Algorithmes génétiques

Jean-Marc Alliot, Nicolas Durand

March 14, 2005

1 Principes généraux

Les algorithmes génétiques sont des algorithmes d'optimisation s'appuyant sur des techniques dérivées de la génétique et de l'évolution naturelle[1] : croisements, mutations, sélection, etc. Les algorithmes génétiques ont déjà une histoire relativement ancienne puisque les premiers travaux de John Holland sur les systèmes adaptatifs remontent à 1962 [Hol62]. L'ouvrage de David Goldberg [Gol89c] a largement contribué à les vulgariser.

Un algorithme génétique recherche le ou les extrema d'une fonction définie sur un espace de données. Pour l'utiliser, on doit disposer des cinq éléments suivants :

1. Un principe de codage de l'élément de population. Cette étape associe à chacun des points de l'espace d'état une structure de données. Elle se place généralement après une phase de modélisation mathématique du problème traité. La qualité du codage des données conditionne le succès des algorithmes génétiques. Le codage binaires ont été très utilisés à l'origine. Les codages réels sont désormais largement utilisés, notamment dans les domaines applicatifs pour l'optimisation de problèmes à variables réelles.

2. Un mécanisme de génération de la population initiale. Ce mécanisme doit être capable de produire une population d'individus non homogène qui servira de base pour les générations futures. Le choix de la population initiale est important car il peut rendre plus ou moins rapide la convergence vers l'optimum global. Dans le cas où l'on ne connaît rien du problème à résoudre, il est essentiel que la population initiale soit répartie sur tout le domaine de recherche.

3. Une fonction à optimiser. Celle-ci retourne une valeur de $\Re^+$ appelée *fitness* ou fonction d'évaluation de l'individu.

4. Des opérateurs permettant de diversifier la population au cours des générations et d'explorer l'espace d'état. L'opérateur de croisement recompose les gènes d'individus existant dans la population, l'opérateur de mutation a pour but de garantir l'exploration de l'espace d'états.

5. Des paramètres de dimensionnement : taille de la population, nombre total de générations ou critère d'arrêt, probabilités d'application des opérateurs de croisement et de mutation.

Le principe général du fonctionnement d'un algorithme génétique est représenté sur la figure 1 : on commence par générer une population d'individus de façon aléatoire. Pour passer d'une génération k à la génération $k+1$, les trois opérations suivantes sont répétées pour tous les éléments de la population k. Des couples de parents P_1 et P_2 sont sélectionnés en fonction de leurs adaptations. L'opérateur de croisement leur est appliqué avec une probabilité P_c (généralement autour de 0.6) et génère des couples d'enfants C_1 et C_2. D'autres éléments P sont sélectionnés en fonction de leur adaptation. L'opérateur de mutation leur est appliqué avec la probabilité P_m (P_m est généralement très inférieur à P_c) et génère des individus mutés P'. Le niveau d'adaptation des enfants ($C1$,C_2) et des individus mutés P' sont ensuite évalués avant insertion dans la nouvelle population. Différents critères d'arrêt de l'algorithme peuvent être choisis :

[1] Il est intéressant de trouver dans l'œuvre d'un spécialiste de zoologie, Richard Dawkins [Daw89], un exemple informatique tendant à prouver la correction de l'hypothèse darwinienne de la sélection naturelle. La méthode utilisée est presque semblable aux techniques génétiques.

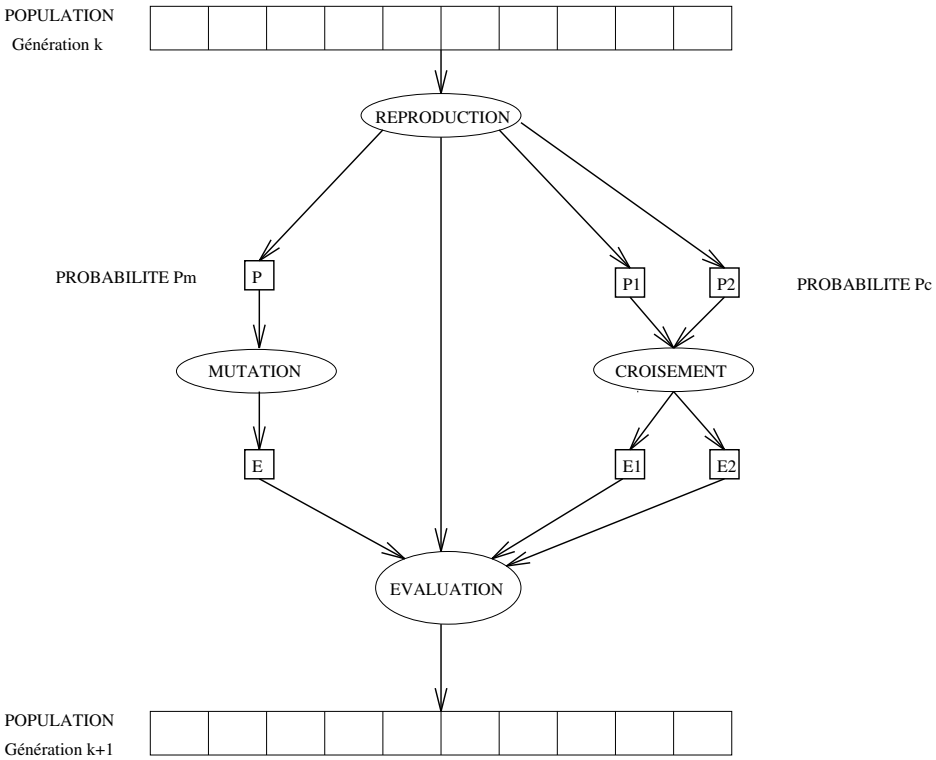

Figure 1: Principe général des algorithmes génétiques

- Le nombre de générations que l'on souhaite exécuter peut être fixé à priori. C'est ce que l'on est tenté de faire lorsque l'on doit trouver une solution dans un temps limité.
- L'algorithme peut être arrêté lorsque la population n'évolue plus ou plus suffisamment rapidement.

Nous allons maintenant détailler chacun de ces points.

2 Description détaillée

2.1 Codage des données

Historiquement le codage utilisé par les algorithmes génétiques était représenté sous forme de chaînes de bits contenant toute l'information nécessaire à la description d'un point dans l'espace d'état. Ce type de codage a pour intérêt de permettre de créer des opérateurs de croisement et de mutation simples. C'est également en utilisant ce type de codage que les premiers résultats de convergence théorique ont été obtenus.

Cependant, ce type de codage n'est pas toujours bon comme le montrent les deux exemples suivants :

- deux éléments voisins en terme de distance de Hamming ne codent pas nécessairement deux éléments proches dans l'espace de recherche. Cet inconvénient peut être évité en utilisant un codage de Gray.
- Pour des problèmes d'optimisation dans des espaces de grande dimension, le codage binaire peut rapidement devenir mauvais. Généralement, chaque variable est représentée par une partie de la chaîne de bits et la structure du problème n'est pas bien reflétée, l'ordre des variables ayant une importance dans la structure du chromosome alors qu'il n'en a pas forcément dans la structure du problème.

Les algorithmes génétiques utilisant des vecteurs réels [Gol91, Wri91] évitent ce problème en conservant les variables du problème dans le codage de l'élément de population sans passer par le codage binaire intermédiaire. La structure du problème est conservée dans le codage.

2.2 Génération aléatoire de la population initiale

Le choix de la population initiale d'individus conditionne fortement la rapidité de l'algorithme. Si la position de l'optimum dans l'espace d'état est totalement inconnue, il est naturel de générer aléatoirement des individus en faisant des tirages uniformes dans chacun des domaines associés aux composantes de l'espace d'état en veillant à ce que les individus produits respectent les contraintes [MJ91]. Si par contre, des informations à priori sur le problème sont disponibles, il parait bien évidemment naturel de générer les individus dans un sous-domaine particulier afin d'accélérer la convergence. Dans l'hypothèse où la gestion des contraintes ne peut se faire directement, les contraintes sont généralement incluses dans le critère à optimiser sous forme de pénalités. Il est clair qu'il vaut mieux, lorsque c'est possible ne générer que des éléments de population respectant les contraintes.

2.3 Gestion des contraintes

Un élément de population qui viole une contrainte se verra attribuer une mauvaise fitness et aura une probabilité forte d'être éliminé par le processus de sélection.

Il peut cependant être intéressant de conserver, tout en les pénalisant, les éléments non admissibles car ils peuvent permettre de générer des éléments admissibles de bonne qualité. Pour de nombreux problèmes, l'optimum est atteint lorsque l'une au moins des contraintes de séparation est saturée, c'est à dire sur la frontière de l'espace admissible.

Gérer les contraintes en pénalisant la fonction fitness est difficile, un "dosage" s'impose pour ne pas favoriser la recherche de solutions admissibles au détriment de la recherche de l'optimum ou inversement.

Disposant d'une population d'individus non homogène, la diversité de la population doit être entretenue au cours des générations afin de parcourir le plus largement possible l'espace d'état. C'est le rôle des opérateurs de croisement et de mutation.

2.4 Opérateur de Croisement

Le croisement a pour but d'enrichir la diversité de la population en manipulant la structure des chromosomes. Classiquement, les croisements sont envisagés avec deux parents et génèrent deux enfants.

Initialement, le croisement associé au codage par chaînes de bits est le croisement à découpage de chromosomes (slicing crossover). Pour effectuer ce type de croisement sur des chromosomes constitués de M gènes, on tire aléatoirement une position dans chacun des parents. On échange ensuite les deux sous-chaînes terminales de chacun des deux chromosomes, ce qui produit deux enfants C_1 et C_2 (voir figure 2).

On peut étendre ce principe en découpant le chromosome non pas en 2 sous-chaînes mais en 3, 4, etc [BG91]. (voir figure 3).

Ce type de croisement à découpage de chromosomes est très efficace pour les problèmes discrets. Pour les problèmes continus, un croisement "barycentrique" est souvent utilisé : deux gènes $P_1(i)$ et $P_2(i)$ sont sélectionnés dans chacun des parents à la même position i. Ils définissent deux nouveaux gènes $C_1(i)$ et $C_2(i)$ par combinaison linéaire :

$$\begin{cases} C_1(i) = \alpha P_1(i) + (1-\alpha) P_2(i) \\ C_2(i) = (1-\alpha) P_1(i) + \alpha P_2(i) \end{cases}$$

où α est un coefficient de pondération aléatoire adapté au domaine d'extension des gènes (il n'est pas nécessairement compris entre 0 et 1, il peut par exemple prendre des valeurs dans l'intervalle $[-0.5, 1.5]$ ce qui permet de générer des points entre, ou à l'extérieur des deux gènes considérés).

Dans le cas particulier d'un chromosome matriciel constitué par la concaténation de vecteurs, on peut étendre ce principe de croisement aux vecteurs constituant les gènes (voir figure 4) :

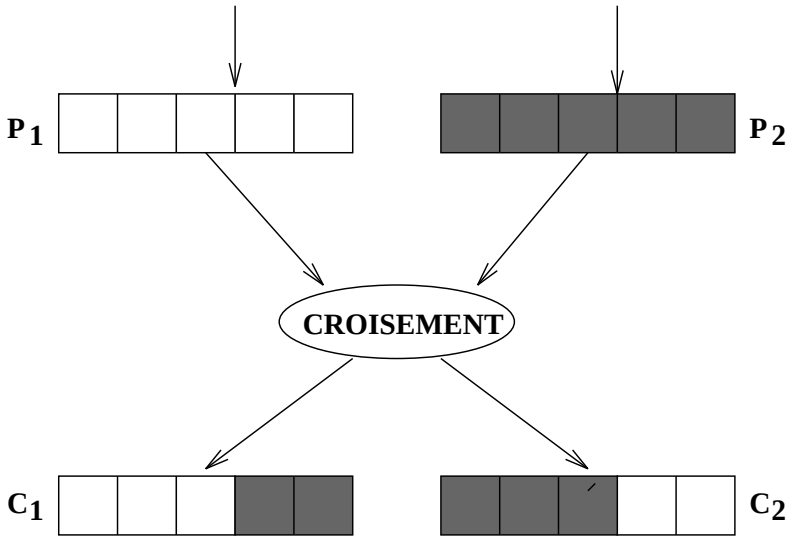

Figure 2: Slicing crossover

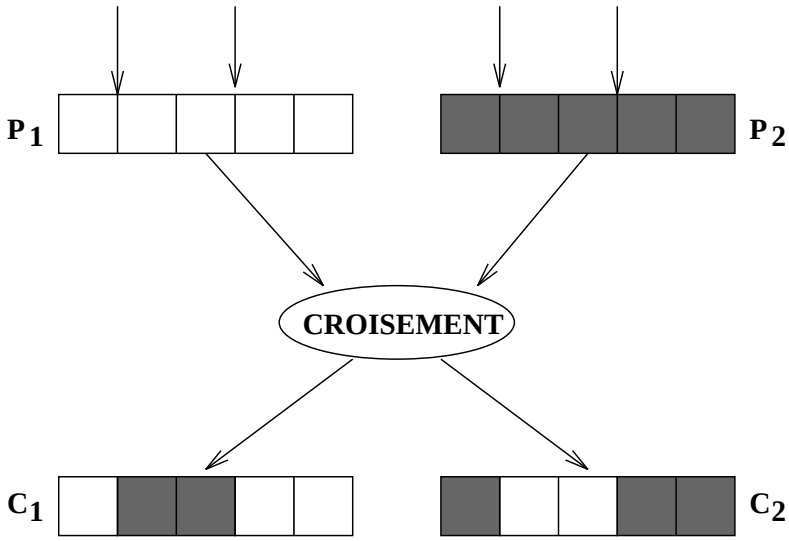

Figure 3: Slicing crossover à 2 points

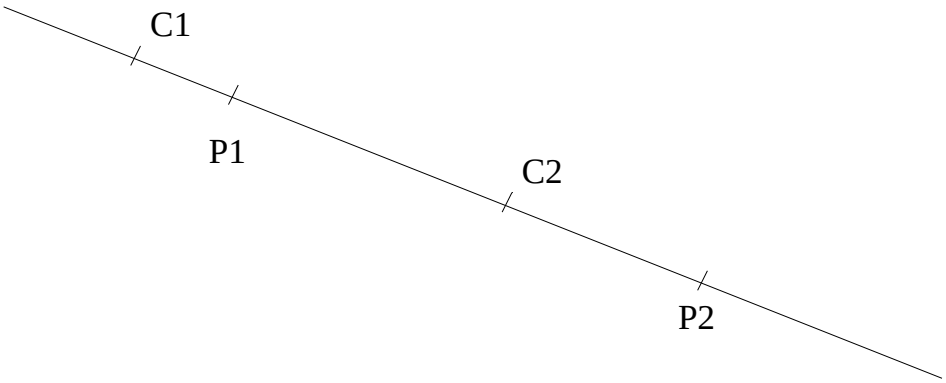

Figure 4: Croisement barycentrique

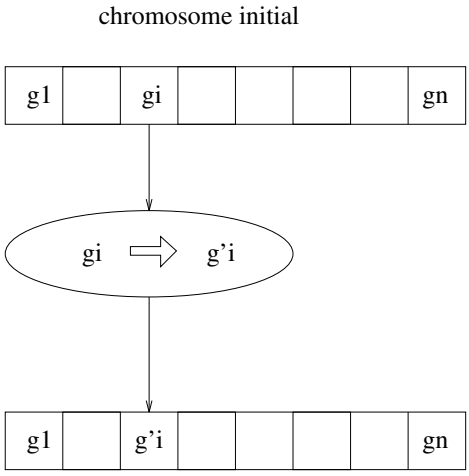

Figure 5: Principe de l'opérateur de mutation

$$\begin{cases} \vec{C}_1(i) = \alpha\vec{P}_1(i) + (1-\alpha)\vec{P}_2(i) \\ \vec{C}_2(i) = (1-\alpha)\vec{P}_1(i) + \alpha\vec{P}_2(i) \end{cases}$$

On peut imaginer et tester des opérateurs de croisement plus ou moins complexes sur un problème donné mais l'efficacité de ce dernier est souvent lié intrinsèquement au problème.

