REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR

ET DE LA RECHERCHE SCIENTIFIQUE

MEMOIRE

Présenté à l'université de BATNA

Faculté des sciences de l'ingénieur - Département d'informatique

Pour l'obtention du diplôme

Magister en informatique

OPTION

INFORMATIQUE INDUSTRIELLE

Par

SAADI LEILA

Thème

Optimisation MultiObjectifs par Programmation Génétique

Soutenue le : 08/07 /2007 Devant le jury composé de

A. ZIDANI	Maître de conférences, **Université de Batna**	Président
M. BATOUCHE	Professeur, **Université de Constantine**	Rapporteur
S. CHIKHI	Maître de conférences, **Université de Constantine**	Examinateur
D. E. SAIDOUNI	Maître de conférences, **Université de Constantine**	Examinateur

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR

ET DE LA RECHERCHE SCIENTIFIQUE

MEMOIRE

Présenté à l'université de BATNA

La faculté des sciences de l'ingénieur - département d'informatique

Pour l'obtention du diplôme

Magister en informatique

OPTION
INFORMATIQUE INDUSTRIELLE

Par

SAADI LEILA

Thème

Optimisation MultiObjectifs par Programmation Génétique

Soutenue le : 08 /07 /2007 Devant le jury composé de

A. ZIDANI	Maître de conférences, **Université de Batna**	Président
M. BATOUCHE	Professeur, **Université de Constantine**	Rapporteur
S. CHIKHI	Maître de conférences, **Université de Constantine**	Examinateur
D. E. SAIDOUNI	Maître de conférences, **Université de Constantine**	Examinateur

Remerciements

Je remercie Dieu tout puissant clément et miséricordieux de m'avoir soigné et aidé.

Je tiens, avant tout, à exprimer ma profonde gratitude à monsieur ***Mohamed Batouche****, professeur à l'université de Constantine, qui a assumé la direction de ce travail. Qu'il veuille bien trouver ici l'expression de ma reconnaissance pour son dévouement, sa patience, sa disponibilité, ses conseils et son aide constant qu'il m'a apporté tout au long de ce travail.*

Je remercie les membres de jury qui ont accepté de juger ce travail et d'y apporter leur caution :

Monsieur ***Dr. A. Zidani****, maître de conférences à l'université de BATNA, qui me fait le grand honneur d'accepter la présidence du jury.*

Monsieur ***Dr. S. Chikhi****, maître de conférences à l'université de Constantine, pour l'honneur qu'il me fait en acceptant de participer à ce jury.*

Monsieur, ***Dr. D. E. Saidouni****, maître de conférences à l'université de Constantine, pour l'honneur qu'il me fait en acceptant également de participer à ce jury.*

J'adresse mes vifs remerciements à tous les enseignants qui, par leur enseignement, leur encouragement et leur aide, ont contribués à ma formation durant toutes mes études à l'université de Batna surtout mon exemplairement Madame ***Behaz Amel*** *pour ses conseils efficaces et ses encouragements pour faire le mieux.*

Dédicaces

Je remercie le Dieu pour m'avoir donné la force d'accomplir ce travail pour aller plus loin.

Je dédie ce travail à mes parents, ma mère pour ses encouragements et ses prières tout au long de mes études, mon père pour tous ce qu'il avait fait pour avoir ce résultat.

Je le dédie à mes frères et sœurs, et je les remercie pour leurs encouragements et leurs aides ainsi que toute ma grande famille.

Je le dédie à l'homme qui m'a choisi pour me donner son nom à jamais, mon mari Adel.

A mes collègues de travail dans l'école Elwafa, le directeur Ferdi Adel, les enseignants « Lotfi, Naima, Rahma, Sophi, Nora, Wafa, Samia, Soraya, Rima Souad, Lamine, Abdelaziz, Djamel, Si Mohamed, radia» ainsi que les secrétaires.

A tous mes amis sans citer les noms.

A mes collègues de la promotion 2003-2004 de post-graduation.

A tous ceux qui aiment Leila et ceux qui Leila aime.

Leila

Table des matières

Table des figures

Le Résumé :

Les problèmes d'optimisation combinatoire (COP : Combinatorial Optimisation Problems) apparaissent naturellement quand il s'agit de modéliser mathématiquement un problème scientifique ou d'ingénierie. A cause de l'aspect NP-Complet de ces problèmes, des heuristiques sont utilisées pour atteindre rapidement des solutions sous-optimales. Ces problèmes sont intrinsèquement parallèles avec beaucoup d'aptitudes pour une implémentation hardware. Cependant, les techniques heuristiques existantes sont largement considérées comme inadaptées pour résoudre des problèmes d'optimisation.

En outre, les problèmes du monde réel nécessitent souvent l'optimisation simultanée d'un nombre important d'objectifs concurrents. En général, il n y a pas de solution unique mais plutôt un ensemble de solutions. Ces dernières sont appelées les solutions optimales de Pareto. Considérons, par exemple, la conception d'un système complexe Hardware/Software. Une conception optimale est une architecture minimisant les coûts et la consommation tout en maximisant les performances du système. Ces différents objectifs sont bien naturellement conflictuels. Un outil permettant d'explorer l'espace de solutions optimales de Pareto serait donc d'une grande utilité pour aboutir à la conception optimale d'un système complexe. Les algorithmes évolutionnaires semblent être une voie très prometteuse.

L'objectif de ce magister est d'appliquer l'un des algorithmes évolutionnaires qui est la programmation génétique aux problèmes d'optimisation multiobjectifs.

Les mots Clé : Optimisation multiobjectifs, Frontières de Pareto, Algorithme évolutionnaire, Programmation génétique, Bloating, SPEA2, TSP.

Introduction Générale

Il y a une tendance claire dans l'industrie envers plus de produits complexes enjambant de nouveaux domaines d'ingénierie. Simultanément, il y a une pression rapide en développant des produits, dans les prix de compétitions, et envers une haute qualité. Pour rencontrer ces domaines, les sociétés de fabrication ont été forcées de localiser leurs efforts dans le processus de développement. Pour cette raison, une issue c'est d'assurer la capacité de processus de développement, qui a résultée des méthodes pour analyser et gérer le processus de conception. Une autre issue c'est de développer des outils et des techniques qui supportent la conception des produits complexes, qui ont abouti une richesse dans les outils d'ingénierie de calcul. Puisque les capacités de calcul des ordinateurs s'accroissent, la possibilité de simulation et de l'optimisation numérique est élargie. Néanmoins, les techniques d'analyse, les modèles de simulation et l'optimisation numérique peuvent être une grande valeur et peuvent permettre des améliorations vastes dans la conception. Dans notre recherche, on a trouvé que toute une voie dans l'ingénierie des ordinateurs est consacrée pour modéliser et résoudre les problèmes d'optimisation pour l'industrie.

Résoudre un problème d'optimisation consiste à trouver la ou les meilleures solutions vérifiant un ensemble de contraintes et d'objectifs définis par l'utilisateur. Pour déterminer si une solution est meilleure qu'une autre, il est nécessaire que le problème introduise un critère de comparaison. Ainsi, la meilleure solution, appelée aussi solution optimale, est la solution ayant obtenu la meilleure évaluation au regard du critère défini. Les problèmes d'optimisation sont utilisés pour modéliser de nombreux problèmes appliqués dans la conception des systèmes de l'industrie, le traitement d'images, les problèmes rencontrés dans les réseaux industriels, la conception d'emplois du temps, . . . etc. La majorité de ces problèmes sont qualifiés de difficiles, car leur résolution nécessite l'utilisation des algorithmes évolués, et il n'est en général pas possible de fournir dans tous les cas une solution optimale dans un temps raisonnable. Lorsqu'un seul critère est donné,

par exemple un critère de minimisation de coût, la solution optimale est clairement définie, c'est celle qui a le coût minimal. Mais dans de nombreuses situations, un seul critère peut être insuffisant. En effet, la plupart des applications traitées intègrent plusieurs critères simultanés, souvent contradictoires. Intégrer des critères contradictoires pose un problème réel. La solution idéale n'existe pas, et il faut donc trouver un compromis. En effet, en considérant deux critères contradictoires *a* et *b*, améliorer *a* détériore forcément *b* et inversement. Le concept de solution optimale devient alors plus difficile à définir. Dans ce cas, la solution optimale cherchée n'est plus un point unique, mais un ensemble de compromis. Résoudre un problème comprenant plusieurs critères, appelé communément problème multiobjectifs, consiste donc à calculer le meilleur ensemble de solutions de compromis appelé « les frontières de Pareto ». De plus, le temps de calcul nécessaire à leur résolution peut devenir si important que l'algorithme développé devient inutilisable en pratique. Il n'existe pas d'algorithme générique capable de résoudre toutes les instances de tous les problèmes efficacement. Un grand nombre de méthodes ont été développées pour tenter d'apporter une réponse satisfaisante à ces problèmes. Parmi celles-ci, nous distinguons deux grandes classes de méthodes: les méthodes mathématiques et les méthodes basées sur les heuristiques « Métaheuristiques ».

Parmi les métaheuristiques, on trouve la famille des algorithmes évolutionnaires qui ont connues un grand succès dans l'optimisation multiobjectifs. Ces méthodes qui utilisent l'évolution et ses étapes pour localiser l'ensemble de solutions répondant à tous les objectifs ou les contraintes d'un problème complexe, appelé « les frontières de Pareto ». Les types des algorithmes évolutionnaires sont : les algorithmes génétiques, les stratégies d'évolution, la programmation évolutionnaire et la programmation génétique dont on va utiliser dans notre recherche. Cette dernière permet d'évoluer les programmes machines pour avoir le meilleur programme résolvant un problème d'optimisation multiobjectifs. A base de ce principe, la programmation génétique a montrée ses succès dans le domaine de conception de circuits électroniques, surtout dans l'informatique quantique.

En outre, notre travail consiste à démontrer l'utilité de la programmation génétique dans le domaine de l'optimisation multiobjectifs comme un premier pas pour trouver de

nouveaux algorithmes dans la résolution des problèmes complexes et de nouveaux circuits à appliquer dans l'implémentation de hardware surtout dans la nouvelle tendance qui est l'informatique quantique.

Organisation du mémoire :

L'organisation de ce mémoire est la suivante : le premier chapitre dresse un état de l'art non exhaustif des domaines de l'optimisation multiobjectifs et les méthodes de résolution utilisées. Nous présentons l'optimisation multiobjectifs tout en introduisant des concepts fondamentaux tels que la dominance, la surface de compromis. Nous décrivant aussi les problèmes d'optimisation multiobjectifs les plus connus dans le domaine de l'industrie. Ainsi les principales approches non Pareto de résolution pour ces problèmes et quelques approches Pareto, appartenant à la classe des métaheuristiques, dédiées aux problèmes d'optimisation multiobjectifs.

Le deuxième chapitre dresse une classe des méthodes de l'optimisation multiobjectifs qui sont les algorithmes évolutionnaires, tout en commençant par un historique de la façon dont ces méthodes deviennent des méthodes d'optimisation multiobjectifs. Nous décrivons un algorithme évolutionnaire, ses concepts et ses opérateurs génétiques. Ensuite nous décrivons les trois types des algorithmes évolutionnaires : les algorithmes génétiques, les stratégies d'évolution et la programmation génétique.

