ALGORITHMES GENETIQUES

Présenté par
Souquet Amédée
Radet Francois-Gérard

TE de fin d'année
Tutorat de Mr Philippe Audebaud

Soutenu le 21/06/2004 devant la commission composée de :

S. Julia
P. Audebaud
G. Dufay

Avant – propos

Les algorithmes génétiques, initiés dans les années 1970 par John Holland, sont des algorithmes d'optimisation s'appuyant sur des techniques dérivées de la génétique et des mécanismes d'évolution de la nature : croisement, mutation, sélection.

Nos travaux consisteront à définir ces algorithmes évolutionnaires, et à les comparer aux méthodes déterministes afin de pouvoir cerner leur utilité en fonction du problème posé. Le devoir s'articulera sur deux axes principaux
qui serviront de fils conducteurs au passage de la théorie à la pratique.
Dans un premier temps, nous nous attacherons à l'origine des algorithmes génétiques, leur appartenance dans le monde de la vie artificielle, et à la théorie de l'évolution des espèces formulée par le naturaliste Charles Darwin, sur laquelle ces premiers sont basés.
Nous montrerons quels en sont les principes, de telle manière à ce que nous puissions poursuivre vers notre premier axe, qui se penchera sur les différentes méthodes de sélection et introduira les différents operateurs, qui soulignent la nécessité de diversité.
Sur ces bases, nous proposerons, dans notre dernière partie, de souligner l'aspect pragmatique de ces algorithmes en illustrant leur utilisation et leur domaine d'application dans les sciences contemporaines.

Introduction

C'est en 1860 que Charles Darwin publie son livre intitulé *L'origine des espèces au moyen de la sélection naturelle ou la lutte pour l'existence dans la nature*. Dans ce livre, Darwin rejete l'existence «de systèmes naturels figés», déjà adaptés pour toujours à toutes les conditions extérieures, et expose sa théorie de l'évolution des espèces : sous l'influence des contraintes extérieurs, les êtres vivants se sont graduellement adaptés à leur milieu naturel au travers de processus de reproductions.

Darwin proposa une théorie qui clarifie l'évolution des espèces en mettant en avant quatre lois :

- La loi de croissance et de reproduction.
- La loi d'hérédité qu'implique quasiment la loi de reproduction
- La loi de variabilité , résultant des condition d'existence.
- La loi de multiplication des espèces qui amène la lutte pour l'existence et qui a pour conséquence la sélection naturelle.

Presque simultanément, en 1866, Mendel (« le moine des poix ») publie l'article retraçant dix années d'expériences d'hybridation chez les végétaux (recombinaison des gênes) et l'adresse aux sociétés scientifiques des quatre coins du monde. Les réactions sont mitigées, voire inexistantes. Le monde scientifique n'est pas prêt à reconnaître la qualité de ses résultat. Ce n'est seulement en 1900, que la publication de trois nouveaux articles signés Hugo de Vries, Carl Correns et Erich von Tschermark révèle des résultats similaires à ceux de Mendel,et feront que ces premiers seront reconnus.

C'est alors à partir du 20 ème siècle que la mutation génétique a été mise en évidence.
Les problèmes de traitement de l'information sont résolus de manières figés : lors de sa phase de conception, le système reçoit toutes les caractéristiques nécessaires pour les conditions d'exploitations connues au moment de sa conception, ce qui empêche une adaptation à des conditions d'environnement inconnues, variables ou évolutives. Les chercheurs en informatique étudient donc des méthodes pour permettrent aux systèmes d'évoluer spontanément en fonction de nouvelles conditions : c'est l'émergence de la programmation évolutionnaire (c*f. Figure 1*).

Dans les années 1960, John Holland étudie les systémes évolutifs et, en 1975, il introduit le premier modèle formel des algorithmes génétiques (*the canonical genetic algorithm AGC*) dans son livre *Adaptation in Natural and Artificial Systems*. Il expliqua comment ajouter de l'intelligence dans un programme informatique avec les croisements (échangeant le matériel génétique) et la mutation (source de la diversité génétique). Ce modèle servira de base aux recherches ultèrieures et sera plus particulièrement repris par Goldberg qui publiera en 1989, un ouvrage de vulgarisation des algorithmes génétiques, et ajouta à la théorie des algorithmes génétiques les idées suivantes :

- un individu est lié à un environnement par son code d'ADN.

- une solution est liée à un problème par son indice de qualité.

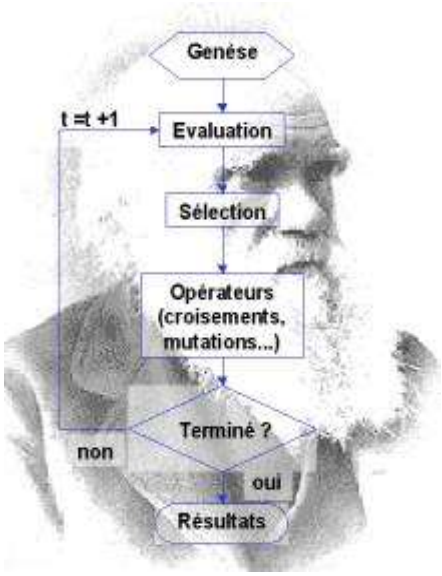

Figure 1. organigramme d'un algorithme évolutionnaire.

Ci-dessus est présenté l'organigramme d'un algorithme évolutionnaire. Il s'agit de simuler l'évolution d'une population d'individus divers (généralement tirée aléatoirement au départ) à laquelle on applique différents opérateurs (recombinaisons, mutations…) et que l'on soumet à une sélection, à chaque génération. Si la sélection s'opère à partir de la fonction d'adaptation, alors la population tend à s'améliorer [Bäck, 1996 et Bäck, 1997]. Un tel algorithme ne nécessite aucune connaissance du problème : on peut représenter celui-ci par une boîte noire comportant des entrées (les variables) et des sorties (les fonctions objectif). L'algorithme ne fait que manipuler les entrées, lire les sorties, manipuler à nouveau les entrées de façon à améliorer les sorties, etc. [Whitley, 1993] C'est ainsi qu'ont procédé les éleveurs pendant des millénaires : ils ont réussi à modifier, selon leurs désirs, de nombreuses espéces animales sans connaissance en génétique ou biologie moléculaire.

Les algorithmes évolutionnaires constituent une approche originale : il ne s'agit pas de trouver une solution analytique exacte, ou une bonne approximation numérique, mais de trouver des solutions satisfaisant au mieux à différents critères, souvent contradictoires. S'ils ne permettent pas de trouver à coup sûr la solution optimale de l'espace de recherche, du moins peut-on constater que les solutions fournies sont généralement meilleures que celles obtenues par des méthodes plus classiques, pour un même temps de calcul.

Ils font parti du champ de la vie artificielle. La vie artificielle est l'étude des systèmes conçus par l'homme, qui présentent des comportements similaires aux systèmes vivants naturels. Elle complète

l'approche traditionnelle de la biologie, définie étymologiquement par *étude des êtres vivants,* en essayant de synthétiser leurs comportements sur support artificiel. La modélisation, s'ajoutant à l'observation, à la théorie et à l'expérience, est un nouvel outil scientifique qui s'est fait valoir

depuis l'avènement de l'informatique. Celle-ci peut contribuer à la biologie théorique en la plaçant dans un contexte plus vaste.

L'objectif est double: d'une part, la modélisation de ces phénomènes permet de mieux les comprendre, et ainsi mettre en évidence les mécanismes qui sont à l'origine de la vie ; d'autre part, on peut exploiter ces phénomènes de façon libre et peuvent donc être diverses.

Le domaine de l'évolution artificielle n'a connu une réelle expansion qu'à partir de ces 15 dernières années. Pourtant, l'idée de simuler sur ordinateurs des phénomènes évolutionnaires remonte aux années 50. Des concepts tels que la représentation des chromosomes par des chaînes binaires étaient déjà présents.

L'essor de l'évolution artificielle, depuis les années 80, peut s'expliquer par deux phénomènes concurrents. Premièrement, cet essor est principalement dû à l'accroissement exponentiel des moyens de calculs mis à la disposition des chercheurs, ce qui leur permet d'afficher des résultats expérimentaux pertinents et prometteurs. Le deuxième point est l'abandon du biologiquement plausible.

Trois types d'algorithmes évolutionnaires ont été développé isolément et à peu prés simultanément, par différents scientifiques : la programmation évolutionniste (L. Fogel 1966), les Stratégies d'évolution (J. Rechenberg 1973) et les Algorithmes Génétiques (J. Holland 1975).

Dans les années 90, ces trois champs ont commencé à sortir de leur isolement et ont été regroupés sous le terme anglo-saxon d'*Evolutionnary Computation*.

Nous traiterons seulement ici les algorithmes génétiques fondés sur le Néo-Darwinisme, c'est-à-dire l'union de la théorie de l'évolution et de la génétique moderne. Ils s'appuient sur différentes techniques dérivées de cette dernière : croisements, mutation, sélection...

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 codages 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 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.

Nous savons maintenant sur quoi se basent les algorithmes génétiques.
Il est désormais temps d'approfondir les mécanismes de séléction de population et la notion de

diversité qui en découle. Nous tacherons également de définir les opérateurs évoqués dans l'organigramme de l' algorithme évolutionnaire (*cf. figure 1*) .
Donner une image à la fois globale et précise des outils principaux des algorithmes génétiques, tel sera notre objectif majeur au cours de notre seconde partie.

I

Fonctionnement des algorithmes génétiques

Les algorithmes génétiques fournissent des solutions aux problèmes n'ayant pas de solutions calculables en temps raisonnable de façon analytique ou algorithmique.
Selon cette méthode, des milliers de solutions (génotypes) plus ou moins bonnes sont crées au hasard puis sont soumises à un procédé d'évaluation de la pertinence de la solution mimant l'évolution des espèces : les plus "adaptés", c'est-à-dire les solutions au problème qui sont les plus optimales survivent davantage que celles qui le sont moins et la population évolue par générations successives en croisant les meilleures solutions entre elles et en les faisant muter, puis en relançant ce procédé un certain nombre de fois afin d'essayer de tendre vers la solution optimale.

Le mécanisme d'évolution et de sélection est indépendant du problème à résoudre : seules changent trois fonctions :

- la fonction qui s'occupe de représenter le problème en codant chaque information caractérisant une solution possible selon un codage bien particulier, chaque information représente alors un gène et toutes les valeurs que peuvent prendre cette caractéristique représentent les allèles possibles pour ce gène, et en concaténant tous ces gènes pour obtenir un chromosome qui lui représente une solution dans son intégralité

- la fonction inverse qui à partir d'un chromosome permet d'obtenir une solution par décodage du génôme.

- la fonction qui évalue l'adaptation d'une solution à un problème, sa pertinence.

Cette technique est d'application générale.

En effet, quand on utilise les algorithmes génétiques , aucune connaissance de la manière dont résoudre le problème n'est requise, il est seulement nécessaire de fournir une fonction permettant de coder une solution sous forme de gènes (et donc de faire le travail inverse) ainsi que de fournir une fonction permettant d'évaluer la pertinence d'une solution au problème donné.

Cela en fait donc un modèle minimal et canonique pour n'importe quel système évolutionnaire et pour n'importe quel problème pouvant être abordé sous cet angle, sous ce paradigme.

Cette représentation nous permet donc d'étudier des propriétés quasiment impossibles à étudier dans leur milieu naturel, ainsi que de résoudre des problèmes n'ayant pas de solutions calculables en temps raisonnables si on les aborde sous d'autres paradigmes, avec des performances quantifiables, facilement mesurables et qu'on peut confrontés aux autres stratégies de résolution.

Les algorithmes génétiques peuvent être particulièrement utiles dans les domaines suivants :

- Optimisation : optimisation de fonctions, planifiacation, etc ...
- Apprentissage : classification, prédiction, robotique, etc ...
- Programmation automatique : programmes LISP, automates cellulaires, etc ...
- Etude du vivant, du monde réél : marchés économiques, comportements sociaux, systèmes immunitaires, etc ...