2.5 Opérateur de mutation

L'opérateur de mutation apporte aux algorithmes génétiques la propriété d'ergodicité de parcours d'espace. Cette propriété indique que l'algorithme génétique sera susceptible d'atteindre tous les points de l'espace d'état, sans pour autant les parcourir tous dans le processus de résolution. Ainsi en toute rigueur, l'algorithme génétique peut converger sans croisement, et certaines implantations fonctionnent de cette manière [FOW66]. Les propriétés de convergence des algorithmes génétiques sont donc fortement dépendantes de cet opérateur sur le plan théorique.

Pour les problèmes discrets, l'opérateur de mutation consiste généralement à tirer aléatoirement un gène dans le chromosome et à le remplacer par une valeur aléatoire (voir figure 5). Si la notion de distance existe, cette valeur peut être choisie dans le voisinage de la valeur initiale.

Dans les problèmes continus, on procède un peu de la même manière en tirant aléatoirement un gène dans le chromosome, auquel on ajoute un bruit généralement gaussien. L'écart type de ce bruit est difficile à choisir a priori. Nous discutons ce problème de façon plus détaillée, en présentant une amorce de solution, dans la section 4.2.

2.6 Principes de sélection

A l'inverse d'autres techniques d'optimisation, les algorithmes génétiques ne requièrent pas d'hypothèse particulière sur la régularité de la fonction objectif. L'algorithme génétique n'utilise notamment pas ses dérivées successives, ce qui rend très vaste son domaine d'application. Aucune hypothèse sur la continuité n'est non plus requise. Néanmoins, dans la pratique, les algorithmes génétiques sont sensibles à la régularité des fonctions qu'ils optimisent.

Le peu d'hypothèses requises permet de traiter des problèmes très complexes. La fonction à optimiser peut ainsi être le résultat d'une simulation.

La sélection permet d'identifier statistiquement les meilleurs individus d'une population et d'éliminer les mauvais. On trouve dans la littérature un nombre important de principes de sélection plus ou moins

adaptés aux problèmes qu'ils traitent. Dans le cadre de notre travail, les deux principes de sélection suivants ont été testés et évalués:

- *Roulette wheel selection* [Gol89c];
- *Stochastic remainder without replacement selection* [Gol89c];

Le principe de *Roulette wheel selection*[2] consiste à associer à chaque individu un segment dont la longueur est proportionnelle à sa fitness. On reproduit ici le principe de tirage aléatoire utilisé dans les roulettes de casinos avec une structure linéaire. Ces segments sont ensuite concaténés sur un axe que l'on normalise entre 0 et 1. On tire alors un nombre aléatoire de distribution uniforme entre 0 et 1, puis on "regarde" quel est le segment sélectionné. Avec ce système, les grands segments, c'est-à-dire les bons individus, seront plus souvent adressés que les petits. Lorsque la dimension de la population est réduite, il est difficile d'obtenir en pratique l'espérance mathématique de sélection en raison du peu de tirages effectués. Un biais de sélection plus ou moins fort existe suivant la dimension de la population.

La *Stochastic remainder without replacement selection* évite ce genre de problème et donne de bons résultats pour nos applications. Décrivons ce principe de sélection :

- Pour chaque élément i, on calcule le rapport r_i de sa fitness sur la moyenne des fitness.
- Soit $e(r_i)$ la partie entière de r_i, chaque élément est reproduit exactement $e(r_i)$ fois.
- La *roulette wheel selection* précédemment décrite est appliquée sur les individus affectés des fitness $r_i - e(r_i)$.

Compte-tenu du fait que des faibles populations seront utilisées par la suite, ce principe de sélection s'avèrera le plus efficace dans les applications pratiques et sera donc utilisé par la suite.

3 Améliorations classiques

3.1 Introduction

Les processus de sélection présentés sont très sensibles aux écarts de fitness et dans certains cas, un très bon individu risque d'être reproduit trop souvent et peut même provoquer l'élimination complète de ses congénères; on obtient alors une population homogène contenant un seul type d'individu. Ainsi, dans l'exemple de la figure 6 le second mode M_2 risque d'être le seul représentant pour la génération suivante et seule la mutation pourra aider à atteindre l'objectif global M_1 au prix de nombreux essais successifs.

Pour éviter ce comportement, il existe d'autres modes de sélection (*ranking*) ainsi que des principes (*scaling, sharing*) qui empêchent les individus "forts" d'éliminer complètement les plus "faibles". On peut également modifier le processus de sélection en introduisant des tournois entre parents et enfants, basé sur une technique proche du recuit.

Enfin, on peut également introduire des recherches multi-objectifs, en utilisant la notion de dominance lors de la sélection.

3.2 Scaling

Le *scaling* ou mise à l'échelle, modifie les fitness afin de réduire ou d'amplifier artificiellement les écarts entre les individus. Le processus de sélection n'opère plus sur la fitness réelle mais sur son image après scaling. Parmi les fonctions de scaling, on peut envisager le scaling linéaire et le scaling exponentiel. Soit f_r la fitness avant scaling et f_s la fitness modifiée par le scaling.

[2] Dans la littérature, cette méthode porte parfois le nom de méthode de Monte-Carlo.

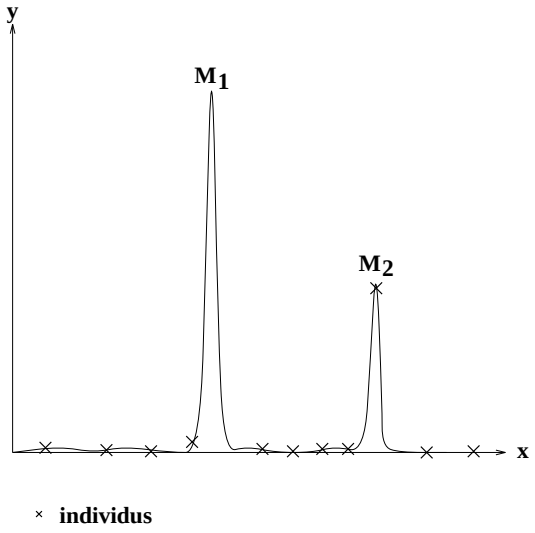

Figure 6: Exemple où les sélections classiques risquent de ne reproduire qu'un individu

3.2.1 Scaling linéaire

Dans ce cas la fonction de scaling est définie de la façon suivante [Mic92] :

$$f_s = af_r + b$$

$$a = \frac{max' - min'}{max - min};\ b = \frac{min'.max - min.max'}{max - min}.$$

En règle générale, le coefficient a est inférieur à un, ce qui permet de réduire les écarts de fitness et donc de favoriser l'exploration de l'espace. Ce scaling est statique par rapport au numéro de génération et pénalise la fin de convergence lorsque l'on désire favoriser les modes dominants.

3.2.2 Scaling exponentiel

Il est défini de la façon suivante [Mic92] (voir figure 7):

$$f_s = (f_r)^{k(n)}$$

où n est la génération courante.

- Pour k proche de zéro, on réduit fortement les écarts de fitness ; aucun individu n'est vraiment favorisé et l'algorithme génétique se comporte comme un algorithme de recherche aléatoire et permet d'explorer l'espace.
- Pour k proche de 1 : le scaling est inopérant.
- Pour $k > 1$ les écarts sont exagérés et seuls les bons individus sont sélectionnés ce qui produit l'émergence des modes.

Dans la pratique, on fait généralement varier k des faibles valeurs vers les fortes valeurs au cours des générations. Pour cela on peut utiliser la formule suivante :

$$k = \left(\tan\left[\left(\frac{n}{N+1}\right)\frac{\pi}{2}\right]\right)^p$$

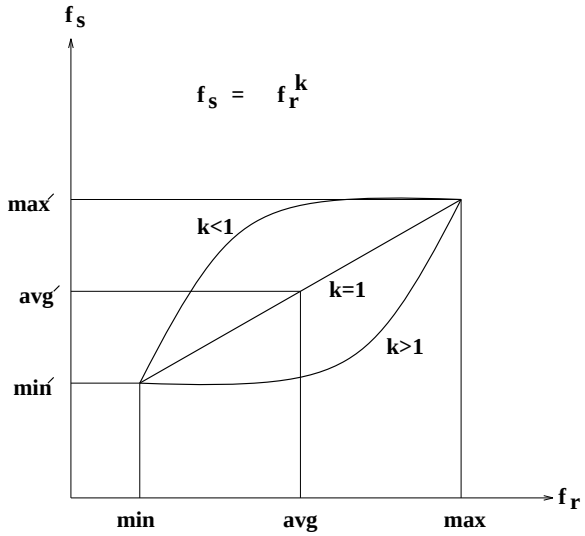

Figure 7: Fonction de scaling exponentielle

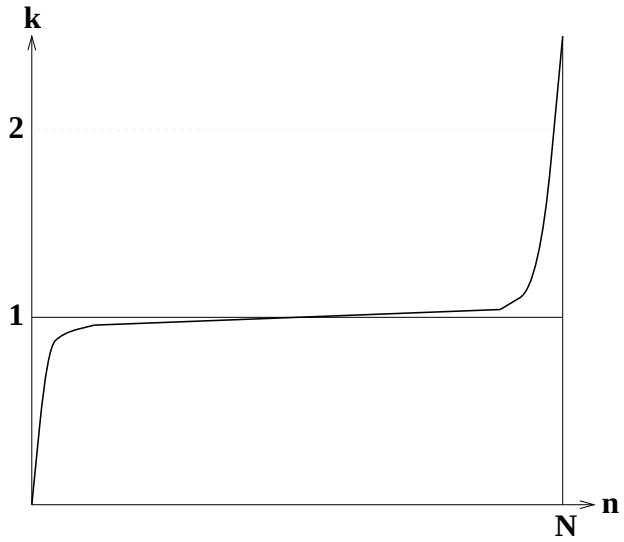

Figure 8: Allure de l'évolution de k en fonction des générations

SANS SHARING (sans mutation) **AVEC SHARING (sans mutation)**

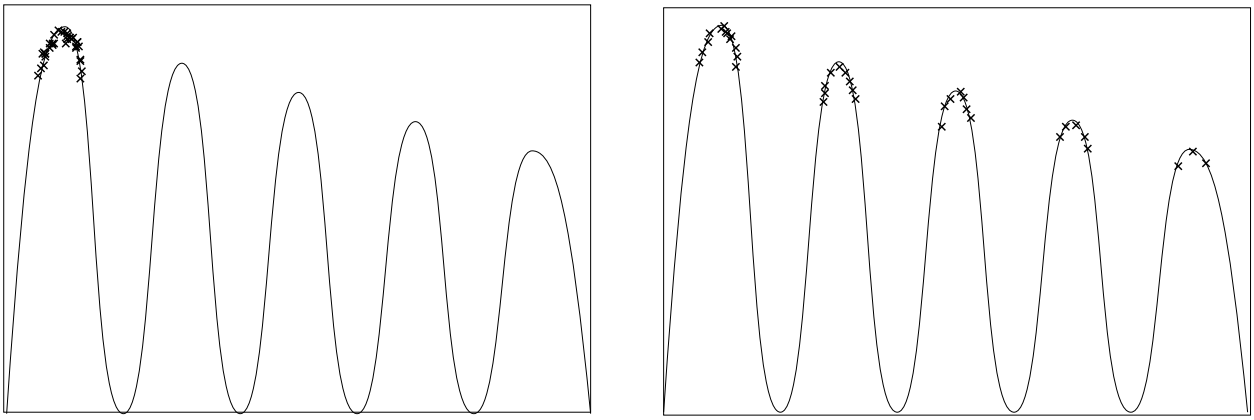

Figure 9: Objectif du sharing

n étant la génération courante, N le nombre total de générations prévues, p un paramètre à choisir. Le choix de $p = 0.1$ s'est avéré pertinent dans les applications. L'évolution de k en fonction de la génération n est donnée par la figure 8.

Ce dernier principe de scaling donne effectivement de meilleurs résultats sur nos problèmes que le scaling linéaire et sera donc systématiquement utilisé. Dans le cas des fonctions objectifs multi-modes présentant des optimaux quasi-équivalents, cette technique de scaling, en amplifiant les écarts de fitness en fin de convergence, va effectivement favoriser le mode dominant mais aussi masquer les modes sous-optimaux qui peuvent tout de même présenter un intérêt. Le scaling permet donc une bonne exploration de l'espace d'état mais ne favorise pas la répartition des individus sur les différents modes de la fonction objectif.

3.3 Sharing

3.3.1 Introduction

L'objectif du sharing est de répartir sur chaque sommet de la fonction à optimiser un nombre d'individus proportionnel à la fitness associée à ce sommet. La figure 9 présente deux exemples de répartitions de populations dans le cas d'une fonction à cinq sommets : le premier sans sharing, le second avec sharing.

