatoire certains individus de la population pour prévenir la convergence prématurée. Plusieurs méthodes existent dans la littérature :

- mutation par inversion de bits ;
- mutation uniforme ;
- mutation dynamique ou non-uniforme ;
- mutation Gaussienne ;
- mutation frontière.

Tout comme le choix d'une méthode de croisement, c'est la nature même du problème qui dicte le choix de l'opérateur de mutation, sinon c'est à l'utilisateur de faire le bon choix.

Un taux de mutation élevé peut être très bon dans l'exploration de nouvelles solutions pour les premières générations de l'algorithme. Par contre, ce même taux est défavorable lorsque l'algorithme est proche de la solution optimale. Un taux de mutation plus petit est requis pour les dernières générations afin d'éviter une solution sous-optimale.

L'application d'un taux de mutation élevé tout au long de l'algorithme occasionne une trop grande diversité génétique, c'est-à-dire que l'exploration de nouvelles solutions est trop grande et que l'algorithme ne peut plus converger. Lorsque le taux de mutation est beaucoup trop faible, la recherche risque de stagner à cause du faible taux d'exploitation. Toutefois, un taux faible permet d'éviter cette dispersion génétique et n'entraîne que quelques modifications sur un nombre limité d'individus.

S'il n'y a aucune mutation, les individus qui sont produits juste après le croisement ne comportent aucun changement. Par contre, si la probabilité de mutation est de 100%, l'individu entier est changé et l'optimisation se transformera en recherche aléatoire.

4.1.1.2. Classification des méthodes de paramétrage

Dans cette section, la classification des différentes méthodes de paramétrage proposées dans la littérature est brièvement passée en revue. Bien que quelques travaux offrent un point de vue différent, la classification la plus fréquemment utilisée est basée sur deux approches principales (montrées à la figure 4.1), soit empirique et adaptative [Eiben et *al.,* 1999], [Lobo, 2000], [Smith et Fogarty, 1997].

Les approches empiriques sont basées sur l'expérimentation et l'observation seulement. Il est impératif de trouver les bonnes valeurs des paramètres avant l'exécution de l'algorithme. Par opposition, les approches par adaptation emploient un paramétrage initial qui est modifié pendant l'exécution de l'algorithme. On retrouve donc par adaptation les bonnes valeurs des paramètres après exécution de l'algorithme. Pour cette dernière, le paramétrage initial n'exige pas la participation de l'utilisateur. La classification de l'approche adaptative se subdivise en deux sous catégories : paramètre adaptatif limité et paramètre auto-adaptatif.

Cette section passe en revue quelques études sur la paramétrisation selon une classification en trois catégories d'approches :

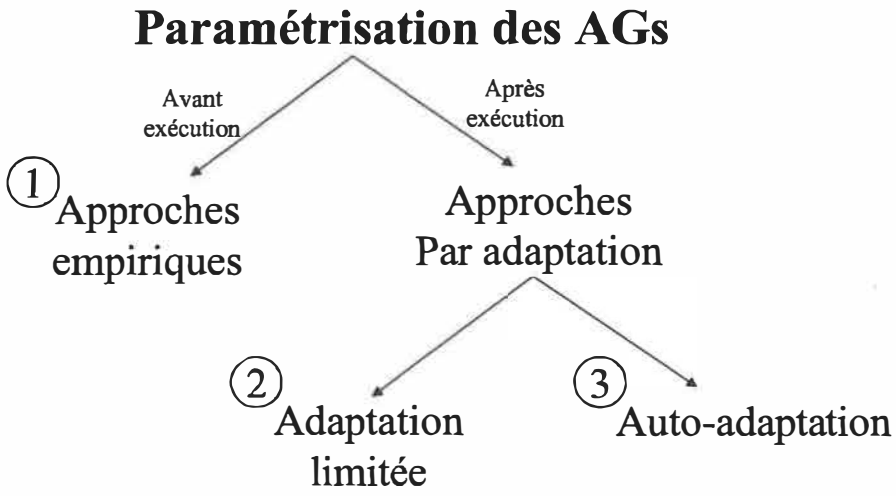


Figure 4.1. Classification des différentes approches d'apprentissage des paramètres des AG

4.1.1.2.1. Approches empiriques

Ces approches reposent fortement sur l'expérimentation et l'observation et elles mesurent la performance de l'AG par essais et erreurs, tout en variant les paramètres de l'algorithme. Il est ainsi nécessaire de trouver de bonnes valeurs pour les paramètres avant d'exécuter l'algorithme. Il est à noter que ces valeurs demeurent constantes pendant l'exécution. Selon la littérature, l'étude de référence pour la paramétrisation avant l'éxécution de l'AG est celle de [De Jong, 1975].

Cette étude montre la performance des AG en testant différentes combinaisons de paramètres sur un jeu de 5 fonctions. Quatre paramètres ont fait l'objet de l'étude : la taille de la population, le taux de croisement, le taux de mutation et la génération *gap*. Le dernier paramètre fait référence à une fraction spécifique de la population qui est remplacée durant le passage à une autre génération. Si la génération *gap* est égale à 1.0, cela signifie que la population entière est

remplacée à chaque génération. Par contre, si la génération *gap* est égale à 0, alors aucun individu de la population n'est remplacé. La population demeure intacte durant le passage à une autre génération. Une faible valeur de génération gap revient au même mode de fonctionnement que la méthode de sélection nommée élitisme. L'élitisme est le nom de la méthode qui copie d'abord le ou les meilleurs individus dans la nouvelle population, puis le reste de la population est construit par le croisement et la mutation. L'élitisme peut rapidement augmenter la convergence de l'AG, parce qu'il empêche ainsi une perte de la meilleure solution trouvée. Aujourd'hui, ces paramètres sont un standard dans l'utilisation des AG. Ils donnent une bonne performance et une solution acceptable.

Taille de la population (individus)	50 à 100
Taux de croisement	0.6
Taux de mutation	0.001
Génération *gap*	1.0
Méthode de sélection	Élitisme

Tableau 4.1. Jeu de paramètres standard selon [De Jong, 1975]

Ces paramètres sont donnés à titre de référence et ne s'appliquent pas à l'ensemble des problèmes. Il faut donc essayer différents paramètres et choisir ceux qui donnent les résultats les plus satisfaisants.

4.1.1.2.2. Approche par adaptation limitée

Les approches par adaptation utilisent une paramétrisation initiale qui est modifiée au cours de l'exécution de l'algorithme. Les approches adaptatives limitées analysent les effets isolés d'un ou deux paramètres sans tenir compte des autres. Elles sont très utiles pour la compréhension de la dynamique des AG. Selon la littérature, [Bäck, 1992] et [Mühlenbein, 1992] montrent tous les deux une étude sur le taux de mutation optimal pour des algorithmes simples du type (1+1), c'est-à-dire une génération produite avec un seul parent qui génère un seul enfant. Le meilleur des deux (parent ou enfant) servira de parent pour la prochaine génération. Les deux auteurs concluent que pour un taux de mutation fixe tout au long de l'exécution, le taux de mutation optimal est de 1/L, où L est la taille du problème.

[Lis, 1996] introduit une technique pour adapter le taux de mutation au modèle des AG parallèles. Plusieurs sous-populations évoluent séparément sur différents processeurs en utilisant différents taux de mutation. Après un certain temps, ces populations sont comparées. Si le meilleur résultat est obtenu d'un processeur avec un taux de mutation élevé, alors les autres taux de mutation des autres processeurs sont augmentés d'un niveau. Si le meilleur résultat est obtenu d'un processeur avec un taux de mutation très bas, alors les taux de mutation des autres processeurs sont diminués d'un niveau.

4.1.1.2.3. Approche par auto-adaptation

L'auto-adaptation des paramètres fait référence aux techniques qui permettent une évolution, une adaptation des divers paramètres de l'AG tout au long de son exécution. Il y a deux mécanismes qui sont utilisés pour caractériser l'approche par adaptation :

1) Un mécanisme basé sur des règles qui se sert des feedbacks de l'état courant de l'espace de recherche pour modifier les valeurs des paramètres ;

2) Un mécanisme qui insère les paramètres dans les chromosomes des individus et dans lequel les changements des valeurs des paramètres sont entièrement dépendants du mécanisme d'évolution.

D'après les études sur l'approche par auto-adaptation, celle de [Grefenstette, 1986] montre une sélection des valeurs des paramètres en utilisant un MA des algorithmes génétiques. Le MA est caractérisé par deux niveaux d'algorithmes génétiques. Le méta-niveau maintient une population de jeux de paramètres. Le niveau de base utilise le jeu de paramètres obtenu du méta-niveau pour résoudre le problème. La mesure de la performance du niveau de base sert de fonction d'adaptation pour le méta-niveau.

Les résultats du tableau 4.2 montrent que l'obtention du meilleur jeu de paramètres n'est que légèrement mieux que le jeu de paramètres standard trouvé par De Jong. Ces résultats donnent le meilleur jeu de paramètres pour optimiser les performances *on-line* de l'algorithme. Le jeu de paramètres entre parenthèse représente les paramètres optimisés pour les performances *off-line* de l'algorithme. L'exécution *on-line* est basée sur la surveillance de la meilleure solution dans chaque génération, alors que l'exécution off-line tient compte de toutes les solutions dans la population.

Paramètre	Grefensette		De Jong
	On-line	Off-line	
Taille de la population (individus)	30	80	50 à 100
Taux de croisement	0.95	0.45	0.6
Taux de mutation	0.01	0.01	0.001
Génération nulle	1.0	0.9	1.0
Méthode de sélection	élitisme	non élitisme	élitisme

Tableau 4.2. Comparaison des meilleurs jeux de paramètres selon [Grefenstette, 1986] et [De Jong, 1975]

[Pham, 1995] propose une technique de sélection de paramètres par établissement d'une compétition entre plusieurs sous-populations qui utilisent différents jeux de paramètres. Plusieurs populations indépendantes évoluent dans un même environnement en utilisant leurs propres jeux de paramètres. Ces populations sont en compétition pour un seul processeur. Ainsi, les populations qui ont un meilleur jeu de paramétrage vont recevoir plus de temps processeur pour évoluer.

Les auteurs [Smith et Smuda, 1995] suggèrent un algorithme pour l'ajustement automatique de la taille de la population au cours de l'exécution de l'AG. Ils proposent un algorithme qui requiert de la précision de la part de l'utilisateur pour l'ensemble de la recherche et ils se basent sur une différence de fonction d'adaptation pour contrôler la sélection de la taille de la population. Ils constatent que généralement l'utilisateur ne connaît pas ce que la taille de la population peut avoir comme effet ou comment elle peut affecter la qualité de la solution du problème. L'algorithme est très limité, car il n'est pas facile de spécifier la précision de l'utilisateur. De plus, ils remarquent que l'estimation de la variance de la fonction d'adaptation est biaisée par le nombre d'échantillons.

4.1.1.3. Principes généraux du prototype APAG

Les AG sont un modèle de génétique artificielle qui s'inspire de près de l'évolution des êtres vivants et de la nature. Partant de ce fait, on peut émettre l'hypothèse que la conception d'une approche de paramétrisation des AG devrait être le plus près possible du modèle évolutif naturel. Malgré de nombreuses approches de paramétrisation pour augmenter la performance des AG, seules certaines approches ont leur contrepartie dans la nature en considérant les AG comme une

mécanique évolutive. Le principe sur lequel reposent les AG est très simple à mettre en œuvre dans certains cas simplistes, mais pour un problème plus complexe, les AG restent souvent mal compris car il n'est pas habituel de penser en terme d'évolution naturelle. Le but de l'amélioration des AG est de réaliser une mécanique évolutive artificielle dont le système maîtrise totalement les paramètres. Un des objectifs complémentaire à cette recherche est d'améliorer notre compréhension des processus naturels d'adaptation et de concevoir des systèmes artificiels possédant des propriétés similaires aux systèmes naturels (voir chapitre 8 pour plus de détails sur une approche hypothétique plus près du modèle de la nature).

La nature dote les individus d'une certaine intelligence, d'un certain contrôle. Certains auteurs suggèrent que le contrôle des paramètres de l'algorithme génétique pourrait être codé dans le chromosome de chaque individu de la population [Gomes et *al.*, 2003], [Hinterding 1997], [Spears 1995], [Bagley, 1967]. Ceci favorise donc un mécanisme qui insère les paramètres dans les chromosomes des individus et les changements des valeurs des paramètres sont entièrement dépendants du mécanisme d'évolution. On peut ainsi retrouver à l'intérieur de chaque individu un système d'apprentissage de paramètres régis par un algorithme génétique.

Dans plusieurs travaux, on retrouve un but commun, soit d'optimiser une population de jeux de paramètres. L'idée première est l'utilisation, à un niveau supérieur, d'un algorithme génétique. Les applications de ce second niveau d'AG se sont réalisées soit par une compétition d'algorithmes génétiques ayant des jeux de paramètres différents [Tongchim et Chongstitvatana, 2002], [Lis, 1996], [Pham, 1995], soit par méta-AG où l'on retrouve deux niveaux distincts d'algorithmes génétiques : l'un pour résoudre le problème posé et l'autre appliqué pour optimiser les jeux de paramètres [Mernik et *al.*, 2000], [Freisleben, 1993], [Grefenstette, 1986], [Mercer et Sampson, 1978]

Le prototype proposé ici se distingue des autres travaux par son aspect autonome, car on implante dans un individu un algorithme génétique pour l'apprentissage de nouveaux jeux de paramètres, ainsi qu'un mécanisme de renforcement pour l'apprentissage du meilleur jeu de paramètres en lien avec l'environnement. L'environnement est représenté par la population dans laquelle l'individu évolue et l'apprentissage dépend entièrement du mécanisme d'évolution.

Le MA est donc appliqué dans la conception d'un individu autonome capable d'apprendre et de prendre des décisions en fonction des contraintes de l'environnement et de la fonction d'adaptation. Ce prototype sera expliqué plus en détail dans le prochain chapitre.

4.1.2. Méta-Apprentissage utilisant les Algorithmes Génétiques (MAAG)

Est-ce possible de construire un système de MA qui a la capacité d'apprendre par lui-même?

La réponse du point de vue des systèmes adaptatifs semble positive du fait que ces systèmes améliorent leurs propres capacités au cours du temps et s'adaptent constamment à leur environnement spécifique.

Jusqu'à présent, les AG qui sont des algorithmes adaptatifs ont souvent été associés aux systèmes artificiels. Cependant, [Kosorukoff et Goldberg, 2002] ont démontré que les AG peuvent s'intégrer à résoudre des problèmes dans des systèmes vivants, soit dans un monde réel et dynamique. Un monde réel apparaît complexe, irrégulier, non-linéaire et inséparable. Cependant, un système artificiel est prédictible, régulier, linéaire et déterministe. L'une des forces des AG est donc de résoudre des problématiques qui proviennent de systèmes complexes. Ceci vient appuyer l'usage des AG dans la conception d'un système de MA.

Pour répondre complètement à la question posée, il est important de souligner ici les étapes de réflexion pour la conception du prototype MAAG. C'est sur ces pistes de réflexion que reposent les bases du système. Pour bien comprendre le cheminement des réflexions faites, cette section est subdivisée en trois sous-sections dont chacune élabore sur une relation entre le MA et les diverses composantes qui la constituent.

- la relation MA et son environnement en tant que système complexe ;
- la relation MA et l'apprentissage lui-même ;
- la relation MA et intelligence humaine (le cerveau).

Après l'analyse des relations, ce chapitre se terminera par une proposition d'un prototype de MA qui utilise les algorithmes génétiques.

4.1.2.1. Méta-apprentissage et environnement

Dans la conception d'un système de MA, plusieurs questions viennent tout naturellement à l'esprit, surtout si le système évolue dans un environnement dynamique. Par exemple :

- Comment construire un système d'apprentissage capable d'agir de manière autonome dans un système complexe ?

- Comment apprendre à apprendre à un algorithme d'apprentissage ?
- Comment concevoir un algorithme de MA qui apprend mieux qu'un algorithme d'apprentissage simple ?
- Comment exploiter les connaissances de l'apprentissage pour améliorer les performances des algorithmes d'apprentissage ?

De plus, on identifie plusieurs conditions nécessaires à la conception d'un système de MA surtout si le but est de démontrer les capacités d'apprendre à apprendre des algorithmes d'apprentissage. On peut ainsi résumer les conditions de la façon suivante :

Le système doit :

- apprendre à s'adapter rapidement aux environnements changeants et inconnus ;
- assurer une certaine qualité des meilleures solutions trouvées ;
- converger rapidement vers une solution ;
- s'adapter à l'environnement pour permettre un apprentissage optimal ;
- prévoir le cas de contextes non prévus ;
- envisager le cas d'informations erronées ou absentes ;
- garantir la fiabilité du système dans le temps ;
- proposer des actions qui sont le résultat de ses simulations les plus prometteuses ;
- anticiper les états possibles de l'environnement ;
- assurer une possibilité de correction des paramètres et de retour si l'information est disponible ;
- tenir compte des possibilités d'interaction du système dans un univers à plusieurs dimensions ;
- analyser les informations et enregistrer leurs interactions dans l'environnement ;

- être en mesure de placer une valeur de confiance sur une information ;
- naviguer d'un niveau d'abstraction à un autre, selon la précision des informations ;
- intégrer l'indexation spatio-temporelle de l'information ;
- extraire des connaissances à partir des informations ;
- utiliser les mécanismes de raisonnement humain.

La liste des conditions ci-dessus n'est qu'une liste partielle, mais elle devient très vite lourde pour la réalisation d'un système. Cette lourdeur et la complexité des conditions répondent au besoin de concevoir un MA selon un modèle de système complexe.

L'une des pistes de solutions possibles est de faire le parallèle (tableau 4.3) avec les systèmes multi-agents (SMA). En partant du fait qu'un système complexe est constitué d'entités autonomes et que ces entités interagissent et évoluent dans un environnement dynamique, on peut facilement faire un lien avec un système multi-agents. En effet, un SMA est constitué d'entités autonomes qui sont des agents. De plus, ces agents sont en interaction étroite avec leur environnement [Jennings, 2000]. On décompose alors le système complexe en sous-systèmes, puis en plusieurs composantes de sous-système. Par analogie, on décompose le système multi-agents en organisation des agents, puis en plusieurs agents. Jennings, Sycara et Wooldridge [Jennings, 1998] ont proposé la définition suivante pour un agent :

Un agent est un système informatique, **situé** dans un environnement, et qui agit d'une façon **autonome** et **flexible** pour atteindre les objectifs pour lesquels il a été conçu.

Les notions situé, autonomie et flexible sont définies comme suit :

- **situé** : l'agent est capable d'agir sur son environnement à partir des entrées sensorielles qu'il reçoit de ce même environnement ;
- **autonome** : l'agent est capable d'agir sans l'intervention d'un tiers (humain ou agent) et contrôle ses propres actions ainsi que son état interne ;

- **flexible** : l'agent dans ce cas est :

    - **capable de répondre à temps** : l'agent doit être capable de percevoir son environnement et élaborer une réponse dans les temps requis ;

    - **proactif** : l'agent doit exhiber un comportement proactif et opportuniste, tout en étant capable de prendre l'initiative au "bon" moment ;

    - **social** : l'agent doit être capable d'interagir avec les autres agents (logiciels et humains) quand la situation l'exige afin de compléter ses tâches ou d'aider ces agents à accomplir les leurs.

Système complexe	Système multi-agents
Sous-systèmes	Organisation des agents
Composants de sous-système	Agents
Interactions entre sous-système et composants de sous-système	Coopération pour atteindre un objectif commun Coordination de leurs actions Négociation pour résoudre un conflit
Relation entre sous-système et composants de sous-système	Mécanisme implicite pour la représentation et la gestion des relations de structure d'organisation pour un modèle collectif

Tableau 4.3. Analogie du système complexe versus un système multi-agents (Adapté de "On Agent-Based Software Engineering" [Jennings, 2000])

En se basant sur cette définition d'agent, un modèle d'apprentissage d'un système peut être proposé ce qui deviendra un guide pour la conception d'un système de méta-apprentissage. La figure 4.2 montre la schématisation d'un système d'apprentissage qui agit d'une manière *autonome* et *flexible,* le tout *situé* dans un environnement.

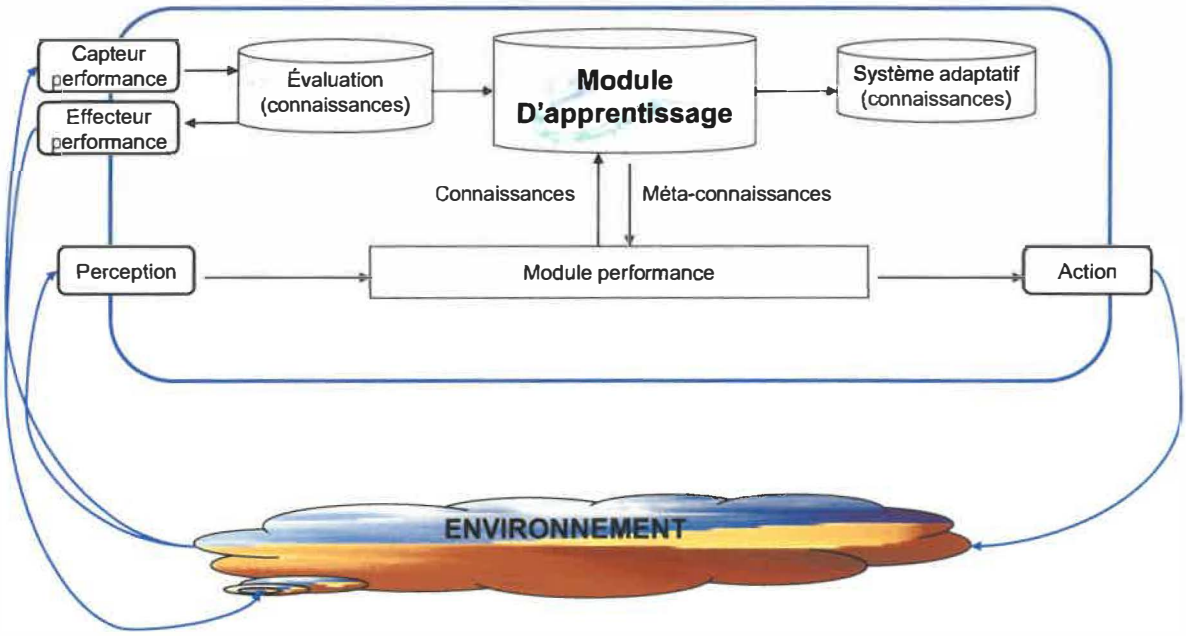


Figure 4.2. Schématisation d'un système d'apprentissage incluant différents modules

- **Module performance :** Prend les perceptions et choisit les actions.
- **Module apprentissage :** Améliore le système par modification du module de performance en utilisant les connaissances et la rétroaction.
- **Module d'évaluation :** Juge la qualité des connaissances en fonction de leur degré de contribution à la résolution du problème.
- **Module de système adaptatif :** Explore de nouvelles connaissances.

Pour la conception d'un système de MA, le système doit reposer sur les caractéristiques des différents modules. Les modules de performance, d'évaluation et de système adaptatif sont des éléments clef du système global qui aide le module d'apprentissage dans son apprentissage. Le

système est donc entièrement centré sur le module d'apprentissage qui est la pièce maîtresse. Mais comment peut-on définir et concevoir ce module si ce n'est que par une bonne compréhension de l'apprentissage ? La prochaine section approfondira davantage cette question.

4.1.2.2. Méta-apprentissage et apprentissage

Qu'est ce que l'apprentissage ? Comment l'être humain apprend-il ?

L'apprentissage utilise les connaissances acquises pour entraîner le cerveau et accélérer la prise de décision. L'apprentissage agit donc au cœur du cerveau. Selon certains chercheurs, il y a une transformation physique du cerveau qui survient lors du processus d'apprentissage. En référence à la neurobiologie, cette transformation est en fait représentée par une sélection de connections neuronales. Selon la situation d'apprentissage, le cerveau réagit différemment et, selon le cas échéant, il sélectionne quelques connexions, en renforçant certaines alors que d'autres disparaissent. Le cerveau reconstruit continuellement sa mémoire lorsqu'un stimulus externe ou interne se produit. Le cerveau est donc un système dynamique, un système complexe.

Lorsque le cerveau humain apprend, différentes zones des hémisphères cérébraux sont utilisées selon le contexte de l'apprentissage. Les niveaux d'activité dans le cerveau ne sont pas identiques dans tous les hémisphères. Plusieurs réseaux de connexions de neurones sont plus fréquemment choisis que d'autres et le tout dépend fortement des stimuli de l'environnement.

Il est possible de voir ces transformations biologiques s'opérant dans le cerveau en utilisant un *PET scan* (Positon Emission Tomography). C'est une technique d'imagerie très importante qui a fait son apparition dans les années '70. La Tomographie à Émission de Positons est un outil d'exploration fonctionnelle in vivo. Elle permet de mener des études physiologiques portant sur le métabolisme, le débit sanguin (consommation en oxygène) et la synthèse protéique. Elle utilise des éléments radioactifs émetteurs β+, l'objectif étant de localiser ces désintégrations afin de cartographier la concentration et la distribution d'une substance injectée dans le corps et marquée par ces traceurs radioactifs. Dans le cas de l'apprentissage, cette technique mesure la quantité de glucose dans le cerveau. Les niveaux de glucose diffèrent avec les niveaux d'activité d'apprentissage et ainsi on peut indiquer quels secteurs du cerveau sont en activité dans le cerveau. La figure 4.3 montre des images d'activité d'apprentissage dans un cerveau humain [Mier et *al.,* 1998]. Plus l'activité est élevée, plus les taches observables sont de couleur rouge. Le but de ces travaux était d'évaluer l'activité du cerveau pendant l'exécution d'un traçage de labyrinthe. Trois aspects ont été abordées : l'identification des secteurs du cerveau impliqués dans les différentes tâches de traçage, la recherche sur les différences et les similitudes dans ces secteurs liés à l'exécution de la main dominante et non dominante et, finalement, l'effet de

l'apprentissage dans ces secteurs. Un échantillon de 32 sujets droitiers a été choisi pour tracer avec la main droite dominante (16 sujets) ou la main gauche non dominante (16 sujets) sans interruption sur des modèles de labyrinthe avec leurs yeux fermés. Ceci se déroulait pendant un balayage de Tomographie à Émission de Positons (PET) pour mesurer le débit sanguin des sujets.

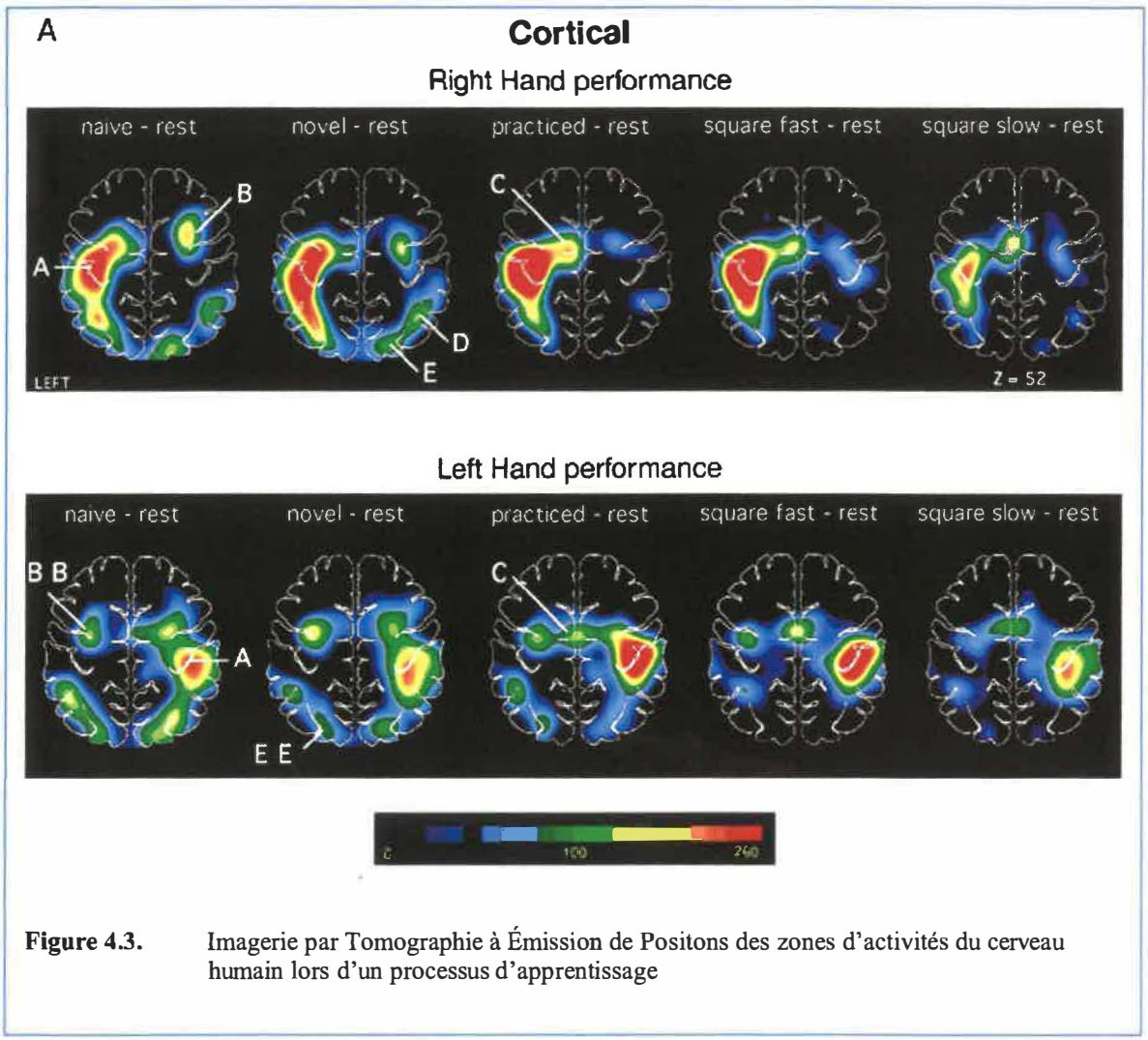


Figure 4.3. Imagerie par Tomographie à Émission de Positons des zones d'activités du cerveau humain lors d'un processus d'apprentissage

La figure 4.3 montre que l'activité corticale change durant le processus d'apprentissage. Différents réseaux de connexions neuronales sont activés, ce qui suggère que l'apprentissage est

en fait une forme de sélection. Cette affirmation sera vérifiée par une étude sur la relation du MA avec le fonctionnement du cerveau ou, en d'autres mots, avec l'intelligence (voir section suivante).

4.1.2.3. Méta-apprentissage et intelligence

L'intelligence fait intervenir le concept de la compréhension et implique un fonctionnement actif de notre cerveau comme système d'apprentissage. Le cerveau humain est donc en contact permanent avec lui-même et les différentes zones qui le constituent pour effectuer l'apprentissage. Comme décrit dans la section précédente, il y a des transformations physiques qui s'opèrent dans ces zones. Ces zones dialoguent entre elles sous forme de mécanismes de réintroduction à l'intérieur même des réseaux de neurones. À chaque instant, il sélectionne un flux de données, en consolide certains, en efface d'autres. Des groupes de neurones se renforcent, d'autres s'affaiblissent en fonction des stimuli de l'environnement. Le cerveau reconstruit ainsi continuellement sa mémoire et, par le fait même, il construit à chaque instant un nouveau monde possible. Le cerveau peut alors fonctionner comme un système sélectif en variant les connexions neuronales, et le processus d'apprentissage devient en fait une forme de sélection. Cette théorie se nomme le Darwinisme neuronal et on peut ainsi élaborer un système d'apprentissage qui repose sur cette théorie. En effet, [Edelman, 1987], [Edelman, 1989], [Edelman, 1992], propose un système d'apprentissage sélectif avec ajustement *a posteriori* : la théorie de la sélection des groupes neuronaux (Darwinisme neuronal). La théorie qu'il a développée repose sur trois postulats fondamentaux :

> **1)** Les neurones se connectent d'abord au hasard puis de plus en plus systématiquement, pour répondre à des contraintes très générales de développement. En effet, au cours du développement cérébral de l'embryon, un schéma de connexions extrêmement variables et individualisées se forme entre les neurones du cerveau.
>
> **2)** Les connexions les plus utilisées se renforcent, d'autres disparaissent. Après la naissance, un schéma de connexions neuronales apparaît pour chaque individu, mais, en réponse aux stimulations sensorielles reçues par le cerveau, certaines combinaisons de connexions sont sélectionnées plutôt que d'autres.
>
> **3)** La réintroduction se produit lorsqu'un stimulus, externe ou d'origine interne, est reçu par l'organisme. Des cartes différentes sont alors excitées en même temps. Des millions de neurones s'activent ainsi en parallèle, s'auto-informant les uns les autres. La sélection de neurones surviendrait plus spécifiquement au niveau de groupes de neurones reliés en

couches, ou cartes, et ces cartes dialogueraient entre elles afin de constituer des catégories de choses et d'événements.

Edelman accorde une importance particulière à la morphologie du cerveau ainsi qu'au principe d'organisation. Il affirme qu'il ne peut exister de connexions spécifiées de façon figée et que le fait que certains principes du développement introduisent une variabilité dans les connexions neuronales confère au cerveau une structure dynamique et non câblée, à la manière d'un ordinateur.

Edelman a également élaboré des modèles informatiques comme une suite d'automates darwiniens capables d'auto-perfectionnement (Darwin I à IV). Certains auteurs adaptent cette théorie dans leurs travaux, comme [Popescu-Belis, 1997] qui propose un système multi-agents adaptatif basé sur la théorie de la sélection des groupes neuronaux. Le système résultant actionne la catégorisation perceptuelle d'un environnement simple, alors que son apprentissage est basé sur des liens de réintroduction entre les agents du dispositif de commande. La stabilité et l'adaptation sont confirmées par expérimentation.

4.1.2.4. Principes généraux du prototype MAAG

Les trois dernières sections proposent des éléments pour concevoir un système de méta-apprentissage. Dès la première relation entre le MA et son environnement, l'aspect modulatoire du système d'apprentissage est repris. De la seconde et troisième relation entre le MA et les éléments proposés (apprentissage et intelligence), une résultante importante de l'apprentissage est dégagée : les connaissances. En effet, ces connaissances sont le reflet même du système d'apprentissage. Par ce fait, le MA est étroitement lié au processus d'acquisition et d'exploitation de la MC [Giraud-Carrier, 2004]. Le traitement des MC permet donc à un système d'observer son propre fonctionnement afin d'en tirer profit. Un tel système utilise une représentation de lui-même pour raisonner sur son propre comportement.

Pour combiner tous ces éléments, une structure d'optimisation doit être mise en place. Des méthodes d'optimisation *a priori* et *a posteriori* peuvent être utiles dans la conception d'un tel système. Donc, une approche qui combine le concept d'une notion *a priori* pour définir les MC du système et une notion *a posteriori* pour affiner les MC par simulations peut être proposée. L'approche par simulation consiste à réaliser un ajustement des MC en fonction des résultats issus de simulations. Cet aspect permet au système de raisonner sur ces connaissances et d'explorer d'autres possibilités dans le cas où le système ne peut pas affiner les MC dynamiquement avec l'environnement.

- ***A priori***

  La notion *a priori* consiste, à partir de données prévisionnelles ou réelles (informations et connaissances), à adapter ces paramètres pour développer les MC. Le système doit donc extraire les MC des connaissances à l'aide de systèmes hybrides d'algorithmes d'apprentissage.

- ***A posteriori***

  La notion *a posteriori* implique la simulation et le choix d'une MC parmi un ensemble de MC possibles. Cette notion est stratégique car elle permet au système de raffiner ces MC et d'en explorer de nouvelles. Elle confère au système une certaine autonomie, ce qui lui permet de réagir aux situations imprévues.

Le cœur de l'apprentissage du système est en fait une forme de sélection et comporte un grand nombre de schémas de connexions neuronales évolutives. Cette évolution est basée sur les algorithmes génétiques et le principe de sélection se déroule plus spécifiquement au niveau de groupes de neurones reliés en couches qui s'informent entre elles afin de constituer des catégories. Cet apprentissage est un module en soi et sera décrit dans le prochain chapitre, ainsi que le reste du système.

4.2.Conclusion

Ce chapitre démontre que pour élaborer un nouveau système de méta-apprentissage en utilisant les AG, l'apprentissage doit s'inspirer des systèmes vivants et de leurs mécanismes spécifiques. Il est important pour un algorithme d'apprentissage d'avoir la capacité de maîtriser son propre fonctionnement. Cette qualité contribuera à une meilleure performance du système global. Ce chapitre montre également le lien étroit entre l'apprentissage et la méta-connaissance. Cette MC est en fait la résultante des expériences acquises et des connaissances antérieures. Le méta-apprentissage prend alors un sens si on utilise ces MC en se posant des questions *a priori* et en utilisant un processus de simulation *a posteriori*.

