Les algorithmes évolutifs*

Philippe Preux
preux@lil.univ-littoral.fr
Laboratoire d'Informatique du Littoral
BP 689
62228 Calais Cedex

Sep. 1995

Résumé

Les algorithmes évolutifs sont des algorithmes stochastiques fondés sur la simulation du processus d'évolution et d'adaptation des organismes dans les milieux naturels, adaptés à la résolution de problèmes dont l'espace de recherche est caractérisé par un grand nombre de dimensions et de nombreux optima locaux.

Dans cet article, nous présentons ces algorithmes, essentiellement comme une méthode d'optimisation de fonctions complexes. Outre leur principe de fonctionnement, nous montrons comment résoudre effectivement un problème avec leur aide. Nous synthétisons ensuite les recherches qui éclairent leur fonctionnement, la manière de les utiliser, leurs possibilités et leurs limites. Enfin, nous présentons brièvement des études où les algorithmes évolutifs sont considérés comme des modèles de systèmes où interagissent de nombreux agents. Des mécanismes de reproduction/sélection entraînent les générations d'individus à co-évoluer et se co-adapter.

1 Introduction

Un problème général en informatique consiste à parcourir un espace pour y trouver un point remarquable, un minimum ou un maximum la plupart du temps. Ainsi, on recherche le point où une fonction atteint son minimum, à minimiser le trajet d'un représentant de commerce, à exécuter au plus vite une séquence de travaux, ...

De nombreuses méthodes ont été étudiées. Nous citerons trois techniques :

- basée sur un gradient
- énumérative
- stochastique

Les méthodes basées sur l'utilisation de la dérivée de la fonction utilisent celle-ci pour estimer la direction à suivre dans le parcours de l'espace de recherche. Si ces méthodes fonctionnent bien pour les fonctions uni-modales (qui ne possède qu'un seul optima), elles sont en général rapidement piégées dans un optimum local quand elles s'attaquent à des fonctions multimodales (qui possèdent plusieurs optima). Par ailleurs, l'utilisation de ce type de méthodes présupposent une fonction connue décrivant l'espace de recherche et l'existence de sa dérivée. Ces deux propriétés sont loin d'être acquises pour tous les problèmes.

Les méthodes énumératives consistent à tester tous les points de l'espace de recherche. Pour chaque point, on mesure sa qualité comme solution au problème. Une fonction, dite *objectif*, est nécessaire pour évaluer les points. Le processus renvoie le point ayant obtenu la meilleure évaluation. Même s'il est possible d'éviter l'examen de certains points en utilisant des heuristiques (*cf.* algorithme branch-and-bound, ...), il est clair que cette méthode devient inutilisable dès que l'espace de recherche devient grand, ce qui est d'autant plus vite atteint que la dimension de l'espace est grande.

Les méthodes stochastiques ont été développées plus récemment. Certaines effectuent « simplement » un parcours stochastique de l'espace de recherche généralement en partant d'un point donné et en observant son voisinage, poursuivant la recherche jusquà ce que l'algorithme ne progresse plus ou qu'un critère d'arrêt soit atteint (*cf.* la méthode TABU par exemple

*2e édition, revue et augmentée du rapport AS-145 du LIFL ou LIL-94-1 du LIL, précédemment intitulé « Les algorithmes génétiques ». Cette publication est disponible sur www.univ-littoral.fr:/~preux, et ftp-lil.univ-littoral.fr:/pub/users/preux

[Glo89]). D'autres méthodes stochastiques se basent sur la simulation d'un système physique ou biologique. Le principe de leur utilisation est illustré à la figure 1. Ce système est composé d'agents simples qui interagissent selon des lois simples également. Le système est « laissé à lui-même » et évolue vers un état globalement stable et cohérent au cours du temps [1]. Le point crucial de cette approche consiste à modéliser le problème par un système d'agents et donc à répondre aux questions (étape 1 de la figure 1) :

- que sont les agents ?
- comment interagissent-ils ?
- comment implante-t-on ce modèle sur ordinateur ?

Un autre point tout aussi crucial, mais généralement facile à résoudre une fois le modèle et son implantation trouvés, concerne l'interprétation du résultat (la synthèse, étape 3). Pour sa part, l'évolution du modèle (étape 2) peut être vue comme une boîte noire qui fait évoluer les agents selon les lois qui y sont décrites. Il est capital pour comprendre les possibilités et les limites des processus stochastiques de bien noter l'intervention humaine lors de ces deux étapes. Comme exemple d'algorithme stochastique, nous citerons le recuit simulé [KJV83]. Cette technique donne de bons résultats, même pour des problèmes dont l'espace de recherche est de très grande dimension (des milliers de variables) ou même sur des fonctions différentiables nulle part [Ros92]. D'une manière générale, ces méthodes ne garantissent pas la découverte de la meilleure solution, mais si la fonction possède plusieurs bonnes solutions proches (en qualité) de l'optimum, elles ont de grandes chances d'en trouver une.

Autre exemple de méthode stochastique, les algorithmes évolutifs (AE) sont fondés sur l'étude de l'adaptation dans les milieux naturels et sur la simulation de ce processus. Les origines de ces techniques remontent aux années 60 et portent les noms de *stratégie évolutive* ([Rec73, Sch81]), de *programmation évolutive* ([FOW66]) et d'*algorithme génétique*. Ces derniers ont été étudiés par John Holland et son équipe de l'Université du Michigan où le terme est apparu au milieu des années 60. Toutes les idées ont été synthétisées vers le milieu des années 70 dans [Hol75]. Dans la suite de cet article, nous allons présenter ce type d'algorithmes. Nous décrivons tout d'abord les idées de base de ces travaux et un bref historique avant d'entrer dans les détails.

1. nous préférons ne pas entrer dans les détails ici, lesquels pourraient restreindre le champ de ce type d'algorithmes. Pour fixer les idées, on peut imaginer les atomes d'un cristal gazéifié, excités par la température qui, au cours du refroidissement, perdent leur énergie pour finalement atteindre un état stable et cohérent, le cristal solide.

1.1 Motivations et bref historique

Les stratégies d'évolution ont été développées comme une méthode d'optimisation numérique. La programmation évolutive l'a été comme une nouvelle voie pour l'intelligence artificielle.

Pour leur part, les algorithmes génétiques sont fondés sur la simulation du processus naturel d'adaptation qui fait qu'un être, ou une espèce, est capable de survivre, en se modifiant si nécessaire, dans un environnement. Pour ces algorithmes, le processus d'adaptation inspiré de l'adaptation des espèces dans les milieux naturels forme l'étape 2 du schéma conceptuel. Leur principe repose donc sur l'évolution d'une population d'individus dans un environnement. Cette population et son environnement forment le modèle. Incidemment, ces algorithmes constituent un modèle de choix pour l'étude des processus évolutifs naturels.

Dans sa monographie [Hol75], J. Holland présente une théorie de l'adaptation des systèmes naturels et montre comment cette théorie permet d'en déduire des systèmes artificiels adaptatifs. Plus précisément, J. Holland cherche à éclaircir deux questions :

- que sont et comment fonctionnent les processus adaptatifs naturels ?
- comment les utiliser, pour construire des systèmes artificiels qui soient capables d'adaptation ?

[Hol75] discute la notion de « plan adaptatif » qui est, techniquement parlant, un algorithme d'adaptation d'une espèce d'organismes à son environnement. De part sa nature, un plan adaptatif fait évoluer la population vers des individus de mieux en mieux adaptés à leur environnement. Notons que les individus obtenus ne sont pas forcément les mieux adaptés à leur environnement, mais le sont suffisamment pour survivre. Ce sont, vraisemblablement, les meilleurs individus qui soient apparus au cours de l'évolution.

Les travaux de Holland ont été poursuivis par ses thésards. Ainsi, K. De Jong discute dans sa thèse de doctorat [De 75] l'utilisation d'algorithmes génétiques pour l'optimisation de fonctions numériques.

Développées ensuite dans l'équipe de Holland, les études sur les AG ainsi que de nombreuses applications ont été réalisées démontrant leur utilité pratique pour résoudre des problèmes ardus.

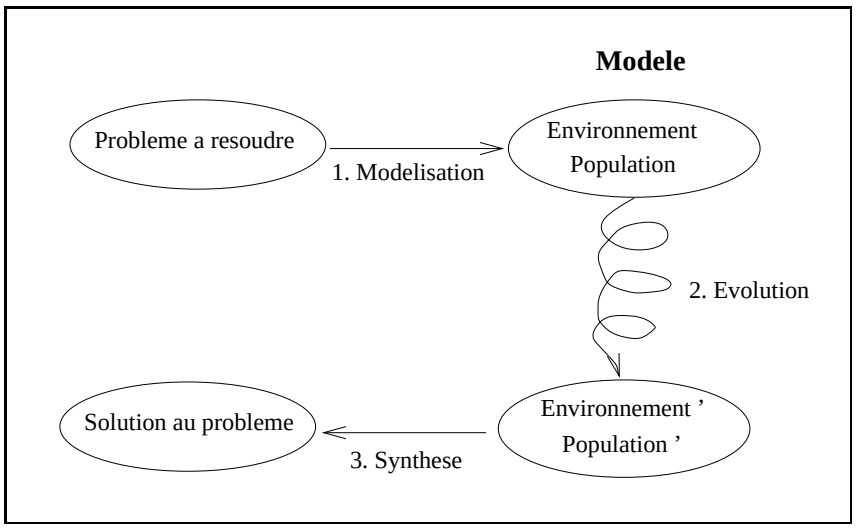


FIG. 1 – *Schéma conceptuel de l'utilisation d'un algorithme stochastique simulant l'évolution d'un système physique : une phase de modélisation met en forme le problème ; le système évolue ; l'interprétation de l'état final du système fournit une solution au problème initial.*

Au début des années 80, les AG se sont répandus progressivement dans différentes équipes de recherche aux Etats-Unis. En 1985 eut lieu la première conférence internationale sur les algorithmes génétiques, laquelle a maintenant lieu tous les deux ans. En marge du développement de nombreuses applications, Holland a poursuivi ses études sur les systèmes adaptatifs dans le domaine cognitif (quel est le rôle de l'évolution et de l'adaptation sur les capacités cognitives d'un organisme ?) et aujourd'hui dans l'étude des systèmes dits *écologiques*. En même temps, de nombreux travaux ont débuté et se poursuivent aujourd'hui pour expliquer le fonctionnement précis des AG et essayer de connaître les limites de leur domaine d'application.

Dans le reste de cet article, nous présentons tout d'abord un modèle générique d'algorithmes évolutifs. Nous instancions ensuite ce modèle pour décrire la version dite canonique des algorithmes génétiques, les algorithmes génétiques en général et d'autres. Nous nous intéressons ensuite à la manière dont les AE parcourent l'espace de recherche et comment ils synthétisent leurs solutions. Nous discutons ensuite des AE comme des heuristiques pour optimiser des problèmes réels. Enfin, nous discutons de quelques variantes aux AE pour l'intelligence artificielle, l'étude du phénomène d'adaptation et de la vie.

Avant d'aller plus avant, nous désirons indiquer que nous utilisons dans cet article un ensemble de termes chargés de sens dans le domaine de la biologie dans un contexte très différent, l'informatique. Ces termes ne sont nullement introduits par nous, mais font partie du « jargon » du domaine. La difficulté vient surtout du fait que l'on utilise ici ses termes avec une vague intuition biologique, ne prenant finalement que la partie du sens qui nous intéresse, celle que l'on arrive à implanter et à utiliser.

2 Un algorithme évolutif générique

Dans cette section, nous donnons une définition générique des algorithmes évolutifs basée sur une (et une seule) population. Cette définition entend recouvrir les différentes variantes ayant été proposées. Le but n'est pas de proposer une définition formelle des algorithmes évolutifs dont nous ne saurions que faire, mais de préciser les caractéristiques que l'on peut considérer comme essentielles pour qualifier un algorithme d'évolutif ou non.

2.1 Définition de l'algorithme

D'une manière générale, un AE simule l'évolution d'une population d'individus. Cette évolution est stochastique et se déroule dans un temps discrétisé. Chaque individu évolue dans un « environnement » auquel il est plus ou moins adapté. Cette mesure de son degré d'adaptation est explicite et indique sa capacité à produire des clones de lui-même. À chaque génération, une méthode de sélection forme la population

de la génération suivante (*cf.* figure 2).

2.2 Modèle

D'une manière générale, on peut donc définir un AE comme un septuple :

$$< \nu, \mathcal{G}, \Omega, \phi, \varphi, \upsilon, s >$$

où :

- ν représente le nombre d'individus formant la population
- $\mathcal{G}$ est l'ensemble de tous les individus possibles
- Ω l'ensemble des opérateurs : $\Omega = \{(\omega_i, p_i)\}$ où ω_i dénote un opérateur particulier, p_i son taux d'application. On distingue ainsi essentiellement deux types d'opérateurs :
  1. opérateurs dits de « recombinaison » qui recombinent deux (ou plus) individus pour en produire un (ou plus) nouveau ;
  2. opérateurs dits de « mutation » qui agissent sur un individu pour en produire un nouveau.
- ϕ la fonction objectif qui s'applique aux individus et rend, généralement, une valeur réelle :

$$\begin{array}{rccc} \phi : & \mathcal{G} & \rightarrow & \mathbb{R} \\ & \chi & \mapsto & \phi(\chi) \end{array}$$

  La valeur rendue par ϕ sera dénommée la « performance » de l'individu.
- φ, la probabilité [2] d'être cloné :

$$\begin{array}{rccc} \varphi : & \mathcal{G} & \rightarrow & \mathbb{R}^+ \\ & \chi & \mapsto & \varphi(\chi) \end{array}$$

  qui indique, pour un individu donné de la population à l'instant t la probabilité qu'il fournisse des clones de lui-même dans la population à la génération suivante. φ est une fonction de ϕ. La valeur rendue par φ sera dénommée le « fitness » de l'individu. On pose : $\varphi_{min} = \min_{\{\chi, \chi \in \mathcal{P}\}}(\varphi(\chi))$ et $\varphi_{max} = \max_{\{\chi, \chi \in \mathcal{P}\}}(\varphi(\chi))$.
- φ donnant un résultat à valeur réelle, l'algorithme ne fonctionnant, bien entendu, que sur des individus complets, une fonction auxiliaire υ est utilisée qui transforme une probabilité pour un individu de fournir des enfants dans le nombre d'enfants qu'il fournit effectivement, soit :

$$\begin{array}{rccc} \upsilon : & \mathcal{G} & \rightarrow & \mathbb{N} \\ & \chi & \mapsto & \upsilon(\chi) \end{array}$$

- s, la stratégie de sélection, un algorithme qui, à partir de la population des parents $\mathcal{P}_t$ et de la population des enfants forme la population des descendants $\mathcal{P}_{t+1}$:

$$\begin{array}{rccc} s : & \mathcal{G}^\nu \times \mathcal{G}^\nu & \rightarrow & \mathcal{G}^\nu \\ & (\mathcal{P}_t, \Omega(\upsilon(\mathcal{P}_t)) & \mapsto & \mathcal{P}_{t+1} \end{array}$$

  où, par abus de notation, on note $\Omega(\mathcal{P})$ la population obtenue par application des opérateurs sur la population $\mathcal{P}$, $\upsilon(\mathcal{P})$ la population formée par les clones de la population de parents $\mathcal{P}$.