3.3.2 Principe

De la même façon que le scaling, le sharing consiste à modifier la fitness utilisée par le processus de sélection. Pour éviter le rassemblement des individus autour d'un sommet dominant, le sharing pénalise les fitness en fonction du taux d'agrégation de la population dans le voisinage d'un individu. Il requiert l'introduction d'une notion de distance. Dans la pratique, il faut définir une distance indiquant la dissimilarité entre deux individus. Cette distance est alors utilisée pour calculer la nouvelle fitness de la façon suivante :

$$f'_i = \frac{f_i}{m'_i};\ m'_i = \sum_{j=1}^{N} S(d(x_i, x_j))$$

avec

$$S(d) = 1 - \left(\frac{d}{\sigma_{share}}\right)^{\alpha} \text{ si } d < \sigma_{share}$$

$$S(d) = 0 \text{ si } d > \sigma_{share}$$

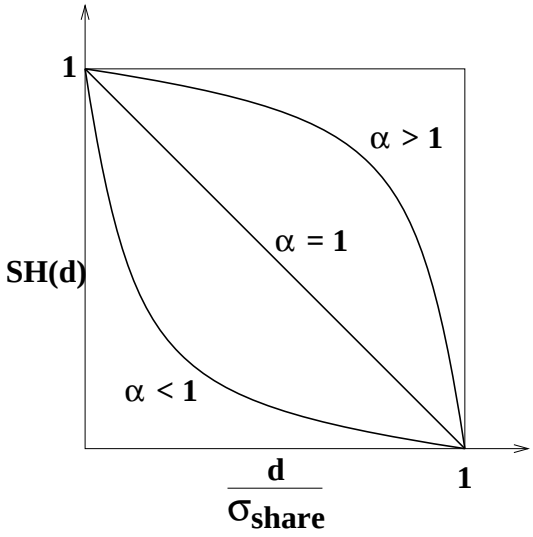

Figure 10: Allure de $S(\frac{d}{\sigma_{share}})$

Le paramètre σ_{share} permet de délimiter le voisinage d'un point et dépend du problème traité. La figure 10 donne l'allure de $S(d)$ pour différentes valeurs de α. Suivant la valeur donnée à α le sharing sera plus ou moins efficace. Ainsi pour $\alpha < 1$, on pénalise les groupes très agglomérés.

Dans la pratique ce type de sharing donne effectivement de bons résultats mais au prix de N^2 calculs de distances entre chromosomes à chaque génération pour une population de taille N. Or les algorithmes génétiques induisent une complexité en N sans sharing et le fait de passer en N^2 peut être pénalisant, notamment pour N grand.

Pour réduire ce nombre, on utilise un sharing "clusterisé".

3.4 Sharing clusterisé

Pour effectuer ce type de sharing [YG93], on commence par identifier les différents "clusters" d'individus dans la population. Ce dernier utilise deux paramètres d_{min} et d_{max} respectivement pour fusionner des clusters ou en créer de nouveaux. Initialement, chaque individu de la population est considéré comme le centre d'un cluster. On applique alors successivement les deux principes suivants :

- si deux centres sont à une distance inférieure à d_{min}, on fusionne ces derniers dans un cluster unique dont le centre résultant est le barycentre des deux centres initiaux.
- un nouvel individu est agrégé à un cluster si sa distance au centre le plus proche est inférieure à d_{max}. Dans ce cas, on recalcule le centre du cluster global. Sinon, on crée un nouveau cluster contenant ce seul individu.

Ce principe de fusion-agrégation permet d'obtenir un nombre de clusters fluctuant avec la répartition des individus dans l'espace d'état. On applique ensuite le principe de sharing en modifiant les fitness de la façon suivante :

$$f_i' = \frac{f_i}{m_i'};\ m_i' = n_c \left(1 - \left(\frac{d_{ic}}{2d_{max}}\right)^{\alpha}\right);$$

avec

- n_c : nombre d'individus contenus dans le cluster auquel appartient l'individu i.
- α : coefficient de sensibilité.

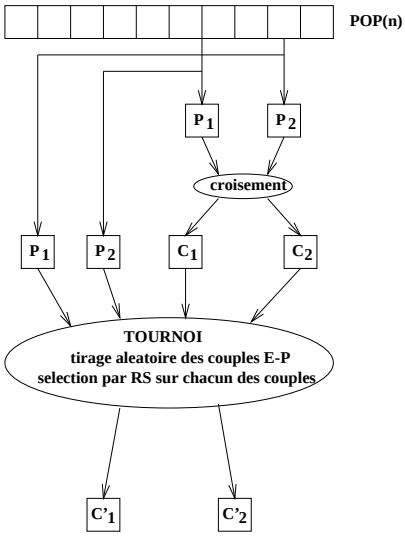

Figure 11: Principe du croisement avec recuit simulé

- d_{ic} : distance entre l'individu i et le centre du cluster c.

On montre que ce type de sharing induit une complexité en $O(N \log N)$[YG93] pour des résultats tout à fait comparables à ceux fournis par le sharing classique. Dans la pratique, on remarque que le réglage des coefficients d_{min} et d_{max} est assez délicat car l'efficacité de ces derniers dépend essentiellement de la connaissance a priori des distances inter-modes dans l'espace d'état, distance qu'il est très difficile d'estimer. Nous présentons dans la section 4.3 une technique permettant de calculer automatiquement ces quantités.

3.5 Algorithmes génétiques et recuit simulé

3.5.1 Introduction

Les algorithmes génétiques et le recuit simulé étant deux techniques d'optimisation stochastique travaillant sur les mêmes types de problèmes, il est logique de chercher à les associer afin de tirer partie de leurs avantages respectifs. Après plusieurs évaluations de ces deux techniques sur les mêmes problèmes test, on remarque que le recuit simulé converge généralement plus vite vers la solution optimale lorsque le problème est de taille raisonnable. Toutefois, il ne donne qu'une solution dans le cas des problèmes multi-modes, ceci confirme les résultats donnés dans [IR92]. A l'inverse, les algorithmes génétiques fournissent plusieurs solutions quasi-optimales mais au prix d'un temps de convergence généralement plus long. Il semble alors naturel d'associer ces deux techniques afin d'améliorer la convergence des algorithmes génétiques.

Il y a eu de nombreuses tentatives de fusion entre les algorithmes génétiques et le recuit simulé, les travaux les plus intéressants étant ceux de Mahfoud et de Goldberg [MG92].

3.5.2 Principe du croisement avec recuit simulé

Pour appliquer ce principe de croisement, on commence par sélectionner deux parents P_1 et P_2 dans la population (voir figure 11). On applique ensuite l'opérateur de croisement classique qui génère deux enfants C_1 et C_2. Un tournoi est alors effectué entre les parents et les enfants pour lequel les deux vainqueurs sont sélectionnés par le schéma de recuit suivant. On considère l'individu C_1. On tire ensuite aléatoirement un des deux parents, soit P_i ce parent :

- si C_1 est meilleur que $P_i \Rightarrow C_1$ est sélectionné.

- sinon C_1 est sélectionné avec la probabilité :

$$P = e^{-\left(\frac{|f_{C_1} - f_{P_i}|}{C_n}\right)}$$

où $c(n)$ est un coefficient décroissant en fonction de la génération courante (n).

On fait de même pour C_2 avec le parent restant et l'on détermine ainsi deux individus C_1' et C_2'.

L'évolution de la variable $C(n)$ se fait de la façon suivante. On utilise un schéma de recuit standard géométrique à un palier de basculement. Pratiquement, on calcule trois "températures" dont les valeurs dépendent de la connaissance des écarts min et max des fitness de la population initiale.

$$\begin{cases} C_s = -\frac{\Delta f_{max}}{ln\left(\frac{1}{k-1}\right)} & k = 0.75 \quad \text{"Température initiale"} \\ C_x = -\frac{\Delta f_{max}}{ln\left(\frac{1}{k-1}\right)} & k = 0.99 \quad \text{"Température de basculement"} \\ C_f = -\frac{\Delta f_{min}}{ln\left(\frac{1}{k-1}\right)} & k = 0.99 \quad \text{"Température finale"} \end{cases}$$

où $\Delta f_{min}, \Delta f_{max}$ représentent les écarts minimum et maximum des fitness de la population initiale. Le schéma géométrique fait évoluer la température courante de la façon suivante :

$$\begin{cases} C_0 = C_s \\ C_{n+1} = \alpha_1 C_n \text{ pour } C_s > C_n > C_x; \\ C_{n+1} = \alpha_2 C_n \text{ pour } C_x > C_n > C_f; \end{cases}$$

avec $0 < \alpha_1 < \alpha_2 < 1$.

Pour chaque palier, on calcule le nombre d'itérations de stabilisation à l'aide des formules :

$$N_1 = \frac{ln\left(\frac{C_x}{C_s}\right)}{ln\,\alpha_1} \quad N_2 = \frac{ln\left(\frac{C_f}{C_x}\right)}{ln\,\alpha_2}$$

Ces deux formules, nous permettent de calculer le nombre total de générations pour un problème donné.

Ce même principe de recuit simulé a été essayé sur l'opérateur de mutation en faisant un schéma de recuit entre l'individu muté et l'individu d'origine mais il ne produit pas l'effet escompté. En effet, on peut supposer que dans ce cas le recuit simulé réduit le brassage de la population provoqué par la mutation en limitant l'espace exploré aux zones qui améliorent statistiquement la fitness en "interdisant" les domaines qui la dégradent. L'exploration de du domaine admissible est fragilisée. Ce constat sur les problèmes que nous avons testés ne permet pas de généraliser.

3.6 Recherche multi-objectifs

3.6.1 Introduction

Dans le cadre de la recherche multi-objectifs, on cherche à optimiser une fonction suivant plusieurs critères, dont certains peuvent d'ailleurs être antagonistes. On définit alors la classique notion de dominance : on dit que le point A domine le point B si, $\forall i, f_i(A) > f_i(B)$, où les f_i représentent les critères à maximiser. L'ensemble des points qui ne sont dominés par aucun autre point forme la surface de Pareto. Tout point de la surface de Pareto est "optimal", dans la mesure où on ne peut améliorer la valeur d'un critère pour ce point sans diminuer la valeur d'au moins un autre critère.

Les algorithmes génétiques peuvent permettre de trouver l'ensemble de la surface de Pareto, car il est possible de répartir la population de l'algorithme génétique sur la dite surface.

3.6.2 Technique employée

La technique que nous employons dérive directement des travaux de Jeffrey Horn et Nicholas Nafpliotis ([HN93]). Le principal changement induit concerne le processus de sélection : en multi-objectifs, comment va t-on décider qu'un individu est meilleur qu'un autre[3] ? On introduit alors une variante de la notion de

[3] En ce qui concerne les autres opérateurs de base à savoir le croisement et la mutation il n'y a aucune modification car ceux-ci travaillent dans l'espace d'état et non dans l'espace résultat.

dominance que l'on définit ainsi : on peut par exemple décider que l'élément E_i domine E_j si le nombre des valeurs contenues dans son vecteur d'adaptation qui sont supérieures aux valeurs correspondantes dans E_j dépasse un certain seuil. A partir de là, la technique proposée pour effectuer la sélection est simple : on tire deux individus au hasard ainsi qu'une sous-population[4] à laquelle ils n'appartiennent pas et qui va servir à les comparer.

Trois cas se présentent alors :

- si le premier élément domine tous ceux de la sous-population et que ce n'est pas le cas pour le second alors le premier sera sélectionné.
- inversement si seul le second domine l'ensemble de la sous-population alors c'est lui qui sera conservé.
- restent maintenant deux possibilités : soit les deux sont dominés, soit les deux dominent. On ne peut se prononcer sans ajouter un autre test, c'est pourquoi dans ce cas il est fait usage d'un nouveau type de sharing qui opère sur l'espace objectif.

Le sharing va conduire à sélectionner celui des deux individus qui a le moins de voisins proches, autrement dit on élimine celui qui se situe dans une zone d'agrégation pour conserver celui qui est dans une région moins dense.

Encore une fois, le terme *voisin proche* n'a pas de signification précise mais il est possible de définir un voisinage (aussi appelé niche) correct en se servant de la distance de Holder :

$$d_H(E_i, E_j) = \left(\sum_{k=1}^{n} |f_i^k - f_j^k|^p \right)^{\frac{1}{p}}$$

f_i^k désignant la k-ième composante du vecteur adaptation de l'élément i. Le paramètre p permet de faire varier la forme et la taille du voisinage. A l'intérieur des voisinages ainsi définis dans l'espace objectif, il suffit de compter le nombre d'individus pour favoriser les zones les moins denses et de cette façon maintenir la diversité de la population. Ainsi la figure 12 montre comment les voisinages sont choisis autour des individus de la région de Pareto lorsque ceux-ci ne peuvent être départagés sans sharing.

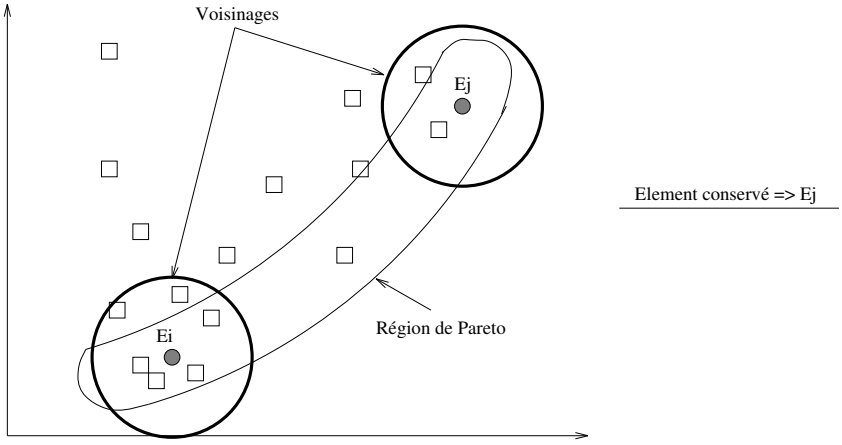

Figure 12: Surface de Pareto et voisinages

3.7 Association des AG avec des méthodes locales

La grande force des algorithmes génétiques est leur capacité à trouver la zone de l'espace des solutions contenant l'optimum de la fonction. En revanche, ils sont inefficaces lorsqu'il s'agit de trouver la valeur

[4]La sous-population tirée aura une taille proportionnelle à celle de la population de départ. Seule une partie des individus est utilisée, ce qui permet de réduire le temps de calcul.

exacte de l'optimum dans cette zone. Or, c'est précisément ce que les algorithmes locaux d'optimisation réalisent le mieux.

Il est donc naturel de penser à associer un algorithme local à l'algorithme génétique de façon à trouver la valeur exacte de l'optimum. On peut aisément le faire en appliquant à la fin de l'algorithme génétique un algorithme local sur le meilleur élément trouvé. Cette technique est d'autant plus efficace que l'on utilise simultanément du clustering, et que l'algorithme local est appliqué à chaque meilleur élément de chaque cluster. En effet, on constate souvent que le meilleur élément trouvé par l'algorithme génétique ne reste pas le meilleur élément après amélioration par l'algorithme local de tous les meilleurs éléments de clusters.

Une autre technique consiste à utiliser un algorithme local associé à l'algorithme génétique pour calculer la fitness d'un élément. On peut, par exemple dans un espace fortement combinatoire, rechercher avec l'algorithme génétique les zones intéressantes de l'espace en générant les éléments, l'adaptation de chaque élément étant calculée par un programme d'optimisation local (linéaire par exemple). Un exemple de ce type est donné dans la section **??**.

En fait, l'association AG–méthodes locales est une quasi-nécessité. Les deux méthodes sont complémentaires et ont des champs d'application différents. L'algorithme génétique permet de faire "disparaître" la combinatoire du problème, laissant alors le champ libre aux méthodes locales dans chacune des zones connexes qu'il pense susceptible de contenir l'optimum global.