Chapitre 5
Description des prototypes

Ce chapitre présente les outils essentiels qui permettent de concevoir les prototypes présentés brièvement dans le chapitre 4. Une attention particulière est donnée à la façon dont les algorithmes fonctionnent, incluant une description des éléments clefs de construction des algorithmes. Ce chapitre suit l'esprit du précédent chapitre en terme de présentation sous deux sections majeures. L'une porte sur l'architecture du prototype par **A**uto-adaptation des **P**aramètres de l'**A**lgorithme **G**énétique (APAG) et l'autre traite de l'architecture du prototype par **M**éta-**A**pprentissage des **A**lgorithmes **G**énétiques (MAAG).

5.1. Prototype APAG

Le prototype par Auto-adaptation des Paramètres de l'Algorithme Génétique (APAG) est, comme le mentionne la section 4.1 du chapitre 4, un méta-apprentissage (MA) essentiellement appliqué dans la conception d'un individu autonome capable d'apprendre et de prendre des décisions en fonction des contraintes de l'environnement et de la fonction d'adaptation. L'apprentissage des paramètres est auto-adaptatif, car les paramètres de l'AG sont ajustés par l'évolution des individus. L'apprentissage est centré sur l'individu et les paramètres sont inscrits dans le chromosome des individus. En comparaison avec le chromosome de l'AG traditionnel, le chromosome artificiel du prototype ressemble de près au chromosome naturel. Cette ressemblance se réalise au niveau des délimitations précises du chromosome et par une distinction plus marquée de l'espace de recherche (génotype) et de l'espace de solution (phénotype) (figure 5.1). Cette distinction donne plus de flexibilité à l'algorithme du prototype, contrairement à l'AG traditionnel où la séparation entre génotype et phénotype est faible. En effet, le mécanisme d'action d'un AG est focalisé sur l'individu (recherche d'une solution optimale), en d'autres mots, il s'agit d'une focalisation d'un point de vue plus phénotypique.

Figure 5.1. Distinction entre génotype et phénotype au niveau du chromosome naturel

D'un point de vue biologique, le génotype est plus associé au bagage génétique d'un individu et représente l'ensemble des gènes qui désignent un ou plusieurs caractères particuliers. Le phénotype, pour sa part, est la manifestation visible ou le résultat observable qui est déterminé par le génotype.

5.1.1. Détails d'implantation et de mise en œuvre

L'auto-adaptation des paramètres fait référence aux techniques qui permettent une évolution, une adaptation des divers paramètres de l'AG tout au long de son exécution. **Il repose sur un mécanisme qui insère les paramètres dans les chromosomes des individus, et les changements des valeurs des paramètres sont entièrement dépendants du mécanisme d'évolution.** Le MA est donc appliqué dans la conception d'un individu autonome capable d'apprendre et de prendre des décisions en fonction des contraintes de l'environnement et de la fonction d'adaptation.

Le prototype se distingue des autres travaux déjà énoncés dans le chapitre 4 par son aspect d'autonomie. Cette autonomie est réalisée par l'implantation dans un individu d'un algorithme génétique pour l'apprentissage de nouveaux jeux de paramètres, ainsi qu'un mécanisme de renforcement pour l'apprentissage du meilleur jeu de paramètres en lien avec l'environnement. L'environnement est représenté par la population dans laquelle l'individu évolue et est en compétition avec les autres (Figure 5.2).

Ainsi, l'individu est basé sur deux niveaux d'apprentissage. Sur le premier niveau, un algorithme génétique est appliqué pour l'apprentissage de nouveaux jeux de paramètres. Ces nouveaux jeux de paramètres sont placés au niveau du chromosome afin d'être classés ultérieurement par renforcement. Ceci entraîne une augmentation de l'adaptation de l'individu au problème à résoudre. Dans le second niveau, un apprentissage par renforcement est utilisé comme moyen d'apprentissage du meilleur jeu de paramètres selon le contexte du problème. Sur la figure 5.2, chacun des jeux de paramètres reçoit une cote d'évaluation en fonction de la valeur d'adaptation de l'individu. L'individu évalue alors la qualité des jeux de paramètres selon leur degré de contribution à la résolution du problème présent.

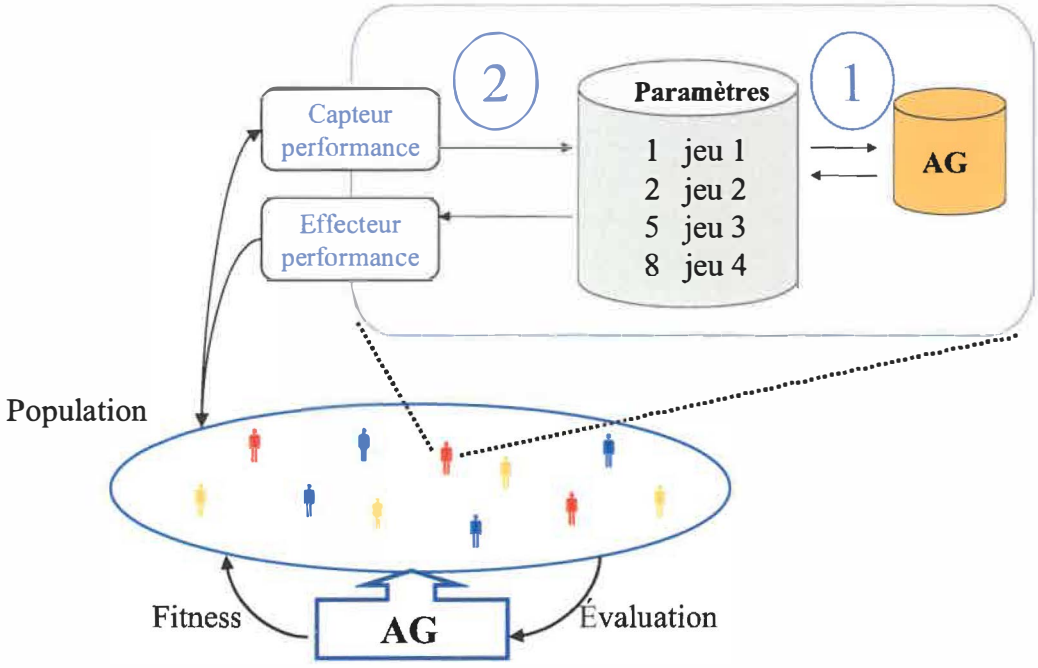


Figure 5.2. Représentation de l'auto-adaptation des paramètres de l'AG. Deux niveaux d'apprentissage : 1. Apprentissage de nouveaux jeux de paramètres. 2. Apprentissage du meilleur jeu de paramètres

D'un point de vue fonctionnel, le développement général de l'algorithme est semblable à l'algorithme génétique, sauf pour les étapes supplémentaires (voir le pseudo code de l'AG du chapitre 2 pour fins de comparaison). Le prototype par auto-adaptation des paramètres est donc une version modifiée de l'algorithme génétique et se déroule comme sur les figures 5.3 et 5.4 suivantes :

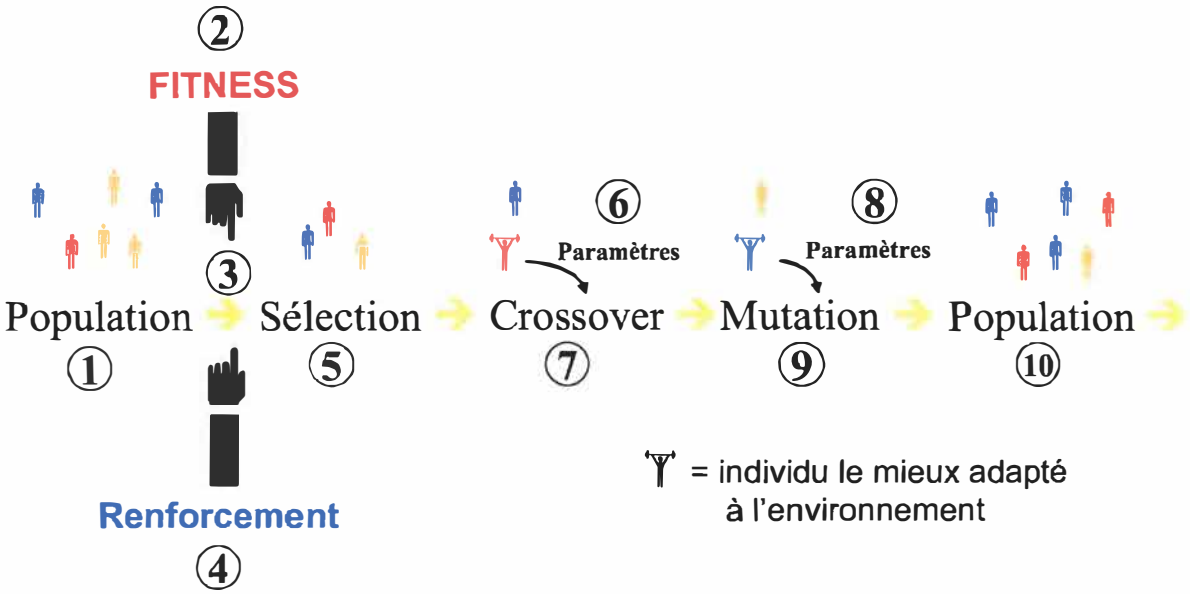


Figure 5.3. Représentation du déroulement de l'algorithme génétique par auto-adaptation (suite à la figure 5.4)

1. **[Initiation]** Générer aléatoirement une population de n individus (les solutions du problème).
2. **[Fitness]** Evaluer la fonction d'adaptation $f(x)$ de chaque individu x dans la population.
3. **[Nouvelle population]** Créer la nouvelle population en répétant les étapes suivantes jusqu'à ce que la population soit complète.
4. **[Renforcement]** Utiliser un apprentissage par renforcement pour déterminer le meilleur jeu de paramètres selon le contexte du problème pour chaque individu.
5. **[Sélection]** Sélectionner deux parents à partir de la population selon la valeur de la fonction d'adaptation. (La meilleure valeur a plus de chance d'être sélectionnée).
6. **[Best crossing-over]** Recherche dans l'espace des opérateurs de croisement le meilleur jeu de paramètres pour l'individu, selon le contexte du problème et la fonction d'adaptation.
7. **[Crossing-over]** Croiser les parents en utilisant le jeu de paramètres de l'individu ayant la meilleure valeur de la fonction d'adaptation associée pour donner les enfants. S'il n'y a pas de croisement, les enfants sont la copie identique des parents.
8. **[Best mutation]** Recherche dans l'espace des opérateurs de mutation le meilleur jeu de paramètres pour l'individu selon le contexte du problème et la fonction d'adaptation.
9. **[Mutation]** Appliquer l'opérateur de mutation sur les enfants avec la probabilité de mutation de l'individu ayant la meilleure valeur de fonction d'adaptation associée.
10. **[Accepter]** Placer les enfants dans la nouvelle population.

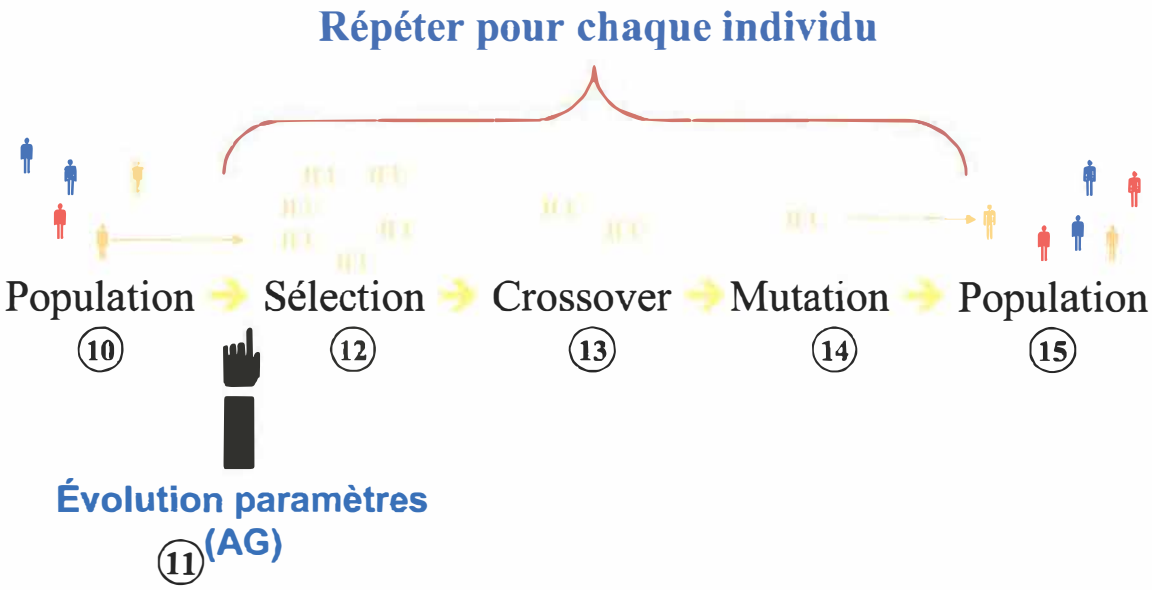


Figure 5.4. Représentation du déroulement de l'algorithme génétique par auto-adaptation (suite de la figure 5.3)

11. **[Évolution des paramètres]** Créer une nouvelle population de jeux de paramètres pour chaque individu en répétant les étapes suivantes jusqu'à ce que la population soit complète.
    12. **[Sélection]** Sélectionner deux jeux de paramètres à partir de la population selon la valeur de renforcement.
    13. **[Crossing-over]** Croiser les jeux de paramètres en utilisant le jeu de paramètres de l'individu. S'il n'y a pas de croisement, les enfants sont la copie identique des parents.
    14. **[Mutation]** Appliquer la mutation sur les enfants avec le jeu de paramètres de l'individu.
    15. **[Accepter]** Placer les enfants dans la nouvelle population.
16. **[Remplacer]** Utiliser la population générée pour exécuter l'algorithme.
17. **[Test]** Si la condition d'arrêt est satisfaite, **stop**, et retourner à la meilleure solution.
18. **[Boucle]** Aller à l'étape **2.**

Dans le pseudo-code précédent, il est important de mentionner que le jeu de paramètres qui sert à l'évolution des populations, aussi bien comme solution que comme jeu de paramètres, est donné par l'individu qui est le mieux adapté à l'environnement. Ce même individu transmettra son

bagage génétique, soit le jeu de paramètres, à la prochaine génération. Ce jeu de paramètres sert donc au fonctionnement de l'algorithme génétique de base.

5.1.2. Structure de l'individu

Le prototype proposé repose sur un individu ayant une structure différente des individus des AG traditionnels. Cette structure est caractérisée par une notion de gènes bien précise. En parallèle avec la structure chez les êtres vivants, la notion de gènes chez une espèce vivante se définit comme une unité héréditaire occupant un espace précis situé sur un chromosome. Un chromosome comporte une série de segments qui ne codent pas pour des protéines et d'autres segments qui forment les gènes de structure. Chaque gène de structure est bordé par une région d'initialisation du début de la transcription et se finit par une région de terminaison de transcription. Au début de chaque gène de structure, il y a présence d'un gène régulateur chargé de contrôler si la transcription du gène de structure s'effectue ou non. À l'intérieur des gènes de structure, on retrouve une alternance de séquences d'introns et d'exons. Les exons constituent une section du gène qui code pour des protéines qui se nomme région codante. Les exons sont séparés par des régions non codantes appelées introns (Figure 5.5). Les introns ne réalisent pas le transfert de l'information génétique ADN en protéine.

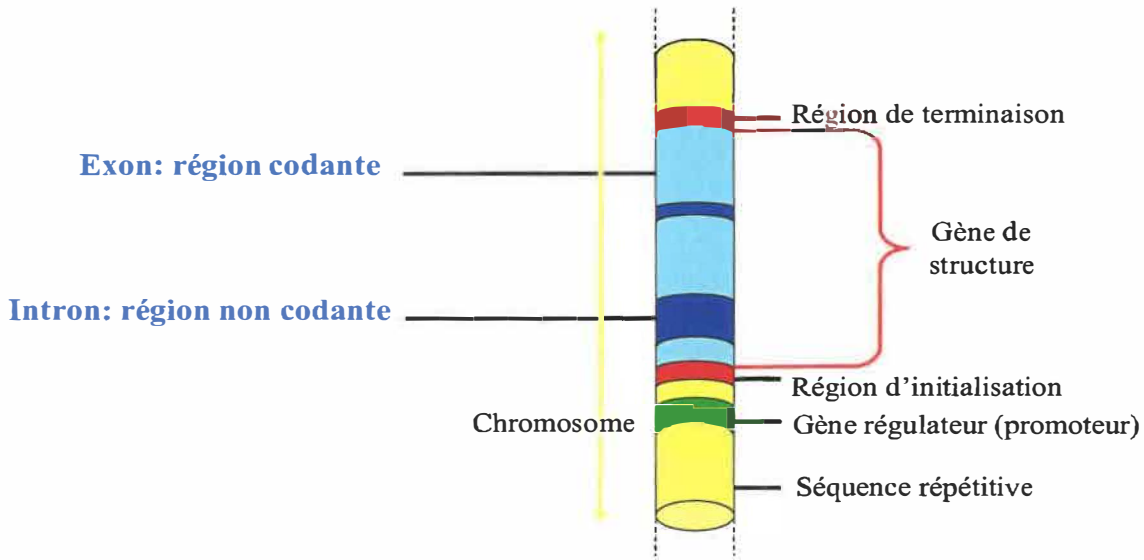


Figure 5.5. Représentation de l'intron et de l'exon dans un gène chez un être vivant (Tiré de Biologie, évolution, diversité et environnement [Mader, 1987])

Ces dernières années, le concept de l'intron et de l'expression du gène ont été incorporés à plusieurs recherche sur les algorithmes génétiques. Ainsi, plusieurs travaux ont repris le fonctionnement du gène naturel et ont modifié les AG en les composant de gènes mobiles, de régions non codantes (introns) et par l'ajout de gènes régulateurs [Chen et Goldberg, 2002], [Chen, 2002], [O'Neill et Ryan, 2000], [Lobo et *al.*,1998]. Ainsi, plusieurs avantages potentiels peuvent être mis à profit pour les AG :

- une séparation de l'espace de recherche (génotype) et de l'espace de solutions (phénotype). Cette séparation donne plus de flexibilité à l'algorithme que de travailler uniquement avec le phénotype;
- une possibilité de maintenir la diversité génétique tout au long de l'exécution de l'algorithme;
- la conservation de la fonctionnalité, tout en permettant la suite de la recherche de la meilleure solution;
- une généralisation du codage qui peut représenter une variété de structures sans besoin d'opérateurs génétiques spécialisés;
- une compression de la représentation avec un génotype relativement petit;
- une implémentation alternative des fonctions, par exemple, par l'utilisation du modèle de gène de régulation;
- des positions indépendantes dans le génome.

La notion de gènes est également employée dans le prototype proposé. La structure de l'individu est donc séparée en deux parties ; soit l'intron et l'exon. L'intron est la partie du chromosome qui contient les jeux de paramètres et l'exon est la partie qui contient la solution (figure 5.6).

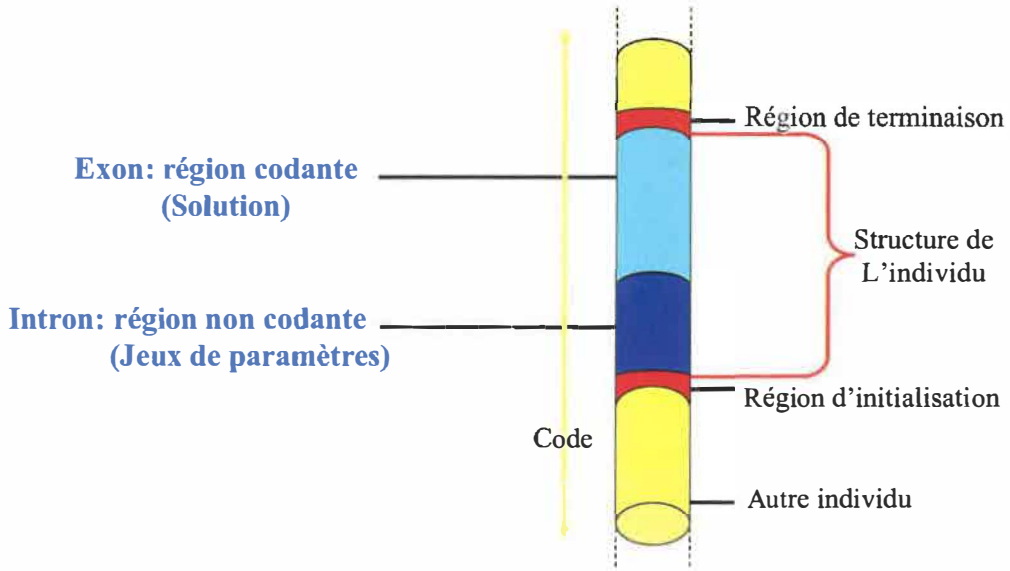


Figure 5.6. Représentation de l'intron et de l'exon au niveau du gène chez un individu de l'approche APAG
(Adapté de Biologie, évolution, diversité et environnement [Mader, 1987])

Les paramètres qui sont impliqués dans l'intron sont des opérateurs de croisement (O_c), des opérateurs de mutation (O_m), la probabilité de croisement (P_c) et la probabilité de mutation (P_m) (figure 5.7).

$$X = (\underbrace{X_1, X_2, X_3, \ldots\ldots\ldots}_{\text{EXON}}, \underbrace{O_c, P_c, O_m, P_m}_{\text{INTRON}})$$

Figure 5.7. Représentation de la structure d'un individu

En apportant la notion de gènes plus structurée aux AG, les opérateurs de croisement et de mutation devront s'adapter aux nouvelles améliorations du chromosome de l'individu.

5.1.3. Opérateur de croissement adaptatif

En modifiant l'algorithme par l'ajout de nouvelles notions, les opérateurs de croisement et de mutation doivent être en mesure de tenir compte de ces changements. **Par conséquent, l'opérateur de croisement n'est plus fixé tout au long de l'exécution de l'algorithme. Il évolue durant l'exécution et s'adapte selon l'individu.** Les paramètres de croisement sont portés par l'individu au niveau de l'intron sur le chromosome (figure 5.8).

$$X= (\underbrace{x_1,x_2,x_3,.....,x_n}_{\text{exon}}, \underbrace{P_c ,O_c}_{\text{intron}})$$

Exon : Partie codante du chromosome qui produit le phénotype (la solution)
Intron : Partie non codante du chromosome qui contient les paramètres de l'AG
P_c : Probabilité de croisement
O_c : Opérateur de croisement

Figure 5.8. Localisation de l'opérateur et de la probabilité de croisement à l'intérieur de l'individu

Le fonctionnement du croisement se fait en deux temps :

1. Recherche dans l'espace de croisement des meilleurs paramètres de croisement pour l'individu X selon la sélection (fonction d'adaptation) du problème ;

2. Croisement de deux individus en utilisant les paramètres de l'individu le plus fort (individu avec la meilleure fonction d'adaptation).

Seul le jeu de paramètres qui donne de bons individus survit. Un bon paramétrage de croisement favorise la convergence plus rapide de l'algorithme. Si la convergence de l'algorithme est trop

rapide, tous les individus tendent à avoir le même code génétique. Cette situation doit être évitée et en général l'apport d'un bon opérateur de mutation permet de contrer cet effet.

5.1.4. Opérateur de mutation adaptatif

Les paramètres de mutation sont portés par l'individu au niveau du chromosome (figure 5.9).

$$X= (\underbrace{x_1,x_2,x_3,.....,x_n}_{\text{exon}}, \underbrace{P_m,O_m}_{\text{intron}})$$

Exon : Partie codante du chromosome qui produit le phénotype (la solution)
Intron : Partie non codante du chromosome qui contient les paramètres de l'AG
P_m : Probabilité de mutation
O_m : Opérateur de mutation

Figure 5.9. Localisation de l'opérateur et de la probabilité de mutation à l'intérieur de l'individu

Le fonctionnement de la mutation se fait en deux temps :

1. Recherche dans l'espace de mutation des meilleurs paramètres de mutation pour l'individu X selon la sélection (fitness) du problème ;

2. Mutation des nouveaux individus enfants en utilisant les meilleurs paramètres.

Les individus qui survivent longtemps résultent de nombreuses mutations successives réussies. Seuls les individus avec de bons paramètres survivent. Un bon paramétrage de mutation accroît ainsi la diversité génétique de la population.

5.1.5. Probabilités de croisement et de mutation adaptatives

Les probabilités de croisement et de mutation dépendent de la diversité génétique de la population. Le but de l'adaptation de la probabilité est de maintenir la diversité génétique dans la population et de prévenir une convergence prématurée de l'algorithme vers un minimum local. Dans une étude sur les améliorations portées aux algorithmes génétiques [Vasconcelos et *al.*, 2001], un concept basé sur la notion de mesure de la diversité génétique (mdg) à été réalisé pour effectuer une adaptation dynamique des probabilités (Figure 5.10).

Mesure de la diversité génétique (mdg)

$$mdg = \frac{\text{Moyenne des valeurs de fitness}}{\text{Maximum des valeurs de fitness}}$$

Diversité génétique élevée et aucune convergence ⟵ $0 \leq mdg \leq 1$ ⟶ Tous les individus ont le même code génétique

Figure 5.10. Représentation du concept de la mesure de la diversité génétique

Cette mesure est le rapport entre la moyenne des valeurs d'adaptation (fitness) sur le maximum des valeurs d'adaptation pour chaque génération. La valeur du mdg est dans la gamme [0 à 1]. Cette valeur de mdg agit un peu comme la variance de la fitness, dans le sens où plus cette variance est forte et plus la sélection agit de manière efficace. Ceci est confirmé par le théorème fondamental de la sélection naturelle selon [Fisher, 1930]. L'avantage de la mdg vient du fait qu'elle est normalisée entre 0 et 1. En effet, plus la mdg tend vers 1, plus tous les individus dans la population tendent à avoir le même code génétique et, par conséquent, l'AG tend à converger

trop rapidement. Par opposition, plus le mdg se rapproche d'une valeur proche de zéro, plus la diversité des solutions est grande et l'algorithme ne peut résoudre le problème et tomber dans un optimum local. Cette notion de diversité est reprise au niveau du prototype d'apprentissage, mais avec une amélioration au niveau du contrôle de cette mesure. L'apport de cette approche à ce concept se traduit par le contrôle de la mdg avec un taux d'apprentissage, ce qui empêche une convergence trop rapide en réduisant la probabilité de croisement et en augmentant le taux de mutation. L'inverse est aussi vrai pour éviter une faible valeur de la mdg. La relation entre les paramètres de probabilités est illustrée dans la figure 5.11. Cette figure illustre des courbes des probabilités de croisement (Pc) et de mutation (Pm) en fonction de la mesure de diversité génétique (mdg). Pour chaque courbe, correspondent des valeurs maximum et minimum de probabilités déterminées par les expérimentations de [Vasconcelos et *al.*, 2001]. Des valeurs de frontières (Vmin, Vmax) sont aussi illustrées pour assurer le fonctionnement de la diversité génétique au-delà de ces frontières.

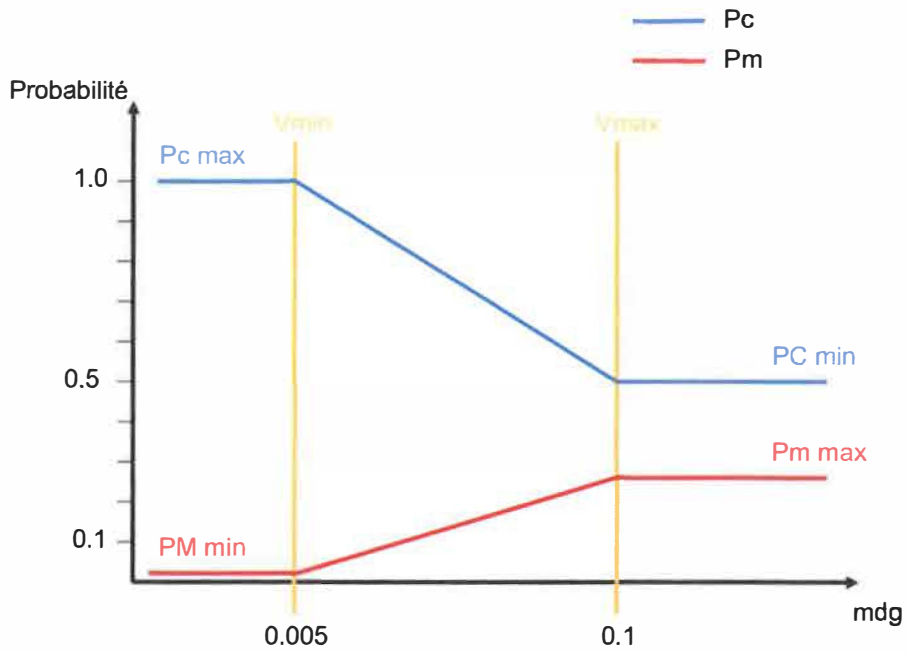


Figure 5.11. Représentation des courbes des probabilités de croisement (Pc) et de mutation (Pm) en fonction de la mesure de diversité génétique (mdg) selon un contrôle via le taux d'apprentissage

5.1.6. Taille de population adaptative

Une des limitations pour le contrôle de la taille de la population est l'élimination, par exemple, d'un individu ayant peu d'intérêt à court terme mais qui, avec le temps, peut se révéler un bon individu en fin de parcours. Cette section propose une méthode pour le contrôle de la taille de la population basée sur la conception d'un arbre phylogénétique.

Le concept de l'arbre phylogénétique est en fait une classification phylogénétique qui suppose que l'on regroupe les individus (dans le cas des AG) en fonction de leurs liens de parenté. Tout groupe systématique renferme donc des êtres vivants proches entre eux génétiquement. Les liens de parenté entre deux membres d'un groupe sont toujours plus étroits que les liens de parenté entre un membre quelconque du groupe et un individu extérieur au groupe. L'illustration de la figure 5.12 montre la conception de l'arbre phylogénétique. Cette méthode a l'avantage de permettre de conserver une partie du bagage génétique dans le déroulement de l'algorithme. L'évolution de l'arbre suit le déroulement cyclique de l'algorithme génétique en terme de générations successives. La sélection des individus, pour la création de la population à chaque génération, est effectuée des branches supérieures de l'arbre vers le bas de celui-ci. Le nombre d'individus sélectionné est dépendant du critère maximal de la taille de la population.

L'arbre phylogénétique se réalise par la mise en évidence des liens de parenté chez les individus, c'est-à-dire, retrouver qui est proche de qui (d'un point de vue gène), et non qui descend de qui. Il faut entreprendre une analyse des gènes pour détecter le gène qui a une grande valeur évolutive par évaluation de la distance phylogénétique.

Le principe de ce concept repose en grande partie sur les gènes. Les gènes sélectionnés doivent toujours être homologues et comporter une grande valeur évolutive. Pour chaque gène, il faut identifier son état primitif ou ancestral et son état dérivé (évolué). La comparaison entre les individus permet de retrouver des homologies au niveau des gènes, soit le partage de gènes évolués.

Chaque branche de l'arbre correspond à la transformation d'un gène qui est transmis de génération en génération et constitue une innovation évolutive. En général, chaque branche est agencée de manière à supposer le minimum de transformations (mutations) de gènes.

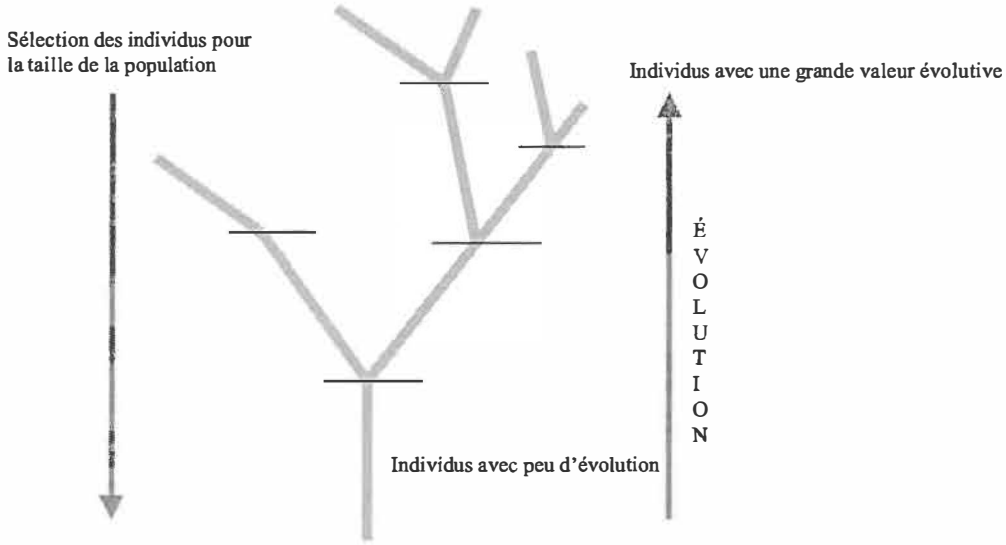


Figure 5.12. Arbre phylogénétique pour le contrôle de la taille de la population

La taille de la population se réalise par la sélection d'un nombre important d'individus des branches supérieures jusqu'à la sélection de quelques individus de toutes les branches inférieures. La sélection d'au moins un représentant de chaque branche favorise une plus grande diversité génétique et conserve les individus pouvant se révéler de bons individus.

5.2. Prototype MAAG

Le développement du prototype de MA des algorithmes génétiques est conditionné par trois relations mentionnées dans le chapitre précédent. Ces relations font intervenir le MA avec l'environnement, l'apprentissage et l'intelligence humaine. Les grandes résultantes de ces relations sont l'aspect modulatoire, les connaissances et les méta-connaissances (MC). Le développement du système de MA se construit donc à partir de ces trois aspects de base, mais en donnant une importance plus marquée aux MC. Le MA est étroitement lié au processus d'acquisition et d'exploitation de la MC ce qui permet au système d'avoir une représentation de lui-même.

Pour obtenir un système cohérent et pouvant agencer tous ces éléments, des méthodes d'optimisation *a priori* et *a posteriori* sont utilisées dans la conception du système. La notion *a priori* est employée pour définir les MC du système, alors que la notion *a posteriori* sert à affiner les MC par simulation. L'approche par simulation consiste à réaliser un ajustement des MC en fonction des résultats issus de simulations. Cet aspect permet au système de raisonner sur ces connaissances et d'explorer d'autres possibilités dans le cas où le système ne peut pas affiner les MC dynamiquement avec l'environnement. Le cœur du système réside dans la définition des ces MC et passe par un système hybride d'algorithmes d'apprentissage comprenant les algorithmes génétiques.

5.2.1. Détails d'implantation et mise en œuvre

Cette section présente le système de MA dans son aspect modulatoire et basé sur une combinaison de méthodes *a priori* et *a posteriori* (Figure 5.13.). Les données d'entrée du système se composent de diverses sources de renseignements provenant de plusieurs capteurs. Ces capteurs peuvent être de nature électronique ou humaine. Le traitement de ces données se fait via un objectif d'apprentissage et vise principalement à optimiser un ensemble de poids d'apprentissage afin de valider le meilleur renseignement. Cette optimisation est réalisée grâce aux algorithmes génétiques. Chaque aspect des modules sera développé dans les sections suivantes.