Les principales différences des algorithmes génétiques par rapport aux autres paradigmes sont les suivantes :

- On utilise un codage des informations : on représente toutes les caractéristiques d'une solution par un ensemble de gènes, c'est-à-dire un chromosome, sous un certain codage (binaire, réél, code de Gray, etc ...), valeurs qu'on concatène pour obtenir une chaîne de caractères qui est spécifique à une solution bien particulière (il y a une bijection entre la solution et sa représentation codée

- On traite une population "d'individus", de solutions : cela introduit donc du parallèlisme.

- L'évaluation de l'optimalité du système n'est pas dépendante vis-à-vis du domaine.

- On utilise des règles probabilistes : il n'y a pas d'énumération de l'espace de recherche, on en explore une certaine partie en étant guidé par un semi-hasard : en effet des opérateurs comme la fonction d'évaluation permet de choisir de s'intéresser à une solution qui semble représenter un optimum local, on fait donc un choix délibéré, puis de la croiser avec une autre solution optimale localement, en général la solution obtenue par croisement est meilleure ou du même niveau que ses parents, mais ce n'est pas assuré, cela dépend des aléas du hasard, et cela et d'autant plus vrai pour l'opérateur de mutation qui ne s'applique qu'avec une certaine probabilité et dans le cas où il s'applique choisit aléatoirement sur quel(s) locus(loci) introduire des modifications.

Un algorithme génétique générique à la forme suivante :

1) Initialiser la population initiale P.

2) Evaluer P.

3) TantQue (Pas Convergence) faire :

a) P ' = Sélection des Parents dans P

b) P ' = Appliquer Opérateur de Croisement sur P '

c) P ' = Appliquer Opérateur de Mutation sur P '

d) P = Remplacer les Anciens de P par leurs Descendants de P '

e) Evaluer P

FinTantQue

Le critère de convergence peut être de nature diverse, par exemple :

- Un taux minimum qu'on désire atteindre d'adaptation de la population au problème,
- Un certain temps de calcul à ne pas dépasser,
- Une combinaison de ces deux points.

Les différents points introduits ci-avant vont maintenant être étudiés en détail dans la section 3.

Section 1

Le codage

Chaque paramètre d'une solution est assimilé à un gène, toutes les valeurs qu'il peut prendre sont les allèles de ce gène, on doit trouver une manière de coder chaque allèle différent de façon unique (établir une bijection entre l'allèle "réél" et sa représentation codée).

Un chromosome est une suite de gène, on peut par exemple choisir de regrouper les paramètres similaires dans un même chromosome (chromosome à un seul brin) et chaque gène sera repérable par sa position : son locus sur le chromosome en question.

Chaque individu est représenté par un ensemble de chromosomes, et une population est un ensemble d'individus.

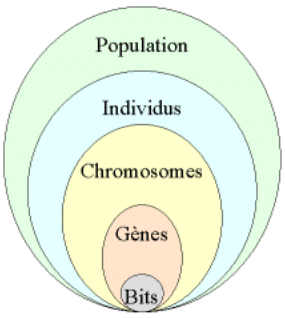

Figure 2: les cinq niveaux d'organisation d'un algorithme génétique

Il y a trois principaux types de codage utilisables, et on peut passer de l'un à l'autre relativement facilement :

- le codage binaire : c'est le plus utilisé.
  Chaque gène dispose du même alphabet binaire {0, 1}
  Un gène est alors représenté par un entier long (32 bits), les chromosomes qui sont des suites de gènes sont représentés par des tableaux de gènes et les individus de notre espace de recherche sont représentés par des tableaux de chromosomes.
  Ce cas peut être généralisé à tout alphabet allélique n-aire permettant un codage plus intuitif, par exemple pour le problème du voyageur de commerce on peut préférer utiliser l'alphabet

allélique {c1, c2, c3, ..., cn} où ci représente la ville de numéro i.

- le codage réél : cela peut-être utile notamment dans le cas où l'on recherche le maximum d'une fonction réelle.

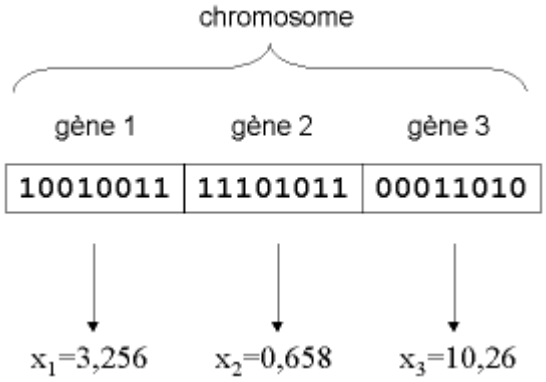

Figure 3 : illustration schématique du codage des variables réelles

- le codage de Gray : dans le cas d'un codage binaire on utilise souvent la "distance de Hamming" comme mesure de la dissimilarité entre deux éléments de population, cette mesure compte les différences de bits de même rang de ces deux sequences.
  Et c'est la que le codage binaire commence à montrer ses limites. En effet, 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" : le codage de Gray est un codage qui a comme propriété que entre un élément n et un élément n + 1, donc voisin dans l'espace de recherche, un seul bit diffère.

Section 2

L'opérateur de sélection

Cet opérateur est chargé de définir quels seront les individus de P qui vont être dupliqués dans la nouvelle population P' et vont servir de parents (application de l'opérateur de croisement).
Soit *n* le nombre d'individus de P, on doit en séléctionner *n/2* (l'opérateur de croisement nous permet de repasser à *n* individus).

Cet opérateur est peut-être le plus important puisqu'il permet aux individus d'une population de survivre, de se reproduire ou de mourir. En règle général, la probabilité de survie d'un individu sera directement reliée à son efficacité relative au sein de la population.

On trouve essentiellement quatre types de méthodes de sélection différentes :

- La méthode de la "loterie biaisée" (roulette wheel) de GoldBerg,
- La méthode "élitiste",
- La sélection par tournois,
- La sélection universelle stochastique.

a) La loterie biaisée ou roulette wheel :

Cette méthode est la plus connue et la plus utilisée.

Avec cette méthode chaque individu a une chance d'être selectionné proportionnelle à sa performance, donc plus les individus sont adaptés au problème, plus ils ont de chances d'être séléctionnés.
Pour utiliser l'image de la "roue du forain", chaque individu se voit attribué un secteur dont l'angle est proportionnel à son adaptation, sa "fitness".
On fait tourner la roue et quand elle cesse de tourner on séléctionne l'individu correspondant au secteur désigné par une sorte de "curseur", curseur qui pointe sur un secteur particulier de celle-ci après qu'elle se soit arrêté de tourner.

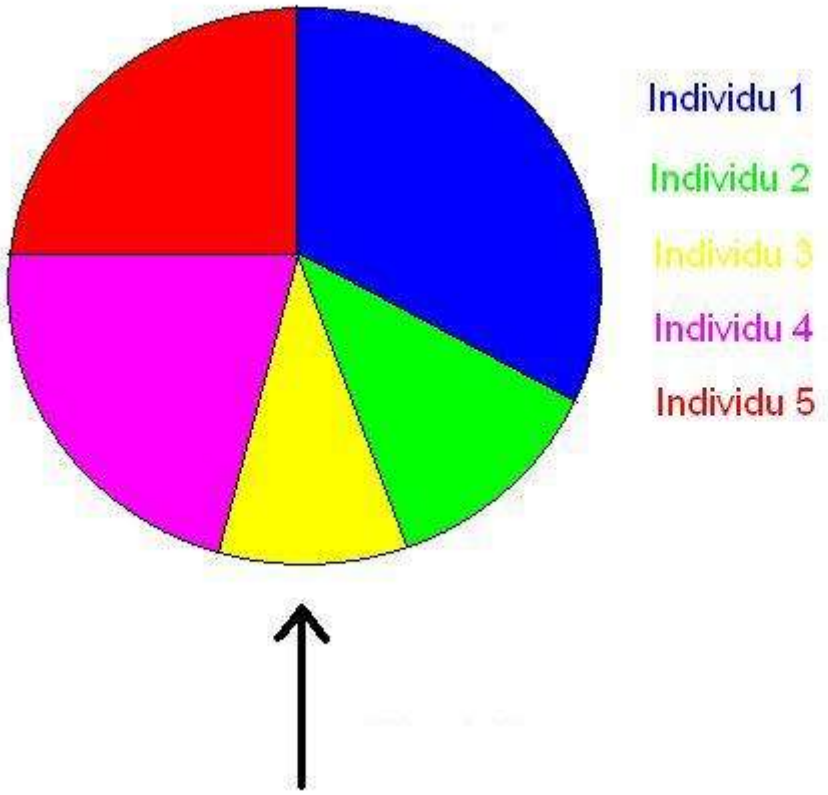

Figure 4: la méthode de sélection de la loterie biaisée

Cette méthode, bien que largement répandue, a pas mal d'inconvénients :

- En effet, elle a une forte variance. Il n'est pas impossible que sur n sélections successives destinées à désigner les parents de la nouvelle génération P', la quasi-totalité, voire pire la totalité des n individus séléectionnés soient des individus ayant une fitness vraiment mauvaise et donc que pratiquement aucun individu voire aucun individu a forte fitness ne fasse partie des parents de la nouvelle génération. Ce phénomène est bien sûr très dommageable car cela va complètement à l'encontre du principe des algorithmes génétiques qui veut que les meilleurs individus soient séléctionnés de manière à converger vers une solution la plus optimale possible.

- A l'inverse, on peut arriver à une domination écrasante d'un individu "localement supérieur". Ceci entrainant une grave perte de diversité. Imaginons par exemple qu'on ait un individu ayant une fitness très élévée par rapport au reste de la population, disons dix fois supérieure, il n'est pas impossible qu'après quelques générations successives on se retrouve avec une population ne contenant que des copies de cet individu. Le problème est que cet individu avait une fitness très élévée, mais que cette fitness était toute relative, elle était très élévée mais seulement en comparaison des autres individus. On se retrouve donc face à problème

  connu sous le nom de "convergence prématurée; l'évolution se met donc à stagner et on atteindra alors jamais l'optimum, on restera bloqué sur un optimum local.
  Il existe certaines techniques pour essayer de limiter ce phénomène, comme par exemple le "scaling", qui consiste à effectuer un changement d'échelle de manière à augmenter ou diminuer lde manière forcée a fitness d'un individu par rapport à un autre selon leur écart de fitness.
  Malgré tout, il est conseillé d'opter plutôt pour une autre méthode de séléction.

b) La méthode élitiste .

Cette méthode consiste à séléctionner les n individus dont on a besoin pour la nouvelle génération P' en prenant les n meilleurs individus de la population P après l'avoir triée de manière décroissante selon la fitness de ses individus.
Il est inutile de préciser que cette méthode est encore pire que celle de la loterie biaisée dans le sens où elle aménera à une convergence prématurée encore plus rapidement et surtout de manière encore plus sûre que la méthode de sélection de la loterie biaisée ; en effet, la pression de la séléction est trop forte, la variance nulle et la diversité inexistante, du moins le peu de diversité qu'il pourrait y avoir ne résultera pas de la sélection mais plutôt du croisement et des mutations.
Là aussi il faut opter pour une autre méthode de séléction.

c) La sélection par tournois :

Cette méthode est celle avec laquelle on obtient les résultats les plus satisfaisants.

Le principe de cette méthode est le suivant : on effectue un tirage avec remise de deux individus de P, et on les fait "combattre". Celui qui a la fitness la plus élevée l'emporte avec une probabilité p comprise entre 0.5 et 1. On répète ce processus n fois de manière a obtenir les n individus de P' qui serviront de parents.
La variance de cette méthode est élevée et le fait d'augmenter ou de diminuer la valeur de *p* permet respectivement de diminuer ou d'augementer la pression de la sélection.

d) La sélection universelle stochastique :

Cette méthode semble être très peu utilisée et qui plus est possède une variance faible, donc introduit peu de diversité, nous n'entrerons donc pas dans les détails, on se contentera d'exposer sa mise en oeuvre :
On prend l'image d'un segment decoupé en autant de sous-segments qu'il y a d'individus. Les individus sélectionnés sont désignés par un ensemble de points équidistants.