Et puisque nous localisons notre travail sur la programmation génétique, alors le troisième chapitre est consacré pour ce type d'algorithme évolutionnaire, tout en décrivant les concepts et les paramètres d'un algorithme de programmation génétique. Nous concluons par le phénomène connu sous le nom de « Bloating » des arbres dans ces algorithmes.

Ensuite, dans le quatrième chapitre, nous décrivons notre travail intitulé par « L'optimisation Multiobjectif par Programmation Génétique », nous démontrons l'effet de la programmation génétique dans les problèmes d'optimisation, ainsi qu'on a introduit une manière pour réduire le problème rencontré dans cette application qui est l'accroissement

des arbres (c'est-à-dire les programmes résultats). Cette méthode conçut d'hybrider la programmation génétique et les techniques d'optimisation multiobjectifs pour : résoudre un problème complexe et réduire le phénomène de bloating en utilisant l'approche SPEA2 (Strength Pareto Evolutionary Algorithm 2). Nous concluons cette partie par des résultats obtenus lors de l'application de cette hybridation sur quelques problèmes complexes, par exemple : le problème de TSP, N-Parity, …

Finalement, nous clôturons ce mémoire par une conclusion générale, dont nous généralisons notre travail à tout problème complexe, soit dans l'industrie, le traitement d'image, les réseaux industriels, …

Chapitre I

L'Optimisation Multiobjectifs

I. Introduction :

Les ingénieurs se heurtent quotidiennement à des problèmes technologiques de complexité grandissante, qui surgissent dans des secteurs très divers, comme dans le traitement des images, la conception des systèmes mécaniques, la recherche opérationnelle, … . Le problème à résoudre peut fréquemment être exprimé sous la forme générale d'un problème d'optimisation, dans lequel on définit une fonction objective, ou fonction de coût (voire plusieurs), que l'on cherche à minimiser par rapport à tous les paramètres concernés. Par exemple, dans le célèbre problème du voyageur de commerce, on cherche à minimiser la longueur de la tournée d'un « voyageur de commerce », qui doit visiter un certain nombre de villes, avant de retourner à la ville de départ. La définition du problème d'optimisation est souvent complétée par la donnée de contraintes : tous les paramètres (ou variables de décision) de la solution proposée doivent respecter ces contraintes, faute de quoi la solution n'est pas réalisable.

Il existe de nombreuses méthodes d'optimisation « classiques » pour résoudre de tels problèmes, applicables lorsque certaines conditions mathématiques sont satisfaites, ainsi, la programmation linéaire traite efficacement le cas où la fonction objective, ainsi les contraintes, s'expriment linéairement en fonction des variables de décision. Malheureusement, les situations rencontrées en pratique comportent souvent une ou plusieurs complications, qui mettent en défaut ces méthodes : par exemple, la fonction objective peut être non linéaire, ou même ne peut pas s'exprimer analytiquement en fonction des paramètres ; ou encore, le problème peut exiger la considération simultanée de plusieurs objectifs contradictoires. [02]

L'arrivée d'une nouvelle classe de méthodes, nommées métaheuristiques, marque une réconciliation des deux domaines : en effet, celles-ci s'appliquent à toutes sortes de problèmes combinatoires, et elles peuvent également s'adapter aux problèmes continus. Ces méthodes, qui comprennent notamment la méthode du recuit simulé, les algorithmes génétiques, la méthode de recherche tabou, les algorithmes de colonies de fourmis, etc.

sont apparues, à partir des années 1980, avec une ambition commune : résoudre au mieux les problèmes d'optimisation difficiles.

Tous ces concepts au tour d'un problème d'optimisation sont décrits dans ce qui suit tout en renforçant nos idées par des exemples de problèmes d'optimisation multiobjectifs et des méthodes utilisées pour les mettre en œuvre.

II. Définitions :

II.1 Un problème :

Un problème dans le point de vue informatique (Computational Problem) veut dire comment faire relier un ensemble de **données** par un ensemble de **résultats**, où les données vont subir un ensemble d'opérations dont on les appelle **le traitement**. Donc il est conçu comme une relation $\prod \subseteq \ell \, x \, S$ entre les entrées ou **instances**, et les sorties ou **solutions**.

Pour qu'une instance d'un problème soit compréhensible par un ordinateur numérique, elle doit être décrite comme une séquence finie de symboles d'un ensemble fini arbitraire appelé **alphabet**. De même, la solution d'une instance d'un tel problème est émise dans ce format. Généralement, ℓ est infini alors que S peut être fini. [01]

II.2 Types des problèmes :

Les problèmes peuvent être classés selon les propriétés de l'ensemble des solutions :

- Un problème de décision : c'est un problème dont la réponse est tout simplement OUI ou NON.

- Un problème polynomial réductible : on a *L1* et *L2* deux problèmes de décision, on dit *L1* est polynomial réductible à *L2* s'il existe un algorithme polynomial qui convertit chaque instance d'entrée de *L1* à une autre instance de *L2*.

- Un problème de la classe P : un problème est dit polynomial s'il existe un algorithme de complexité polynomiale permettant de répondre à la question posée dans ce problème, quelque soit la donnée de celui-ci. La classe P est l'ensemble de tous les problèmes de reconnaissances polynomiaux.

- Un problème de la classe NP : un problème de reconnaissance est dans la classe NP si, pour toute instance de ce problème, on peut vérifier, en un temps polynomial par rapport à la taille de l'instance, qu'une solution proposée ou devinée permet d'affirmer que la présence est « oui » pour cette instance. [66]

- Un problème de la classe NP-hard : on dit qu'un problème est dans la classe NP-hard si chaque autre problème dans NP est réductible d'une manière polynomiale dans ce dernier.

- Un problème de la classe NP-complet : s'il appartient à NP et chaque autre problème dans NP est réductible d'une manière polynomiale dans ce dernier. C'est un problème de décision et NP-hard qui cherche à trouver l'optimum. [03]

III. Un problème combinatoire :

Un problème combinatoire est toute situation dont on cherche d'avoir une solution tout en respectant la présence d'un ensemble de contraintes. La solution c'est un résultat de faire combiner ces contraintes ensemble d'une manière qu'on maximise quelques uns et on minimise les autres, ces contraintes ont une caractéristique primordiale, c'est que chaque contrainte influe sur les autres soit quand on minimise sa valeur ou on la maximise, dans un autre terme on dit que les contraintes sont conflictuelles.

Par exemple, le schéma suivant présente une situation de problème combinatoire: où on veut acheter une voiture dans la mode et en même temps avec un prix raisonnable qui ne peut pas dépasser certaine limite. Si on maximise la première contrainte (une bonne voiture) on va avoir un prix maximale, dans le contraire on va aboutir à une mauvaise voiture mais avec un prix minimale dans les limites; on constate dans cet exemple que c'est difficile d'arranger ces deux contraintes dans nos besoins.

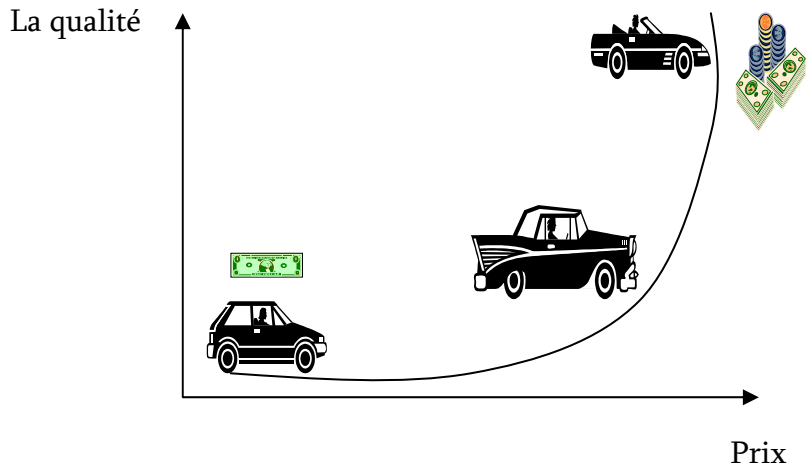


Figure 1 : une figure illustrant un problème combinatoire.

IV. Un problème d'optimisation :

IV.1 Définition :

Les problèmes d'optimisation ont été définis pour représenter les problèmes décrits dans le paragraphe précédent. Ces problèmes occupent actuellement une place de choix dans la communauté scientifique. Non pas qu'ils aient été un jour considérés comme secondaires mais l'évolution des techniques informatiques a permis de dynamiser les recherches dans ce domaine.

Le monde réel offre un ensemble très divers de problèmes d'optimisation :

- Problème combinatoire ou à variables continues.
- Problèmes à un ou plusieurs objectifs.
- Problèmes statistiques ou dynamiques.
- Problème dans l'incertain.

Cette liste n'est évidemment pas exhaustive, et un problème peut être à la fois multiobjectifs et dynamique. [04]

Un tel problème est caractérisé par :

➔ **Un problème d'optimisation** est défini par un **espace d'état**, une ou **plusieurs fonction(s) objective(s)** et **un ensemble de contraintes**.

➔ **L'espace d'état** est défini par l'ensemble de domaines de définition des variables du problème.

➔ **Les variables** du problème peuvent être de nature diverse (réelle, entière, booléenne, etc.) et sont exprimées de données qualitatives ou quantitatives.

➔ **Une fonction objective** représente le but à atteindre pour le décideur (minimisation de coût, de durée, d'erreur, ...). Elle définit un espace de solutions potentielles au problème.

➔ **L'ensemble de contraintes** définit des conditions sur l'espace d'état que les variables doivent satisfaire. Ces contraintes sont souvent des contraintes d'inégalité ou d'égalité et permettent en général de limiter l'espace de recherche.

➔ **Une méthode d'optimisation** cherche le point ou un ensemble de points de l'espace de état possible qui satisfait au mieux un ou plusieurs critères. Le résultat est appelé valeur optimale ou optimum. [04]

On peut dire qu'un problème d'optimisation se définit comme la recherche du minimum ou du maximum (de l'optimum donc) d'une fonction donnée. On peut aussi trouver des problèmes d'optimisation pour lesquels les variables de la fonction à optimiser sont des contraintes à évoluer dans une certaine partie de l'espace de recherche.

De cette définition de problème d'optimisation, il apparaît qu'un problème multiobjectifs ou multicritères peut être définit comme un problème dont on cherche l'action qui satisfait un ensemble de contraintes et optimise un vecteur de fonctions objectives. La difficulté principale d'un problème multiobjectifs est qu'il n'existe pas de définition de la solution optimale. Le décideur peut simplement exprimer le fait qu'une solution est préférable à une autre mais il n'existe pas une solution meilleure que toutes les autres.

Ce besoin d'optimisation vient de la nécessité de l'ingénieur de fournir à l'utilisateur un système qui répond au mieux au cahier des charges. Ce système devra être calibré de manière à:

➔ Occuper le volume nécessaire à son bon fonctionnement (coût des matières premières),

➔ Consommer le minimum d'énergie (coût de fonctionnement),

➔ Répondre à la demande de l'utilisateur (cahier des charges).