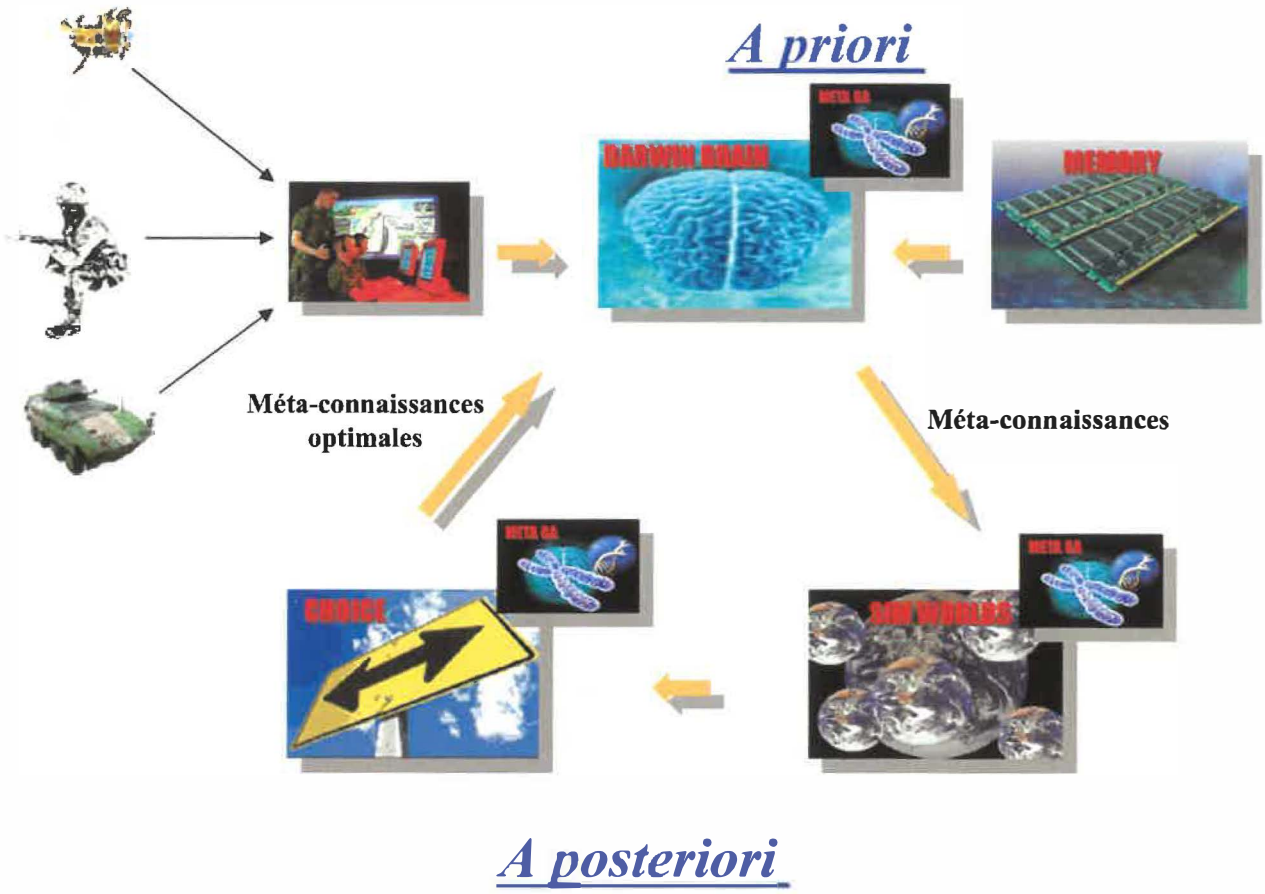


Figure 5.13. Système de méta-apprentissage basé sur les algorithmes génétiques

5.2.2. A priori

La notion *a priori* du système se veut une étape d'acquisition de MC à partir de données prévisionnelles et réelles, c'est-à-dire, créer ou faire évoluer les MC en exploitant les renseignements, les connaissances et les MC obtenus *a posteriori* lors de l'approche par simulation (figure 5.14). Pour se faire, il est nécessaire de définir le contexte d'apprentissage de chaque MC. Cette notion conduit à un ensemble des connaissances possibles pour le système d'apprentissage. Cette aptitude à extraire les MC des connaissances est acquise à l'aide de systèmes hybrides d'algorithmes d'apprentissage qui constituent le module *Darwin Brain*. Celui-ci représente le cœur du système d'apprentissage.

Figure 5.14. Illustration de la notion *a priori* qui acquière des méta-connaissances via les renseignements, connaissances et les méta-connaissances du système

5.2.2.1. Darwin Brain

Le module *Darwin Brain* est un composant majeur du système de MA. Son but est d'extraire et de faire évoluer les MC à partir des connaissances de base et des renseignements qui entrent dans le système (figure 5.15). À la base du système, on retrouve donc le renseignement qui est fourni comme entrée du système. À chaque entrée correspond un vecteur de poids du degré de confiance déterminé par les analystes (voir tableau 2.1. sur l'évaluation de la fiabilité des sources et de la crédibilité de l'information). Le but du système de MA est d'apprendre le meilleur poids qui devrait être associé au renseignement entrant. Il effectue l'apprentissage non seulement sur le renseignement, mais également sur le système lui-même et sur ses propres structures de la connaissance.

Pour ce faire, le module est basé sur le principe de la sélection naturelle de Darwin qui regroupe des algorithmes d'apprentissage qui sont les algorithmes génétiques et le Darwinisme neuronal.

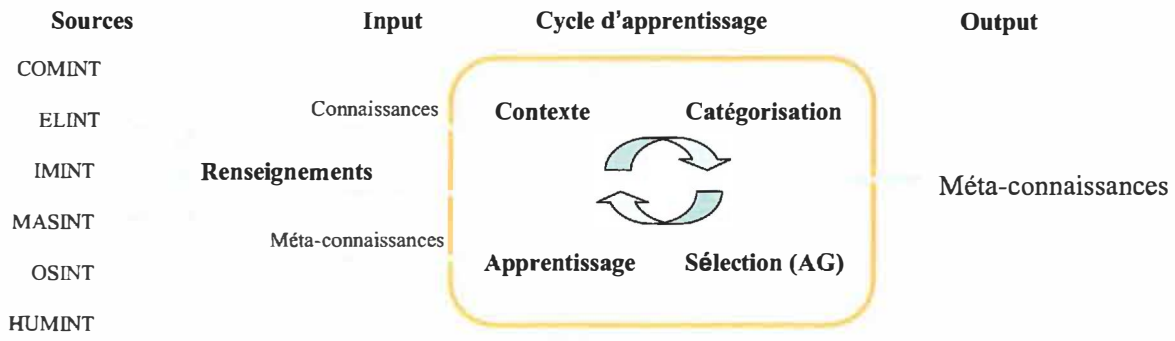


Figure 5.15. Représentation des éléments qui entrent et sortent du module *Darwin Brain*

À partir de la figure 5.15, on observe qu'une part importante des données d'entrée dans le système d'apprentissage est accordée aux renseignements. Le renseignement est fonction de la fiabilité de la source (FS) et de la crédibilité de l'information (CI) et peut se visualiser comme suit :

$$renseignement = f(I_i, S_i, FS_i, CI_i) = f_i(x)$$

$$i = (1,...n)$$

L'apprentissage peut alors se définir comme :

$$F(x) \rightarrow y$$

Ou

$$Min(f_i(x)) = Min(f_1(x), f_2(x),...f_n(x)) \rightarrow y$$

$x = (x_1, x_2,...x_n) \in X$ x = Vecteur de renseignement

$y = (y_1, y_2,...y_m) \in Y$ y = Vecteur de connaissance

Chaque élément x_i est associé à un poids *wj* :

$$f_i(x) = \sum_{j=1}^{n} w_j \cdot x_j \qquad j = (1,...n)$$

Également, chaque fonction de renseignement $f(x_i)$ est associée à un poids d'apprentissage Aw_j . L'accumulation de la valeur pondérée du renseignement donne une valeur combinée du renseignement :

$$F(x) = Min(f_i(x)) = \sum_{i=1}^{n} Aw_i \cdot f_i(x)$$

Le système d'apprentissage essaie de réduire au minimum la distance d'un vecteur d'objectif donné $y_g \in Y$ dans l'espace des vecteurs de connaissances.

$$F(x) = \left\| f(x) - y_g \right\|$$

Le pseudo code suivant décrit plus précisément le fonctionnement du module *Darwin Brain* :

[Start] Requête d'apprentissage

Définition d'un objectif d'apprentissage → fonction d'adaptation (fitness) du système

[Input] Liste de renseignements incluant:

Liste d'informations pondérées ($i_1w_1 + i_2w_2 + i_3w_3 + \ldots + i_nw_n$)

Liste des sources pondérées associées à l'information ($s_1w_1 + s_2w_2 + \ldots + s_nw_n$)

[Validation] Vérifier si l'information et la source ont un poids d'évaluation

Si déjà donné → valeur OK

Sinon → attribution d'une valeur inconnue

[Initialisation] Création d'une topologie de réseau de neurones

Création des couches d'entrées composées de renseignements et/ou de connaissances. Une couche intermédiaire d'apprentissage est créée selon les différentes catégories. Une couche de sortie est composée de neurones résultants associés à la couche d'entrée.

[Propagation] Attribution d'une valeur d'apprentissage sur le renseignement suivant une fonction de transfert [0,1]

Donner par le système avec les connaissances *a priori*, les connaissances acquises et selon le contexte de l'environnement et l'objectif d'apprentissage.

[Fitness 1] Évaluation de la performance selon la valeur d'apprentissage

La sélection des individus se fait au niveau des poids des vecteurs de renseignements ou de connaissances.

[Algorithme génétique 1] Application de plusieurs cycles complets d'AG

Évolution d'une population de vecteurs comprenant des poids d'apprentissage.

[Solution 1] Détermination du meilleur individu

La solution représente le meilleur vecteur de renseignement selon l'attribution des poids d'apprentissage.

[Fitness 2] Évaluation de la performance selon la solution 1 et les gènes dominants

La sélection des individus se fait au niveau du résultat du croisement selon la solution 1 et au niveau des gènes dominants les plus performants. Les gènes dominants sont définis par le contexte d'apprentissage.

[Algorithme génétique 2] Application d'un seul cycle d'AG

Évolution d'une population de vecteurs de renseignement ou connaissances.

[Solution 2] Détermination du meilleur individu

La solution représente le meilleur vecteur de renseignement selon l'attribution des poids d'apprentissage. Elle représente la MC qui est la résultante du module.

La description du pseudo code ci-dessus énonce l'utilisation de deux AG. Chacun de ces algorithmes a besoin d'une fonction d'adaptation (fitness) et produit une solution propre à lui-même. Il y a une utilisation du cycle d'apprentissage et ce cycle est montré à la figure 5.16. Une description plus précise est donnée dans la prochaine section, qui fait état de son fonctionnement.

5.2.2.2. Cycle d'apprentissage appliqué au module *Darwin Brain*

Cette section décrit le fonctionnement du cycle d'apprentissage afin d'obtenir la MC, ce qui est la partie majeure du module *Darwin Brain*. Le cœur du système comporte donc un cycle d'apprentissage qui comprend quatre phases distinctes : le contexte, la catégorisation, la sélection et l'apprentissage (figure 5.16).

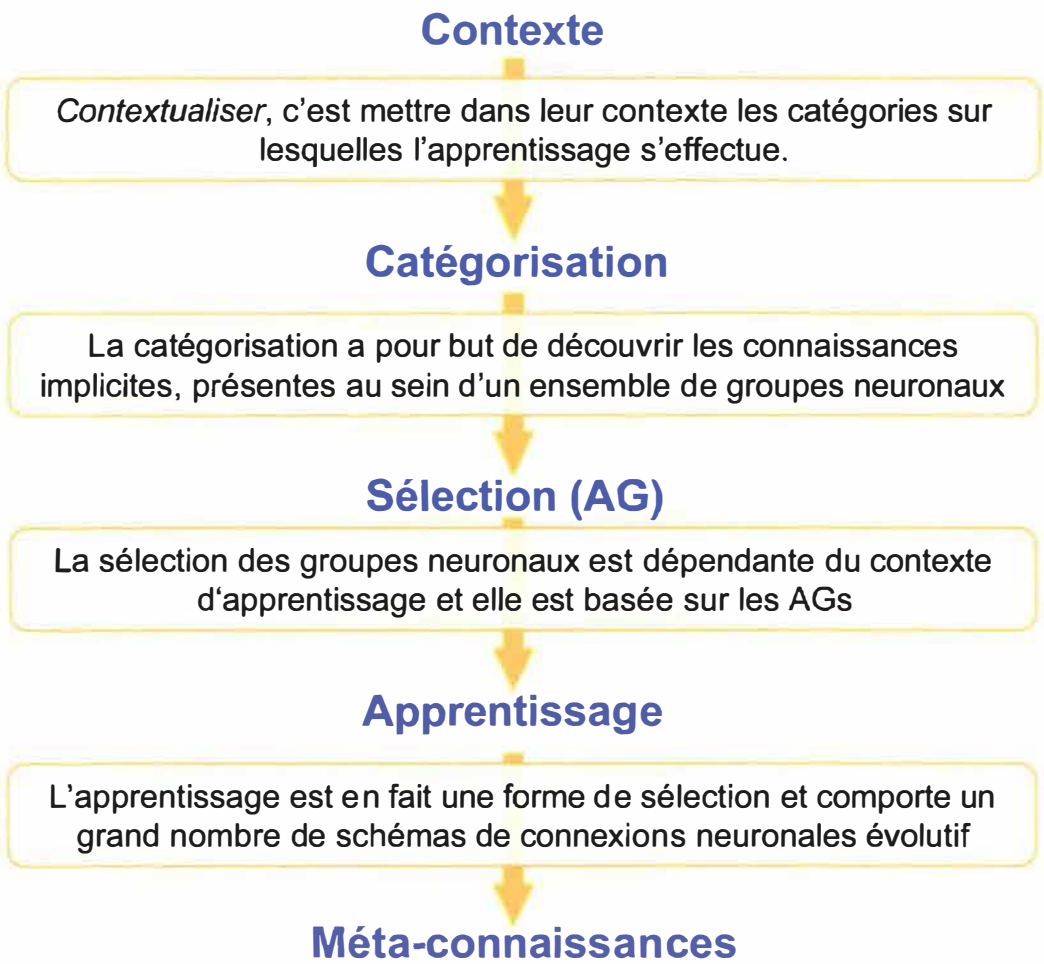


Figure 5.16. Cycle d'apprentissage du module Darwin Brain

5.2.2.2.1. Contexte

Quand le cerveau humain apprend, différents secteurs sont utilisés selon le contexte d'apprentissage. Les niveaux d'activité ne sont pas identiques partout dans le cerveau. Plusieurs schémas de connexions neuronales sont plus fréquemment choisis que d'autres et dépendent ainsi fortement du contexte d'apprentissage. Par conséquent, *contextualiser*, c'est mettre dans leur contexte les catégories sur lesquelles l'apprentissage s'effectue.

5.2.2.2.2. Catégorisation

La catégorisation correspond à un apprentissage dit non supervisé et a pour but de découvrir les connaissances implicites, présentes au sein d'un ensemble de données. Cette découverte de connaissances à partir de données se réalise en détectant les similarités ou les différences qui existent au sein de ces données. La catégorisation apporte une aide efficace dans l'analyse de renseignements HUMINT ou toute autre source de renseignements.

Pour réaliser cette catégorisation, l'utilisation du darwinisme neuronal est mise à contribution (section 4.4.2.3). Cette notion repose sur l'élaboration de connexions de neurones comme système d'apprentissage. Ainsi, l'utilisation d'un réseau de neurones sert comme outil de base du système MAAG. Dans le cas du présent système, les éléments topologiques pour la création des connexions neuronales et de la fonction de transfert pour la normalisation du poids d'apprentissage entre 0 et 1, sont utilisés.

Il y autant de neurones pour la première couche du réseau qu'il y a de vecteurs de renseignement fournis comme entrée du système. Le nombre de neurones de la couche intermédiaire du réseau de neurones est proportionnel au nombre de catégories présentes. Les connexions s'effectuent alors entre les groupes de neurones des couches qui sont reliées ensemble afin de constituer les schémas de connexions neuronales. Ces schémas de connexions sont alors évalués en terme de similarité ou de différences entre les données des neurones des groupes et normalisés par une fonction de transfert.

5.2.2.2.3. Sélection

L'évolution des schémas de connexions neuronales est basée sur les algorithmes génétiques. Certaines combinaisons de connexions sont sélectionnées plutôt que d'autres via leur évaluation en terme de poids d'apprentissage. Cette sélection survient plus spécifiquement au niveau de groupes de neurones reliés en couches avec les catégories. Le choix des groupes neuronaux plus performants est réalisé par une double optimisation et dépend du contexte d'apprentissage.

5.2.2.2.4. Apprentissage

Partant du fait que l'apprentissage est en fait une forme de sélection et comporte un grand nombre de schémas de connexions neuronales évolutives, l'apprentissage est en conséquence une résultante de la théorie de la sélection des groupes neuronaux (Darwinisme neuronal). En reprenant les trois postulats de bases de la théorie, l'apprentissage se décrit comme suit :

1) Les neurones qui représentent les renseignements se connectent d'abord au hasard puis de plus en plus systématiquement aux catégories afin de répondre à des contraintes d'apprentissage (figure 5.17). C'est en fait le contexte d'apprentissage qui définit les catégories. Les catégories représentent la couche intermédiaire d'apprentissage et associent la couche d'entrée à la couche de sortie du réseau de neurones.

2) Les connexions entre les renseignements et les catégories qui sont les plus performantes se renforcent via le poids d'apprentissage, d'autres disparaissent ou la valeur du poids est plus faible.

3) La réintroduction se produit lorsqu'un stimulus, externe ou d'origine interne, est reçu par le système. Les stimuli externes prennent la forme de nouvelles connaissances qui s'introduisent au système. Les stimuli d'origine interne correspondent au MC optimales qui sont données par la potion *a posteriori* du système. Alors, des catégories différentes sont impliquées en même temps et plusieurs neurones s'activent alors en parallèle, s'auto-informant les uns les autres.

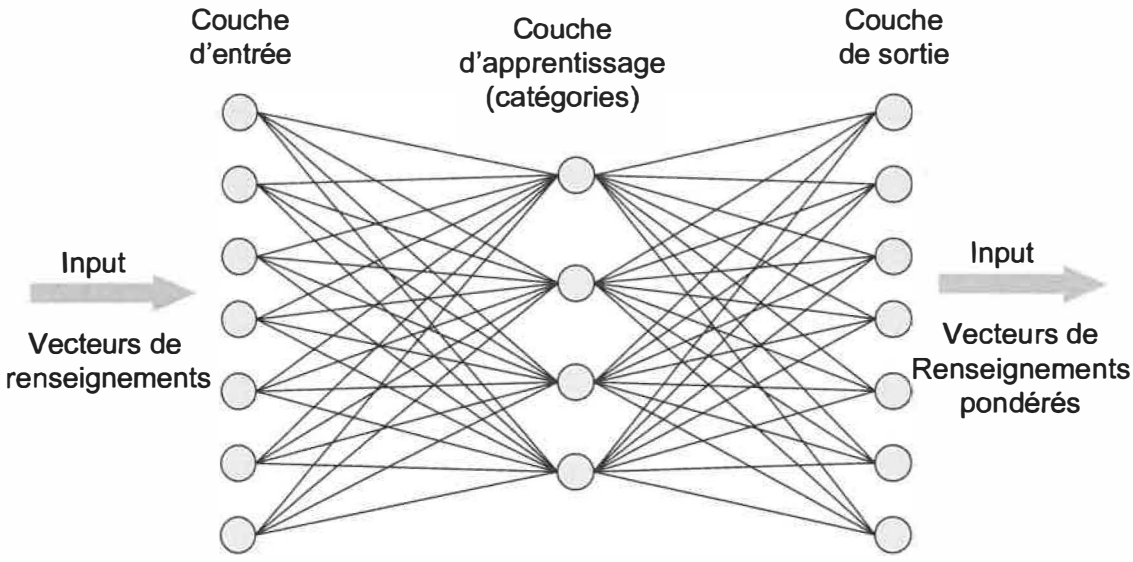


Figure 5.17. Exemple de représentation des connexions des neurones lors des étapes d'apprentissage

5.2.2.2.5. Méta-connaissances

La MC est représentée par une structure qui contient le renseignement ainsi que les poids de confiance associés, le poids d'apprentissage et un indice du niveau d'apprentissage. Cet indice apporte une caractéristique supplémentaire sur l'état de la connaissance par rapport à son état antérieur.

Les modules de la notion *a posteriori* (sections suivantes) emploient la MC produite par le module *Darwin Brain* pour accroître l'apprentissage du système. En effet, le module "des mondes de Sim" augmente la performance du système en simulant la MC et aide à développer de nouvelles connaissances.

5.2.2.3. Memory

Le module *Memory* stocke toutes les connaissances qui seront employées par le module *Darwin Brain* pour apprendre de nouvelles connaissances. La structure de stockage du module *Memory* est basée sur la représentation en 5 dimensions du monde physique défini par [Pigeon, 2002a], [Pigeon, 2002b]. Il propose une représentation des éléments dans l'espace (x, y, z), dans le temps (t), et dans les états ou mondes possibles (w).

La dimension de l'espace représente les coordonnées spatiales de la connaissance tant physiques qu'abstraites. Par exemple, si on traite une connaissance de type qualitatif, celle-ci peut-être une instance d'une autre connaissance. Dans ce cas, la représentation spatiale sert à définir la dimension de l'instance.

La dimension temporelle sert à caractériser l'évolution de la connaissance dans le temps. Elle peut aussi être utilisée pour stocker la connaissance liée à une capacité de ressource disponible dans le temps. À partir de cette dimension, l'estimation des états futurs dans le temps implique une autre dimension qui est celle des mondes possibles. Cette dimension sert à définir la connaissance dans l'exploration de toutes les possibilités d'états que peut posséder la connaissance. Les cinq dimensions sont donc considérées comme attributs de la connaissance pour des fins de stockage dans le module *Memory*. Toutes les connaissances entreposées constituent la mémoire du système d'apprentissage.

5.2.3. A posteriori

La notion *a posteriori* se rapporte au raffinement des MC (figure 5.18). Elle utilise les MC obtenues afin de rechercher les connaissances idéales par un processus de simulation (module *Sim Worlds*) et de sélection (module *Choice*). Elle consiste à matérialiser le comportement que pourrait avoir notre système avec des connaissances définies lors de la phase *a priori*. Pour ce faire, les MC doivent être appliquées au système réel ou appliquées à un système approximatif permettant de simuler le système réel. Cette simulation doit être le plus efficace possible afin d'explorer un maximum de configurations différentes de connaissances et, ainsi, optimiser les chances de détecter une connaissance optimale. Le choix d'une connaissance optimale passe par une étape d'évaluation de la qualité de ces connaissances par rapport à un objectif déterminé. Elle confère au système une certaine autonomie, ce qui lui permet de réagir aux situations imprévues. Le processus de raffinement de l'apprentissage *a priori* peut être défini comme un MA ou un apprentissage autonome du système.

Figure 5.18. Illustration de la notion *a posteriori* qui se rapport au raffinement des méta-connaissances

5.2.3.1. Sim worlds

Le module *Sim Worlds* a pour objectif de simuler les mondes possibles ou états possibles dans lesquels les MC peuvent se retrouver. Pour ce faire, le simulateur doit être basé sur la représentation des connaissances : soit les dimensions de l'espace et du temps. En faisant varier les éléments des vecteurs de MC, on peut découvrir de nouveaux vecteurs de connaissances implicites présents au sein d'un ensemble de MC. Cette phase est transitoire car elle doit préparer les MC pour le module *Choice* afin d'évaluer les performances vis-à-vis l'environnement du système.

5.2.3.2. Choice

Le but de ce module est d'effectuer une évaluation des performances d'un ensemble de connaissances. Ce module se concentre sur l'estimation de la qualité des simulations obtenues à partir du module *Sim Worlds*. L'évaluation des performances d'un ensemble de connaissances se fait dans un contexte appliqué au système réel ou appliqué à un système approximatif simulant le vrai système. La prédiction du comportement du système vis-à-vis les connaissances est un bon indicateur de la performance. Les meilleures connaissances seront alors employées par le module *Darwin Brain* pour améliorer les apprentissages au niveau du système de MA.

5.3 Conclusion

Ce chapitre décrit les prototypes proposés dans les objectifs de recherche. En effet, le prototype APAG est essentiellement basé sur le concept de l'autonomie. L'autonomie se reflète par une architecture spéciale au niveau de l'individu de l'algorithme par rapport aux individus des AG traditionnels. Le prototype MAAG se caractérise par son aspect modulatoire avec une orientation importante sur la notion de méta-connaissances. Le module principal du prototype MAAG est le module *Darwin Brain* qui renferme le cœur du système d'apprentissage. L'apprentissage est basé sur l'évolution par algorithme génétique, mais aussi en combinaison avec le darwinisme neuronal. Le darwinisme apporte la partie structurale de l'apprentissage qui peut se comparer aux neurones du cerveau humain. L'élaboration théorique de telles architectures comme système d'apprentissage se devait d'être validée par une expérimentation, qui fait l'objet du chapitre suivant, et qui peut démontrer les performances d'apprentissage.

Chapitre 6
Modèles expérimentaux

Pour des raisons d'échéance de réalisation, le développement du prototype APAG a été réalisé en premier dans le projet de recherche. De plus, le système de la Défense Nationale dans un contexte d'intervention en zone urbaine n'étant pas disponible à ce moment, un autre modèle d'expérimentation a été retenu (problème du commis voyageur). Ce modèle de développement correspond à un problème de recherche de chemin, ce qui rejoint un des objectifs du système de la Défense Nationale (voir section 6.2).

Afin de répondre à la contrainte d'échéance de réalisation du projet et de la complexité des différents modules du prototype MAAG, l'expérimentation se concentre uniquement sur le cœur de l'apprentissage, soit la réalisation du module *Darwin Brain*. Les autres modules (*Memory, Choice, Sim Worlds*) restent à l'état de théorie. Des travaux futurs devront être réalisés pour le développement complet du prototype afin d'accroître le perfectionnement du système de méta-apprentissage.

6.1. Prototype APAG

Pour examiner les possibilités d'apprentissage du prototype par auto-adaptation des paramètres de l'algorithme génétique, un modèle basé sur le problème du commis voyageur a été retenu. C'est le problème d'optimisation classique le plus connu dans lequel un vendeur doit traverser un ensemble de villes en un coût minimum. Il doit toutes les traverser en passant par chaque ville exactement une fois. Il commence par une ville quelconque et termine en retournant à la ville de départ. Connaissant les distances entre les villes, on doit choisir le chemin qui correspond à la plus courte distance total à parcourir.

L'intérêt pour ce type de problème réside dans le fait qu'il fournit un bon exemple d'étude d'un problème NP-complet dont les méthodes de résolution peuvent s'appliquer à résoudre d'autres problèmes de recherche de chemin à coût minimum. En effet, la résolution du problème de recherche de chemin en zone urbaine permettant d'obtenir des solutions exactes en un temps raisonnable pour de grandes instances (grand nombre d'intersections) du problème est un probleme NP-Complet. Dans le cas du système de la Défense Nationale, le problème de recherche de chemin fait face à un graphe composé de 38 894 arcs (segments routiers) et de 15 422 nœuds (intersections), seulement pour le réseau urbain de la ville de Québec.

La classe NP-Complet regroupe des problèmes exponentiels et polynomiaux, pour lesquels la recherche d'une solution consiste à parcourir un arbre. La hauteur d'un tel arbre est polynomiale, par contre son nombre de branches est exponentiel et chaque branche représente une solution potentielle. L'espace des solutions possibles croît exponentiellement en fonction de la profondeur de l'arbre. Par exemple, si l'arbre est binaire et de profondeur n, alors le nombre de branches est en 2^n, ce qui implique possiblement le parcours complet de toutes les solutions.

Plusieurs études ont appliqué avec succès les algorithmes génétiques à la résolution du problème du commis voyageur [Ahn and Ramakrishna, 2002],[Freisleben and Merz, 1996], ce qui en fait un modèle parfait de comparaison pour valider les performances du prototype APAG, puisque l'approche proposée est une variation améliorée de l'algorithme génétique simple.

6.1.1. Description de l'environnement

La résolution du problème a été abordée par deux approches à des fins comparatives. L'une par l'utilisation des algorithmes génétiques dits simple et l'autre par l'Auto-adaptation des Paramètres de l'Algorithme Génétique. L'approche par algorithme génétique a été réalisée par [Dubot, 1997] et l'implémentation de l'approche APAG s'est faite par ajout et modification de classes de l'applet.

Le but était de comparer, à partir de données uniques, la résolution d'un problème par deux approches d'intelligence artificielle différentes.

Le nouvel applet reprend des éléments de l'ancienne mais avec de nouvelles fonctionnalités (voir figure 6.1).

6.1.1.1.Paramètres de la population

Les paramètres de la population doivent être fixés avant l'exécution de l'algorithme et ils ont une importance marquée dans la résolution du problème.

- **Nombre d'individus de la population**

  Ce paramètre vise à fixer le nombre d'individus dans la population et ce pour toute la durée de l'exécution de l'algorithme. La taille de la population ne doit pas être trop grande car après une certaine limite la performance de l'algorithme diminue. En effet un nombre d'individus trop élevé affecte la rapidité de la résolution du problème. Selon la littérature et tel que discuté à la section 4.1.1.2.1, un nombre variant de 50 à 100 individus est acceptable et donne une bonne performance.

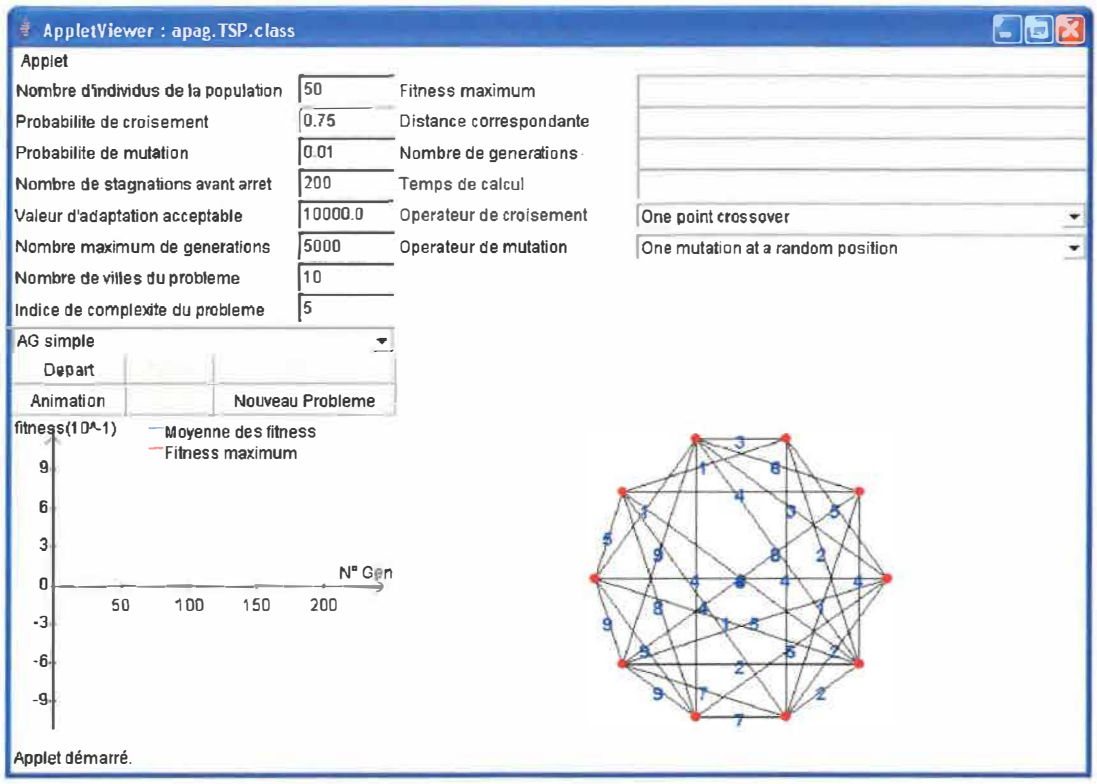


Figure 6.1. Fenêtre principale de l'applet APAG

- **Probabilité de croisement**

La probabilité de croisement indique le taux de participation à la reproduction, soit la proportion de la population qui se reproduit par croisement. Si la probabilité de croisement est de 100%, alors toute la population participe au croisement. Par contre, si elle est de 0%, la nouvelle génération au complet est la copie exacte des individus de l'ancienne population, mais ceci ne signifie pas que la nouvelle génération est identique. Généralement, pour une utilisation simple d'algorithmes génétiques, le taux de croisement peut se situer entre 60% et 95%. La probabilité de croisement pour le prototype APAG est définie par un nombre aléatoire dans l'intervalle [0.0, 1.0].

- **Probabilité de mutation**

  La probabilité de mutation indique la probabilité que chaque gène de chaque individu subisse une mutation lors d'une phase de reproduction. Si le taux de mutation est de 0%, les individus qui sont produits juste après le croisement ne comportent aucun changement. Par contre, si la probabilité de mutation est de 100%, tout le chromosome de l'individu est changé. Un bon taux de mutation se doit d'être bas et se retrouve entre 0.1% et 1% de probabilité dans l'utilisation d'AG. Cependant, le prototype APAG utilise un taux tiré aléatoirement dans un intervalle de probabilités entre [0.0, 1.0].

- **Opérateur de croisement**

  L'opérateur de croisement réalise la reproduction entre les individus. Le type et l'implémentation de l'opérateur sont dépendants du codage du chromosome et de la nature du problème. Dans la plupart des problèmes, l'implémentation d'un opérateur avec un ou deux points de croisements est utilisé. Dans cet applet, 5 opérateurs de croisement sont définis et énumérés dans le tableau 6.1. Un opérateur de croisement avec un point de coupure est un opérateur qui choisit aléatoirement un point de croisement dans un chromosome, ou individu, puis échange les gènes des deux parents pour produire deux enfants. L'opérateur à deux points de coupures choisit aléatoirement deux points de croisement dans un chromosome, puis échange les deux chromosomes des parents entre ces points pour produire la nouvelle descendance.

  Un opérateur de croisement uniforme est un opérateur qui décide, avec une certaine probabilité déterminée à l'avance, quel parent contribuera à chacune des valeurs de gène dans les chromosomes des descendants. Ceci permet aux chromosomes des parents d'être mélangés au niveau du gène plutôt qu'au niveau de portions de chromosome. Généralement, la probabilité est de 0.5, ce qui signifie que la moitié des gènes dans la progéniture viendra d'un parent et l'autre moitié viendra de l'autre parent. La probabilité du croisement uniforme peut prendre d'autres valeurs [Tongchim and Chongstitvatana, 2002].

- **Opérateur de mutation**

  L'opérateur de mutation effectue ou non des changements au niveau des gènes des individus après la phase de reproduction. Tout comme les opérateurs de croisement, le type et l'implémentation de l'opérateur sont dépendants du codage du chromosome et de la nature du problème. 5 opérateurs de mutation sont aussi définis et énumérés dans le tableau 6.1.

Les trois premiers opérateurs changent la valeur d'un ou de deux gènes selon une position déterminée aléatoirement. L'opérateur de mutation uniforme fonctionne dans la mesure où chaque gène possède une probabilité de subir une mutation (déterminée aléatoirement). Quelques opérateurs de mutation ont été modifiés selon la notion de l'amélioration. Dans ce cas-ci, la mutation se produit seulement si la mutation améliore l'individu. Cette notion d'amélioration est basée sur la différence entre la valeur d'adaptation d'un individu avant mutation et la valeur d'adaptation du même individu après mutation.

Opérateur de croisement (O_c)	1. Un point de croisement 2. Deux points de croisement 3. Croisement uniforme avec une probabilité de 0.5 4. Croisement uniforme avec une probabilité de 0.1 5. Croisement uniforme avec une probabilité de 0.2
Opérateur de mutation (O_m)	1. Une mutation à une position aléatoire 2. Deux mutations à une position aléatoire 3. Deux mutations à une position aléatoire, seulement si amélioration possible 4. Mutation uniforme 5. Mutation uniforme, seulement si amélioration possible
Probabilité de croisement (P_c)	Nombre aléatoire dans l'intervalle [0.0, 1.0]
Probabilité de mutation (P_m)	Nombre aléatoire dans l'intervalle [0.0, 1.0]

Tableau 6.1. Tableau représentant la définition des paramètres pour le prototype APAG

6.1.1.2. Paramètres des conditions d'arrêt

L'arrêt de l'algorithme dans la recherche de la meilleure solution intervient dès qu'un des paramètres atteint la valeur fixée.

- **Nombre de stagnations avant arrêt**

  La stagnation se fait au niveau de la valeur d'adaptation du meilleur individu. Lorsque la valeur d'adaptation (*fitness*) du meilleur individu de la population se répète constamment d'une génération à l'autre, on peut considérer que cette valeur est optimale et terminer l'exécution de l'algorithme. Une valeur de 200 stagnations avant arrêt est attribuée comme valeur de départ.

- **Valeur d'adaptation acceptable**

  Ce test d'arrêt intervient lorsque l'on est capable de dire à partir de quelle limite on sera satisfait par la solution. Cela se traduira par la définition d'une valeur d'adaptation minimum de la solution (c'est le paramètre à donner). L'algorithme interrompra donc les recherches dès que la valeur d'adaptation du meilleur individu atteindra ou dépassera cette valeur.

- **Nombre maximum de générations**

  C'est un nombre qui limite le nombre d'évolutions de la population que l'algorithme exécute. La recherche est ainsi arrêtée après un certain nombre de générations et qui est de 5000 comme valeur par défaut pour l'applet.

6.1.1.3. Paramètres de la génération de graphe

Ces paramètres décrivent l'architecture topologique du graphe.

- **Nombre de villes du problème**

  C'est le nombre de villes que le commis voyageur doit traverser. Par défaut, un nombre de 10 villes est fixé. Plus le nombre de villes augmente, plus le problème augmente en difficulté et devient lourd pour la résolution.

- **Indice de complexité du problème**

  Le nombre de l'indice de complexité du problème sert à la construction de la matrice des distances et correspond au nombre de solutions possibles. Plus cet indice est élevé plus le graphe comportera d'arcs. Le nombre de possibilités augmente et la complexité de résolution augmente aussi par le fait même.

6.1.1.4. Le choix de la résolution du problème

L'applet a comme but de tester la performance du prototype par auto-adaptation des paramètres de L'AG. Le choix entre deux méthodes de résolution du même problème permet donc de visualiser les performances de chacun.

- AG simple
  Lance une résolution du problème par approche AG traditionnelle.

- AUTOAG
  Lance une résolution du problème par l'approche d'auto-adaptation des paramètres de l'AG (APAG).