4 Autres améliorations

Les raffinements décrits dans cette section ont été développés au sein de notre équipe, et sont donc, en principe, originaux. Notons que l'algorithme génétique que nous avons développé implante l'intégralité des techniques présentées dans la section précédente, celles présentées dans cette section, ainsi que les différents types de parallélismes décrits dans la section 5. Nous pensons qu'il s'agit d'un des programmes les plus efficaces actuellement.

4.1 Croisement adapté pour les fonctions partiellement séparables

4.1.1 Introduction

Dans beaucoup de problèmes d'optimisation, la fonction à optimiser dépend de nombreuses variables, et peut se décomposer comme somme de sous-fonctions prenant en compte une partie des variables seulement ([GT82]). Les algorithmes génétiques s'avèrent être de bons outils d'optimisation globale en raison de leur capacité à sortir des minima locaux. Néanmoins, leurs performances sont souvent pénalisées par le caractère très aléatoire des opérateurs de croisement et de mutation. Pour remédier à ce phénomène, certains évolutionnistes utilisent des heuristiques qui permettent de favoriser les "bons" croisements ou les "bonnes" mutations et d'éviter les "mauvaises". Nous présentons ici un opérateur de croisement efficace, utilisable dans des problèmes où la fonction à optimiser peut se décomposer en somme de sous-fonctions positives ne dépendant pas de toutes les variables du problème.

Après une présentation formelle du type de problèmes auxquels s'adresse cette méthode, nous présenterons l'opérateur de croisement. Une illustration de l'efficacité de cette méthode sera ensuite proposée au travers de plusieurs exemples simples comportant jusqu'à une cinquantaine de variables. Enfin, [DAN94] propose une application de cet outil au classique problème du Voyageur de Commerce, largement étudié dans la littérature, aussi bien pour son intérêt pratique qu'en raison de sa grande complexité [Bra90, GL85, GGRG85, HGL93, OSH89, WSF89]. En effet, pour tester la performance d'un algorithme d'optimisation globale, le problème du Voyageur de commerce, NP-Complet, permet d'offrir de nombreux exemples de très grande taille comportant beaucoup d'optima locaux dont on connaît les solutions.

4.1.2 Problèmes concernés

Il s'agit de problèmes ou fonctions partiellement séparables, à savoir des problèmes où la fonction F à optimiser dépend de plusieurs variables x_1, x_2, .., x_n, et peut se décomposer en somme de fonctions F_i positives ne dépendant pas de toutes les variables. On remarquera que bon nombre de fonctions multiplicatives peuvent se transformer en fonction additives grâce à des transformations simples. On s'intéresse donc

aux fonctions pouvant s'écrire :

$$F(x_1, x_2, .., x_n) \quad = \quad \sum_{i=1}^{m} F_i(x_{j_1}, x_{j_2}, .., x_{j_{n_i}})$$

Soit E_k l'ensemble des indices i tels que x_k soit une variable de F_i :

$$E_k \quad = \quad \{i / x_k \in variables\, de\, F_i\}$$

Pour définir notre opérateur de croisement, nous allons introduire pour chaque variable x_k sa "fitness locale" $G_k(x_1, x_2, .., x_n)$ définie comme suit :

$$G_k(x_k, x_1, x_2, .., x_n) \quad = \quad \sum_{i \in E_k} F_i(x_{j_1}, x_{j_2}, .., x_{j_{n_i}})$$

La fitness locale associée à une variable isole sa contribution. Le but de cette section est de montrer comment utiliser cette fitness locale pour définir un opérateur de croisement permettant de réduire la taille des populations et de converger plus rapidement vers la solution optimale. Ces opérateurs sont particulièrement efficaces lorsqu'on les combine avec le sharing.

4.1.3 L'opérateur de croisement

L'opérateur de croisement consiste, à partir de deux chromosomes parents, à créer deux nouveaux chromosomes.

Le codage que nous adopterons pour appliquer notre croisement consiste à représenter le chromosome par la liste brute des variables du problème. S'il s'agit de bits, nous aurons une liste de bits, s'il s'agit de variables réelles, nous aurons une liste de réels. On pourra éventuellement avoir un panachage de différents types de variables. L'idée intuitive est la suivante : dans le cas d'un problème complètement séparable, optimiser le problème global peut se réduire à l'optimisation de chaque variable séparément. Nous essayons d'adapter ce raisonnement au cas de fonctions partiellement séparables. La technique que nous proposons consiste donc à retenir pour chaque variable, lors de la conception des fils, celle des deux parents qui a la meilleure fitness locale, ceci à un facteur Δ près. Par exemple, dans le cas où l'on cherche un minimum pour F, pour la $k^{\text{ième}}$ variable, si :

$$G_k(pere_1) \quad < \quad G_k(pere_2) - \Delta$$

alors le fils 1 se verra affecter la variable x_k du père 1. Si par contre :

$$G_k(pere_1) \quad > \quad G_k(pere_2) + \Delta$$

il se verra affecter la variable x_k du père 2. Si enfin :

$$\|G_k(pere_1) - G_k(pere_2)\| \quad \leq \Delta$$

la variable x_k du fils 1 sera choisie aléatoirement.

Lors du croisement, on souhaite créer deux fils à partir de deux parents. Si l'on applique la même stratégie pour le fils 2 que celle décrite ci-dessus pour le fils 1, les deux fils risquent de se ressembler fortement, surtout si Δ est faible. Ceci n'est pas souhaitable. On peut donc pour le deuxième fils choisir une autre valeur de Δ ce qui permet d'être plus ou moins déterministe, ou alors, comme pour les techniques de croisement classiques choisir le complémentaire du fils 1, à savoir affecter au fils 2 la variable des deux pères non affectée au fils 1.

On peut remarquer qu'il est facile d'introduire un opérateur de mutation qui s'appuie sur le même principe et modifie avec une plus forte probabilité les variables ayant une mauvaise fitness locale.

4.2 Mutation adaptative

Un des gros inconvénients de l'algorithme génétique est sa mauvaise convergence locale. Il est possible de raffiner le comportement des AGs en utilisant une technique que nous qualifions de "mutation adaptative" dans tous les problèmes à variable réelle.

Le principe est le suivant : dans les problèmes à variable réelle, l'opérateur de mutation consiste généralement à ajouter un bruit gaussien à l'élément de population concerné. Le problème est de bien choisir ce bruit gaussien. S'il est trop petit, les déplacements dans l'espace sont insuffisants en début de convergence, et l'algorithme peut rester bloqué dans un optimum local. Si le bruit est trop fort, l'AG trouvera certes une zone contenant l'optimum, mais sera incapable de converger localement. Il s'agit donc de modifier le bruit au fil des générations. Pour ce faire, on calcule l'écart moyen pour chacune des coordonnées entre le meilleur élément à la génération n et tous les autres éléments : ce nombre est alors utilisé comme écart type du bruit gaussien sur la coordonnée en question à la génération $n+1$.

L'efficacité de cette méthode est démontrée dans la section **??**.

4.3 Clustering adaptatif

Nous avons vu dans la section 3.4 une technique de sharing clusterisé permettant de diminuer la complexité du sharing traditionnel. Cependant, comme nous l'avons souligné, les quantités d_{min} et d_{max} sont difficiles à calculer à priori. Nous présentons ici un algorithme permettant de calculer automatiquement ces quantités de façon adaptative pendant l'exécution de l'algorithme génétique.

Au départ on initialise le processus avec $d_{moy} = 0$ et $\Delta = 2.0$. Ensuite à chaque génération on utilise ce qui suit :

1. Calcul de d_{min} et d_{max} :
$$\begin{cases} d_{max} = \frac{d_{moy}}{\Delta} \\ d_{min} = \frac{d_{max}}{3} \end{cases}$$

2. Évaluation de la distance moyenne des individus par rapport aux centroïdes des clusters :
$$d_{moy} = \frac{\sum_{i=1}^{n} \left(\sum_{j=1}^{N_c} d(E_i, C_j) \right)}{nN_c}$$
si l'on note C_j le centre du groupe j et N_c le nombre total de clusters.

3. on compte ensuite le nombre de clusters dont le meilleur individu a une fitness supérieure à un certain pourcentage[5] de celle du meilleur élément de la population , on le note N_{opt}

4. on met à jour le paramètre Δ en utilisant la règle :
$$\begin{cases} \Delta = \Delta * 1.05 \text{ si } \frac{N_{opt}}{N_c} > S_1 \text{ et } \Delta < 100 \\ \Delta = \Delta * 0.95 \text{ si } \frac{N_{opt}}{N_c} < S_2 \text{ et } \Delta > 1 \\ \Delta \text{ reste inchangé dans les autres cas} \end{cases}$$

Les valeurs S_1 et S_2 désignent deux seuils : si beaucoup de clusters sont suffisamment bons (c'est à dire quand leur proportion dépasse S_1) alors on diminue d_{min} et d_{max} pour accroître le nombre total de groupes et ainsi chercher d'autres optima. Au contraire, s'ils sont peu nombreux alors on va diminuer la quantité de clusters en augmentant les deux distances afin de rechercher des zones intéressantes de l'espace d'état. Les valeurs utilisées habituellement pour les seuils sont $S_1 = 0.85$ et $S_2 = 0.75$.

Cette technique donne d'excellents résultats pour rechercher les optima multiples de fonctions réelles.

[5] Egal au taux de sharing.

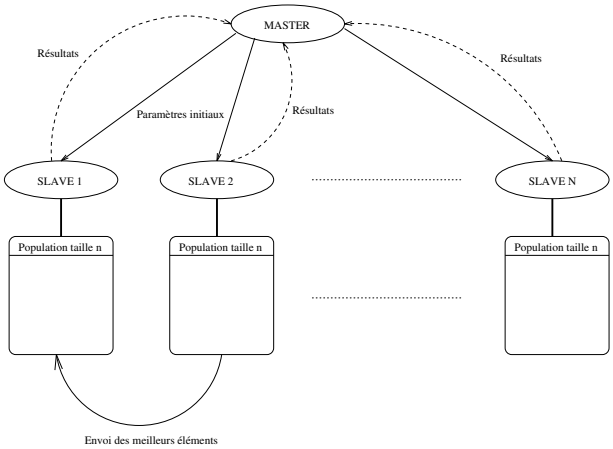

Figure 13: Principe de fonctionnement du parallélisme par îlots

5 Parallélisme

L'intérêt de la parallélisation des algorithmes génétiques est de gagner en temps de calcul. Il existe pour cela au moins deux méthodes utilisées classiquement (on pourra se reporter à [BD93],[CJ91],[Muh89],[PLG87] et [SPF93] pour plus de précisions sur les modèles de parallélisme utilisables dans les AG) :

- la première consiste à diviser la population de taille n en N sous-populations et à les répartir sur l'ensemble des machines dont on dispose.
- la seconde maintient la population totale sur une seule machine mais se sert des autres pour y envoyer les évaluations afin qu'elles se fassent en même temps.

Dans les deux cas il est nécessaire d'avoir un mécanisme de communication inter-processus. Les résultats présentés ici ont été obtenus en utilisant PVM. PVM a été réalisé par le Oak Ridge National Laboratory. Il permet à un réseau hétérogène de machines d'apparaître comme une seule entité ayant plusieurs processeurs capables de communiquer entre eux (comparable à un ordinateur à mémoire distribuée). La communication entre les divers composants de cette machine virtuelle se fait par l'envoi de paquets contenant des données ou encore des messages de contrôle. Pour plus de détails, on peut se reporter à [GBD+94].

5.1 Parallélisme par îlots

L'idée ici est de faire fonctionner plusieurs algorithmes génétiques en parallèle avec une population *réduite* pour chacun. Le programme maître lance N occurrences du programme d'algorithmes génétiques appelées esclaves sur des machines différentes en passant à chacune les paramètres nécessaires à leur bon fonctionnement comme le montre la figure 13.

Ensuite chaque processus fait évoluer sa population indépendamment jusqu'à ce qu'il décide (selon une probabilité fixée à l'avance) de rassembler ses meilleurs individus pour en transmettre une certaine quantité (proportionnelle à la taille de la population) à une autre machine de son choix. La machine réceptrice intègre alors ces nouveaux éléments dans sa propre population en éliminant les moins bons de ses individus. L'intérêt du parallélisme par îlots est qu'il offre la possibilité de travailler sur de grandes populations (n_0) tout en donnant des résultats dans un temps raisonnable puisque la durée nécessaire est à peu de choses près celle qu'il faudrait pour une population de taille $\frac{n_0}{N}$ si N est le nombre d'ordinateurs disponibles et si l'on néglige les temps de communication.

La méthode introduit un clustering forcé car chaque îlot peut être considéré comme un cluster subdivisé en petits groupes. Chaque machine a la possibilité de converger vers des optima qui seront différents de ceux calculés sur les autres ce qui correspond au comportement introduit avec le clustering. D'autre part, le surcoût de temps passé pour les communications n'est a priori pas excessif puisqu'il n'y a de gros paquets

à transmettre que de temps en temps. Il faut tout de même garder à l'esprit qu'une subdivision en sous-populations de taille trop réduite risque de conduire à faire tourner des algorithmes génétiques non fiables statistiquement. En effet, il faut quand même qu'une population contienne suffisamment d'individus pour que l'espace d'état puisse être exploré de façon correcte afin que les résultats aient une certaine valeur.

Des tests ont été faits par Yann LeFablec [LeF95]pour déterminer l'efficacité de la méthode (voir figure 14). Il ne faut s'attacher qu'à la forme générale de la courbe, dans la mesure où les charges locales des ma-

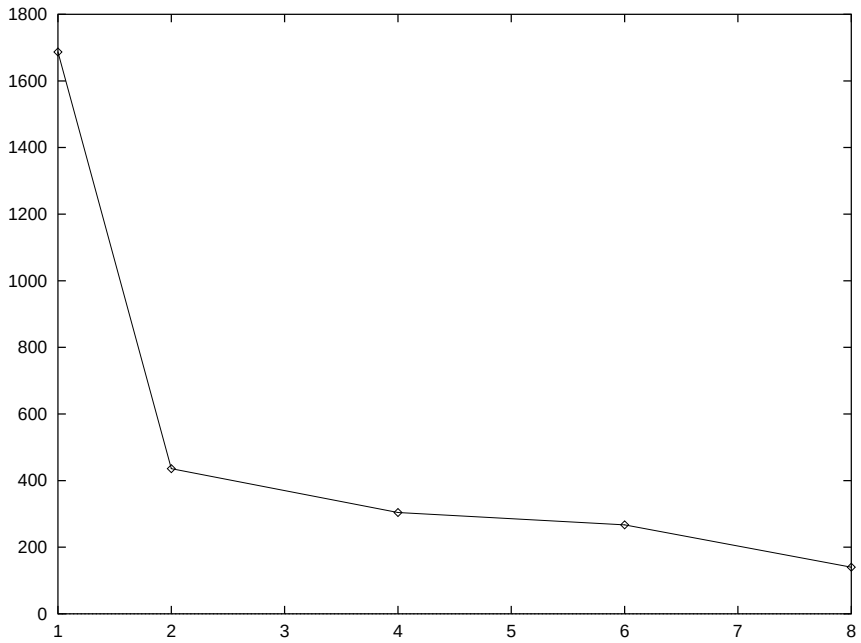

Figure 14: Évolution des temps de calcul

chines peuvent modifier de façon sensible les résultats. L'efficacité de la méthode est cependant largement mise en évidence.