En cas de besoin, nous indexerons ces quantités par un indice t symbolisant la génération qui nous intéresse. Ainsi, $\mathcal{P}_t$ dénotera la population de l'AE à la génération t ; si la cardinalité de la population varie au cours du temps, on notera ν_t, ...

2.3 Définition des individus de la population

Un individu est défini par son « génotype », c'est-à-dire, une structure de données. Ce génotype est exprimé par la fonction objectif pour obtenir son « phénotype ». Celui-ci indique l'aptitude de l'individu à subsister dans son environnement (donc à se reproduire).

Le génotype d'un individu est constitué d'un ensemble de gènes que nous noterons γ_i avec $1 \leq i \leq \lambda$, ou λ dénote le nombre de gènes des individus. Dans certains AE, ce nombre est variable en fonction du temps ou des individus, auquel cas nous l'indexerons de manière appropriée.

Chaque gène peut prendre un certain nombre de valeurs dénommées "allèles". On notera $\gamma_i \in \{\alpha_i^j\}$. Généralement, l'ensemble des allèles est fixé et constant au cours de l'évolution.

2.4 Quelques définitions dérivées

Nous définissons ici un ensemble de termes que nous utiliserons ensuite librement dans le reste de ce rapport. Mis à part la dernière définition (pression

2. par abus de langage, nous parlons également de probabilité pour des nombres plus grands que 1. Dans ce cas, il faut comprendre que l'individu fournit le nombre de clones spécifié par la partie entière, et a la probabilité correspondante à la partie fractionnaire d'en produire un de plus.

de sélection), les autres définitions sont utilisées par tout le monde.

Définition 1 (Performance moyenne de la population) *est la moyenne sur l'ensemble de la population à une génération t fixée de la performance des individus composant la population $\mathcal{P}_t$:*

$$\frac{\sum_{\chi \in \mathcal{P}_t} \phi(\chi)}{\nu}$$

Définition 2 (Performance moyenne d'un allèle) *est la moyenne sur l'ensemble de la population à une génération t fixée de la performance moyenne des individus dont un certain gène fixé γ_i possède un certain allèle α_i^j. Notant P l'ensemble de ces individus, la performance moyenne de l'allèle sera :*

$$\frac{\sum_{\chi \in P} \phi(\chi)}{|P|}$$

Enfin, nous proposons de définir la notion de « pression de sélection », notion qui n'est pas toujours clairement définie, de la manière suivante :

Définition 3 (pression de sélection) *On définit la pression de sélection ρ par : $\rho = \varphi_{max} - \varphi_{min}$*

Ainsi définie, la pression de sélection est un paramètre important de la manière dont l'algorithme évolue (*cf* section 4.1).

2.5 Évolution de l'algorithme

Ces définitions étant posées, un AE part d'une population initiale d'individus. Ceux-ci représentent des solutions au problème que l'on veut résoudre. Ils sont générés aléatoirement ou non. La population évolue alors effectuant à chaque génération le cycle indiqué plus haut de reproduction/sélection. La population de la génération suivante $t+1$ est constituée soit exclusivement des individus qui viennent d'être engendrés à la génération t, soit par une sélection d'individus ayant joués le rôle de parents à la génération t et d'individus qui viennent d'être engendrés (*cf.* figure 2).

Au cours de son évolution, la population tend à converger, c'est-à-dire que les individus tendent à se ressembler de plus en plus. Quant la population s'est uniformisée en grande partie, les individus fournissent une bonne approximation d'un optimum du problème. Si cet optimum n'est pas toujours l'optimum global, c'est généralement un optimum local proche (en qualité) de celui-ci. Il est clair que l'AE étant stochastique, deux exécutions peuvent converger vers des optima différents.

3 Instanciations du modèle

Nous décrivons dans cette section plusieurs instanciations du modèle générique. Parmi celles-ci, nous nous intéressons longuement aux algorithmes génétiques de J. Holland et à une version particulière dite *canonique* du fait de son importance théorique. Nous décrivons également rapidement d'autres instanciations du modèle qui forment des branches où se déroulent des recherches actives et qui ont donné lieu à des applications concrètes.

3.1 L'algorithme génétique canonique

Dans cette section, nous présentons la version dite canonique des algorithmes génétiques (abrégé désormais en AGC). Ce n'est pas strictement l'algorithme décrit originalement par J. Holland, mais il en reste très proche.

3.1.1 Codage des individus

Dans sa version canonique, les gènes sont des bits, donc les allèles 0 et 1. Les individus sont des chaînes de bits de longueur λ fixée et identique pour tous les individus. La cardinalité de la population ν est constante au cours du temps.

Nous décrivons maintenant les différentes composantes de l'AGC : le clonage des individus, les opérateurs génétiques et les mécanismes de sélection proposés pour l'AGC. Nous discutons également de la fonction objectif.

3.1.2 Clonage des individus

Les individus les mieux adaptés tendent à fournir la descendance la plus nombreuse : le nombre d'enfants (clones) effectivement engendrés par un individu est biaisé par sa performance. Dans l'AGC, on engendre autant d'enfants qu'il y a de parents, donc ν. Plusieurs méthodes de clonage ont été proposées. À chacune correspond une fonction fitness φ. Nous présentons deux méthodes. La première, dite de la roulette, est celle qui définie véritablement l'algorithme canonique. Cependant, posant un certain nombre de problèmes lors de la résolution de problèmes réels, d'autres méthodes ont été proposées afin d'y remédier. Nous présentons l'une d'elles que nous nommons méthode du rang (*ranking* en anglais).

La méthode la roulette La méthode la plus ancienne consiste à ce que le nombre d'enfants d'un individu soit proportionnel à sa performance. Celle-ci est dénommée la méthode de la roulette car elle revient

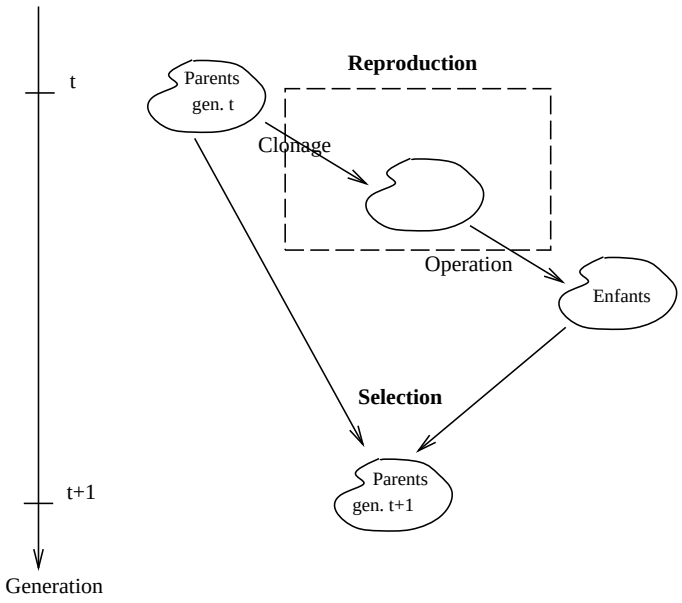


FIG. 2 – *Principe de base des actions réalisées à chaque génération d'un algorithme évolutif: reproduction des parents (composée d'une phase de clonage suivie d'une phase d'opération) pour obtenir une population d'enfants, suivie d'une sélection parmi les parents et les enfants qui formeront la population de parents de la génération suivante.*

à attribuer à chaque individu une section de surface proportionnelle à sa performance sur une roulette de casino puis à la faire tourner. Celle-ci a d'autant plus de chance de s'arrêter sur un numéro que la section correspondante est de surface plus importante. Cependant, dès qu'un individu parvient à une performance largement supérieure à celle de ses congénères (ce qui peut être du au hasard, lors du choix de la population initiale), si on lui attribue un nombre de descendants directement lié à sa performance, les générations suivantes ne compteront bientôt plus que ses descendants, à quelques individus près. Si cet individu représente un optimum local, on perd pratiquement toute chance de trouver un meilleur optimum.

Méthode du rang Afin de pallier ces problèmes, J. Baker[Bak85] a proposé plusieurs méthodes dont celle du rang que nous décrivons ici. Elle consiste tout d'abord à ordonner les individus en fonction de leur performance, le meilleur ayant le rang 1, le moins bon le rang ν. Supposons alors que nous voulions que le meilleur individu engendre au plus $\lceil \eta_{max} \rceil$ (généralement égal à 2) descendants et au moins $\lfloor \eta_{min} \rfloor$ (généralement égal à 0). On prend alors :

$$\varphi(i) = \eta_{max} - (\eta_{max} - \eta_{min})\frac{i-1}{\nu - 1}$$

où i est le rang de l'individu, $\eta_{min} = 2 - \eta_{max}$ et $1 \leq \eta_{max} \leq 2$. Chaque individu engendre $\upsilon(i)$ clones :

$$\upsilon(\chi_i) = \lfloor \varphi(i) \rfloor + 1((\varphi(i) - \lfloor \varphi(i) \rfloor) \geq r_i)$$

où r_i est un nombre aléatoire compris entre 0 et 1, la fonction $1(p)$ s'applique à un prédicat p est vaut :

$$1(p) = \begin{cases} 1 & \text{si p est vrai} \\ 0 & \text{sinon} \end{cases}$$

Le principe de ce calcul est illustré à la figure 3. L'intérêt de cette méthode réside dans le fait que la population n'est plus rapidement dominée par un « super-individu » comme dans le cas de la méthode de la roulette.

Discussion Le but poursuivi dans ces différents algorithmes est de ne pas se laisser submerger par des individus forts qui ne sont pas forcément des optima globaux. Il s'agit là d'un compromis où il est nécessaire de garder les meilleurs, sans se laisser dominer par eux pour poursuivre la quête de meilleurs individus.

Pour éviter une trop grande modification de la population à chaque génération, la « stratégie stable[3] »

3. *steady-state strategy*

rang	rang modifié	ρ	Nombre de descendants
1	2.0	-	2
2	1.8	0.267	2
3	1.6	0.736	1
4	1.4	0.534	1
5	1.2	0.102	2
6	1.0	-	1
7	0.8	0.823	0
8	0.6	0.002	1
9	0.4	0.231	1
10	0.2	0.344	0
11	0.0	-	0

FIG. 3 – *Méthode du rang : en prenant* $\eta_{max} = 2, \nu = 11$

[Whi88] limite la génération de nouveaux individus à quelques individus (en général, 1 ou 2 individus seulement). La convergence est alors plus lente mais tombe moins facilement dans le piège des optima locaux.

Une variante à cette stratégie est la stratégie sans doublons[4] qui évite l'apparition de doublon dans la population, c'est-à-dire d'engendrer des individus qui y sont déjà présents.

3.1.3 Les opérateurs

Après clonage, les opérateurs génétiques manipulent les individus-enfants. Communément, deux types d'opérateurs sont utilisés (*crossover* et *mutation*). Ces deux opérateurs ont été initialement décrits dans [Hol75]. Le crossover travaille par couples d'individus. Les clones sont donc regroupés par paires. Le crossover est appliqué avant les autres opérateurs. L'idée est ici de mimer la reproduction naturelle sexuée où le génotype d'un enfant est obtenu par recombinaison des génotypes des deux parents après quoi, des modifications stochastiques (mutations) peuvent intervenir sur le génotype de l'enfant.

Le crossover Dans l'AGC, l'opérateur génétique considéré comme le plus important est l'opérateur de recombinaison. C'est un opérateur de « reproduction sexuée » qui prend deux individus et combine leurs gènes. Dans l'AGC, l'opérateur de recombinaison porte le nom de crossover, son fonctionnement ressemblant par certains aspects au mécanisme génétique de crossing-over.

4. *steady-state without duplicates*

L'opérateur de crossover agit sur des paires d'individus (*cf.* fig. 4(a), 4(b), 4(c)). De nombreuses variantes existent. Un crossover consiste à couper en un ou plusieurs points deux individus (aux mêmes endroits dans les deux individus), et à échanger les gènes situés entre ces points. On distingue ainsi :

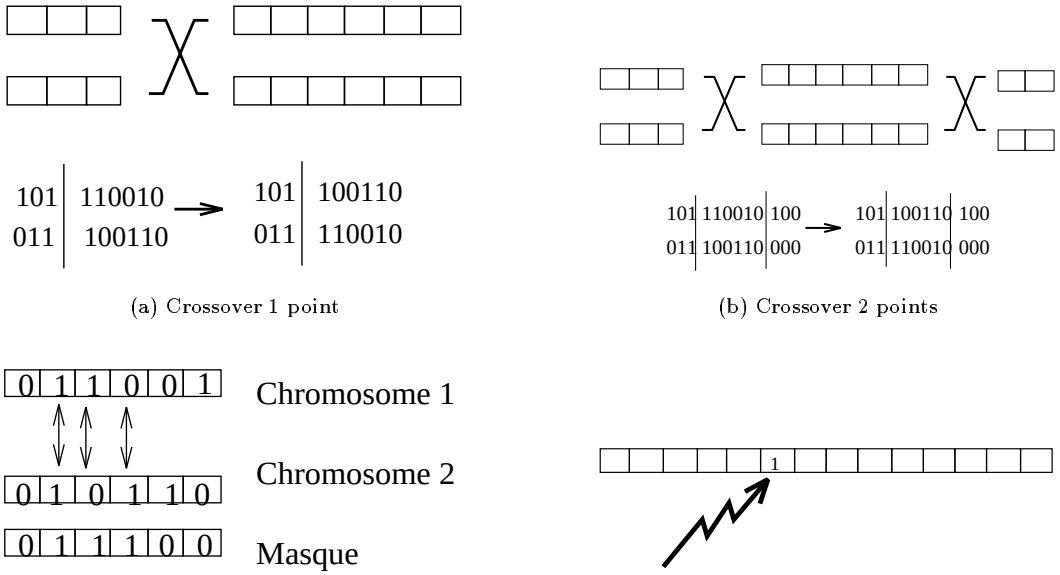
- le crossover 1-point ([Hol75], *cf.* fig. 4(a)) où les individus sont coupés en un point et l'une des extrémités des deux génotypes échangées entre les individus ;
- le crossover 2-points ([De 75], *cf.* fig. 4(b)) où les individus sont coupés en deux points et les parties centrales des deux individus sont échangées ;
- le crossover n-point (généralisation du précédent, [Sys89]) où les individus sont coupés en n points et où un tronçon sur deux est échangé ;
- le crossover uniforme ([Sys89], *cf.* fig. 4(c)) utilise une chaîne de bits générée aléatoirement et de même longueur que les individus. Les gènes des individus initiaux sont échangés en fonction de cette chaîne aléatoire, lorsque le bit correspondant vaut 1. Dans la chaîne de bits, les 0 et les 1 ne sont pas forcément générés de manière équi-probable.

