Ce texte est extrait d'un cours donné à l'École Polytechnique, et se veut une (brève) introduit aux algorithmes évolutionnaires.

1 Les algorithmes évolutionnaires

Les phénomènes physiques ou biologiques ont été à la source de nombreux algorithmes s'en inspirant plus ou moins librement. Ainsi les réseaux de neurones artificiels s'inspirent du fonctionnement du cerveau humain, l'algorithme de recuit simulé de la thermodynamique, et les *algorithmes évolutionnaires* (AEs) (dont les plus connus sont les *algorithmes génétiques*) de l'évolution darwinienne des populations biologiques.

Mais les algorithmes évolutionnaires sont avant tout des méthodes stochastiques d'optimisation globale. Et la souplesse d'utilisation de ces algorithmes pour des fonctions objectifs non régulières, à valeurs vectorielles, et définies sur des espaces de recherche non-standard (e.g. espaces de listes, de graphes, ...) permet leur utilisation pour des problèmes qui sont pour le moment hors d'atteinte des méthodes déterministes plus classiques.

Comme il a été indiqué, ce texte présente succinctement les algorithmes évolutionnaires. On n'hésitera pas à consulter le récent ouvrage de référence tout à fait général [9], ou, pour plus de détails, les ouvrages plus anciens [22, 3, 1].

Nous commencerons par donner les grandes lignes de ce qu'est un algorithme évolutionnaire, remontant très brièvement aux racines, la théorie de Darwin. Nous passerons ensuite en revue les implantations de la sélection "naturelle", indépendantes de toute application, et en donnerons les origines historiques. Enfin, nous détaillerons les implantations des opérateurs de variation dépendant de l'application dans les trois cas d'espaces de recherche les plus courants que sont les chaînes de bits, les vecteur de variables réelles et les arbres de la Programmation Génétique.

1.1 Le paradigme darwinien

1.1.1 La théorie de Darwin en 200 mots

Résumer la théorie de Darwin en quelques mots est bien sûr impossible. Disons que la (petite) partie qui est (grossièrement) imitée et caricaturée lors de la conception des AEs est basée sur l'idée que l'apparition d'espèces adaptées au milieu est la conséquence de la conjonction de deux phénomènes : d'une part la *sélection naturelle* imposée par le milieu – *les individus les plus adaptés survivent et se reproduisent* – et d'autre part des *variations non dirigées* du matériel génétique des espèces. Ce sont ces deux principes qui sous-tendent les algorithmes évolutionnaires.

Précisons tout de suite que le paradigme darwinien qui a inspiré ces algorithmes ne saurait en aucun cas être une justification pour leur emploi – pas plus que le vol des oiseaux ne peut justifier l'invention de l'avion[1]! Il y a maintenant

[1]on peut même utiliser ce type d'algorithme en n'étant pas un convaincu du darwinisme,

suffisamment d'exemples de succès pratiques (a posteriori) de ces algorithmes pour ne pas avoir à recourir à ce type d'argument.

1.1.2 Terminologie et notations

Soit à optimiser une fonction J à valeurs réelles définie sur un espace métrique Ω. Le parallèle avec l'évolution naturelle a entraîné l'apparition d'un vocabulaire spécifique (et qui peut paraître légèrement ésotérique) :

- La fonction objectif J est appelée fonction *performance*, ou fonction *d'adaptation* (*fitness* en anglais) ;
- Les points de l'espace de recherche Ω sont appelés des *individus* ;
- Les tuples d'individus sont appelés des *populations* ;
- On parlera d'une *génération* pour la boucle principale de l'algorithme.

Le temps de l'évolution est supposé discrétisé, et on notera Π_i la population, de taille fixe P, à la génération i.

1.1.3 Le squelette

La pression de l' "environnement", qui est simulée à l'aide de la fonction d'adaptation J, et les principes darwiniens de *sélection naturelle* et de *variations aveugles* sont implantés dans l'algorithme de la manière suivante :

- **Initialisation** de la *population* Π_0 en choisissant P individus dans Ω, généralement par tirage aléatoire avec une probabilité uniforme sur Ω ;
- **Évaluation** des individus de Π_0 (i.e. calcul des valeurs de J pour tous les individus) ;
- La génération i construit la population Π_i à partir de la population Π_{i-1} :
  - **Sélection** des individus les plus performants de Π_{i-1} au sens de J (*les plus adaptés se reproduisent*) ;
  - Application (avec une probabilité donnée) des **opérateurs de variation** aux *parents* sélectionnés, ce qui génère de nouveaux individus, les *enfants* ; on parlera de *mutation* pour les opérateurs unaires, et de *croisement* pour les opérateurs binaires (ou n-aires) ; à noter que cette étape est toujours **stochastique** :
  - **Évaluation** des enfants ;
  - **Remplacement** de la population Π_{i-1} par une nouvelle population créée à partir des enfants et/ou des "vieux" parents de la population Π_{i-1} au moyen d'une sélection darwinienne (*les plus adaptés survivent*).

---

d'où d'ailleurs l'emploi du néologisme "évolutionnaire", inspiré de l'anglais *evolutionary*.

- L'évolution stoppe quand le niveau de performance souhaité est atteint, ou qu'un nombre fixé de générations s'est écoulé sans améliorer l'individu le plus performant.

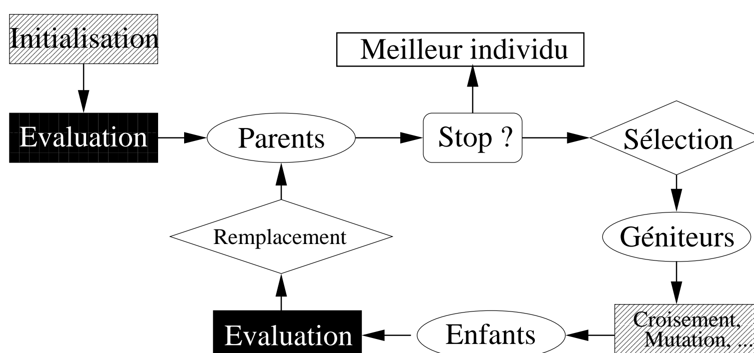


Figure 1: *Squelette d'un algorithme évolutionnaire*

Il est important de noter que, dans les applications à la plupart des problèmes numériques réels en tout cas (comme par exemple . . . les problèmes d'optimisation topologique de structures), l'essentiel du coût-calcul de ces algorithmes provient de l'étape d'évaluation (calcul des performances) : les tailles de populations sont de l'ordre de quelques dizaines, le nombre de générations de quelques centaines, ce qui donne lieu le plus souvent à plusieurs dizaines de milliers de calculs de J.

La suite de cette section va détailler les principaux composants d'un algorithme évolutionnaire, en en donnant des exemples concrets. Mais nous allons au préalable définir quelques notions-clés pour la compréhension du fonctionnement de ces algorithmes en pratique.

1.1.4 Deux points de vue

La composante principale de l'algorithme, qui est même en fait préalable à toutes les autres, est la *représentation*, ou choix de l'espace de recherche. Dans de nombreux cas, l'espace de recherche est totalement déterminé par le problème (i.e. c'est l'espace Ω sur lequel est définie la fonction objectif J). Mais il est toujours possible de transporter son problème dans un espace habilement choisi ("changement de variables") dans lequel il sera plus aisé de définir des opérateurs de variation efficaces. Cet espace est alors appelé *espace génotypique*, et l'espace de recherche initial Ω, dans lequel est calculée la performance des individus, est alors aussi appelé *espace phénotypique*.