Dans ce sens, nous pouvons définir :

IV.1.1 Fonction objective :

C'est le nom donné à la fonction ***f*** (on l'appelle encore fonction de coût ou critère d'optimisation). C'est cette fonction que l'algorithme d'optimisation va devoir « optimiser » (trouver un optimum).

Mathématiquement parlant, un problème d'optimisation se présentera sous la forme suivante :

$$\begin{cases} \min\ f(\vec{x}) & (\textit{fonction à optimiser}) \\ \textit{avec} \quad \vec{g}(\vec{x}) \le 0 & (\textit{m contraintes d'énigalité}) \\ \textit{et} \quad \vec{h}(\vec{x}) = 0 & (\textit{p contraintes d'égalité}) \end{cases}$$

On a $\vec{x} \in \Re^n$, $\vec{g}(\vec{x}) \in \Re^m$ *et* $\vec{h}(\vec{x}) \in \Re^p$

Ici les vecteurs $\vec{g}(\vec{x})$ *et* $\vec{h}(\vec{x})$ représentent respectivement *m* contraintes d'inégalité et *p* contraintes d'égalité. [02]

IV.1.2 Variables de décision :

Elles sont regroupées dans le vecteur $\vec{x}$. C'est en faisant varier ce vecteur que l'on cherche un optimum de la fonction ***f***.

IV.1.3 Minimum global :

Un « point » $\vec{x}^*$ est un minimum global de la fonction ***f*** si on a :

$f(\vec{x}^*) < f(\vec{x})$ Quel que soit $\vec{x}$ tel que $\vec{x}^* \neq \vec{x}$.

IV.1.4 Minimum local fort :

Un « point » $\vec{x}^*$ est un minimum local fort de la fonction ***f*** si on a :

$f(\vec{x}^*) < f(\vec{x})$ quel que soit $\vec{x} \in V(\vec{x}^*)$ *et* $\vec{x}^* \neq \vec{x}$, où $V(\vec{x}^*)$ définit un « voisinage » de $\vec{x}^*$.

IV.1.5 Minimum local faible :

Un « point » $\vec{x}^*$ est un minimum local faible de la fonction ***f*** si on a :

$f(\vec{x}^*) \leq f(\vec{x})$ quel que soit $\vec{x} \in V(\vec{x}^*)$ *et* $\vec{x}^* \neq \vec{x}$, où $V(\vec{x}^*)$ définit un « voisinage » de $\vec{x}^*$. [02].

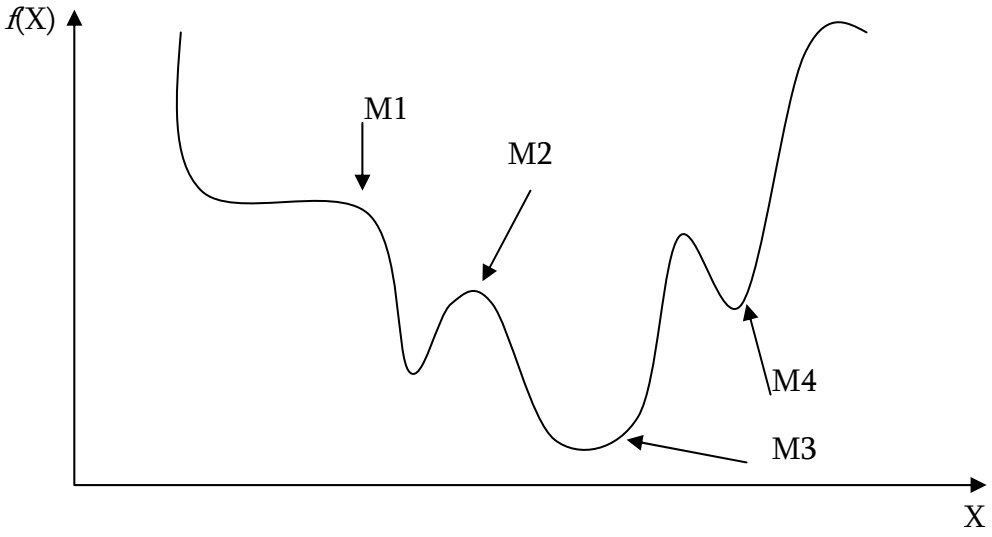


Figure 2 : Les différents Minima [02].

En résumant ces notions mathématiques représentant un problème d'optimisation multiobjectifs par la figure suivante :

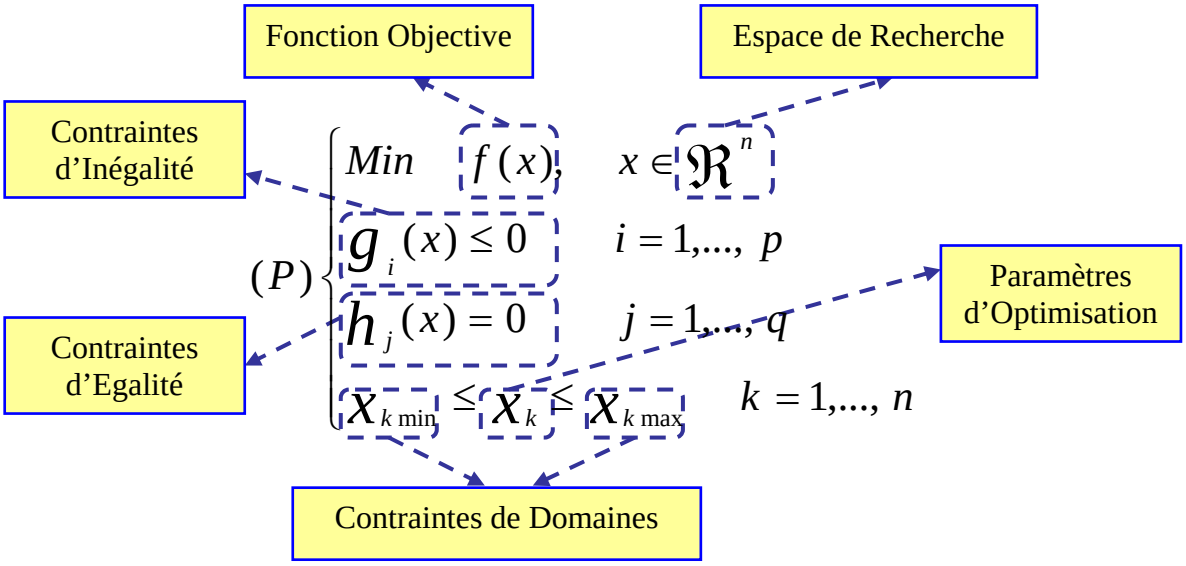


Figure 3 : La présentation mathématique d'un problème d'optimisation multiobjectifs. [05]

IV.2 Les variables d'un problème multiobjectifs :

Les variables du problème sont les paramètres dont le concepteur doit ajustés pour modifier le système à concevoir. Il y a plusieurs types de ces variables :

IV.2.1 Les variables indépendantes :

Sont les quantités actuelles dont le concepteur s'occupe directement, comme la géométrie, les propriétés du matériel, le volume de production, la surface finale, la configuration des composants, les propriétés de lubrification et autres …. Les variables indépendantes sont usuellement connues comme les **variables de problème** ou les **paramètres de problème**.

IV.2.2 Les variables dépendantes :

Sont les variables dont le concepteur ne peut attribuer des valeurs mais il les utilise pendant les paramètres du problème. Les variables dépendantes sont usuellement nommées les **caractéristiques** ou les **attributs du problème**. Exemple de caractéristiques du problème l'énergie de consomption, le prix et l'erreur de contrôle. La valeur du problème est largement une fonction des caractéristiques du problème. Dans l'optimisation, la valeur de la fonction objective correspond à la valeur du caractéristique particulière. Une fonction objective c'est ainsi la relation entre les paramètres du problème et les valeurs des caractéristiques particulières. Pour un problème d'optimisation général, il faut être très difficile ou bien impossible de représenter analytiquement tous comme des relations. Généralement, les caractéristiques doivent être le résultat de la simulation complexe, ou ils doivent inclure en quantifiable le jugement de l'homme.

IV.2.3 Les variables d'état :

Sont le type intermédiaire des variables de problème entre les variables dépendantes et les variables indépendantes, comme la pressure de cylindre hydraulique ou le courant dans un moteur électrique. On ne peut pas attribué directement des valeurs aux variables d'état, aussi ces variables ne peuvent pas directement contribuer dans la valeur de problème comme les caractéristiques.

IV.2.4 Les variables opérationnelles :

Sont les variables qui peuvent être changées par l'opérateur après la conception actuellement construise.

IV.2.5 Les variables de l'environnement :

Ou les variables externes sont des facteurs écologiques qui affectent le problème au cours de l'utilisation, comme le changement de charge, la température extrême et celle d'usage. Le concepteur a pour déterminer les conditions de travail du problème les variables de l'environnement et les variables opérationnelles dans ses expériences.

Le problème peut être formulé quant à l'attribution des valeurs aux paramètres du problème pour être sûr que les variables d'état et les caractéristiques sont les meilleures au cours du champ large des variables opérationnelles et de l'environnement. Ça nécessite une optimisation de problème multiobjectifs complexe. [06]

IV.3 La classification des problèmes multiobjectifs :

On peut classer les différents problèmes d'optimisation que l'on rencontre dans la vie courante en fonction de leurs caractéristiques :

1- Nombre de variables de décision :

- ✓ une ⇒ mono variable.
- ✓ Plusieurs ⇒ multi variable.

2- Type de variable de décision :

- ✓ Nombre réel continu ⇒ continu.
- ✓ Nombre entier ⇒ entier ou discret.
- ✓ Permutation sur un ensemble fini de nombres ⇒ combinatoire.

3- Type de fonction objective :

- ✓ Fonction linéaire des variables de décision ⇒ linéaire.
- ✓ Fonction quadratique des variables de décision ⇒ quadratique.
- ✓ Fonction non linéaire des variables de décision ⇒ non linéaire.

4- Formulation de problème :

- ✓ Avec contraintes ⇒ contraint.
- ✓ Sans contraintes ⇒ non contraint. [02]

V. *La dominance :*

La plupart des problèmes d'optimisation réels sont décrits à l'aide de plusieurs objectifs ou critères souvent contradictoires devant être optimisés simultanément. Alors que, pour les problèmes n'incluant qu'un seul objectif, l'optimum cherché est clairement défini, celui-ci reste à formaliser pour les problèmes d'optimisation multiobjectifs. En effet, pour un problème à deux objectifs contradictoires, la solution optimale cherchée est un ensemble de points correspondant aux meilleurs compromis possibles pour résoudre notre problème. [07]

Lorsque l'on cherche à obtenir une solution optimale à un problème d'optimisation multiobjectifs donné, on s'attend souvent à trouver une solution et une seule. En effet, on rencontre rarement ce cas de figure. La plupart de temps, on trouve une multitude de solutions, du fait que certaines des objectifs sont contradictoires.