6.1.1.5. Les boutons de commande

Les boutons de commande servent au fonctionnement et au contrôle de l'exécution des algorithmes pour la résolution du problème.

- Départ
  Lance ou relance une résolution du problème courant.

- Pause
  Permet de suspendre la résolution du problème en cours d'exécution.

- Fin
  Permet d'interrompre une résolution en tout temps.

- Animation
  Lorsqu'une solution à un problème a été trouvée, on peut lancer une animation permettant de visualiser la solution proposée.

- Nouveau problème
  Provoque la génération d'un nouveau graphe représentant un nouveau problème à résoudre.

6.1.1.6. Les sorties

Plusieurs types de sorties permettent de suivre en temps réel l'exécution des algorithmes, comme le montre la figure 6.2. Ces sorties permettent ainsi de comparer la performance des algorithmes pour la résolution d'un même problème.

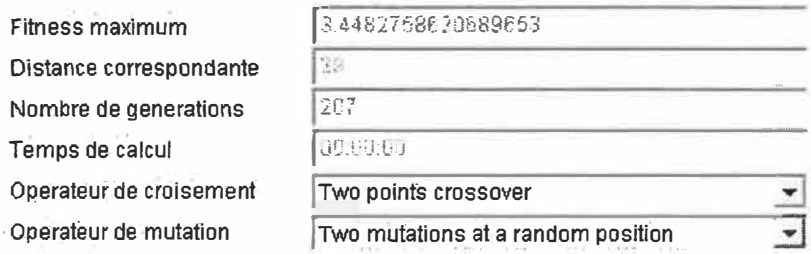


Figure 6.2. Affichage des sorties après résolution du problème

- **Fitness maximum**

  Affichage de la valeur d'adaptation du meilleur individu de la population courante.

- **Distance correspondante**

  Affichage de la distance minimum, ce qui représente la solution trouvée par les algorithmes génétiques.

- **Nombre de générations**

  Affichage du nombre de générations parcourues par l'algorithme.

- **Temps de calcul**

  Affichage du temps de calcul pour trouver une solution.

- **Graphique des courbes de la valeur d'adaptation**

  Affichage du dessin des courbes de la meilleure valeur d'adaptation et de la moyenne des valeurs d'adaptation de toutes les populations en fonction du nombre de générations.

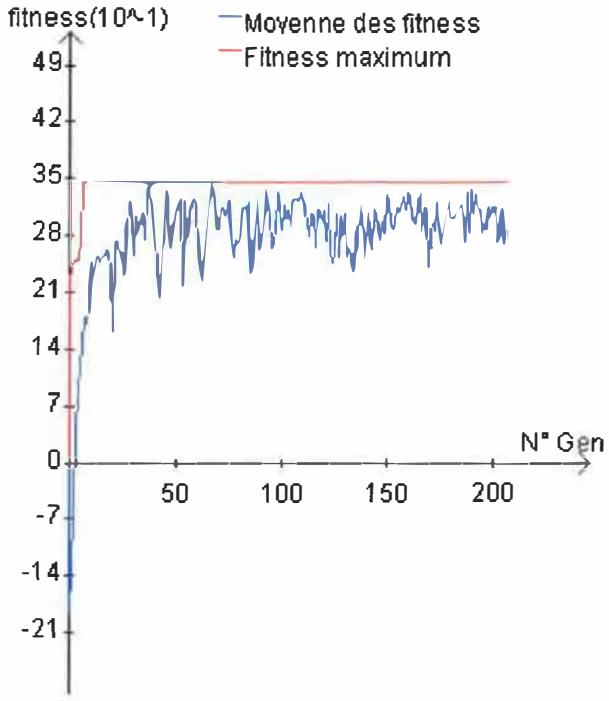


Figure 6.3. Graphique montrant les courbes de la valeur d'adaptation et de la moyenne des valeurs d'adaptation

La courbe indique la tendance de la convergence de l'algorithme vers la solution optimale. La courbe de la valeur d'adaptation maximum (rouge) représente l'individu le mieux adapté à la situation. La courbe de la moyenne des valeurs d'adaptation (bleu) a tendance à se rapprocher de la courbe maximum lorsque l'exécution de l'algorithme est normale. La variation au niveau de cette courbe par rapport à la courbe maximum est dépendante de la diversité génétique En effet, plus la variation ou l'écart entre les deux courbes est petite, plus les individus dans la population tendent à avoir le même code génétique. Par conséquent, un écart élevé entre les deux courbes provoque une grande diversité génétique et aucune convergence n'est observée.

- **Dessin de la solution courante**

  Affichage du dessin du graphe à résoudre avec le parcours optimum trouvé comme solution au problème (figure 6.4). Toutes les villes sont symbolisées par un point rouge sur un cercle. Les chemins possibles entre deux villes sont représentés par un segment noir entre les deux points correspondants. La distance entre les deux villes est écrite en bleu au milieu du segment. Lorsque l'algorithme propose une solution, elle est représentée par des segments en vert et la distance correspondante est écrite en rouge dans un ovale jaune

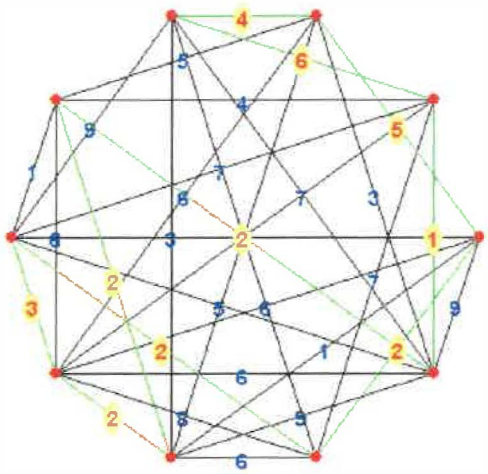


Figure 6.4. Dessin du parcours (lignes vertes) de la solution au niveau du graphe

6.1.2. Implémentation de l'applet

L'implémentation de l'applet a été développée à partir de trois classes abstraites adaptées des classes de [Dubot, 1997] et qui constituent le fondement logique des algorithmes génétiques, soit :

1. *Gene.java*
2. *Individu.java*
3. *Population.java*

En effet, l'algorithme génétique a comme but de faire évoluer une population de solutions dans le temps. Cette population est par le fait même constituée d'individus représentant ou comportant une partie de la solution possible au problème à résoudre. Un mécanisme de codage de ces individu est alors représenté par une classe *Gene*. Plusieurs classes dérivent de ces trois classes abstraites pour bien représenter l'applet, elles sont toutes illustrées dans le diagramme de classe de la figure 6.5.

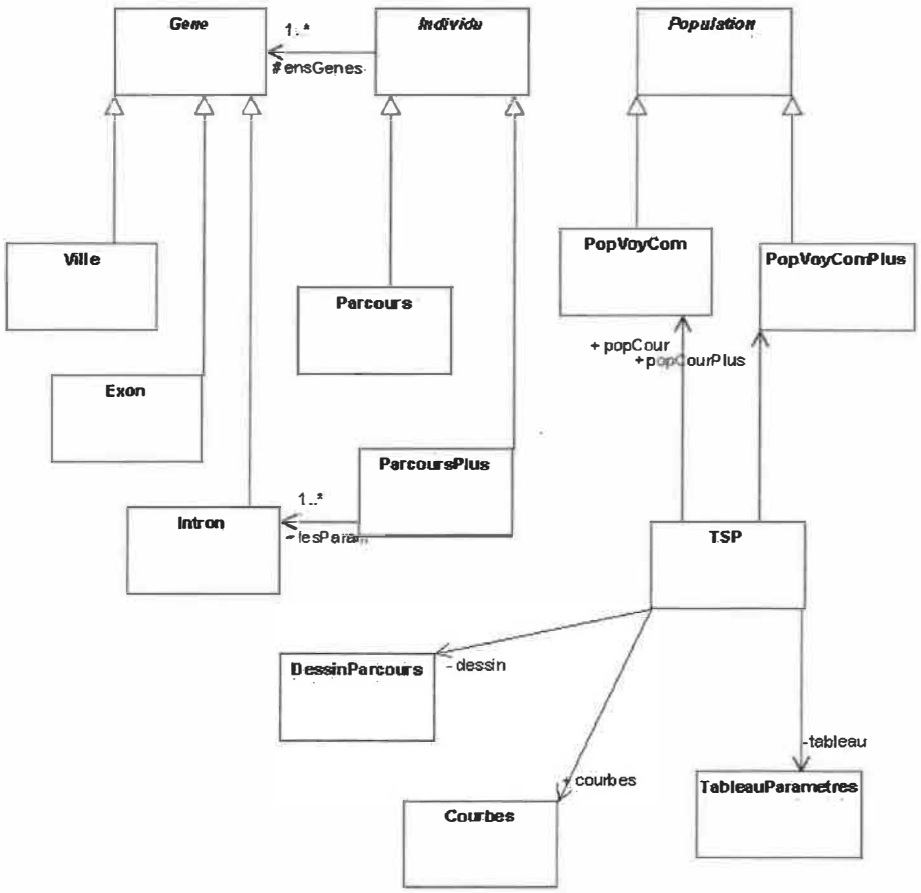


Figure 6.5. Diagramme de classe représentant l'applet

Les prochaines sections vont traiter en profondeur de chaque classe abstraite et de leurs classes dérivées mentionnées ci-haut, ainsi que des classes utiles au fonctionnement de l'applet.

6.1.2.1. La classe *Gene*

La classe *Gene* (figure 6.6) représente l'élément de codage des individus qui composent la population. Cette classe ne comporte qu'une méthode abstraite de la mutation d'un gène. Généralement, la mutation est associée au gène et elle ne représente que la création aléatoire d'un gène. L'implémentation de la méthode de mutation se retrouve au niveau de l'individu. Si l'on fait abstraction du simple fonctionnement de l'algorithme, la méthode de mutation pourrait aussi être applicable au niveau d'autres classes (voir le chapitre 8).

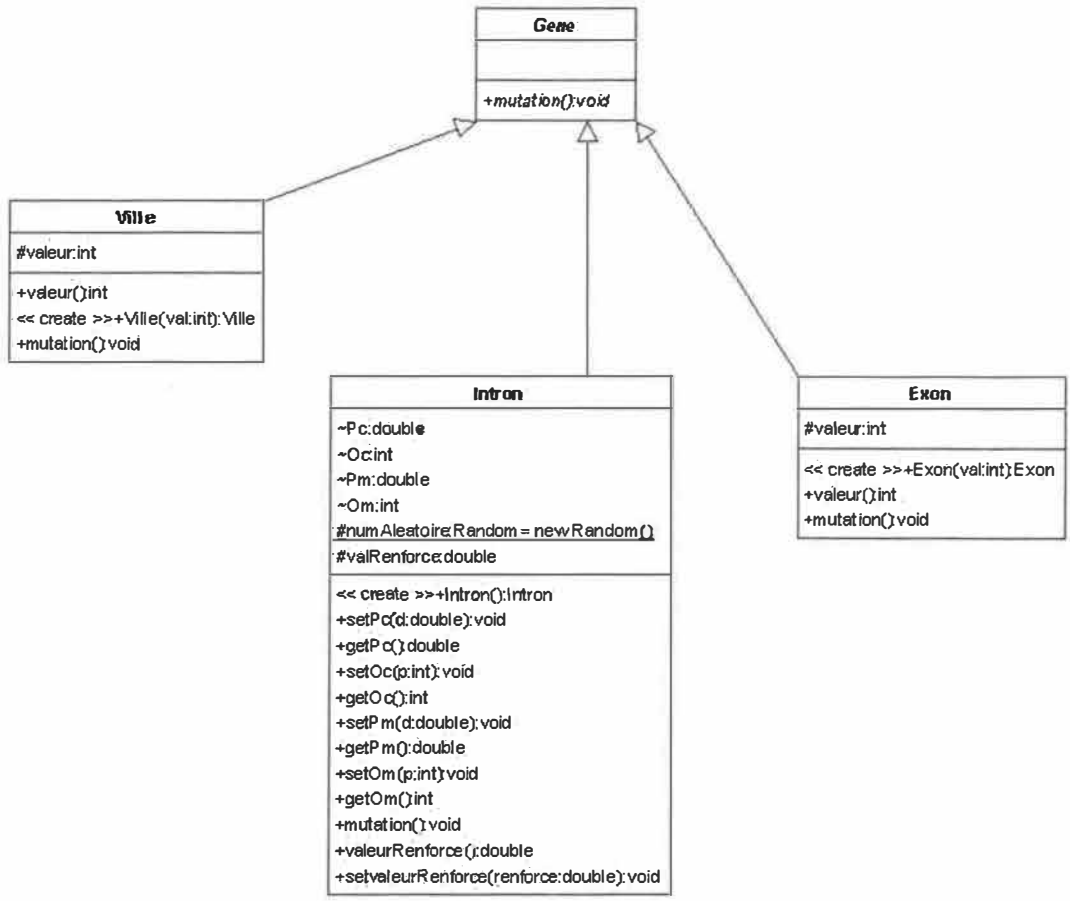


Figure 6.6. Diagramme UML de la classe *Gene*

La sous-classe *Ville* définit la numérotation des villes qui est en fait la plus petite unité de représentation du problème à traiter. Cette classe sert uniquement pour la résolution du problème par l'algorithme génétique simple. Par conséquent, les classes dérivées *Intron* et Exon sont

consacrées uniquement pour la résolution du problème par le prototype APAG. La classe *Intron* porte les paramètres des individus. C'est la partie non codante de l'individu, ce qui implique que les paramètres sont propres au fonctionnement interne de l'individu. Un intron est donc défini par des paramètres de probabilité et d'opérateur ainsi que d'une valeur de renforcement. Cette valeur donne un état de performance par rapport à l'environnement d'évolution de l'individu. La classe *Exon* exerce la même fonction que la classe *Ville*, soit de porter le numéro des villes. En effet, cette classe a comme but de porter les éléments constituant la solution, soit la partie codante de l'individu.

6.1.2.2. La classe *Individu*

La classe *Individu* (Figure 6.7) est représentée par un ensemble de gènes ainsi qu'une valeur d'adaptation qui caractérise l'individu. Les méthodes liées à la valeur d'adaptation de cette classe permettent de lire et d'écrire cette valeur de l'individu. D'autres méthodes permettent d'accéder à un des gènes spécifiques de l'individu, d'effectuer un croisement avec un autre individu, et de réaliser les mutations des gènes de l'individu.

Les méthodes de croisement, de mutation et de calcul de la valeur d'adaptation sont des méthodes définies comme abstraites, car leur implémentation dépend du problème à résoudre. Il faut donc redéfinir les méthodes selon le cas. Par contre, on peut définir une certaine fonctionnalité générale, comme dans le cas de la méthode de mutation suivante :

> Pour chaque gène de l'individu
>
> 1. On tire au sort un nombre entre 0 et 1.
>
> 2. Si ce nombre est plus petit que la probabilité de mutation, alors on fait appel à la méthode de mutation de ce gène.
>
> Si un des gènes a été modifié, alors on recalcule la valeur d'adaptation.

La classe dérivée *Parcours* définit un parcours pour l'approche AG simple qui est composé d'un ensemble de villes. Cet ensemble de villes à visiter est regroupé dans un vecteur et les méthodes de croisement et de mutation permettent de modifier cet ensemble. En effet, la méthode de croisement réalise la reproduction entre deux individus selon le choix de l'opérateur préalablement sélectionné par l'usager dans l'interface.

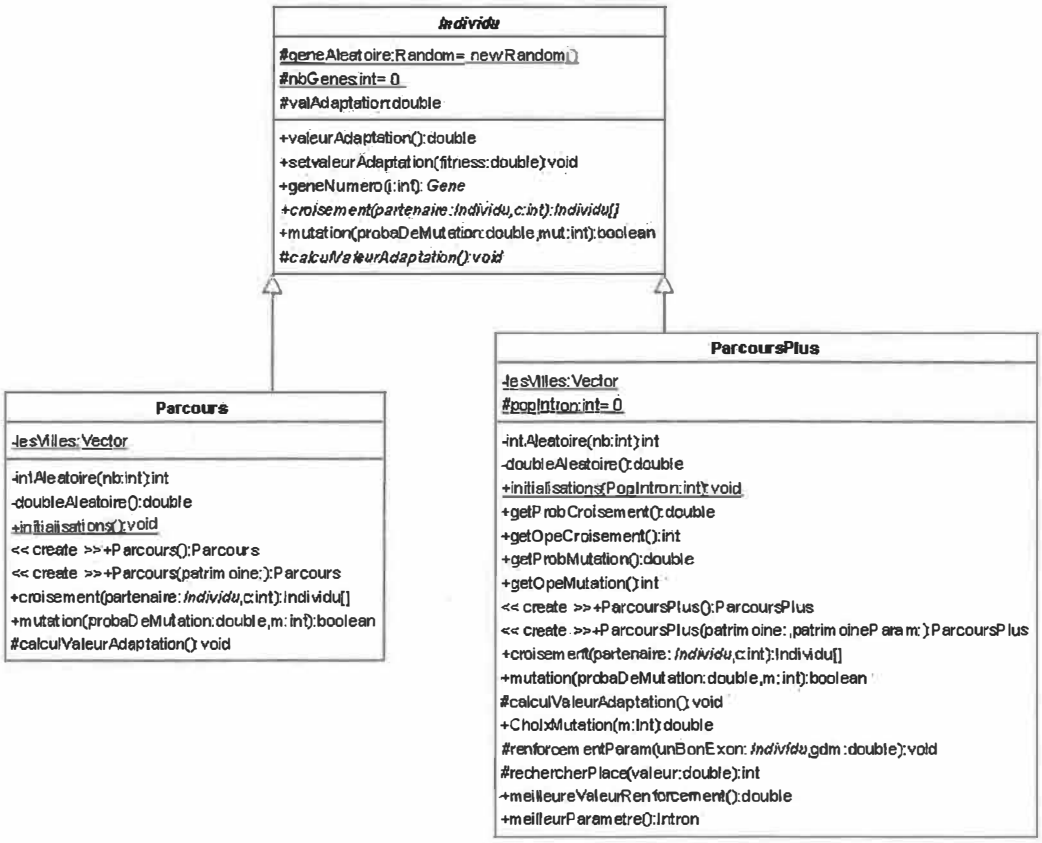


Figure 6.7. Diagramme UML de la classe *Individu*

La méthode de mutation effectue les mutations, si il y lieu, dans le parcours passé en paramètre. Le choix de la méthode de mutation est aussi sélectionné au niveau de l'interface et passé en paramètre à la méthode. Pour chaque modification par ces méthodes, il faut effectuer le calcul de la valeur d'adaptation. Ce calcul se fait par l'implémentation de la méthode abstraite de l'individu et passe par une vérification de la validité du parcours. Si le parcours est valide, alors la valeur d'adaptation sera positive et elle prendra comme valeur l'inverse de la longueur du parcours. Par contre, si le parcours n'est pas valide, la valeur d'adaptation sera négative. Elle sera déterminée par la négation du nombre de chemins impossibles qu'une solution peut contenir. Cela représente un individu qui n'est pas une solution au problème.

La classe dérivée *ParcoursPlus* est semblable à la classe *Parcours* au niveau du vecteur de villes à visiter, mais diffère par une variable supplémentaire qui est une population d'intron. Cette population sert à l'évolution des jeux de paramètres.

La méthode de croisement est également semblable, mais comporte une différence au niveau des enfants. Cette différence se reflète par l'ajout des paramètres du meilleur individu participant à la reproduction. Les méthodes de mutation et de calcul de la valeur d'adaptation sont aussi semblables à celle de la classe *Parcours*. Une des différences entre les deux classes se retrouve au niveau de la méthode *renforcementParam*. Cette méthode représente un apprentissage des paramètres par renforcement. Le fonctionnement de cette méthode passe par une vérification de la valeur d'adaptation de l'individu par rapport au contexte du problème. Dès lors, la performance du jeu de paramètres est évaluée selon une mesure de la diversité génétique. S'il y modification au niveau du jeu de paramètres suite à une évolution, alors il y a reclassement des jeux de paramètres selon leur performance.

6.1.2.3. La classe *Population*

La classe *Population* (figure 6.8) permet de créer un ensemble d'individus et de les faire évoluer grâce aux opérateurs génétiques. La population que l'on crée possède plusieurs paramètres qui régissent son évolution. Ces paramètres sont détaillés dans l'applet et certaines méthodes permettent une modification de ces paramètres tandis que d'autres en définissent les constantes.

La classe *Population* implémente donc toutes les méthodes qui permettent de définir ces paramètres de la population. Une fois que la population a été créée et que tous ces paramètres sont fixés, deux méthodes de cette classe permettent de passer d'une génération à la suivante. La plus simple effectue les étapes d'évolution suivantes :

1. *Croisement :* On répète les étapes suivantes jusqu'à ce que le nombre d'individus soit augmenté dans les proportions fixées par le taux de reproduction. Puis, on tue autant d'individus, parmi les plus mauvais, qu'il est nécessaire pour revenir au nombre d'individus initial de la population.

2. *Mutation :* On fait appel à la méthode *mutation* de chacun des individus en passant en paramètre la probabilité de mutation. Si l'individu est modifié par la mutation, on le range en fonction de sa nouvelle valeur d'adaptation.

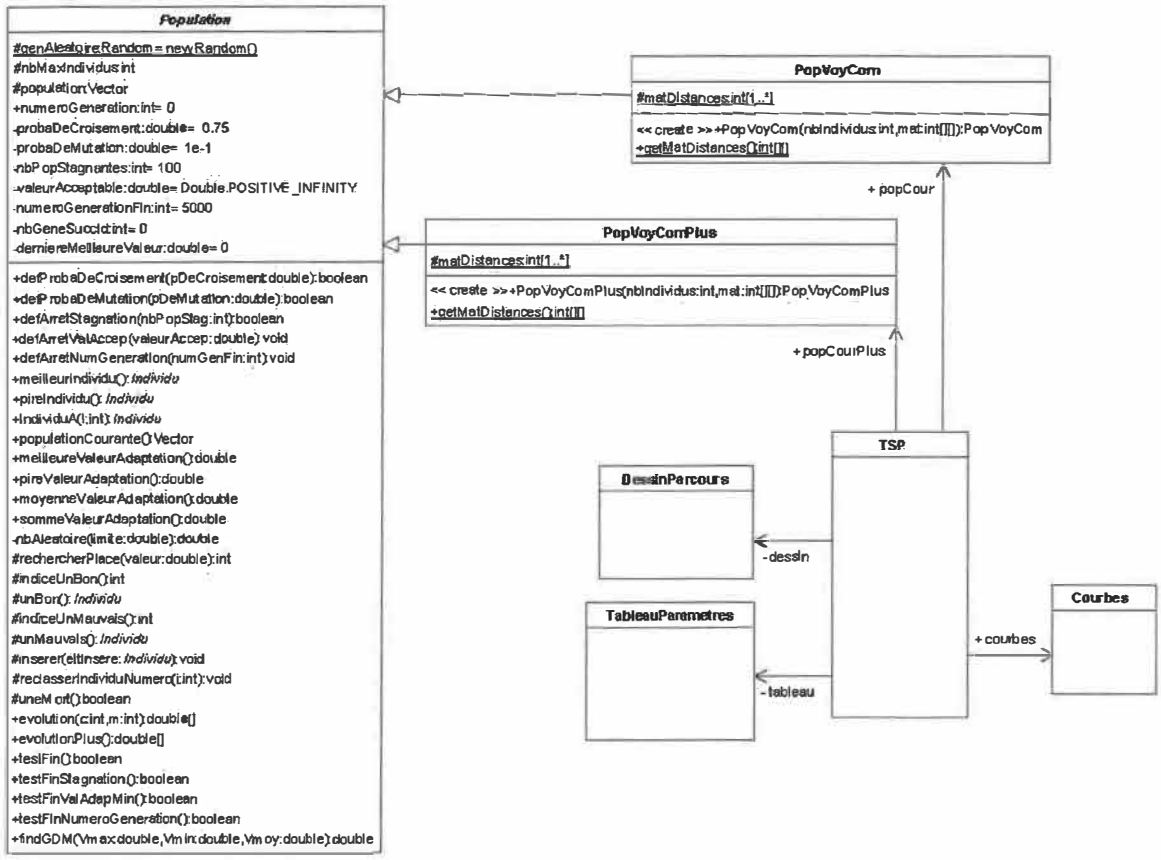


Figure 6.8. Diagramme UML de la classe *Population*

Dans l'autre méthode, on fait appel à une double évolution. On reprend les étapes décrites précédemment et on les applique aux solutions et aux jeux de paramètres de tous les individus de la population.

Entre deux évolutions, la classe propose de tester trois critères d'arrêt de l'algorithme. Les critères sont :

1. le nombre de stagnation avant arrêt ;
2. la valeur d'adaptation acceptable ;
3. le nombre maximum de générations.

D'autres méthodes implantées dans la classe permettent un suivi sur la population et donnent ainsi des informations comme : la moyenne des valeurs d'adaptation des individus, la meilleure valeur d'adaptation, le meilleur individu, le pire individu, la population courante, etc. Le meilleur individu de la dernière génération représente la solution au problème proposée par l'algorithme.

La sous-classe *PopVoyCom* représente la population des individus de la classe *Parcours*. Cette classe est assez simple car elle ne comprend qu'un constructeur pour la réalisation de la population. La classe *PopVoyComPlus* est aussi constituée d'individus, mais provenant de la classe *ParcoursPlus*. L'intérêt de cette classe n'est que de faciliter les appels au niveau de la classe moteur *TSP* lors des cycles d'évolution dans l'algorithme génétique.

La classe *TSP* est la classe maîtresse de l'applet et comprend l'interface utilisateur. Elle a besoin des 3 classes suivantes pour son fonctionnement :

1. *Courbe.java*

   Cette classe sert essentiellement à la création et à l'affichage du dessin des courbes de la meilleure valeur d'adaptation et de la valeur d'adaptation moyenne.

2. *DessinParcours.java*

   La classe *DessinParcours* effectue l'affichage des solutions (ou parcours possibles). Ces solutions sont sous forme d'un graphe. Dans ce graphe, chaque nœud représente une ville. Chaque arc du graphe indique un chemin entre deux villes et la pondération de l'arc est la distance qu'il y a entre les deux villes. Ce graphe est implémenté dans une structure de données sous forme d'une matrice (figure 6.9).

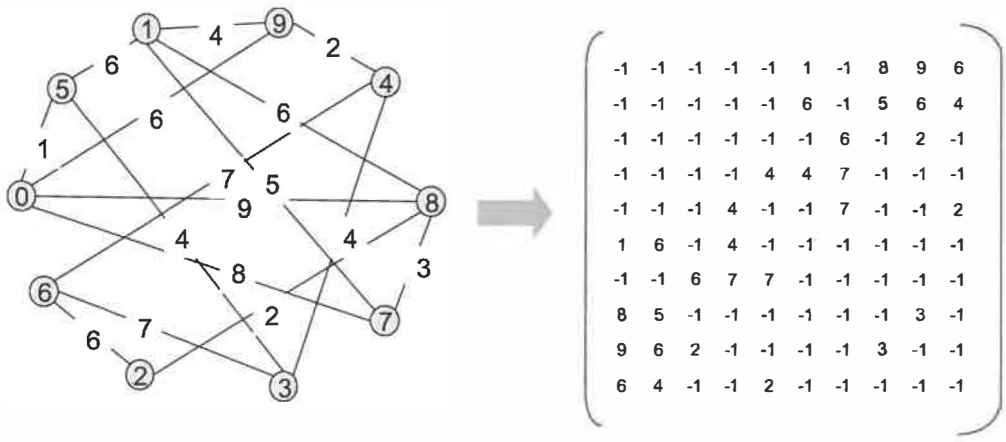


Figure 6.9. Représentation des solutions sous forme de graphe pour l'affichage dans l'applet et sous forme d'une matrice pour les calculs dans l'algorithme

3. *TableauParametres.java*

Cette classe représente un tableau contenant les paramètres que l'on veut récupérer dans les zones de texte au niveau de l'applet. Les paramètres que l'on retrouve sont :

- le nombre d'individus que devra comporter la population ;
- le nombre de générations de stagnation à partir duquel on interrompt l'exécution de l'algorithme ;
- la valeur d'adaptation acceptable à partir de laquelle on peut arrêter l'exécution ;
- le nombre maximum de générations que devra exécuter l'algorithme ;
- la probabilité de mutation de chaque gène de chaque individu de la population ;
- le taux de reproduction ou croisement entre les individus de la population ;
- le nombre de villes associé au graphe ;
- la complexité du graphe.

6.1.3. Fonctionnement des cycles d'évolution des algorithmes génétiques

Le fonctionnement de la classe *TSP* repose sur les cycles d'évolution des solutions. Ces solutions représentent en fait une population qui doit évoluer dans le temps. La population est représentée par plusieurs parcours tirés initialement au hasard. Les parcours représentent en soi une solution éventuelle. Ils sont donc représentés par un vecteur contenant toutes les villes une fois et une seule à partir du graphe (figure 6.10).

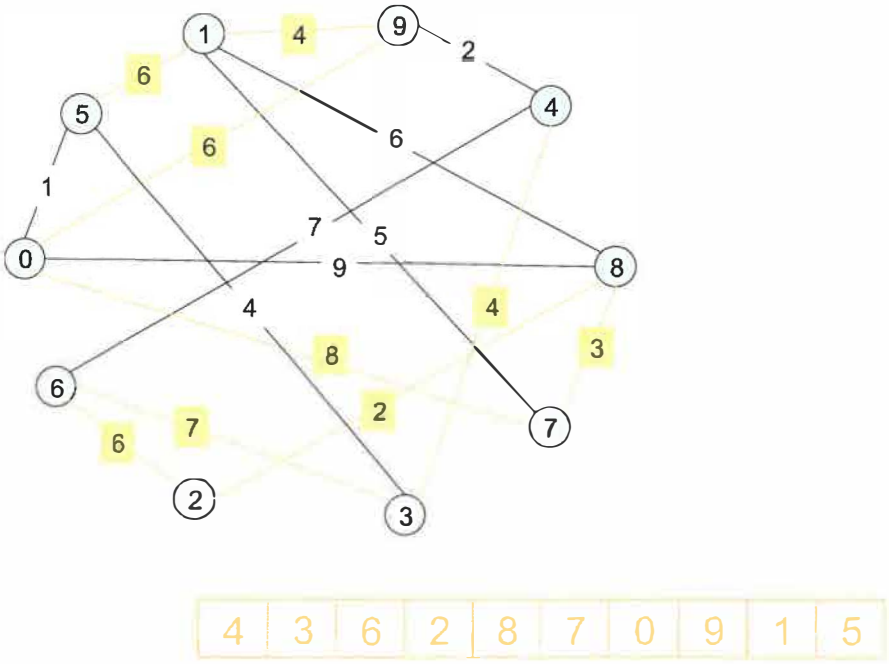


Figure 6.10. Représentation d'une solution sous forme d'un vecteur contenant toutes les villes choisies une seule fois

Dans la représentation d'une solution, il paraît évident qu'on peut avoir plusieurs représentations différentes du même parcours selon la ville de départ que l'on choisit. Un cycle typique d'algorithmes génétiques commence donc par la création d'une population. Par la suite, l'évolution de la population se fait par croisement de deux parcours. Selon le type d'opérateur de croisement, généralement on intervertit, entre les deux parcours, les parties qui se trouvent entre un point de séparation. Ainsi on augmente les chances d'avoir une solution plus performante.

Étant donné les deux parcours, il faut combiner pour la reproduction ces deux parcours pour en construire deux autres d'après les étapes suivantes :

1. À des fins de démonstration, un seul opérateur de croisement est défini pour la compréhension des étapes de reproduction. L'opérateur utilisé comprend deux points de croisement. On choisit alors aléatoirement deux points de coupe au niveau des individus parents, tel qu'illustré à la figure 6.11.

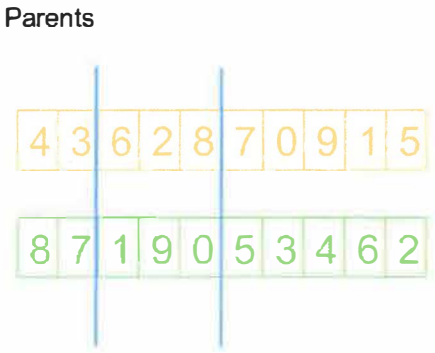


Figure 6.11. Représentation des individus parents avec deux points de croisement

2. On interverti, entre les deux parcours, les parties qui se trouvent entre ces deux points (figure 6.12).

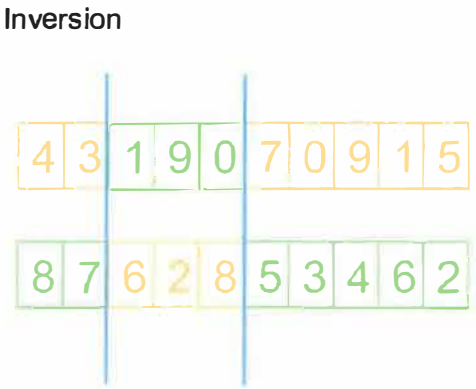


Figure 6.12. Représentation de l'étape d'inversion au niveau du croisement entre individus

3. On supprime (à la figure 6.13), à l'extérieur des points de coupe, les villes qui sont déjà placées entre les points de coupe.

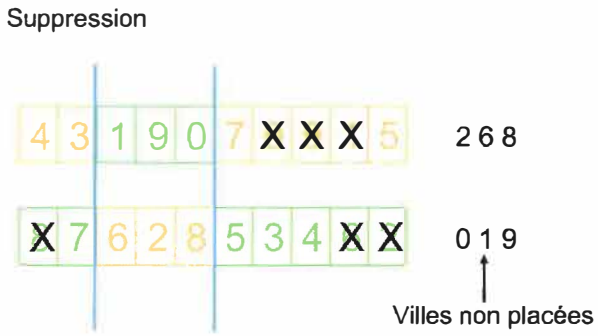


Figure 6.13. Représentation de l'étape de suppression des gènes au niveau du croisement

4. On recense les villes qui n'apparaissent pas dans chacun des deux parcours. Puis, on remplit aléatoirement les trous dans chaque parcours pour donner des individus enfants comme dans la figure 6.14.

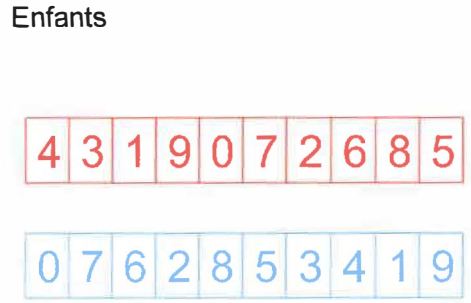


Figure 6.14. Représentation des individus enfants à la fin du croisement

Le processus d'évolution implique aussi des mutations qui peuvent apporter de nouvelles solutions (figure 6.15). La mutation se produit dans un individu au niveau des villes. Toujours selon l'opérateur de mutation utilisé, il s'agit de modifier un ou plusieurs des éléments constitutifs de la solution, ici c'est une ville. Quand une ville doit être mutée, on choisit aléatoirement une autre ville et on la remplace par une autre.

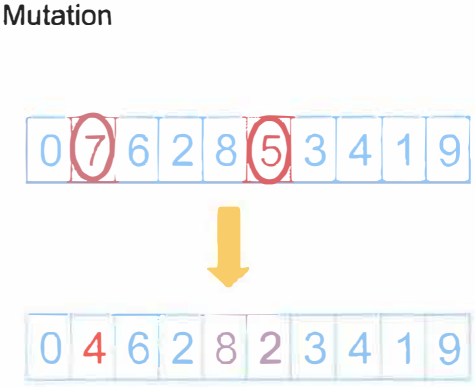


Figure 6.15. Représentation de la mutation au niveau des individus

Les étapes décrites ci haut décrivent un cycle d'évolution pour un algorithme génétique simple. Pour inclure l'approche par auto-adaptation des paramètres, d'autres étapes viennent enrichir le cycle d'évolution. En effet, avant même de faire un cycle d'évolution de la population de

solutions, les jeux de paramètres des individus de la population doivent être évolués et évalués par rapport au contexte du problème. Au départ, on cible le meilleur individu de la population, il servira comme valeur de référence pour les étapes subséquentes (figure 6.16).

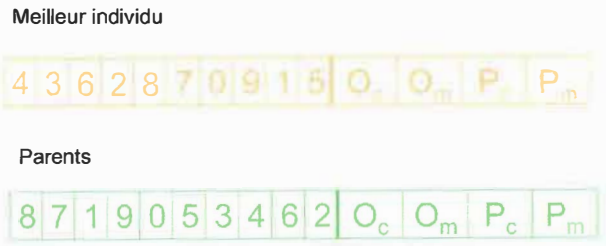


Figure 6.16. Représentation du meilleur individu et d'un parent quelconque de la population avec leur jeu de paramètres

L'une des premières étapes consiste à prendre tous les jeux de paramètres de chaque individu de la population et de les faire passer par un cycle d'évolution (figure 6.17).