5.2 Parallélisation des calculs

Contrairement à la méthode qui vient d'être décrite qui divise la population totale, on utilise ici des démons de calcul de fitness dont la seule fonction est de recevoir un individu et de renvoyer son adaptation. Pour utiliser la puissance de calcul parallèle offerte de manière optimale, il faut retarder au maximum les calculs pour les envoyer en blocs aux démons et faire en sorte qu'aucun ne reste inactif. Le principe se trouve résumé sur la figure 15 : le programme maître se charge de faire la sélection, les croisements, etc... en d'autres termes il fait évoluer la population, puis répartit les calculs dont il a besoin sur l'ensemble des démons. Enfin, dès qu'il a reçu tous les résultats, l'algorithme commence une nouvelle génération.

Il faut noter que ce mécanisme demande un grand nombre de communications pour envoyer les données et les évaluations. La méthode n'est donc intéressante que si le temps passé pour un calcul d'adaptation est grand devant le temps de communication, elle sera par conséquent utilisée pour des problèmes dont les évaluations de fitness prennent du temps, on pense essentiellement à des cas faisant appel à des réseaux de neurones ou à de gros calculs matriciels.

La courbe 16 montre comment évolue le temps de calcul en fonction du nombre de machines et du temps de communications. On constate que plus on utilise de machines, plus la pénalisation due aux communications est forte.

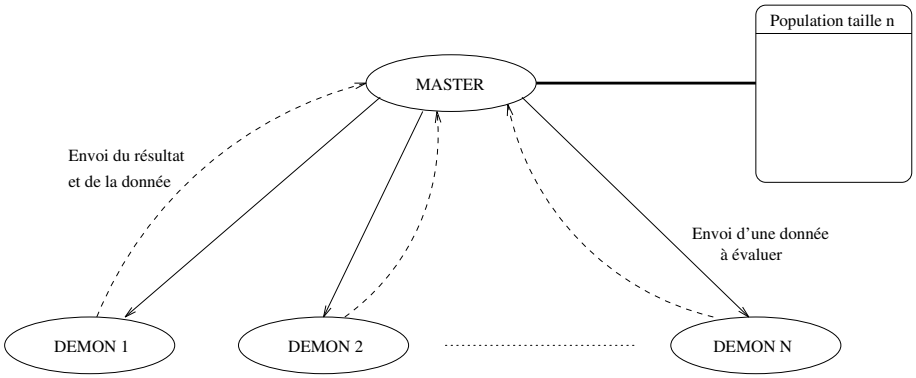

Figure 15: Principe de fonctionnement de la parallélisation des calculs

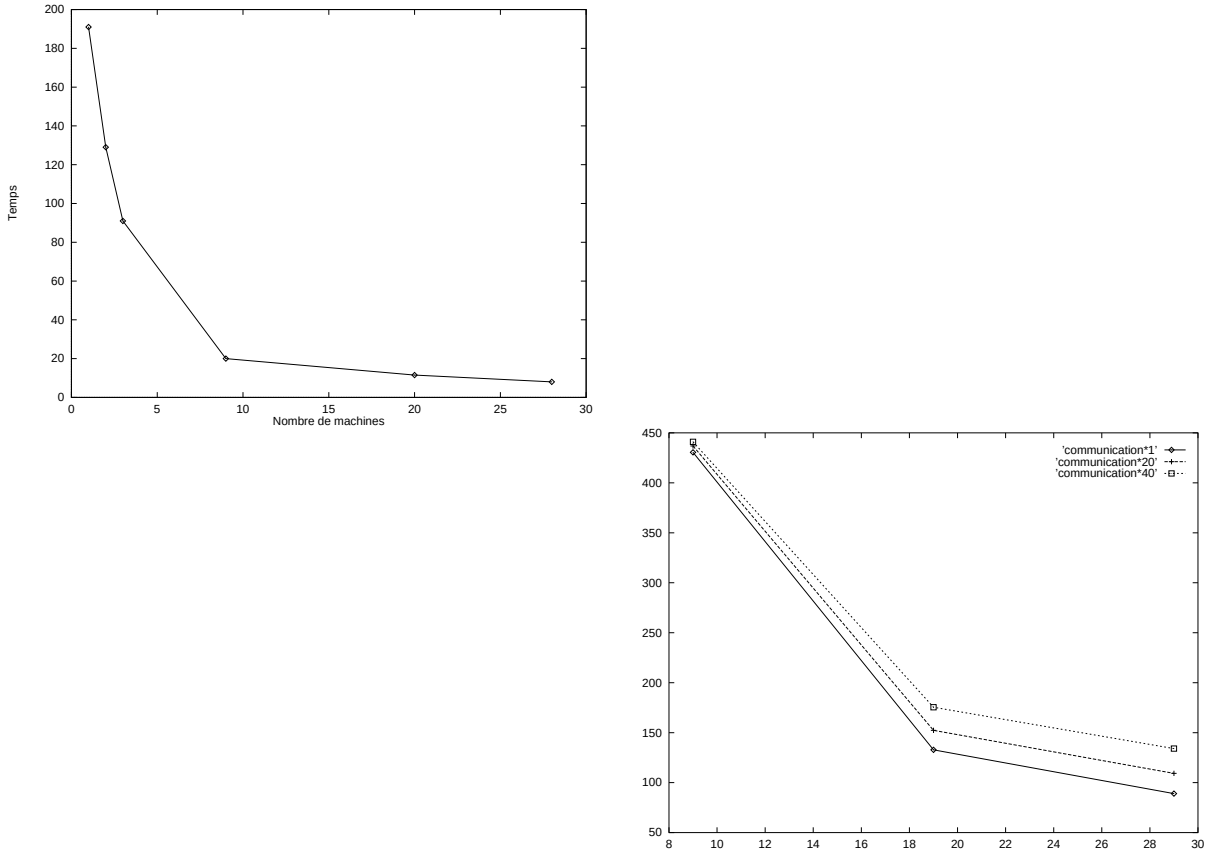

Figure 16: Évolution du temps de calcul

5.3 Conclusion

La parallélisation est extrêmement efficace pour accélérer les temps de résolution des algorithmes génétiques. Il faut certes bien étudier le problème afin d'utiliser le bon mécanisme de parallélisme, mais les gains en temps sont alors importants.

6 Résultats théoriques sur les algorithmes binaires

Historiquement, les algorithmes génétiques binaires sont les plus anciens et ont donc été les plus étudiés sur le plan théorique. Goldberg [Gol89a] a largement développé cette théorie résumée dans [AS92]. Nous allons tout d'abord les étudier avant de présenter les modélisations théoriques basées sur les chaînes de Markov.

6.1 Définitions fondamentales

Définition 1 (Séquence). *On appelle séquence A de longueur $l(A)$ une suite $A = a_1 a_2 \dots a_l$ avec $\forall i \in [1, l], a_i \in V = \{0, 1\}$.*

En codage binaire, les chromosomes sont des séquences.

Définition 2 (Schéma). *On appelle schéma H de longueur l une suite $H = a_1 a_2 \dots a_l$ avec $\forall i \in [1, l], a_i \in V^+ = \{0, 1, *\}$.*

Une $*$ en position i signifie que a_i peut être indifféremment 0 ou 1.

Définition 3 (Instance). *Une séquence $A = a_1 \dots a_l$ est une instance d'un schéma $H = b_1 \dots b_l$ si pour tout i tel que $b_i \neq *$ on a $a_i = b_i$.*

Ainsi, $H = 010 * 0101$ est un schéma et les séquences 01000101 et 01010101 sont des instances de H (ce sont même les seules).

Définition 4 (Position fixe, position libre). *Soit un schéma H. On dit que i est une position fixe de H si $a_i = 1$ ou $a_i = 0$. On dit que i est une position libre de H si $a_i = *$.*

Définition 5 (Ordre d'un schéma). *On appelle ordre du schéma H le nombre de positions fixes de H. On note l'ordre de H $o(H)$.*

Par exemple, le schéma $H = 01 * *10 * 1$ a pour ordre $o(H) = 5$, le schéma $H' = * * * * * 101$ a pour ordre $o(H') = 3$. On remarquera qu'un schéma H de longueur $l(H)$ et d'ordre $o(H)$ admet $2^{(l(H)-o(H))}$ instances différentes.

Définition 6 (Longueur fondamentale). *On appelle longueur fondamentale du schéma H la distance séparant la première position fixe de H de la dernière position fixe de H. On note cette longueur fondamentale $\delta(H)$.*

Ainsi, le schéma $H = 1 * *01 * **$ a pour longueur fondamentale $\delta(H) = 5 - 1 = 4$, le schéma $H' = 1 * * * * * * 1$ a pour longueur fondamentale $\delta(H') = 8 - 1 = 7$, et pour le schéma $H'' = * * 1 * * * * *$ nous avons $\delta(H'') = 3 - 3 = 0$.

Définition 7 (Adaptation d'une séquence). *On appelle adaptation d'une séquence A une valeur positive que nous noterons $f(A)$.*

f est la fonction objectif ou fitness du problème à résoudre.

Définition 8 (Adaptation d'un schéma). *On appelle adaptation d'un schéma H la valeur*

$$f(H) = \frac{\sum_{i=1}^{2^{(l(H)-o(H))}} f(A_i)}{2^{(l(H)-o(H))}}$$

où les A_i décrivent l'ensemble des instances de H, c'est-à-dire la moyenne des adaptations de ses instances.

6.2 Effets de la reproduction

Soit un ensemble $S = \{A_1, \ldots, A_i, \ldots, A_n\}$ de n séquences de bits tirées aléatoirement. Durant la reproduction, chaque séquence A_i est reproduite avec une probabilité :

$$p_i = \frac{f(A_i)}{\sum_{i=1}^{n} f(A_i)}$$

Supposons qu'il y ait à l'instant t un nombre $m(H,t)$ de séquences représentant le schéma H dans la population S. A l'instant $t+1$, statistiquement, ce nombre vaut :

$$m(H,t+1) = m(H,t).n.\frac{f(H)}{\sum_{i=1}^{n} f(A_i)}$$

Posons

$$\overline{f_t} = \frac{\sum_{i=1}^{n} f(A_i)}{n}$$

$\overline{f_t}$ représente la moyenne de l'adaptation des séquences à l'instant t. La formule précédente devient :

$$m(H,t+1) = m(H,t)\frac{f(H)}{\overline{f_t}}$$

Posons $c_t(H) = \frac{f(H)}{\overline{f_t}} - 1$. On obtient alors :

$$m(H,t+1) = (1 + c_t(H))m(H,t)$$

Il est donc clair qu'un schéma, dont l'adaptation est au-dessus de la moyenne, voit son nombre de représentants augmenter, suivant une progression qui est de type géométrique si nous faisons l'approximation que $c_t(H)$ est constant dans le temps. Alors :

$$m(H,t) = (1 + c(H))^t.m(H,0)$$

Si la reproduction seule était en jeu, les schémas forts élimineraient très rapidement les schémas faibles.

6.3 Effets des croisements

Nous allons nous intéresser à la probabilité de survie $p_s(H)$ d'un schéma H lors d'une opération de croisement. Par exemple, soit le schéma $H = **10*1**$. Supposons qu'une instance de ce schéma soit croisée avec une autre instance. Quelle est la probabilité pour que la séquence résultante soit encore une instance de H ? Il est impossible de répondre exactement à la question, tout au plus peut-on donner une borne inférieure de cette valeur. Il est clair que H ne sera pas détruit si le site de croisement qui est tiré au sort est inférieur à 3 (avant le premier 1) ou s'il est supérieur à 6 (après le dernier 1)[6].

On voit donc immédiatement qu'une borne inférieure de la probabilité de détruire un schéma H est $\delta(H)/(l-1)$. Donc la probabilité de survie dans un croisement est $1 - \delta(H)/(l-1)$. Si d'autre part on ne croise qu'une fraction p_c de séquences dans une population donnée, la probabilité de survie est donnée par :

$$p_s \geq 1 - p_c\frac{\delta(H)}{l-1}$$

De ce résultat et des résultats précédents découle la loi d'évolution d'une population[7] :

$$m(H,t+1) \geq m(H,t)(1 + c_t(H))(1 - p_c\frac{\delta(H)}{l-1})$$

[6] Dans tous les autres cas, H peut être (ou ne pas être) détruit.

[7] Les croisements et les mutations se font sur la population reproduite, et non sur la population initiale.

6.4 Effets des mutations

Soit p_m la probabilité de mutation d'un bit dans une séquence. Dans un schéma H, seules les positions fixes peuvent être détruites. La probabilité de survie d'un bit étant $1 - p_m$, la probabilité de survie d'un schéma H contenant $o(H)$ positions fixes est $(1-p_m)^{o(H)}$. La probabilité de mutation étant supposée petite devant 1, un développement limité au premier ordre donne une probabilité de survie égale à $1 - o(H)p_m$.

L'équation finale s'écrit donc :

$$m(H, t+1) \geq m(H, t)(1 + c_t(H))(1 - p_c \frac{\delta(H)}{l-1} - o(H)p_m)$$

6.5 Conclusion sur le codage binaire

Des calculs précédents découlent deux résultats :

- les schémas dont la longueur fondamentale est petite sont plus favorisés que les autres, lors de la génération d'une nouvelle population.
- les schémas dont l'ordre est petit sont plus favorisés que les autres, lors de la génération d'une nouvelle population.

Ces résultats conditionnent la qualité du codage des données. En effet, les schémas qui codent les données "intéressantes" pour le problème considéré doivent avoir un ordre et une longueur fondamentales faibles[8], alors que les données "sans intérêt" doivent être codées par des schémas qui ont un ordre et une longueur fondamentale élevés.

Les résultats présentés ci-dessus ne sont qu'une introduction à la théorie des schémas, il existe de nombreux approfondissements. On trouvera dans les références suivantes les principaux modèles théoriques sur la théorie des schémas et ses extensions :[Gol89c, Vos91, BM93, Gol89b, BG87, CS88, BM94, DM92, SGE91]

7 Modélisation par chaîne de Markov

Cette dernière approche est la plus satisfaisante tant sur le plan mathématique, que sur celui de la modélisation, les différents opérateurs étant présentés comme "perturbant" un processus Markovien représentant la population à chaque étape. Ici encore il apparaît que seul l'opérateur de mutation est important, le croisement pouvant être totalement absent.