Section 3

L'opérateur de croisement ou crossover

Le crossover utilisé par les algorithmes génétiques est la transposition informatique du mécanisme qui permet, dans la nature, la production de chromosomes qui héritent partiellement des caractéristiques des parents.
Son rôle fondamental est de permettre la *recombinaison* des informations présentes dans le patrimoine génétique de la population.

Cet opérateur est appliqué après avoir appliqué l'opérateur de séléction sur la population P; on se retrouve donc avec une population P' de *n/2* individus et on doit doubler ce nombre pour que notre nouvelle génération soit complète.

On va donc créer de manière aléatoire *n/4* couples et on les fait se "reproduire".
Les chromosomes (ensembles de paramètres) des parents sont alors copiés et recombinés de façon à former deux descendants possédant des caractéristiques issues des deux parents.

Détaillons ce qui se passe pour chaque couple au niveau de chacun de leurs chromosomes :

Un, deux, voire jusqu'à *lg - 1* (où lg est la longueur du chromosome) points de croisements (loci) sont tirés au hasard, chaque chromosome se retrouve donc séparé en "segments". Puis chaque segment du parent 1 est échangé avec son "homologue" du parent 2 selon une probabilité de croisement p_c . De ce processus résulte 2 fils pour chaque couple et notre population P' contient donc bien maintenant *n* individus.

On peut noter que le nombre de points de croisements ainsi que la probabilité de croisement p_c permettent d'introduire plus ou moins de diversité.

En effet, plus le nombre de points de croisements sera grand et plus la probabilité de croisement sera élévée plus il y aura d'échange de segments, donc d'échange de paramètres, d'information, et plus le nombre de points de croisements sera petit et plus la probabilité de croisement sera faible, moins le croisement aportera de diversité.

Ci-dessous, un schéma illustrant un croisement en un point, un autre pour un croisement en deux points, et enfin un schéma représentant un croisement avec *lg - 1* points de croisements (on notera d'ailleurs sur ce schéma que l'échange d'un segement avec son homologue ne se fait pas toujours) :

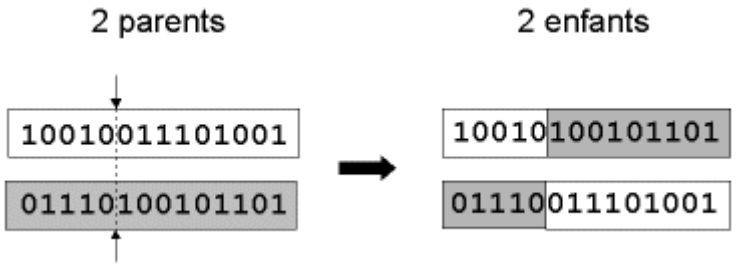

Figure 5: croisement avec un point de crossover

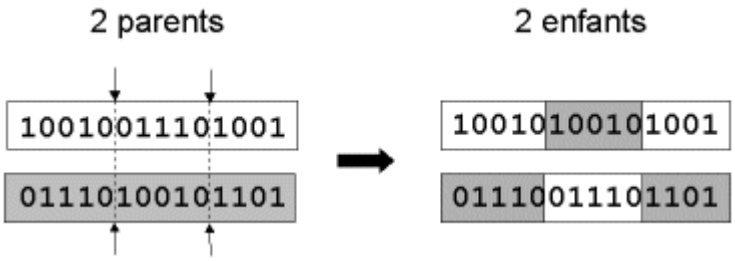

Figure 6: croisement avec 2 points de crossover

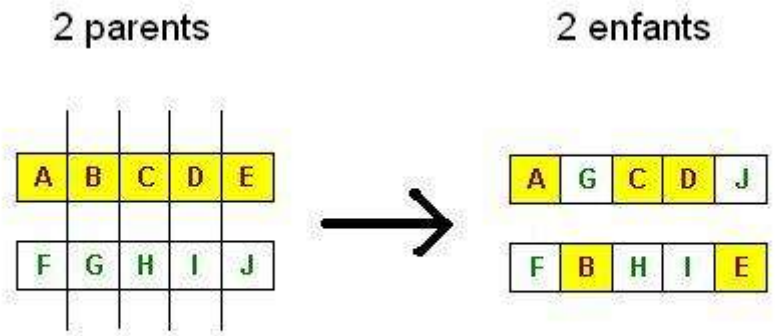

Figure 7: croisement uniforme

On peut citer aussi une autre méthode très utilisée dans le cas des problèmes modélisés par un codage binaire, il s'agit du ***croisement uniforme***. La mise en oeuvre de ce procédé est fort simple, elle consiste à définir de manière aléatoire un "masque", c'est-à-dire une chaîne de bits de même longueur que les chromosomes des parents sur lesquels il sera appliqué. Ce masque est destiné à savoir, pour chaque locus, de quel parent le premier fils devra hériter du gène s'y trouvant; si face à un locus le masque présente un 0, le fils héritera le gène s'y trouvant du parent n° 1, si il présente un 1 il en héritera du parent n° 2. La création du fils n° 2 se fait de manière symétrique : si pour un

gène donné le masque indique que le fils n° 1 devra recevoir celui-ci du parent n° 1 alors le fils n° 2 le recevra du parent n°2, et si le fils n° 1 le reçoit du parent n° 2 alors le fils 2 le recevra du parent n° 1.

L'opérateur de croisement favorise l'exploration de l'espace de recherche. En effet, considérons deux gènes A et B pouvant être améliorés par mutation. Il est peu probable que les deux gènes améliorés A' et B' apparaissent par mutation dans un même individu. Mais si un parent porte le gène mutant A' et l'autre le gène mutant B', l'opérateur de croisement permettra de combiner rapidement A' et B' et donc de créer un nouvel individu possèdant cette combianaison, combinaison grâce à laquelle il est possible qu'il soitt encore plus adapté que ses parents. L'opérateur de croisement assure donc le brassage du matériel génétique et l'accumulation des mutations favorables. En termes plus concrets, cet opérateur permet de créer de nouvelles combinaisons des paramètres des composants.

Malgré tout, il est possible que l'action conjointe de la sélection et du croisement ne permette pas de converger vers la solution optimale du problème.
En effet, imaginons que nous avons une population d'individus possédant un seul chromosome. Considérons un gène particulier de ce chromosome, on l'appelera G, gène ayant 2 allèles possibles :

0 et 1; si aucun individu de la population initiale ne possède l'allèle 1 pour ce gène, aucun croisement possible ne permettra d'introduire cet allèle pour notre gène G. Si la solution optimale au problème est telle que notre gène G possède l'allèle 1, il nous sera impossible d'atteindre cette solution optimale simplement par sélection et croisement.
C'est pour remédier entre autre à ce problème que l'opérateur de mutation est utilisé.

Section 4

L'opérateur de mutation

Cet opérateur consiste à changer la valeur allélique d'un gène avec une probabilité p_m très faible, généralement comprise entre 0.01 et 0.001.
On peut aussi prendre $p_m = 1 / lg$ où l*g* est la longueur de la chaîne de bits codant notre chromosome.

Une mutation consiste simplement en l'inversion d'un bit (ou de plusieurs bits, mais vu la probabilité de mutation c'est extrèmement rare) se trouvant en un locus bien particulier et lui aussi déterminé de manière aléatoire; on peut donc résumer la mutation de la façon suivante :

On utilise une fonction censée nous retourner *true* avec une probabilité p_m .

Pour chaque locus **faire**

Faire appel à la fonction

Si cette fonction nous renvoie *true* **alors**

on inverse le bit se trouvant à ce locus

FinSi

FinPour

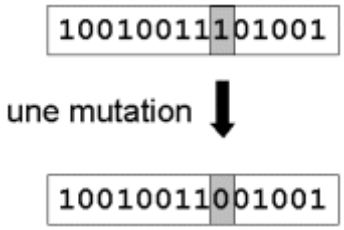

Figure 8 : une mutation

L'opérateur de mutation modifie donc de manière complètement aléatoire les caractéristiques d'une solution, ce qui permet d'introduire et de maintenir la diversité au sein de notre population de solutions. Cet opérateur joue le rôle d'un "élément perurbateur", il introduit du "bruit" au sein de la population.

Cet opérateur dispose de 4 grands avantages :

- Il garantit la diversité de la population, ce qui est primordial pour les algorithmes génétiques.

- Il permet d'éviter un phénomène connu sous le nom de ***dérive génétique***. On parle de dérive génétique quand certains gènes favorisés par le hasard se répandent au détriment des autres et sont ainsi présents au même endroit sur tout les chromosomes. Le fait que l'opérateur de mutation puisse entraîner de manière aléatoire des changements au niveau de n'importe quel locus permet d'éviter l'installation de cette situation défavorable.

- Il permet de limiter les risques d'une convergence prématurée causée par exemple par une méthode de sélection élitiste imposant à la population une pression sélective trop forte. En effet, dans le cas d'une convergence prématurée on se retrouve avec une population dont tous les individus sont identiques mais ne sont que des optimums locaux. Tous les individus étant identiques, le croisement ne changera rien à la situation. En effet, l'échange d'informations par crossover entre des individus strictement identiques est bien sûr totalement sans conséquences; on aura beau choisir la méthode de croisement qu'on veut on se retrouvera toujours à échanger des portions de chromosomes identiques et la population n'évoluera pas. L'évolution se retrouvant bloquée on n'attendra jamais l'optimum global.

  La mutation entrainant des inversions de bits de manière aléatoire permet de réintroduire des différences entre les individus et donc de nous extirper de cette situation.

  Il est quand même utile de garder à l'esprit que ceci n'est pas une solution "miracle" et qu'il est bien entendu plus intelligent de ne pas utiliser de méthodes de sélection connues pour entrainer ce type de problème.

- La mutation permet d'atteindre la propriété d' ***ergodicité***.

  L'ergodicité est une propriété garantissant que chaque point de l'espace de recherche puisse être atteint.

  En effet, une mutation pouvant intervenir de manière aléatoire au niveau de n'importe quel

locus, on a la certitude mathématique que n'importe quel permutation de notre chaîne de bits peut apparaître au sein de la population et donc que tout point de l'espace de recherche peut être atteint.

Grâce à cette propriété on est donc sûr de pouvoir atteindre l'optimum global.

On notera que la mutation règle donc le problème exposé à la fin du Section sur le croisement.

Section 5

L'opérateur de remplacement

Cet opérateur est le plus simple, son travail consiste à réintroduire les descendants obtenus par application successive des opérateurs de sélection, de croisement et de mutation (la population P') dans la population de leurs parents (la population P).

Ce faisant il vont remplacer une certaine proportion de ceux-ci, proportion pouvant bien sûr être choisie. Le rapport entre le nombre d'individus nouveaux allant être introduits dans la population P et le nombre d'individus de cette population est connu sous le nom de ***generation gap***.

On trouve essentiellemnt 2 méthodes de remplacement différentes :

- Le ***remplacement stationnaire :*** dans ce cas, les enfants remplacent automatiquement les parents sans tenir compte de leurs performances respectives, et le nombre d'individus de la population ne varie pas tout au long du cycle d'evolution simulé, ce qui implique donc d'initialiser la population initiale avec un nombre suffisant d'individus. Cette méthode peut être mise en oeuvre de 2 façons différentes :

    - La première se contente de remplacer la totalité de la population P par la population P', cette méthode est connue sous le nom de ***remplacement générationnel*** et on a donc un generation gap qui vaut 1.

    - La deuxième méthode consiste à choisir une certaine proportion d'individus de P' qui remplaceront leurs parents dans P (proportion égale à 100 % dans le cas du remplacement générationnel.