On effet, si l'on prend d'exemple de dimensionnement d'une poutre devant supporter une charge donnée, on va vouloir obtenir une poutre de section la plus petite possible, produisant la plus petite déformation possible, lorsque la charge repose sur le milieu de la poutre. Dans cet exemple (représenté à la figure 4) de manière intuitive, on s'aperçoit que répondre à l'objectif « poutre de petite section » ne va pas du tout dans le sens de répondre à l'objectif « petite déformation ».

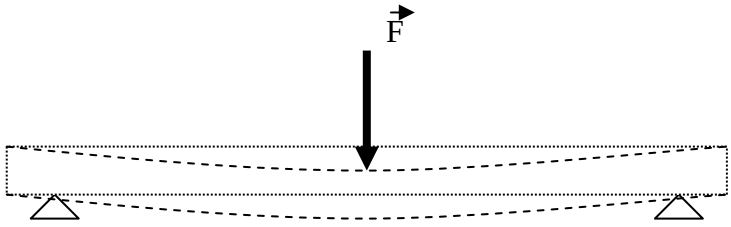


Figure 4 : Déformation d'une poutre subissant une contrainte [02]

Donc, quand on résoudra un problème d'optimisation multiobjectifs, on obtiendra une grande quantité de solutions. Ces solutions comme on peut s'en douter, ne seront pas optimales, au sens où elles ne minimiseront pas tous les objectifs du problème.

Un concept intéressant, qui nous permettra de définir les solutions obtenues, est le **compromis**. En effet, les solutions que l'on obtient lorsqu'on a résolu le problème sont des

solutions de compromis. Elles minimisent un certain nombre d'objectifs tout en dégradant les performances sur d'autres objectifs.

V.1 La dominance au sens de Pareto :

Comme la solution optimale est une multitude de points de $\mathbf{R}^n$, il est vital pour identifier ces meilleurs compromis de définir une relation d'ordre entre ces éléments. Dans le cas des problèmes d'optimisation multiobjectifs, ces relations d'ordre sont appelées relations de dominance. Plusieurs relations de dominance ont déjà été présentées: la A-dominance [12], la dominance au sens de Geoffrion [13], la cône-dominance [02], . . .

Mais la plus célèbre et la plus utilisée c'est la dominance au sens de Pareto. Au XIX^{ème} siècle, Vilfredo Pareto, un mathématicien italien, formule le concept : « dans un problème multiobjectif, il existe un équilibre tel que l'on ne peut pas améliorer un critère sans détériorer au moins un des autres critères ».

De manière à définir clairement et formellement cette notion, les relations =, ≤ et < usuelles sont étendues aux vecteurs. [07]

Soient u et v, deux vecteurs de même dimension,

$$\begin{aligned} u = v \quad & ssi \quad \forall i \in \{1,2,...,m\},\ u_i = v_i \\ u \leq v \quad & ssi \quad \forall i \in \{1,2,...,m\},\ u_i \leq v_i \\ u < v \quad & ssi \quad u \leq v \wedge u \neq v \end{aligned}$$

Les relations ≥ et > sont définies de manière analogue.

Les relations définies précédemment ne couvrent pas tous les cas possibles. En effet, il est impossible de classer les points a = (1, 2) et b = (2, 1) à l'aide d'une de ces relations.

Contrairement aux problèmes à un seul objectif où les relations usuelles <, . . . suffisent pour comparer les points, elles sont insuffisantes pour comparer des points issus de problèmes multiobjectifs. Nous définissons donc maintenant la relation de dominance au sens de Pareto permettant de prendre en compte tous les cas de figures rencontrés lors de la comparaison de deux points (ici des vecteurs).

Lorsque nous avons résolu un problème d'optimisation multiobjectifs, nous allons obtenu une multitude de solutions. Seul un nombre restreint de ces solutions va nous intéresser. Pour qu'une solution soit intéressante, il faut qu'il existe une relation de dominance entre la solution considérée et les autres solutions, dans le sens suivant :

On dit que le vecteur $\vec{x}_1$ domine le vecteur $\vec{x}_2$ si :

- $\vec{x}_1$ est au moins aussi bon que $\vec{x}_2$ dans tous les objectifs, et,
- $\vec{x}_1$ est strictement meilleur que $\vec{x}_2$ dans au moins un objectif.

$x \succ y$ *si*

$$\forall i \in \{1,2,...k\}: f_i(x) \leq f_i(y) \text{ and } \exists j \in \{1,2,...k\}: f_j(x) < f_j(y)$$

Les solutions qui dominent les autres mais ne se dominent pas entre elles sont appelées **solutions optimales au sens de Pareto** (ou solutions non dominées). Dans cette ensemble de solutions il existe deux définitions d'optimalité : l'optimalité locale et l'optimalité globale au sens de Pareto.

Optimalité locale au sens de Pareto

Un vecteur $\vec{x} \in \Re^n$ est optimal localement au sens de Pareto s'il existe un réel $\delta > 0$ tel qu'il n'y ait pas de vecteur $\vec{x}'$ qui domine le vecteur $\vec{x}$ avec $\vec{x}' \in \Re^n \cap \mathrm{B}(\vec{x}, \delta), où\ \mathrm{B}(\vec{x}, \delta)$ représente une boule de centre $\vec{x}$ et de rayon δ.

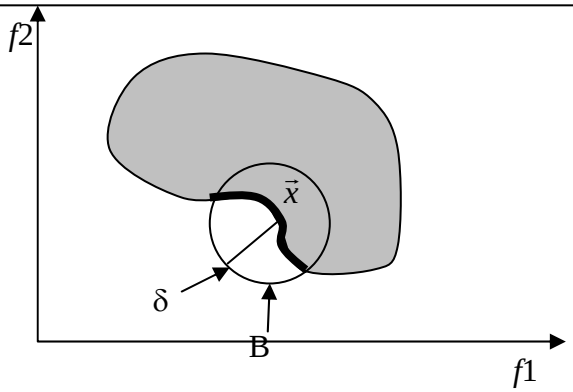


Figure 5 : L'optimalité locale au sens de Pareto

Optimalité globale au sens de Pareto

Un vecteur $\vec{x}$ est optimal globalement au sens de Pareto (ou optimal au sens de Pareto) s'il n'existe pas de vecteur $\vec{x}'$ tel que $\vec{x}'$ domine le vecteur $\vec{x}$.

La différence entre cette définition et celle de l'optimalité locale tient dans le fait que l'on ne considère plus une restriction de l'ensemble $\mathbf{R^n}$.

On dit aussi qu'un vecteur $\vec{x}$ est optimal au sens de Pareto pour un problème d'optimisation multiobjectifs donné si :

$(C^- + \vec{x}) \cap F = \{\vec{x}\}$ ⇨ Le théorème de contact (figure 6).

Où ***F*** désigne l'espace de solutions réalisables.

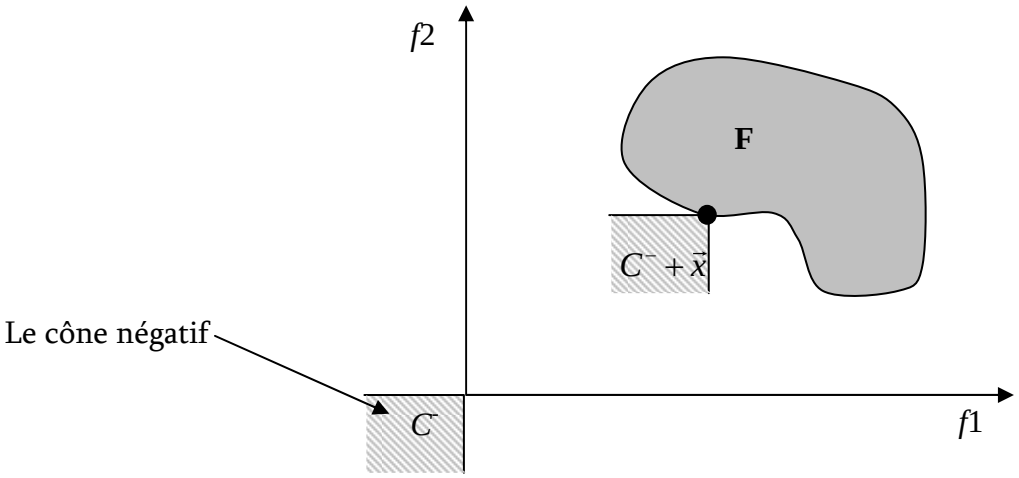


Figure 6 : Le théorème de contact. [02]

Le cône négatif

Un cône négatif est définit dans $\boldsymbol{R^k}$ de la manière suivante :

$C^- = \{\vec{x} \mid \vec{f}(\vec{x}) \in \Re^k \ \ et \ \ \vec{f}(\vec{x}) \leq 0\}$ (Voir la figure 6)

Lorsqu'on applique la définition de la dominance, on peut définir quatre régions auxquelles on peut attribuer des niveaux de préférence. Ces régions sont présentées à la figure 7. Cette figure reprend le découpage défini par le cône négatif que l'on a introduit précédemment et l'étend à tout l'espace.

Par exemple, si ce graphique est centré sur une solution A et que l'on compare cette solution avec une solution B, on aura les possibilités suivantes :

- Si la solution B se trouve dans le quadrant 1, alors la solution A est préférée à la solution B.
- Si la solution B se trouve dans le quadrant 3, alors la solution A est dominée par la solution B.

- Si la solution B se trouve dans l'un des quadrants 2 ou 4, alors on ne peut pas prononcer sur la préférence de A par rapport à B ou B par rapport à A.

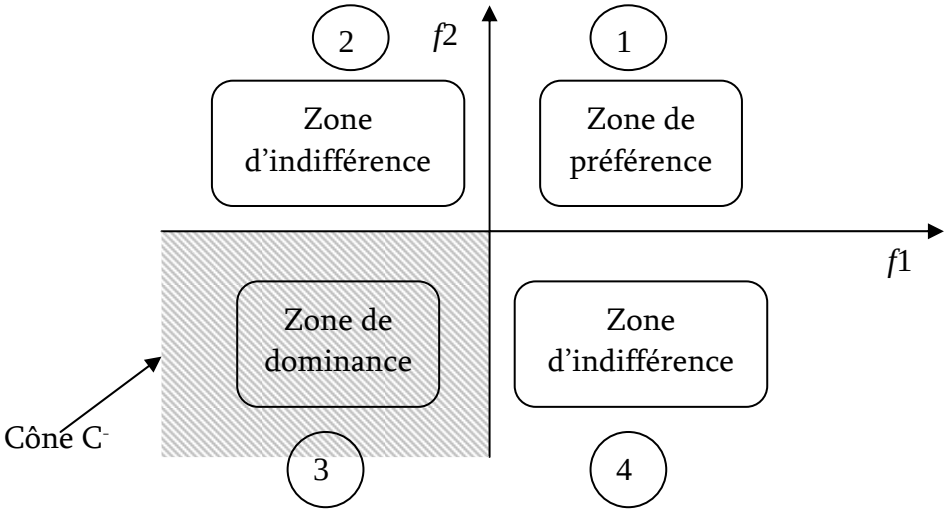


Figure 7 : Les niveaux de préférence dans la relation de dominance. [02]

V.2 La surface de compromis :

Les solutions obtenues lors du classement basé sur le principe de dominance de Pareto forme ce que l'on appelle **la surface de compromis** (ou Pareto Front).