Les principaux résultats asymptotiques portant directement sur les algorithmes génétiques, ont été développés par R. Cerf [Cer94] sur la base des travaux de Catoni [Cat90] et de Trouvé [Tro93]. Ces travaux se fondent sur la théorie des petites perturbations aléatoires d'un processus dynamique de type Markovien. Plus particulièrement, la théorie de Freidlin et Wentzell [FW83] constitue la pierre d'angle de ces études. Nous donnons ici, quelques uns des résultats de la dynamique des algorithmes génétiques développés par Cerf. Nous les présentons simplifiés et dans un cadre restreint, laissant le lecteur intéressé se reporter à la difficile lecture des références citées.

Nous travaillerons sur la base d'un codage binaire, P représentant le nombre de bits utilisés pour le codage. La fonction d'évaluation, f est définie sur l'espace $E = \{0, 1\}^P$ à valeurs dans $\Re^+$. Le problème est donc de localiser l'ensemble des maxima globaux de f , ou, à défaut, de trouver rapidement et efficacement des régions de l'espace où se situent ces maxima.

Comme nous l'avons vu, l'algorithme génétique est un algorithme stochastique itératif qui opère sur des ensembles de points, et qui est bâti à l'aide de trois opérateurs: mutation, croisement et sélection, que nous présentons plus formellement à présent.

[8] Les données intéressantes sont bien entendu les données qui sont proches de la solution optimale. Un bon codage des données implique donc d'avoir une *idée* de la forme de l'optimum.

7.1 Description rapide d'un algorithme génétique

Soit N la taille (fixe) de la population, notons X_k la population de la génération k : il s'agit d'une matrice $X_k = (X_k^1, X_k^2, .. X_k^N)$ de E^N dont les N éléments sont des chaînes de bits (chromosomes) de taille P. Le passage de la génération k à la génération $k+1$, c'est à dire de X_k à X_{k+1} se décompose en trois étapes :

$$X_k \overset{Mutation}{\longrightarrow} Y_k \overset{Croisement}{\longrightarrow} Z_k \overset{Selection}{\longrightarrow} X_{k+1}$$

Chacune de ces étapes peut être modélisée formellement.

- Mutation $X_k \longrightarrow Y_k$:

  L'opérateur considéré ici est le suivant : pour chaque composante de chaque élément X_k^i, une variable de Bernouilli de paramètre P_m est tirée indépendamment et suivant le résultat l'élément binaire examiné est changé ou non. (0 est changé en 1 et 1 en 0).

  La probabilité P_m de mutation doit être préalablement choisie et est généralement faible.

  Comme nous le verrons par la suite, cet opérateur joue un rôle clé dans la convergence de l'algorithme génétique.

- Croisement $Y_k \longrightarrow Z_k$:

  L'opérateur étudié ici est l'opérateur à un point (slicing crossover). Ici encore, la probabilité de croisement P_c est fixée initialement. Pour construire la population Z_k, $N/2$ couples sont formés à partir de la population Y_k (par exemple en appariant les individus consécutifs de Y_k, ou bien en choisissant au hasard et uniformément des individus dans Y_k). Pour chaque couple, une variable de Bernoulli de paramètre P_c est tirée pour décider si le croisement a lieu. Si c'est le cas, un site de coupure est tiré au hasard, et les segments finaux des deux chromosomes sont échangés.

  Une nouvelle paire d'individus est ainsi obtenue (identique à l'ancienne s'il n'y a pas eu de croisement) et est stockée dans la population Z_k. En général, le paramètre P_c est choisi grand.

  Remarquons que les opérateurs de mutation et de croisement ne font pas intervenir la fonction f, ce sont des opérateurs stochastiques d'exploration. C'est le troisième et dernier opérateur, la sélection, qui guide la population vers les valeurs élevées de la fonction f.

- Sélection $Z_k \longrightarrow X_{k+1}$

  Les N individus de la population X_{k+1} sont obtenus après sélection des individus de Z_k. On conserve ainsi les "meilleurs" individus de Z_k, indépendamment à l'aide d'une distribution de probabilité qui favorise les individus de Z_k les mieux adaptés.

  Le choix le plus fréquent est l'unique distribution telle que la probabilité d'un individu soit proportionnelle à son adaptation, i.e la probabilité de sélection de l'individu Z_k^i est :

$$P_i = P(Z_k^i) = \frac{f(Z_k^i)}{\sum_{j=1}^{N} f(Z_k^j)}$$

  En tirant les individus dans la population Z_k conformément aux probabilités P_i, on constitue la nouvelle génération X_{k+1}.

7.2 Modélisation

La présentation rapide des opérateurs nous permet de modéliser la suite des $(X_k)_{k\in\mathbb{N}}$ en une chaîne de Markov, d'espace d'états $E = \left(\{0,1\}^P\right)^N$. L'algorithme génétique ne doit donc pas être interprété comme une procédure d'optimisation mais plutôt comme une marche aléatoire dans l'espace d'état, attirée vers les fortes valeurs de f .

La propriété première de cette formalisation est que la loi de X_k est déterminée de manière unique par :

- la loi de la génération initiale X_0
- le mécanisme de transition de X_k à X_{k+1}, mécanisme scindé en trois étapes détaillée précédemment.

Ce mécanisme de transition possède toutefois des propriétés essentielles qui font l'intérêt et la puissance de cette formalisation, (voir [Cer94]) :

- Il est homogène (c'est à dire indépendant de la génération, k, considérée)
- Il est irréductible, (la probabilité de joindre deux points quelconques de l'espace d'état, en un nombre fini de générations est non nulle) soit :

$$\forall x, y \in E \qquad \exists r \in \mathbb{N} \qquad P\left[X_{k+r} = y \mid X_k = x\right] > 0$$

  Le mécanisme permet donc d'explorer tout point de l'espace d'état, avec une probabilité non nulle.
- Il est apériodique, cette hypothèse n'est cependant pas fondamentale.

Ces propriétés permettent de conclure à l'ergodicité de la chaîne de Markov, et à l'existence d'un processus limite.

Théorème 1. *Une chaîne de Markov homogène irréductible apériodique d'espace d'états fini est ergodique et possède une unique mesure de probabilité stationnaire ou invariante.*

Cette mesure stationnaire correspond à la loi régissant l'équilibre du processus, elle est définie , pour tout y, comme :

$$\mu(y) = \lim_{k \to \infty} P\left[X_k = y \mid X_0 = x\right]$$

Nous savons également que tout élément de l'espace d'état est de probabilité non nulle pour cette mesure.

Toutefois, si ce résultat nous permet de savoir qu'il existe une dynamique de fond de l'algorithme génétique, il nous reste à en déterminer les propriétés, l'influence des opérateurs (et des paramètres associés) qui jouent un grand rôle dans le processus.

Pour cela nous introduisons les notations suivantes :

Si $x = (x_1, ..., x_N)$ est un élément de E^N et i un point de E, nous noterons :

$$\begin{aligned} \widehat{f}(x) &= \widehat{f}(x_1, ..., x_N) = \max\left\{f(x_i) : 1 \leq i \leq N\right\} \\ \widehat{x} &= \left\{x_k \in \arg\max f(x)\right\} \\ [x] &= \left\{x_k \ : \ 1 \leq k \leq N\right\} \end{aligned}$$

De manière générale, les lettres z, y, z, u, v.. désignent des populations, i.e. des éléments de E^N, et les lettres i, j des points de E.

7.2.1 Processus de fond (X_k^∞)

C'est à partir de ce processus de fond qu'est reconstitué l'algorithme génétique en étudiant ses perturbations aléatoires par les différents opérateurs. Il est défini comme processus limite, lorsque les perturbations ont disparu. C'est également une chaîne de Markov sur E^N dont le mécanisme de transition est très simple puisque correspondant à la situation limite suivante :

Les N composantes de X_{k+1}^∞ sont choisies indépendamment et suivant la loi uniforme sur l'ensemble $\widehat{X_k^\infty}$.

- Les individus dont l'adaptation n'est pas maximale en k, sont éliminés et n'apparaissent pas dans la génération $k+1$,
- Les individus dont l'adaptation est maximale, ont des chances de survies égales

Cette chaîne est tout d'abord piégée dans l'ensemble S des populations ayant la même adaptation (ou ensemble des population d'*équi-adaptation*),

$$S = \left\{ x = (x_1, ..., x_N) \in E^N \quad : \; f(x_1) = f(x_2) = \cdots = f(x_N) \right\}$$

Cette population représente les *attracteurs* de la chaîne (voir 7.3 plus loin), puis elle est absorbée par une population uniforme, de sorte que :

$$\forall x \in E^N \qquad P\left[\exists x_i \in \widehat{x} \quad \exists K \quad \forall k \geq K \quad X_k^\infty = x_i \mid X_0^\infty = x_{ini}\right] = 1$$

Lorsque la population est devenue uniforme et en l'absence ici de perturbations, il ne se passe plus rien.

Ceci peut également se traduire en définissant les populations uniformes comme les *états absorbants* de la chaîne X_k^∞. Nous allons maintenant étudier la situation où ce processus est perturbé.

7.2.2 Processus perturbé (X_k^l)

La modélisation proposée par Cerf, part du processus de fond (X_k^∞),décrit ci-dessus, qui est perturbé aléatoirement, les perturbations sont indicées par le paramètre l. La chaîne de Markov (X_k^∞) devient donc une suite de chaînes de Markov (X_k^l), dont le mécanisme de transition est donné par la succession des transformations générées par les opérateurs.

$$X_k^l \quad \overset{Mutation}{\longrightarrow} \quad U_k^l \quad \overset{Croisement}{\longrightarrow} \quad V_k^l \quad \overset{Selection}{\longrightarrow} \quad X_{k+1}^l$$

Il nous faut pour cela modéliser précisément les opérateurs.

- Mutation $X_k^l \longrightarrow U_k^l$:

  Les mutations sont définies comme de petites perturbations aléatoires indépendante des individus, de la population X_k^l. Il est assez naturel d'introduire la probabilité $p_l(i,j)$ de transition[9] de mutation entre les points i et j de E, comme un noyau Markovien p_l.

  Trivialement on a :

  $$\forall i \in E \qquad \sum_{j \in E} p_l(i,j) = 1$$

  Sur la chaîne X_k^l, la probabilité de transition entre les points x et u de E^N est :

  $$P\left[U_k^l = u \mid X_k^l = x\right] = p_l(x_1, u_1) \cdot p_l(x_2, u_2) \cdots p_l(x_N, u_N)$$

  Plus précisément et afin d'analyser la dynamique de (X_k^l) lorsque l tend vers l'infini, nous reportons ici les hypothèses sur le mode et la vitesse de convergence des probabilités de transition. Pour cela nous supposons l'existence d'un noyau irréductible, α, sur E, i.e. : $\forall \; i, j \in E, \exists \; i_o, \; i_1, \cdots, \; i_r$ (c'est à dire un chemin dans E) tels que $i_0 = i$ et $i_r = j$ tels que :

  $$\prod_{0 \leq s \leq r-1} \alpha(i_k, i_{k+1}) > 0$$

  L'hypothèse d'irréductibilité du noyau α est essentielle, elle assure que tout point de l'espace est potentiellement visitable.

  La vitesse de convergence du noyau p_l, est caractérisée par le réel positif $\mathbf{a}$, tel que p_l admette le développement suivant :

  $$\forall i, j \in E \qquad \forall s \qquad p_l(i,j) = \begin{cases} \alpha(i,j) \cdot l^{-\mathbf{a}} + o(l^{-s}) & si \; i \neq j \\ 1 - \alpha(i,j) \cdot l^{-\mathbf{a}} + o(l^{-s}) & si \; i = j \end{cases} \tag{1}$$

[9] C'est la probabilité $P_l(i,j)$ pour un point i de E de se transformer par mutation en un point j de E

La condition de positivité de $\mathbf{a}$ nous permet de faire disparaître les perturbations lorsque l tend vers l'infini.

$$\forall i,j \in E \qquad \lim_{l\to\infty} p_l(i,j) = \delta(i,j) = \begin{cases} 0 & si\ i \neq j \\ 1 & si\ i = j \end{cases} \tag{2}$$

- Croisement $U_k^l \longrightarrow V_k^l$: Ici encore l'opérateur est modélisé comme effectuant de petites perturbations aléatoires sur des couples de la population U_k^l. Ces couples sont ici formés par les éléments successifs de la population, les transitions sont gérées par le noyau Markovien q_l sur $E \times E$, cette fois, de sorte que :

$$P\left[V_k^l = v \mid U_k^l = u\right] = q_l\left((u_1,u_2)\cdot(u_3,u_4)\cdots(u_{N-1},u_N)\right)$$

Pour ce noyau q_l nous supposerons l'existence d'un noyau irréductible β sur $E \times E$, la vitesse de convergence est alors paramétrée par le réel positif $\mathbf{b}$ tel que :

$$\forall\,(i_1,j_1) \in E\times E \qquad \forall\,(i_2,j_2) \in E\times E \qquad \forall s$$

$$q_l\left((i_1,j_1),(i_2,j_2)\right) = \begin{cases} \beta\left((i_1,j_1),(i_2,j_2)\right)\cdot l^{-\mathbf{b}} + o(l^{-s}) & si\ \ (i_1,j_1) \neq (i_2,j_2) \\ 1 - \beta\left((i_1,j_1),(i_2,j_2)\right)\cdot l^{-\mathbf{b}} + o(l^{-s}) & si\ \ (i_1,j_1) = (i_2,j_2) \end{cases} \tag{3}$$

L'évanouissement asymptotique des croisements est également imposée par la positivité de $\mathbf{b}$:

$$\forall i_1,\ i_2,\ j_1,\ j_2 \in E \qquad \lim_{l\to\infty} q_l\left((i_1,i_2)(j_1,j_2)\right) = \delta(i_1,i_2)\cdot\delta(j_1,j_2) \tag{4}$$

- Sélection $V_k^l \longrightarrow X_{k+1}^l$: C'est l'opérateur le plus compliqué et également le plus important puisqu'il permet la convergence vers les optima de f.

  Il est modélisé à l'aide d'une fonction de sélection F_l dont Cerf nous donne une définition précise, pouvant être résumée par :

$$\begin{aligned} F_l \quad : \quad & \{1,\cdots,N\}\times\left(\Re^+\right)^N \quad \longrightarrow [0,1] \\ & (i,\ f_1,f_2,\cdots f_N) \qquad\quad \longrightarrow F_l\left(i,\ f_1,f_2,\cdots,f_N\right) \end{aligned}$$

  telle que :

  1. $F\left(\cdot,f_1,f_2,\cdots f_N\right)$ est une probabilité sur $\{1,\cdots,N\}$
  2. Cette probabilité est indépendante de l'indexation des $f_1,f_2,\cdots f_N$ (on peut permuter les f_i)
  3. La probabilité favorise les éléments i associés à des valeurs élevées (i.e.)

$$Si \quad f_1 \geq f_2 \geq \cdots \geq f_N \qquad Alors$$

$$F_l\left(1,\ f_1,f_2,\cdots,f_N\right) \geq F_l\left(2,\ f_1,f_2,\cdots,f_N\right) \geq \cdots \geq F_l\left(N,\ f_1,f_2,\cdots,f_N\right)$$

Cet outil nous permet d'écrire la probabilité de transition correspondant à la dernière étape.

$$P\left[X_{k+1}^l = x \mid V_k^l = v\right] = \prod_{r=1}^{N} \Upsilon_l(x_r,v_r)$$

Ceci signifie que la probabilité de transition est le produit des probabilités sur chacune des N composantes de E.