  Ce type de remplacement engendre une population ayant une grande variation et de se fait favorise la dérive génétique qui se manifeste d'autant plus que la population est de petite taille. De plus dans bien des cas, étant donné que même un enfant ayant une faible performance remplace forcement un parent, on n’atteint pas la meilleure solution mais on s’en approche

seulement.

- Le ***remplacement élitiste*** : dans ce cas, on garde au moins l'individu possédant les meilleures performances d'une génération à la suivante. En général, on peut partir du principe qu'un nouvel individu (enfant) prend place au sein de la population que s'il remplit le critère d'être plus performant que le moins performant des individus de la population précédente. Donc les enfants d'une génération ne remplaceront pas nécessairement leurs parents comme dans le remplacement stationnaire et par la même la taille de la population n'est pas figée au cours du temps.
  Ce type de stratégie améliore les performances des algorithmes évolutionnaire dans certains cas. Mais présente aussi un désavantage en augmentant le taux de convergence prématuré.

Néanmoins, des implémentations plus fines procèdent de manière différente. Dans ce cas là, le taux de remplacement n'est pas de 100 %, la taille de la population augmente donc au cours des générations successives, on dit qu'il y a ***overcrowding.*** Il faut donc trouver un moyen pour sélectionner les parents qui seront supprimés, qui vont mourir. De Jong a proposé la solution suivante : imaginons qu'on veuille remplacer 30 % des parents, soit *np* le nombre de parents correspondants à ce pourcentage, on remplacera les *np* parents les plus proches de leurs descendants de P'. Cette méthode permet donc premièrement de maintenir la diversité et deuxièmement d'améliorer la fitness globale de la population.

Section 6

Un petit exemple mettant en oeuvre tous les opérateurs

Le petit exemple suivant va permettre de mettre en oeuvre sur une génération tous les opérateurs étudies de manière à pouvoir observer le résultat de l'application successive de ces opérateurs.

Cet exemple est dû à Goldberg (1989). Il consiste à trouver le maximum de la fonction **f(x) = x** sur l'intervalle [0, 31] où *x* est un entier naturel.

On a 32 valeurs possibles pour *x* on choisit donc un codage discret sur 5 bits : on obtient donc la séquence 0,1,1,0,1 pour 13, la séquence 1,1,0,0,1 pour 25, etc...

On initialise la population initiale de manière aléatoire et on fixe sa taille à 4 individus. On définit simplement la fitness comme étant la valeur de *x*, vu qu'on en cherche la valeur maximum sur

l'intervalle [0, 31] plus la valeur de *x* sera élevée plus on se rapprochera du maximum de la fonction identité et donc plus la fitness sera grande. Soit la population initiale suivante :

Individu	Séquence	Fitness	% du total
1	01011	11	18.6
2	10011	19	32.2
3	00101	5	8.5
4	11000	24	40.7
Total		59	100

Figure 9: la population initiale

On opte pour une sélection par la méthode la loterie biaisée :

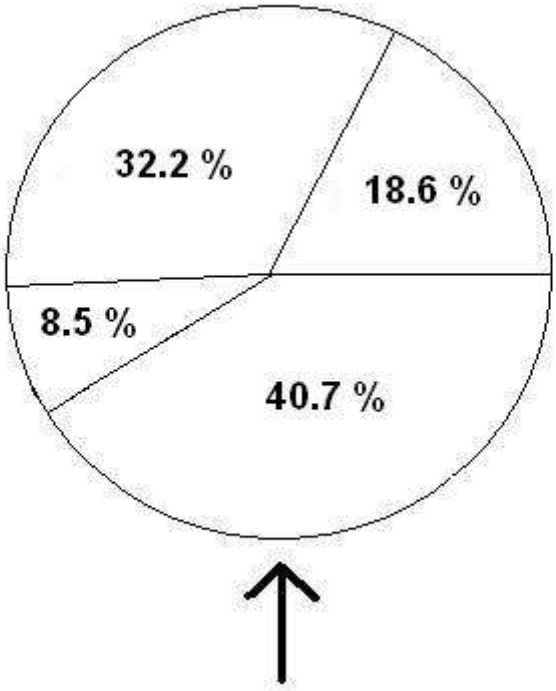

Figure 10: application de la méthode de sélection de la loterie biaisée sur la population

On fait tourner la roue 4 fois de suite, en général on fait tourner *n / 2* fois, soit 2 fois dans ce cas, mais le nombre 2 étant trop petit on décide de la faire tourner 4 fois. On obtient la nouvelle population :

Individu	Séquence
1	11000
2	00101
3	11000
4	01011

Figure 11: individus séléctionnés par la méthode de la loterie biaisée

On applique l'opérateur de croisement en utilisant un seul point de crossover. Normalement chaque couple donne 2 enfants qu'on ajoute à notre nouvelle population P' la faisant passer de *n / 2* individus à n individu, mais vu que dans le cas de notre exemple nous avons déjà atteint nos *n* individus, les 2 enfants du couple remplaceront leurs parents.

Deux couples sont formés de manière aléatoire :

- couple 1 : l'individu 2 avec l'individu 3
- couple 2 : l'individu 1 avec l'individu 4.

Les point de crossover sont eux aussi tirer au hasard.

On obtient le résultat suivant :

Parents	Enfants
00 101 01 011	01011 00101
110 00 010 11	01011 11000

Figure 12: résultat de l'application de l'opérateur de croisement avec un point de crossover sur les individus sélectionnés par la loterie biaisée

On applique l'opérateur de mutation qui choisit de manière aléatoire si on doit faire une mutation et sur quel locus la faire :

Chromosome avant mutation	Chromosome après mutation
01011	11011
00101	00101
01011	01111
11000	11000

Figure 13: résultat de l'application de l'opérateur de mutation sur les individus engendrés par croisement

Puis on applique l'opérateur de remplacement qui décide de remplacer 100% de la population P, la population P est donc entièrement remplacée par P' et sa taille reste fixe :

Individu	Séquence	Fitness	% du total
1	11011	27	38
2	00101	5	7
3	01111	15	21.1
4	11000	24	33.8
Total		71	100

Figure 14: la nouvelle population après application des différents opérateurs

En une seule génération le maximum est passé de 24 à 27, et la fitness globale de la population a relativement augmentée pour passer de 59 à 71. On s'arrête ici pour cet exemple mais dans la réalité on continuera à engendrer des générations successives jusqu'à obtenir le maximum global : 31.

Le seul intérêt de cet exemple était de montrer une trace simple, sur une génération, de l'application successive des différents opérateurs. Il est bien sûr évident qu'un tel "problème" ne nécessitait pas l'utilisation d'algorithmes génétiques.

Dans la partie suivante nous allons exposer différents problèmes, problèmes considérés comme difficiles, nécessitant un temps de calcul important avec une approche algorithmique classique, et exposer des mises en oeuvre possibles à l'aide d'algorithmes génétiques. On montrera en quoi les algorithmes génétiques améliorent la rapidité de résolution, ou carrément permettent une résolution qui n'aurait pas été possible autrement vu la taille des entrées du problème et sa complexité en temps.

II

Applications des algorithmes génétiques à des problèmes concrets

Section 7

Application au problème du voyageur de commerce

Nous allons maintenant nous interesser à une application plus concréte : le problème du voyageur de commerce. Cet exemple est un classique appartenant à la classe des problèmes NP-complets.

Il consiste à visiter un nombre N de villes en un minimum de distance sans passer deux fois par la même ville. Il s'agit donc d'optimiser le coût d'un parcours dans un graphe complet possédant un certain nombre de sommets, en passant une et une seule fois par chacun. Des méthodes déterministes existent déjà pour résoudre le problème, mais le temps de calcul est très long : elles reviennent à parcourir toutes les solutions possibles et à déterminer la moins coûteuse.

Le but sera ici de montrer comment modéliser le problème à partir d'algorithmes génétiques et des divers opérateurs que nous avons à disposition, qui ont été définis antérieurement.
Ceci nous conduira ensuite à montrer les avantages de cet outil par rapport à une résolution de type déterministe.

Représentation du problème :

Le problème du voyageur de commerce peut se modéliser à l'aide d'un graphe complet de *n*

sommets dont les arrêtes sont pondérées par un coût strictement négatif. Pour l'implantation de notre algorithme, l'instance sera modélisée comme distance euclidienne sur *n* points du plan, pour construire la matrice de coût.

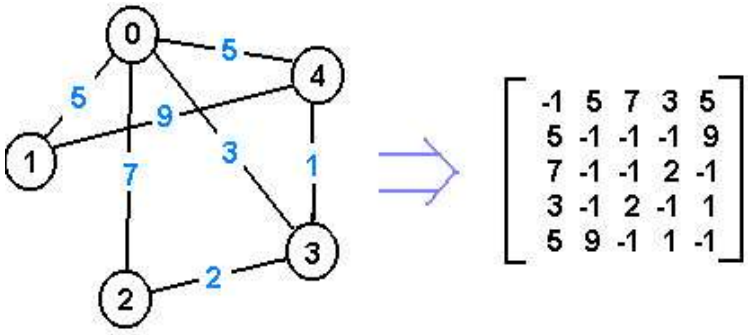

Figure 15 : modélisation du graphe sous forme de matrice

L'espace de recherche :

L'espace de recherche est l'ensemble des permutations de **{1, 2, ..., n }**.
Un point de cet espace de recherche est représenté par une de ces permutations.

Codage des points de l'espace de recherche :

Une première idée serait de coder chaque permutation (i.e : chaque point de l'espace de recherche) par une chaîne de bits.

Par exemple, pour *n* = 10 :

0011 0111 0000 0100 0001 0010 0101 0110

représente la permutation :

3 7 0 4 1 2 5 6

On s'aperçoit bien que chaque élément de l'ensemble de permutation est codé sur :

i = E (ln (*n*) / ln (2)) bits , E étant la fonction de partie entière.

Sachant qu'une permutation est de taille *n,* la chaine de bits sera alors de taille *n* * i.

Représentation d'une solution :

Comme nous l'avons déja dit le voyageur de commerce doit revenir à son point de départ et passer par toutes les villes une fois et une seule. Nous avons donc codé une solution par une structure de données comptant autant d'éléments qu'il y a de villes, c'est à dire une permutation. Chaque ville y apparait une et une seule fois. Il est alors évident que selon la ville de départ que l'on choisit on

peut avoir plusieurs représentations différentes du même parcours.

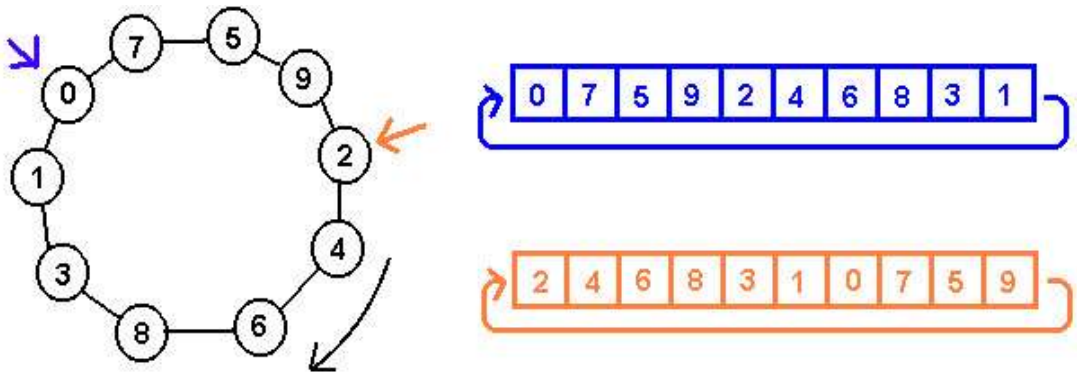

Figure 16 : codage d'une solution (ensemble de villes) dans un tableau .