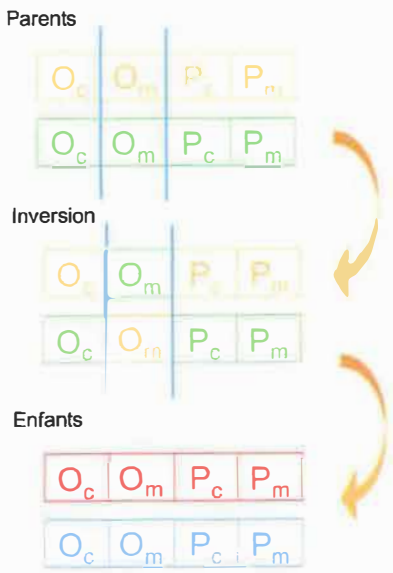


Figure 6.17. Représentation d'un cycle d'évolution des jeux de paramètres pour un individu de la population

Ce cycle, au niveau des paramètres, comprend aussi une étape de mutation illustrée à la figure 6.18. Les opérateurs et probabilités de croisement ou de mutation sont déterminés par le meilleur jeu de paramètres que l'individu possède.

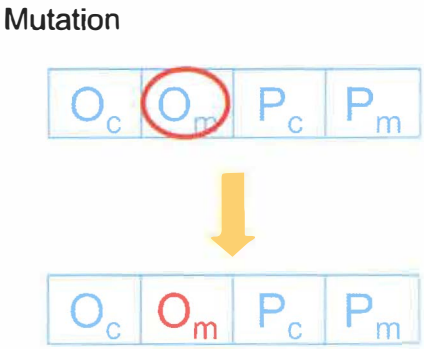


Figure 6.18. Représentation de la mutation au niveau des jeux de paramètres d'un individu

La valeur d'adaptation des nouveaux jeux de paramètres est calculée. Par la suite, on évalue la population de jeux de paramètres selon leur niveau d'implication dans la résolution du problème. Cette évaluation passe par la mesure de diversité génétique discutée dans la section 5.1.5 et qui est en lien très étroit avec la résolution du problème. On peut alors établir une valeur de renforcement pour les différents jeux de paramètres (figure 6.19).

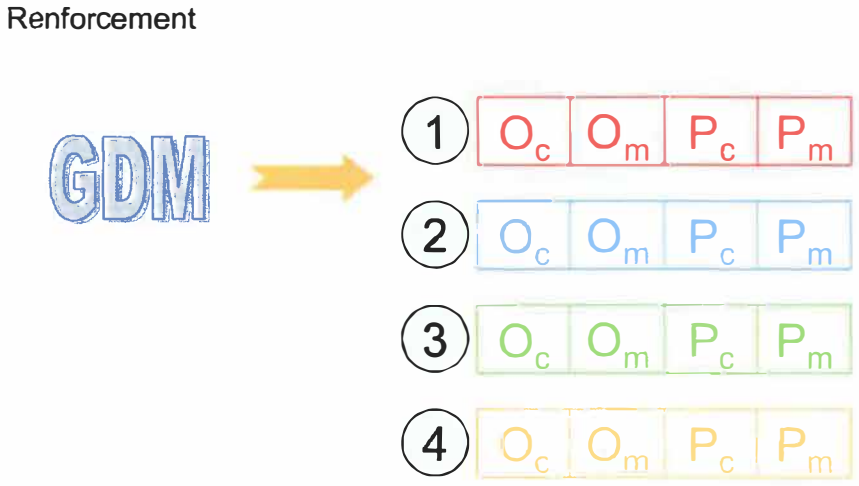


Figure 6.19. Représentation de l'application de la valeur de renforcement au niveau des jeux de paramètres en fonction de la mdg

Le meilleur jeu de paramètres peut ainsi être déterminé et remplacé au besoin dans le chromosome de l'individu (figure 6.20). On peut alors appliquer le cycle d'évolution sur la population de solutions. Les étapes sont similaires à la description décrite ci-haut, à l'exception près que les opérateurs génétiques et leurs taux associés sont choisis par rapport au meilleur individu participant à la reproduction. En effet dans l'application d'un cycle d'évolution avec l'approche d'un algorithme génétique simple, le processus de croisement implique des opérateurs génétiques fixes (et leur taux associés) durant tous les cycles.

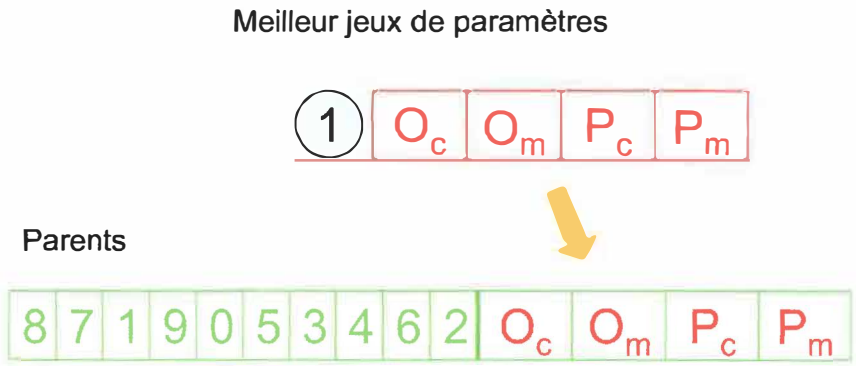


Figure 6.20. Représentation du meilleur jeu de paramètres après apprentissage

6.2. Prototype MAAG

L'expérimentation du prototype de méta-apprentissage s'est fait en terme d'une intégration future dans un système au niveau des opérations urbaines du nom de *SCIPIO Urban suite* (figure 6.21). Ce système est réalisé par la Défense Nationale et avec la participation de Thales Systems Canada.

Le projet porte le nom de SCIPIO en référence au grand Général romain Scipio Africanus qui considérait la ville de Carthage d'un point de vue stratégique. En effet, selon lui, Carthage était la région centrale pour contrôler toute la Méditerranée. Par conséquent, *Scipio Urban suite* considère le module central *AIThink* comme la plaque centrale du système et est responsable de cinq modules du système : *UrbanWish*, *OptiPath*, *OptiEvents*, *UrbanDispatch* et *CounterDeception*.

Figure 6.21. Fenêtre d'accueil du système SCIPIO Urban Suite

Le module central *AIThink* comprend toute la connaissance disponible et joue le rôle d'agent intelligent dans la globalité du système, tout comme le cerveau pour l'humain. Le projet de prototype de méta-apprentissage prend alors un sens en faisant partie intégrante du système d'intelligence. Il s'intègre parmi d'autres composantes intelligentes et se distingue par ses capacités d'apprentissage pour donner à *AIThink* toute la robustesse requise.

Le prototype d'apprentissage repose sur les connaissances apprises par le système et aussi sur celles venant de l'extérieur. De par la complexité de SCIPIO, l'expérimentation se limite au niveau d'un seul module pour l'apprentissage de connaissances. Ce module ce nomme *Optipath* et vise principalement la recherche de parcours optimum dans un environnement urbain dynamique [Pigeon and Van Chestein, 2004]. La dynamique se définit par tout changement de conditions dans le temps qui viendront modifier un parcours. Un bon système peut ainsi anticiper à l'avance ces changements et donner un parcours valide dans le temps.

Les types de modification d'un parcours peuvent être de nature menaces (foule, véhicule ennemi,...) ou de nature à affecter le déplacement du véhicule de l'utilisateur du système. On constate que la recherche de chemin est un problème constant dans l'environnement urbain et particulièrement lors d'opérations militaires ou civiles, où le temps est primordial.

6.2.1. La problématique du deuxième parcours optimum

Optipath est un outil puissant pour trouver un parcours optimum mais il est fortement dépendant des renseignements entrants. L'une des sources principales de renseignement est de nature HUMINT et c'est la source d'intelligence qui est la plus complexe. Le simple fait de vouloir se déplacer d'un endroit à un autre dans un milieu urbain devient rapidement chaotique si le traitement des renseignements n'est pas réalisé efficacement. La nécessité de parcours alternatifs devient alors urgente sous certaines conditions.

Il apparaît clairement qu'il faut résoudre le problème suivant : suite à une requête de parcours dans Optipath, trouver le deuxième chemin optimum selon les conditions de la première requête. Par exemple, si on cherche le chemin le plus court en temps entre deux points et sous certaines conditions, on obtient le parcours illustré à la figure 6.22.

Si on suppose qu'une menace hostile peut être rapportée sur un des segments du parcours, l'utilisateur peut ainsi envisager l'utilisation potentielle d'un deuxième parcours optimum (figure 6.23).

Figure 6.22. Illustration du parcours planifié entre deux points dans un réseau routier

Figure 6.23. Illustration du parcours alternatif suite à l'apparition d'une menace sur le réseau routier

La recherche d'un second parcours peut être réalisée par l'apprentissage. Pour tester le système d'apprentissage, il faut un ensemble de renseignements. On peut alors utiliser les informations que renferment les parcours provenant de plusieurs requêtes d'*Optipath* sous diverses conditions. Ces conditions sont aussi variées qu'il y a de paramètres pour déterminer la recherche d'un parcours (*Quickest, Shortest, Safest,...*). Ainsi, en accumulant plusieurs parcours, on obtient une banque de données ou de renseignements qui peuvent être traités par le système d'apprentissage. En donnant les conditions initiales du premier parcours, le système peut ainsi apprendre à déterminer le second parcours optimum. La provenance de ces renseignements via *Optipath* est de nature ELINT. L'apprentissage permet aussi de traiter des renseignements de nature HUMINT suivant des entrées de parcours de la part de l'utilisateur.

L'obtention du deuxième parcours optimum par le système d'apprentissage se doit d'être évalué en termes de comparaison de la performance de l'apprentissage par rapport à un système efficace. Ainsi, l'utilisation d'*Optipath* comme *benchmark* est employée en appliquant une seconde requête sous contrainte. Ainsi, en bloquant volontairement le premier segment, on force le module *Optipath* à trouver un second parcours tout en respectant les conditions initiales.

Cette application simple permet de mieux comprendre l'apprentissage et ainsi de pouvoir transposer avec plus de confiance le prototype à des situations où les résultats ne sont pas toujours connus d'avance.

6.2.2. Description de l'environnement

Lorsque l'application est exécutée, une interface de départ (figure 6.24 à la page 131) apparaît à l'écran pour présenter le titre du projet. Cette interface permet également de charger le graphe en mémoire, ce qui prend quelques secondes. Le graphe représente les données de base, soit le système routier de la ville de Québec et ses environs tels que montrés aux figures 6.25 et 6.26. Le graphe est orienté, pondéré et composé de 38 894 arcs et de 15 422 nœuds. La fenêtre principale de l'application (Figure 6.27 à la page 131) apparaît lorsque le graphe est complètement chargé en mémoire.

La fenêtre de l'application se décompose en deux blocs majeurs, soit l'affichage des détails des parcours et l'exécution de l'algorithme d'apprentissage. Ces deux points sont traités dans les sous-sections qui suivent.

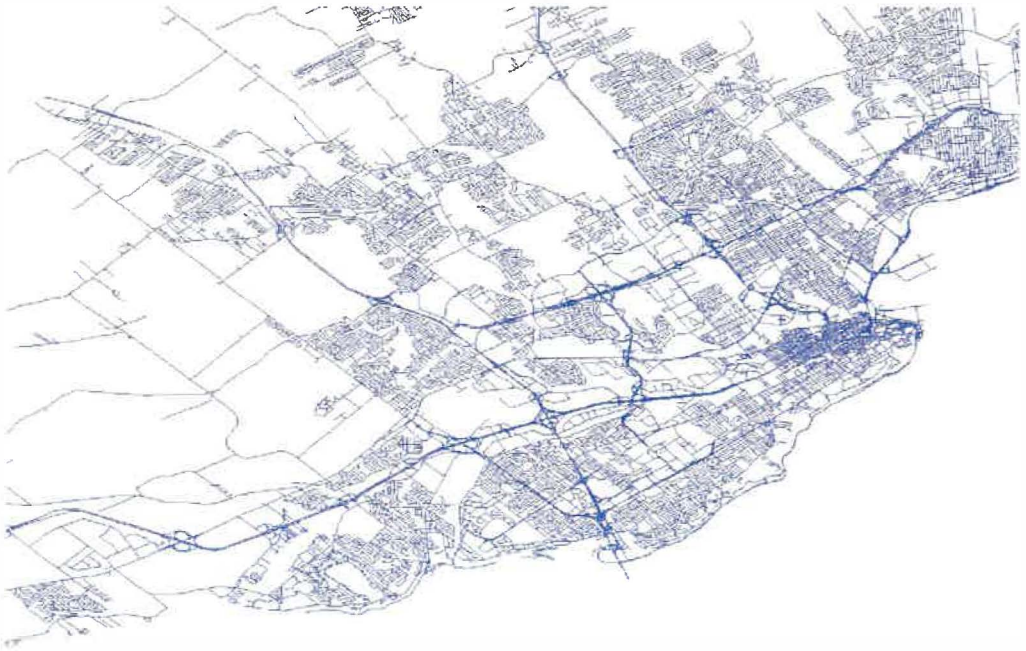


Figure 6.25. Représentation du système routier de la ville de Québec et ses banlieues

Figure 6.26. Image satellite de la ville de Québec avec le système routier

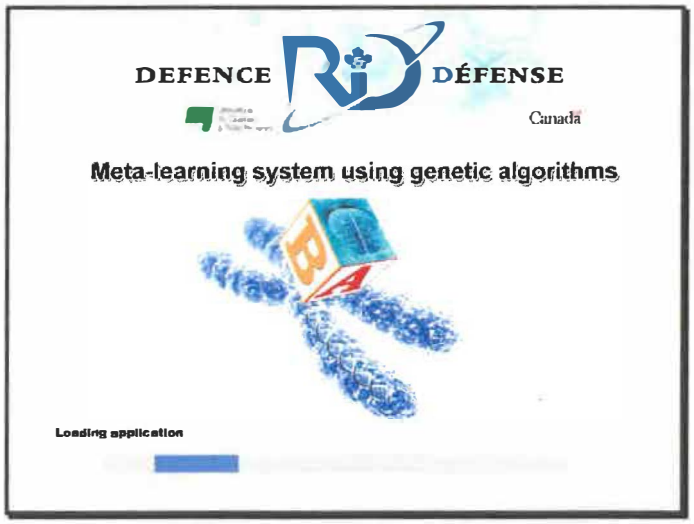


Figure 6.24. Interface de départ lors de l'exécution de l'application

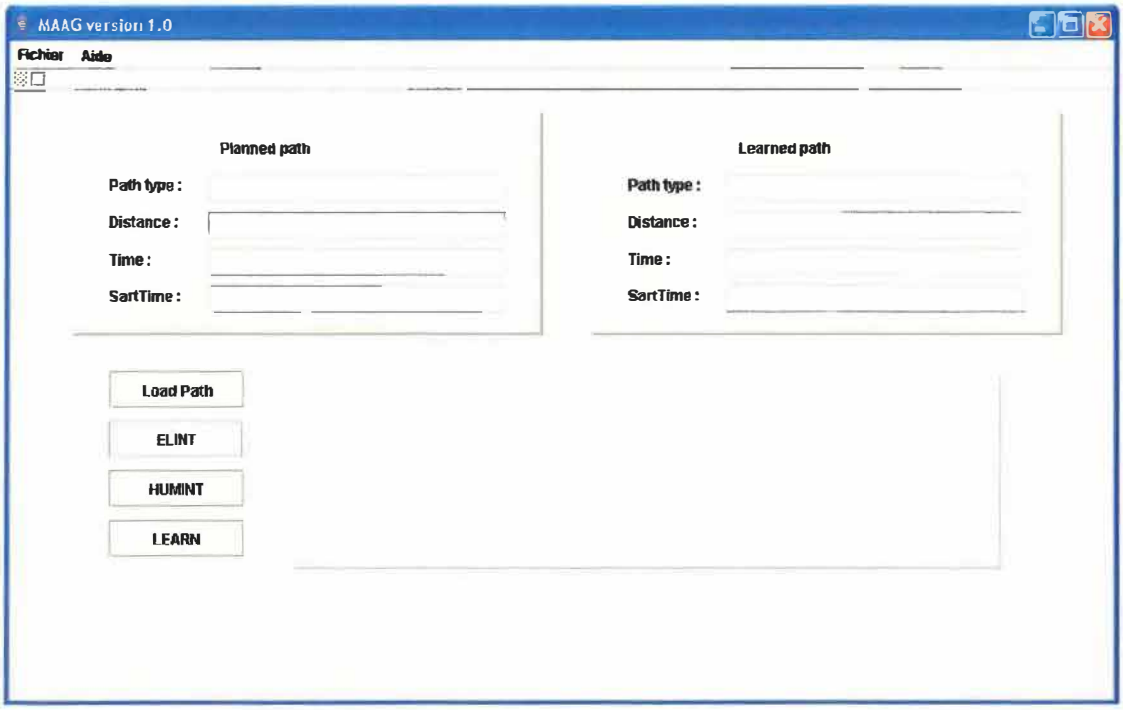


Figure 6.27. Fenêtre principale de l'application MAAG

6.2.2.1. Affichage du parcours planifié et du parcours appris

L'affichage des résultats de parcours se fait au niveau du parcours qui est planifié à l'aide du module *Optipath*, ainsi qu'au niveau du parcours qui est appris à l'aide du méta-apprentissage. Dans les deux cas, les éléments de description du parcours trouvé sont les suivants :

- **Path Type**
  Affichage du type de parcours dans cette zone texte. Il y a deux types de parcours que l'on retrouve dans l'application : *Shortest* et *Quickest*. Un parcours de type *Shortest* correspond au chemin qui parcourt la distance minimum. Un type *Quickest* représente le chemin le plus court en temps de parcours.

- **Distance**
  Affichage de la distance totale du parcours représentée en kilomètres par heure. La distance totale est la somme des distances de tous les segments qui forment le parcours partant du point de départ et se terminant à l'arrivée.

- **Time**
  Affichage du temps de parcours total représenté en minutes et secondes. Le temps de parcours total est la somme des temps de parcours de tous les segments qui forment le parcours partant du point de départ et se terminant à l'arrivée. Pour chaque segment, le temps de parcours est calculé selon le rapport de la distance sur la plus petite des vitesses maximum entre le véhicule et la limite permise sur le segment.

- **StartTime**
  Affichage du temps pris au moment de la requête du parcours. Le temps est synchronisé avec le temps horloge du système.

6.2.2.2. Les boutons de commande

Les boutons de commande servent au fonctionnement et au contrôle de l'exécution de l'application. L'ordre de présentation des boutons au niveau de la fenêtre principale (figure 6.27) suit un ordre logique de fonctionnement. Cette disposition des étapes fonctionnelles permet de réaliser le processus général pour exécuter une session d'apprentissage. L'ordre logique fait intervenir le bouton *LoadPath*, *ELINT*, *HUMINT* et *Learn*.

- **Load Path**
  Ce bouton a comme fonction la sélection du parcours que l'on veut charger dans l'interface principale. Ce parcours servira aussi comme référence de base pour déterminer

le second meilleur parcours. Lorsque le bouton est activé, une nouvelle fenêtre apparaît (figure 6.28) et propose plusieurs choix de parcours à l'utilisateur.

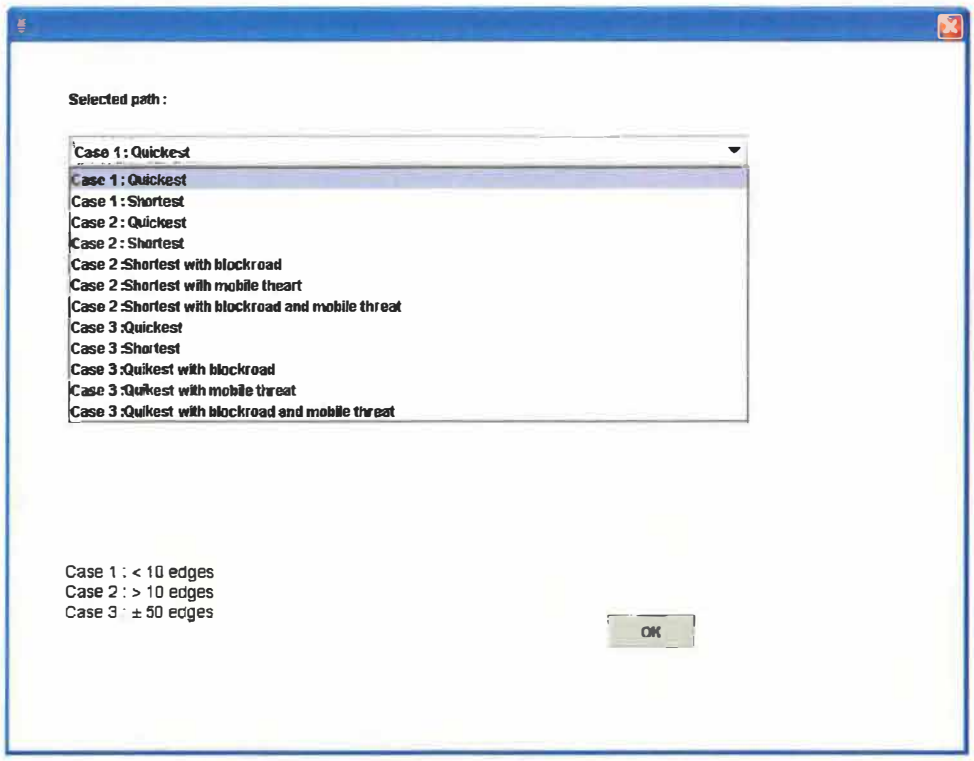


Figure 6.28. Fenêtre activée par le bouton *LoadPath* pour la sélection du cas d'utilisation

À des fins de comparaison de performance, trois niveaux de complexité de parcours ont été déterminés (cas 1, cas 2 et cas 3). À l'intérieur de chaque cas, le point de départ et le point de destination sont les mêmes, mais ces points diffèrent entre les cas. La complexité des cas se définit en terme de nombre d'arcs ou de segments composant le parcours. Ainsi, pour le premier cas d'utilisation, une complexité de moins de 10 arcs par parcours est définie. À l'intérieur d'un cas de complexité, on retrouve une subdivision en fonction du type de parcours (*Shortest* et *Quickest*). Également, une sous subdivision en une variante d'événement pouvant survenir sur un segment. Deux possibilités sont alors notées :

- Un barrage routier. Dans ce cas, le segment bloqué par un barrage routier n'est jamais accessible dans le temps pour un parcours.

- Une menace mobile. La menace mobile fait référence, par exemple, à un véhicule qui peut engendrer un risque si on se retrouve sur le même segment au même moment. Le segment ou les segments projetés ne sont pas accessibles pour un parcours dans un temps donné.

Lorsqu'un parcours est sélectionné et que le bouton *OK* est enfoncé, les calculs de recherche de chemin sont exécutés par le module *Optipath*. Le parcours optimum trouvé est alors affiché au niveau de la fenêtre principale. Ce parcours est un parcours planifié et sa description est détaillée. De plus, une nouvelle fenêtre apparaît (figure 6.29) pour illustrer le parcours planifié.

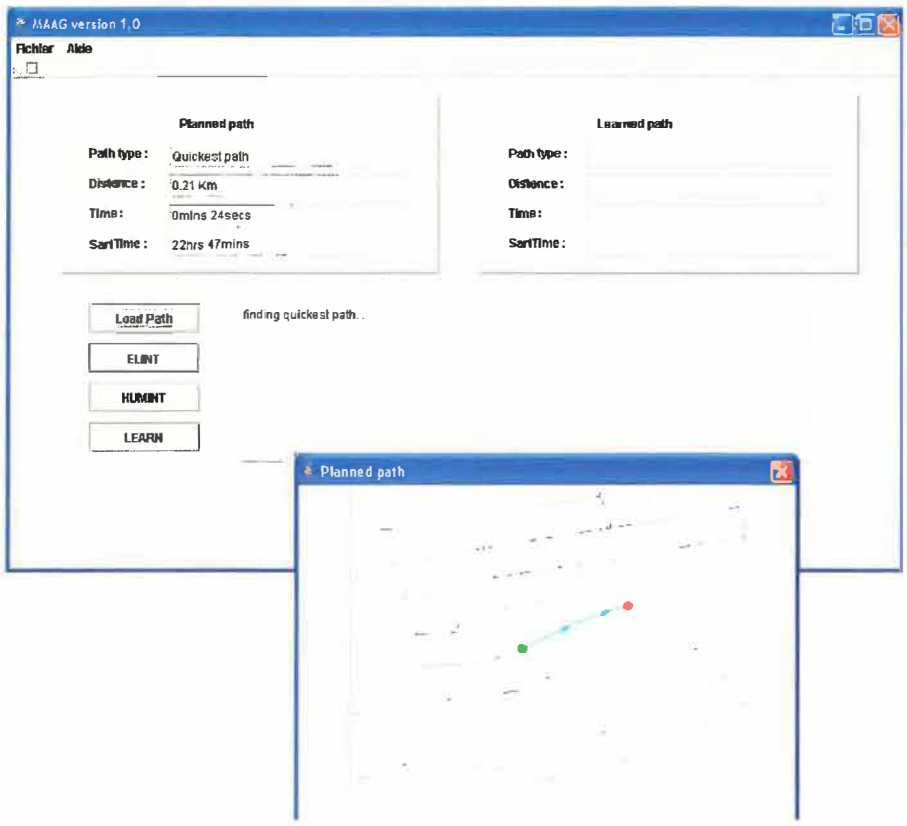


Figure 6.29. Fenêtre principale avec l'affichage du parcours planifié. Fenêtre secondaire avec l'illustration du parcours planifié

- **ELINT**

  Lorsque le bouton *ELINT* est enfoncé, il génère des renseignements de nature électronique. Les renseignements de nature électronique proviennent de plusieurs requêtes de chemin au module *Optipath* en conservant les points de départ et d'arrivée du parcours planifié. Les requêtes sont réalisées sous diverses conditions pour obtenir un ensemble non homogène de renseignements.

  Les conditions de recherche de parcours sont regroupées par type et les différentes requêtes représentent une combinaison des ces types. Il y a autant de requêtes qu'il y a de combinaisons possibles.

  Types de recherche

  - *Shortest*
  - *Quickest*

  Le type de recherche définit si on optimise la recherche de parcours selon la distance ou selon le temps.

  Comportements de la recherche

  - Pessimiste
  - Optimiste
  - Moyenne

  Le comportement de la recherche correspond au niveau de confiance que l'on accorde aux évènements dynamiques. Dans un cas pessimiste, on envisage les événements sous leur plus mauvais aspect. Dans un cas optimiste, tout est pour le mieux dans l'environnement. La moyenne reflète, par exemple, une vitesse moyenne du véhicule sur les différents types de route présente dans le système routier.

  Contraintes de la recherche

  - *Block time*
  - *Without threats*

- *With static threat*
- *With mobile threat*

Les contraintes au niveau de la recherche sont multiples et se représentent soit par un délai au point de départ ou par la présence d'une ou de plusieurs menaces. L'action des menaces se traduit par le blocage d'un ou de plusieurs segments, les rendant inaccessibles pour un parcours.

Dans le cas précédant (figure 6.29), les résultats des différents parcours trouvés sont illustrés dans la figure 6.30. Les parcours sont transformés en vecteurs de renseignement afin d'être traités par l'apprentissage.

Figure 6.30. Illustration des différents parcours trouvés pour déterminer les vecteurs de renseignement

- **HUMINT**

  Le bouton HUMINT permet la génération de renseignement de type humain. Une fenêtre apparaît (figure 6.31) pour permettre l'acquisition des informations de la part de l'usager.

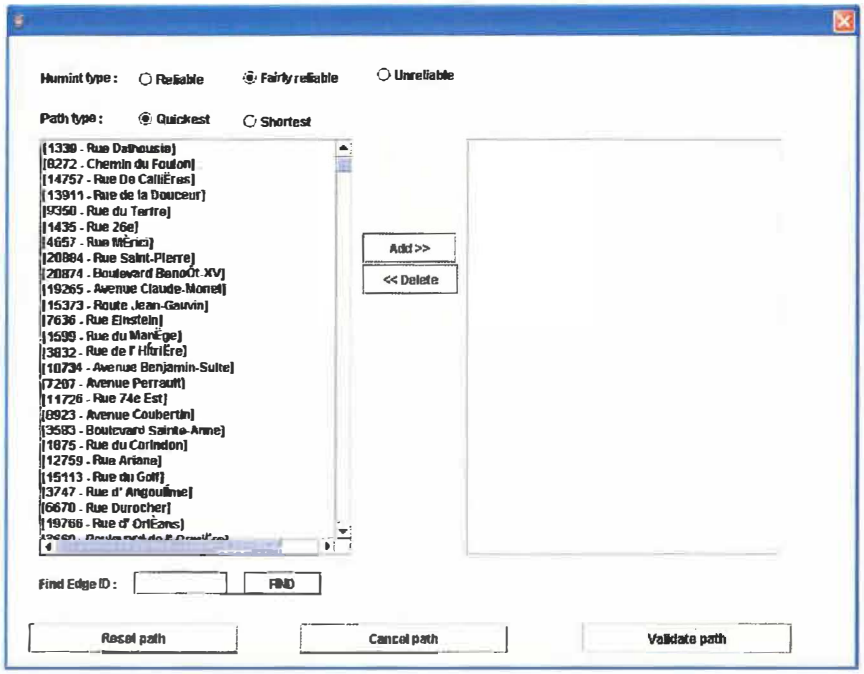


Figure 6.31. Fenêtre d'acquisition pour les renseignements de type HUMINT

L'usager doit spécifier son niveau de fiabilité selon trois critères : fiable, assez fiable ou non fiable. Il doit déterminer le type de parcours qu'il propose et, finalement, sélectionner les différents segments qui composent le parcours proposé. Un outil de recherche par numéro d'identifiant de segment est situé au bas de la liste de sélection. Cet outil sert à la recherche d'un segment précis dans la liste.

À la fin de l'acquisition des informations, l'utilisateur a le choix entre :

- recommencer la sélection des segments pour un autre parcours ;
- annuler la saisie de l'information et revenir à la fenêtre principale ;
- valider son parcours en termes de renseignement pour le système.

Pour la dernière option, le parcours n'est pas un parcours valide, un message d'avertissement apparaît et permet de continuer ou de recommencer la sélection des segments. Si l'utilisateur

approuve un chemin qui n'est pas valide, ceci permet alors d'avoir un renseignement HUMINT ou la source est fiable mais que l'information est non crédible.

- **LEARN**

  Le bouton *Learn* correspond à l'exécution de l'algorithme d'apprentissage. Le cycle d'apprentissage est alors réalisé et passe par les étapes de contexte, de catégorisation, de sélection et d'apprentissage. Une description plus précise de l'exécution au niveau de la classe *Learn* est donnée à la section 6.2.3. Lors de l'activation du bouton, une liste des renseignements disponibles ainsi qu'un contexte d'apprentissage sont passés à l'algorithme. La liste de renseignements provient autant de la génération réalisée par le bouton *ELINT* que *HUMINT*. Le contexte d'apprentissage est défini à partir des critères du parcours planifié. La résultante de l'apprentissage se traduit par le deuxième meilleur parcours et son détail est affiché au niveau de la fenêtre principale et également dans une nouvelle fenêtre (figure 6.32)

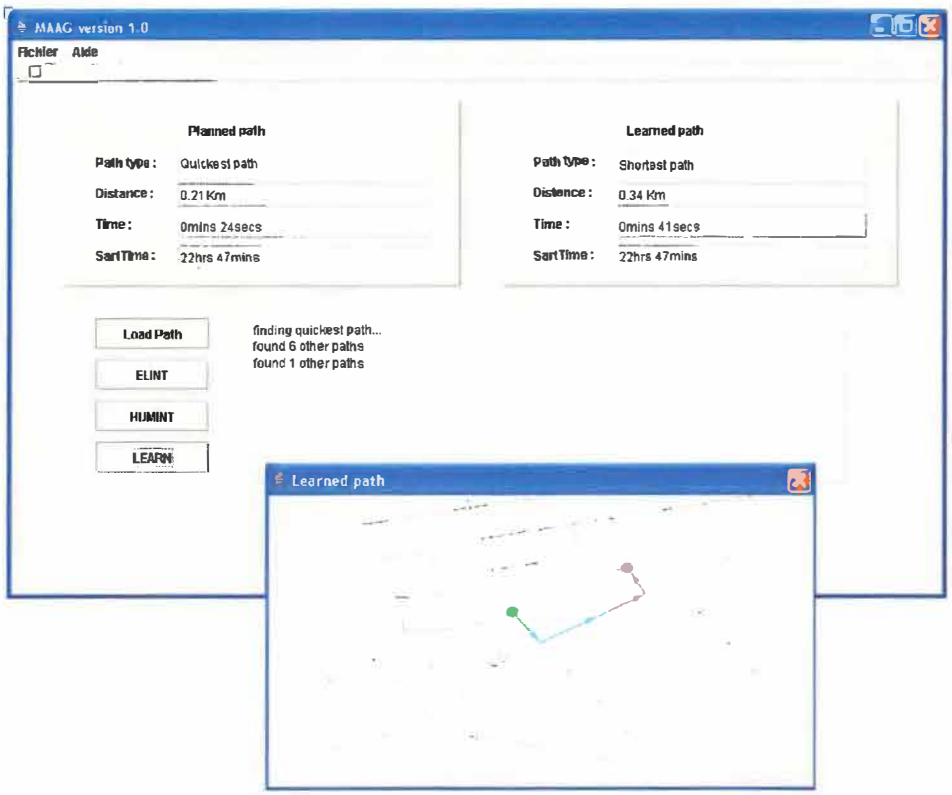


Figure 6.32. Fenêtre principale avec l'affichage du parcours appris. Fenêtre secondaire avec l'illustration du parcours appris

6.2.3. Implémentation de l'apprentissage dans le prototype

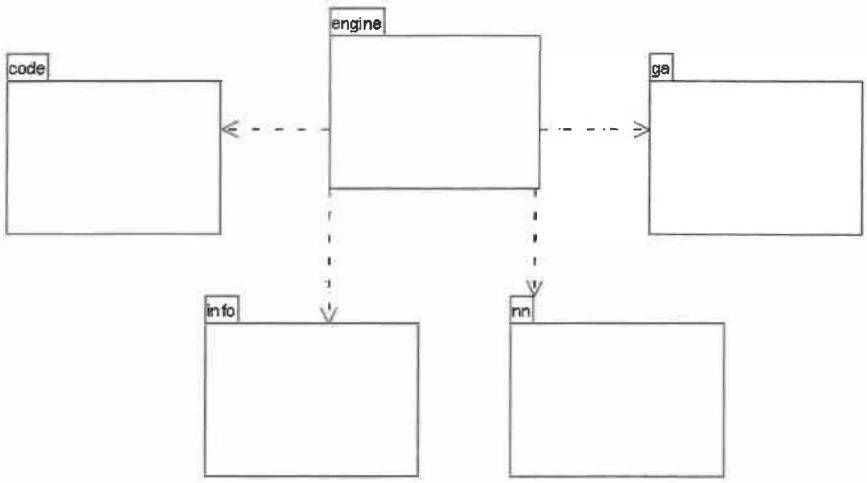


Figure 6.33. Représentation des packages et des liens correspondants pour l'application MAAG

L'implémentation du prototype a été développée en cinq couches pour donner un état cohérent à l'environnement (figure 6.33). Chaque couche représente un package incluant des classes spécifiques pour le fonctionnement de l'apprentissage. Les flèches du package *engine* vers les autres packages indiquent le lien de dépendance de celui-ci envers les autres packages. Les packages du projet *SCIPIOLearning* sont donc :

6.2.3.1. Le package *engine*

C'est le package principal de l'application et il contient la représentation graphique (GUI) et l'aspect moteur de l'environnement. Pour fonctionner, il dépend de quatre autres packages présents dans le projet. Le diagramme de classe de la figure 6.34 représente les classes *ApplicationGUI*, *LearnGUI*, *Learn* et *learningContext* qui composent ce package.

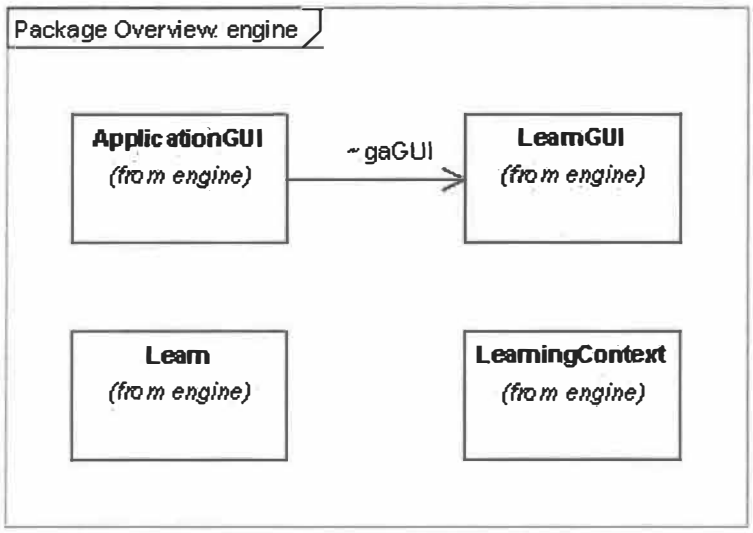


Figure 6.34. Diagramme UML du package *engine*

La classe *ApplicationGUI* est une classe d'interface qui crée et gère un autre composant d'interface spécifique à l'apprentissage (LearnGUI). La classe *LearnGUI* représente cette interface d'apprentissage. À la création de celle-ci, un graphe orienté et pondéré est chargé en mémoire. L'interface initialise les divers composants fonctionnels de l'application.