On peut alors répartir les diverses étapes de l'algorithme en deux groupes : les étapes relatives au **darwinisme artificiel** (sélection et remplacement), qui ne dépendent que des valeurs prises par J, et pas de la représentation choisie, c'est-à-dire pas de l'espace génotypique ; et les étapes qui sont intimement liées à la nature de cet espace de recherche : ainsi, l'**initialisation** et les

opérateurs de variation sont spécifiques aux types de génotypes, mais par contre ne dépendent pas de la fonction objectif J (c'est le principe Darwinien des variations *aveugles*, ou *non dirigées*).

1.1.5 Les points-clé

Le terme de *diversité génétique* désigne la variété des génotypes présents dans la population. Elle devient nulle lorsque tous les individus sont identiques – on parle alors (a posteriori!) de *convergence* de l'algorithme. Mais il est important de savoir que lorsque la diversité génétique devient très faible, il y a très peu de chances pour qu'elle augmente à nouveau. Et si cela se produit trop tôt, la convergence a lieu vers un optimum local – on parle alors de *convergence prématurée*. Il faut donc préserver la diversité génétique, sans pour autant empêcher la convergence. Un autre point de vue sur ce problème est celui du *dilemme exploration-exploitation*.

A chaque étape de l'algorithme, il faut effectuer le compromis entre **explorer** l'espace de recherche, afin d'éviter de stagner dans un optimum local, et **exploiter** les meilleurs individus obtenus, afin d'atteindre de meilleurs valeurs aux alentours. Trop d'exploitation entraîne une convergence vers un optimum local, alors que trop d'exploration entraîne la non-convergence de l'algorithme.

Typiquement, les opérations de sélection et de croisement sont des étapes d'exploitation, alors que l'initialisation et la mutation sont des étapes d'exploration (mais attention, de multiples variantes d'algorithmes évolutionnaires s'écartent de ce schéma général). On peut ainsi régler les parts respectives d'exploration et d'exploitation en jouant sur les divers paramètres de l'algorithme (probabilités d'application des opérateurs, pression de sélection, ...). Malheureusement, il n'existe pas de règles universelles de réglages et seuls des résultats expérimentaux donnent une idée du comportement des divers composantes des algorithmes.

Nous allons maintenant passer en revue les différentes méthodes de sélection possible, en gardant à l'esprit qu'elles sont génériques et quasiment interchangeable quel que soit le problème traité.

1.2 Sélections naturelles ... artificielles

La partie darwinienne de l'algorithme comprend les deux étapes de **sélection** et de **remplacement**. Répétons que ces étapes sont totalement indépendantes de l'espace de recherche.

D'un point de vue technique, la différence essentielle entre l'étape de sélection et l'étape de remplacement est qu'un même individu peut être sélectionné plusieurs fois durant l'étape de sélection (ce qui correspond au fait d'avoir plusieurs enfants) alors que durant l'étape de remplacement, chaque individu est sélectionné une fois (et il survit) ou pas du tout (et il disparaît à jamais). Enfin, comme il a déjà été dit, la procédure de remplacement peut impliquer soit les enfants

seulement, soit également la population précédente dans son ensemble. Ceci mis à part, les étapes de sélection et de remplacement utilisent des procédures similaires de choix des individus, dont les plus utilisées vont maintenant être passées en revue.

On distingue deux catégories de procédures de sélection ou de remplacement (par abus de langage, nous appellerons sélection les deux types de procédures) : les procédures déterministes et les procédures stochastiques.

1.2.1 Sélection déterministe

On sélectionne les meilleurs individus (au sens de la fonction performance). Si plus de quelques individus doivent être sélectionnés, cela suppose un tri de l'ensemble de la population – mais cela ne pose un problème de temps calcul que pour des très grosses tailles de population.

Les individus les moins performants sont totalement éliminés de la population, et le meilleur individu est toujours sélectionné – on dit que cette sélection est *élitiste*.

1.2.2 Sélection stochastique

Il s'agit toujours de favoriser les meilleurs individus, mais ici de manière stochastique, ce qui laisse une chance aux individus moins performants. Par contre, il peut arriver que le meilleur individu ne soit pas sélectionné, et qu'aucun des enfants n'atteigne une performance aussi bonne que celle du meilleur parent . . .

Le **tirage de roulette** est la plus célèbre des sélections stochastiques. Supposant un problème de maximisation avec uniquement des performances positives, elle consiste à donner à chaque individu une probabilité d'être sélectionné proportionnelle à sa performance. Une illustration de la roulette est donnée Figure 2 : on lance la boule dans la roulette, et on choisit l'individu dans le secteur duquel la boule a fini sa course.

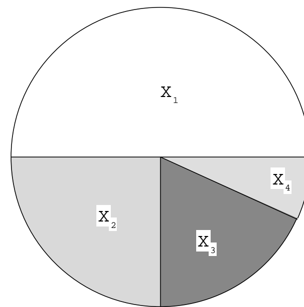


Figure 2: Sélection par tirage de roulette : cas de 4 individus de performances respectives 50, 25, 15 et 10. Une boule est "lancée" dans cette roulette, et l'individu dans le secteur duquel elle s'arrête est sélectionné.

Le tirage de roulette présente toutefois de nombreux inconvénients, en particulier reliés à l'échelle de la fonction performance : alors qu'il est théoriquement équivalent d'optimiser J et $\alpha J + \beta$ et J pour tout $\alpha > 0$, il est clair que le comportement de la sélection par roulette va fortement dépendre de α dans ce cas. C'est pourquoi, bien qu'il existe des procédure ajustant les paramètres α et β à chaque génération (mécanismes de *mise à l'échelle*), cette sélection est presque totalement abandonnée aujourd'hui.

La **sélection par le rang** consiste à faire une sélection en utilisant une roulette dont les secteurs sont proportionnels aux **rangs** des individus (P pour le meilleur, 1 pour le moins bon, pour une population de taille P). La variante linéaire utilise directement le rang, les variantes polynomiales remplaçant ces valeurs par $\frac{i}{P}^{\alpha}, \alpha > 0$. Le point essentiel de cette procédure de sélection est que les valeurs de J n'interviennent plus, seuls comptent les positions relatives des individus entre eux. Optimiser J et $\alpha J + \beta$ sont alors bien totalement équivalents.

La **sélection par tournoi** n'utilise aussi que des comparaisons entre individus – et ne nécessite même pas de tri de la population. Elle possède un paramètre T, taille du tournoi. Pour sélectionner un individu, on en tire T uniformément dans la population, et on sélectionne le meilleur de ces T individus. Le choix de T permet de faire varier la *pression sélective*, c'est-à-dire les chances de sélection des plus performants par rapport aux plus faibles. A noter que le cas $T = 2$ correspond, en espérance et au premier ordre en fonction de P, à la sélection par le rang linéaire.

1.3 Sélections multi-critères

Toutes les techniques de sélection présentées ci-dessus concernent le cas d'une fonction objectif à valeurs réelles. Cependant, la plupart des problèmes réels sont en fait des problèmes multi-critères, c'est-à-dire que l'on cherche à optimiser simultanément plusieurs critères contradictoires (typiquement, maximiser la qualité d'un produit en minimisant son prix de revient).

Or les algorithmes évolutionnaires sont une des rares méthodes d'optimisation permettant la prise en compte de telles situations : il "suffit" de modifier les étapes Darwiniennes d'un algorithme évolutionnaire pour en faire un algorithme d'optimisation multi-critère. Cette section présente quelques procédures de sélection multi-critères.