Imaginons que nous avons un problème à deux objectifs (minimiser *f*1 et minimiser *f*2 sous les contraintes $\vec{g}(\vec{x}) \le 0$ et $\vec{h}(\vec{x}) = 0$) :

- On appelle S l'ensemble de valeurs du couple ($f1(\vec{x})$, $f2(\vec{x})$) quand à $\vec{x}$ respecte les contraintes $\vec{g}(\vec{x})$ et $\vec{h}(\vec{x})$.
- On appelle P la surface de compromis.

  On représente S et P sur la figure suivante :

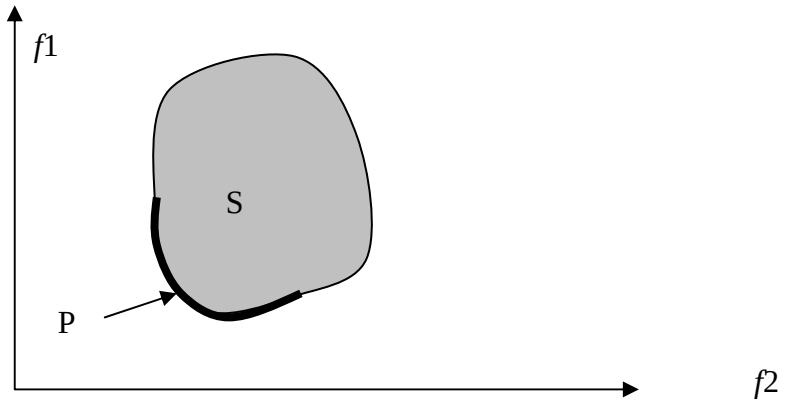


Figure 8 : Représentation de la surface de compromis

Une propriété est remarquable : en fonction du type du problème que l'on cherche à résoudre, on obtient une forme de surface de compromis. Les formes les plus courantes de surfaces de compromis sont réunies à la figure 9. Ces formes de surface de compromis sont typiques pour un problème d'optimisation multiobjectifs sur un ensemble de solutions convexes. C'est se type d'ensemble que l'on rencontre la plupart de temps.

On observe deux points caractéristiques associés à une surface de compromis :

Point idéal :
Les coordonnées de ce point sont obtenues en optimisant chaque fonction objective séparément On dit aussi que les coordonnées du point idéal correspondent aux meilleures valeurs de chaque objectif des points de la frontière Pareto.

Point « nadir » :
Les coordonnées de ce point correspondent aux pires valeurs obtenues par chaque fonction objective lorsque l'on restreint l'espace des solutions à la surface de compromis.

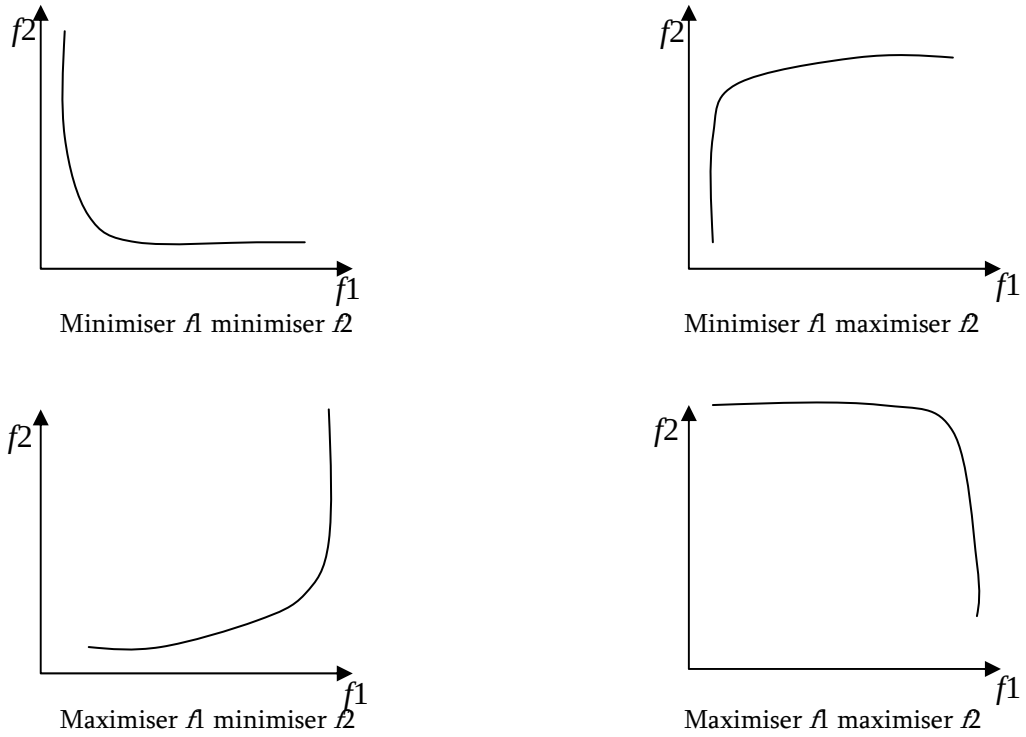


Figure 9 : Formes les plus courantes de la surface de compromis dans le cas de deux objectifs.

Le point idéal est utilisé dans beaucoup de méthodes d'optimisation comme point de référence. Le point nadir, lui, sert à restreindre l'espace de recherche ; il est utilisé dans certaines méthodes d'optimisation interactives. [02]

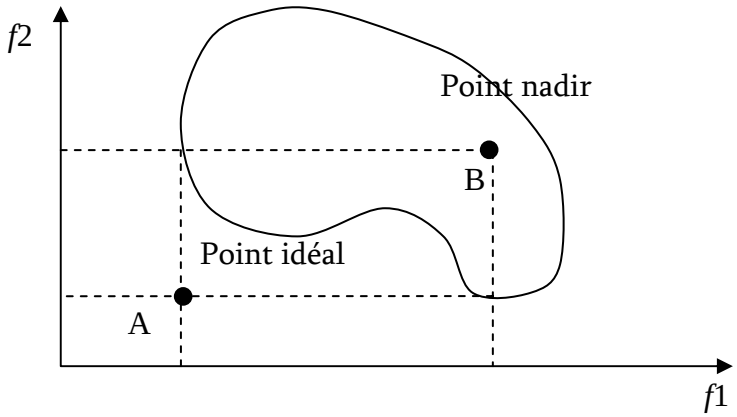
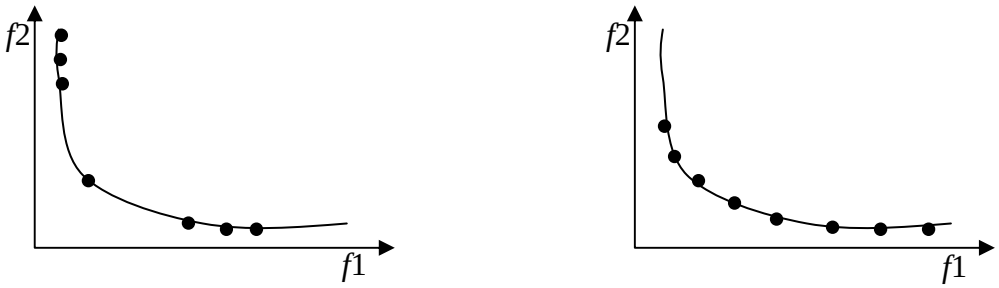


Figure 10 : représentation du point idéal et de point « nadir ». [02]

V.2.1 Représentation de la surface de compromis :

Toutes les représentations de la surface de compromis, pour un même problème, ne sont pas équivalentes. En effet, la représentation idéale de la surface de compromis devra être constituée de points solution de notre problème répartis de manière uniforme sur la surface de compromis (figure 11 : la représentation de la surface de compromis).



Une mauvaise représentation de la surface de compromis.

Une bonne représentation de la surface de compromis.

Figure 11 : La représentation de la surface de compromis. [02]

Dans le premier cas, les points, représentant la surface de compromis ne sont pas répartis de manière uniforme. L'utilisateur n'aura alors pas en sa possession un ensemble de solutions très utiles. En effet, s'il décide que la solution qu'il avait choisie ne lui convient pas, le choix d'une autre solution risque de se faire varier brusquement tous ses objectifs, et cette nouvelle solution ne lui conviendra pas non plus. Il est alors probable

que la solution offrant le « meilleur » compromis se trouve dans une zone qui ne soit pas représentée par des points solution.

VI. Exemples de problèmes combinatoires :

VI.1 Le voyageur de commerce :

Parmi les problèmes les plus célèbres et connus dans le domaine d'optimisation multiobjectifs le problème de **Voyageur de Commerce.** C'est un problème d'affectation assignant une route pour un véhicule entre quelques nœuds, dont l'algorithme cherche un ensemble de solutions qui répondent à quelques critères.

Le Problème du Voyageur de Commerce (**Traveling Salesman Problem**, **TSP**) possède une formulation élémentaire: étant données n villes, quel chemin doit-on emprunter pour les parcourir tout en minimisant la longueur totale du trajet? Il doit rendre régulièrement visite aux clients de son département. Sa clientèle étant répartie dans N localités différentes de sa région, y compris son domicile, il constate un jour que ses frais de route sont relativement élevés et surtout qu'ils diffèrent considérablement d'une tournée à l'autre. Il finit par s'adresser à un informaticien pour qu'il lui rédige un programme permettant de minimiser ses frais de route. Une première version de l'algorithme pourrait donc être :

(1) Minimiser les frais de route.

Cette première formulation trop vague ne permet guère de reconnaître le vrai problème à résoudre. Ainsi par exemple les données d'entrée (représentées ici par les causes des frais) ne sont pas mentionnées du tout. Une deuxième approche consiste donc à essayer de dégager des informations supplémentaires sur le champ de données en entrée. Finalement, cette deuxième étape fait savoir à l'informaticien que la totalité des frais de route se compose :

- Des frais d'essence pour le trajet en voiture et,
- Des frais de logement pour la nuit.

Alors, il a constaté que l’action « Minimiser les frais de route » c’est « minimiser les frais d’essence et les frais de logement ». Si dans une troisième approche l'informaticien demande la raison des différences de frais, il apprend que l'ordre dans lequel les différentes localités sont visitées joue un rôle non négligeable, puis, la question est de savoir « s’il y a des clients qu’il a visités plus souvent que d’autres ? ». Cette dernière approche consiste donc à essayer de simplifier le problème initial pour lui trouver une solution acceptable, à défaut d'être optimale.