Sélection :

Nous utilisons ici la méthode de sélection par roulette. On calcule d'abord la valeur moyenne de la fonction d'évaluation dans la population :

$$\overline{f} = \frac{1}{m} \sum_{i=0}^{m-1} f(P_i),$$

où P_i est l'individu i de la population et m la taille de la population. La place d'un individu P_i dans la roulette est proportionnel à $\frac{\overline{f}}{f(P_i)}$. On sélectionne alors $m/2$ individus pour la reproduction. Il y a aussi la possibilité d'avoir une politique d'«élitisme». C'est à dire qu'à chaque étape de sélection le meilleur chromosome est automatiquement sélectionné (cf. partie 2).

Croisement :

Etant donné deux parcours il faut combiner ces deux parcours pour en construire deux autres.

Nous pouvons suivre la méthode suivante :

1. On choisi aléatoirement deux points de découpe.
2. On interverti, entre les deux parcours, les parties qui se trouvent entre ces deux points.
3. On supprime, à l'extérieur des points de coupe, les villes qui sont déja placées entre les points de coupe.
4. On recense les villes qui n'apparaissent pas dans chacun des deux parcours.
5. On remplit aléatoirement les trous dans chaque parcours.

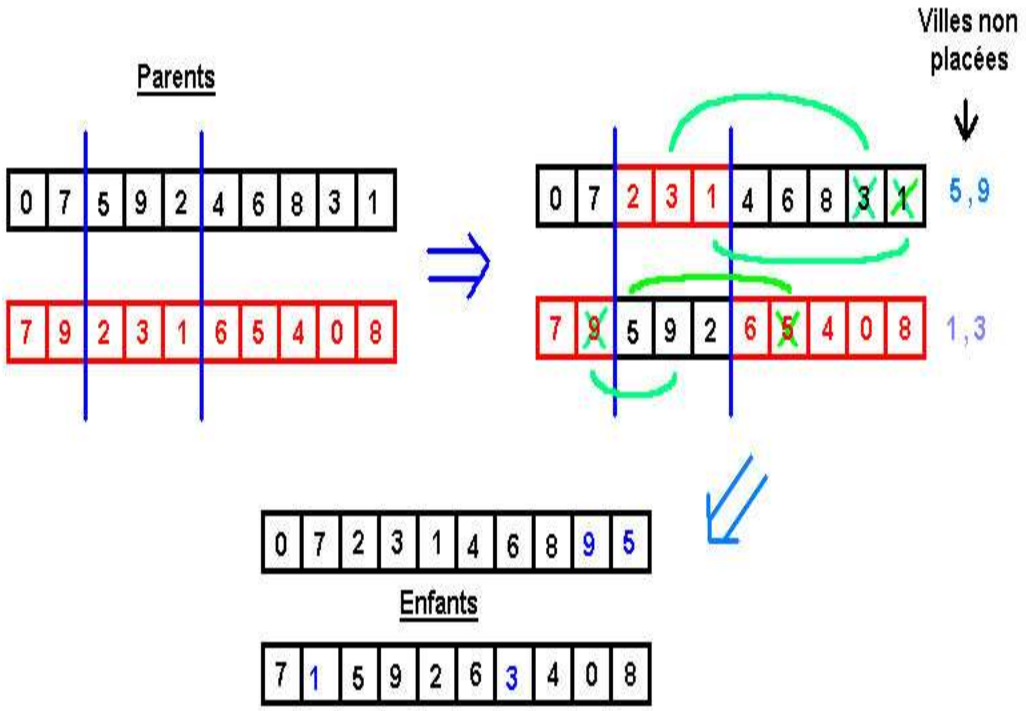

Figure 17 : exemple de croisement.

Mutation :

Il s'agit ici de modifier un des éléments d'un point de l'espace de recherche, soit d'une permutation. Dans notre cas, cela correspond donc à une ville. Quand une ville doit être mutée, on choisit aléatoirement une autre ville dans ce problème et on intervertit les deux villes.

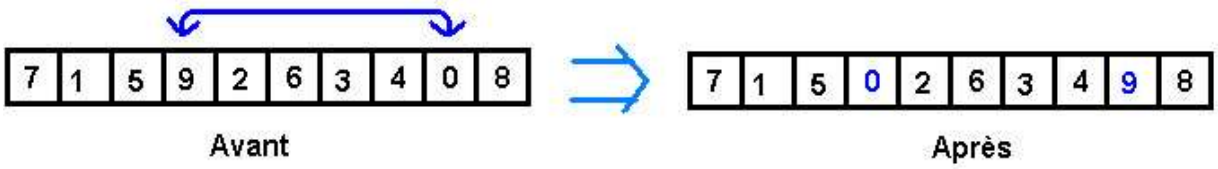

Figure 18 : exemple de mutation

Calcul de fitness ou dite fonction d'évaluation :

Le seul impératif sur la valeur d'adaptation est qu'elle soit croissante avec l'adaptation de la solution au problème. Un parcours valide renverra une valeur supérieur à un individu (solution potentielle) qui n'est pas solution au problème. On pourra décider par la suite qu'une solution non correcte sera de fitness négative et dans l'autre cas sera alors positive.

Lorsque cette dernière éventualité se produit, il faut que le comportement suivant soit respecté, à

savoir : plus notre chemin sera court, plus la fitness qui lui est associée sera forte.

On en déduit alors rapidement qu'il faut que la valeur d'adaptation varie dans le sens inverse de la distance correpondante au parcours. Par souci de simplicité, on pourra éventuellement choisir comme valeur d'adaptation, l'inverse de la longueur du parcours.

On s'aperçoit bien alors que cette algorithme est aléatoire (ou dit approximatif) dans le sens ou elle se base sur des méthodes de calculs non déterministes. L'expérience (i.e la programmation) montre que les résultats obtenus sont convaincants, nous parvenons à obtenir un ensemble de bonnes solutions en un temps raisonnable.

ALGORITHME ABSTRAIT :

Il s'agit dans un premier temps d'établir un ensemble de gênes G. Ceci étant fait, il faut alors définir deux fonctions, qui respectivement permettront de croiser deux ensembles de gênes et de réaliser une mutation sur deux ensembles de gênes.

```
Croiser(G g1,G g2){
	//résultat du croisement
	G r;
	//l'idée est tout simplement pour chaque variable de choisir
	  aléatoirement la variable de g1 ou de g2.
	Pour chaque variable v de r
		Si un nombre aléatoire de 0 à 99 est inférieur à 50 alors
			copier la variable correspondante à v de g1 dans v
		Sinon
			copier la variable correspondante à v de g2 dans v
		Fin Si
	Fin Pour

	Retourner r;
}
```

Et ci dessous la procedure de mutation :

```
Muter(G g){
	Pour chaque variable v de g
		Si un nombre aléatoire de 0 à 99 est inférieur à un certain nombre entre 0 et 99
// On echange deux variables car on ne peut avoir 2 villes semblables
   dans le meme parcours.
			Echanger v et vAutre;
	Fin Pour
}
```

Dès lors, il nous faut une procédure qui puisse « noter » les génomes ou permutations dénotant un parcours (dans le cas de notre exemple du voyageur de commerce).

Voici donc une procédure sélection qui renvoie les éléments les plus interessants d'un génomes (ce sera dans le cas du voyageur de commerce, les éléments de fitness la plus grande). Le choix est sinon laissé au programmeur.

```
SelectionNaturelle(G g[],N){
// On trie les éléments à la fitness la plus grande vers ceux elle est la plus
   petite.
      Trier g par les g[i] dans l'ordre décroissant.
      Retourner les N premiers éléments de g
}
```

A présent que nous avons tous les éléments dont nous avons besoin, nous pouvons construire l'algorithme final :

```
AlgoGene(G g[N]){
      // Tant qu'il y a des parcours .
      Tant que (g[i] != NULL)
       // N pour gi, N-1 pour gj .
            G r[N*(N-1)];
            k=0;
            Pour chaque gi dans g[N]
                  Pour chaque gj dans g[N]
                        Si gi!=gj alors
                              r[k]=Croiser(gi,gj);
                              Muter(r[k++]);
                        Fin Si
                  Fin Pour
            Fin Pour
            g=SelectionNaturelle(r,N);
      Fin Tant Que
}
```

Comparaison de complexité :

Ce problème est un représentant de la classe des problèmes NP-complets. L'existence d'un algorithme de compléxité polynomiale reste inconnue.

Les algorithmes pour résoudre le problème du voyageur de commerce peuvent être répartis en deux classes :

- les algorithmes déterministes qui trouvent la solution optimale

- les algorithmes d'approximation qui fournissent une solution presque optimale

complexité via une méthode de résolution classique (méthode déterministe) :

Un calcul rapide de la complexité montre qu'elle est en *O(n!)* où *n* est le nombre de villes.
En supposant que le temps pour effectuer un trajet est d' 1 µs, le tableau 1 témoigne de l'explosion combinatoire.

NB VILLES	***NB POSSIBILITES***	***TEMPS DE CALCUL***
5	120	120µs
10	181440	0.18 ms
15	43 MILLIARDS	12 h
20	60 E +15	1928 ans
25	310 E + 21	9,8 Milliards A.

Figure 19 : Nombre de possibilités de chemins et temps de calcul en fonction du nombre de villes

complexité via une méthode de résolution avec algorithmes génétiques (méthode approximative) :

Un calcul de complexité de l'algorithme abstrait nous montre qu'elle est en $O(n^2)$ ou *n* est le nombre de villes.

Détail du calcul : O (Algo Gene) = O(n ²) + O (Muter) + O (Croiser) + O(SelectionNaturelle)
= O (n²) + O(n) + O (2n) + O(n)
= O (n²)

NB VILLES	***NB POSSIBILITES***	***TEMPS DE CALCUL***
5	25	25 µs
10	100	100 µs
15	225	225 µs
20	400	400 µs

Figure 20 : Nombre de possibilités de chemins et temps de calcul en fonction du nombre de villes

Les résultats entre la méthode deterministe et approximative ne sont pas comparables en soi, elles utilisent des méthodes différentes pour arriver à des résultats différents. Comme dis précédemment, dans un cas, on cherche la solution optimale , dans le second (cf. tableau 2), une solution presque optimale.

On s'aperçoit bien à l'aide de ces calculs que pour parvenir à ces solutions on consomme respectivement un temps exponentiel (explosion combinatoire) et un temps polynomial (quadratique) par rapport au nombre de villes.

Maintenant que nous avons vu comment résoudre le problème du voyageur de commerce à l'aide des algorithmes génétiques, et que nous en avons fait les comparaisons avec une méthode de résolution classique, nous allons aborder le domaine d'application au data mining .

Section 8

Application au data-mining

Le data-mining consiste à extraire des informations intéressantes à partir d'une masse importante de données. L'exemple qu'on présentera ici est issu du travail de recherche de Wesley Romão, Alex A. Freitas et Itana M. de S. Gimenes. Dans cet exemple, la découverte d'informations intéressantes, de connaissances utiles, sera abordée sous l'angle de la recherche de règles de prédiction IF-THEN, forme de représentation très populaire dans le domaine du data-mining.
Une règle IF-THEN peut être décrite de la manière suivante : si un ensemble de conditions sont présentes, sont vérifiées, alors on peut en déduire une information, une connaissance.

Dans cet exemple on s'attachera à étudier un algorithme génétique conçu pour découvrir des règles de prédiction floues, c'est-à-dire représentées en ***logique floue***. Cet algorithme cherchera à découvrir des règles de prédiction qui sont intéressantes car nouvelles et surprenantes pour l'utilisateur. Ceci est mis en oeuvre en utilisant une technique se basant sur les goûts, les impressions de l'utilisateur. Plus précisément, une règle de prédiction sera considérée comme intéressante ou surprenante dans la mesure où elle apporte de l'information, de la connaissance qui non seulement n'était pas connue préalablement de l'utilisateur, mais qui plus est contredit ses impressions, ses goûts initiaux.
De plus, l'utilisation de la logique floue nous aide à améliorer la compréhensibilité des règles découvertes par l'algorithme génétique. Ceci est dû à l'utilisation de termes linguistiques qui sont,

par définition, et par opposition à un algorithme, naturelles, compréhensibles par l'utilisateur.