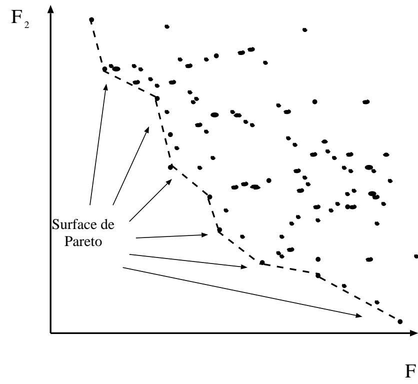
1.3.1 Front de Pareto

Dans un problème multi-critère dans lequel on cherche à optimiser plusieurs objectifs contradictoires, on appellera *front de Pareto* du problème l'ensemble des points de l'espace de recherche tels qu'il n'existe aucun point qui est strictement meilleur qu'eux sur tous les critères simultanément. Il s'agit de l'ensemble des meilleurs compromis réalisables entre les objectifs contradictoires, et l'objectif de l'optimisation va être d'identifier cet ensemble de compromis optimaux entre les critères.

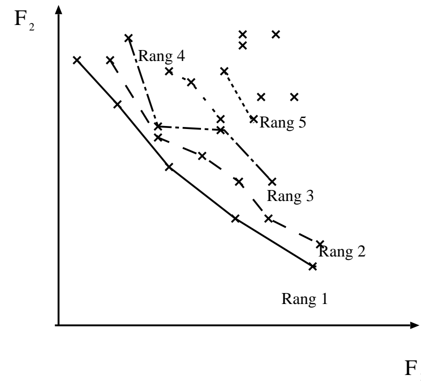
Plus formellement, soient $J_1, \ldots, J_n$ les objectifs dont on suppose qu'on cherche à les minimiser sur l'espace de recherche Ω.
Definition Soient x et y deux points de Ω. On dira que x *domine* y au sens de Pareto, noté $x \succ y$ si

$$\forall i \in [1, n], J_i(x) \leq J_i(y)$$
$$\exists j \in [1, n], J_j(x) < J_j(y)$$

La Figure 1.3.1-(a) donne un exemple de front de Pareto : l'ensemble de l'espace de recherche est représenté dans l'espace des objectifs, et les points extrémaux pour la relation de dominance au sens de Pareto forment le front de Pareto du problème (notez que l'on n'est pas toujours dans une situation aussi régulière que celle présentée Figure 1.3.1, et que le front de Pareto peut être concave, discontinu, ...)



(a) Les points extrémaux de l'ensemble de l'espace de recherche forment le front de Pareto du problème.



(b) Une population donnée est partiellement ordonnée par la relation de dominance au sens de Pareto.

Figure 3: *Front de Pareto et rangs de Pareto pour un problème de minimisation de deux objectifs.*

1.3.2 Sélections de Pareto

Lorsque l'on s'intéresse à l'optimisation multi-critère au sens de Pareto, il est possible de remplacer l'étape de sélection telle qu'elle a été décrite dans la Section 1.2 par des sélections basées sur la notion de dominance au sens de Pareto. Cependant, la relation d'ordre définie par la dominance étant une relation d'ordre partiel, il faudra rajouter une procédure de choix secondaires entre individus non comparables au sens de Pareto.

Critères de Pareto. Le plus utilisé des critères de sélection au sens de Pareto est le *rang de Pareto*, défini itérativement de la manière suivante : les individus non-dominés de la population courante sont dits de rang 1. Ils sont retirés de la population, et la procédure est itérée pour obtenir les rang 2, 3, ... Les individus de rang 1 (i.e. non dominés) vont approcher le front de Pareto du problème.

La *force de domination* est le nombre d'individus de la population courante qu'un individu donné domine : plus un individu domine d'autres individus, plus il est intéressant à conserver comme géniteur. De même, la *profondeur au sens de Pareto* d'un individu est le nombre d'autres individus de la population qui dominent un individu donné – un individu est d'autant plus intéressant à conserver qu'il est dominé par un petit nombre d'autres collègues.

Critères de diversité. Les critères précédent ne permettent pas d'ordonner toute la population – en fait, assez rapidement, les individus de la population courante qui vont approcher le front de Pareto du problème seront non comparables entre eux. Il faut donc ajouter un autre critère pour choisir entre eux. Les différents critères proposés favorisent la diversité. Le plus populaire aujourd'hui est la *distance de peuplement*, définie comme suit:

- pour chaque objectif i, on ordonne la population par valeurs croissantes de l'objectif
- pour chaque individu p, on définit d_i, *distance de peuplement partielle selon l'objectif i* comme la somme des distances de p à ses deux plus proches voisins dans la liste ordonnée
- la distance de peuplement totale D_p est donnée par la somme, sur l'ensemble des objectifs, des distances partielles.

Tournois au sens de Pareto. Pour sélectionner un géniteur a l'aide des outils définis ci-dessus, on utilise un *tournoi* (voir Section 1.2.2) en comparant tout d'abord les individus selon le critère de Pareto choisi, puis, en cas d'égalité, suivant le critère de diversité retenu.

1.4 Les moteurs d'évolution

On regroupe sous ce nom les ensembles sélection/remplacement, qui ne peuvent être dissociés lors des analyses théoriques du darwinisme au sein des algorithmes évolutionnaires. Un moteur d'évolution est donc la réunion d'une procédure de sélection et d'une procédure de remplacement. Toute combinaison des procédures présentées plus haut (et de bien d'autres encore) est licite. Toutefois, certaines combinaisons sont plus souvent utilisées, que ce soit pour des raisons historiques, théoriques ou expérimentales. Pour cette raison, les noms donnés sont souvent les noms des écoles historiques qui les ont popularisées – mais gardons à l'esprit que ces schémas sont totalement indépendants de l'espace de recherche, alors que nous verrons que les écoles historiques travaillaient sur des espaces de recherche bien précis.

Algorithme génétique générationnel (*GGA*)
Ce moteur utilise une sélection stochastique pour sélectionner exactement P parents (certains parents peuvent donc être sélectionnés plusieurs fois, d'autres pas du tout). Ces P parents donnent ensuite P enfants par application des opérateurs de variation (avec probabilité donnée, voir section 1.6.1). Enfin, ces P enfants remplacent purement et simplement les P parents pour la génération suivante. La variante élitiste consiste à garder le meilleur des parents s'il est plus performant que le meilleur des enfants.

Algorithme génétique stationnaire (*Steady-state GA – SSGA*)
Dans ce moteur, un individu est sélectionné, généralement par tournoi, un second si le croisement doit être appliqué, et l'enfant résultant (après croisement et mutation éventuels) est réinséré dans la population en remplacement d'un "vieil" individu sélectionné par un tournoi inversé, dans lequel le moins performant (ou le plus vieux) "gagne" ... et disparaît).

Stratégies d'évolution ($(\mu \overset{+}{,} \lambda)$-*ES*)
Deux moteurs sont regroupés sous ces appellations. Dans les deux cas, l'étape de sélection est un tirage uniforme (on peut dire qu'il n'y a pas de sélection au sens darwinien). À partir d'une population de taille μ (notations historiques!), λ enfants sont générés par application des opérateurs de variation. L'étape de remplacement est alors totalement déterministe. Dans le schéma (μ, λ)-ES (avec $\lambda > \mu$), les μ meilleurs enfants deviennent les parents de la génération suivante, alors que dans le schéma $(\mu + \lambda)$-ES, ce sont les μ meilleurs des $\mu + \lambda$ parents plus enfants qui survivent.

Algorithme multi-objectif NSGA-II
Cet algorithme utilise l'ordre total basés sur le rang de Pareto d'une part (ordre partiel) et la distance de peuplement en cas d'égalité du critère de Pareto (voir Section 1.3). Un tournoi basé sur cette relation d'ordre est utilisé pour la sélection des géniteurs, et le remplacement se fait de manière déterministe (suivant ce même ordre) parmi les parents plus enfants.