Ceci dit, la détermination des individus à croiser laisse encore plusieurs possibilités : croise-t-on des individus choisis au hasard ou des individus dont la performance est proche ou dont le génotype est différent ... ? Pour sa part, [Hol75] propose de croiser un individu choisit en fonction de ses performances avec un individu pris au hasard. La question demeure ouverte.

La mutation L'opérateur de mutation agit sur un individu (*cf.* fig. 4(d)). Il consiste à choisir un ou plusieurs gènes au hasard dans un individu et à modifier leurs allèles de manière aléatoire, en les forçant ou en les complémentant.

3.1.4 La sélection

Dernière étape durant une génération, la sélection doit constituer la population de la génération suivante à partir des parents et des enfants de la génération courante. Dans l'AGC, il est engendré autant d'enfants qu'il y a de parents et tous les enfants remplacent leurs parents à chaque génération. Cette stratégie est dite « générationnelle ». Rapidement, K. De Jong [De 75] a proposé de ne pas remplacer l'ensemble de la population à chaque génération et



(a) Crossover 1 point

(b) Crossover 2 points

(c) Crossover uniforme

(d) Mutation

FIG. 4 – *Les opérateurs génétiques de crossover et mutation de l'AGC*

a introduit pour cela la notion d' « écart entre les générations[5] », un nombre compris entre 0 et 1 qui indique la proportion de parents qui sont remplacés par des enfants. Du moment que l'on conserve certains individus mais pas tous, la question du choix des individus conservés apparaît. Celui-ci est généralement basé sur la performance des individus : plus un individu est performant, plus il a de chances de survivre. Certaines stratégies prennent les meilleurs de manière déterministe. D'autres utilisent la performance comme biais pour la sélection.

La sélection des survivants étant stochastique, le meilleur individu d'une génération peut ne pas survivre. Aussi, la notion de stratégie élitiste (resp. k-élitiste) a été également introduite pour être certain que le meilleur individu (resp. les k meilleurs individus) d'une population soit gardé d'une génération à la suivante. Notons que dans les stratégies de reproduction stable, les descendants remplacent généralement les plus mauvais individus de la population.

3.1.5 La fonction objectif

Le choix de la fonction objectif, ou « fonction d'évaluation » ou encore « fonction coût », est un problème subtil. Le champ de ce qui va être dit ici ne se restreint d'ailleurs pas au seul AGC mais à tous les AE.

5. *generation gap*

La fonction objectif prend en argument un individu et agit en deux temps :

- décodage de l'individu, c'est-à-dire interprétation de la chaîne de bits. Cela peut être vu comme l'exhibition du phénotype de l'individu ;
- calcul de la valeur de ce phńotype comme une solution au problème fournissant la performance de l'individu.

Ces deux points sont d'une importance cruciale. Le premier point sera abordé un peu plus loin sur un exemple (*cf.* section 3.2.1). Nous nous intéressons ici au calcul de la performance elle-même, une fois l'individu décodé. Ne connaissant pas la solution au problème, la fonction d'évaluation doit « guider » l'algorithme vers l'optimum Aussi, il est clair qu'une simple fonction à résultat booléen du type :

$$f(\chi) = \begin{cases} 1 & \text{si l'individu } \chi \text{ est l'optimum global} \\ 0 & \text{sinon} \end{cases}$$

n'a aucun intérêt. La fonction d'évaluation définit l'environnement dans lequel se développe cette "culture" d'individus. Pour que les individus évoluent dans la direction voulue (les individus les mieux adaptés, c'est-à-dire, les meilleures solutions au problème), la fonction doit implicitement réaliser une pression de sélection dans cette direction, laquelle est a priori

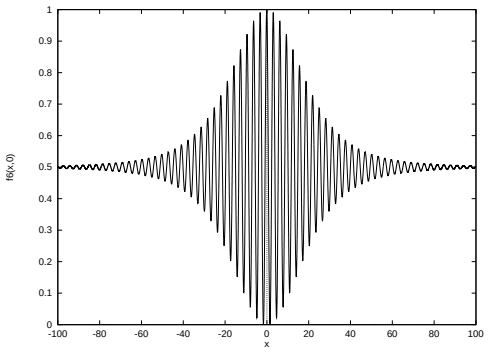


FIG. 5 – *Le graphe de la fonction f, pour y nul et x variant entre -20 et +20*

inconnue. Si pour la plupart des problèmes numériques il est facile de trouver une fonction respectant de telles conditions, il devient beaucoup plus difficile de trouver la fonction d'évaluation pour des problèmes plus formels où il existe une et une seule solution, toutes les autres étant, du point de vue du résultat lui-même, également mauvaises. Par exemple, la question a été étudiée pour la résolution du problème de satisfiabilité d'une expression booléenne [6] : soit la valuation des variables donne à l'expression une valeur 1, soit elle lui donne la valeur 0. Elle revient à la fonction citée plus haut et déjà jugée inutile. Nous laissons ce type de problème de côté et nous intéressons pour l'instant aux cas relativement simples de fonctions pour optimiser des problèmes numériques (on pourra consulter [DS89]).

3.2 Exemple de résolution d'un problème avec l'AGC

Nous présentons un exemple simple d'utilisation de l'AGC. Nous résolvons un problème numérique typiquement difficile pour de nombreuses méthodes car présentant de nombreux optima locaux (cf. figure 5).

La fonction f à résoudre est la suivante :

$$f(x,y) = 0.5 - \frac{(\sin\sqrt{x^2+y^2})^2 - 0.5}{(1.0+0.001(x^2+y^2))^2}$$

Il s'agit de trouver le point (x_0, y_0) pour lequel $f(x,y)$ est maximale. Cette fonction est symétrique par rapport à l'axe des ordonnées. Le maximum est situé au point (0, 0) où la fonction vaut 1.

6. le problème de satisfiabilité d'une expression boolénne consiste à trouver une valuation des variables intervenant dans l'expression telle que la valeur de l'expression soit 1

Pour résoudre un problème à l'aide de l'AGC, il faut déterminer les points suivants :

- la taille des individus,
- la fonction objectif,
- les paramètres de l'algorithme.

Nous traitons chacun de ces points puis donnons les résultats des simulations.

3.2.1 Le codage des individus

L'équation possède deux variables. Aussi, les individus sont formés de deux parties, chacune représentant la valeur d'une variable. Les représentations binaires des variables sont concaténées pour composer les individus. Le codage d'une variable nécessite la spécification de l'intervalle dans lequel la résolution a lieu (bornes x_{max} et x_{min}), le nombre de bits utilisés $\lambda/2$ indiquant la précision de la résolution. L'intervalle de recherche est simplement découpé en $2^{\lambda/2}$ parties. Pour la valeur de la variable, la précision obtenue est $\frac{x_{max}-x_{min}}{2^{\lambda}}$. Le principe de ce codage binaire acquis, on peut encore choisir entre utiliser un codage binaire que nous dirons pur (celui que l'on vient de décrire) ou un codage de Gray. Le codage de Gray est sensé rendre codage plus « continu » évitant les « falaises de Hamming [7] », c'est-à-dire que quel que soit le nombre x codé, la probabilité d'obtenir $x \pm 1$ par mutation ne dépend pas de x en codage de Gray, contrairement au codage dit binaire pur. Ainsi, pour muter 0 en 1 ou 7 en 8, il suffit de muter 1 bit en codage de Gray. En binaire pur, il faut une mutation pour passer de 0 à 1, 4 pour passer de 7 à 8. Autrement dit, en codage de Gray, l'action de la mutation sur un génotype ne dépend pas de son phénotype.

3.2.2 La fonction objectif

La performance d'un individu sera obtenue par application de la fonction f sur l'individu. L'algorithme cherchera alors à maximiser la performance des individus. La fonction objectif reçoit un individu qui devra être décodé pour que sa performance $f(x,y)$ puisse être calculée.

3.2.3 Simulation

Les simulations ont été réalisées avec *genesis* ([Gre90]) un simulateur disponible sur réseaux électroniques. Il s'agit d'une plate-forme développée il y a une dizaine d'années et qui a été reprise maintes fois et de laquelle

7. *hamming cliff*

sont dérivés de nombreux autres simulateurs. Ce simulateur permet d'exécuter l'AGC en faisant varier différents paramètres, tels :

- la population $\mathcal{P}_0$ initiale (aléatoire ou non) ;
- la cardinalité ν de la population ;
- le nombre de bits λ codant un individu ;
- les taux d'application des opérateurs de crossover et de mutation ;
- l'écart entre les générations ;
- le nombre d'appels à la fonction d'évaluation (pour limiter le nombre de générations) ;
- stratégie élitiste ou non ;
- codage binaire ou codage de Gray.

Notons que ces paramètres sont fixés pour le déroulement d'une simulation entière et ne peuvent varier dynamiquement.

Les expériences Nous codons chaque variable sur 30 bits consécutifs. L'individu est donc une chaîne de bits de $\lambda = 60$ bits. Pour chaque variable, l'intervalle de recherche est $[-5.12, 5.12]$, la résolution d'environ 10^{-8}.

Etant donnée la nature stochastique des AG, le résultat de l'exécution d'une seule simulation n'est pas suffisant. Pour un même jeu de paramètres et une fonction objectif donnée, deux exécutions donnent des résultats différents. Aussi, pour chacune des simulations discutées ici, nous avons exécuté cinquante expériences avec le même jeu de paramètres. Nous résumons en quelques mots les résultats obtenus.

Nous avons utilisé les paramètres standards de *genesis* (taux de crossover : 0.6, taux de mutation : 0.001, stratégie générationnelle élitiste, utilisation d'un codage de Gray, clonage par ranking). Dans ces conditions, 95 % des individus convergent vers une bonne approximation de la solution (ϵ_x, ϵ_y) (les ϵ_x et ϵ_y sont différents dans chaque cas) dans environ 30 % des expériences en réalisant 10000 évaluations. Dans les autres cas, l'algorithme converge vers un des otpima très proches.

Ces valeurs non nulles (c'est-à-dire, des allèles non nuls) sont dues à deux causes : la présence d' « autostoppeurs » (*cf.* 4.4) et un « bruit » causé par la mutation qui est toujours active. Aussi, la précision obtenue est limitée par ce bruit de fond qui agite les gènes peu significatifs des individus (les bits significatifs eux convergent car la sélection élimine les individus dont les bits significatifs seraient modifiés par mutation, entraînant de mauvaises performances de l'individu). Notons qu'il existe des techniques modifiant le principe strict de l'AGC pour obtenir une meilleure précision sur des problèmes numériques (*cf.* [SB92], [WMF91]).

3.2.4 Phénomènes génétiques dans l'AGC

Dans le cas de l'AGC, les individus sont manipulés par les opérateurs via leur génotype et ce de manière totalement aveugle (c'est-à-dire, sans s'intéresser au phénotype de l'individu en question). Le génotype étant implanté sous forme d'une suite de bits contigus en mémoire, ces manipulations entraînent des phénomènes classiques en génétique d'épistasie, de dérive génétique et d'oscillation de la fréquence allèlique pour des gènes ayant une faible incidence sur la performance des individus ([Pre95]).

Épistasie signifie non addivité des performances des allèles dans le génotype. C'est-à-dire qu'il y a des combinaisons d'allèles qui, lorsqu'elles sont présentes dans un génotype, modifient fortement la performance de l'individu par rapport à une combinaison incomplète (même s'il ne manque qu'un seul allèle).

La dérive génétique est un phénomène dû au caractère fini de la population. Il s'agit du fait que certains allèles disparaissent de la population par hasard, parce que les individus dans lesquels ils apparaissent n'ont pas été sélectionnés. Dès lors, la probabilité de leur ré-apparition du fait d'une mutation est très faible. Ce phénomène peut empêcher certaines combinaisons d'allèles épistatiques, donc la perte d'autres allèles qui auraient pu, par combinaison, être très bénéfiques.

L'oscillation de la fréquence allélique pour un gène donné est due au caractère stochastique du processus et aux deux phénomènes qui viennent d'être mentionnés, épistasie et dérive génétique (*cf.* figure 6). Ces oscillations rendent complexe l'analyse de l'évolution de la population d'un AGC [Pre95].

3.3 Les algorithmes génétiques

Basées sur le principe de l'AGC, des variantes ont été proposées où le codage et les opérateurs diffèrent afin de résoudre efficacement des problèmes divers d'optimisation. Nous indiquons quelques directions :

- utiliser un alphabet d'allèles non binaire,

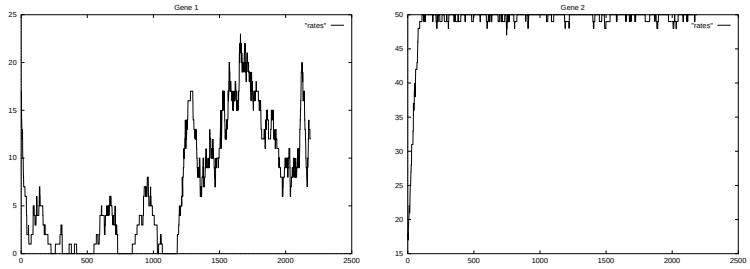


FIG. 6 – *On a représenté la fréquence allélique de deux gènes montrant des évolutions possibles de cette fréquence : oscillation et domination de l'un des allèles.*

- utiliser de nouveaux opérateurs génétiques, dédiés, par exemple, à des allèles non binaires ou spécifiques à la résolution du problème traité,
- avoir des populations de taille non constante au cours de l'évolution,
- avoir des individus de taille non constante au cours de l'évolution,
- avoir plusieurs populations évoluant concurremment,
- avoir plusieurs espèces d'individus,
- prendre en compte un environnement évoluant dans le temps,
- prendre en compte un environnement non déterministe,
- utiliser des opérateurs adaptatifs.

Ces points et de nombreux autres sont des sujets de recherche. Dans la suite, nous discutons de l'intérêt d'utiliser des allèles non binaires et de définir de nouveaux opérateurs. Des études allant dans ce sens ont été développées depuis longtemps et des applications réelles des AG sont fondées sur ces principes.

3.3.1 Alphabets non binaires

L'utilisation d'alphabets d'allèles non binaires consiste à considérer les individus comme des structures de données composées de champs, plutôt que comme des chaînes de bits. Toute structure de données étant finalement codée en bits, on peut considérer ce choix comme une question d'ergonomie : plutôt que chercher à exprimer les individus sous forme de chaînes de bits que la fonction objectif doit interpréter avant d'évaluer, les individus sont des structures de données manipulées en tant que telles.