La classe *learningContext* contient toutes les informations définissant le contexte d'apprentissage, c'est-à-dire l'objectif à atteindre durant une itération d'apprentissage. Cette classe renferme des informations sur le type de recherche de parcours, l'estimation du temps de parcours, la distance totale de parcours, les points d'arrivée et de départ, le niveau d'urgence du déplacement de la part de l'utilisateur et les autres informations décrivant un parcours complet.

La classe *Learn* est la classe clé de l'application. Elle gère l'exécution des différents algorithmes d'apprentissage. L'appel à cette classe implique de lui passer une liste des renseignements disponibles ainsi qu'un contexte d'apprentissage. La liste des renseignements sert à la création de la topologie du réseau de neurones. Elle sert également, conjointement avec le contexte d'apprentissage, à l'initialisation des valeurs dans le réseau afin de produire une population de poids d'apprentissage. Ces poids sont optimisés pour donner le meilleur vecteur de poids. Une seconde optimisation centrée sur les renseignements est appliquée pour déterminer le meilleur

vecteur de renseignement. Le résultat de l'apprentissage est récupéré via la méthode *getSolution* et représente le meilleur deuxième parcours sous la forme d'un objet Information.

6.2.3.2. Le package *ga*

Ce package contient l'aspect apprentissage génétique de l'application, soit les classes des algorithmes génétiques. Tel que mentionné dans la section 6.1.2, l'implémentation des classes de l'algorithme génétique a été développé à partir de trois classes abstraites qui constituent le fondement logique des algorithmes génétiques. Ces classes sont : *Gene.java*, *Individu.java* et *Population.java*. La figure 6.35 représente le diagramme UML de ce package.

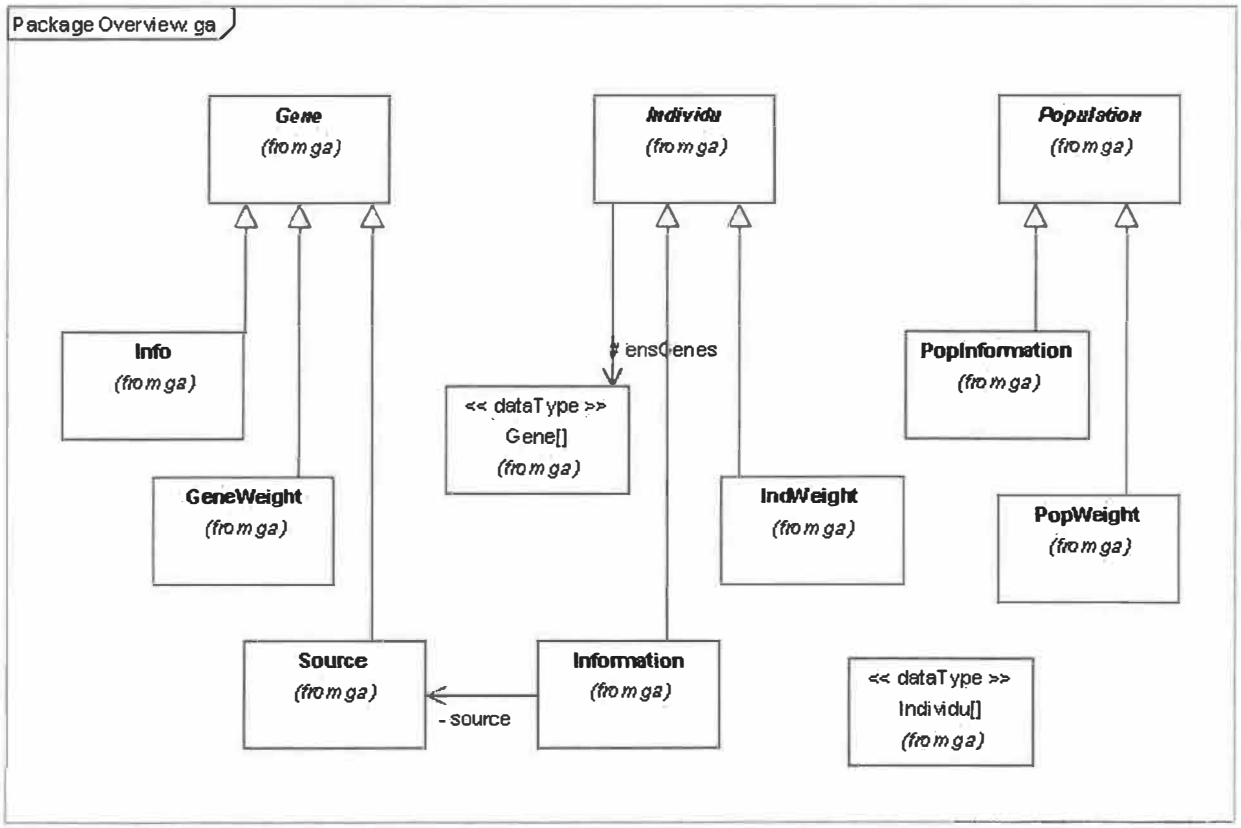


Figure 6.35. Diagramme UML du package *ga*

La super classe *Gene* représente l'élément de codage des individus qui composent la population. Cette classe ne comporte qu'une méthode abstraite de la mutation d'un gène. Les éléments de codage présents dans cette application se retrouvent sous les classes *GeneWeight*, *Info* et *Source*. La première classe permet de construire un gène en lui affectant un double comme élément de base. Les deux autres classes représentent les constituants d'un objet Information. L'objet Info se compose d'un nom, d'une valeur de type *Object*, d'un poids de crédibilité, d'un poids d'apprentissage et d'un poids delta. Le poids delta correspond à une différence du poids d'apprentissage si celui-ci a subi un changement. L'objet Source contient également un nom, un poids de fiabilité, un poids d'apprentissage et un poids delta. Les poids présents dans ces classes sont importants pour le fonctionnement de l'apprentissage car ils indiquent le niveau d'importance attribué à l'objet lui-même.

La super classe *Individu* permet de créer un objet Individu qui représente l'élément de base d'une population. La classe *Information* représente un élément *Individu* définissant un vecteur d'information. Ce vecteur est composé d'une liste d'*Info* et d'un objet *Source*. La méthode *initWeightGene* de cette classe permet d'initialiser les poids des gènes lors de la construction de l'objet *PopInformation*. Pour le fonctionnement de l'algorithme génétique, des méthodes de croisement des individus et de calcul de la valeur d'adaptation sont présentes. La classe *IndWeight* réalise un individu comportant un vecteur de poids. Un constructeur présent dans cette classe permet de construire un individu vecteur à partir d'un individu *Information*.

La super classe *Population* permet de créer un ensemble d'individus et de le faire évoluer grâce aux opérateurs génétiques. Il y a création de deux populations différentes, l'une attribuée à l'apprentissage des vecteurs de renseignement et l'autre pour l'évolution des poids d'apprentissage. Dans le cas de la classe qui optimise les vecteurs de renseignement, soit la classe *PopInformation*, la méthode d'évolution se limite à une seule itération et l'évaluation des individus est bonifiée par la notion de gènes dominants. Cette notion affecte le calcul de la valeur d'adaptation en accordant plus d'importance à certains gènes.

6.2.3.3. Le package *nn*

Ce package contient l'aspect logique neuronal de l'application, soit les classes des réseaux de neurones. Ces classes sont illustrées à la figure 6.36 en respectant la logique d'une architecture neuronale artificielle définissant un réseau de neurones.

La classe *Perceptron* représente l'architecture du réseau de neurones. Elle se compose de différentes couches (*Layers*) de neurones. Des méthodes propres à la classe permettent de réaliser les connexions entre les couches du réseau et aussi d'associer des vecteurs de renseignement en entrée à des vecteurs résultants. La méthode *initPerceptron* prend des vecteurs

de renseignement et le vecteur du contexte d'apprentissage et les propage dans le réseau. Les neurones d'entrée sont alors initialisées avec chacun des vecteurs de renseignement. L'information donnée à un réseau de neurones est propagée couche par couche de la couche d'entrée à la couche de sortie en passant par soit aucune, une ou plusieurs couches intermédiaires.

La classe *Layer* définit les couches du réseau et se compose de vecteurs de neurones pour chaque couche. Le constructeur de cette classe prend en paramètre le nom de la couche ainsi que le nombre de neurones qui constituent la couche. La méthode *computeOutput* ne fait que l'énumération de chaque vecteur et appel à son tour la méthode *computeOutput* de la classe neurone.

La classe *Synapse* modélise la connexion pondérée entre deux neurones. Il y a présence d'une connexion synaptique qui est en fait une jonction ou communication entre deux neurones.

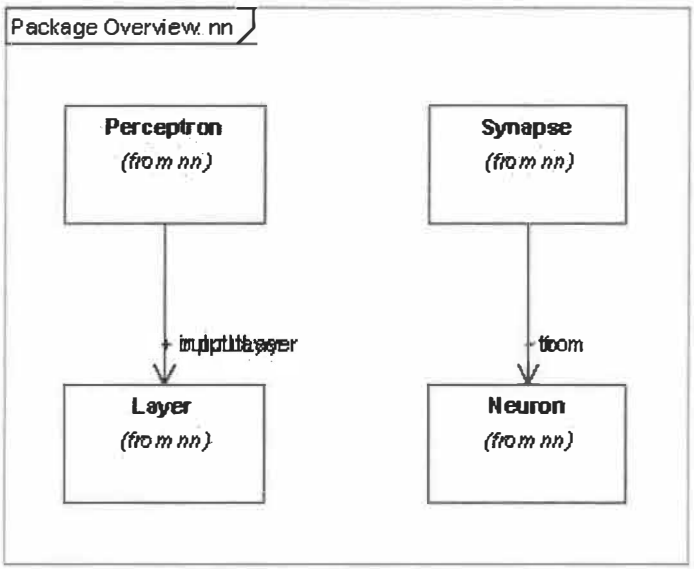


Figure 6.36. Diagramme UML du package *nn*

La classe *Neurone* représente l'élément de base constituant le réseau de neurones. Dans le cas présent, il représente les éléments constituant un vecteur d'information. Le traitement de ces

éléments, au niveau du neurone, utilise une fonction de transfert modifiée comprise dans la méthode *computeOutput*. La fonction sert à introduire une non-linéarité dans le fonctionnement du neurone. Cette fonction est de type sigmoïde (figure 6.37), ce qui la rend compatible avec les conditions que doit remplir une fonction de sortie, à savoir qu'elle utilise des nombres réels dans [0,1].

$$f_{sigmoïde}(x) = \frac{1.0}{\left(1.0 + e^{-sum}\right)}$$

Figure 6.37. Fonction de transfert de type sigmoïde

La méthode *computeOutput* consiste à calculer un poids d'apprentissage à partir des différentes valeurs de poids présentes dans le vecteur d'entrée. Le calcul est réalisé en fonction d'un jeu de données (vecteurs de renseignement) présentées en entrée du réseau et d'un jeu de données d'apprentissage. Le but de cette méthode est de déterminer une valeur de précision pour chaque vecteur de renseignement entrant.

6.2.3.4. Le package *info*

Le package *info* comprend les données nécessaires à l'utilisation du package principal au niveau de la création des vecteurs de renseignement. Ces vecteurs de renseignement sont importants pour le fonctionnement de l'apprentissage. Le package est constitué de trois classes qui génèrent des vecteurs de renseignement spécifiques à chacun (figure 6.38).

La classe *ContextInformationGenerator* sert à la réalisation d'un vecteur d'apprentissage renfermant les mêmes éléments de base qu'un vecteur de renseignement. En fait, le vecteur d'apprentissage est un vecteur de renseignement sur lequel le système se base pour l'apprentissage. C'est le point de référence pour l'apprentissage.

La classe *ELINTInformationGenerator* est un générateur de vecteurs de renseignement de nature électronique. Les données d'information pour créer les vecteurs de renseignement proviennent des résultats calculés par le module *Optipath*. Le traitement de ces résultats sert à la création des vecteurs de renseignement.

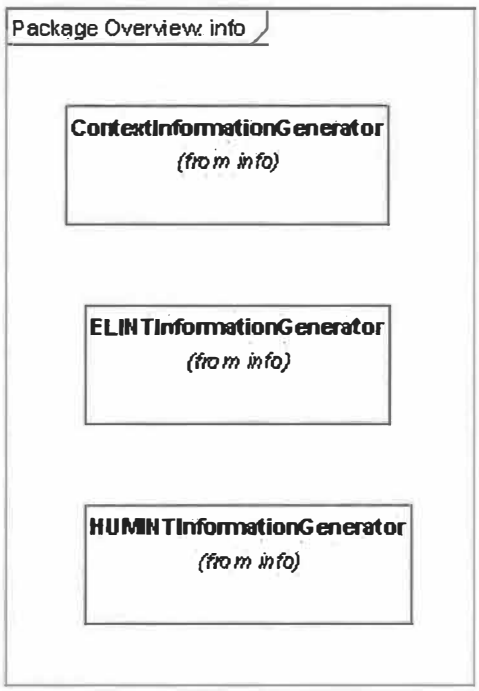


Figure 6.38. Diagramme UML du package *info*

La classe *HUMINTInformationGenerator* sert à la génération de vecteur de renseignement de type HUMINT. Pour générer ce genre de vecteur, il faut des données en provenance de l'utilisateur. Ces données sont passées en paramètre au niveau du constructeur et servent à la réalisation d'un vecteur de renseignement.

6.2.3.5. Le package *code*

Ce package dispose des données nécessaires à l'utilisation de la classe *ContexteLearning*. En effet, des codes au niveau du type de routes et du comportement de la recherche de chemin sont essentiels dans la définition du contexte d'apprentissage. La figure 6.39 montre un diagramme UML de ce package.

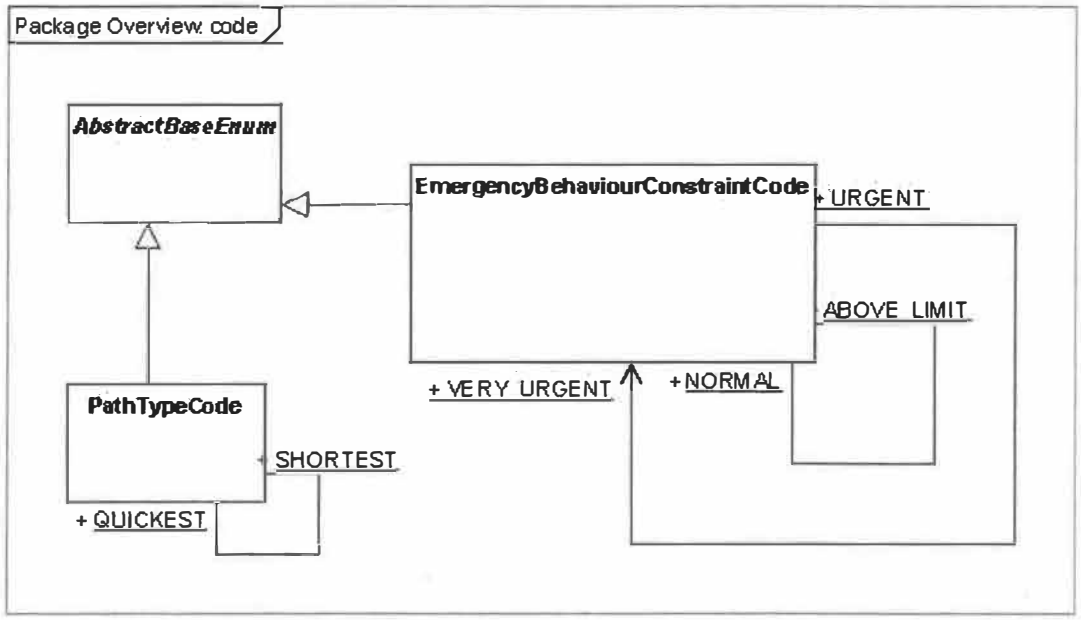


Figure 6.39. Diagramme UML du package *code*

La classe *AbstractBaseEnum* est une classe abstraite servant à l'énumération d'un type de base. L'objet comporte une valeur ainsi qu'une description relative à l'objet. Cette classe sert à la création de classes spécialisées comme la sous-classe *PathTypeCode*. Cette sous-classe sert à la codification du type de parcours selon la demande de recherche de chemins. La recherche du chemin le plus court peut s'effectuer de deux manières : soit en fonction de la distance ou bien en fonction du temps. L'autre sous-classe que l'on retrouve dans le package est la classe *EmergencyBehaviourConstraintCode*. L'utilité de cette classe réside dans la qualification du comportement que l'utilisateur veut adopter vis-à-vis la recherche optimum de parcours. Les comportements que l'utilisateur peut prendre varient de normal à très urgent.

6.3. Conclusion

Les modèles expérimentaux développés dans ce chapitre vont permettre de vérifier la performance des systèmes d'apprentissage à l'étude. Cette performance va se mesurer par la capacité de l'AG de maîtriser son propre paramétrage. Idéalement, la maîtrise du fonctionnement

de l'AG par l'AG lui-même devrait accroître ses performances pour trouver une bonne solution. La mesure de la performance est aussi associée à l'apprentissage de nouveaux vecteurs d'information à partir d'un ensemble hétérogène de vecteurs et ce, sans exemples précis d'apprentissage. Dans ce cas, pour une mesure plus juste des performances de l'apprentissage, une expérimentation à l'aide d'un système externe comme le module Optipath est nécessaire.

Chapitre 7
Expérimentations

7.1. Résultats d'expérimentation APAG

L'expérimentation du prototype APAG à l'aide du problème du commis voyageur permet de confirmer que l'auto-adaptation des paramètres permet une meilleure performance des algorithmes génétiques. Cette augmentation de performance est en partie attribuable au choix entre les divers opérateurs génétiques tout au long de l'exécution de l'algorithme. La comparaison du prototype APAG avec un modèle d'algorithme génétique traditionnel suit le modèle expérimental décrit dans le chapitre 6. Les paramètres utilisés pour l'AG simple sont résumés dans le tableau 7.1 et restent fixes pendant l'expérimentation.

Opérateur de croisement	Deux points de croisement
Opérateur de mutation	Une mutation à une position aléatoire
Probabilité de croisement	0.75
Probabilité de mutation	0.01

Tableau 7.1. Jeu de paramètres pour l'AG simple pour l'expérimentation APAG

7.1.1. Comparaison APAG avec AG traditionnel

Le tableau 7.2 montre des résultats d'une comparaison de l'AG simple versus l'AG avec l'auto-adaptation des paramètres pour 10 exécutions du même problème. La distance correspond à la solution trouvée pour le problème. Pour cette distance correspondent un nombre de générations et un temps d'exécution en millisecondes.

AG simple										
Distance	27	30	27	27	29	**25**	29	27	29	27
Générations	215	210	236	362	213	185	215	295	201	341
Temps exéc. (millisecondes)	500	500	501	801	501	500	500	701	401	801
AG avec auto-adaptation des paramètres										
Distance	**25**	**25**	**25**	**25**	**25**	**25**	27	**25**	**25**	**25**
Générations	68	15	64	78	91	21	244	44	123	91
Temps exéc. (millisecondes)	4576	4987	55149	1525	8903	1202	4316	5197	3515	4106

Tableau 7.2. Résultats d'exécution pour un AG simple et un AG avec auto-adaptation des paramètres

7.1.1.1. Solution optimale

Les résultats indiquent clairement que l'AG avec un processus d'auto-adaptation des paramètres permet de trouver une solution optimale, dans la majorité des cas d'exécution. La solution optimale étant égale à 25 unités pour la distance, l'algorithme APAG trouve cette solution 9 fois sur 10 dans l'expérimentation présentée. Dans le cas de l'AG simple, la solution optimale est atteinte dans seulement 10% des cas.

7.1.1.2. Nombre de générations

Le tableau 7.2 montre que le nombre de générations est significativement diminué dans le cas adaptatif. En effet, une moyenne de 84 générations est observée comparativement à une moyenne de 247 générations dans le cas d'un AG traditionnel. Ceci confirme une forte convergence de l'algorithme APAG vers la meilleure solution et ce, sans tomber dans un optimum local.

Cette convergence plus forte vers la meilleure solution s'observe graphiquement par la moyenne des solutions trouvées (distance) en fonction du nombre de générations. Ainsi, le graphique de l'évolution de la distance versus les générations est représenté à la figure 7.1. Ce graphique met en évidence la performance des AG avec un système d'adaptation des paramètres.

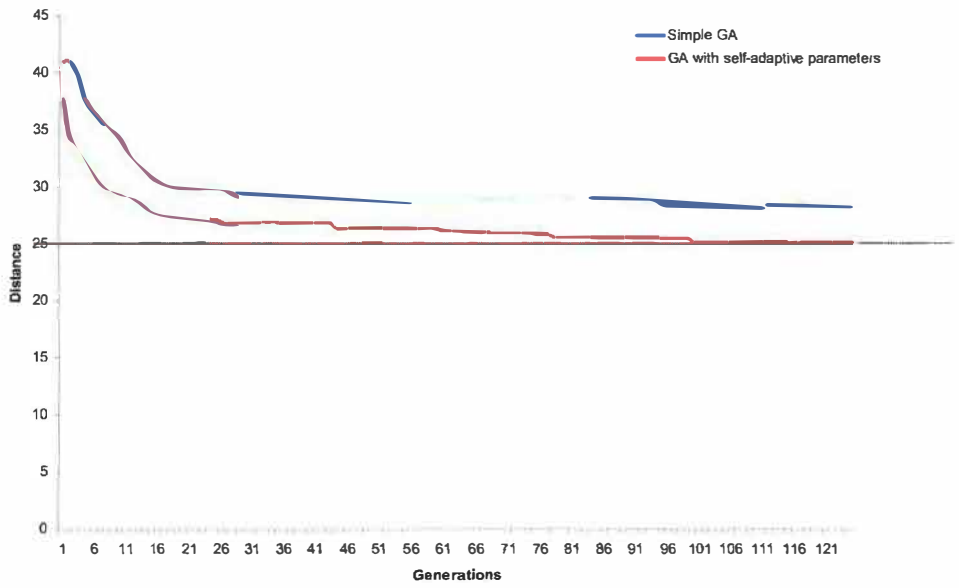


Figure 7.1. Graphique de la performance de la convergence des AG

7.1.1.3. Temps d'exécution

L'observation des temps d'exécution montre une perte significative en temps de calculs pour l'algorithme adaptatif. La moyenne en temps d'exécution pour un déroulement normal avec l'AG adaptatif est de l'ordre de 5 secondes. La moyenne pour un AG simple dans le cas présent tourne autour d'une demi-seconde pour l'exécution. Le prototype APAG est composé de deux algorithmes imbriqués, ce qui peut engendrer des temps d'exécution plus longs. De plus, la rapidité en temps de calcul n'était pas un des critères importants lors de la conception du prototype, le but premier étant de valider le concept d'auto-adaptation des paramètres.

Des résultats semblables au niveau de la performance de la convergence sont rapportés dans les travaux de [Mernik et *al.*, 2000]. Ses travaux ont montré que la combinaison des opérateurs génétiques était supérieure à l'utilisation d'un seul opérateur. Ils proposent que les différents opérateurs de croisement préservent les différentes propriétés qui sont utiles pour l'exploration de nouvelles solutions dans l'espace de recherche du problème.

Les exécutions produites dans le tableau 7.2 au niveau de l'AG avec auto-adaptation des paramètres sont réalisées à partir de plusieurs jeux de paramètres. Pour plusieurs répétitions d'un même problème, le tableau 7.3 montre qu'il existe différents jeux de paramètres possibles pour

résoudre un problème. Ceci démontre qu'il n'y a pas seulement un seul jeu de paramètres pour résoudre un problème donné, au moins pour le cas du commis voyageur.

	Opérateur de croisement adaptatif	Opérateur de mutation adaptatif
Cas 1	Un point de croisement Taux = 0.99	Mutation uniforme, seulement si amélioration possible Taux = 0.07
Cas 2	Croisement uniforme avec une probabilité de 0.5 Taux = 0.95	Une mutation à une position aléatoire Taux = 0.60
Cas 3	Deux points de croisement Croisement uniforme avec une probabilité de 0.5 Taux = [0.60, 0.80]	Mutation uniforme Taux = [0.40, 0.50]

Tableau 7.3. Résultats de trois exécutions de l'AG avec auto-adaptation des paramètres pour un même problème

Dans les trois cas du tableau 7.3, la distribution des paramètres permet de déterminer des jeux différents pour les trois répétitions. En général, pour chaque cas, un seul opérateur génétique avec son taux associé sont trouvés à la fin de l'exécution. Cependant, dans certains cas (cas 3 dans le tableau 4), une certaine fluctuation dans l'évolution des jeux de paramètres ne permet pas de déterminer clairement un opérateur ou un taux en particulier. D'autres résultats sur l'évolution des opérateurs génétiques pendant l'exécution sont présentés dans les prochaines sous-sections.

7.1.2. Opérateurs de croisement adaptatif

La figure 7.2 montre l'évolution des opérateurs de croisement en relation avec les différentes générations successives. Chaque point sur le graphique illustre un opérateur de croisement présent dans la population de paramètres à une génération donnée. Ainsi, pour les 5 premières générations, les 5 opérateurs de croisement sont présents au sein de la population de jeux de paramètres. À partir de la 6^e génération, deux opérateurs de croisement sont non sélectionnés et ne font plus partie de la population de paramètres. Le choix de seulement un opérateur par l'AG apparaît à la 13^e génération. Cette convergence vers un seul opérateur suggère que l'AG, dans son processus d'apprentissage en lien avec le contexte du problème, raisonne sur les opérateurs qui ont un apport positif sur la résolution du problème. À tout moment pendant l'exécution de l'algorithme, le choix d'un autre opérateur est possible et l'évaluation de l'opérateur a lieu à chaque génération.

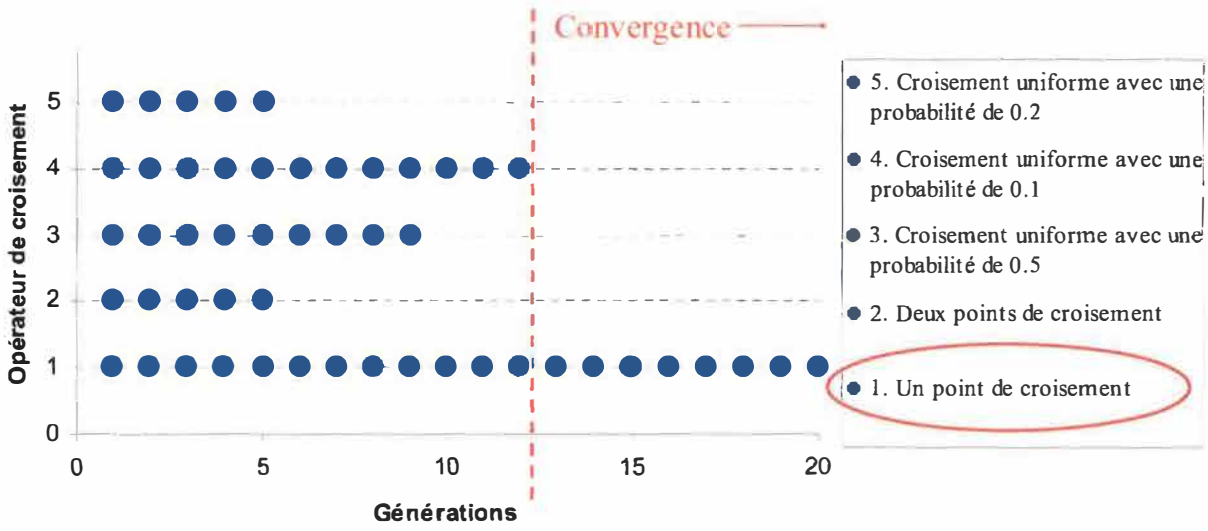


Figure 7.2. Représentation de l'évolution des opérateurs de croisement

7.1.3. Opérateur de mutation adaptative

La figure 7.3 permet d'illustrer l'évolution de différents opérateurs de mutation à chaque génération. Au début de l'exécution, on retrouve la présence des 5 opérateurs de mutation au niveau de la population de paramètres. Pendant l'évolution, certains opérateurs sont sélectionnés et d'autres disparaissent. Le résultat de la convergence vers un seul opérateur se produit à la 12^{e} génération. Cette convergence dans l'évolution des opérateurs de mutation correspond au choix d'un opérateur de mutation qui a un impact positif sur la résolution du problème. Les résultats d'évolution sont similaires à ceux des opérateurs de croisement, ce qui vient appuyer le fait que les AG acquièrent la capacité de contrôler leur propre fonctionnement.

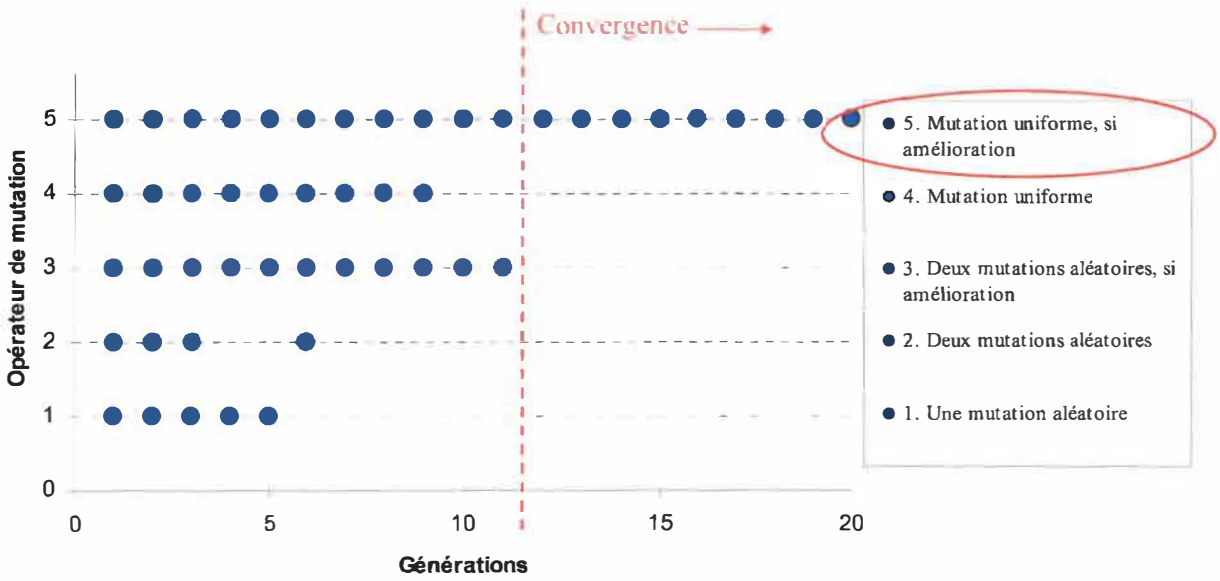


Figure 7.3. Représentation de l'évolution des opérateurs de mutation

7.1.4. Probabilité de croisement et de mutation

L'évolution des deux probabilités des opérateurs génétiques est illustrée à la figure 7.4. Chaque point de ce graphique a été obtenu en calculant les moyennes des probabilités à chaque génération. Les courbes montrent une tendance vers une stabilisation des paramètres qui apparaît après seulement 10 générations. Le comportement des courbes indique que l'évolution du taux de croisement est croissante et conduit à un taux élevé de probabilité. Ce résultat est validé par la comparaison avec les valeurs théoriques qui suivent un intervalle entre [0.60, 0.95]. La courbe correspondant à l'évolution du taux de mutation suit une tendance donnant des valeurs faibles de probabilité de mutation. Ce résultat concorde aussi avec les valeurs théoriques de probabilité de mutation qui se situent généralement entre [0.001, 0.1].

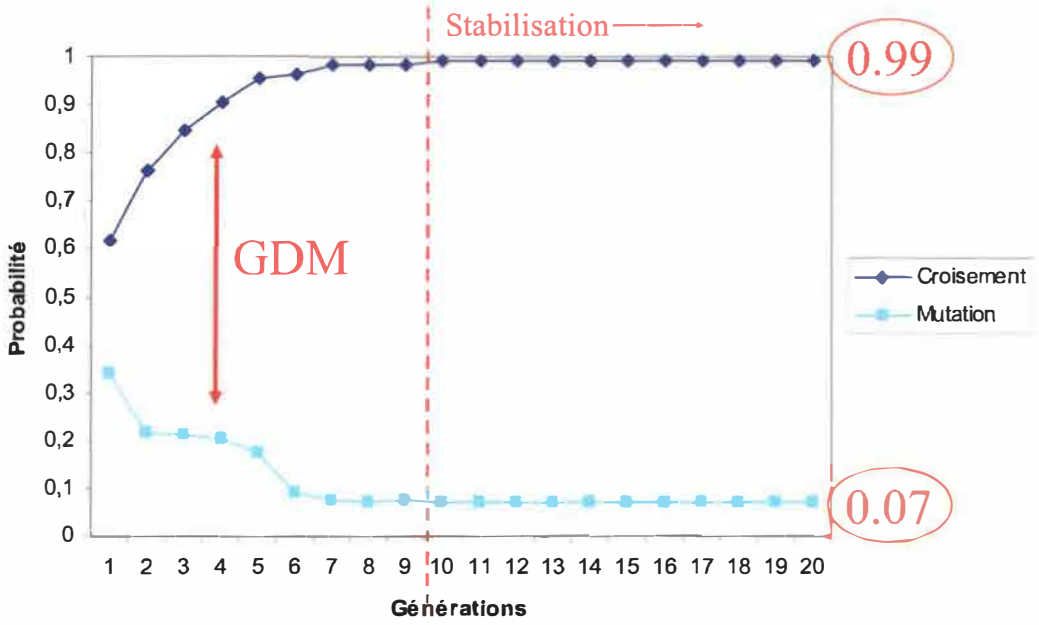


Figure 7.4. Graphique de l'évolution des opérateurs de probabilités en fonction du nombre de générations

Le comportement des courbes illustrées tend à une interdépendance entre les deux opérateurs. Ce comportement est expliqué par le fait que ces opérateurs sont définis pour maintenir la diversité génétique (GDM) dans la population. Quand les valeurs d'un opérateur de probabilité changent, les valeurs de l'autre opérateur doivent également changer pour obtenir un bon rapport de gdm. Ainsi, l'apprentissage du taux de croisement et de mutation dépend de la diversité génétique.

7.1.5. Conclusion

L'expérimentation du prototype APAG à l'aide du problème du commis voyageur permet de bien démontrer le caractère adaptatif des AG. En effet, le mécanisme interne des AG s'adapte et évolue pendant la résolution du problème. Si l'objectif de résolution d'un problème change dans le temps, les AG qui possèdent un mécanisme d'auto-adaptation trouveront le meilleur jeu de paramètres pour cette situation. Cette capacité d'adaptation est démontrée dans cette expérimentation et nos résultats indiquent qu'il n'y a aucun jeu de paramètres spécifique pour une résolution donnée. Le choix d'un jeu de paramètres est le résultat d'une combinaison des

paramètres avec d'autres paramètres et ce, en étroite interaction avec le problème et le mécanisme d'évolution des AG. Le résultat du caractère adaptatif des AG du prototype APAG est prometteur pour le développement d'un système de méta-apprentissage basé sur les AG. Dans un contexte militaire urbain, le système d'apprentissage a besoin d'une certaine souplesse d'adaptation vis-à-vis l'état dynamique du milieu auquel le système est confronté.

7.2. Résultats d'expérimentation MAAG

L'expérimentation à l'aide du prototype MAAG permet de montrer la capacité d'apprentissage des algorithmes génétiques dans un processus de méta-apprentissage menant à la production de méta-connaissances. En effet, les résultats d'expérimentation préliminaire démontrent une tendance de l'algorithme à déterminer une solution d'apprentissage très similaire à celle trouvée par le module Optipath.

Les tableaux 7.4, 7.5 et 7.6 montrent les résultats d'une expérimentation sur la répétition de l'apprentissage (10 essais) par le prototype MAAG pour un même cas d'utilisation. Chaque tableau compile les parcours trouvés par l'application MAAG et le parcours idéal déterminé par le module Optipath. La représentation d'un parcours se fait par la description de celui-ci en termes de type de parcours (quickest, shortest), de nombre d'arcs, de distance et de temps de parcours.

Le tableau 7.4 montre les résultats pour l'utilisation du premier cas qui correspond à une complexité de moins de 10 arcs par parcours. Le tableau 7.5 compile les résultats d'apprentissage pour un cas d'utilisation avec plus d'une dizaine d'arcs et le tableau 7.6 correspond au cas d'utilisation avec une complexité de plus ou moins 50 arcs.