Il existe de nombreuses autres variantes de moteurs d'évolution multi-objectif, qu'il serait hors de notre propos de discuter ici. On se référera aux ouvrages [6, 4] pour plus de détails.

Jusqu'à présent, nous n'avons évoqué dans cette section que des techniques génériques, applicables à tout problème et surtout à tout espace de recherche. Nous allons maintenant faire un rapide survol des différentes écoles historiques d'algorithmes évolutionnaires, chacune étant de fait plus ou moins dédiée à un espace de recherche particulier. Les trois principaux contextes ainsi définis seront ensuite détaillés dans la dernière sous-section de cette introduction aux algorithmes évolutionnaires.

1.5 Les algorithmes historiques

On distingue quatre grandes familles historiques d'algorithme – et les différences entre elles ont laissé des traces dans le paysage évolutionnaire actuel, en dépit

d'une unification de nombreux concepts.

Algorithmes Génétiques (*GA*), proposés par J. Holland [18], dés les années 60, et popularisés son élève D.E. Goldberg [14], dans le Michigan, USA.
Les GA ont été imaginés initialement comme outils de modélisation de l'adaptation, et non comme outils d'optimisation, d'où un certain nombre de malentendus [7]. Ils travaillent dans l'espace des chaînes de bits $\{0,1\}^n$ avec les moteurs GGA et SSGA. Ce sont les plus connus des algorithmes évolutionnaires, et (malheureusement ?) souvent les seuls variantes connues des chercheurs des autres disciplines.

Stratégies d'évolution (*ES*), inventées par I. Rechenberg [25] et H.P. Schwefel [27], 1965, Berlin.
Les ES ont été mises au points par ces deux jeunes élèves ingénieurs travaillant sur des problèmes d'optimisation de tuyères (avec évaluation en soufflerie !), et les moteurs d'évolution sont ... les $(\mu \overset{+}{,} \lambda)$-ES. Un énorme progrès a été apporté par les techniques **auto-adaptatives** d'ajustement des paramètres de mutation, et aujourd'hui le meilleur algorithme pour les problèmes purement numériques est un descendant de ces méthodes historiques, l'algorithme CMA-ES [17, 15], basé sur une adaptation déterministe de la matrice de covariance de la mutation gaussienne (toutes ces mutations sont décrites en détail Section 1.8).

Programmation évolutionnaire (*EP*), imaginée par L.J. Fogel et ses co-auteurs [12] dans les années 60 également, et reprise par son fils D.B. Fogel [10] dans les années 90, en Californie, USA.
Mise au point initialement pour la découverte d'automates à états finis pour l'approximation de séries temporelles, EP a rapidement été généralisée à des espaces de recherche très variés. Le moteur utilisé ressemble à s'y méprendre à un $(P+P)$-ES – quoique développé complètement indépendamment – avec toutefois l'utilisation fréquente d'un remplacement plus stochastique que déterministe dans lequel les plus mauvais ont tout de même une (petite) chance de survie.

Programmation génétique (*GP*, *Genetic Programming*), amenée à maturité par J. Koza [19, 20], en Californie, USA.
Apparue initialement comme sous-domaine des GAs [5], GP est devenu une branche à part entière (conférence, journal, ...). La spécificité de GP est l'espace de recherche, un espace de programmes le plus souvent représentés sous forme d'arbres. GP cherche (et réussit parfois !) à atteindre un des vieux rêves des programmeurs, "écrire le programme qui écrit le programme". Les moteurs d'évolution utilisés sont souvent de type SSGA, mais avec des tailles de population énormes. Et les tendance récentes sont pour GP ... la parallélisation systématique et sur de grosses grappes de stations. Ainsi, les résultats récents les plus spectaculaires obtenus par Koza l'ont été avec des populations de plusieurs centaines de milliers d'individus, utilisant le modèle en îlots (une population par processeur, avec *migration* régulière des meilleurs individus entre processeurs voisins) sur des grappes *Beowulf*.

La dernière sous-section de cette introduction aux algorithmes évolutionnaires va maintenant détailler les parties spécifiques en terme de représentations et opérateurs de variation de trois des familles ci-dessus, AG, ES et GP.

1.6 Représentations et opérateurs de variation

Les composantes de l'algorithme qui dépendent intimement de la représentation choisie sont d'une part l'**initialisation**, i.e. le choix de la population initiale, dont le principe général est d'échantillonner le plus uniformément possible l'espace de recherche Ω, dans l'optique d'optimisation globale, et d'autre part les **opérateurs de variation**, qui créent de nouveaux individus à partir des parents sélectionnés. On distingue les opérateurs de croisement (binaires, ou plus généralement n-aires) et les opérateurs de mutation, unaires.

- L'idée générale du **croisement** est *l'échange de matériel génétique* entre les parents : si deux parents sont plus performants que la moyenne, on peut espérer que cela est dû à certaines parties de leur génotype, et que certains des enfants, recevant les "bonnes" parties de leurs deux parents, n'en seront que plus performants. Ce raisonnement, trivialement valable pour des fonctions performance linéaires sur des espaces de recherches réels par exemple, est extrapolé (et expérimentalement vérifié) à une classe plus étendue de fonctions, sans que les résultats théoriques aujourd'hui disponibles ne permettent de délimiter précisément la classe de fonctions pour lesquelles le croisement est utile. On adoptera donc une approche pragmatique : on tentera de définir un croisement en accord avec le problème traité (le lecteur intéressé pourra consulter les travaux de Radcliffe [24, 28]), et on le validera expérimentalement.
- L'idée directrice de la **mutation** est de permettre de visiter tout l'espace. Les quelques résultats théoriques de convergence des algorithmes évolutionnaires ont d'ailleurs tous comme condition l'**ergodicité** de la mutation, c'est-à-dire le fait que tout point de l'espace de recherche peut être atteint en un nombre fini de mutations. Mais la mutation doit également pouvoir être utile à l'ajustement fin de la solution – d'où l'idée d'une mutation de "force" réglable, éventuellement au cours de l'algorithme lui-même (voir section 1.8).

1.6.1 Application des opérateurs de variation

Tous les opérateurs de variation ne sont pas appliqués systématiquement à tous les individus à chaque génération. Le schéma le plus courant est d'appliquer *séquentiellement* un opérateur de croisement, puis un opérateur de mutation, chacun avec une probabilité donnée (un paramètre de l'algorithme, laissé au choix de l'utilisateur). On notera p_c et p_m les probabilités respectives d'application du croisement et de la mutation.

Il est par contre relativement fréquent de disposer de plusieurs opérateurs de chaque type (croisement ou mutation) et de vouloir les utiliser au sein du même

algorithme (e.g. le croisement par échange de coordonnées et le croisement barycentrique dans le cas de l'optimisation réelle). Il faut alors introduire de nouveaux paramètres, à savoir l'importance relative de chaque opérateur par rapport aux autres (e.g. on veut faire 40% de croisements par échange de coordonnées, et 60% de croisement barycentriques, voir section 1.8).

Il est bien sûr possible d'imaginer bien d'autres schémas d'application des opérateurs de variation, ainsi d'ailleurs que d'autres types d'opérateurs ni unaires ni binaires (alors appelés opérateurs d'*orgie*).

Nous allons maintenant donner trois exemples d'espaces de recherche parmi les plus utilisés – et détaillerons pour chacun les composantes de l'algorithme qui dépendent de la représentation. Il ne faut toutefois pas perdre de vue que la puissance des algorithmes évolutionnaires vient de leur capacité à optimiser des fonctions définies sur des espaces de recherche bien plus variés que ces trois espaces.