Comme précisé ci-avant, l'idée charnière du data-mining est d'extraire de la connaissance à partir de données et, dans notre exemple, on s'attachera à une tâche bien particulière du data-mining : la découverte de règles de prédiction. C'est-à-dire que l'utilisateur définit un but, but ayant plusieurs attributs dont on doit déterminer les valeurs respectives. Le rôle de l'algorithme génétique est donc de découvrir dess règles de prédiction, c'est-à-dire des règles qui à partir d'un antécédent : une conjonction de conditions, va pouvoir déterminer la valeur d'un attribut du but à atteindre. On suppose que l'ensemble des buts est relativement petit et déterminé par l'utilisateur en fonction de ses besoins, de ses centres d'intérêts.
On peut remarquer que cette tâche peut être considérée comme une généralisation d'une tâche bien connue du data-mining : la ***classification***. Dans la classification, les buts n'ont qu'un seul attribut alors qu'ici on autorise des buts ayant plusieurs attributs. On notera que bien que l'on autorise des

buts à plusieurs attributs, chaque règle ne prédit la valeur que d'un seul attribut.

La motivation principale pour utiliser des algorithmes génétiques pour la découverte de règles de prédictions floues intéressantes est due à leur capacité de réaliser une recherche globale et au fait qu'ils s'en sortent mieux avec les intéractions entre les différents attributs que les les algorithmes gloutons.

La justification du caractère intéressant et flou des règles découvertes par l'algorithme génétique est la suivante : en général, la logique floue est une méthode flexible pour traiter les incertitudes qu'on trouve typiquement dans les applications traitant des problèmes du monde réél. En particulier, dans le contexte du data-mining, la logique floue semble être une façon naturelle de traiter les attributs continus. En effet, en utilisant des termes linguistiques flous comme par exemple "bas" et "élevé", on peut plus naturellement représenter les conditions des règles impliquant des attributs continus que par discrétisation forcée de leurs valeurs. Par exemple, la condition floue :
$salaire = faible$ est plus naturelle pour un utilisateur que $salaire < 1300\ €$.

Bien qu'on utilise la logique floue pour améliorer la compréhensibilité des règles de déduction découvertes par l'algorithme génétique, le but n'est pas l'utilisation de la logique floue mais plutôt la découverte de règles intéressantes. En effet, de nombreux algorithmes de découverte de règles de prédiction se concentrent sur l'évaluation de la précision des règles de prédiction, sans chercher à savoir si les règles découvertes sont réellement eintéressante pour l'utilisateur. Il faut savoir qu'une règle de prédiction peut avoir une très grande précision mais ne pas être intéressante du tout pour l'utilisateur car elle représente une évidence, quelque chose qu'il sait déjà. Par exemple, la règle :
si le patient attend un enfant **alors** le patient est une femme, est bien évidemment une évidence pour l'utilisateur et par la même n'a aucun intérêt. En conséquence, cet exemple utilise un algorithme génétique qui recherche des règles de prédiction qui ne sont pas seulement précises mais qui sont aussi intéressantes, car surprenantes pour l'utilisateur, comme nous le verrons plus tard. Le coeur de l'algorithme génétique consiste donc à élaborer une fonction de fitness qui prenne en compte ces deux aspects qualitatifs pour évaluer une règle.

On évalue les règles, leur qualité, en effectuant une mise en correspondance floue entre ces règles et l'ensemble des données qu'on étudie au travers du data-mining, on utilise pour cela des fonctions

d'appartenance des conditions aux différents ensembles flous sur lesquels on travaille; ce point sera développé plus bas. On prend aussi en compte les impressions générales de l'utilisateur par rapport au domaine d'application, de manière à favoriser la découverte de règles surprenantes. On est donc dans une approche subjective de la découverte de règles de prédiction, car l'utilisateur intervient dans le processus de décision, l'utilisateur devant avoir alors une certaine connaissance du domaine d'application.Au contraire, l'approche objective utilise des méthodes de découverte de règles de prédiction et de mesure de leurs qualité indépendante de l'utilisateur.
L'approche objective est donc plus générale et autonome que l'approche subjective, alors que l'approche subjective a le grand avantage d'utiliser les connaissances de l'utilisateur pour guider la recherche des règles de prédiction. Par conséquent, si le domaine d'application est bien défini et qu'un utilisateur expert exécute cette application, il est intéressant d'utiliser cette approche.

Le problème étant maintenant posé et l'intérêt de cette approche ayant été exposé, on va tâcher d'en expliquer la mise en oeuvre de l'algorithme génétique sur lequel elle est basée.

Le codage :

Chaque individu représente une règle de prédiction. Plus précisément, chasue individu représente l'antécédent (partie IF) d'une règle de prédiction. La conséquence de la règle (la partie THEN) n'est pas encodée dans le génome. Elle est fixe tout au long de l'exécution de l'algorithme génétique, de sorte qu'à chaque exécution de l'algorithme, la totalité de la population est formée d' individus représentant des règles ayant la même conséquence, c'est-à-dire la même valeur prédite pour un attribut du but fixé par l'utilisateur.
De plus, les règles de prédiction représentées par les individus sont floues. On insiste sur le fait que seulement les antécédents de la règle sont "flouttés", c'est-à-dire voient les valeurs de leurs éventuels attributs continus être remplacées par un très petit nombre de termes linguistiques dits flous; cela ne concerne pas bien sûr les attributs dits catégoriels comme par exemple le sexe, la nationalité, etc... qui ne doivent bien sûr pas être flouttés. Les conséquences des règles, quant à elles, sont toujours précises.

Le génome d'un individu représente une conjonction de conditions spécifiant un antécédent de règle de déduction. Chaque condition est représentée par un gène qui consiste en un doublet attribut-valeur de la forme $A_i = V_{i,j}$ où A_i est le i-ème attribut et $V_{i,j}$ est la j-ème valeur appartenant au domaine de A_i .
De manière à simplifier l'encodage des conditions dans le génome, on utilise un encodage positionnel où la i-ème condition est encodée dans le i-ème gène. Par conséquent, il est seulement nécessairede représenter la valeur $V_{i,j}$ de la i-ème condition dans le génome puisque l'attribut de la i-ème condition est implicitement déterminée par la position du gène dans le génome.
De plus, chaque antécédent contient un flag booléen qui indique si la i-ème condition est présente ou pas dans l'antécédent de la règle. Par conséquent, bien que tous les individus aient la même longueur pour leur génome, des individus différents représentent des règles de longueurs différentes selon la valeur des flags booléens.
Cette représentation flexible est bien-sûr souhaitable dans des règles de prédiction. Comme personne ne connait, a priori, combien de conditions seront nécessaires pour créer une bonne règle de prédiction, ce nombre doit être ajusté automatiquement par l'algorithme génétique en se basant sur l'ensemble de données qu'on est en train d'étudier.
La structure du génome d'un individu est illustrée ci-dessous avec *m* le nombre d'attributs qu'on analyse.

Figure 21 : structure du génome d'un individu représentant un antécédent de règle

Floutter des attributs continus :

Dans notre algorithme génétique, le flouttage des attributs continus est réalisé de la manière suivante : chaque attribut continu est associé avec 2 ou 3 termes linguistiques qui correspondent aux valeurs des attributs flouttés, comme par exemple {bas, élevé} ou {bas, moyen, élevé}. Quand on passe d'un attribut ayant des valeurs continues à un attribut ayant des valeurs floues on perd bien sûr de la précision car on passe d'un nombre de valeurs infini à typiquement 3 valeurs relatives. Donc, et contrairement à la théorie classique des ensembles ou un élément donné soit appartient soit n'appartient pas un ensemble, avec la logique floue on travaille sur des ensembles flous et il est donc clair qu'un élément appartient à un ensemble flou "dans une certaine mesure", selon sa valeur continue; par exemple si les valeurs 80 et 90 de l'attribut âge appartiennent toutes deux à l'ensemble flou "élévé", il est clair que la valeur 80 y appartient "plus" que la valeur 90. On attend donc de l'utilisateur qu'il nous fournisse, après avoir défini le nombre et la nature des termes linguistiques à utiliser pour réaliser le flouttage, des fonctions d'appartenance pour chaque ensemble flou. Ces fonctions sont chargés de déterminer, pour chaque ensemble, qu'elles sont les anciennes valeurs continues qu'y maintenant doivent être considérées équivalentes au terme linguistique qu'il représente (l'ensemble), et dans quelle mesure.

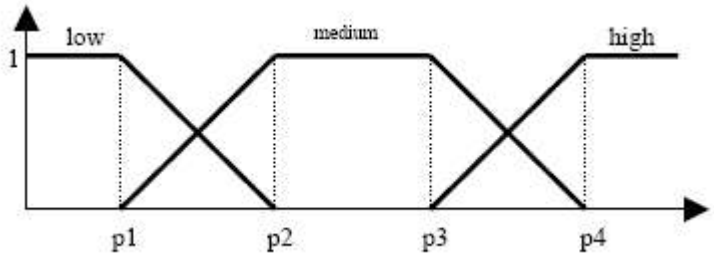

Figure 22 : allure des fonctions d'appartenance

Le schéma ci-dessus illustre le cas où un attribut continu peut être floutté en 3 valeurs linguistiques différentes. On remarque que les fonctions d'appartenance aux 3 valeurs linguistiques différentes, aux 3 ensembles, sont spécifiées par 4 paramètres notés p_1 , p_2 , p_3 et p_4 avec $p_1 < p_2 < p_3 < p_4$. Chaque paramètre représente une valeur continue, qui est utilisée pour spécifier les coordonnées de 2 sommets de trapèze appartenant à un doublet de fonctions d'appartenance "adjacentes". Cette représentation impose de nombreuses contraintes souhaitables sur les ensembles flous. Premièrement, chaque valeur de départ (continue) d'un attribut appartient à au moins un des ensembles flous. Deuxièmement, chaque ensemble flou est unimodal et normal, la condition de normalité veut dire que le degré d'appartenance maximum de tout élément est 1. Troisièmement, il contient un petit nombre un petit nombre de valeurs linguistiques. En général, ces contraintes aident à ce que les ensembles flous aient une bonne compréhensibilité pour l'utilisateur.

La fonction de fitness :

On rappelle que dans la grande majorité des applications de data-mining, le critère principal d'évaluation de la qualité d'une règle est la précision de sa prédiction. Dans le cas de notre exemple, ce critère est important mais ce n'est pas le seul, en effet une règle peut être précise mais totalement inutile. Pour éviter cela, la fonction de fitness de notre exemple prend en compte à la fois le critère de précision de la règle qu'on notera *Acc* et le critère d'intérêt, de surprise qu'on notera *Surp*. Ces deux critères sont combinés en une seule et unique formule de la façon suivante :

Fitness(i) = Acc(i) * Surp(i)

Nous allons mettant nous attacher à exposer comment calculer le degré de précision et le degré de surprise d'une règle de prédiction.

Mesure de la précision :

La première étape pour mesurer la précison prédictive d'une règle floue est de calculer la mesure dans laquelle un individu appartient à un antécédent de règle, c'est-à-dire une conjonction de conditions. On utilise l'operateur flou ET où le degré d'appartenance d'un individu à un antécédent de règle est donné par :

$$\min_{i=1}^{z}(\mu_i)$$

où μ_i représente le degré d'appartenance de l'individu à la i-ème condition de l'antécédent, *z* est le nombre de conditions dans l'antécédent. Le degré d'appartenance d'un individu à la i-ème condition est directement déterminé par la valeur (entre 0 et 1) de la fonction d'appartenance de l'attribut de la condition à son ensemble flou. Du coup, les conditions strictes (c'est-à-dire catégorielle, non continues) peuvent seulement avoir un degré d'appartenance valant 0 ou 1.