C'est le représentant de commerce qui spécifie enfin qu'il rend visite exactement une fois par mois à chaque client de sa région, qu'une tournée de visites a une durée de cinq jours, et qu'il passe les nuits dans des hôtels à même prix. Ainsi, les frais de logement pour la nuit étant fixes, l'informaticien peut en conclure que l'on peut les négliger lors de la résolution du problème de minimisation. L'informaticien finit par formuler le problème du voyageur de commerce de la manière suivante :

«Une fois par mois, à partir de son domicile, un représentant de commerce rend visite à sa clientèle qui est répartie sur N localités différentes (y compris son propre domicile) de sa région. Lors d'un même tour, il ne passe par chaque localité qu'une seule fois. C'est ainsi que son trajet se trouve divisé en exactement N parties différentes et que la totalité des frais de route s'obtient par addition des frais de route individuels, c.-à-d. des différentes parties. Chaque partie de route occasionnant des frais individuels différents, la totalité des frais varie selon l'ordre dans lequel le représentant fait sa tournée. Le problème consiste à déterminer le tour dont la somme des frais de route individuels est minimale. En réalité, la consommation d'essence pour une partie de route n'est évidemment pas proportionnelle à la distance parcourue, mais elle dépend des conditions particulières des routes (autoroute, route nationale, route droite - route sinueuse, route plane - route raide) et notamment du sens dans lequel la route est parcourue.»

Par cette formulation de problème on déduit qu'il appartient à la classe des problèmes dits NP, ce qui signifie qu'il n'existe pas d'algorithme fournissant le chemin optimal en temps polynomial. Ce paradoxe en fait un problème stimulant et fécond qui donne lieu à des stratégies heuristiques originales et à des applications au domaine en plein essor de la bioinformatique. [08]

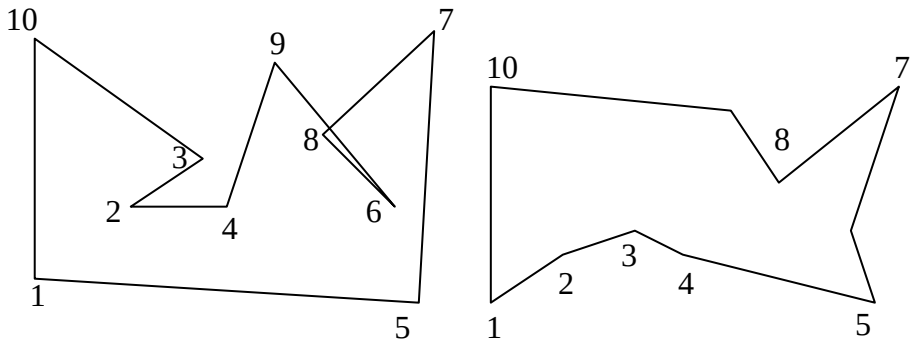


Figure 12 : Le tour d'un voyageur de commerce dans 10 villes [52]

VI.1.1 L'historique :

La résolution du problème PVC a passé par plusieurs étapes durant le siècle passé, en résumant ce chemin par:

19ème siècle

Les premières approches mathématiques exposées pour le problème du voyageur de commerce ont été traitées au 19ème siècle par les mathématiciens Sir William Rowan Hamilton et Thomas Penyngton Kirkman. Hamilton en a fait un jeu : Hamilton's Icosian game : les joueurs devaient réaliser une tournée passant par 20 points en utilisant uniquement les connections prédéfinies.

années 1930

Le PVC est traité plus en profondeur par Karl Menger à Harvard. Il est ensuite développé à Princeton par les mathématiciens Hassler Whitney et Merril Flood. Une attention particulière est portée sur les connections par Menger et Whitney ainsi que sur la croissance du PVC.

L'année 1954

Solution du PVC pour 49 villes par Dantzig, Fulkerson et Johnson par la méthode du cutting-plane.

L'année 1975

Solution pour 100 villes par Camerini, Fratta and Maffioli

L'année 1987

Solution pour 532, puis 2392 villes par Padberg et Rinaldi

L'année 1998

Solution pour les 13 509 villes des Etats-Unis.

L'année 2001

Solution pour les 15 112 villes d'Allemagne par Applegate, Bixby, Chvàtal et Cook des universités de Rice et Princeton. [11]

VI.1.2 La complexité :

Ce problème est un représentant de la classe des problèmes NP-complet. L'existence d'un algorithme de complexité polynomiale reste inconnue. Un calcul rapide de la complexité montre qu'elle est en $O(N!)$ où N est le nombre de villes.

La conception d'un algorithme correspondant qui donne une solution optimale n'est pas du tout triviale si l'on songe qu'il y a $(N - 1)! = (N 1)*(N - 2)*(N – 3) \ldots 3*2*1$ tours possibles pour N localités ce qui donne pour $N = 20$ exactement 1 2165,1017 tournées possibles, d'autant plus qu'un tel algorithme pourrait se révéler très coûteux en temps d'exécution. [10]

En supposant que le temps pour évaluer un trajet est de 1 μs, le tableau montre l'explosion combinatoire du PCV.

Nombre de villes	Nombre de possibilités	Temps de calcul
5	12	12 µs
10	181440	0,18 ms
15	43 milliards	12 heures
20	60 E+15	1928 ans
25	310 E+21	9,8 milliards d'années

Tableau 1 : Nombre de possibilités de chemins et temps de calcul en fonction du nombre de villes (on suppose qu'il faut 1 µs pour évaluer une possibilité) [11]

Ce qui donne plus de complexité de problème PVC c'est que le parcours des nœuds se fait de la manière que: les nœuds parcourus ne se re-parcourent pas, il faut retourner au nœud de départ et il faut minimiser le chemin parcouru à la plus petite valeur possible toute en se basant sur les distances entre les nœuds comme le montre la figure 13.

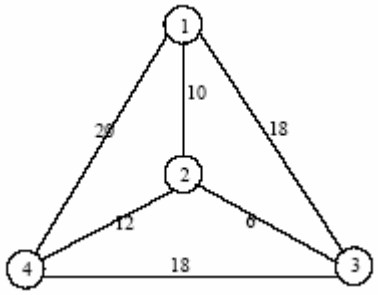


Figure 13: Un graphe pour le problème de TSP [36]

VI.1.3 Intérêt :

Le PVC fournit un exemple d'étude d'un problème NP-complet dont les méthodes de résolution peuvent s'appliquer à d'autres problèmes de mathématiques discrète. Néanmoins, il a aussi des applications directes, notamment dans les transports et la logistique. Par exemple, trouver le chemin le plus court pour les bus de ramassage scolaire ou, dans l'industrie, pour trouver la plus courte distance que devra parcourir le bras mécanique d'une machine pour percer les trous d'un circuit imprimé (les trous représentent les villes) [11]. Ainsi que le chemin le plus cours pour le routage des paquets de messages dans un réseau informatique.

VI.2 Le sac à dos (knapsack Problem) :

Un randonneur doit décider de l'équipement qu'il veut emporter pour sa prochaine expédition. Pour des raisons évidentes, il veut limiter le poids total des objets emportés et s'est fixé une borne supérieure de b kg à ne pas dépasser.

Chacun des n objets qu'il peut emporter possède un poids a_i, $i=1,\ldots,n$ ainsi qu'une utilité c_i, $i=1,\ldots,n$, cette dernière étant d'autant plus grande que le randonneur juge l'objet intéressant.

Définissant pour chaque objet une variable de décision binaire

- $X_i = 1$ si l'objet i est retenu
- $X_i = 0$ sinon

Déterminer une sélection optimale revient à résoudre le problème d'optimisation suivant (programme linéaire en nombre entiers) : [14]

$$Max\, Z = \sum_{i=1}^{n} c_i x_i$$

$$\sum_{i=1}^{n} a_i x_i \leq b$$

VI.2.1 Le problème de sac à dos multiobjectifs multidimensionnel :

Soit un ensemble d'objets, à chacun étant associé un vecteur de gains et de poids. Le problème de sac à dos multiobjectifs multidimensionnel (**MOKP**) consiste à sélectionner un sous-ensemble d'objets maximisant une fonction multiobjectives tout en satisfaisant un ensemble de contraintes. Plus formellement, le **MOKP** peut être défini comme suit :

$$\begin{cases} \max & z^j(x) = \sum_{i=1}^{n} c_i^j x_i & j = 1,\ldots,o \\ s.t. & \sum_{i=1}^{n} w_i^l \leq b_l & l = 1,\ldots,m \\ & x_i \in \{0,1\} & i = 1,\ldots,n \end{cases}$$

Où n est le nombre d'objets, x_i une variable de décision, o le nombre d'objectifs, z_j la j^{ème} composante de la fonction multiobjectifs z, et m le nombre de contraintes du problème.

Comme le problème NP-difficile de sac à dos multidimensionnel (**MKP**), le **MOKP** peut être utilisé pour modéliser de nombreux problèmes réels comme la répartition de budgets et l'allocation de ressources. Plusieurs algorithmes heuristiques ont été développés pour résoudre le MOKP. [11]

Le problème de sac à dos a été étudié pour la première fois à la fin des années 50. Dès cette date un grand nombre d'algorithmes et de techniques ont été proposés. Un peu d'algorithmes exacts pour le simple problème objectif sac à dos ont été proposés dans la littérature. Le plus célèbre algorithme a été proposé par Martello et Toth [15] et c'est une procédure de **Branch and Bound**.

Le simple problème objectif de sac à dos peut être aussi résous d'une manière heuristique pour s'approcher de la solution optimale. Une heuristique typique pour ce problème est l'algorithme de **Glouton** qui peut résoudre soit des problèmes simples ou complexes dans un temps polynomiale.

VI.3 Problème SAT (Problème de satisfaction) :

Dans 1996, Yokoo [19] propose le problème SAT qui modélise les variables en deux valeurs logiques et utilise une technique d'avant vérification pour trouver l'affectation de valeur satisfaisable. Après, les auteurs divisent un algorithme de **backtracking** et expérimentent les heuristiques d'ordre de variables [20]. [21] furent les premiers à proposer un accélérateur reconfigurable qui implémente le **backtracking** avec la propagation de variables logiques [22] comme une stratégie de déduction basique. Les auteurs en plus suggérèrent deux extensions pour leur architecture avec des techniques d'analyse de conflits qui allouent pour **backtracking** non chronologique et l'addition de clause dynamique [23]. Platzner et De Micheli [24] présentèrent plusieurs architectures de SAT basées sur le **backtracking** avec 3 valeurs logiques.