1.7 Les chaînes de bits

L'espace de recherche est ici $\Omega = \{0,1\}^N$ (espace des *bitstring* en anglais). Historiquement (voir Section 1.5) il s'agit de la représentation utilisée par l'école des algorithmes génétiques, et la justification de l'utilisation intensive de cet espace de recherche particulier était fondé à la fois sur un parallèle encore plus précis avec la biologie (une chaîne de bits étant assimilée à un chromosome) et sur des considérations théoriques qui ne seront pas détaillées ici (voir [9], ainsi que les références de la Section 1.5 à ce sujet). Ce contexte reste toutefois utilisé dans certains domaines – mais il permet surtout une présentation aisée des diveres composantes de l'algorithme.

1.7.1 Initialisation

Dans le cadre des chaînes de bits, il est possible tirer les individus de la population initiale uniformément sur l'espace Ω: chaque bit de chaque individu est tiré égal à 0 ou à 1 avec une probabilité de $\frac{1}{2}$.

1.7.2 Croisement

Plusieurs opérateurs de croisement ont été proposés, qui tous échangent des bits (à position fixée) entre les parents. La Figure 4 donne l'exemple du croisement à 1 point, et nous laissons au lecteur le soin d'écrire les croisement à 2, 3, $\ldots N$ points. Un autre type de croisement, appelé croisement *uniforme*, consiste à tirer indépendamment pour chaque position (avec probabilité 0.5) de quel parent proviendra le bit correspondant chez chaque enfant.

1.7.3 Mutation

Les opérateurs de mutation de chaînes de bits modifient tous aléatoirement certains bits. Le plus usité, appelé *bit-flip*, consiste à inverser chaque bit de

$$\begin{matrix} (b_1, \ldots, b_N) \\ (c_1, \ldots, c_N) \end{matrix} \Bigg\} \xrightarrow{P_c} \Bigg\{ \begin{matrix} (b_1, \ldots, b_l, c_{l+1}, \ldots, c_N) \\ (c_1, \ldots, c_l, b_{l+1}, \ldots, b_N) \end{matrix}$$

Figure 4: Croisement à un point : l'entier l est tiré uniformément dans $[1, N-1]$, et les deux moitiés des chromosomes sont échangées.

l'individu muté indépendamment avec une (petite) probabilité $p_,$.

$$(b_1, b_2, \ldots, b_N) \xrightarrow{p_m} (b_1, b_2, \ldots, 1 - b_l, b_{l+1}, \ldots, b_N)$$

Une autre possibilité est de prédéfinir un nombre k de bits à modifier (généralement 1), et de choisir ensuite au hasard k positions dans l'individu et d'inverser les bits correspondants.

1.8 Les vecteurs de réels.

C'est bien sûr le cas le plus fréquent en calcul numérique : Ω est un sous-ensemble de $\mathbb{R}^n$, borné ou non. On parle alors aussi d'*optimisation paramétrique*.

1.8.1 Initialisation

Si $\Omega = \prod [a_i, b_i]$ (cas borné), on tire en général uniformément chaque coordonnée dans l'intervalle correspondant. Par contre, si Ω n'est pas borné, il faut faire des choix. On pourra soit utiliser un sous-ensemble borné de Ω et effectuer un choix uniforme dans cet ensemble, soit par exemple tirer mantisses et exposants uniformément dans des intervalles bien choisis. Bien entendu, on pourrait dire que les nombres réels représentés en machine sont de toute façon bornés – mais il est néanmoins généralement préférable de distinguer les deux cas.

1.8.2 Le croisement

On peut bien entendu appliquer des opérateurs d'échange de coordonnées comme dans le cas des chaînes de bits. Mais on peut également – et c'est en général bien plus efficace – “mélanger” les deux parents par combinaison linéaire. On parle alors de *croisement arithmétique* :

$$(\mathbf{X}, \mathbf{Y}) \xrightarrow{p_c} \alpha \mathbf{X} + (1 - \alpha) \mathbf{Y}, \alpha = U[0, 1]$$

ou

$$(\mathbf{X}, \mathbf{Y}) \xrightarrow{p_c} (\alpha_i X_i + (1 - \alpha_i) Y_i)_{i=1 \ldots n}, \alpha_i = U[0, 1], \text{ indépendants}$$

La première version revient à choisir l'enfant uniformément sur le segment $[\mathbf{XY}]$ alors que la deuxième revient à tirer l'enfant uniformément sur l'hypercube dont $[\mathbf{XY}]$ est une diagonale. Remarquons que l'échange de coordonnées revient à choisir comme enfant un des sommets de cet hypercube. Et signalons qu'on

peut également choisir les coefficients des combinaisons linéaires dans un intervalle plus grand (e.g. $[-0.5, 1.5]$) afin d'éviter la contractance de l'opérateur de croisement, source de perte de diversité génétique (voir Section 1.1.5).

1.8.3 La mutation

Dans le cadre de l'optimisation paramétrique, la mutation la plus employée est la mutation *gaussienne*, qui consiste à rajouter un bruit gaussien au vecteur des variables. La forme la plus générale est alors

$$\mathbf{X} := \mathbf{X} + \sigma N(0, C) \tag{1}$$

où σ est un paramètre positif, appelé le *pas* de la mutation (*step-size* en anglais), et $N(0, C)$ représente un tirage de loi normale centrée de matrice de covariance C (symétrique définie positive).

Tout l'art est alors bien sûr dans le choix des paramètres σ et C. L'influence de σ est intuitive : des grandes valeurs résulteront en une exploration important

On peut évidemment demander à l'utilisateur de fixer cette valeur. Mais il est clair – et des études théoriques sur des fonctions simples l'ont démontré – que cette valeur devrait décroître au fil des générations en fonction de l'avancement de la recherche, et il est impossible de fixer a priori un schéma de décroissance qui soit synchrone avec l'éventuelle convergence de l'algorithme, pour une fonctions quelconques.

L'état de l'art a longtemps été la mutation *auto-adaptative*. Due aux pères des stratégies d'évolution (I. Rechenberg et H.-P. Schwefel, voir Section 1.5), ce type de mutation considère en fait les paramètres de la mutation eux-mêmes comme des variables supplémentaires, et les fait également évoluer via croisement et mutation !

L'idée sous-jacente est que, bien que la sélection soit faite sur les valeurs de la fonction objectif J et non pas directement sur les paramètres de la mutation, un individu ne peut pas survivre longtemps s'il n'a pas les paramètres de mutation adaptés à la "topographie" de la portion de la surface définie par J où il se trouve : schématiquement par exemple, les valeurs de σ doivent être petites lorsque le gradient de J est important, afin d'avoir plus de chance de faires des "petits pas".

On distingue trois cas suivant la complexité du modèle de matrice de covariance :

- Le cas **Isotrope**: il y a un σ par individu (soit $C = Id$). La mutation consiste alors à muter tout d'abord σ selon une loi log-normale (afin de respecter la positivité de σ, et d'avoir des variations symétriques par rapport à 1), puis à muter les variables à l'aide de la nouvelle valeur de σ :

$$\begin{cases} \sigma := \sigma \; e^{\tau N_0(0,1)} \\ X_i := X_i + \sigma N_i(0,1) \;\; i = 1, \ldots, d \end{cases}$$

  Les $N_i(0, 1)$ sont des réalisations indépendantes de variables aléatoires scalaires gaussiennes centrées de variance 1.

- Le cas **Non-isotrope**: il y a un σ par individu (soit $C = \text{diag}(\sigma_1, \ldots, \sigma_d)$. A noter que la mutation des σ_i comporte deux termes de forme log-normale, un terme commun à tous les σ_i et un terme par direction :

$$\begin{cases} \kappa = \tau N_0(0,1) \\ \sigma_i := \sigma_i \ e^{\kappa + \tau' N_i(0,1)} \quad i = 1, \ldots, d \\ X_i := X_i + \sigma_i N_i'(0,1) \quad i = 1, \ldots, d \end{cases}$$

  Les $N_i(0,1)$ et $N_i'(0,1)$ sont des réalisations indépendantes de variables aléatoires scalaires gaussiennes centrées de variance 1.