La probabilité Υ_l entre deux composantes (x_r, v_r) est donnée par :

$$\Upsilon_l(x_r, v_r) = \sum_{k\,:\,v_k = x_k} F_l\left(k,\ f(v_1), f(v_2), \cdots, f(v_N)\right)$$

De même que pour les autres opérateurs, la fonction de sélection doit être choisie et sa vitesse de convergence caractérisée :

$$F_l\left(i,\ f_1, f_2, \cdots, f_N\right) = \frac{\exp\left(\mathbf{c} \cdot f_i \cdot \ln(l)\right)}{\sum_{r=1}^{N} \exp\left(\mathbf{c} \cdot f_r \cdot \ln(l)\right)} \tag{5}$$

Ce choix correspond bien à une probabilité de sélection avantageant les fortes adaptations au détriment des faibles, le réel positif c indexant cette fonction.

Le mécanisme de sélection opérant sur le processus de fond (X_k^∞), correspond à la fonction de sélection F_∞ définie par :

$$F_\infty\left(k, f(x_1), f(x_2), \cdots, f(x_N)\right) = \frac{\mathbf{1}_{\widehat{x}}(x_k)}{card(\widehat{x})}$$

C'est à dire, la loi uniforme sur l'ensemble $\widehat{x} = \{x_k \in \arg\max f(x)\}$

La suite $(F_l)_{l\in\mathbb{N}}$ des fonctions de sélection tend vers cette loi uniforme

$$\begin{array}{l} \forall x \in E^N \qquad \forall k \\ \lim_{l\to\infty}\ F_l\left(k, f(x_1), f(x_2), \cdots, f(x_N)\right) = F_\infty\left(k, f(x_1), f(x_2), \cdots, f(x_N)\right) \end{array} \tag{6}$$

Les conditions 2, 4, et 6 nous permettent d'assurer que le mécanisme de transition de la chaîne (X_k^l) converge vers celui du processus de fond (X_k^∞) :

$$\forall y, z \in E^N \qquad \lim_{l\to\infty}\ P\left[X_{k+1}^l = z \mid X_k^l = y\right] = \ P\left[X_{k+1}^\infty = z \mid X_k^\infty = y\right]$$

C'est également en ce sens que l'on interprète la chaîne (X_k^l) comme une perturbation de la chaîne (X_k^∞).

Les vitesses de convergence intervenant dans chacun des opérateurs jouent un rôle important. La formulation proposée en (1), (3) et (5), permet un ajustement équitable de ces vitesses (elles sont logarithmiquement du même ordre) de sorte qu'aucun opérateur ne domine les autres dans la dynamique. lorsque l tend vers l'infini, les conditions (2), (4), et (6) nous permettent d'assurer que le mécanisme de transition de la chaîne (X_k^l) converge vers celui du processus de fond (X_k^∞), et on a :

$$\forall y, z \in E^N \qquad \lim_{l\to\infty}\ P\left[X_{k+1}^l = z \mid X_k^l = y\right] = \ P\left[X_{k+1}^\infty = z \mid X_k^\infty = y\right]$$

La chaîne (X_k^l) se comporte alors comme le ferait (X_k^∞). La théorie de Freidlin-Wentzell nous donne les outils pour simplifier l'étude de ces processus à temps continu.

7.3 Application de la théorie de Freidlin et Wentzell

7.3.1 Principe

Soit le système différentiel de $\Re^N$ satisfaisant les équations déterministes :

$$\left\{ \begin{array}{l} dx_t = b(x_t)\ dt \\ x_0 = x_{ini} \end{array} \right. \tag{7}$$

Sous de "bonnes" hypothèses, il existe une solution (trajectoire) unique, $x(t)$ à l'équation (7) et à la condition initiale associée. l'une des préoccupation immédiates est de savoir si cette solution va, ou non, tendre vers un équilibre (qui n'est pas forcément unique). Et si oui, quel en est l'ensemble de stabilité. L'équilibre est défini comme une fonction constante x^* telle que $x^* = \lim_{t\to\infty} x_t$, et l'ensemble de stabilité comme l'ensemble $K(\ x^*)$ des points de départ qui mènent à cet équilibre[10]. On peut élargir cette notion, d'équilibre et de stabilité, par celles, très proches, d'attracteur et de bassin d'attraction.

Un attracteur du système est le voisinage compact K_i d'un point visité une infinité de fois, et le bassin d'attraction l'ensemble des points de départ qui mènent à cet attracteur. Nous supposerons que $\Re^d$ possède un nombre fini d'attracteurs $K_1, \cdots, K_r$.

La théorie de Freidlin-Wentzell étudie l'évolution du système 7 lorsqu'il subit des perturbations Brownièenes, d'intensité ε. Le système déterministe 7 devient alors un système différentiel stochastique.

$$\begin{cases} dX_t^\varepsilon = b(X_t^\varepsilon)\ dt + \varepsilon d\omega_t \\ X_0^\varepsilon = x_{ini} \end{cases} \tag{8}$$

Le processus $(X_t^\varepsilon)_{t\in\Re^+}$ est maintenant un processus stochastique perturbé par le mouvement brownien $(\omega_t)_{t\in\Re^+}$ et dépendant de ε. La situation change alors puisque les perturbations brownienne permettent au processus de s'échapper de n'importe quel bassin d'attraction, et en fait le processus les visite tous.

De plus, le processus est ergodique et admet une unique mesure de probabilité invariante, i.e.

$$\forall \mathcal{B}\ borelien\ \in \Re^d \qquad \lim_{t\to\infty} P\left[X_t^\varepsilon \in \mathcal{B} \mid X_0^\varepsilon = x_{ini}\right] = \mu^\varepsilon(\mathcal{B})$$

existe et $\mu^\varepsilon(\mathcal{B})$ est la probabilité de présence du processus dans le Borélien $\mathcal{B}$, lorsque le système a atteint son état d'équilibre. Cette probabilité μ^ε est invariante avec le point de départ x_{ini}.

Lorsque les perturbations cessent, le processus se comporte comme dans 7 et reste presque sûrement au voisinage $V(K_1 \cup \cdots \cup K_r)$ des attracteurs, tandis que la probabilité de présence dans n'importe quel Borélien $\mathcal{A}$ disjoint de $K_1 \cup \cdots \cup K_r$ disparaît.

$$\begin{aligned} \lim_{\varepsilon\to 0} \mu^\varepsilon\left(V(K_1 \cup \cdots \cup K_r)\right) &= 1 \\ \lim_{\varepsilon\to 0} \mu^\varepsilon\left(\mathcal{A}\right) &= 0 \end{aligned}$$

Le résultat principal de Freidlin et Wentzell repose sur l'équivalence du processus $(X_t^\varepsilon)_{t\in\Re^+}$ à temps continu et espace d'état $\Re^d$ et du processus $(Z_n^\varepsilon)_{n\in\mathbb{N}}$ à temps discret et espace d'état fini $\{1, \cdots .r\}$ décrivant les visites au $n$ième attracteur.

La construction précise de $(Z_n^\varepsilon)_{n\in\mathbb{N}}$, n'est pas reportée ici mais nous en donnons un aperçu afin de mieux comprendre ce dernier processus.

- Si x_{ini} est "proche" de l'attracteur K_h alors $Z_o^\varepsilon = h \in \{1, \cdots .r\}$
- puis le processus, sous l'influence de (ω_t), est attiré par K_s et $Z_1^\varepsilon = s$
- etc..

La chaîne de Markov[11] ainsi créée a pour espace d'états $\{1, \cdots .r\}$, est irréductible, et possède une unique mesure de probabilité invariante ν^ε.

Théorème 2. *L'étude du comportement asymptotique de la mesure μ^ε est "équivalente" à l'étude du comportement asymptotique de la mesure ν^ε*

[10] L'ensemble de stabilité de l'équilibre x^* est :

$$K(x^*) = \left\{ x_{ini} \in \Re^N,\ t.q.\ pour\ x_t\ solution\ de\ 7\ ;\ \lim_{t\to\infty}\ x_t = x^* \right\}$$

Pour chaque équilibre on définit ainsi son ensemble de stabilité. Cet équilibre est stable s'il contient un voisinage de l'équilibre, et instable s'il existe des points de départ infiniment proche de l'équilibre qui ne mènent pas à celui-ci.

[11] La nature Markovienne de ω_t, nous permet de montrer qu'il s'agit bien là d'une chaine de Markov.

Nous passons sous silence l'étude des probabilité de transition $P\left[Z_n^\varepsilon = i \mid Z_n^\varepsilon = j\right]$ de la chaîne $(Z_n^\varepsilon)_{n\in\mathbb{N}}$ qui s'écrivent comme des intégrales sur l'ensemble des fonctions ϕ qui lient les attracteurs K_i et K_j, laissant le lecteur intéressé se reporter à la lecture de Freidlin et Wentzell, ou de Cerf.

Notons toutefois que ces probabilité de transition s'écrivent :

$$P\left[Z_n^\varepsilon = i \mid Z_n^\varepsilon = j\right] \underset{\ln}{\sim} \exp -\frac{V(i,j)}{\varepsilon^2}$$

où $V(i,j) = \inf\{V(\phi), \phi(\cdot) \text{ continue } [0.1] \longmapsto \Re^d, \phi(0) \in K_i, \phi(T) \in K_j\}$ et $V(\phi) = \int_0^1 \left|\dot{\phi}(t) - b(\phi(t))\right|^2 dt$.
est une constante associée à ϕ et caractérisant sa vitesse de convergence.

La quantité $V(i,j)$ ou *coût de communication,* mesure le coût de passage de l'attracteur K_i à l'attracteur K_j.

Les intensités de transitions de la chaîne $(Z_n^\varepsilon)_{n\in\mathbb{N}}$, nous ouvrent la voie pour déterminer la mesure invariante ν^ε.

7.3.2 Mesure invariante ν^ε

Les outils qui permettent de déterminer cette mesure invariante ont été développés, une nouvelle fois par Freidlin et Wentzell, nous aurons besoin de certains d'entre eux.

Définition 9. *Soit i un élément de $\{1,...,r\}$. Un $i-$graphe sur $\{1,...,r\}$ est un graphe g sur $\{1,...,r\}$ possédant les propriétés suivantes :*

- $\forall j \neq i$, *le graphe g contient une unique flèche issue de j*
- *Il existe un chemin dans g qui mène de j à i*
- *g ne contient pas de flèche issue de i*

Il s'agit donc d'un graphe sans cycles formé de chemins qui aboutissent en i. On note $G(i)$ l'ensemble des $i-$graphes sur $\{1,...,r\}$.

Définition 10. *La fonction d'énergie virtuelle W est la fonction de $\{1,...,r\}$ dans $\Re^+$ définie par :*

$$\forall i \in \{1,...,r\} \qquad W(i) = \min_{g\in G(i)} \sum_{(\alpha\to\beta)\in g} V(\alpha,\beta)$$

A cette fonction est associé l'ensemble W^* des minima globaux de W.

Finalement, la mesure invariante ν^ε est caractérisée par :

$$\forall i \in \{1,...,r\} \qquad \nu^\varepsilon(i) \underset{\ln}{\sim} \exp -\frac{W(i) - W(W^*)}{\varepsilon^2}$$

où $W(W^*) = \min\{W(i) \ : \ i \in \{1,...,r\}\}$.

Le comportement asymptotique de ν^ε (et par la même occasion de μ^ε) est donc connu : la mesure ν^ε se concentre sur les attracteurs dont l'indice est dans W^* et décroît vers zéro à la vitesse $\exp -\frac{Cste}{\varepsilon^2}$ pour les autres attracteurs. Il existe donc un sous-ensemble de W^* de l'ensemble des attracteurs sur lequel se concentre la mesure invariante du processus.

$$\lim_{\varepsilon\to 0} \lim_{t\to\infty} P\left[X_t^\varepsilon \in V\left(\bigcup_{i\in W^*} K_i\right) \mid X_0^\varepsilon = x_{ini}\right] = 1$$

La dynamique du processus est donc caractérisable.

7.3.3 Dynamique du processus

Dans sa thèse, Cerf nous donne une très claire interprétation de la hiérarchie des cycles qui caractérisent la dynamique du processus. Supposons que le processus soit initialement dans le bassin d'attraction de K_1. Il quitte K_1 au bout d'un temps fini. Parmi toutes les trajectoires de sortie, il en existe une plus "probable" que les autres, qui l'amène vers un nouvel attracteur; par exemple K_2 puis, bientôt K_3. L'ensemble des attracteurs étant par hypothèse fini, le processus finit par revisiter un attracteur formant un cycle d'ordre 1 sur lequel le processus tourne longtemps, très longtemps. Englobons maintenant ces trois attracteurs dans une boîte. Comme toujours, les perturbations browniennes finissent par pousser le processus hors de cette boîte, et ici encore, il existe une trajectoire de sortie canonique qui fait tomber le processus dans un nouveau bassin d'attraction, ou plus généralement, dans un autre cycle d'ordre 1.

Les cycles d'ordre 1 sont aussi en nombre fini, et le processus finit par revisiter un cycle d'ordre 1: un cycle d'ordre 2 est alors formé, dans lequel le processus reste piégé très longtemps. En continuant de la sorte, il est possible de construire toute une hiérarchie de cycles qui épuise l'ensemble des attracteurs et fournit une image très précise de la dynamique asymptotique du processus. A chaque transition entre cycles est associée une constante qui caractérise la difficulté de la transition.

Enfin, lorsque ε décroît avec le temps ($\varepsilon = \varepsilon(t)$ est une fonction de t qui tend en décroissant vers 0), nous obtenons un processus de Markov inhomogène (le mécanisme de transition dépend du temps).

- Si $\varepsilon(t)$ décroît très lentement, de sorte qu'à chaque instant la loi de X_t soit proche de l'état d'équilibre associé au niveau de perturbation $\varepsilon(t)$, la situation ne change pas fondamentalement. La loi limite correspond à la limite de la suite des lois d'équilibre.

- Si au contraire $\varepsilon(t)$ décroît très rapidement, le processus risque de rester piégé dans certains sous-ensembles d'attracteurs: plus précisément, dans la hiérarchie des cycles, certaines transitions ne pourront être effectuées qu'un nombre fini de fois, alors que d'autres, plus "faciles", seront réalisées une infinité de fois avec probabilité 1. La loi limite dépend alors fortement du point de départ.