Cependant, des problèmes dus au mode de fonctionnement des opérateurs peuvent survenir : ceux-ci opèrent sur des chaînes de bits. Aussi, ils peuvent entraîner l'apparition d'allèles non valides[8]. Pour y pallier, plusieurs approches sont possibles :

- réparer les gènes invalides ;
- modifier les opérateurs, ou en introduire de nouveaux, pour qu'ils n'engendrent que des gènes valides ;
- utiliser des mécanismes de pénalités pour amoindrir la performance des individus malformés.

Nous allons illustrer le propos sur l'exemple du problème du voyageur de commerce. Nous montrons sur cet exemple le choix de la forme du génotype et des conséquences sur le fonctionnement des opérateurs. Nous montrons également comment de nouveaux opérateurs sont introduits.

Le problème du voyageur de commerce consiste à parcourir un ensemble de villes caractérisées par leurs coordonnées sur une carte, de manière à minimiser la longueur du voyage. On se place ici dans le cas le plus simple où tous les chemins entre tous les couples de villes existent et où les distances sont symétriques entre deux villes. Dans ces conditions, la donnée du problème est une liste de couples de coordonnées (deux nombres).

Le chemin le plus court étant forcément un cycle hamiltonien, on prend généralement $\mathcal{G} = \{\text{cycle hamiltonien}\}$. Les villes étant numérotées de 0 à $\lambda - 1$, les allèles sont l'ensemble des entiers de 0 à $\lambda - 1$: $\gamma_i \in \{\alpha \in [0, \lambda - 1]\}$. Pour qu'un individu code un cycle hamiltonien, l'ensemble des allèles d'un individu doit être équipotent à $[0, \lambda - 1]$.

8. loin d'être un phénomène marginal, si l'on n'y prend garde, la plupart des individus générés seraient invalides

La fonction objectif renvoie simplement la longueur du chemin codé dans l'individu qui lui est proposé. En cherchant à minimiser cette valeur, l'algorithme convergera naturellement vers des chemins de plus en plus courts.

Utilisant cette structure pour les individus, nous devons modifier le fonctionnement des opérateurs génétiques pour qu'ils la prennent en compte et la respecte. Par ailleurs, nous devons rejeter les opérateurs qui n'ont pas de sens dans ce nouveau contexte et, pourquoi pas, ajouter de nouveaux opérateurs qui sembleraient adaptés.

On peut se représenter la structure des individus comme des listes chaînées dont chaque élément est un champ contenant un numéro de ville. Les opérateurs ne peuvent alors modifier que les liens reliant les champs des individus, ou la valeur d'un élément de la liste.

Le crossover choisit un, deux, ... n liens à couper (aux mêmes endroits dans les deux individus) et échange, entre les deux individus concernés, les gènes situés entre deux points de coupure. En agissant ainsi, nous sommes sûrs que l'individu résultant consistera bien en une liste de villes (chaque gène est valide). Cependant, des doublons risquent fort d'apparaître ou des villes de manquer. Aussi, nous devons encore modifier le crossover pour qu'il respecte également le fait qu'un individu doive coder un cycle hamiltonien. Différentes manières de construire un chemin valide sur ce principe ont été proposées et ont donné naissance à des opérateurs connus sous les noms de OX, CX, PMX, ER (*cf.* [SMM+91]) et MPX [Müh91]. En guise d'exemple, nous présentons la première version de l'opérateur de recombinaison d'arcs (ER) proposé par D. Whitley & *al* [WSS91].

L'opérateur de recombinaison d'arcs est intéressant parce qu'il fonctionne suivant un mécanisme indirect qui est nouveau par rapport à ce que nous avons dit pour l'instant. Il s'applique sur deux parents et fournit un enfant. À l'aide des génotypes des deux parents, on commence par construire la table des arcs qui va être utilisée pour construire le génotype de l'enfant. Cette carte est une table comportant λ entrées, chacune correspondant à une ville et contenant les numéros des villes auxquelles elle est reliée (il y a donc toujours 2, 3 ou 4 valeurs associées). Ensuite, on exécute l'algorithme suivant :

1. choisir une ville initiale (la ville courante) ;
2. retirer cette ville des listes de la table des arcs ;
3. si la ville courante possède une entrée non vide dans la table, la ville de cette entrée dont l'entrée est la plus courte devient la ville courante ; sinon, prendre au hasard une ville non encore visitée qui devient la ville courante. Si toutes les villes ont été visitées, l'algorithme est terminé ;
4. retourner en 2.

Cet algorithme est illustré à la figure 7.

Connu comme le meilleur opérateur de recombinaison pour le TSP, plusieurs améliorations en ont ensuite été proposées [MW92], [TL94].

La mutation qui consiste à modifier arbitrairement la valeur d'un gène est totalement inadaptée pour ce type d'individus. La nouvelle valeur, pour être valide, devra être le numéro d'une ville différent de l'allèle du gène muté. Or, toutes les villes se trouvent dans un individu : si on ne modifie que la valeur d'un seul gène, il y aura ensuite un doublon dans l'individu. Nous ne devons donc pas nous attacher plus longtemps aux mutations de ce genre.

De nouveaux opérateurs Remplaçant la mutation, nous pouvons aisément imaginer un opérateur qui conservera toujours la structure correcte des individus : une permutation de la valeur de 2, 3, ... n gènes d'un individu. On peut également utiliser une heuristique en place de la mutation comme une méthode de recherche locale, la méthode TABU, ou, dans le cas du TSP, l'heuristique 2-opt de S. Lin et B.W. Kernighan [LK73]

3.3.2 AG pour la résolution de problèmes numériques

L'AGC a été utilisé plus haut pour résoudre un problème d'optimisation numérique. Il s'agissait alors de montrer sur un exemple simple qu'un AE simple obtient l'optimum. Notre propos n'était en aucun cas de dire que l'AGC est l'AE le mieux adapté à ce type de problème. Ainsi, pour des fonctions numériques multi-dimensionnelles pour lesquelles il s'agit de trouver $x =< x_1, x_2, ...x_\lambda >$ tel que :

$$\begin{cases} f(x) \text{ doit être optimum} \\ \text{les } x_i \text{ vérifient un certain nombre de contraintes} \end{cases}$$

Z. Michalewicz [Mic92] a proposé d'utiliser un génotype où les gènes sont les composantes x_i et où les opérateurs effectuent des manipulations numériques. Nous donnons un exemple d'opérateur de recombinaison et un exemple de mutation.

Entrée de la table	1	2	3	4	5	6
A	BCEF	CEF	EF	EF	F	
B	ACDF					
C	ABD	AD				
D	BCE	CE	E			
E	ADF	ADF	ADF	AF		
F	ABE	AE	AE	AE	A	
Résultat	B	BC	BCD	BCDE	BCDEA	BCDEAF

FIG. 7 – *L'opérateur de recombinaison d'arcs. Outre la légende, les 6 premières lignes du tableau indiquent l'évolution au cours de l'algorithme de la carte des arcs (5 itérations). La dernière ligne montre le génotype de l'enfant en cours de construction. On suppose que les parents sont les chemins ABCDEF et BDCAEF.*

Recombinaison Étant donnés deux individus $x =< x_1, x_2, ..., x_\lambda >$ et $x' =< x'_1, x'_2, ..., x'_\lambda >$, on obtiendra v tel que :

$$v_i = a \times x_i + b \times x'_i$$

où a et b sont des constantes réelles.

Mutation Étant donné un individu $x =< x_1, x_2, ...x_\lambda >$, on obtient v tel que :

$$v_i = x_i \pm \Delta(t, \text{borne-sup} - x_i)$$

où $\Delta(t, y)$ donne une valeur dans l'intervalle $[0, y]$ telle que la probabilité d'être proche de 0 augmente avec t.

Sur un ensemble de problèmes numériques, la supériorité de cette technique a été montrée, non seulement sur l'AGC, mais aussi sur les méthodes plus classiques qui se laissent généralement piégées dans des optima locaux [JM91].

3.4 Des algorithmes évolutifs hybrides

Partant de l'idée d'AE, de nombreuses voies ont été explorées afin de les rendre plus efficaces de différentes manières : plus efficace dans leur exploration de l'espace de recherche, dans son exploitation, dans leur implantation, ... Pour cela, on a proposé de coupler l'AE de base avec d'autres algorithmes spécialisés, ce qui a donné naissance aux algorithmes évolutifs hybrides [Dav91]. Nous rangeons également dans cette catégorie des modèles parallèles d'AE ou des algorithmes permettant de trouver plusieurs optima à un problème à la fois en formant plusieurs classes ou *espèces* d'individus.

3.4.1 AE itérés

Une idée assez immédiate consiste, quand l'AE a convergé, à relancer son exécution à partir de zéro. Trivialement, si la probabilité de trouver l'optimum global lors d'une exécution est 0.1, dix exécutions augmenteront notablement cette probabilité. Basée sur ce principe, plusieurs techniques ont été développées ; nous en présentons brièvement deux : l'hyper-mutation cataclysmique et le nichage séquentiel.

L'hyper-mutation cataclysmique Proposée par L. Eshelman [Esh91], cette technique (CHC) consiste, quand l'algorithme a convergé, à effectuer une très forte mutation de tous les individus dans la population en conservant tel quel l'optimum trouvé. Les individus hyper-mutés et l'optimum forment alors la population initiale avec laquelle l'algorithme est relancé.

Le nichage séquentiel Afin d'éviter de converger plusieurs fois vers le même optimum, D. Beasley et ses collègues [BBM93] ont proposé de modifier la fonction objectif lorsque l'algorithme a convergé en « arrasant » l'optimum trouvé et en relançant l'algorithme ensuite, avec cette nouvelle fonction objectif.

3.4.2 Mutations lamarckiennes

La mutation a été présentée plus haut comme une action aveugle, c'est-à-dire agissant sur le génotype sans se pré-occuper du phénotype de l'individu, ou du sens des gènes (l'opérateur de recombinaison d'arcs, dans ce sens, n'est pas aveugle). Aussi, la mutation peut soit améliorer l'individu sur lequel elle agit, soit le rendre moins performant. Aussi, il est tentant de raffiner quelque peu le mécanisme en retenant le meilleur individu parmi l'individu original (avant mutation) et l'individu issu de la mutation :

$\{\chi\}$
$\chi' \leftarrow \text{muter}(\chi)$
si$\varphi(\chi') > \varphi(\chi)$alors$\chi \leftarrow \chi'$

En référence à Pierre Lamarck, ce type de mutation qui retient dans le génotype les mutations favorables est qualifiée de lamarckien. Une variation à ce type de mutation consiste à itérer le processus tant que l'individu obtenu améliore sa performance :

$\{\chi\}$
$\chi' \leftarrow \text{muter}(\chi)$
tant-que $\varphi(\chi') > \varphi(\chi)$ faire
$\chi \leftarrow \chi'$
$\chi' \leftarrow \text{muter}(\chi)$
fait

Ce type d'opérateurs a été appliqué avec succès sur divers problèmes et a apporté des accélérations considérables de la vitesse de convergence, ainsi qu'une amélioration de la qualité des solutions trouvées. Il est clair que ce type de mutations risque fort de précipiter l'algorithme dans un optimum local. Cependant, pour des problèmes difficiles de grande taille, l'essentiel est généralement de trouver un bon optimum, l'optimum global étant inconnu.

3.4.3 AE parallèles

Les AE étant des algorithmes assez lents lors de la résolution de problèmes de grande taille, leur parallélisation a naturellement été envisagée [Tan89]. Pour rendre l'implantation efficace, différents modèles d'AE parallèles ont été proposés :

- machines SIMD : algorithme cellulaire, un individu par processeur élémentaire, avec sélection par tournoi pour n'effectuer que des communications entre des processeurs voisins ;
- machines MIMD : modèle insulaire où chaque processeur exécute un AE sur une population qui lui est propre (un *dème*) et échange des individus avec les autres processeurs de temps en temps.

Ces implantations sont présentées dans [Tal95].

À un niveau plus conceptuel, partant de l'idée de modèle insulaire, il a été proposé de remplacer un, ou plusieurs, AE par un autre algorithme afin d'obtenir une véritable coopération entre algorithmes de recherche [PT95]. Ces travaux s'insèrent dans un courant d'idées où, de la coopération (évolution concurrente de plusieurs algorithmes) émerge naturellement des structures plus riches [KMD95]. Il est clair que des algorithmes coopérant ont un comportement fondamentalement différent des mêmes algorithmes considérés indépendemment les uns des autres. Cette voie est encore peu développée aujourd'hui mais nous semble fructueuse à poursuivre.

3.4.4 Nichage et spéciation

Pour que l'AE converge naturellement vers plusieurs points, K. De Jong [De 75] puis d'autres (*cf.* Goldberg et Richardson [GR87]) ont proposé de considérer la valeur du point de l'espace comme indicateur d'une ressource que les individus présents en ce point doivent se partager pour survivre. Aussi, plus le nombre d'individus en un point de l'espace croît, moins ils ont chacun pour se nourrir et donc, plus ils sont « tentés » de chercher de la nourriture ailleurs, les « incitant » donc à trouver d'autres bons points de l'espace, d'autres optima donc. Intuitivement, cela s'obtient très simplement en prenant :

$$\varphi(\chi) = \frac{\phi(\chi)}{|\{\chi', d(\chi, \chi') < \text{seuil}\}|}$$

où $d(\chi, \chi')$ est une distance indiquant le degré de similitude entre les individus χ et χ', c'est-à-dire si χ et χ' partagent la même ressource dans l'espace, se trouvent sur le même optimum donc. Ainsi, plus le dénominateur est grand, plus $\varphi(\chi)$ diminue poussant les individus à migrer vers d'autres sources de nourriture, d'autres « niches ». Un équilibre se crée de lui-même, les individus se regroupant dans des niches, créant des « espèces » différentes.

À l'aide de cette technique, plusieurs optima peuvent être découverts et conservés en même temps. La grande difficulté dans sa mise en œuvre consiste dans la définition de la distance d : en effet, des individus très proches au niveau de leur génotype peuvent différer complétement au niveau de leur position dans l'espace, donc ne pas partager la même ressource. De même, des individus ayant des performances proches peuvent très bien être différents au niveau de leur génotype et donc se trouver sur des optima différents également. Cependant, ces techniques ont permis d'obtenir de bons résultats sur des fonctions dites massivement multi-modales. On consultera la thèse de doctorat de S. Mahfoud pour un état de l'art sur la question [Mah95].

3.5 Autres instanciations du modèle

Nous présentons succinctement trois autres types d'AE, chacun représentant un champ de recherche important dans le domaine des AE.