Les résultats des répétions d'apprentissage à partir de MAAG ne devraientt pas être interprétés en terme d'une comparaison parfaite avec les parcours déterminés par Optipath. Le but du système d'apprentissage n'est pas d'obtenir la solution parfaite mais d'avoir une solution très près de la réalité et d'évoluer selon les conditions présentes. En effet, à des fins de simplification pour l'expérimentation, le système raisonne seulement sur certaines caractéristiques du parcours planifié. L'optimisation de l'apprentissage se concentre donc sur les critères de distance et de temps.

Cas1		Path type	Edges	Distance (Km)	Time
Optipath		**quickest**	**3**	**0,34**	**41s**
MAAG	Essai 1	shortest	3	0,34	**41s**
	Essai 2	shortest	3	0,34	**41s**
	Essai 3	quickest	4	0,47	56s
	Essai 4	shortest	4	0.47	56s
	Essai 5	quickest	3	0.34	**41s**
	Essai 6	quickest	4	0,47	56s
	Essai 7	shortest	3	0,34	**41s**
	Essai 8	shortest	3	0,34	**41s**
	Essai 9	quickest	3	0,34	**41s**
	Essai 10	quickest	3	0,34	**41s**

Tableau 7.4. Résultats des parcours obtenus par apprentissage et par le module Optipath pour une complexité de moins de 10 arcs par parcours

Cas2		Path type	Edges	Distance (Km)	Time
Optipath		**quickest**	**13**	**1,59**	**2m 41s**
MAAG	Essai 1	shortest	14	1,55	**2m 49s**
	Essai 2	shortest	18	1,92	3m 28s
	Essai 3	shortest	14	1,55	**2m 49s**
	Essai 4	quickest	13	1,59	**2m 41s**
	Essai 5	shortest	14	1,55	**2m 49s**
	Essai 6	quickest	13	1,59	**2m 41s**
	Essai 7	shortest	18	1,92	3m 28s
	Essai 8	quickest	13	1,59	**2m 41s**
	Essai 9	quickest	13	1,59	**2m 41s**
	Essai 10	quickest	13	1,59	**2m 41s**

Tableau 7.5. Résultats des parcours obtenus par apprentissage et par le module Optipath pour une complexité de plus de 10 arcs par parcours

Cas3		Path type	Edges	Distance (Km)	Time
Optipath		**quickest**	**50**	**8,85**	**8m 41s**
MAAG	Essai 1	shortest	50	8,85	8m 41s
	Essai 2	shortest	51	8,89	8m 44s
	Essai 3	shortest	51	8,89	8m 44s
	Essai 4	shortest	50	8,96	8m 49s
	Essai 5	**quickest**	**50**	**8,85**	**8m 41s**
	Essai 6	quickest	50	8,96	8m 49s
	Essai 7	shortest	50	8,85	8m 41s
	Essai 8	shortest	50	8,85	8m 41s
	Essai 9	shortest	51	8,89	8m 44s
	Essai 10	shortest	50	8,85	8m 41s

Tableau 7.6. Résultats des parcours obtenus par apprentissage et par le module Optipath pour une complexité de plus ou moins 50 arcs par parcours

Entre les cas d'utilisation, l'accroissement de la complexité en terme d'arc n'est pas un facteur limitatif pour le système d'apprentissage. Dans les trois cas d'utilisation, près de 8 parcours appris sur 10 ont des temps très près du temps calculé par Optiptah. Les écarts entre les valeurs de temps très près l'une de l'autre sont attribuables à l'évaluation de la pondération lors de l'apprentissage. Cette pondération attribue un poids à l'information pertinente selon le contexte d'apprentissage. À défaut d'avoir des valeurs de validation, la pondération se veut moins précise pour éviter de ne pas avoir de solution. Cette précision est dépendante des valeurs d'adaptation (fitness) et est illustrée par le nuage de points de la figure 7.5. Plus la valeur de fitness est grande, plus l'écart entre le parcours de référence (Optipath) et le parcours appris s'accentue. Il est important de préciser qu'on cherche ici la plus petite des valeurs d'adaptation possible pour satisfaire une bonne solution.

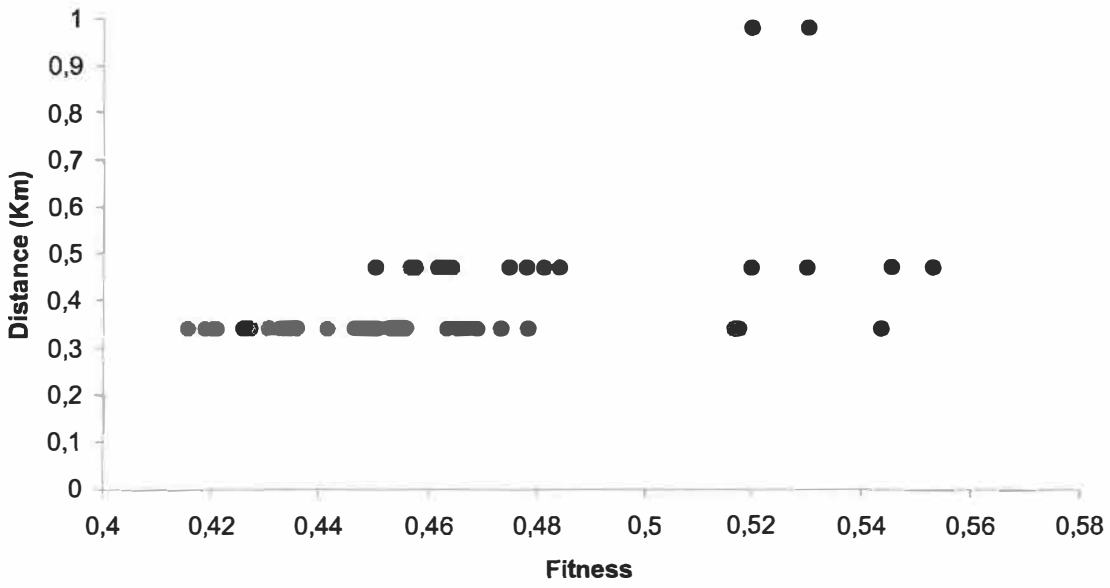
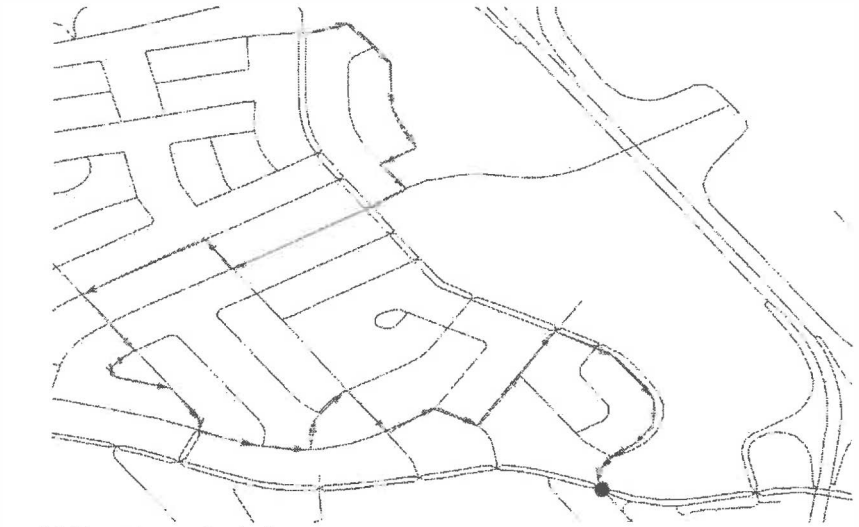


Figure 7.5. Nuage de points représentant la distribution du facteur distance en fonction de la valeur d'adaptation (fitness)

La figure 7.5 représente la relation des facteurs importants de l'apprentissage (soit la distance ou le temps) par rapport aux valeurs d'adaptation de l'algorithme génétique. Le nuage de points est obtenu par l'analyse de 50 parcours appris pour le premier cas d'utilisation. Plus la valeur d'adaptation est petite, plus la valeur de la distance du parcours appris correspond à la valeur du parcours de référence. Cette valeur d'adaptation est critique pour la détermination du meilleur vecteur de renseignement.

7.2.1. Expérimentations avec renseignement HUMINT

L'expérimentation des cas d'utilisation précédents correspond uniquement au traitement de renseignements de type ELINT. Le renseignement de type HUMINT étant complexe, l'expérimentation se focalise uniquement sur les éléments caractérisant un parcours. Ainsi, l'apprentissage avec le renseignement humain est réalisé par l'ajout d'un parcours le plus long possible à l'aide de l'interface HUMINT. Ce parcours correspond à un parcours valide et le choix des segments de route est illustré à la figure 7.6.



```
pathType :  Quickest
pathNbEdges :  26
pathDistance :  3.13 Km
pathTime :  5m 56s
```

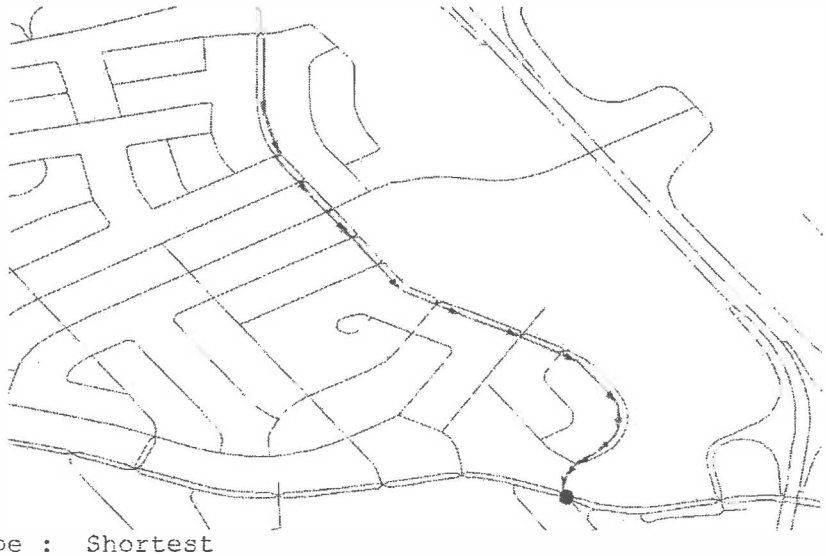
Figure 7.6. Illustration sur le réseau routier du parcours HUMINT le plus long possible

Les résultats de cette expérimentation sont résumés dans le tableau 7.7. Ceux-ci montrent que dans tous les essais d'apprentissage, le vecteur de renseignement HUMINT introduit n'est jamais sélectionné. Ceci suggère qu'un vecteur de renseignement comportant des informations ayant des écarts de valeurs importantes avec les valeurs visées, se voit attribuer un poids d'apprentissage qui ne favorise pas sa sélection via les algorithmes génétiques. L'attribution du poids d'apprentissage est toujours dépendante du contexte ou l'on se trouve. Dans le cas présent, le contexte d'apprentissage favorise un parcours rapide avec très peu de segments routiers.

Par conséquent, dans ce même contexte d'apprentissage, un vecteur de renseignement humain comportant un parcours plus court en distance devrait être sélectionné plus souvent comme solution possible. Le deuxième cas d'utilisation de renseignement humain comporte donc un choix de segments de routes pour un parcours plus court et il est illustré à la figure 7.7.

Cas1 HUMINT		Path type	Edges	Distance (Km)	Time
Optipath		**quickest**	**13**	**1,59**	**2m 41s**
MAAG	Essai 1	quickest	13	1,59	2m 41s
	Essai 2	shortest	14	1,55	2m 49s
	Essai 3	shortest	18	1,92	3m 28s
	Essai 4	quickest	13	1,59	2m 41s
	Essai 5	quickest	13	1,55	2m 41s
	Essai 6	quickest	13	1,59	2m 41s
	Essai 7	shortest	18	1,92	3m 28s
	Essai 8	shortest	14	1,55	2m 49s
	Essai 9	quickest	13	1,59	2m 41s
	Essai 10	quickest	17	1,96	3m 20s

Tableau 7.7. Résultats des parcours obtenus par apprentissage avec du renseignement HUMINT (parcours le plus long possible)



```
pathType :  Shortest
pathNbEdges :  10
pathDistance :  1.43 Km
pathTime :  2m 8s
```

Figure 7.7. Illustration sur le réseau routier du parcours HUMINT le plus court en distance

Dans la figure 7.7, le parcours choisi par l'utilisateur comporte un segment à sens unique et ce parcours est considéré valide par le système d'apprentissage (solution possible). Dans cette expérimentation préliminaire, pour des raisons de complexité du renseignement humain, le type de renseignement HUMINT est considéré comme fiable de la part du système d'apprentissage. Dans des travaux futurs, un apprentissage sur cette fiabilité serait intéressant à valider.

Le tableau 7.8 montre que le renseignement HUMAIN est sélectionné au moins 4 fois sur 10 dans une répétition d'apprentissage pour le même problème. Les résultats des autres essais montrent des solutions très près de la solution optimale (Optipath). Dans tous les essais d'apprentissage, les solutions trouvées par le système d'apprentissage ne montrent pas d'écart majeur avec les solutions résolues par le module Optipath. Ceci confirme la capacité d'apprentissage réalisée par les algorithmes génétiques au niveau des vecteurs de renseignement.

Cas2 HUMINT		Path type	Edges	Distance (Km)	Time
Optipath		**quickest**	**13**	**1,59**	**2m 41s**
MAAG	Essai 1	**shortest**	**10**	**1,43**	**2m 8s**
	Essai 2	**shortest**	**10**	**1,43**	**2m 8s**
	Essai 3	shortest	13	1,59	2m 41s
	Essai 4	quickest	13	1,59	2m 41s
	Essai 5	shortest	14	1,55	2m 49s
	Essai 6	quickest	13	1,59	2m 41s
	Essai 7	**shortest**	**10**	**1,43**	**2m 8s**
	Essai 8	quickest	13	1,59	2m 41s
	Essai 9	**shortest**	**10**	**1,43**	**2m 8s**
	Essai 10	quickest	13	1,59	2m 41s

Tableau 7.8. Résultats des parcours obtenus par apprentissage avec du renseignement HUMINT (parcours le plus court en distance)

7.2.2. Conclusion

L'expérimentation actuelle sur l'apprentissage de vecteurs de renseignement donne des résultats prometteurs et met en perspective la capacité d'apprentissage des algorithmes génétiques. À la lumière des résultats, la sélection de parcours par le système d'apprentissage se révèle efficace par rapport aux résultats déterminés par le module Optipath. Les solutions trouvées par apprentissage ne sont pas toutes optimales, ce qui pourrait être causé par un apprentissage trop spécifique sur certains critères. En effet, dans l'expérimentation MAAG, le système raisonne seulement sur certaines caractéristiques du parcours planifié et ne tient pas compte de l'ensemble du contenu des vecteurs de renseignement. Ceci suggère qu'un élargissement des critères pour l'apprentissage permettrait d'augmenter l'efficacité dans la sélection d'une solution optimale. Par ailleurs, l'introduction d'une étape d'apprentissage supplémentaire sur les valeurs d'adaptation permettrait d'augmenter la précision du système. Cette précision vise une pondération plus adéquate des vecteurs afin d'éloigner des vecteurs de renseignement moins importants. Concernant le renseignement HUMINT, il serait intéressant d'optimiser le système d'apprentissage pour le traitement de la fiabilité de la source. Cette optimisation permettrait de rejeter ou d'accepter dans des proportions différentes, les informations transmises par la source. Idéalement, le prototype développé devrait inclure plusieurs itérations d'apprentissage sur lui-même, afin de compléter l'élaboration du système de méta-apprentissage décrit dans les chapitres précédents. Ces itérations font référence à la notion *a posteriori* définie dans le chapitre 5 et qui ne fait pas l'objet d'une expérimentation à ce stade de développement du prototype.

Chapitre 8
Conclusion et travaux futurs

8.1. Conclusion

L'aspect de l'apprentissage qui traite de l'ajustement des paramètres de l'AG par l'algorithme lui-même a clairement été décrit et validé par expérimentation. Le prototype APAG repose sur un principe d'autonomie des individus des AG en interaction étroite avec le contexte du problème. Les résultats prouvent que les AG dotés des capacités adaptatives peuvent apprendre et évaluer la qualité des jeux de paramètres selon leur degré de contribution à la résolution du problème. Les résultats indiquent également que le choix des opérateurs génétiques pendant l'exécution de l'algorithme augmente la convergence de celui-ci comparativement à un AG traditionnel. L'un des points innovateurs de la méthode proposée dans ce mémoire est sa simplicité d'utilisation qui ne requiert aucune participation *a priori* de l'utilisateur pour fixer le paramétrage. Malgré des résultats prometteurs, l'expérimentation du prototype APAG se limite à la résolution du problème du commis voyageur. Il serait intéressant d'appliquer le même principe pour la résolution sur d'autres problèmes et de vérifier les performances adaptatives de cette méthode. Cette capacité adaptative permet de suggérer que le prototype APAG pourrait résoudre des problèmes qui ont une solution variant dans le temps. L'un des aspects non développé dans l'expérimentation est relié au paramètre adaptatif du taux de la population. Dans APAG le contrôle de la population suit le même fonctionnement que dans un AG traditionnel. Des travaux futurs portant sur cet aspect pourraient accroître les performances mais donneraient à l'AG un contrôle complet de tous les paramètres de fonctionnement.

Le second aspect de l'apprentissage abordé dans ce mémoire fait référence à l'utilisation des AG comme outils d'apprentissage dans un système de méta-apprentissage. Pour le développement de l'architecture du prototype MAAG, une combinaison de méthodes d'optimisation *a priori* et *a posteriori* sont employées dans un environnement modulatoire. La notion *a priori* est employée pour définir les méta-connaissances du système, alors que la notion *a posteriori* sert à affiner les MC par simulations. Il est à noter que la notion a posteriori n'a pas été implémentée dans le cadre de ce projet. Il serait intéressant de tester cette notion et de valider l'apport de plusieurs itérations d'apprentissage sur le système lui-même et les MC produites. Le cœur du système réside dans la définition des ces MC et passe par un système hybride d'algorithmes d'apprentissage comprenant les algorithmes génétiques et le Darwinisme Neuronal. Les résultats d'expérimentation permettent de valider la capacité des AG comme outils d'apprentissage. Les données d'entrée du système se composent de diverses sources de renseignements provenant de plusieurs capteurs. Ces capteurs peuvent être de nature électronique ou humaine. Le traitement

de ces renseignements se fait via un objectif d'apprentissage et vise principalement à optimiser un ensemble de poids d'apprentissage afin de valider le meilleur renseignement. Selon les résultats obtenus par expérimentation, cette optimalité est atteinte par comparaison avec des données souhaitables. En effet, les solutions d'apprentissage trouvées sont les vecteurs de renseignement qui sont les plus probables d'être sélectionnés comme meilleure solution. Le point important à noter est que l'apprentissage ne repose pas sur des cas d'exemples pour réaliser son processus. Ces résultats sont encourageants, surtout s'ils sont utilisés comme aide dans l'évaluation et le traitement de l'information. Des travaux futurs devront être réalisés pour le développement complet du prototype afin d'accroître le perfectionnement du système de méta-apprentissage.

Dans la recherche d'un apprentissage parfait, il faut tenir compte qu'il n'y a pas de vérité absolue, il faut toujours que le système se pose des questions pour qu'il s'adapte, pour qu'il évolue. L'apprentissage, c'est en grande partie se poser des questions. Ce n'est pas de connaître les réponses exactes. C'est le désir de voir ce qu'il y a derrière la prochaine colline qui nous permet d'aller plus loin. On ne doit jamais arrêter de se poser des questions, de vouloir comprendre même quand on sait que les chances de trouver les réponses sont minimes. On doit continuer à se poser des questions pour apprendre.

8.2. Améliorations fonctionnelles des algorithmes génétiques

Cette section se veut un complément d'information sur les améliorations possibles à apporter aux AG dans d'éventuels travaux futures. La section précédente fait mention de quelques améliorations pour perfectionner les prototypes développés. Les améliorations qui suivent sont applicables aux AG en général.

Dans la conception d'un système évolutif artificiel, on oublie souvent la dimension biologique, avec ses fondements les plus simples. Par exemple, l'évolution des espèces se fait sur une très longue période de temps. En simulant rapidement le processus d'évolution, on élimine certains facteurs ou processus critiques qui peuvent influencer l'adaptation des individus d'une espèce. Le problème que résout une espèce animale est la survie, c'est-à-dire, la survie entre les espèces pour la maîtrise de leur environnement (conservation de l'espèce). Cette résolution du problème de survie passe par une adaptation de l'espèce (évolution) avec une très forte dépendance au contexte, soit l'environnement dans lequel elle évolue. Par opposition, dans la plupart des cas, l'objectif des systèmes artificiels est de réaliser une mécanique évolutive artificielle dont on maîtrise totalement les paramètres. Or, le modèle naturel n'a pas vraiment de contrôle sur ces paramètres car l'environnement change constamment. Jusqu'où peut on aller dans la réalisation d'un système d'évolution artificielle? Suite à une rétrospective des analogies avec la nature que

l'algorithme génétique peut posséder, une proposition d'amélioration du fonctionnement de l'algorithme sera détaillée.

8.2.1. Analogie avec les concepts de l'évolution naturelle

Les AG sont un modèle de génétique artificielle qui s'inspire de près de l'évolution des êtres vivants et de la nature. La convergence entre les AG et le modèle naturel n'est pas parfaite. En effet, le but d'un AG est de trouver rapidement une bonne solution, ce qui n'est pas le cas dans la nature. Un des objectifs complémentaires de recherche est d'améliorer notre compréhension des processus naturels d'adaptation. Partant de ce fait, on peut émettre l'hypothèse de concevoir des systèmes artificiels possédant des propriétés similaires aux systèmes naturels. Cette hypothèse de travail est discutable et possède certaines similarités avec le débat opposant IA forte et IA faible.

- **IA Forte (approche cognitive)**

  Les AG doivent raisonner à la manière des systèmes naturels en utilisant les mêmes mécanismes de fonctionnement.

- **IA Faible (approche pragmatique)**

  Les AG doivent aboutir aux mêmes solutions que les systèmes naturels, peu importe la méthode employée.

8.2.1.1. Crossing-over

L'une des étapes clé des AG est le *crossing-over*. Par définition, dans la nature, le *crossing-over* est un échange de segments entre chromatides provenant de chromosomes appariés durant la méiose (figure 8.1), la méiose étant un mode de division nucléaire à la suite de laquelle les gamètes se forment. Il y a donc, chez un même individu, appariement des chromosomes, possibilité de *crossing-over* et formation des gamètes. C'est à partir de gamètes provenant de deux individus qu'il y a recombinaison pour former un nouvel individu.

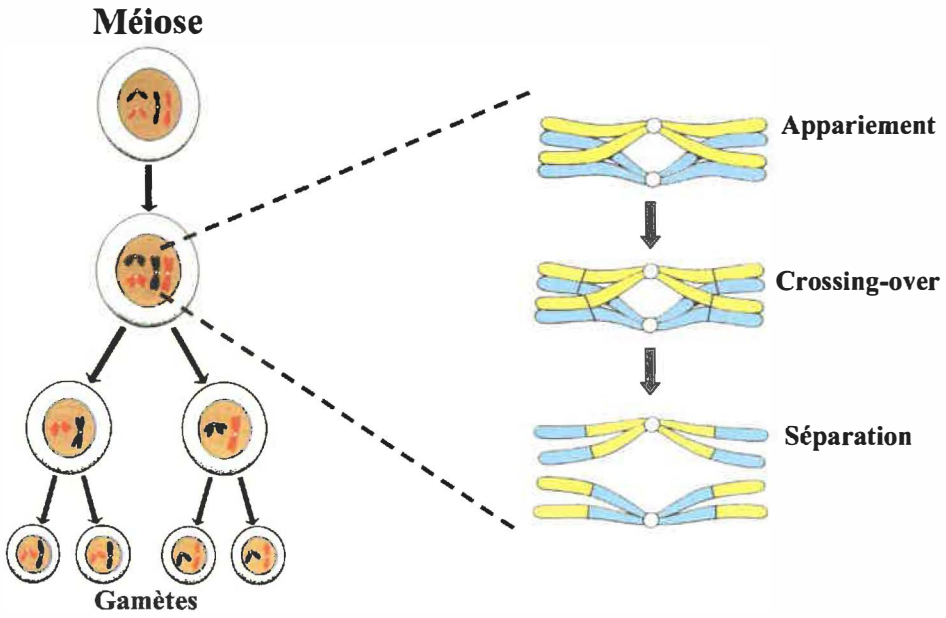


Figure 8.1. Phénomène de *crossing-over* durant la méiose chez un être vivant (adapté de Biologie, évolution, diversité et environnement [Mader, 1987])

En ce qui concerne les étapes des AG, il n'y a pas de formation de gamètes et de recombinaison. Le crossing-over s'effectue directement au niveau de deux individus différents pour former, automatiquement, deux nouveaux individus (figure 8.2). La notion de descendance est absente car la technique ne tient pas compte du nombre d'enfants possible et par conséquent on ne retrouve que des individus n'ayant aucun sexe. Cette notion de descendance reliée au sexe des individus est décrite plus en détail dans la prochaine section.

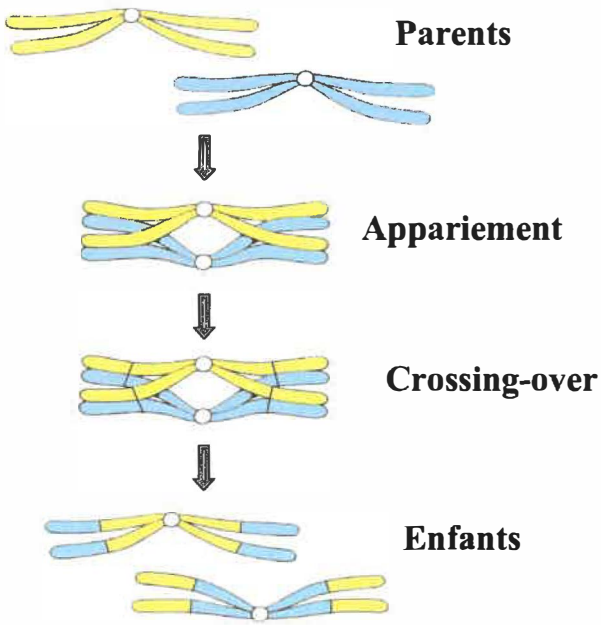


Figure 8.2. Phénomène de *crossing-over* dans les algorithmes génétiques

8.2.1.2. Sexe

Dans les travaux de [Werner et Dyer, 1991], le sexe des individus joue un rôle important et permet d'avoir deux sous populations spécialisées. La différence entre les sexes est précisément liée à la variance du nombre d'enfants (figure 8.3). En effet, un mâle peut biologiquement concevoir un nombre potentiellement important d'enfants, mais il y a une faible probabilité d'en avoir. En effet, si les chances du mâle de se reproduire son nulles, alors il ne peut avoir d'enfant. Une femelle n'aura qu'un nombre limité d'enfants au cours de sa vie, mais elle a en revanche une probabilité relativement élevée d'en avoir. Les espèces naturelles qui présentent des sexes spécialisés sont diploïdes, or les AG sont haploïdes, c'est-à-dire qu'ils ne possèdent qu'un seul chromosome. Le concept de la diploïdie sera discuté plus en profondeur dans la prochaine section.

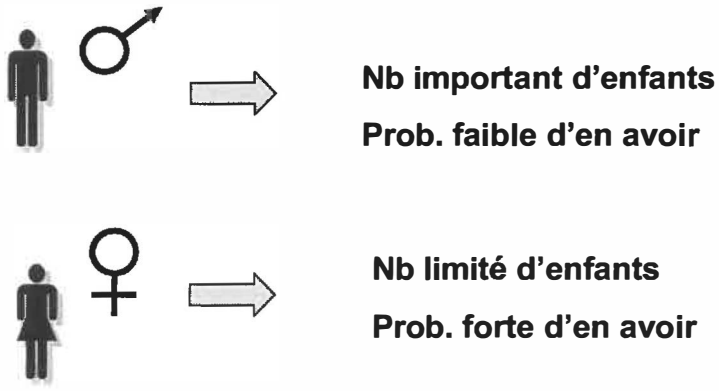


Figure 8.3. Variabilité du nombre d'enfants selon le sexe

8.2.1.3. Diploïdie

La diploïdie a fait l'objet de plusieurs travaux [Ryan et Collins, 1998], [Osmera et *al.*, 1997], [Ng et Wong, 1995], [Goldberg, 1987]. L'avantage que procure la diploïdie, est la présence en double des chromosomes, ce qui permet d'introduire la notion de gènes dominants et récessifs pour un même caractère. Un gène dominant est un gène qui empêche un gène récessif de se manifester.

Prenons l'exemple de la figure 8.4, un individu géant aux yeux bruns. Les gènes dominants donnent les caractères géants et yeux bruns, et les récessifs codent pour les caractères nain et yeux bleus. Ainsi le croisement de deux individus (géant aux yeux bruns) diploïdes qui ont tous deux des gênes récessifs nain et yeux bleus pourrait produire 2 géants aux yeux bruns, un géant aux yeux bleus et 1 nain aux yeux bruns. Si on applique le même principe de croisement avec un algorithme génétique, on n'obtient que deux individus géants aux yeux bruns.

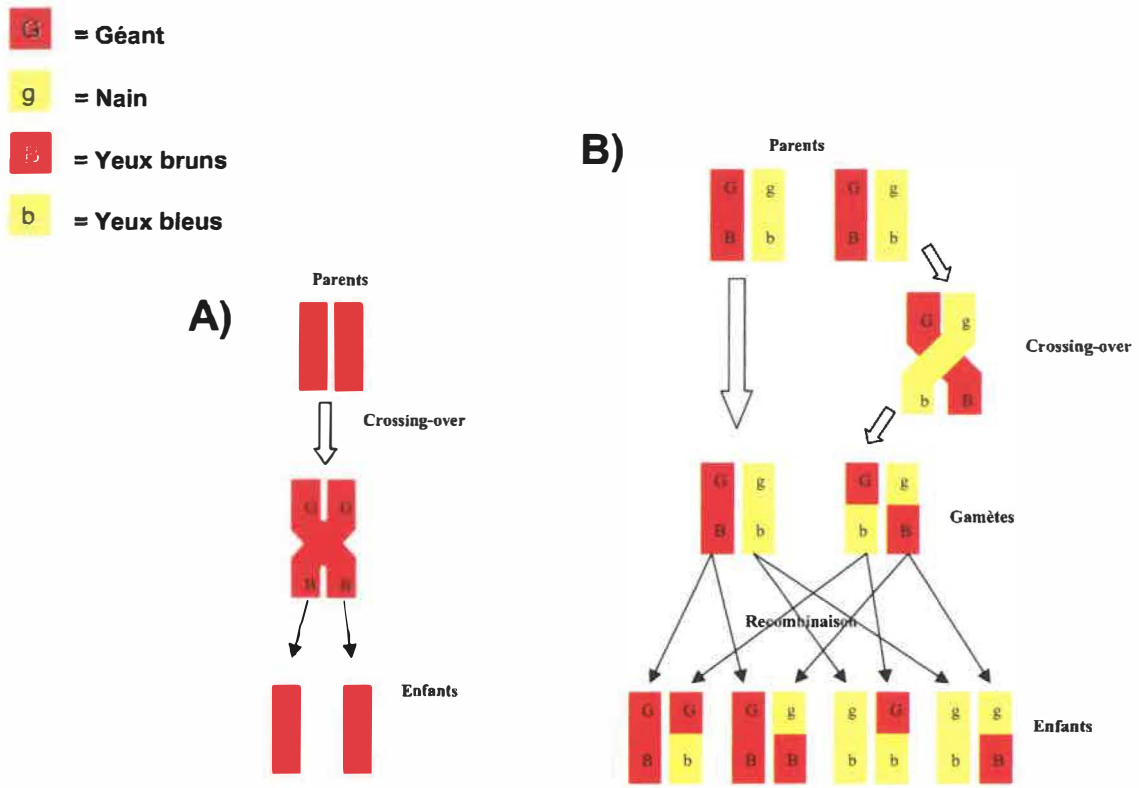


Figure 8.4. A) croisement avec un algorithme génétique simple, B) croisement de deux individus diploïdes (Tiré de Biologie, évolution, diversité et environnement [Mader, 1987])

On remarque que la notion de dominance accroît les possibilités de solutions et il a été montré que les individus diploïdes des AG présentent plus de variabilité dans leur fonction de d'adaptation que les individus d'une population haploïde. Ils peuvent également mieux tolérer les changements de l'environnement [Calabretta et *al*, 1997].

L'application de la diploïdie dans les AG implique la modification du programme pour la formation de gamètes et, par le fait même, conduit à l'attribution d'un sexe aux individus. Par conséquence, il y aura modification au niveau de l'étape du *crossing-over*, pour l'appliquer uniquement sur les chromosomes homologues d'un même individu avant la formation des gamètes.

La fonctionnalité de la diploïdie à l'intérieur des AG est dépendante de gènes dominants et récessifs. Il faut prévoir un mécanisme de gènes, fonctionnels ou non, dans le génome de l'individu. Ce mécanisme de contrôle de l'expression des gènes artificiels serait plus facile si, comme les gènes naturels, ils possédaient une certaine autonomie dans le génome.

8.2.1.4. Expression des gènes

La notion de gène chez une espèce vivante se définit comme une unité héréditaire occupant un espace précis situé sur un chromosome. Un chromosome comporte une série de segments qui ne codent pas pour des protéines et d'autres segments qui forment les gènes de structure. Chaque gène de structure est bordé par une région d'initialisation du début de la transcription et se finit par une région de terminaison de transcription. Au début de chaque gène de structure, il y a présence d'un gène régulateur chargé de contrôler si la transcription du gène de structure s'effectue ou non. À l'intérieur des gènes de structure, on retrouve des régions qui ne codent pas appelées introns (figure 8.5).

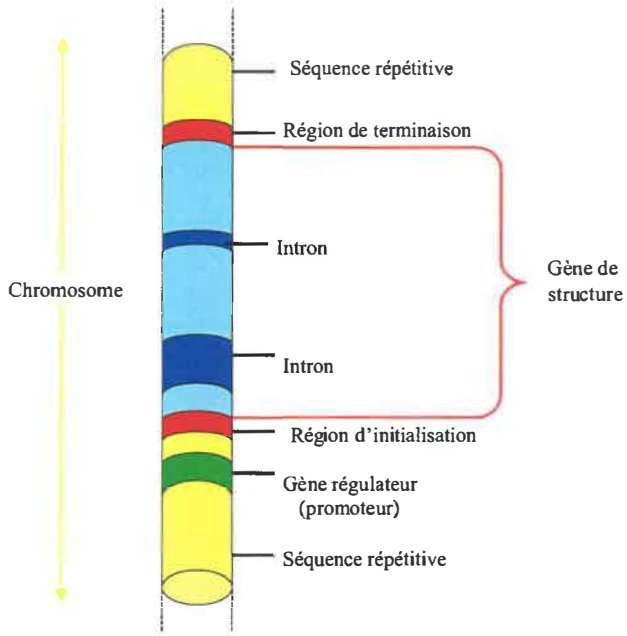


Figure 8.5. Schéma d'un gène de structure
(Tiré de Biologie, évolution, diversité et environnement [Mader, 1987])

Plusieurs travaux ont repris le fonctionnement du gène naturel et ont modifié les AG simples en les composant de gènes mobiles, de régions non codantes (introns) et en ajoutant des gènes régulateurs [Chen et Goldberg, 2002], [Chen, 2002], [O'Neill et Ryan, 2000], [Lobo et *al.*,1998]. Ainsi, plusieurs avantages potentiels peuvent être mis à profit pour les AG :

- une séparation de l'espace de recherche (génotype) et de l'espace de solutions (phénotype). Cette séparation donne plus de flexibilité à l'algorithme que de travailler uniquement avec le phénotype;
- une possibilité de maintenir la diversité génétique tout au long de l'exécution de l'algorithme;
- la conservation de la fonctionnalité, tout en permettant la suite de la recherche de la meilleure solution;
- une généralisation du codage qui peut représenter une variété de structures sans besoin d'opérateurs génétiques spécialisés;
- une compression de la représentation avec un génotype relativement petit;
- une implémentation alternative des fonctions, par exemple, par l'utilisation du modèle de gène de régulation;
- des positions indépendantes dans le génome.

En apportant la notion de gènes aux AG, les opérateurs de croisement et de mutation devront s'adapter aux nouvelles améliorations.