La hiérarchie des cycles permet ainsi de décrire les dynamiques possibles de (X_t) en fonction de la vitesse de décroissance de $\varepsilon(t)$.

7.3.4 Résultats de convergence

Lorsque l croit vers l'infini, les perturbations affectant le processus (X_k^l) diminuent de sorte que cette chaîne se comporte, presque sûrement, comme la chaîne (X_k^∞). Plus précisément, nous savons que les attracteurs de la chaîne (X_k^∞) sont les populations d'équi-adaptation S et les populations uniformes (attracteurs stables). La chaîne (X_k^l) va donc être attirée par ses attracteurs, en commençant par l'ensemble S.

La théorie de Freidlin et Wentzell nous permet de reporter cette étude sur celle de la chaîne des (Z_k^l) des visites successives de (X_k^l) à l'ensemble S. Nous poserons donc $Z_k^l = X_{T_k}^l$ où T_k est l'instant de la $k$ième visite de (X_k^l) dans S.

Les probabilités de transition de la chaîne (Z_k^l), sont estimées à l'aide des opérateurs définis en 7.2 et selon le schéma développé ci-dessus. Les fonctions de coût de communication $V(i,j)$ et d'énergie virtuelle W sont définies et estimées.

Nous savons alors que la suite des mesures stationnaires de la chaîne (X_k^l) se concentre sur l'ensemble W^* des minima de W :

$$\forall x_{ini} \in E^N \qquad \lim_{l\to\infty} \lim_{k\to\infty} P\left[X_k^l \in W^* \mid X_0^l = x_{ini}\right] = 1$$

L'un des principaux résultats indique qu'il existe une taille de la population de (X_k^l), (taille critique) telle que les maxima de f soient atteints asymptotiquement avec la probabilité 1.

7.3.5 Taille critique

Supposons fixés l'espace d'état E, la fonction d'adaptation f, les noyaux de transition de mutation α et de croisement β, ainsi que les constantes positives gérant les trois opérateurs $\mathbf{a}$, $\mathbf{b}$, et $\mathbf{c}$.

Théorème 3. *(Cerf 1993)*
Il existe une valeur critique N^*, telle que lorsque la taille de la population de l'algorithme génétique dépasse N^*, l'ensemble f^* des maxima globaux de f, contient l'ensemble W^*.
Cette taille critique N^* , dépend fortement de l'espace d'état E, de la fonction d'adaptation f, des noyaux de transition de mutation α et de croisement β, ainsi que des paramètres $\mathbf{a}$, $\mathbf{b}$, et $\mathbf{c}$.

Une borne grossière, mais lisible de N^* est :

$$N^* \leq \frac{\mathbf{a}R + \mathbf{c}(R-1)\Delta}{\min(\mathbf{a},\ \frac{\mathbf{b}}{2},\ \mathbf{c}\delta)}$$

où :

- R est le nombre minimal de transition permettant de joindre deux points arbitraires de E par mutation
- Δ et δ sont des paramètres d'échelle :
  - $\Delta = \max\{|f(i) - f(j)| : i, j \in E\}$ paramètre mesurant les écarts maximaux de f
  - $\delta = \min\{|f(i) - f(j)| : i, j \in E,\ f(i) \neq f(j)\}$ mesurant les écarts minimaux de f.

Il est intéressant de relever que le résultat est obtenu sans faire intervenir l'opérateur de croisement, qui n'est donc pas indispensable. L'exploration par mutation et la sélection suffisent à assurer la convergence vers f^* ([Zhi91]).

Ce premier résultat nous indique que dès que $N \geq N^*$, la suite des mesures stationnaires de la chaîne (X_k^l) se concentre asymptotiquement sur f^* lorsque l tend vers l'infini. On peut dans une étape suivante faire évoluer l , et donc l'intensité des perturbations, en fonction de la génération. Nous obtenons alors une chaîne de Markov inhomogène $(X_k^{l(k)})$ dont le mécanisme de transition dépend alors de la génération k.

7.3.6 Vitesse de convergence

Le principal problème est de savoir si cette chaîne inhomogène peut avoir un comportement proche de celui de la chaîne homogène (X_k^l), et si oui, sous quelles conditions. La vitesse de croissance de $l(k)$ vers l'infini, est bien évidemment, au centre du débat.

- Si $l(k)$ croît "lentement", alors la loi de X_k sera proche de la loi stationnaire $\mu^{\varepsilon(l(n))}$ de niveau de perturbation $\varepsilon(l(n))$ associé à $l(n)$.
- Si $l(k)$ croît "rapidement", alors X_k risque de rester piégé dans des bassins d'attraction ne correspondant pas aux maxima de f, l'intensité des perturbations devenant trop faible pour pouvoir s'en échapper.

La vitesse recherchée se situe entre ces deux extrêmes, permettant à X_k de s'échapper des "mauvais" bassins d'attraction (ne correspondant pas à des maxima de f) et de rester piégé dans le "bon" (celui des points de f^*).

La vitesse de convergence de la suite $l(k)$ est caractérisée par l'*exposant de convergence*[12], λ.

Définition 11. *L'exposant de convergence λ de la suite $l(k)$ est l'unique réel λ tel que :*

$$\sum_{k\in\mathbb{N}} l(k)^{-\theta} \quad \begin{array}{ll} \rightarrow & converge\ pour\ \theta > \lambda \\ \rightarrow & diverge\ pour\ \theta < \lambda \end{array}$$

[12] Egalement appelé *rayon de convergence.*

Deux conditions pour la colonisation de f^* sont également données par Cerf, l'une nécessaire, l'autre suffisante.

Théorème 4. *Condition nécessaire pour la colonisation de f^**
Pour que :

$$\forall x_{ini} \in E^N \qquad P\left[\exists K \quad \forall k \geq K \quad [X_{T_k}] \subset f^* \mid X_0 = x_{ini}\right] = 1$$

c'est à dire, pour que la chaîne $Z_k = X_{T_k}$ des visites successives des attracteurs soit piégée dans f^* après un nombre fini K de transitions, il est nécessaire que l'exposant de convergence λ de la suite $l(k)$ appartienne à l'intervalle $]\phi, \psi[$.
Les constantes ϕ et ψ sont des caractéristiques du problème,l'intervalle $]\phi, \psi[$ est alors non vide pour N assez grand.

Théorème 5. *Condition suffisante pour la colonisation de f^**
Il existe deux constantes η et ρ telles que si l'exposant de convergence λ de la suite $l(k)$ appartient à l'intervalle $]\eta, \rho[$, alors :

$$\forall x_{ini} \in E^N \qquad P\left[\exists K \quad \forall k \geq K \quad [X_{T_k}] \subset f^* \ , \ \widehat{X_k} \subset f^* \mid X_0 = x_{ini}\right] = 1$$

ce qui signifie qu'après un nombre fini de transitions, nous avons presque sûrement, la situation suivante :

- *la chaîne X_{T_k} est piégée dans f^*,*
- *la population X_k contient toujours un ou des individus appartenant à f^*.*

7.3.7 En guise de conclusion

D'autres résultats existent, tant dans le travail de Cerf, que dans la littérature citée dans cette section. Ils demandent cependant un investissement supplémentaire dans la compréhension des outils développés. Le but de cette section était de convaincre le lecteur que l'étude théorique des algorithmes génétiques donne (déjà) de substantiels résultats. De nombreuses interrogations demeurent cependant concernant les relations réelles entre les différents paramètres caractérisant l'algorithme génétique et les choix pratiques de ceux-ci. Dans ce domaine, la pratique devance encore la théorie, même si les mécanismes commencent à être plus clairs. Il reste également à étudier les nombreux raffinements (tels que le scaling, sharing, le clustering, l'élitisme) indispensables dans la pratique.

References

[AS92] Jean-Marc Alliot and Thomas Schiex. *Intelligence Artificielle et Informatique Théorique*. Cepadues, 1992. ISBN: 2-85428-324-4.

[BD93] A. Bertoni and M. Dorigo. Implicit parallelism in genetic algorithms. *Artificial Intelligence*, 61(2):307–314, 1993.

[BG87] C.L Bridges and D.E Goldberg. An analysis of reproduction and crossover in a binary-coded genetic algorithm. In *Proceedings of the Second International Conference on Genetic Algorithm*. ICGA, 1987.

[BG91] C.L Bridges and D.E Goldberg. An analysis of multipoint crossover. In *Proceedings of the Foundation Of Genetic Algorithms*. FOGA, 1991.

[BM93] T.N Bui and B.R Moon. Hyperplane synthesis for genetic algorithms. In *Proceedings of the Fifth International Conference on Genetic Algorithm*. ICGA, 1993.

[BM94] T.N Bui and B.R Moon. Analysing hyperplane synthesis in genetic algorithms using clustered schemata. Technical Report Cse-94-026, Dep. of Comp. Sci. and Engineering, Penn. State University, March 1994.

[Bra90] H. Braun. On traveling salesman problems by genetic algorithms. In *1st Workshop on Parallel Problem Solving from Nature*, October 1990.

[Cat90] O. Catoni. *Large deviations for Annealing*. PhD thesis, Université de Paris XI, 1990.

[Cer94] R Cerf. *Une Théorie Asymptotique des Algorithmes Génétiques*. PhD thesis, Université Montpellier II (France), 1994.

[CJ91] R.J. Collins and D.R. Jefferson. Selection in massively parallel genetic algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, 1991.

[CS88] R.A Caruana and J.D Schaffer. Representation and hidden bias : Gray versus binary coding for genetic algorithms. In *Proceedings of the Fifth International Conference on Machine Learning*, 1988.

[DAN94] Nicolas Durand, Jean-Marc Alliot, and Joseph Noailles. Algorithmes genetiques : un croisement pour les problemes partiellement separables. In *Proceedings of the Journees Evolution Artificielle Francophones*. EAF, 1994.

[Daw86] Richard Dawkins. *The blind watchmaker*. Norton, New-York, 1986.

[Daw89] Richard Dawkins. *L'horloger aveugle*. Robert Laffont, 1989. Edition orignale [Daw86].

[DM92] D Dasgupta and D.R McGregor. A structured genetic algorithm. Technical Report IKBS-8-92, Dep. of Computer Science. University og Strathclyde, Clasgow. UK, 1992.

[FOW66] L.J Fogel, A.J Owens, and M.J Walsh. *Artificial Intelligence Through Simulated Evolution*. Wiley and sons. NY, 1966.

[FW83] M.I Freidlin and A.D Wentzell. *Random Perturbations of Dynamical Systems*. Springer-verlag, New-York, 1983.

[GBD+94] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. Pvm 3 user's guide and reference manual. Technical report, Oak Ridge National Laboratory, 1994.

[GGRG85] J. Grefenstette, R. Gopal, B. Rosmaita, and D. Gucht. Genetic algorithms for the traveling salesman problem. In *1st International Conference on Genetic Algorithms and their Applications*, 1985.

[GL85] D. Goldberg and R. Lingle. Alleles, loci, and the travelling salesman problem. In *1st International Conference on Genetic Algorithms and their Applications*, 1985.

[Gol89a] David Goldberg. *Genetic Algorithms*. Addison Wesley, 1989. ISBN: 0-201-15767-5.

[Gol89b] D.E Goldberg. Genetic algorithms and walsh functions. part 1 and 2. *Complex Systems*, 3:129–171, 1989.

[Gol89c] D.E Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading MA Addison Wesley, 1989.

[Gol91] D.E Goldberg. Real-coded genetic algorithms, virtual alphabets and blocking. *Complex Systems*, 5:139–167, 1991.

[GT82] A. Griewank and Ph. L. Toint. On the unconstrained optimization of partially separable functions. In M. J. D. Powell, editor, *Nonlinear Optimization 1981*, pages 301–312, London and New York, 1982. Academic Press.

[HGL93] A. Homaifar, S. Guan, and G. Liepins. A new genetic approach on the traveling salesman problem. In *Fifth International Conference on Genetic Algorithms*, July 1993.

[HN93] J. Horn and N. Nafpliotis. Multiobjective optimization using the nitched pareto genetic algorithm. Illigal Report 93005, University of Illinois at Urbana, 1993.

[Hol62] John Holland. Outline for a logical theory of adaptive systems. *Journal of the Association of Computing Machinery*, 3, 1962.

[IR92] L Ingber and B Rosen. Genetic algorithms and very fast simulated re-annealing. *Mathematical Computer Modeling*, 16(11):87–100, 1992.

[LeF95] Yann LeFablec. Optimisation par algorithmes gntiques parallles et multi-objectifs. Master's thesis, Ecole Nationale de l'Aviation Civile (ENAC), 1995.

[MG92] S.W Mahfoud and D.E Goldberg. Parallel recombinative simulated annealing : A genetic algorithm. Illigal report 92002, University of Illinois, Urbana, IL 61801-2996, April 1992.

[Mic92] Z Michalewicz. *Genetic algorithms + Data Structures = Evolution Programs*. Springer-verlag, 1992.

[MJ91] Z Michalewicz and C.Z Janikov. Handling constraints in genetic algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithm*. ICGA, 1991.

[Muh89] H. Muhlenbein. Parallel genetic algorithms, population genetics and combinatorial optimization. In *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.

[OSH89] I. Oliver, D. Smith, and J. Holland. Permutation crossover operators on the travelling salesman problem. In *Second International Conference on Genetic Algorithms*, July 1989.

[PLG87] C. Pettey, M. Leuze, and J. Grefenstette. A parallel genetic algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms*, 1987.

[SGE91] R.E Smith, D.E Goldberg, and J.A Earickson. *SGA-C: A C-language implementation of a Simple Genetic Algorithm*, May 1991. TCGA report No. 91002.

[SPF93] R.E. Smith, A.S. Perelson, and S. Forrest. Searching for diverse, cooperative populations with genetic algorithms. *Evolutionary Computation*, 1(2):127–149, 1993.

[Tro93] A. Trouvé. *Parallélisation massive du recuit simulé*. PhD thesis, Université de Paris XI, 1993.

[Vos91] M.D Vose. Generalizing the notion of schema in genetic algorithms. *Artificial Intelligence*, 50:385–396, 1991.

[Wri91] A.H Wright. Genetic algorithms for real parameter optimization. In *Proceeding of the Foundation Of Genetic Algorithms*. FOGA, 1991.

[WSF89] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesman: The genetic edge recombination operator. In *Third International Conference on Genetic Algorithms*, 1989.

[YG93] X Yin and N Germay. A fast genetic algorithm with sharing scheme using cluster analysis methods in multimodal function optimization. In *Proceedings of the Artificial Neural Nets and Genetic Algorithms*, 1993.

[Zhi91] Anatoly A. Zhigljavsky. *Theory of Global Random Search*. Kluwer Academic Plubishers, 1991.