3.5.1 Stratégies d'évolution

Les stratégies d'évolution (SE) ont été imaginées pour résoudre des problèmes d'optimisation numérique. Les individus représentent des points dans un

espace multi-dimensionnel (plusieurs centaines de dimensions parfois). Initialement, les SE n'agissent pas sur une population d'individus mais sur un seul individu. Par ailleurs, il n'y a qu'un opérateur de mutation. Cet opérateur agit en ajoutant un bruit gaussien sur chacune des composantes de l'individu, l'écart-type de ce bruit se réduisant au fur et à mesure du temps. Plus précisément, on a :

$$x_i'(t) = x_i(t) + \sigma z_i(t)$$

avec chacun des $z_i(t)$ qui suit la densité de probabilité :

$$p(z_i(t)) = \frac{1}{\sqrt{2\pi}} \exp -1/2 z_i^2(t)$$

Par la suite, un individu a donné naissance à plusieurs individus par mutation, parmi lesquels un survivant est sélectionné. Enfin, les SE sont devenus des techniques basées sur une population, un opérateur de recombinaison étant éventuellement utilisé. Deux stratégies de reproduction/sélection ont alors été proposées :

1. (μ, λ) où, avec μ individus (les parents), $\lambda > \mu$ enfants sont engendrés parmi lesquels les μ meilleurs sont sélectionnés pour composer la population de descendants ;

2. $(\mu + \lambda)$ où μ individus en engendrent $\lambda \leq 1$. La population de descendants est composée des μ meilleurs parmi les $\lambda + \mu$ individus.

Si ces schémas d'évolution sont classiques, la grande innovation des SE est l'auto-adaptation des paramètres ; la mutation est en effet paramétrée par l'écart-type du bruit. La grande idée a été de coder cet écart-type dans les gènes eux-mêmes et de lui permettre de varier et de s'adapter au cours du temps (codage dit « endogène ») ; ainsi, la valeur du gène et l'écart-type s'adaptent tous deux au cours du temps. Par ailleurs, des covariances entre chacune des composantes des individus correllent la mutation appliquée à chacune des composantes. Aussi, la valeur des gènes, leurs variances et les covariances évoluent au cours du temps.

T. Bäck [Bäc91] a proposé d'utiliser cette technique d'auto-adaptation pour les AG pour contrôler le taux d'application de la mutation des individus, chaque individu ayant son propre taux, ou chaque gène de chacun des individus ayant son propre taux.

On consultera [Bäc95] pour un état de l'art sur les SE.

3.5.2 Programmation évolutive

Après une période creuse suite aux travaux de L. Fogel, D. Fogel a remis au goût du jour la programmation évolutive qui partage de nombreuses similitudes avec les SE : les individus sont, a priori, des variables multi-dimensionnelles réelles et il n'y a pas d'opérateur de recombinaison. La sélection suit une stratégie de type $(\mu + \mu)$. Comme pour les stratégies d'évolution, la mutation est également contrôlée par un paramètre endogène (sa variance) dans la version dite « meta-EP » et la covariance a été introduite dans la version « Rmeta-EP ».

[Fog95] propose un état de l'art de la PE. [BRS93] fait le point sur les différences et similitudes entre ES et EP.

3.5.3 Programmation génétique

La caractéristique essentielle de la programmation génétique (PG) est que les génotypes sont des S-expressions Lisp (c'est-à-dire, des expressions Lisp parenthésées) ou, en toute généralité, des programmes. Le but est de synthétiser des fonctions Lisp (ou des programmes), devant effectuer un traitement donné. Clairement, la taille des individus n'est donc pas limitée. De nombreux opérateurs ont été proposés ; la définition de bons opérateurs qui soient efficaces est une tâche difficile : l'espace de recherche est virtuellement illimité.

Si l'utilisation de la PG pour résoudre des problèmes n'est pas toujours convaincante, on notera l'application qu'en a faite F. Gruau pour la synthèse de réseaux de neurones [Gru94] et qui donne des résultats remarquables. La PG est également utilisé pour résoudre des problèmes d'optimisation numérique (*cf.* [WS95]).

Pour le reste, nous préfèrons renvoyer aux livres de J. Koza, l'inventeur de la PG, [Koz92, Koz94] et à un recueil de travaux [Kin94].

3.6 Conclusion

Le modèle générique proposé n'a absolument pas l'ambition de définir une fois pour toute ce qu'est et sera dans l'avenir un algorithme évolutif. Des algorithmes que nous n'imaginons pas aujourd'hui pourraient fort bien mériter ce qualificatif dans l'avenir. Nous préférons ainsi bien préciser à nouveau que notre seul souhait, en proposant ce modèle, est de donner une définition de ce qu'est un AE aujourd'hui, étant donnés les algorithmes qui ont été étudiés.

On ne comprendra bien l'évolution des recherches dans le domaine des AEs que si l'on sait que SE, PE

et AG ont été définis indépendemment les uns des autres, sans même que les équipes se connaissent les unes les autres et connaissent l'existence des autres. C'est seulement à la fin des années 80, début des années 90 que les communautés ont commencé à se rencontrer, constatant qu'elles pouvaient mutuellement s'apporter des éléments ([SDB+93]).

4 L'évolution d'un AE

4.1 Phénoménologie

Nous nous intéressons ici à comprendre comment évolue l'AGC décrit plus haut. Nous nous attachons à véritablement comprendre ce qui se passe au cours de l'évolution plutôt qu'à le modéliser sous forme d'équations. Cette approche sera vue un peu plus loin. Nous préférons la laisser de côté pour l'instant car elle ne permet pas de comprendre qualitativement le fait que la population converge ; par ailleurs, la modélisation formelle des AG n'a été réalisée que pour l'AGC et son pouvoir de prédiction est encore faible, même pour cette version simple des AG. Allant dans le même sens dans notre argumentation, nous verrons plus loin que pour utiliser les AE comme des heuristiques efficaces vis-à-vis des autres heuristiques, la version canonique est inadéquate, ce qui réduit encore l'intérêt de la modélisation de son comportement.

Après création d'une population initiale (initialisation que nous supposerons pour l'instant aléatoire), le cycle d'un AG se compose donc d'une phase de clonage, suivie d'une phase d'application des opérateurs, terminée par une sélection des survivants. Observons chacune de ces étapes.

Lors du clonage, on forme une population temporaire formée de copies conformes des génotypes des parents. Chaque parent contribue (stochastiquement) un nombre de copies en accord avec sa performance ; donc, les meilleurs individus tendent à donner un plus grand nombre de copies que les plus mauvais.

Une caractéristique importante du crossover est qu'il ne fait qu'échanger les allèles entre les gènes de 2 individus. Aussi, si un gène possède le même allèle dans l'ensemble de la population des clones, ce gène conservera cet allèle après la recombinaison. Le crossover a donc pour rôle de mélanger les allèles de gènes différents et donc de fournir de nouvelles combinaisons d'allèles dans un même individu (cette combinaison d'allèles dans un individu entraînant la performance, donc le fitness de cet individu résultat de la recombinaison).

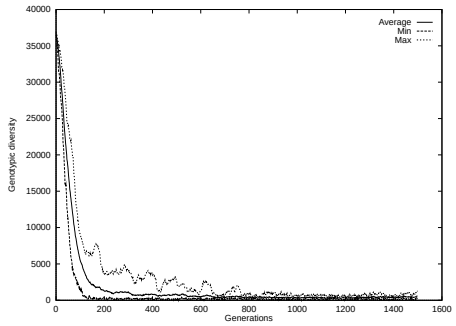


FIG. 9 – *Évolution de la diversité génotypique de la population d'un AGC au cours du temps. À chaque génération, on a représenté la moyenne sur 10 expériences et les valeurs minimale et maximale de la diversité. Cette évolution est typique quelle que soit la fonction objectif.*

La mutation modifie l'allèle de gènes pris au hasard dans la population. Notons que le taux de mutation est toujours très faible dans l'AGC (la probabilité pour un gène de subir une mutation est de l'ordre du millième).

Définissons la diversité génotypique Δ d'une population comme :

$$\Delta(\mathcal{P}_t) = \sum_{\{(\chi,\chi')\in\mathcal{P}_t\times\mathcal{P}_t\}} \delta(\chi,\chi')$$

où $\delta(x,y)$ est la distance de Hamming entre les deux génotypes χ et χ'. Si l'on mesure la diversité génotypique de la population au cours du temps, on observe que celle-ci diminue rapidement en début d'évolution pour stagner ensuite (*cf.* fig. 9).

Cette diminution s'explique par la phase de clonage qui duplique les bons individus, donc qui entraîne le "choix" de certains allèles pour certains gènes (les gènes qui sont les plus importants dans le génotype). Ces bons allèles se trouvent dupliqués dans la population à chaque phase de clonage, les mauvais n'étant pas ou très peu reproduits disparaissant petit à petit. Il faut bien noter que le crossover ne modifie pas la diversité et que la mutation, en moyenne, ne la modifie pas beaucoup.

La diversité, une fois largement diminuée, demeure relativement stable du fait de la présence de gènes dont les deux allèles subsistent dans la population car ayant peu d'influence sur le fitness des individus.

Le paramètre fondamental réglant la vitesse de décroissance de la diversité est la pression de sélection. Ainsi, si la pression est forte, c'est-à-dire que les bons individus produisent beaucoup plus de clones que les

AE	$\mathcal{G}$	Ω
AGC	$\{0,1\}^\lambda$	crossover et mutation binaires
SE	$\mathbb{R}^\lambda$	mutation : bruit gaussien
PE	$\mathbb{R}^\lambda$	mutation : bruit gaussien
PG	{S-exp}*	

FIG. 8 – *Ce tableau récapitule les traits caractéristiques des AE présentés ici, en fonction des paramètres du modèle d'AE générique.*

moins bons, la décroissance est très rapide, donc la convergence de l'algorithme très rapide. Si la pression est faible, c'est-à-dire que les bons individus produisent à peine plus de clones que les moins bons, la décroissance est beaucoup plus lente. Aussi, si la pression de sélection est forte, la convergence est rapide, minimisant donc la phase pendant laquelle l'algorithme produit de nouveaux génotypes par recombinaison, limitant donc la phase d'exploration.

La mutation n'étant pas nulle et la reproduction stochastique, lorsque l'algorithme a convergé vers un optimum, la probabilité que l'algorithme produise un nouvel optimum, meilleur que celui qu'il avait trouvé précédemment n'est pas nulle (mais extrêmement faible, il est vrai). Aussi, on peut prévoir que si on le laisse évoluer suffisamment longtemps, l'algorithme va converger vers un nouvel optimum. Si on n'utilise pas une stratégie élitiste, il est même possible que l'algorithme converge vers un optimum de qualité inférieure. Par référence aux théories d'évolution des espèces en biologie, ce phénomène est nommé équilibres ponctués : à une phase durant laquelle la population évolue très peu succède une phase d'évolution rapide menant vers une nouvelle espèce d'individus, où l'évolution marquera le pas, et ainsi de suite.

Notons que si on modifie le taux de mutation en l'augmentant fortement pendant quelques générations, une fois l'algorithme convergé, on augmente la probabilité de générer un nouvel individu vers lequel l'algorithme pourra converger. C'est le principe utilisé dans CHC (*cf.* section 3.4.1).

4.2 Analyse de l'AGC comme un processus stochastique

L'état à un instant t donné de l'AGC, c'est-à-dire la composition de sa population, dépend uniquement de son état à l'instant immédiatement antérieur $t-1$. Par ailleurs, le passage d'un état au suivant étant parfaitement déterminé par l'algorithme et ses différents paramètres, on peut, pour chaque état E à un instant t calculer la probabilité qu'il soit dans un état E' à l'instant $t+1$. Aussi, l'évolution peut être modélisée à l'aide de chaînes de Markov : on peut calculer la matrice de transition qui, pour l'AGC, est constante au cours du temps. On peut également calculer, pour un instant quelconque, la probabilité pour l'AGC d'être dans un certain état, donc la probabilité qu'il a de converger vers un point donné. Les valeurs propres de la matrice de transition donnent les points vers lesquels l'AGC peut converger et la probabilité qu'il le fasse. Citons les travaux de M. Vose (voir notamment [Vos93]) et ceux de G. Rudolph [Rud94] où l'auteur montre que l'AGC ne convergera jamais vers l'optimum global quel que soit la population initiale, le taux d'application des opérateurs et la fonction objectif, à moins que l'on utilise une stratégie de sélection élitiste. On notera également les travaux de R. Cerf [Cer94] qui considère l'AGC comme un système dynamique et étudie l'effet de perturbations sur celui-ci.

Même si théoriquement ces résultats sont une avancée dans l'analyse du comportement des AE, force est de constater qu'ils manquent quelque peu de capacité d'applications pratiques : il est impossible de calculer la matrice de transition pour un AGC ayant une population de taille normale (de l'ordre de la centaine d'individus et des génotypes possédant une centaine de gènes), la matrice étant d'ordre $|\mathcal{G}| = \mathcal{C}^{2^\lambda-1}_{\nu+2^\lambda-1}$. Il est donc tout aussi impossible de calculer les résultats intéressants que sont le temps que met l'AGC à converger vers un optimum, notamment vers l'optimum global.

La faiblesse de ces analyses est encore renforcée lorsque l'on sait que :

- la cardinalité dela population ν et la pression de sélection ρ et le taux d'application de la mutation doivent varier au cours du temps pour obtenir un comportement optimal de l'AE. Ceci implique que les coefficients de la matrice de transitions varient au cours du temps et que l'ordre de la matrice de transitions varie lui-aussi

- l'AGC ne peut être utilisé comme une bonne heuristique pour résoudre des problèmes pratiques

(bonne dans le sens où elle fonctionne aussi bien que les autres heuristiques, ce qui est le seul argument intéressant en optimisation). En optimisation, on utilise des AE qui n'ont pas grand chose à voir avec l'AGC si ce n'est le mécanisme de base de reproduction/sélection des meilleurs (*cf. infra* section 5.2).