8.2.1.5. Mutation

Tout en apportant des modèles de la nature aux AG, comme la diploïdie et l'expression des gènes, il faut tenir compte des autres paramètres de l'algorithme. Dans l'algorithme génétique traditionnel, l'opérateur de mutation cible uniquement l'aspect gène. Il n'a pas d'effet sur les autres niveaux de l'algorithme. Ainsi, l'opérateur de mutation ne pourra plus agir seulement sur le gène, mais il devra aussi agir à d'autres niveaux comme par exemple, sur le chromosome. Dans le modèle naturel, il y a plusieurs types de mutation possibles (tableau 8.1), dont les mutations génétiques et les mutations chromosomiques.

mutation chromosomique	réorganisation de segments chromosomiques +/- responsable variation phénotypique types : délétion duplication inversion translocation
mutation génétique	modification du code d'un gène responsable d'une variation phénotypique types : délétion addition substitution
mutation germinale	modification d'un gène de gamètes variation aux descendants
mutation somatique	modification d'un gène de cellule variation à l'individu

Tableau 8.1. Tableau représentant les différents types possibles de mutation

8.2.2. Implémentation selon l'évolution naturelle

L'amélioration du fonctionnement de l'algorithme génétique d'un point de vue évolution naturelle passe par les différents points énumérés dans les sections précédentes. Par exemple, pour le premier cas énuméré à la section 8.2.1.1, l'ajout d'un système reproducteur aux individus de la population de l'algorithme génétique permet la production de gamètes. La figure 8.6 illustre l'approche du système reproducteur au niveau de l'individu. Ceci permet également de concevoir un algorithme qui tient compte des sexes des individus. La formation de deux systèmes de reproduction distincts est alors nécessaire. De plus, la conception du système reproducteur implique une duplication du chromosome, ce qui engendre le concept de diploïdie de la section 8.2.1.3. Comme déjà mentionné dans cette section, l'avantage que procure la diploïdie est de permettre l'introduction de la notion de gènes dominants et récessifs pour un même caractère. Une régulation des gènes est alors de mise pour le bon fonctionnement de ce système.

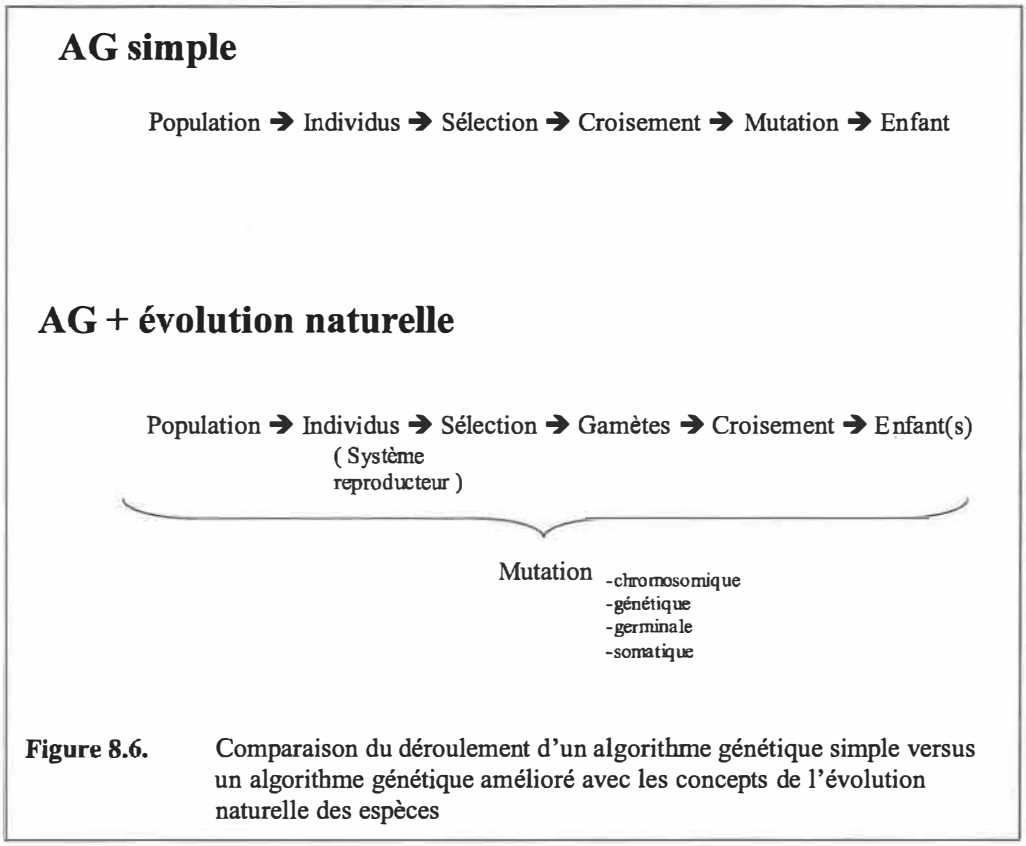


Figure 8.6. Comparaison du déroulement d'un algorithme génétique simple versus un algorithme génétique amélioré avec les concepts de l'évolution naturelle des espèces

La régulation de l'algorithme d'un point de vue diversité génétique est exercée par l'opérateur de mutation. Cet opérateur intervient à tous les niveaux pour un algorithme génétique suivant de plus près le modèle naturel. Un tel système de reproduction, avec ses diverses composantes, peut s'implémenter à partir d'un AG simple. La figure 8.7 illustre un diagramme UML des différentes classes d'un algorithme génétique typique. La figure 8.8 montre, par comparaison avec la figure 8.7, l'implantation des améliorations apportées a l'algorithme.

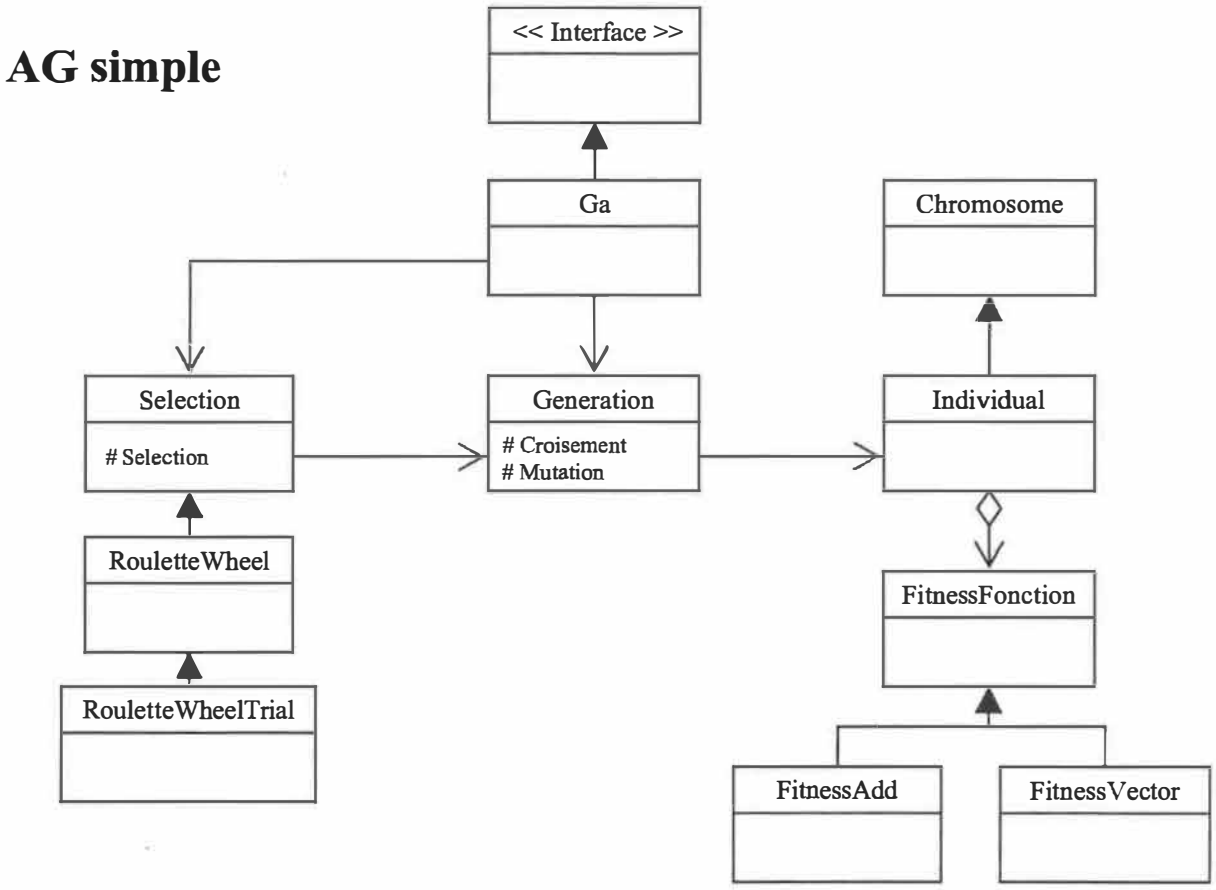


Figure 8.7. Diagramme UML générale d'un algorithme génétique simple

La classe *Ga* (figure 8.7) est responsable de créer et de faire succéder les générations. Pour ce faire, elle utilise l'objet de type *Selection* qui est responsable de la sélection des individus d'une population. La classe *Selection* est surclassée pour répondre à un type de sélection particulier qui est donné par les classes *RouletteWheel* et *RouletteWheeltrial*. Un objet de type *Selection* sélectionne des objets de type *Individual* se trouvant dans la classe *Generation*.

La classe *Generation* sert à la création des individus d'une population et comporte plusieurs méthodes permettant, entre autre, la mutation des gênes du chromosome et le croisement des individus. La super classe Chromosome contient un tableau des gènes qui définit un individu.

La sous-classe *Individual* représente un individu au sein d'une population. Cette classe étend la classe Chromosome par l'ajout de fonctionnalités touchant la fonction d'adaptation. Un objet de type *Individual* teste alors son bagage génétique grâce à un objet implantant l'interface *IFitnessFunction*. L'interface *IFitnessFunction* définit une méthode qui permet de générer des gènes et d'évaluer leur performance selon une fonction d'adaptation spécifique au problème traité. Les classes *FitnessAdd* et *FitnessVector* sont des implantations de la classe *IFitnessFunction*.

L'implémentation des améliorations ne change pas l'architecture générale du diagramme sauf pour l'ajout d'une seule classe représentant un objet de type *Gamètes* (figure 8.8). Les améliorations sont généralement réalisées par l'ajout de nouvelles méthodes au niveau des classes existantes.

La Classe *Selection* ainsi que ses sous classes reçoivent deux méthodes différentes pour la sélection au niveau des individus. Ces méthodes offrent des possibilités de sélection différentes, selon la sous population. Dans ces méthodes, le mode de sélection des individus est reliée à la notion de sexe, ce qui peut influencer le nombre de descendants.

La classe *Generation* acquiert une méthode appelée *recombinaison* qui correspond au nouveau processus de reproduction. Cette méthode sert à former un nouvel individu à partir de gamètes provenant de deux individus. La classe comporte aussi une fonction mutation (*mutationGen*) qui reprend la fonctionnalité de l'opérateur de mutation de l'algorithme génétique simple par des transformations au niveau d'un gène.

La classe *Individual* acquière une variable supplémentaire. La variable sexe au niveau de l'individu sert à préciser à laquelle des deux sous populations il appartient. L'ajout d'une méthode *mutationSom* correspond à la mutation somatique. Cette mutation affecte uniquement les gènes de l'individu et ne peut se transmettre aux descendants.

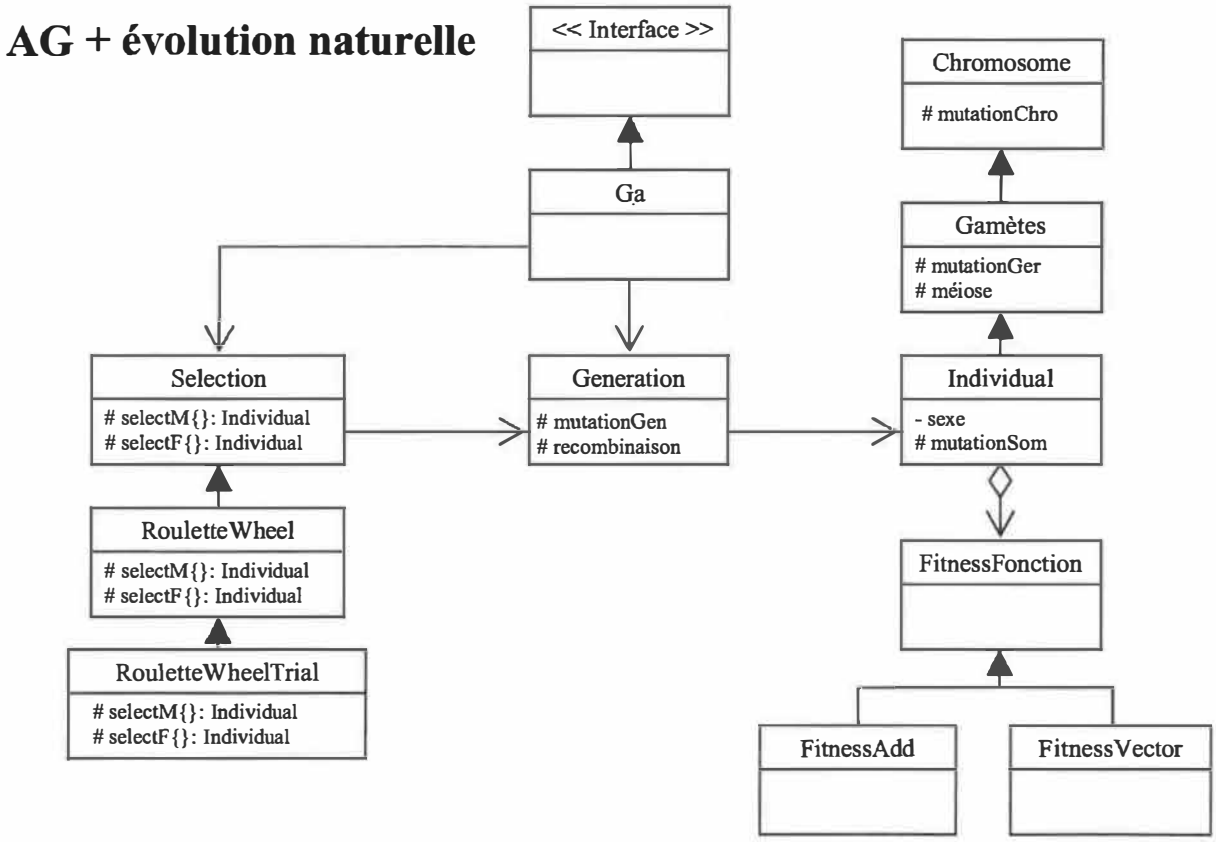


Figure 8.8. Diagramme UML générale d'un algorithme génétique amélioré avec les concepts de l'évolution naturelle

Dans la classe *gamètes*, la méthode mutation réfère à la mutation germinale c'est-à-dire que la méthode apporte des modifications aux gènes des gamètes. Ce type de mutation affecte directement les descendants, c'est-à-dire qu'elle se transmet de génération en génération. Par exemple, tout individu ayant une mutation au niveau des gamètes a un risque de 1 sur 2 de transmettre la mutation à ses enfants. Une nouvelle méthode pour la reproduction apparaît, c'est la méthode pour effectuer la méiose. Celle-ci, sert à former les gamètes. Chaque gamète contient la moitié du nombre de chromosomes qui caractérise les individus. Le principe d'opérateur de croisement s'applique encore dans l'algorithme et se retrouve à ce niveau.

La méthode mutation de la classe *Chromosome* correspond à la mutation chromosomique. Cette mutation affecte tout changement dans la structure ou dans le nombre des chromosomes. Elle affecte une portion du chromosome soit par délétion, duplication, inversion ou translocation.

8.2.3. Conclusion

La nature constitue une source d'inspiration efficace, car elle fonctionne à merveille en tant que système complexe capable de résoudre divers problèmes. L'hypothèse posée suggère que plus on sera proche du modèle naturel, meilleure sera la conception d'un système évolutif artificiel. Plusieurs travaux de recherche cités dans ce chapitre montrent des avantages d'utilisation de certains concepts naturels sur les AG. Le défi n'est pas de spécifier LA méthode pour un cas donné, c'est de spécifier une méthode plus générale pour permettre à l'algorithme de résoudre toute une classe de problème. Dans de futurs travaux de recherche, il serait intéressant de valider ce concept par une expérimentation rigoureuse.

Références

[Aamodt et Plaza, 1994], Aamodt, A., and Plaza, E., "Case-based reasoning: Foundational issues, methodological variations, and system approaches", *Artificial Intelligence Communications* 7(1):39-59, 1994.

[Ahn and Ramakrishna, 2002], Ahn C. W. and Ramakrishna R.S., "A Genetic Algorithm For Shortest Path Routing Problems And Sizing Of Populations", *IEEE Transactions on Evolutionary Computation*, 6(6): 566-579, 2002.

[Amat et Yahiaoui, 1996], Amat, J.L., Yahiaoui, G., "Techniques avancées pour le traitement de l'information", *Réseaux de neurones, logique floue, algorithmes génétiques*, Cépaduès-Éditions, 198 p., 1996.

[Bäck, 1992], Bäck, T., "Self-Adaptation In Genetic Algorithms", In Varela, F. J., & Bourgine, P. (Eds.), *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, 263-271, Cambridge, MA: The MIT Press, 1992.

[Bagley, 1967], Bagley, J. D., *The Behavior Of Adaptive Systems Which Employ Genetic And Correlation Algorithms*, Doctoral dissertation, University of Michigan. (University Microfilms No. 68-7556), 1967.

[Bengio, 1997], Benio, Y., "Algorithmes d'apprentissage", notes de cours ift6265, département d'informatique et de recherche opérationnelle, 1997.
http://www.iro.umontreal.ca/~bengioy/ift6264

[Bounsaythip, 1998], C. Bounsaythip, "Algorithmes Heuristiques et Evolutionnistes : Application à la Résolution du Problème de Placement de Formes Irrégulières", thèse de l'USTL, NO: 2336, soutenue le 9 October 1998.

[Cadon et al., 2000], Cadon A., Galinho T.,Vacher J.-P., "Genetic Algorithm using Multi-Objective in a Multi-Agent System", *Robotic and Autonomous Systems*, Elsevier, 33 (2-3): 179-190, 2000.

[Calabrettaet *al*, 1997] Calabretta Raffaele, Galbiati Riccardo, Nolfi Stefano and Parisi Domenico, "Investigating the role of diploidy in simulated populations of evolving individuals", *Electronic Proceedings of the 1997 European Conference on Artificial Life*, 1997.

[Canada National Defence, 2001], Canada National Defence, "Land Force Information Operations: Intelligence", Chief of the Defence Staff Publication B-GL-357-001/FP-001, 2001.

[Cette Semaine, 2001], Cette Semaine n°83, sept/oct, pp. 13-15., 2001.

[Chen et Goldberg, 2002], Chen Y, Goldberg D.E., "Introducing Start Expression Genes To The Linkage Learning Genetic Algorithm", University of Illinois at Urbana-Champaign, 2002.

[Chen, 2002] Chen Ying-Ping, Using start Expression genes for building-block separation in the linkage learning genetic algorithm, University of Illinois at Urbana-Champaign, 2002.

[Darwin, 1980], Darwin Charles, "L'origine des espèces", petite collection maspero, Paris, 1980.

[De Jong, 1975], De Jong, K.A., *An Analysis Of Behavior Of A Class Of Genetic Adaptative Systems*, PhD thesis, University of Michigan, Ann Arbor, 1975.

[De Jong1988], De Jong K., "Learning with Genetic Algorithms: An Overview", *Machine Learning*, 3(2): 121-138, 1988.

[Dewar, 1992] Colonel Michael Dewar, *War in the Streets. The Story of Urban Combat from Calais to Khafji*, London, BCA, p. 16, 1992.

[Dubot, 1997], Dubot, E., "Algorithmes génétiques", 1997.
http://wwwsi.supelec.fr/yb/projets/algogen/Algogen.htm

[Edelman, 1987], Edelman G. M., *Neural Darwinism: The Theory of Neuronal Group Selection.* New York: Basic Books, 1987.

[Edelman, 1989], Edelman, G. M., *The Remembered Present: A Biological Theory of Consciousness.* New York: Basic Books, 1989.

[Edelman, 1992], Edelman, G.M., *Bright Air, Brilliant Fire: On the Matter of the Mind.* New York: Basic Books,1992.

[Eiben et *al.*, 1999], Eiben, A.E., Hinterding, R., Michalewicz, Z., "Parameter Control In Evolutionary Algorithms", *IEEE Transactions on Evolutionary Computation*, 3(2):124-141, 1999.

[Fàbrega et Guiu, 1999], Fàbrega X.L., Guiu J.M.G.I., "GENIFER: A Nearest Neighbour based Classifier System using GA", *Proceeding of the Genetic and Evolutionary Computation Conference (GECCO99)*, 1999.

[Fisher, 1930], Fisher, R. A., *The Genetical Theory of Natural Selection* , Dover Publications, Inc., New York.,1930.

[Freisleben et Hartfelder, 1993], Friesleben, B. and Hartfelder, M., "Optimisation of genetic algorithms by genetic al-gorithms", Albrecht, R., Reeves, C., and Steele, N. (eds), *Artifical Neural Networks and Genetic Algorithms*, 392–399, 1993. Springer Verlag.

[Freisleben et Merz, 1996], Freisleben B., Merz, P., "A Genetic Local Search Algorithm For Solving Symmetric And Asymmetric Traveling Salesman Problems", *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, IEEE Press, 616-621, 1996.

[Gao, 2003], Gao Y., "Population Size and Sampling Complexity in Genetic Algorithms". *Proceedings of the Bird of a Feather Workshops(GECCO2003), Learning, Adaptation, and Approximation in Evolutionary Computation*, 178-181, 2003.

[Giraud-Carrier *et al.,* 2004], Giraud-Carrier C., Vilalta R., and Brazdil P., "Introduction to the Special Issue on Meta-Learning", *Machine Learning*, 54(3):187-193., 2004.

[Goldberg et Smith, 1987] Goldberg, D. E. and R. E. Smith, Nonstationary function optimization using genetic dominance and diploidy. *Proceedings of the Second International Conference on Genetic Algorithms,* (pp. 59-68), Lawrence Erlbaum Associates, 1987.

[Goldberg, 1987], Goldberg D.E., Richardson J., "Genetic algorithms with sharing for multi-modal fonction optimization". In J.J. Grefensfette, editor, *Proceedings of ICGA-87.* Pages41-49. Lawrence Erlbaum Associates.

[Goldberg, 1989], Goldberg, David Edward , *Genetic algorithms in search, optimization, and machine learning*, Addison-Wesley, 412 p., 1989.

[Goldberg, 1990] Goldberg, David Edward, "Real-coded Algorithms, virtuel Alphabets, and Blocking", University of Illinois at Urbana Champaign, 1990.

[Goldberg,1987] Goldberg D.E., Richardson J., Genetic algorithms with sharing for multi-modal fonction optimization. In J.J. Grefensfette, editor, *Proceedings of ICGA-87*, Lawrence Erlbaum Associates, pp. 41-49., 1987.

[Gomes et *al.*, 2003], Gómez, J., Dasgupta, D.,. González, Fabio A., "Using Adaptive Operators In Genetic Search", *GECCO 2003*, pp. 1580-1581, 2003.

[Gomez et Dasgupta, 2002], Gomez J. and Dasgupta D., "Using Competitive Operators And A Local Selection Scheme In Genetic Search," in Late-breaking papers *GECCO 2002*, 2002.

[Grefenstette, 1986], Grefenstette, J.J., "Optimization of control parameters for genetic algorithms", *IEEE Trans. Systems, Man, and Cybernetics, SMC-16*(1):122-128, 1986.

[Grenfenstte, 1993], Grefenstette, J. J., "Genetic Algorithms and Machine Learning", Invited talk, *Proc. Sixth Annual ACM Conf. Computational Learning Theory (COLT 93)*, ACM, 1993.

[Hao et *al.*, 1999], Hao, j.k., Galinier, p., Habib, m., "Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraites.", *Revue d'Itelligence Artificielle*, 13(2) :283-324., 1999.

[Hekanaho, 1996], Hekanaho Jukka, "Testing Different Sharing Methods in concept Learning", Turku Centre for Computer Science, Decembre 1996.

[Herrera et Lozano, 1996], Herrera F. and Lozano M., "Adaption of genetic algorithm parameters based on fuzzy logic controllers.", In Herrera F. and Verdegay J. (eds) *Genetic Algorithms and Soft Computing*, Physica Verlag., pp. 95-125., 1996.

[Hinterding 1997], Hinterding, R., "Self-adaptation using Multi-chromosomes", *Proceedings of the 4th IEEE International Conference on Evolutionary Computation*, 1997

[Holland, 1975] Holland J.H., *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence*, Cambridge, Mass. : MIT Press , 1992

[Holmes, 2002], Holmes J.H., Lanzi P.L., Stolzmann W., Wilson S.W., "Learning Classifier Systems: New Models, Successful Applications", *Inf. Process. Lett.*, 82(1): 23-30, 2002.

[Jennings, 2000], Jennings N.R.,"On Agent-Based Software Engineering", *Artificial Intelligence*, 117(2):277-296., 2000.

[Kirkpatrick et al., 1983], S. Kirkpatrick et al., "Optimization by Simulated Annealing ", *Science* 220:671-680, 1983.

[Knight et Sen, 1995], Knight L., Sen S., "PLEASE: A Prototype Learning System Using Genetic Algorithms", *ICGA*, pp. 429-435, 1995.

[Kosorukoff et Goldberg, 2001], Kosorukoff, A., Goldberg, D.E., "Genetic Algorithms for Social Innovation and Creativity", *IlliGAL Report* No. 2001005, 2001.

[Legault, 2000], Legault Rock, "Le champ de bataille urbain et l'armée : changements et doctrines", *Revue militaire canadienne*, automne 2000, p. 39.

[Lieutenant-colonel Timothy, 2000], Lieutenant-colonel Timothy L. Thomas, US Army, "Grozny 2000: Urban Combat Lessons Learned", paru dans *Military Review*, juillet-août 2000.

[Lis, 1996], Lis, J., "Parallel Genetic Algorithm With The Dynamic Control Parameter", *Proceedings of IEEE International Conference on Evolutionary Computation*, 72:324-329, 1996.

[Lobo et *al.*,1998], Lobo, F.G., Deb K., Goldberg D.E., Harik G.R., "Compressed Introns In A Linkage Learning Genetic Algorithm", University of Illinois at Urbana-Champaign, 1998.

[Lobo, 2000], Lobo, F.G., *The Parameter-Less Genetic Algorithm : Rational And Automated Parameter Selection For Simplified Genetic Algorithm Operation*, PhD thesis, University of Lisbon, Portugal, 2000.

[Mader, 1987], Mader, S.S., *Biologie, évolution, diversité et en*vironnement, Éditions du Trécarré, Ottawa, 767 p., 1987

[Maniezzo, 1994], Maniezzo V., "Genetic Evolution of the Topology and Weight Distribution of Neural Networks", *IEEE Transaction on Neural Networks*, 5(1):39-53., 1994.

[Marks and Schnabl, 1999], Marks R.E., and Schnabl H. "Genetic Algorithms and Neural Networks: a comparison based on the Repeated Prisoner's Dilemma", Thomas Brenner (ed.), *Computational Techniques for Modelling Learning in Economics*, in the series Advances in Computational Economics 11, (Dordrecht: Kluwer Academic Publishers), pp. 197-219., 1999

[Martin-Bautista et *al.,* 1999], Martin-Bautista, Vila Miranda, M.A., Larsen, H.L., "A Fuzzy Genetic Algorithm Approach to an Adaptive Information Retrieval Agent." *JASIS* 50(9):760-771., 1999.

[Mercer et Sampson, 1978], Mercer, R. E.,Sampson, J.R., "Adaptive Search Using A Reproductive Meta-Plan", *Kybernetes*, 7:215-228, 1978.

[Mernik et *al.*, 2000], Mernik, M., Crepinsek, M.,Zumer, V., "A Metaevolutionnary Approach In Searching Of The Best Combination Of Crossover Operators For TSP", V: HAMZA, M. H. (ur.). *Proceedings of the IASTE Iinternational Conference on Neural Networks NN'2000*, pp. 32-36, 2000.

[Mier et *al.,* 1998], van Mier, H., L. W. Tempel, J. S. Perlmutter, M. E. Raichle, and S. E. Petersen., " Changes in Brain Activity During Motor Learning Measured With PET: Effects of Hand of Performance and Practice ", *The Journal of Neurophysiology*, 80(4):2177-2199, 1998.

[Montana, 1995], D. Montana, "Neural Network Weight Selection Using Genetic Algorithms", in *Intelligent Hybrid Systems, S. Goonatilake and S. Khebbal (eds.)*, 1995.

[Mühlenbein, 1992], Mühlenbein, H., "How Genetic Algorithms Really Work: 1. Mutation And Hill Climbing", Manner, R. and Manderick, B. (eds), *Proceedings Of The Second Conference On Parallel Problem Solving From Nature*, 2:15–25. Elsevier Science, 1992.

[Ng et Wong, 1995] Ng, K. and Wong, K., A new diploid scheme and dominance change mechanism for non-stationary function optimisation. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, 1995.

[O'Neill et Ryan, 2000], O'Neill M, Ryan C., "Incorporating Gene Expression Models Into Evolutionary Algorithms", *Proceedings Of The Workshops Of GECCO 2000, 2nd Genetic and Evolutionary Computation Conference*. With Conor Ryan, 2002.

[Osmera et *al.*, 1997], Osmera, P., Kvasnicka, V., and Pospichal,J., Genetic Algorithms with Diploid Chromosomes. In *Proceedings of Mendel'97: 3rd In-ternational Conference on Genetic Algorithms, Optimization Problems, Fuzzy Logic, Neural Networks and Rough Sets*, Brno, Czech Repub-lic, 1997.

[Pellerin et *al.*, 2004a], Pellerin, E., Pigeon, L., Delisle, S., "Self-Adaptive Parameters in Genetic Algorithms", SPIE Defence & Security Symposium 2004 -- *Proceedings of the Conference on Data Mining and Knowledge Discovery: Theory, Tools, and Technology VI*, Orlando (Florida, USA), 12-16 avril 2004, 5433:53-64., 2004.

[Pellerin et *al.*, 2004b], Pellerin, E., Pigeon, L., Delisle, S., "A Meta-Learning System Based on Genetic Algorithms", SPIE Defence & Security Symposium 2004 -- *Proceedings of the Conference on Data Mining and Knowledge Discovery: Theory, Tools, and Technology VI*, Orlando (Florida, USA), 12-16 avril 2004, 5433:65-73., 2004.

[Pham, 1995], Pham, Q.T., "Competitive Evolution: A Natural Approach To Operator Selection", *Progress in Evolutionary Computation, Lecture Notes in Artificial Intelligence*, X. Yao(*ed.*), Springer-Verlag, Heidelberg, 956:49-60, 1995.

[Pigeon *et al.*, 2001] Pigeon Luc, Inglada Jordi, Solaiman Bassel. Genetic algorithms for multi-agent fusion system learning. *Proceedings of SPIE, Sensor Fusion: Architectures, Algorithms, and Applications V, International Symposium on Aerospace/Defense Sensing, Simulation, and Controls*, Orlando, 4385:87-95, April 2001.

[Pigeon et Van Chestein, 2004], Pigeon, L., Van Chestein, Y., "Navigation in extreme environments : the Optipath capability", *Proc. RTO SCI Symposium on Systems, Concepts and Integration (SCI) Methods and Technologies for Defence Against Terrorism*, London, United Kingdom, 25-27 October 2004, RTO-MP-SCI-158 16 – 1.

[Pigeon, 2002a], Pigeon L., "An Advanced C2 Concept For Urban Operations", *Proceedings of the 7th International Command and Control Research and Technology Symposium*, Quebec City, September 2002.

[Pigeon, 2002b], Pigeon L., "A Conceptual Approach For Military Data Fusion", *Proceedings of the 7th International Command and Control Research and Technology Symposium*, Quebec City, September 2002.

[Popescu-Belis, 1997], Popescu-Belis A.,"Design of an Adaptive Multi-Agent System Based on the "Neural Darwinism" Theory". *Autonomous Agents'98 (First International Conference on Autonomous Agents)*, Marina del Rey, California, pp. 484-485., 1997.

[Rovithakis et al., 2003], George A. Rovithakis, M. Maniadakis, M. Zervakis, "A Hybrid Neural Network/Genetic Algorithm Aproach to Optimizing Feature Extraction for Signal Classification", *IEEE Transactions on Systems, Man, and Cybernetics*, Part B: Cybernetics,to appear, 2003.

[Ryan et Collins, 1998] Ryan, C. and Collins, J.J., Polygenic Inheritance - A Haploid Scheme that Can Outperform Diploidy. In *Procs. of the Fifth Int. Conf. on Parallel Problem Solving from Nature (PPSN V)*., Amsterdam, Springer LNCS 1498, pp. 178-187., September 1998.

[Schaffer et Eshelman, 1991], Schaffer, J.D., and Eshelman, L.J., "On crossover as an evolutionary viable strategy". In R.K. Belew and L.B. Booker, editors. *Proceedings of the 4th International Conference on Genetic Algorithms, Morgan Kaufmann* pages, 61-68, 1991.

[Schmidhuber, 1987], Schmidhuber J., Evolutionary Principles in Self-Referential Learning, or on Learning How to Learn: The Meta-Meta-... Hook. Diploma thesis, Institut für Informatik, Technische Universität München, 1987.

[Schmidhuber, 1996], Schmidhuber J., Zhao J., Wiering M., "Simple Principles of Metalearning", Technical Report IDSIA-, IDSIA, pp. 69-96, 1996.

[Sebag,1994]. Sebag, M. and Schoenauer, M. and Ravisé C., Evolution darwinienne ou évolution civilisée, *1ère Conférence sur l'Evolution Artificielle (EA'94)*, Toulouse, France,

[Smith et Fogarty, 1997], Smith, J.E. and Fogarty T.C., "Operator and Parameter Adaptation in Genetic Algorithms", *Soft Computing* , 1(2):81-87, 1997.

[Smith et Smuda, 1995], Smith, R.E., Smuda, E.: "Adaptively resizing populations: Algorithm, analysis, and first results". *Complex Systems* 9: 47-72., 1995.

[Spears 1995], Spears, W. M., "Adapting Crossover In Evolutionary Algorithms", McDonnel J.R., Reynolds, R.G., and Fogel, D.B.. (eds), *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, MIT Press, pp. 367–384, 1995.

[Spears W.M, De Jong, 1990], Spears W.M, De Jong K., "Using Genetic Algorithms for Supervised Concept Learning", *IEEE 1990 Second International Conference on Tools for Artificial Intelligence*, pp. 335–341, 1990.

[The Age, 2004], The Age, "Urban Warfare", 11 février 2004, www.theage.com.au/media/2003/03/27/1048653810604.html

[Tongchim et Chongstitvatana, 2002], Tongchim, S., Chongstitvatana, P., "Parrallel Genetic Algorithm With Parameter Adaptation", *Information Processing Letters*, 82(1):47-54, 2002.

[Touzet, 1992], Touzet, C., "Les réseaux de neurones artificiels : Introduction au connexionnisme ", Cours, exercices et travaux pratiques, 1992.
http://avalon.epm.ornl.gov/~touzetc/Book/Bouquin.htm#4.1

[U.S. Army, 2000] Headquarters Department of the Army. *U.S. Handbook for Joint Urban Operations (2000 - Primer for future JP 3-06: Doctrine for Joint Urban Operations*). http://www.dtic.mil/doctrine/jel/other_pubs/juohdbk1.pdf, 2000.

[United Nations Population Division, 2004], United Nations Population Division , Department of Economic and Social Affairs of the United Nations Secretariat (New York), www.unpopulation.org

[Vasconcelos et *al.*, 2001], Vasconcelos, J. A., Ramirez, J.A., Takahashi, R.H.C., Saldanha, R.R., "Improvements in Genetic Algorithms", *IEEE Trans. on AP*, 37(1):3414–3417, 2001.

[Vasconcelos et *al.*, 2001], Vasconcelos, J. A., Ramirez, J.A., Takahashi, R.H.C., Saldanha, R.R., "Improvements in Genetic Algorithms", *IEEE Trans. on AP*, 37(1):3414–3417, 2001.

[Vilalta et Drissi, 2001], Vilalta R. and Drissi Y., "Research Directions in Meta-learning", *Proceedings of the International Conference on Artificial Intelligence, (ICAI01)*, Las Vegas, Nevada. Ed. H. R. Arabnia., 2001.

[Vilalta et Drissi, 2002], Vilalta R., Drissi Y., "A Perspective View and Survey Of Meta-Learning", *Journal of Artificial Intelligence Review*, 18(2): 77-95, 2002.

[Werner et Dyer, 1991] Werner, G. and M. Dyer , "Evolution of Communication in Artificial Organisms.", In: Langton, C., et.al. (ed.) *Artificial Life II*. Addison-Wesley Pub. Co. Redwood City, Ca., 22:659-687., 1991.

[Wolpert, 1992], Wolpert D.H., "Stacked generalization", *Neural Networks*, 5(2):241-259, 1992.

[Wooldridge et Jennings, 1995], Wooldridge, M. and Jennings, N., 1995. "Intelligent Agents: Theory and Practice", *The Knowledge Engneering Review*, 10(2):115-152, 1995.

[Wooldridge, 2002], M. Wooldridge M., *An Introduction to Multiagent Systems*, John Wiley and Sons Ltd, February 2002.