Ils utilisèrent les techniques de déduction, parmi les qui ne s'intéressent pas aux variables et aux implications. Une compilation d'environnement coutume pour ces accélérateurs SAT est discutée par Mencer [25]. Abramovici et De Sousa [26] et Abramovici et Saab [27] présentèrent une architecture basée sur l'algorithme de **PODEM** utilisé pour la génération automatique de test de modèles. Leur architecture utilise le **backtracing** plutôt que le **backtracking** et propage le résultat requis pour le formula booléenne face au variables. [28]

Le problème de satisfiabilité propositionnelle (SAT) c'est de déterminé si une expression booléenne aurait un étiquetage satisfaisant (ensemble d'affectations vraies). Les problèmes sont usuellement exprimés dans la forme normale conjonctive : une conjonction pour les clauses $C_1 \Lambda \ldots \Lambda C_m$ où chaque clause C est une disjonction de littérales $l1 V \ldots Vln$ est chacun a une variable booléenne X ou sa négation X'. Une variable booléenne peut être étiqueté chacune vrai ou faux. Une affectation satisfaisante a au moins une littérale vraie dans chaque clause. [30]

La plupart de problèmes combinatoires ont été avec succès modélisés et résolus comme SAT, mais pas tous les problèmes ne succombent facilement aux méthodes de SAT. Une raison c'est que les modèles de SAT peuvent s'avérer pour être très larges. Quelques chercheurs ont proposé plus de langages expressifs, partiellement, pour réduire les tailles de modèles. Des exemples incluent les équivalences nichées [31], le ou exclusive [32], les disjonctions et les conjonctions de littérales [33], les contraintes de cardinalité [34], …. En particulier, quelques modèles de SAT peuvent être réduis exponentiellement dans la taille par les reformuler aux modèles **Pseudo Booléen** (**PB**) [35]. [30]

Le meilleur problème d'expertise c'est le problème de satisfiabilité booléenne (SAT). Donnant :

- Un ensemble de n variables booléenne $x_1, x_2, \ldots, x_n$,

- Un ensemble de littérales, constituées des variables x_i et leurs compléments x'_i, et

- Un ensemble de m propositions $C_1, C_2, \ldots, C_n$, constituées par les littérales combinées par le ou logique ou l'opérateur +.

SAT est en quête d'une affectation de valeurs vraies à des variables qui réalise la conjonction normale à partir de **CNF** $C_1 * C_2 * \ldots * C_m$ vraie, ou * dénote l'opérateur logique.

VI.4 Problème de coloration de graphe (graph coloring) :

C'est un problème parmi les problèmes NP-Complet grâce à la difficulté trouvée lors de sa résolution, il consiste d'affecter un nombre défini de couleurs k aux sommets d'un graphe non orienté d'une manière que les couleurs des nœuds adjacents sont différentes. La coloration minimale utilise le petit nombre possible de couleur (**couleur chromatique**). La version décisive de coloration de graphe (**k-coloring**) demande quels sommets dans le graphe peut être colorés en utilisant un nombre $\leq k$ couleurs pour un k connu. [17]

L'inapproximation du problème de coloration de graphe suggère qu'il peut être plus difficile pour le résoudre. En utilisant ce problème quelques applications sont résolues dans un temps raisonnable comme :

Les problèmes de « **Time Tabling And Scheduling** » fréquemment rejettent certaines tâches en parallèle à cause les dépendances entre computations. **Planification des programmes** avec le minimum hardware peuvent fréquemment être formalisée sous forme du problème de coloration de graphe.

Le problème d'allocation de registres: qui cherche à assigner des variables à un nombre limité de registres durant l'exécution de programme. Deux variables ne peuvent pas être assignées dans le même registre si se sont « vive » en même temps. L'assignement de variables en plus amène à l'exécution rapide comme moins de variables nécessitent d'être rapportées de la mémoire. Pour formaliser ce problème, on crée un graphe dont les nœuds représentent les variables et les arêtes représentent les conflits entre les variables. Des diagrammes colorés pour l'assignement de libre conflit, et si le nombre de registres dépasse le nombre chromatique, un assignement de registre libre existe. [18]

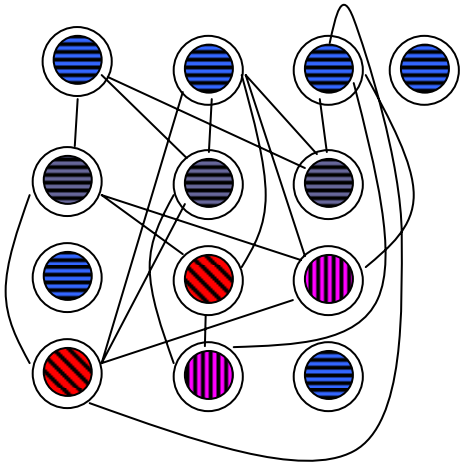


Figure 14: Exemple de graphe coloré.

VI.5 Problème de N-Queen :

Dans le problème de 8-Queens, on a un tableau d'échec régulier (8 par 8) et huit rênes qui doivent être placées dans le tableau dans un état dont pas de deux rênes dont une touche l'autre. Ce problème peut être naturellement généralisé, qui cède facilement le problème de N-queens. Beaucoup d'approches classiques d'IA qui s'appliquent à ce problème travaillent d'une manière constructive ou incrémentale : un commence par placer une rêne et après placer les n rênes, un tente de placer la ($n^{ième}$ +1) rêne dans la meilleure position, c'est-à-dire une position dont la nouvelle rêne ne peut touche les autres. Typiquement, quelque sorte du mécanisme **Backtracking** est appliqué : s'il n'y a pas de position faisable pour la ($n^{ième}$ +1) rêne, la $n^{ième}$ est déplacée dans une autre position. [16]

Figure 15 : le problème de 8-Queens [29]

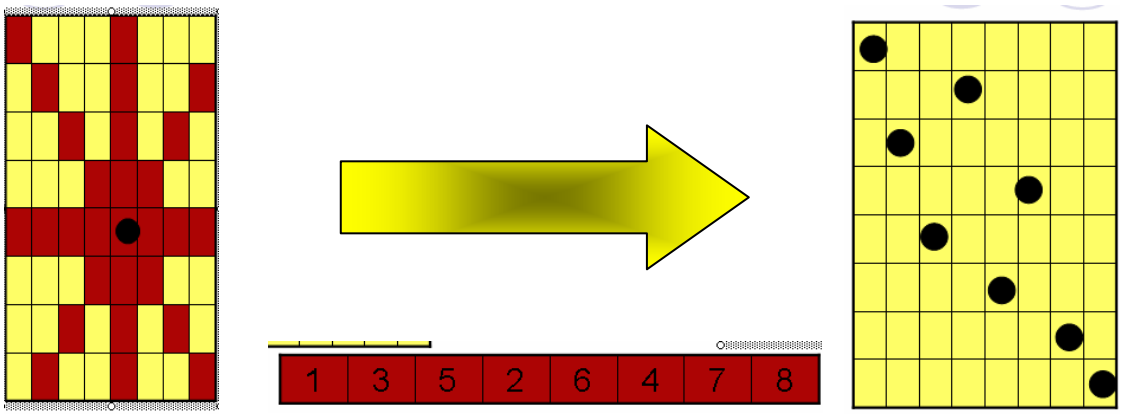


Figure 16 : l'illustration de la solution du problème de 8-Queens.

VII. Les méthodes d'optimisation Multiobjectifs :

Dès lors résoudre un problème multiobjectifs ne consiste pas à rechercher la solution optimale, mais l'ensemble des solutions satisfaisantes, pour lesquelles on ne pourra pas effectuer une opération de classement. Dans ce cas il faut utiliser des méthodes capables de trouver l'ensemble de Pareto tout en se basant sur les contraintes et la définition de problème. Dans la littérature, il existe plusieurs méthodes pour ce but, les premières tendances aboutissent à des méthodes purement mathématiques dont elles ne peuvent pas tendre vers l'ensemble de solutions dans un temps optimal et elles sont compliquées, par exemple, la programmation linéaire, la méthode de Simplexe, ... On outre, une deuxième tendance a pour but d'avoir des méthodes non mathématiques et tend vers l'ensemble de solutions dans un temps optimal et d'une manière simple en se basant sur le principe des heuristiques, cette tendance est « **les métaheuristiques** », comme les algorithmes évolutionnaires, recuit simulé, …

VII.1 Schéma de méthodes :

Il existe un nombre important de méthodes, dont ils apparaissent plusieurs types de classement selon des critères différents. Quelques auteurs les classifient en cinq groupes :

- **Les méthodes scalaires**,
- **Les méthodes interactives**,
- **Les méthodes floues**,
- **Les méthodes exploitant une métaheuristiques**,
- **Les méthodes d'aide à la décision**.

Ces méthodes de ces cinq groupes peuvent aussi être rangées en trois familles de méthodes d'optimisation multiobjectifs [37] :

- **Les méthodes à préférence a priori** :

Dans ces méthodes, l'utilisateur définit le compromis qu'il désire réaliser (il fait part de ses références) avant de lancer la méthode d'optimisation.

On retrouve dans cette famille la plupart des méthodes par agrégation (où les fonctions objectives sont fusionnées en une seule).

- **Les méthodes à préférence progressive** :

Dans ces méthodes, l'utilisateur affine son choix de compromis au fur et à mesure du déroulement de l'optimisation.

On retrouve dans cette famille les méthodes interactives.

- **Les méthodes à préférence a posteriori** :

Dans ces méthodes, l'utilisateur choisit une solution de compromis en examinant toutes les solutions extraites par la méthode d'optimisation.

Les méthodes de cette famille fournissant, à la fin de l'optimisation, une surface de compromis.

Il existe des méthodes d'optimisation multiobjectifs qui n'entrent pas exclusivement dans une famille. Par exemple, on peut utiliser une méthode à référence a priori en lui fournissant des préférences choisies au hasard. Le résultat sera alors un grand nombre de solutions qui seront présentées à l'utilisateur pour qu'il décide de la solution de compromis. Cette combinaison forme alors une méthode à préférence a posteriori. [02]

D'autres types de classement classifient les méthodes d'optimisation multiobjectifs selon le principe mathématique, c'est-à-dire est ce que les méthodes utilisent les dérivées des fonctions objectives ou non. Les méthodes dérivables exigent l'utilisation des dérivées de la fonction objective, et les méthodes non dérivables appelées aussi méthodes stochastiques ou les métaheuristiques.

VII.2 Les méthodes scalaires :

Parmi ces méthodes on trouve :

VII.3.1 La méthode de pondération de fonctions objectives :

Cette approche de la résolution de problème d'optimisation multiobjectifs est la plus évidente. D'ailleurs, on appelle aussi cette méthode l'« approche naïve » de l'optimisation multiobjectifs [38]. Le but, ici, est de revenir à un problème de l'optimisation multiobjectifs, dont il existe de nombreuses méthodes de résolution. La manière la plus simple de procéder consiste à prendre chacune des fonctions objectives, à

leur appliquer un coefficient de pondération et à faire la somme des fonctions objectives pondérées. On obtient alors une nouvelle fonction objective. [02]

VII.3.2 La méthode de Keeney-Raiffa :

Cette méthode utilise le produit des fonctions objectives pour se ramener à un problème d'optimisation multiobjectifs. L'approche utilisée ici est semblable à celle utilisée dans la méthode de pondération des fonctions objectives. La fonction objective ainsi obtenir s'appelle la fonction d'utilité de Keeney-Raiffa.

On retrouve cette fonction dans la théorie de la fonction utilité multi-attribut exposée dans [39] (M.A.U.T Multi Attribute Utility Theory). Cette théorie issue des sciences d'aide à la décision, traite des propriétés de la manière de créer une « fonction d'utilité ». [02]

VII.3.3 La méthode de la distance à un objectif de référence :

Cette méthode permet de transformer un problème d'optimisation multiobjectifs en un problème d'optimisation mono-objectif.