- Le cas général, dit **corrélé**, dans lequel C est une matrice symétrique définie positive quelconque. On utilise alors pour pouvoir transformer C par mutation tout en gardant sa positivité une représentation canonique en produit de $d(d-1)/2$ rotations par une matrice diagonale. La mutation s'effectue alors en mutant d'une part la matrice diagonale, comme dans le cas non-isotrope ci-dessus, puis en mutant les angles des rotations :
  $\vec{N}(0, C(\vec{\sigma}, \vec{\alpha})) = \prod_{i=1}^{d-1} \prod_{j=i+1}^{d} R(\alpha_{ij}) \vec{N}(0, \vec{\sigma})$

$$\begin{cases} \sigma_i \ = \sigma_i e^{\tau' N_0(0,1) + \tau N_i(0,1)} \quad i = 1, \ldots, d \\ \alpha_j := \alpha_j + \beta N_j(0,1) \quad j = 1, \ldots, d(d-1)/2 \\ \vec{X} := \vec{X} + \vec{N}(0, C(\vec{\sigma}, \vec{\alpha})) \end{cases}$$

  Ici encore, les divers $N_i(0,1)$ apparaissant dans les formules ci-dessus sont des réalisations indépendantes de variables aléatoires scalaires gaussiennes centrées de variance 1.

Suivant Schwefel [27], les valeurs recommandées (et relativement robustes) pour les paramètres supplémentaires sont $\tau \propto \frac{1}{\sqrt{2\sqrt{d}}}$, $\tau' \propto \frac{1}{\sqrt{2d}}$, $\beta = 0.0873$ (=5°)

1.8.4 État de l'art pour la mutation gaussienne

Mais l'état de l'art aujourd'hui en matière d'optimisation paramétrique évolutionnaire est sans conteste la récente méthode *Covariance Matrix Adaptation* (CMA). Partant du constat que d'une part l'adaptation de la matrice C dans les méthodes auto-adaptatives ci-dessus est très lente, et d'autre part les pas successifs de l'algorithme contiennent de l'information sur la fonction objectif, N. Hansen [16, 17, 15] a mis au point une adaptation **déterministe** de la matrice de covariance C (la mutation est donnée par la formule (1).

Tout d'abord, il a été expérimentalement constaté qu'il était préférable de tirer tous les enfants d'un unique parent, la moyenne des μ individus de la population à l'étape n :

$$<X>^n = \frac{1}{n} \sum_{i=1}^{\mu} X_i^n$$

L'algorithme consiste alors à tirer λ enfants à partir de $<X>^n$, et à sélectionner les μ meilleurs pour la génération suivante (voir section 1.8).

Une deuxième constatation concerne ensuite le pas de mutation: si deux mutations successives ont eu lieu dans la même direction, alors il faudrait sans doute augmenter le pas. De manière formelle, cela donne :

$$\begin{cases} p_\sigma^{n+1} = (1-c_\sigma)p_\sigma^n + \sqrt{\mu}\sqrt{c_\sigma(2-c_\sigma)}(C^n)^{-\frac{1}{2}}\frac{<X>_\mu^{n+1} - <X>_\mu^n}{\sigma^n} \\ \sigma^{n+1} = \sigma^n \exp\left(\frac{1}{d_\sigma}\left(\frac{||p_\sigma^{n+1}||}{E(||\mathcal{N}(0,I_d)||)} - 1\right)\right). \end{cases}$$

L'idée des facteurs $(1-c_\sigma)$ et $\sqrt{c_\sigma(2-c_\sigma)}$ est que si $p_\sigma^n \sim \mathcal{N}(0,I_d)$ et $\sqrt{\mu}(C^n)^{-\frac{1}{2}}\frac{<X>^{n+1} - <X>^n}{\sigma^n} \sim \mathcal{N}(0,I_d)$ et s'ils sont indépendants, alors $p_\sigma^{n+1} \sim \mathcal{N}(0,I_d)$.

De plus, si on suppose qu'il n'y a pas de sélection ($\lambda = \mu$), alors $\sqrt{\mu}\frac{<X>^{n+1} - <X>^n}{\sigma^n} = \sqrt{\lambda}\frac{\frac{1}{\lambda}\sum_{i=1}^{\lambda} X_{i:\lambda}^n - <X>^n}{\sigma^n} \sim \mathcal{N}(0,I_d)$ et dans ce cas on ne veut pas modifier σ^n, d'où le terme en $\frac{||p_\sigma^{n+1}||}{E(||\mathcal{N}(0,I_d)||)} - 1$.

A noter que la constante $E[\|\mathcal{N}(0,I_d)\|] = \sqrt{2}\Gamma(\frac{n+1}{2})/\Gamma(\frac{n}{2})$ est approchée en pratique par $\sqrt{d}(1 - \frac{1}{4d} + \frac{1}{21d^2})$.

Enfin, en ce qui concerne la matrice de covariance elle-même, on la met à jour en lui ajoutant une matrice de rang 1 de direction propre la dernière direction de descente utilisée :

$$\begin{cases} p_c^{n+1} = (1-c_c)p_c^n + \sqrt{\mu}\sqrt{c_c(2-c_c)}\frac{<X>_\mu^{n+1} - <X>_\mu^n}{\sigma^n} \\ C^{n+1} = (1-c_{\mathrm{cov}})C^n + c_{\mathrm{cov}}p_c^{n+1}p_c^{n+1\,T} \end{cases}$$

À noter qu'il existe une variante dans laquelle la mise à jour se fait à l'aide d'une matrice de rang plus élevé dans les cas de grandes dimensions [15].

1.9 La programmation génétique

La programmation génétique (GP, pour *Genetic Programming* en anglais) peut être vue comme l'évolution artificielle de “programmes”, ces programmes étant représentés sous forme d'arbres (il existe des variantes de GP qui utilisent une représentation linéaire des programmes, dont il ne sera pas question ici). Elle constitue aujourd'hui une des branches les plus actives des algorithmes évolutionnaires.

On suppose que le langage dans lequel on décrit les programmes parmi lesquels on cherche un programme optimal est constitué d'*opérateurs* et d'*opérandes de base*, tout opérateur pouvant opérer sur un nombre fixe d'opérandes, et rendant lui-même un résultat pouvant à son tour être l'opérande d'un des opérateurs.

L'idée de base consiste à représenter de tels programmes sous forme d'arbres. L'ensemble des *noeuds* de l'arbre $\mathcal{N}$ est l'ensemble des opérateurs, et l'ensemble des *terminaux* de l'arbre $\mathcal{T}$ est l'ensemble des opérandes de bases.

L'intérêt d'une telle représentation, qui permet d'utiliser les principes évolutionnaires sur ce type d'espace de recherche, est la fermeture syntaxique de l'opérateur de croisement : en effet, alors qu'il est difficile d'imaginer croiser deux programmes séquentiels écrits dans un langage de haut niveau (pensez à du C ou du Java) en obtenant un programme valable comme résultat du croisement, le croisement d'arbres ne pose aucun problème, et donne toujours un arbre représentant un programme valide, comme nous allons le voir ci-dessous.

Les premiers travaux en GP optimisaient des programmes écrits dans un sous-ensemble du langage LISP travaillant sur des variables booléennes. Les noeuds étaient constitués d'opérations logiques (e.g., AND, OR, . . .) et de tests (e.g. l'opérateur ternaire IF Arg1 THEN Arg2 ELSE Arg3), les opérandes des variables du problème.