4.3 Paysages de fitness

On comprend bien le fonctionnement des phases de clonage et de sélection. Par contre, le fonctionnement des opérateurs est loin d'être compris ; plus précisément, on n'a pas de réelle compréhension/vision de la manière dont l'espace de recherche est parcouru par l'action des opérateurs. Dans le cas de l'AGC, pour la mutation, on peut aisément calculer, partant d'un génotype χ la probabilité de le muter en un χ' donné. On obtient alors une distribution de probabilité que l'on peut facilement visualiser. L'action conjuguée de la mutation et de la sélection revient alors à effectuer une montée de gradient sur une population d'individus. Par contre, pour le crossover, la visualisation de cette même distribution n'est pas simple (c'est un euphémisme). Cela est encore compliqué par le fait que les opérateurs sont utilisés de concert. Par analogie avec la biologie, la notion de « paysage de fitness » est apparue. Initialement, un paysage peut-être vu comme la représentation graphique du fitness des individus par rapport à leur génotype (à cet égard, la figure 5 est un paysage de fitness). Cependant, pour que cette notion soit utilisable, il faut que la représentation soit en rapport avec la manière dont l'AE parcourt l'espace des génotypes : pour que le paysage est un sens réel, il faut que deux génotypes voisins dans le paysage soient effectivement proches pour l'AE, c'est-à-dire que l'AE puisse transformer l'un dans l'autre directement sinon cette intuition de distance va nous égarer. À cet égard, la figure 5 peut représenter un paysage de fitness pour un *hill-climber* très particulier. En général, pour un hill-climber travaillant sur des génotypes binaires de λ bits, chaque point possède λ voisins et le paysage doit être « visualisé » dans un espace à λ dimensions ! Si l'on se tourne vers un AE, la définition est encore compliquée : le codage et les opérateurs doivent être pris en compte pour définir un paysage pertinent. Pour l'instant, l'idée de paysage est très attirante mais force est de constater que leur mise en œuvre est encore à réaliser. T. Jones a travaillé cette notion dans sa thèse [Jon95], sans arriver à en donner une définition réllement utilisable pratiquement. Il a au moins eu le mérite d'essayer de clarifier cette notion de paysage dans le contexte des AE. On notera également les travaux du biologiste S. Kauffman [Kau89] et sa définition des paysages NK où on peut facilement régler la rugosité du paysage, c'est-à-dire le nombre de points dont dépend le fitness d'un point, d'où la notion de paysages corrélés signifiant que le fitness d'un point est en rapport avec celui de ses voisins (*cf.*. la notion de continuité en analyse).

4.4 Discussions de quelques notions

Un certain nombre de notions ont joué un grand rôle dans la compréhension du fonctionnement des AG (théorème des schémas, parallélisme implicite, hypothèse des briques de base). Celles-ci ont été récemment remises en question et ont perdu de leur importance. Nous pensons que leur présentation est nécessaire car elle permet de mieux saisir le fonctionnement des AG en comprenant pourquoi il ne faut pas les prendre au pied de la lettre.

L'explication classique du fonctionnement des AG (voir à ce sujet le livre de D. Goldberg, [Gol89, p. 40–45]) suppose que par le jeu du crossover et de la sélection, l'algorithme teste des combinaisons d'allèles et sélectionne les meilleures. Ces combinaisons d'allèles bénéfiques portent le nom de « briques de base ». Par combinaison de ces briques de base, l'algorithme synthétise des individus de qualité croissante. Cette explication porte le nom « d'hypothèse des briques de base ».

D. Goldberg [Gol87] a proposé la notion de « problème décevant » pour confirmer expérimentalement cette hypothèse qui a ensuite été étudiée plus avant par D. Whitley et ses collègues [Whi91]. On a alors pensé pouvoir caractériser de cette manière la notion de difficulté pour un AGC de résoudre un problème donné : un problème décevant est un problème difficile ; un problème non décevant est facile. Cependant, des observations attentives de l'hypothèses des briques de base et la déception a permis à J. Grefenstette [Gre93] de proposer une amélioration de l'AGC pour laquelle les problèmes décevants sont faciles à résoudre et un problème non décevant très difficile à résoudre. Par ailleurs, voulant confirmer l'hypothèse des briques de base, M. Mitchell *et al.* [MFH91] ont étudié des fonctions dites « voie royale » qui ont clairement infirmé l'hypothèse des briques de base en montrant que le crossover ne coupe pas forcément là où il le faut (et après tout, pour quelle raison une opération aveugle comme le crossover de l'AGC couperait-il les génotypes aux bons endroits ?) et que des allèles clandestins (qualifiés d'« auto-stoppeurs » par référence à la génétique) non favorables sont transmis lors de la recombinaison. Par ailleurs, sur le problème consis-

tant à maximiser le nombre de 1 dans un génotype binaire, soit disant le problème le plus simple qui soit pour l'AGC, nous avons obtenu des résultats expérimentaux où, sur des génotypes de plus en plus long, les meilleurs génotypes contiennent de plus en plus d'allèles valant 0 [Pre95].

Sur un autre plan, un problème dans l'utilisation des AE, hormis la spécification de la forme des individus et de l'écriture de la fonction objective, concerne la valuation des paramètres tels les taux d'application des opérateurs crossover et mutation, le choix de la taille de la population et de la stratégie de sélection. Des auteurs ont pensé trouver des paramètres robustes. Ainsi, [De 80], [Gre86], [SCED89] ont effectué des études systématiques recherchant les paramètres optimaux pour résoudre des problèmes numériques (et symboliques pour [SCED89]). Depuis, [HB91] ont montré que de tels paramètres ne peuvent pas exister quelle que soit la fonction objectif.

5 Applications des AE

5.1 Optimisation numérique

Si les AG ont initialement été proposés pour étudier le phénomène d'adaptation, les stratégies d'évolution ont immédiatement eu pour vocation de résoudre des problèmes d'optimisation numérique. Aujourd'hui, ces algorithmes ainsi que les AG proposés par Z. Michalewicz [Mic92] ont permis de résoudre des problèmes consistant à optimiser plusieurs centaines de variables réelles avec succès, là où aucune autre méthode ne donne satisfaction. Parmi d'autres, nous citerons des applications dans le calcul de la forme optimale de l'electrode d'une bougie de démarrage, en interférométrie pour le placement de radiotélescopes, en optique, ...

5.2 Optimisation combinatoire

Dans un autre domaine important de l'optimisation, les AG fournissent aujourd'hui des heuristiques pour la résolution de problèmes NP-complets ou NP-durs. Le problème de la représentation des solutions et des opérateurs s'est posé avec insistance ici, l'AGC étant mal adapté à la résolution de ce type de problèmes. On a posé plus haut (*cf.* 3.3.1) concrétement le problème dans le cas du voyageur de commerce (TSP).

Les AE n'ont d'intérêt effectif que s'ils se posent comme une heuristique valable face aux autres heuristiques, que ce soit des heuristiques spécifiques à un problème donné, ou des heuristiques générales comme les algorithmes de recherche locale, le TABU, le recuit simulé, ... Si dans le monde clos des heuristiques générales les AE donnent de très bons résultats, force est de constater que les AE font généralement grise mine face aux heuristiques spécialisées. Pour obtenir de bons résultats, il faut hybrider les AE avec d'autres algorithmes [FF95a]. Dans le cas du TSP, on a par exemple utilisé l'heuristique *2-opt* [LK73] comme opérateur de mutation lamarckienne [Gre87b, MGSK88]. Différents auteurs ont proposé d'hybrider les AE avec la méthode TABU pour le problème d'affectation quadratique [FF94], le partitionnement de graphes [TMS94], pour le problème d'ordonnancement de processus (job-shop-scheduling), avec des méthodes de recherche locale [DPT95b] ou avec des heuristiques développées spécifiquement pour, notamment, le partitionnement d'ensembles [Lev94, CB95] ou la construction d'emplois du temps [RCF94], le coloriage de graphes [FF95b].

Un autre paramètre important lors de l'utilisation d'AE pour optimiser des fonctions est la population initiale ; différents auteurs ont proposé d'engendrer la population initiale à l'aide d'une heuristique également. Il est encore possible, à la suite de l'exécution de l'AE d'appliquer une heuristique à la population obtenue. On consultera [PT95] pour un tour d'horizon des différents modes d'hybridation mis en œuvre avec les AE. Parmi les problèmes actuellement étudiés, la satisfaction de contraintes est également traitée par AE ; on consultera [ERR94] pour un état de l'art récent sur la question.

5.3 Divers problèmes

Parmi les applications à vocation plus directement industrielle, on notera le routage de robots dans des environnements dynamiques [ATBM92], la conception de profil d'avions, la synthèse de circuits intégrés, l'optimisation de placement ou de routage d'objets, la prédiction de la structure ternaire de protéines, ... On consultera [Dav91] pour une présentation de nombreuses applications de ce type ou les actes des conférences ICGA qui consacre toujours une large part aux applications concrètes des AE.

Enfin, les AE ont des applications en intelligence artificielle et en modélisation de certains phénomènes naturels. Nous discutons ces aspects dans la dernière section de ce rapport.

5.4 Discussion

D'aucuns, enthousiastes à propos des AE, ont pensé que l'ultime méta-heuristique avait été découverte,

qu'elle allait rapidement supplanter tous les autres algorithmes. Il convient ici de faire le point le plus honnêtement possible à leur encontre : les AE sont une très intéressante méta-heuristique, ... au même titre que d'autres. Il est clair que les AE donnent d'excellents résultats sur des problèmes difficiles d'optimisation numérique et sur des problèmes industriels pour lesquels on ne dispose pas de bons algorithmes. Il est tout aussi clair que sur des problèmes étudiés depuis longtemps avec des méthodes traditionnelles et pour lesquels des heuristiques ont été développées, les AE seuls ne fournissent pas les meilleurs résultats connus. Sur certains problèmes, des résultats très proches de l'optimum ont été obtenus avec des AE hybrides (voyageur de commerce notamment, voir à ce sujet [Esh91] ou [MGSK88]). [DPT95a] présente, dans le cadre du problème de job-shop-scheduling, une comparaison de plusieurs méta-heuristiques stochastiques où les AE seuls donnent des résultats assez décevants, leur hybridation améliorant nettement leurs performances.

5.5 Théorème NFL

Il faut ici mentionner le théorème NFL [9] [WM95, RS95] qui, sous certaines hypothèses, montrent que, dans l'espace de toutes les fonctions objectifs possibles, toutes les méthodes stochastiques se comportent en moyenne de la même manière. C'est-à-dire que s'il existe des problèmes difficiles pour une heuristique $\mathcal{H}$, faciles pour une autre heuristique $\mathcal{H}'$, il existe autant d'autres problèmes qui seront faciles pour $\mathcal{H}$ et difficiles pour $\mathcal{H}'$.

Nous faisons ici nôtres, et adaptons, les remarques de G. Weisbuch [Wei91, p. 173] faites dans le domaine des réseaux de neurones :

> « [...]
>
> « – il n'existe pas d'algorithme universel qui soit optimal en dehors de toute application
>
> « – pour un problème particulier, un algorithme classique peut être plus efficace qu'un AE [...]
>
> « – les algorithmes canoniques [...] sont rarement adaptés à la résolution de problèmes concrets de taille réelle [...]
>
> [...] »

9. NFL pour *no-free-lunch*

6 D'autres algorithmes évolutifs

Nous n'avons pour l'instant parlé des AE que comme des algorithmes d'optimisation de fonctions. Cependant, ce serait une erreur de croire que les AE se réduisent à cela. J. Holland [Hol75] et K. De Jong [De 93] insistent sur le fait que l'idée originale consistait à étudier les processus d'adaptation [10]. Ainsi, les AE sont aujourd'hui au cœur de travaux dont le but est avant tout d'étudier ces processus d'adaptation et d'évolution dans des environnements artificiels assez simples. On peut citer trois domaines importants :

- l'apprentissage avec les *systèmes de classifieurs*,
- l'étude de systèmes où évoluent des populations d'individus d'espèces différentes,
- l'étude de la vie artificielle.

Ces trois types de systèmes sont essentiellement basés sur la simulation de plusieurs espèces d'individus qui co-évoluent et se co-adaptent. La notion de fonction objective disparaît. La performance d'un individu dépend de sa capacité à interagir avec les autres individus. Le principe de la reproduction des individus reprend les techniques des algorithmes génétiques (duplication et recombinaison des génotypes, mutation).

Nous reprenons très brièvement ces trois thèmes dans la suite.

6.1 Les systèmes de classifieurs

L'idée de système de classifieurs remonte aux premiers travaux de Holland dans les années 60. Le premier système de classifieurs, CS-1 – Cognitive System-1, est présenté dans [HR78]. Depuis, leur étude s'est énormément développée. [BGH89] présente un état de l'art récent et contient une liste des applications des systèmes de classifieurs. Ces systèmes sont basés sur un fonctionnement génétique, mais sont d'une complexité et d'une richesse bien plus grande. En deux mots, on peut les caricaturer comme des "systèmes experts génétiques". Cependant, nous précisons immédiatement que ces systèmes ne sont absolument pas nés dans la mouvance des travaux concernant les systèmes experts et n'ont pas été conçus en tant que tels. Ce sont des systèmes censés s'adapter par apprentissage à un environnement dans lequel ils évoluent. Ils reçoivent des inputs de la part de l'environnement et réagissent en fournissant des outputs. Ces

10. Holland a publié son livre sur l'étude des processus d'adaptation avant que De Jong ne publie sa thèse qui est une application de ces techniques d'adaptation au problème de l'optimisation d'une fonction.

outputs sont les conséquences de règles déclenchées directement ou indirectement par les inputs.

Nous pouvons résumer leur principe avec les points suivants (cf. fig 10) :

- les individus sont des chaînes de symboles pris dans $\{0, 1, \#\}$. Ils représentent des règles avec une partie condition composée de prémisses et une partie conclusion (une seule conclusion par règle). Ces règles sont nommées des *classifieurs* ;
- prémisses et conclusions de règles sont composées du même nombre de symboles. Les prémisses et les conclusions sont codées dans $\{0, 1, \#\}^l$;
- une structure "à la" tableau noir, dénommée *liste des messages*, contient des messages codés dans $\{0, 1, \#\}^l$;
- un message provient d'un capteur (input) ou de la conclusion d'une règle dont les conditions ont été vérifiées ;
- un message peut déclencher l'activation d'une règle ou l'activité d'un affecteur (output) ;
- un mécanisme génétique génère périodiquement de nouvelles règles pour que le système évolue, s'adapte ;
- un mécanisme renforce (et donc protège de la destruction par les opérateurs génétiques) les règles ayant démontré une certaine utilité au cours de l'évolution du système. Les règles les moins fortes ont de grandes chances de disparaître.

Notons que l'environnement dans lequel évolue le système de classifieurs peut changer au cours de l'évolution du système. Celui-ci s'adapte alors remettant éventuellement en cause des règles. D'une manière générale, les systèmes de classifieurs sont capables d'induire de nouvelles règles par généralisation à partir d'exemples [HHNT86]. Les systèmes de classifieurs sont utilisés pour résoudre des problèmes réels, notamment dans l'industrie.

6.2 Les systèmes écologiques

Poursuivant l'étude de métaphores artificielles de la nature, les chercheurs simulent des modèles très simples inspirés de phénomènes biologiques. Ces simulations se basent sur la modélisation d'agents et de leurs interactions. Plusieurs espèces d'individus co-évoluent. L'un des points étudiés concerne l'étude de l'émergence de comportements sociaux en partant de comportements individuels et a priori individualistes. Comme D. Hostadter [Hof85] le fait remarquer, chaque fourmi d'une fourmilière a un comportement a priori aléatoire et individuel. Or, l'ensemble des fourmis réalise des actions utiles globalement à leur société. Certains travaux s'inspirent ainsi de modèles du comportement d'insectes [11]. Une caractéristique fondamentale des systèmes écologiques concerne la prise en compte d'une notion d'espace physique. Les individus sont situés dans l'espace et s'y déplacent. Les interactions entre les individus dépendent de la distance « physique » entre les individus. Sur un autre plan, les individus peuvent être constitués par des assemblées de cellules et non plus par de simples individus. Le rôle de ces cellules et leur organisation sont décrits dans l'individu.