Par exemple, considérons un antécédent de règle avec les 2 conditions suivantes :
age = bas ET sexe = femme, où la première condition est floue et la seconde est stricte. Supposons que l'exemple donné ait les valeurs *23* et *femme* pour ces deux attributs respectifs. Supposons aussi que la fonction d'appartenance pour le terme linguistique *bas* renvoie 0.8 pour la valeur 23 et renvoie donc 1 pour l'attribut catégoriel *femme*. Alors le degré d'appartenance de cet individu à cet antécédent de règle est min(0.8, 1) = 0.8 .

Soit A l'antécédent d'une règle donnée. Une fois qu'a été calculé le degré d'appartenance de chaque exemple à A, la précision du i-ème individu (règle floue) notée *Acc(i)* est calculée par la formule :

$$Acc(i) = \frac{(CorrPred - \frac{1}{2})}{TotPred}$$

Dans cette formule, *CorrPred* (le nombre de précisions correctes) est la somme des degrés d'appartenance à A de tous les individus qui ont la valeur V_i, j prédite par la règle. *TotPred* (nombre total de prédictions) est la somme des degrés d'appartenance à A de tous les individus.

La raison pour laquelle on soustrait $\frac{1}{2}$ de *CorrPred* est de pénaliser les règles trop spécifiques, qui s'adaptent trop aux données. Par exemple, supposons qu'on ait un *CorrPred* valant 1 et un *TotPred* valant lui aussi 1. Si on ne soustrait pas $\frac{1}{2}$ de *CorrPred*, la formule renverra une précision de prédiction de 100 % pour la règle, ce qui est intuitivement une estimation sur-optimiste de sa précision. Cependant, si on soustrait $\frac{1}{2}$ à *CorrPred*, la formule renverra une précision de 50 %, ce qui semble être une estimation plus plausible de la précision, étant donné que la règle est trop spécifique. EN clair, pour de grandes valeurs de *CorrPred* et de *TotPred*, la soustraction de $\frac{1}{2}$ n'aura pas une influence significative sur la valeur retournée par la formule, de sorte que la soustraction ne pénalise quer les règles très spécifiques, qui ne couvrent qu'un nombre très limité d'individus.

Mesure du degré de surprise d'une règle :

On rappelle qu'une règle est considérée comme surprenante pour l'utilisateur si il ne la connaissait pas mais qui plus est contredit ses impressions, ses croyances de départ. On utilise donc une approche subjective, l'utilisateur nous fournit au départ, avant de lancer l'algorithme génétique, ses impressions, ses connaissances, ses croyances, ce qu'il pense être vrai sur le sujet qu'on étudie au travers du data-mining. Les impressions générales, de même que les règles de prédiction, sont exprimées sous la forme de règles IF-THEN. La différence principale entre les deux est que les premières sont spécifiées manuellement par l'utilisateur alors que les secondes sont découvertes automatiquement par l'algorithme génétique.

Soit $\{R_1, \ldots, R_i, R_{|R|}\}$ l'ensemble des règles dans la population actuelle où $|R|$ est le nombre de règles, d'individus de la population, et soit $\{GI_1, \ldots, GI_j, GI_{|GI|}\}$ l'ensemble des impressions générales (initiales) de l'utilisateur, où $|GI|$ est le nombre de ces impressions initiales. On remarquera que cet enesemble est spécifié par l'utilisateur avant de lancer l'algorithme génétique et ne change pas au cours de l'exécution de l'algorithme.

Da manière à calculer le degré de surprise d'une règle (individu) de la population, chaque règle est mise en correspondance avec chaque impression générale de l'uilisateur.

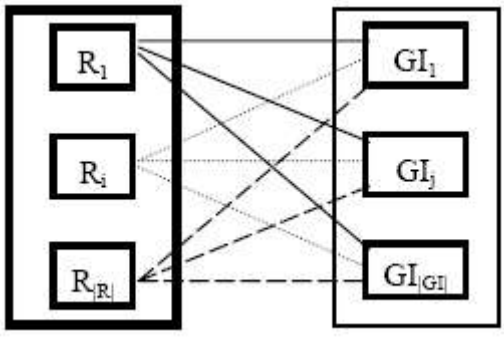

Figure 23 : Mise en correspondance de chaque règle avec chaque impression générale

Une règle est considérée comme surprenante, contredisant les impressions générales de l'utilisateur, dans la mesure où R_i et GI_j ont des antécédents similaires mais des conséquences contradictoires. En d'autres termes, plus la similarité entre les antécédents de R_i et de GI_j est importante et plus le degré contradictoire des conséquences de R_i et de GI_j est important, plus le degré de surprise de R_i par rapport à GI_j est élevé.

Pour chaque paire de règles R_i et GI_j, avec *i* variant de 1 à $|R|$ et j variant de 1 à $|GI|$, l'algorithme génétique calcule le degré de surprise de R_i par rapport à GI_j en trois étapes :

1ère étape : trouver les impressions générales dont les conséquences contredisent la conséquence de R_i.

La conséquence de R_i contredit la conséquence d'une impression générale GI_j si et seulement si R_i et GI_j ont le même attribut de but mais une valeur différente pour cet attribut. Par exemple, cela se produirait dans l'exemple suivant : si R_i prédit que "salaire = bas" et que GI_j prédit quant à lui que "salaire = élevé".
Si R_i et GI_j prédisent des attributs de but différents ou si ils prédisent la même valeur pour le même attribut de but, alors elles ne sont pas considérés comme contradictoires. Dans ce cas, le degré de surprise de R_i par rapport à GI_j est considéré comme nul (il vaut 0), et l'étapes 2 et 3 sont ignorées.

2ième étape : calculer le degré de similarité entre les antécédents de R_i et GI_j

Pour chaque impression générale GI_j trouvée à l'étape précédente (c'est-à-dire pour chaque impression générale GI_j contredite par R_i), on calcule le degré de similarité entre les antécédents de R_i et de GI_j. Ce degré de similarité, noté $AS_{(i,j)}$, est calculé par la formule suivante :

$$AS_{(i,j)} = \frac{|A_{(i,j)}|}{(max(|R_i|, |GI_j|))}$$

où $|R_i|$ est le nombre de conditions (doublets attribut-valeur) de la règle R_i, $|GI_j|$ est le nombre de conditionsdans l'impression générale GI_j et $|A_{(i,j)}|$ est le nombre de conditions qui sont exactement identiques (c'est-à-dire qui ont le même attibut et la même valeur pour cet attribut) dans R_i et GI_j.

3ième étape : calculer le degré de surprise de R_i par rapport à GI_j.

Soit *Surp(i,j)* le degré de surprise de R_i par rapport à GI_j. *Surp(i,j)* dépend de $AS_{(i,j)}$, calculée dans la deuxième étape, et de la différence entre les conséquences de R_i et GI_j calculée dans la première étape.
Les valeurs de l'attribut de but dans les conséquences de R_i et de GI_j peuvent être soit une

valeur de l'ensemble {bas, élevé} soit une valeur de l'ensemble {bas, moyen, élevé}, cela dépend de l'attribut de but (le choix entre ces 2 domaines pour un attribut est fait par l'utilisateur pour chaque attribut).
Si la différence entre les conséquences de R_i et GI_j est que l'une vaut *bas* et l'autre vaut *élevé*, caractérisant la plus grande différence possible entre ces conséquences, alors *Surp(i,j)* sa voit assignée la valeur de $AS_{(i,j)}$. Si la différence entre les conséquences de R_i et GI_j est que l'une vaut *moyen* et l'autre vaut *bas* ou *élevé*, caractérisant une différence plus limitée entre ces conséquences, alors *Surp(i,j)* se voit assignée la moitié de la valeur de $AS_{(i,j)}$, c'est-à-dire que $Surp(i,j)=0.5*AS_{(i,j)}$.

Une fois que ces 3 étapes ont été faites pour toutes les impressions générales, par rapport à une règle donnée R_i , ont été calculé tous les degrés de surprise de R_i par rapport à chaque impression générale de l'utilisateur. On est donc en mesure de calculer le degré de surprise de R_i par la formule suivante :

$$Surp(i)=max\left[\sum_{j=1}^{|GI|} AS_{(i,j)}\right]$$

Sélection, croisement et mutation et autres opérateurs :

L'algorithme génétique fait appel à la méthode des tournois pour effectuer une sélection, cela fonctionne de la manière suivante : tout d'abord *k* individus sont choisis au hasard et avec remise (typiquement 2 dans notre cas). L'individu ayant la plus grande fitness parmi les *k* individus tirés au hasard gagne le tournoi. Ce processus est répété *P* fois où *P* est la taille de la population, puis les *P* individus se voient appliquer l'opérateur de croisement puis celui de mutation.

L'algorithme génétique utilise des opérateurs de croisement et de mutation relativement simples.

Pour le croisement il utilise la méthode du croisement uniforme. On fixe une probabilité de croisement entre 2 individus et une probabilité d'échange entre chaque paire de gènes (les 2 gènes représentant une valeur particulière pour le même attribut). La première est fixée à 0.85 et la seconde à 0.5 . Le choix du croisement uniforme est motivé par le fait qu'avec cette méthode la probabilité d'échange chaque paire de gènes, d'attributs, est indépendante de leur position dans le génome. Ceci est souhaitable dans notre exemple où les antécédents de règle représentés par le génome consiste en une ensemble non-ordonné de conditions.

L'opérateur de mutation se contente quant à lui de transformer de manière aléatoire la valeur d'un attribut (d'un gène) en une autre valeur appartenant au domaine de l'attribut.

En plus des opérateurs de croisement et de mutation, l'algorithme génétique utilise également des opérateurs chargés d'insérer ou de supprimer certaines conditions d'une règle. Le premier fait passer à *true* le booléen d'un gène (d'une condition d'antécédent), la rendant par la même présente, alors que le second le fait passer à *false,* supprimant cette condition de l'antécédent de la règle. Ces 2 opérateurs réalisent donc respectivement une opération de spécialisation et de généralisation. En conséquence de quoi ils contribuent à une exploration plus large de l'espace de recherche, facilitant l'exploration de zones de l'espace de recherche qui n'aurait pas été accessible aussi facilement en

utilisant seulement le croisement et la mutation.

Algorithme abstrait :

Spécifier, avec l'aide de l'utilisateur, les fonctions d'appartenance des attributs devant être flouttés.

Obtenir les impressions générales de l'utilisateur.

PourChaque conséquence de règle (doublet attribut de but - valeur) **faire**

Calculer la fréquence relative de la valeur de cet attribut de but dans l'ensemble qu'on étudie au travers du data-mining.

Générer aéatoirement la population.

Appeler la fonction Calcule_Fitness.

Pour g **de** 1 **à** nombre de générations à engendrer **faire**

Générer une nouvelle population.

Appliquer l'opérateur de sélection.

Appliquer l'opérateur de croisement.

Appliquer l'opérateur de mutation.

Appliquer l'opérateur d'insertion de conditions dans une règle.

Appliquer l'opérateur de suppression de conditions d'une règle.

Appeler la fonction Calcule_Fitness.

FinPour

Renvoyer à l'utilisateur la règle ayant la fitness maximale avec les contraintes suivantes : (Surp > 0) ET (Acc > max(0.5, FreqRel))

FinPourChaque

Procédure Calcule_Fitness

Calculer la précision (Acc) de chaque règle (individu) en réalisant une mise en correspondance floue entre chaque règle et les données de l'ensemble étudié au travers du data-mining.

Calculer le degré de surprise (Surp) de chaque règle en mettant en correspondance la règle avec chaque impression générale de l'utilisateur.

Calculer la fitness de chaque règle.