De nombreux autres langages ont été utilisés dans le cadre de GP, mais nous nous contenterons ici de donner l'exemple trivial du cas de programmes opérant sur des valeurs réelles, avec pour terminaux soit des valeurs constantes soit l'un des symboles X et Y ($\mathcal{T} = \{X, Y, \mathcal{R}\}$) et pour noeuds les opérations d'addition et de multiplication ($\mathcal{N} = \{+, *\}$). L'ensemble des programmes que décrivent de tels arbres est alors l'ensemble des polynômes réels à 2 variables X et Y (voir Figure 5).

Signalons enfin qu'en programmation génétique comme en programmation manuelle, il est possible et même souhaitable d'utiliser les concepts de programmation structurée : la notion de *subroutine* par exemple a été introduite rapidement dans les programmes-arbres sous la forme des *ADF – Automatically Defined Functions* [20], de même que des structures de contrôles au sein des opérateurs élémentaires (boucles, récursion, . . .).

Examinons maintenant les composantes spécifiques d'un algorithme évolutionnaire manipulant des arbres.

1.9.1 Initialisation

L'idée la plus immédiate pour initialiser un arbre consiste à donner à chacun des noeuds et des terminaux une probabilité d'être choisi, et à tirer au sort à chaque étape un symbole parmi l'ensemble des noeuds et des terminaux. Si c'est un noeud, on renouvelle la procédure pour chacun des arguments dont l'opérateur a besoin.

On voit tout de suite qu'une telle procédure peut ne jamais se terminer, et qu'il faut imposer une profondeur maximale aux arbres, profondeur à partir de laquelle on n'autorise plus que les terminaux.

Le problème devient alors le choix des probabilités des différents symboles : si le poids global des noeuds est plus grand que celui des opérateurs, la plupart des arbres auront pour profondeur la profondeur maximale. Au contraire, si le poids des terminaux l'emporte sur celui des noeuds, les arbres seront très rarement profonds. Dans les deux cas, la **diversité** de la population initiale de l'algorithme sera très réduite.

C'est pourquoi la procédure la plus utilisée aujourd'hui initialise la population en plusieurs étapes, chaque étape utilisant des profondeurs maximales différentes, et appliquant soit la procédure décrite ci-dessus, soit générant systématiquement des arbres de profondeur égale à la profondeur en cours. Cette initialisation s'appelle *ramped half-and-half* et est décrite en détail dans l'ouvrage de référence en GP, [1].

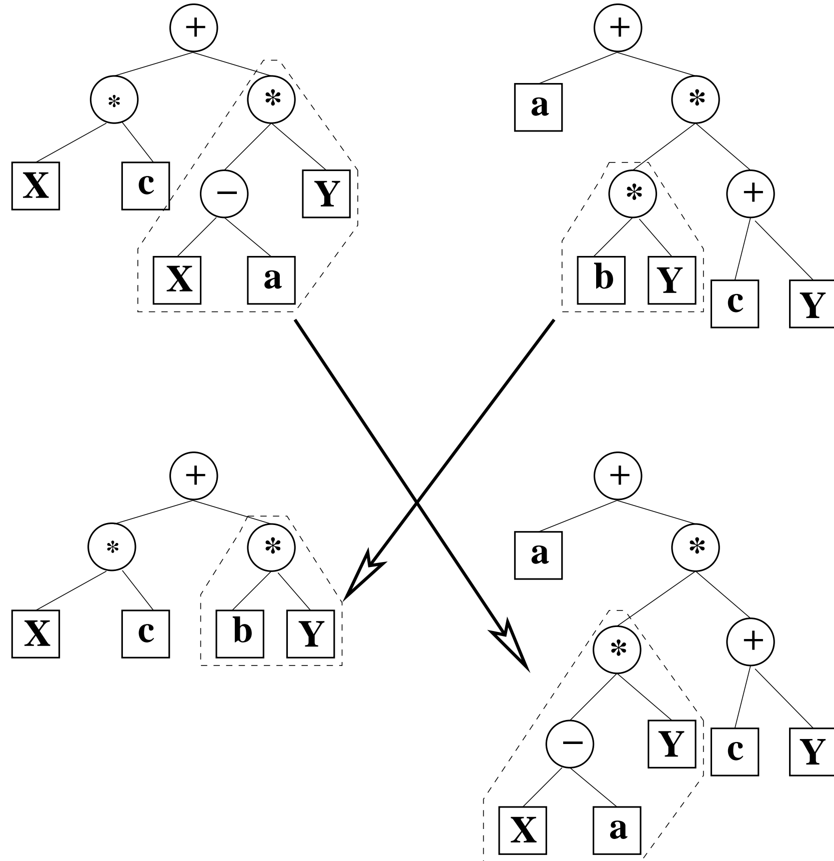


Figure 5: Exemple de croisement en programmation génétique, dans le cadre de la représentation des polynômes à 2 variables réelles : l'échange des sous-arbres délimités par les pointillés entre les polynômes $cX + XY - aY$ et $a + bY^2 + bcY$ donne ici $cX + bY$ et $XY^2 + cXY - aY^2 - acY + a$

1.9.2 Croisement

Définir une procédure de croisement n'offre a priori aucune difficulté : on choisit aléatoirement un sous-arbre dans chacun des parents, et on échange les deux

sous-arbres en question (voir Figure 5). Du fait de l'homogénéité des types, le résultat est toujours un programme valide – l'ensemble des programmes est **fermé** pour l'opérateur de croisement.

On remarquera cependant que le croisement de deux parents de même taille (nombre de noeuds) a tendance à générer un parent long et un parent court. De plus, les arbres longs ont en moyenne une plus grande expressivité que les arbres courts. La conjonction de ces deux phénomènes résulte en une croissance continue et incontrôlable de la taille des programmes, dénommée **bloat**. C'est un des problème majeurs de la programmation génétique en pratique, que de nombreux travaux tentent de comprendre (l'explication ci-dessus est loin d'être la seule à avoir été proposée), et de contrôler. Mais une des difficulté est qu'il est a été constaté expérimentalement qu'un minimum de bloat est nécessaire pour espérer obtenir des arbres performants.

1.9.3 Mutation

Les travaux originaux de Koza n'utilisaient comme opérateur de variation que le croisement. Mais il utilisait des populations de très grande taille (plusieurs milliers), et pouvait ainsi espérer que la plupart des "morceaux de programme" intéressants étaient présents dans la population initiale et qu'il suffisait de les assembler . . . par croisement.

Aujourd'hui, la plupart des travaux utilisent des opérateurs de mutation, et ce d'autant plus que la taille de la population est petite. De très nombreux opérateurs de mutation ont été imaginés, le plus simple (et le plus destructeur et moins performant) étant de remplacer un sous-arbre aléatoirement choisi par un sous-arbre aléatoirement construit à l'aide de la procédure d'initialisation. Signalons les mutations par changement de noeud (on remplace un noeud choisi aléatoirement par un noeud de même arité), par insertion de sous-arbre (plutôt que de remplacer totalement un sous-arbre, on insère à un emplacement donné un opérateur dont l'un des opérandes est le sous-arbre qui était présent à cet endroit), et son inverse (qui supprime un noeud et tous ses arguments sauf un, qui prend sa place).

1.9.4 Les terminaux réels

Les valeurs numériques réelles utilisées comme terminaux d'arbres de type réel tiennent une place particulière dans la programmation génétique. D'une part, elles ont été longtemps absente des travaux publiés (Koza n'utilisait que des valeurs entières choisies parmi les 10 premiers entiers par exemple). D'autre part, elle requièrent un traitement particulier, tant en terme d'initialisation (on ne choisit pas une valeur parmi un nombre fini de possibles) qu'en terme de variation.