Nous décrivons en quelques mots le simulateur Echo [Hol92, p. 186 :195], [FJ94]. Dans Echo, les agents sont mono-cellulaires. Ils se développent dans un espace plan discret. Des « fontaines de ressources » nécessaires à leur survie et leur reproduction, enjeu de leur rivalité, sont réparties aléatoirement dans l'espace. Chaque agent se reproduit, combat d'autres agents ou fait du commerce avec d'autres agents. Les stratégies suivies pour ces différentes actions sont encodées dans le génotype de l'agent. Ce dernier est transmis lors de la reproduction qui peut consister en un simple clonage ou en une reproduction « sexuée » où interviennent deux agents. Leurs individus sont alors recombinés comme dans les algorithmes génétiques pour fournir l'individu du descendant.

Nous noterons également le développement de travaux simulant des modèles simples du système immunitaire des organismes vivants [SFP93] [FJSP93]. Le système immunitaire est d'une formidable adaptabilité : il est incessamment soumis aux attaques de nouveaux agents pathogènes et doit donc continuellement engendrer de nouveaux anticorps pour défendre l'organisme auquel ils appartiennent. Les simulations se fondent sur deux populations différentes qui interagissent : les antigènes et les anticorps. Tous ces individus sont des chaînes de bits. Un anticorps détruit

11. Il faut clairement distinguer la différence entre les systèmes écologiques présentés ici et les systèmes exhibant des comportements grégaires – *flocking behavior* – ou d'intelligence collective – *swarm intelligence*. Contrairement à ceux-ci, les individus des systèmes écologiques ont leur comportement régulé plus ou moins par un génotype. Par ailleurs, il y a reproduction des individus avec, à cette occasion, mélange des matériels génétiques des deux parents. Aussi, il y a évolution des génotypes de la population au cours du temps. Dans les systèmes cités ici, il n'y a pas cet aspect évolution génétique, même s'il peut y avoir évolution du comportement des individus par apprentissage. On pourra consulter [DTB91] pour une présentation de l'intelligence collective.

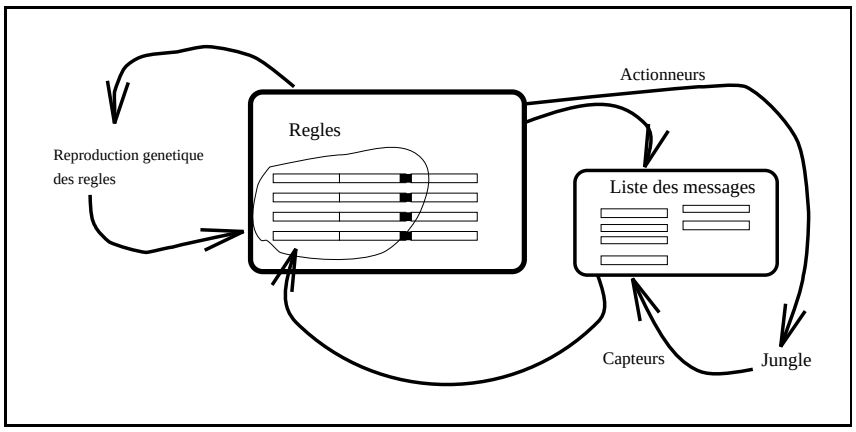


FIG. 10 – *Un système de classifieurs*

un antigène si les deux agents ont des représentations binaires complémentaires proches en utilisant la distance de Hamming. Lorsque le nombre d'anticorps est suffisant, ceux-ci se spécialisent, par classes ou individuellement, pour combattre chaque antigène. Par contre, si le nombre d'anticorps est insuffisant par rapport au nombre d'antigènes, des anticorps généralistes, capables de combattre plusieurs types d'antigènes, se développent.

6.3 La vie artificielle

Le nom de cette discipline fera sourire, irritera ou remplira d'enthousiasme. Nous ne prendrons pas partie et nous bornerons à indiquer en quelques lignes les objectifs de cette discipline encore très jeune. Dans un article récent, T. Ray définit la vie artificielle de la manière suivante [Ray94b] :

> « Discipline tentant de comprendre la biologie en réalisant des phénomènes biologiques à partir de composants artificiels, plutôt qu'en décomposant la vie naturelle en ses parties constitutives. »

La vie artificielle tente ainsi de « simuler la vie » dans divers media : le hardware, le software et le « wetware ». Concernant la simulation par logiciel, la vie artificielle ne consiste pas vraiment à simuler des agents co-évoluant dans la mémoire d'un ordinateur. Il s'agit plutôt de mettre en œuvre un « environnement logiciel » dans lequel des « organismes » seraient à même de « vivre », c'est-à-dire se développer, se reproduire, ... mourir. Ainsi, [Ray94b] tente d'imaginer ce que pourraient être ces organismes, ce milieu, ce que cela peut signifier de « vivre » dans ce contexte et les contraintes liées au média où la « vie » se déroule. T. Ray a développé un système de simulation de vie artificielle, Tierra, où les organismes sont des programmes écrits dans un langage interprété de type assembleur. [Ray94a] présente quelques expériences réalisées avec Tierra.

Concernant des aspects orientés hardware, R. Brooks travaille sur des robots dont le comportement n'est pas guidé par un programme monolithique. Au contraire, il tente d'assembler des robots à partir de membres possédant chacun des capacités d'adaptation et qui co-évoluent. L'idée de R. Brooks est que l'intelligence ne peut être désincarnée et qu'il faut donc construire des êtres physiques capables d'adaptation (cf. [Bro91c, Bro91a, Bro91b]) pour pouvoir obtenir des comportements intelligents, capables de s'adapter à l'environnement et affronter des situations imprévues. Ce courant porte le nom de « comportement adaptatif » (*adaptive behavior*) et nous semble conceptuellement très important, en rupture notable avec l'approche IA classique et en phase avec des travaux dans différentes disciplines comme les travaux de G. Edelman en neuro-physiologie [Ede92] ou de J.E.R. Staddon en psychologie [Sta83]. On pourra se référer à [MG94] pour un état de l'art récent sur la question.

Au niveau wetware, le but est d'engendrer des molécules par évolution, mutation, sélection, ayant les propriétés voulues ([Joy93]).

Nous n'irons pas plus avant dans cette description de la vie artificielle. Nous ne pourrions que trahir les idées qui sous-tendent ces travaux si nous tentions de les présenter en quelques lignes ou même quelques pages. Les projets sont ambitieux et font aisément rêver : il y a 30 ans, on rêvait de reproduire l'intelligence. Aujourd'hui, on s'attaque à la reproduction artificielle de la vie, l'intelligence n'en étant plus considérée que comme un épi-phénomène.

7 Conclusion

Les algorithmes évolutifs forment un domaine jeune en pleine expansion (voir la multiplication des conférences spécialisées ou des sessions qui leur sont dédiées depuis 10 ans). Cet engouement repose sur leur capacité à fournir des heuristiques capables de résoudre des problèmes très complexes. En marge de cette activité, une réflexion de fond s'engage dans le but d'évaluer leurs capacités et limites objectivement. En regardant autour d'elle, la communauté AE est en train de fédérer ses efforts, de reprendre des idées classiques dans d'autres domaines et, réciproquement, d'apporter ses connaissances à d'autres branches (*cf.* les méta-heuristiques hybrides). Bien que peu influent, il faut noter que l'étude de l'adaptation participe à un mouvement de fond allant des sciences humaines (philosophie, psychologie, ...) aux sciences de l'ingénieur où le phénomène d'adaptation est vu comme une altenative aux approches classiques (cognitivisme en philosophie et psychologie, symbolisme, IA classique, ...).

7.1 Lectures conseillées

Outre les références citées dans le rapport, on consultera les actes des conférences bi-annuelles ICGA [Gre85, Gre87a, Sch89, BB91, For93, Esh95] qui ont lieu aux États-Unis en alternance avec la version européenne PPSN [SM91, MM92a, DSM94a], les nouvelles conférences IEEE [iee94] les workshops FOGA qui se concentrent sur les aspects théoriques [Raw91a, Whi93, WV94], les conférences sur la programmation évolutive [FA93, Fog94b], les conférences américaines sur la vie artificielle ALIFE [FLRT91, Lan93, BM94] et européennes [VB92], conférence AISB aux Royaumes-Unis [Fog94a], aux conférences Évolution Artificielle en France [MSR94, MSR95]. On pourra également s'abonner à la revue électronique `GA-Digest` (`GA-List-Request@aic.nrl.navy.mil`) et aux revues papiers « Evolutionary Computation » et « Artificial Life », MIT Press.

Sur un autre plan, celui de l'étude de l'évolution et de l'adaptation biologiques, on lira avec intérêt [Lam94] qui fait un état de nos connaissances au jour d'aujourd'hui, [DL90] et [May93] offrant une perspective historique et montrant l'évolution des idées depuis le XIX^{e} siècle.

Références

[ATBM92] J. M. Ahuactzin, E.G. Talbi, P. Bessière, and E. Mazer. Using genetic algorithms for robot motion planning. In B. Neumann, editor, *2nd Europeen Conference on Artificial Intelligence ECAI92*, pages 671–675, Vienna, Austria, Aug 1992. John Wiley and Sons.

[Bäc91] Thomas Bäck. Self-adaptation in genetic algorithms. In *[VB92]*, pages 263–271. MIT Press, Cambridge, MA, USA, 1991.

[Bäc95] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1995.

[Bak85] James E. Baker. Adaptive selection methods for genetic algorithms. In *[Gre85]*, pages 101–111, 1985.

[BB91] Richard K. Belew and Lashon B. Booker, editors. *Proc. of the Fourth International Conference on Genetic Algorithms*, La Jolla, CA, USA, July 1991. Morgan Kaufmann, San Mateo, CA, USA.

[BBM93] David Beasley, David R. Bull, and Ralph R. Martin. A sequential niche technique for multimodal function optimization. *Evolutionary Computation*, 1(2):101–125, 1993.

[BGH89] L.B. Booker, D.E. Goldberg, and J.H. Holland. Classifier systems and genetic algorithms. *Artificial Intelligence*, 40:235–283, 1989.

[BM94] Rodney A. Brooks and Pattie Maes, editors. *Proc. of the Artificial Life IV*. MIT Press, Cambridge, MA, USA, 1994.

[Bro91a] Rodney Brooks. Artificial life and real robots. In *[VB92]*, pages 3–10, 1991.

[Bro91b] Rodney Brooks. Elephants don't play chess. In *[Mae91]*, 1991.

[Bro91c] Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–160, January 1991.

[BRS93] Thomas Bäck, Günter Rudolph, and Hans-Paul Schwefel. Evolutionary programming and evolution strategies: Similarities and differences. In *[FA93]*, pages 11–22, 1993.

[CB95] P. C. Chu and J.E. Beasly. A genetic algorithm for the set partioning problem. Technical report, The Management School, Imperial College, London, April 1995.

[Cer94] Raphaël Cerf. *Une théorie asymptotique des algorithmes génétiques*. PhD thesis, Université de Montpellier II, March 1994.

[Dav87] Lawrence Davis, editor. *Genetic Algorithms and Simulated Annealing*. Research Notes in Artificial Intelligence. Morgan Kaufmann, San Mateo, CA, USA, 1987. ISBN: 0-934613-44-3.

[Dav91] Lawrence Davis, editor. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, USA, 1991. ISBN: 0-442-00173-8.

[De 75] Kenneth A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975. *Dissertation Abstracts International* 36(10), 5140B (University Microfilms 76-9381).

[De 80] Kenneth A. De Jong. Adaptive system design: A genetic approach. *IEEE Transactions on Systems, Man and Cybernetics*, 10(9):566–574, September 1980.

[De 93] Kenneth A. De Jong. Genetic algorithms are NOT function optimizers. In *[Whi93]*, pages 5–17, 1993.

[DL90] Jean-Marc Drouin and Charles Lenay, editors. *Théories de l'évolution*. Presses Pocket, 1990. ISBN : 2-226-03486-3.

[DPT95a] D. Duvivier, P. Preux, and E-G. Talbi. Stochastic algorithms for optimization and application to the job-shop-scheduling problem. Technical Report LIL-95-5, Laboratoire d'Informatique du Littoral, Laboratoire d'Informatique du Littoral, Calais, September 1995. (submitted).

[DPT95b] D. Duvivier, Ph. Preux, and E-G. Talbi. Parallel genetic algorithms for optimization and application to NP-complete problem solving. In *Combinatoire et Informatique*, pages 41–43, July 1995.

[DS89] Kenneth A. De Jong and William Spears. Using genetic algorithms to solve NP-complete problems. In *[Sch89]*, pages 124–132, 1989.

[DSM94a] Yuval Davidor, H-P Schwefel, and R. Maenner, editors. *Proc. of the Third Conf. on Parallel Problem Solving in Nature*, Jerusalem, Israel, 1994. Springer-Verlag, Berlin. Lecture Notes in Computer Science, vol. 866.

[DSM94b] Yuval Davidor, H-P Schwefel, and R. Maenner, editors. *Proc. of the Third Conf. on Parallel Problem Solving in Nature*, Jerusalem, Israel, 1994. Springer-Verlag, Berlin. Lecture Notes in Computer Science, vol. 866.

[DTB91] Jean-Louis Deneubourg, Guy Theraulaz, and Ralph Beckers. Swarm-made architectures. In *[VB92]*, pages 123–133, 1991.

[Ede92] Gerald Edelman. *Biologie de la conscience*. Odile Jacob, 1992. ISBN : 2-02-021889-5.

[ERR94] Ágoston Eiben, Paul-Erik Raué, and Zsófia Ruttkay. GA-easy and GA-hard constraint satisfaction problems. In *Constraint Processing*. 1994. paper present at the workshop on Constraint Processing, 11th ECAI Conf, 1994.

[Esh91] Larry J. Eshelman. The CHC adaptive search algorithm: how to have safe search when engaging in non-traditional genetic recombination. In *[Raw91b]*, pages 265–283, 1991.

[Esh95] Larry J. Eshelman, editor. *Proc. of the Sixth International Conference on Genetic Algorithms*, Pittsburgh, PA, USA, July 1995. Morgan Kaufmann, San Mateo, CA, USA.

[FA93] David B. Fogel and Wirt Atmar, editors. *Proc. of the 2^{nd} Annual Conf. on Evolutionary Programming*, San Diego, CA, February 1993. Evolutionary Programming Society.

[FF94] Charles C. Fleurent and Jacques A. Ferland. Genetic hybrids for the quadratic assignment problem. *DIMACS Series in Discrete Mathematics*, 16:173–188, 1994.