FinProcédure

Chaque itération de la boucle principale (la boucle PourChaque) cherche à découvrir la meilleure règle possible (au niveau de la précision et de la surprise) prédisant une conséquence donnée. L'algorithme renvoie une règle pour chaque conséquence devant être prédite. La règle renvoyée doit satisfaire à 2 conditions : Surp > 0 et ACC > max(0.5, FreqRel). La première condition requiert

simplement que le degré de surprise de la règle soit supérieur à 0, de sorte à ce que cette règle puisse être au moins considérée comme potentiellement intéressante à défaut de l'être réellement. L'intérêt de la deuxième condition peut être exposé en étudiant 2 cas bien précis.

Premier cas :

supposons que la fréquence relative (dans l'ensemble de données sur lequel on travaille) de la valeur

de l'attribut de but prédit par cette règle soit supérieure à 0.5. Il est alors logique de demander à ce qu'une règle prédite par l'algorithme génétique ait une précision supérieure à sa fréquence relative d'apparition dans l'ensemble de données sur lequel on travaille.

Deuxième cas :

suppossons maintenant que la fréquence relative de la valeur de l'attribut du but prédit par cette règle soit inférieure à 0.5. Il est toujours logique de demander à ce que la précision de la règle prédite par l'algorithme génétique soit supérieure à sa fréquence relative d'apparition dans l'ensemble de donnés qu'on étudie, mais maintenant cette condition est beaucoup plus facile à satisfaire. Par exemple si la fréquence relative vaut 0.2 et que la précision de la règle produite par l'algorithme génétique vaut 0.3 alors la condition (précision > fréquence relative) serait satisfaite, mais la règle resterait quand même une mauvaise règle car ayant une précision faible. C'est pour cela qu'on force la précision de la règle devant être prédite à être supérieure à 0.5, pour imposer que les règles prédiction découvertes par l'algorithme génétique aiant toujours une précision acceptable, raisonnable. La condition Acc > max(0.5, FreqRel) nous permet donc d'imposer ces deux contraintes en une formule simple et concise.

Résultats :

Les 3 chercheurs ont effectué des tests de leur algorithme génétique sur une base de données de 5690 enregistrements. Chaque enregistrement contenait des attributs décrivant un chercheur et sa production scientifique. De cette base de données ils ont dégagés 24 attributs et toutes les enregistrements de la base de données ne contenant pas une valeur pour chacun de ces 24 attributs ont été éliminés.

Parmi ces 24 attributs, 6 ont été utilisés comme des attributs dont on devait prédire la valeur à l'aide d'une règle de prédiction que l'algorithme génétique devait se charger de découvrir, et les 18 autres ont été utilisés comme des attributs servant à déduire les valeurs des 6 autres.

Parmi les 18 attributs servant à prédire les valeurs des 6 autres, 8 étaient catégoriel et donc stricts :

- la nationalité du chercheur,
- son continent d'origine,
- son sexe,
- l'état dans lequel il/elle vit,
- la ville dans laquelle il/elle vit,
- son niveau d'anglais écrit,
- si il/elle a déjà été à la tête d'une équipe de recherche,
- son domaine de recherche.

Les 10 autres étaient continus et ont donc été flouttés en 2 termes linguistiques {low, high} ou en 3 termes linguistiques {low, medium, high}, grâce aux informations données par l'utilisateur (fonctions d'appartenance); voici la liste de ces 10 attibuts continus :

- le niveau d'instruction,
- le nombre d'années depuis le dernier diplôme obtenu,
- l'âge,
- le nombre de projets techniques menés à bien,
- le nombre de cours donnés,
- le nombre de thèses supervisées,
- le nombre de masters supervisés,
- le nombre d'essais de recherche supervisés (à un niveau diplômant),
- le nombre de projets de fin de premier cycle supervisés,
- le nombre d'étudiants de premier cycle encadrés avec un projet de recherche.

Les 6 attributs de but dont on doit prédire les valeurs pour un chercheur donné en se basant sur les informations dont on dispose dans la base de données, notés G_1 , ... , G_6 ,sont les suivants :

- G_1 est le nombre d'articles publiés dans des journaux nationaux, il peut prendre trois valeurs floues différentes : low, medium, ou high.
- G_2 est le nombre d'articles publiés dans des journaux internationaux, il peut prendre trois valeurs floues différentes : low, medium, ou high.
- G_3 est le nombre de chapitres écrits dans des livres édités au niveau national, il peut prendre trois valeurs floues différentes : low, medium, ou high.
- G_4 est le nombre de chapitres écrits dans des livres édités à un niveau international, il peut prendre 2 valeurs floues différentes : low ou hogh,
- G_5 est le nombre de livres écrits qui sont édités au niveau national, il peut prendre 2 valeurs floues différentes : low ou high.
- G_6 est le nombre de livres écrits qui sont édités à un niveau international, il peut prendre 2 valeurs floues : low ou high.

Tous les résultats obtenus ont été comparés avec ceux obtenus par un algorithme de construction d'arbres de décision très connu du data-mining, mais qui n'est pas un algorithme génétique, l'**algorithme J4.8**, voivi les résultats qu'ils ont obtenus :

Goal attrib.	Predicted value	Freq. (%)	J4.8 (%)	GA (%)
G_1	low	46.9	**64.9**	58.8 ±14.0
	medium	50.6	**63.9**	60.4 ±13.3
	high	2.5	**9.1**	00.0 ±0.0 (-)
G_2	low	64.2	76.6	**90.7 ±5.4** (+)
	medium	29.7	**45.3**	40.0 ±16.3
	high	6.1	**32.2**	25.0 ±13.4
G_3	low	76.9	82.2	**95.2 ±3.0** (+)
	medium	21.2	45.3	**56.7 ±15.8**
	high	1.9	**27.4**	25.0 ±13.4
G_4	low	93.2	93.4	**98.4 ±0.7** (+)
	high	6.8	**51.7**	14.3 ±10.4 (-)
G_5	low	83.5	86.0	**89.5 ±9.9**
	high	16.5	54.7	**56.9 ±14.0**
G_6	low	97.9	97.9	**98.9 ±0.5** (+)
	high	2.1	0.0	00.0 ±0.0

Figure 24 : Taux de précison des prédictions réalisées par l'algorithme génétique et l'algorithme J4.8

La première colonne identifie l'attribut de but prédit par la règle de prédiction découverte, la deuxième identifie la valeur prédite pour cet attribut, la troisième donne la valeur de la fréquence relative de cet attribut de but dans la base de données qu'on étudie, enfin la quatrième et la cinquième colonne donne le taux de précision de la prédiction effectuée respectivement par l'algorithme J4.8 et par l'algorithme génétique.On voit que sur ce point là les résultats fournit par les 2 algorithmes se valent.

Rule consequent	Interestingness for the user	
	J4.8	GA
G_1 = low	low	**high**
G_1 = medium	*medium*	*medium*
G_1 = high	low	*medium*
G_2 = low	low	**high**
G_2 = medium	low	*medium*
G_2 = high	low	low
G_3 = low	low	**high**
G_3 = medium	low	low
G_3 = high	low	low
G_4 = low	*medium*	*medium*
G_4 = high	low	*medium*
G_5 = low	low	**high**
G_5 = high	low	low
G_6 = low	N/A	**high**
G_6 = high	N/A	low

Figure 25: Mesure de l'intérêt des règles découvertes par l'algorithme J4.8 et de celles découvertes par l'algorithme génétique.

La première colonne identifie l'attribut de but prédit par la règle de prédiction découverte, la seconde et la troisième donne le degré d'intérêt (surprise et contradiction des impressions générales, des connaissances initiales de l'utilisateur) des règles découvertes respectivement par l'algorithme génétique J4.8 et par l'algorithme génétique.

Sur ce point la supériorité de l'algorithme génétique sur l'algorithme J4.8 apparait nettement.

Au travers de ces résultats les chercheurs ont observés que plus une règle est simple (c'est-à-dire qu'elle a un petit nombre de conditions dans son antécédent), plus elle est intéressante pour l'utilisateur. Les chercheurs ont alors comparé l'intérêt d'une règle en fonction de son nombre du nombre de conditions qu'elle contient dans son antécédent et là l'avantage de l'algorithme génétique sur l'algorithme J4.8 est vraiment flagrant.

Number of conditions	Interestingness for the user			Total
	low	medium	high	
1	1	1	0	2
2	0	0	0	0
3	1	1	0	2
4	6	0	0	6
5	3	0	0	3
0	11	2	0	13

Figure 26: Analyse de la simplicité d'une règle découverte par l'algorithme J4.8 en fonction du nombre de conditions qu'elle contient dans son antécédent.

Number of conditions	Interestingness for the user			Total
	low	medium	high	
1	0	2	4	6
2	2	1	1	4
3	1	2	0	3
4	1	0	0	1
5	1	0	0	1
Total	5	5	5	15

Figure 27: Analyse de la simplicité d'une règle découverte par l'algorithme génétique en fonction du nombre de conditions qu'elle contient dans son antécédent.

En conclusion, au niveau de la précision des règles de prédictions découvertes, les deux algorithmes se valent, mais en ce qui concerne l'intérêt pour l'utilisateur de ces règles, l'algorithme génétique présente un intérêt flagrant sur l'algorithme J4.8, et par la même son choix semble s'imposer comme une évidence dans le cadre d'un data-mining guidé par l'utilisateur comme par exemple celui effectué par des sites de vente en ligne comme Amazon. En effet, sur ce site, les goûts, les impressions générales de l'utilisateur, de l'acheteur, sont déterminés en se basant sur les articles qu'il achète et sur les articles dont il consulte la page. Le site tente alors de proposer à l'acheteur des articles pouvant l'intéresser car en rapport avec ses goûts qu'on a pu observer jusque

là, et même lui proposer des articles par lesquels il n'aurait jamais cru pouvoir être intéressé mais qu'il va finalement apprécier et se décider à acheter !

Conclusion

On sait que les applications des algorithmes génétiques sont multiples : optimisation de fonctions numériques difficiles , traitement d'image , optimisation d'emplois du temps, contrôle de systèmes industriels [Beasley, 1993a], cryptographie, apprentissage des réseaux de neurones [Renders, 1995], etc.

Nos exemples d'application nous ont permis de nous rendre compte que le codage des données pour modéliser un probléme est comlexe. D'autre part, nous nous sommes aussi aperçus des difficultés pour choisir pertinemment de bons paramètre pour les divers opérateurs (mutation , croisement , séléction , remplacement). Des choix par rapport aux opérateurs eux mêmes sont aussi à gérer, sachant que certains sont plus appropriés au problème et qu'il permettent d'optimiser.

Les algorithmes génétiques seuls ne sont pas très efficaces dans la résolution d'un problème. Ils apportent cependant assez rapidement une solution acceptable. Néanmoins, il est possible de l'améliorer assez efficacement en le combinant avec un algorithme déterministe.

Table des matières

Bibliographie :

Ouvrages qui nous ont permis d'approfondir notre recherche :

Etude de l'adaptivité de systèmes évolutionnsite en environnement à fitness dynamique - Alessio Gaspar

Algorithmes Génétiques et polymorphisme - Manuel Cleber

Voici des ouvrages pour permettre de pousser plus votre étude :

T. Bäck, Evolutionary Algorithms in Theory and Practice,
Oxford University Press, 1996

L. Davis, The Handbook of Genetic Algorithms, Van
Nostrand& Reinhold, 1991

D.B. Fogel, Evolutionary Computation, IEEE Press, 1995

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

J. Koza, Genetic Programming, MIT Press, 1992

Z. Michalewicz, Genetic Algorithms + Data Structures = Evolutionary Programs, Springer, 3ème Ed., 1996

H.P. Schwefel, Evolution and Optimum Seeking, Wiley &
Sons, 1995

Quelques références francophones :

Intelligence artificielle et informatique théorique. (J.M ALLIOT, SCHIEX).

AG et réseaux de neurones. (Jean-Michel RENDERS).

L'ordinateur génétique. Paris. Hermès, 1996. (Dessales.J-L).

Algorithmes génétiques, exploration, optimisation et apprentissage automatique. Paris 1994. (Goldberg D.E).