L'initialisation se fait généralement uniformément sur un intervalle donné. Le croisement se contente d'échanger des sous-arbres, et les valeurs numériques n'y jouent aucun rôle particulier. Par contre, la mutation des constantes est un élément crucial pour le succès d'une optimisation par programmation génétique

mettant en jeu des valeurs numériques. On utilise généralement des mutations gaussiennes (voir section 1.8, éventuellement augmentées de paramètres auto-adaptatifs (mais limités à une variance par terminal). Il est même fréquent, si le coût CPU le permet, d'optimiser localement les valeurs numériques pour une structure d'arbre donnée, soit par algorithme déterministe si le problème le permet [26] soit même en utilisant un algorithme évolutionnaire imbriqué [8].

La programmation génétique est un formidable outil de créativité – et John Koza ne désespère pas d'obtenir un jour des résultats brevetables [21].

1.10 Conclusion

Ce texte a présenté les algorithmes évolutionnaires en toute généralité. Il faut en retenir d'une part leur robustesse vis-à-vis des optima locaux, et d'autre part leur souplesse d'utilisation, particulièrement en terme de champ d'application (choix de l'espace de recherche). Si la programmation génétique a été l'unique illustration de cette souplesse pour le moment, il existe aujourd'hui de nombreuses applications qui en témoignent également, que nous ne citerons par peur d'en oublier, renvoyant aux ouvrages beaucoup plus complets décrits dans la dernière section.

2 Bibliographie

Nous allons citer ici les principaux ouvrages généraux d'introduction aux algorithmes évolutionnaires. Pour ce faire, le domaine des algorithmes évolutionnaire étant encore en constant mouvement, nous adopterons une démarche par ordre chronologique inverse, commençant par les ouvrages considérés comme l'état de l'art aujourd'hui pour citer ensuite, pour des raisons historiques, les références plus anciennes . Mais signalons tout de même que les références les plus à jour, mais qu'il est impossible de citer ici pour des raisons évidentes, restent les articles publiés récemment dans les journaux du domaine (citons les principaux par ordre chronologique d'apparition, *Evolutionary Computation*, MIT Press, *IEEE Transactions on Evolutionary Computing*, IEEE Press, et *Genetic Programming and Evolvable Machines*, Kluwer), voire dans les multiples conférences dédiées aux algorithmes évolutionnaires.

L'ouvrage le plus complet aujourd'hui, sans doute parce que le plus récent, comme il vient d'être indiqué, est [9]. Toujours parmi les livres d'introduction, mais plus spécialisés, citons [23] sur les algorithmes génétiques, [2] pour les stratégies d'évolution, et [1] pour le programmation génétique.

Deux essais sont à distinguer : certes centré sur les algorithmes génétiques d'un point de vue évolutionnaire, [13] tente de dégager des idées plus générale sur l'aide à l'innovation que peut apporter ces algorithmes dans tous les domaines ; et l'un des premiers à avoir utilisé le terme "Evolutionary Computation" est [10].

Un ouvrage collectif se voulant LA référence a été assemblé en 1997 [3]. Toutefois, les plans initiaux de mise à jour continue n'ont jamais été contrétisés, et cet imposant volume est aujourd'hui légèrement dépassé.

Enfin, il faut citer les ouvrages fondateurs, au premier rang desquels [14], qui a été pour des générations de chercheurs la première rencontre avec les algorithmes génétiques, et a durablement orienté leurs recherches futures. Et, bien sûr, les ouvrages séminaux déjà cités dans la partie historique (Section 1.5), [18] pour les algorithmes génétiques, [25] et [27] pour les stratégies d'évolution, [12] pour la programmation évolutionnaire et [19] pour la programmation génétique (quoique ce dernier ouvrage soit beaucoup plus récent que les précédents, il marque le début du domaine de la programmation génétique). Et pour terminer avec l'histoire, il faut rappeler que la plupart des idées de base des algorithmes évolutionnaires ont été proposées ... vers la fin des années 50, et on en trouvera une compilation intéressante dans [11].

References

[1] W. Banzhaf, P. Nordin, R.E. Keller, and F.D. Francone. *Genetic Programming — An Introduction On the Automatic Evolution of Computer Programs and Its Applications.* Morgan Kaufmann, 1998.

[2] T. Bäck. *Evolutionary Algorithms in Theory and Practice.* New-York:Oxford University Press, 1995.

[3] Th. Bäck, D.B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation.* Oxford University Press, 1997.

[4] C. A. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems.* Kluwer Academic Publishers, 2002.

[5] N.J. Cramer. A representation for the adaptive generation of simple sequential programs. In J. J. Grefenstette, editor, *Proceedings of the* 1st *International Conference on Genetic Algorithms*, pages 183–187. Laurence Erlbaum Associates, 1985.

[6] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms.* John Wiley, 2001.

[7] K. A. DeJong. Are genetic algorithms function optimizers ? In R. Manner and B. Manderick, editors, *Proceedings of the* 2nd *Conference on Parallel Problems Solving from Nature*, pages 3–13. North Holland, 1992.

[8] M. Ebner. Evolutionary design of objects using scene graphs. In C. Ryan et al., editor, *Proc. EuroGP 2003.* Springer-Verlag, 2003.

[9] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing.* Springer Verlag, 2003.

[10] D. B. Fogel. *Evolutionary Computation. Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, NJ, 1995.

[11] D.B. Fogel. *Evolutionary Computing: The Fossile Record*. IEEE Press, 1998.

[12] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. New York: John Wiley, 1966.

[13] D. Goldberg. *The Design of Innovation: Lessons from and for Genetic and Evolutionary Algorithms*. MIT Press, 2002.

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

[15] N. Hansen, S. Müller, and P. Koumoutsakos. Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). *Evolution Computation*, 11(1), 2003.

[16] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaption. In *Proceedings of the Third IEEE International Conference on Evolutionary Computation*, pages 312–317. IEEE Press, 1996.

[17] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[18] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[19] J. R. Koza. *Genetic Programming: On the Programming of Computers by means of Natural Evolution*. MIT Press, Massachusetts, 1992.

[20] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Massachussetts, 1994.

[21] J. R. Koza. Human-competitive machine intelligence by means of genetic algorithms. In L. Booker, S. Forrest, M. Mitchell, and R. Riolo, editors, *Festschrift in honor of John H. Holland*, pages 15–22. Ann Arbor, MI: Center for the Study of Complex Systems, 1999.

[22] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, New-York, 1992-1996. 1st-3rd edition.

[23] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1996.

[24] N. J. Radcliffe. Forma analysis and random respectful recombination. In R. K. Belew and L. B. Booker, editors, *Proceedings of the* 4th *International Conference on Genetic Algorithms*, pages 222–229. Morgan Kaufmann, 1991.

[25] I. Rechenberg. *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution*. Fromman-Hozlboog Verlag, Stuttgart, 1972.

[26] M. Schoenauer, M. Sebag, F. Jouve, B. Lamy, and H. Maitournam. Evolutionary identification of macro-mechanical models. In P. J. Angeline and Jr K. E. Kinnear, editors, *Advances in Genetic Programming II*, pages 467–488, Cambridge, MA, 1996. MIT Press.

[27] H.-P. Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, New-York, 1981. 1995 – 2nd edition.

[28] P.D. Surry and N.J. Radcliffe. Formal algorithms + formal representations = search strategies. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Proceedings of the* 4th *Conference on Parallel Problems Solving from Nature*, number 1141 in LNCS, pages 366–375. Springer Verlag, 1996.