[FF95a] Charles C. Fleurent and Jacques A. Ferland. Algorithmes génétiques hybrides pour l'optimisation combinatoire. *RAIRO*, 1995.

[FF95b] Charles C. Fleurent and Jacques A. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operation Research*, 1995.

[FJ94] Stephanie Forrest and Terry Jones. Modeling complex adaptive systems with Echo: Mechanisms of adaptation. In R.J. Stonier and X.H. Yu, editors, *Complex Systems: Mechanisms of Adaptation*, pages 3–21. IOS Press, 1994.

[FJSP93] Stephanie Forrest, Brenda Javornik, Robert E. Smith, and Alan S. Parelson. Using genetic algorithms to explore pattern recognition in the immune system. *Evolutionary Computation*, 1, 1993.

[FLRT91] D. Farmer, C. Langton, S. Rasmussen, and C. Taylor, editors. *Artificial Life II*. Addison-Wesley, 1991.

[Fog94a] Terence C. Fogarty, editor. *Proc. Evolutionary Computing, AISB Workshop*, Leeds, UK, April 1994. Springer-Verlag, LNCS 865.

[Fog94b] David B. Fogel, editor. *Proc. of the 3^{rd} Annual Conf. on Evolutionary Programming*, San Diego, CA, February 1994. Evolutionary Programming Society.

[Fog95] David Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, 1995.

[For93] Stephanie Forrest, editor. *Proc. of the Fifth International Conference on Genetic Algorithms*, Urbana-Champaign, IL, USA, July 1993. Morgan Kaufmann, San Mateo, CA, USA.

[FOW66] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence Through Simulated Adaptation*. Wiley, New York, 1966.

[Glo89] F. Glover. Tabu search — part I. *ORSA Journal of Computing*, 1(3):190–206, 1989.

[Gol87] David Goldberg. Simple genetic algorithms and the minimal deceptive problem. In *[Dav87]*, pages 74–88. 1987.

[Gol89] David Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[GR87] David Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *[Gre87a]*, pages 41–49, 1987.

[Gre85] J. J. Grefenstette, editor. *Proc. of the First International Conference on Genetic Algorithms*, Pittsburgh, 1985. Lawrence Erlbaum Associates: Hillsdale, New-Jersey.

[Gre86] John Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, 16(1):122–128, January 1986.

[Gre87a] J.J. Grefenstette, editor. *Proc. of the Second International Conference on Genetic Algorithms*, MIT, Cambridge, MA, USA, 1987. Lawrence Erlbaum Associates: Hillsdale, New-Jersey.

[Gre87b] John J. Grefenstette. Incorporating problem specific knowledge into genetic algorithms. In *[Dav87]*, pages 42–60. 1987.

[Gre90] John J. Grefenstette. *A User's guide to Genesis Version 5.0*, October 1990. The *genesis* package is available via anonymous ftp on `ftp.aic.nrl.navy.mil:pub/galist/src/ga/genesis.tar.Z`.

[Gre93] John J. Grefenstette. Deception considered harmful. In *[Whi93]*, pages 75–91, 1993.

[Gru94] Frédéric Gruau. Genetic microprogramming of neural networks. In *[Kin94]*. 1994.

[HB91] William E. Hart and Richard K. Belew. Optimizing an arbitrary function is hard for the genetic algorithm. In *[BB91]*, pages 190–195, 1991.

[HHNT86] John H. Holland, K. J. Holyoak, R. E. Nisbett, and P. R. Thagard. *Induction: Processes of inference, learning, and discovery*. MIT Press, Cambridge, MA, USA, 1986.

[Hof85] Douglas Hofstadter. *Gödel, Escher et Bach; les brins d'une guirlande éternelle*. InterEditions, Paris, 1985. ISBN: 2-7296-0040-X.

[Hol75] John H. Holland. *Adaptation in Natural and Artificial Systems*. Michigan Press University, Ann Arbor, MI, 1975.

[Hol92] John H. Holland. *Adaptation in Natural and Artificial Systems*. A Bradford Book. MIT Press, Cambridge, MA, USA, second edition, 1992. ISBN: 0-262-58111-6.

[HR78] J. H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern Directed Inference Systems*, pages 313–329. Academic Press, 1978.

[iee94] *Proc. of the First IEEE Conf. on Evolutionary Computation*. IEEE, 1994.

[JM91] Cezary Z. Janikov and Zbigniew Michalewicz. An experimental comparison of binary and floating point representations in genetic algorithms. In *[BB91]*, pages 31–36, 1991.

[Jon95] Terry Jones. *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, University of New Mexico, Santa Fe, NM, USA, May 1995.

[Joy93] Gerald F. Joyce. L'évolution moléculaire dirigée. *Pour la Science*, 184:72–79, February 1993.

[Kau89] Stuart Kauffman. Adaptation on rugged fitness landscapes. In D.L. Stein, editor, *Lectures in the Science of Complexity*, pages 527–618. Addison-Wesley, 1989.

[Kin94] Kim Kinnear, editor. *Advances in Genetic Programming*. MIT Press, Cambridge, MA, USA, 1994.

[KJV83] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, May 1983.

[KMD95] Stuart Kaufman, William G. Macready, and Emily Dickinson. Divide to conquer: Optimization reconsidered. Technical report, Santa Fe Institute, April 1995.

[Koz92] John R. Koza. *Genetic Programming*. A Bradford Book. MIT Press, Cambridge, MA, USA, 1992. ISBN: 0-262-11170-5.

[Koz94] John R. Koza. *Genetic Programming II*. A Bradford Book. MIT Press, Cambridge, MA, USA, 1994.

[Lam94] Maxime Lamotte. *Théorie moderne de l'évolution*. Hachette, 1994. ISBN : 2-01-235090-9.

[Lan93] C. Langton, editor. *Artificial Life III*. Addison-Wesley, 1993.

[Lev94] David Levine. *Parallel Genetic Algorithms for the Set Partioning Problem*. PhD thesis, Argonne National Laboratory, Maths and Computer Science Divivsion, May 1994. report ANL 94/23.

[LK73] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operation Research*, 11:498–516, 1973.

[Mae91] Pattie Maes, editor. *Designing autonomous agents – Theory and Practice from Biology to Engineering and Back*. A Bradford Book. MIT Press, Cambridge, MA, USA, 1991. ISBN: 0-262-63135-0, special issue of *Robotics and autonomous agents*.

[Mah95] Samir W. Mahfoud. *Niching Methods for Genetic Algorithms*. PhD thesis, University of Illinois, 1995.

[May93] Ernt Mayr. *Darwin et la pensée moderne de l'évolution*. O. Jacob, 1993. ISBN : 2-7381-0202-6.

[MFH91] Melanie Mitchell, Stephanie Forrest, and John H. Holland. The royal road for genetic algorithms: Fitness landscape and GA performance. In *[VB92]*, pages 245–254, 1991.

[MG94] Jean-Arcady Meyer and Agnès Guillot. From SAB90 to SAB94: Four years of animat research. In J-A. Meyer and D. Cliff, editors, *Proc. Third Conference on Simulated Adaptive Behavior*. MIT Press, Cambridge, MA, USA, 1994.

[MGSK88] H. Mülhenbein, M. Gorges-Schleuter, and O. Krämer. Evolution algorithms in combinatorial optimization. *Parallel Computing*, 7:65–85, 1988.

[Mic92] Zbigniew Michalewicz. *Genetic Algorithm + Data Structure = Evolution Programs*. Springer-Verlag, 1992.

[MM92a] R. Männer and B. Manderick, editors. *Proc. of the Second Conf. on Parallel Problem Solving in Nature*. Elsevier Science Publishers, Amsterdam, 1992.

[MM92b] R. Männer and B. Manderick, editors. *Proc. of the Second Conf. on Parallel Problem Solving in Nature*. Elsevier Science Publishers, Amsterdam, 1992.

[MSR94] E. Lutton M. Schonauer, J-M. Alliot and Edmund Ronald, editors. *Proc. of Évolution Artificielle*, 1994.

[MSR95] E. Lutton M. Schonauer, J-M. Alliot and Edmund Ronald, editors. *Proc. of Évolution Artificielle*. Springer-Verlag, 1995. (to appear).

[Müh91] Heinz Mühlenbein. Evolution in time and space – the parallel genetic algorithm. In *[Raw91b]*, pages 316–337. Morgan Kaufmann, San Mateo, CA, USA, 1991.

[MW92] Keith Mathias and Darrell Whitley. Genetic operators, the fitness landscape and the traveling salesman problem. In *[MM92b]*, pages 219–228, 1992.

[Pre95] Philippe Preux. Short-term evolution of the population of a GA's. Technical Report LIL-95-, Laboratoire d'Informatique du Littoral, 1995.

[PT95] Ph. Preux and E-G. Talbi. Assessing the evolutionary algorithm paradigm to solve hard problems. Technical Report LIL-95-4, Laboratoire d'Informatique du Littoral, 1995. (a summary was published in the *Proc. of the Workshop on Studying and Solving Really Hard Problems, Constraint Processing'95*, Marseille, 1995).

[Raw91a] Gregory J. E. Rawlins, editor. *Workshop on the Foundations of Genetic Algorithm and Classifier*, Bloomington, IN, USA, 1991. Morgan Kaufmann, San Mateo, CA, USA.

[Raw91b] Gregory J. E. Rawlins, editor. *Workshop on the Foundations of Genetic Algorithm and Classifier*, Bloomington, IN, USA, 1991. Morgan Kaufmann, San Mateo, CA, USA.

[Ray94a] Thomas S. Ray. Evolution, complexity, entropy, and artificial reality. *Physica D*, 1994. (submitted).

[Ray94b] Thomas S. Ray. An evolutionary approach to synthetic biology, Zen and the art of creating life. *Artificial Life*, 1(1), 1994.

[RCF94] Peter Ross, Dave Corne, and Hsiao-Lan Fang. Improving evolutionary timetabling with delta evaluation and directed mutation. In *[DSM94b]*, pages 556–565, 1994.

[Rec73] Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog Verlag, Stuttgart, 1973.

[Ros92] Bruce Rosen. Function optimization based on advanced simulated annealing. In *Proc. of Workshop on Physics and Computation, PhysComp'92*, Dallas, TX, 1992.

[RS95] Nicholas J. Radcliffe and Patrick D. Surry. Fundamental limitations on search algorithms. In *Lecture Notes in Computer Science 1000*. Springer-Verlag, 1995.

[Rud94] Günter Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, 5(1):96–101, January 1994.

[SB92] Nicol N. Schraudolph and Richard K. Belew. Dynamic parameter encoding for genetic algorithms. *Machine Learning*, 9:9–21, 1992.

[SCED89] J. David Schaffer, Richard A. Caruana, Larry J. Eshelman, and Rajarshi Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In *[Sch89]*, pages 51–60, 1989.

[Sch81] Hans-Paul Schwefel. *Numerical Optimization of Computer Models*. Wiley, Chichester, 1981.

[Sch89] J.D. Schaffer, editor. *Proc. of the Third International Conference on Genetic Algorithms*, Bloomington, IN, USA, 1989. Morgan Kaufmann, San Mateo, CA, USA.

[SDB+93] William M. Spears, Kenneth De Jong, Thomas Bäck, David B. Fogel, and Hugo de Garis. An overview of evolutionary computation. In *Proc. of the First European Conf. on Machine Learning*, 1993.

[SFP93] Robert E. Smith, Stephanie Forrest, and Alan S. Perelson. Searching for diverse, cooperative populations with genetic algorithms. *Evolutionary Computation*, 1(1), 1993.

[SM91] H-P. Schwefel and R. Männer, editors. *Proc. of the First Parallel Problem Solving in Nature*. Springer-Verlag, Berlin, 1991.

[SMM+91] T. Starkweather, S. McDaniel, K. Mathias, D. Whitley, and C. Whitley. A comparison of genetic sequencing operators. In *[BB91]*, pages 69–76, July 1991.

[Sta83] J.E.R. Staddon. *Adaptive behavior and learning*. Cambridge University Press, 1983.

[Sys89] Gilbert Syswerda. Uniform crossover in genetic algorithms. In *[Sch89]*, pages 2–9, 1989.

[Tal95] E-G. Talbi. Algorithmes génétiques parallèles : Techniques et applications. *Ecole CAPA95, à paraitre dans Parallélisme et applications irrégulières, edition Hermes, Cauterets, France*, Fev 1995.

[Tan89] Reiko Tanese. Distributed genetic algorithms. In *[Sch89]*, pages 434–439, 1989.

[TL94] Anthony Yiu-Cheung Tang and Kwong-Sak Leung. A modified edge recombination operator for the travelling salesman problem. In *[DSM94b]*, pages 180–188, 1994.

[TMS94] E. G. Talbi, T. Muntean, and I. Samarandache. Hybridation des algorithmes génétiques avec la recherche tabou. In *Evolution Artificielle EA94*, Toulouse, France, Sep 1994.

[VB92] Francesco Varela and Paul Bourgine, editors. *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*. MIT Press, Cambridge, MA, USA, December 1992.

[Vos93] Michael D. Vose. Modeling simple genetic algorithms. In *[Whi93]*, pages 63–73, 1993.

[Wei91] Gérard Weisbuch, editor. *Complex Systems Dynamics*. Santa Fe Institute Studies in the Sciences of Complexity. Addison-Wesley Publishing Company, 1991. SFI Studies in the Science of Complexity, Lectures Notes Vol. II, ISBN: 0-201-62732-9.

[Whi88] Darrell Whitley. GENITOR: A different genetic algorithm. In *Proc. of the Rocky Mountain Conference on Artificial Intelligence*, Denver, CO, USA, 1988.

[Whi91] Darrell Whitley. Fundamental principles of deception in genetic search. In *[Raw91b]*, pages 221–241, 1991.

[Whi93] Darrell Whitley, editor. *Proc. of the Workshop on Foundations of Genetic Algorithms*, Vail, CO, USA, 1993. Morgan Kaufmann, San Mateo, CA, USA.

[WM95] David H. Wolpert and William G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

[WMF91] D. Whitley, K. Mathias, and P. Fitzhorn. Delta coding: An interactive search strategy for genetic algorithms. In *[BB91]*, pages 77–84, 1991.

[WS95] Zhiming Wu and Marc Schonauer. éléments finis adaptatifs par programmation génétique. In *[MSR94]*, 1995.

[WSS91] Darrell Whitley, Timothy Starkweather, and Daniel Shaner. The traveling salesman and sequence scheduling: Quality solutions using genetic edge recombination. In *[Dav91]*, chapter 22, pages 350–372. 1991.

[WV94] Darrell Whitley and Michael D. Vose, editors. *Proc. of the Workshop on Foundations of Genetic Algorithms*, Estes Park, CO, USA, 1994. Morgan Kaufmann, San Mateo, CA, USA.