La somme que l'on va utiliser ici prendra la forme d'une distance ($\sqrt[k]{\sum(\ )^k}$) ou la racine k^e de la somme d'éléments élevés à une certaine puissance k). De plus, dans certains cas on normalisera les éléments par rapport à une valeur avant d'en faire la somme [47, 48]

VII.3.4 La méthode de compromis :

Cette méthode permet de transformer un problème d'optimisation multiobjectifs en un problème mono-objectif comportant quelques contraintes supplémentaires.

La démarche est la suivante : [02]

- On choisit un objectif à optimiser prioritairement,
- On choisit un vecteur de contraintes initial ;
- On transforme le problème conservant l'objectif prioritaire et on transforme les autres objectifs en contraintes d'inégalité.

On appelle aussi cette méthode la méthode de **la contrainte ε** [40].

VII.3 Les méthodes interactives :

Les méthodes interactives permettent de chercher une et une seule solution. Elles forment la famille des méthodes progressives et permettent à l'utilisateur de déterminer ses préférences vis-à-vis d'un compromis entre objectifs au cours de l'optimisation.

Ces méthodes sont à comparer aux méthodes :

- A préférence a priori, où l'utilisateur choisit le compromis à réaliser entre objectifs avant de lancer l'optimisation ;
- A préférence a posteriori, où l'utilisateur n'a pas à choisir de compromis avant de lancer l'optimisation. La méthode l'optimisation calcule toutes les solutions efficaces et l'utilisateur peut ainsi les comparer et en choisir une. [02]

Nous allons maintenant présenter quelques méthodes interactives (ou plutôt quelques principes d'interaction).

VII.3.1 La méthode de compromis par substitution :

Cette méthode (Surrogate Worth Tradeoff (SWT) ou « méthode de compromis par substitution » a été très utilisée pour l'optimisation de ressource en eau [41]. Elle est basée sur la méthode de compromis, à laquelle on a ajouté un processus interactif, pour que la méthode converge vers la solution la plus susceptible de satisfaire l'utilisateur. [02]

VII.3.2 La méthode de Jahn :

Cette méthode est complètement différente. On commence par choisir un point de départ, puis le problème est résolu pas à pas à travers l'espace de recherche en direction de la solution optimale au sens de Pareto [42].

Cette méthode est fondée sur une méthode de minimisation cyclique (aussi appelée « méthode d'optimisation scalaire dans les directions réalisables » de zoutendjik).

VII.3.3 La méthode de geoffrion :

Cette approche similaire à la méthode de John. Elle repose sur un algorithme nommé: l'algorithme de Frank-Wolfe [42]

VII.3.4 La méthode simplex :

Cette méthode n'a rien à voir avec la méthode du simplexe utilisée en programmation linéaire. Il s'agit la d'une méthode de recherche séquentielle [43] de l'optimum d'un problème d'optimisation. Le logiciel [multi simplex] réalise cette optimisation pour un problème d'optimisation multiobjectifs de manière interactive.

Cette méthode utilise $\kappa+1$ essais (ou κ représente la dimension de la variable de décision $\overleftarrow{\chi}$) pour définir des fonction objectif est obtenue en utilisant une méthode d'agrégation floue.

On commence par choisir au hasard $\kappa+1$ valeur pour la variable de décision $\overleftarrow{\chi}$. L'algorithme évalue alors les $\kappa+1$ points et supprime le point le moins « efficace ». Il crée alors un nouveau points à partir du point supprimé (voir figure 17) et recommence l'évaluation.

L'algorithme comporte quelques règles, qui permettent de choisir des points qui évitent de tourner auteur d'une mauvaise solution. Les deux principales règles sont les suivant :

Règle 1 : **rejeter les pires solutions.**

Une nouvelle position de la variable de décision $\overleftarrow{\chi}$ est calculée par réflexion de la position rejetée (voir la figure 17). Après cette transformation, on recherche le nouveau pire point. La méthode est alors répétée en éliminant ce point, …etc. A chaque étape, on se rapproche de la zone où se trouve l'optimum recherché.

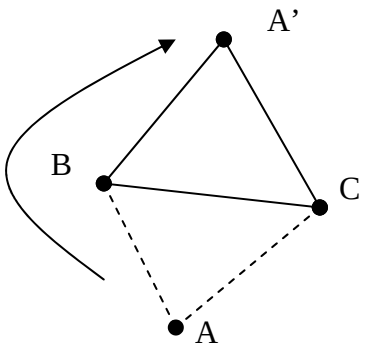


Figure 17 : Choix d'un nouveau point par symétrie [02]

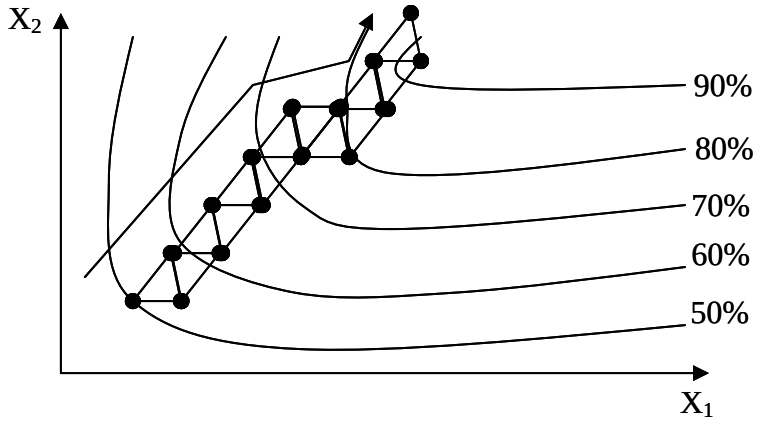


Figure 18 : Cheminement de la méthode du Simplex. [02]

Règle 2 : **ne jamais revenir sur un point qui vient juste d'être rejeté.**

Sans cette règle, l'algorithme pourrait osciller entre Deux « mauvais » points. [02]

W : point le moins favorable ou point venant d'être rejeté.

B : point le plus favorable.

R : le point le plus favorable.

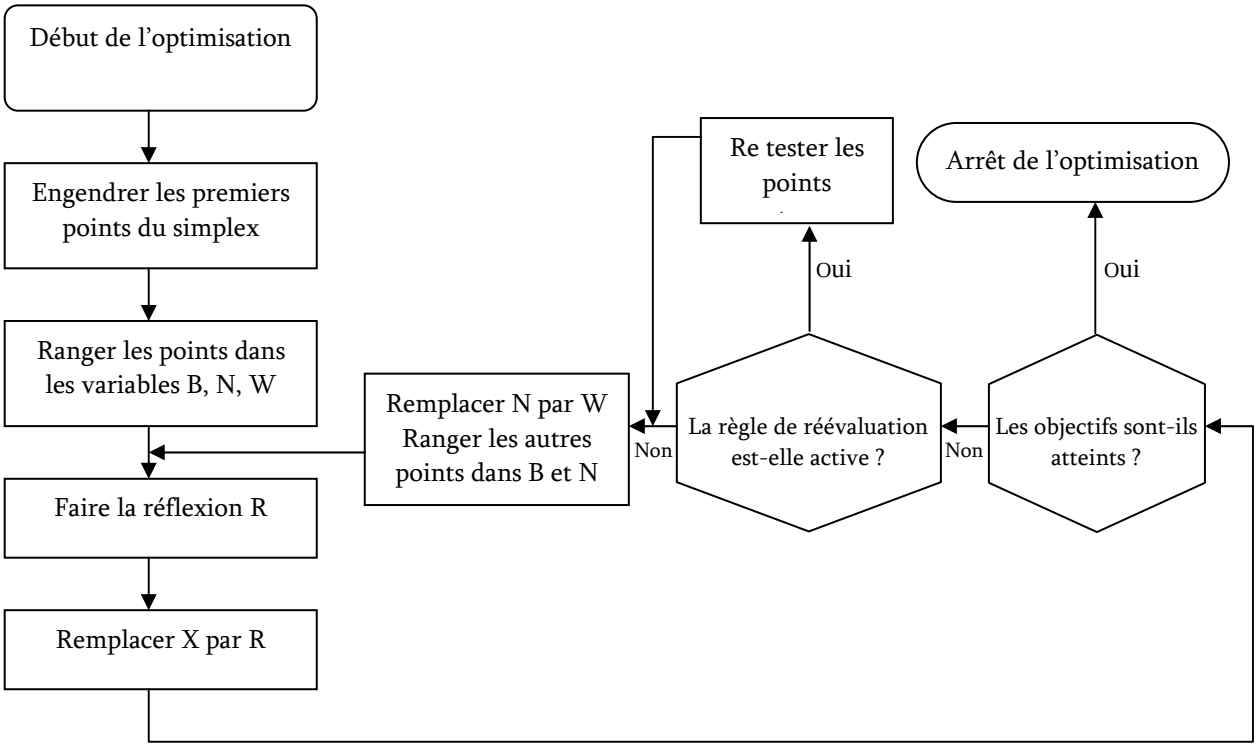


Figure 19 : L'algorithme de la méthode de Simplex. [02]

VII.4 Les méthodes floues :

Dans la vie courante, tout ne peut être décrit de manière binaire. Par exemple, la transition entre le jour et la nuit se fait progressivement, l'action sur l'embrayage d'un véhicule est, elle aussi, progressive.

Longtemps, le seul outil de description en logique était binaire. Tout en logique a été décrit en termes de VRAI ou FAUX. Le problème de cette description simpliste en logique et qu'elle ne permet pas de traiter l'incertitude de l'imprécision des connaissances humaines.

L'automaticien L. A. Zadeh a élaboré une nouvelle logique basée sur les ensembles flous. Elle permet de traiter l'imprécision et l'incertitude dans la connaissance humaine ainsi que les transitions progressives entre états.

La différence principale entre une logique classique et cette logique floue est l'existence d'une transition progressive entre le VRAI et le FAUX [44]

VII.4.1 La méthode de Sakawa :

Cette méthode fait intervenir la logique floue à tous les niveaux (sur les paramètres du problème ainsi que sur les paramètres des contraintes). L'ensemble de solutions que l'on va trouver, lui aussi, fera intervenir la logique floue. Ces solutions auront un niveau d'appartenance. Ce seront des solutions qui auront une corrélation variable avec l'objectif initial [45] (le niveau de corrélation avec l'objectif initial est fixé par l'utilisateur). [02]

VII.5 Les méthodes exploitant une métaheuristique :

VII.5.1 Qu'est-ce qu'une métaheuristique ?

Les métaheuristiques, sont des méthodes générales de recherche dédiées aux problèmes d'optimisation difficile [46]. Ces méthodes sont, en général, présentées sous la forme de concept d'inspiration. Comme nous le verrons plus tard, elles reprennent des idées que l'on retrouve parfois dans la vie courante. Ces méthodes ont des inspirations de l'éthologie comme les colonies de fourmis, de la physique comme le recuit simulé, et de la biologie comme les algorithmes évolutionnaires. Dans la figure suivante, on voit un classement de ces méthodes selon le principe d'inspiration utilisé, est ce qu'il est basé

sur le principe de population de solution ou non, aussi est ce que ces méthodes sont exactes ou non.

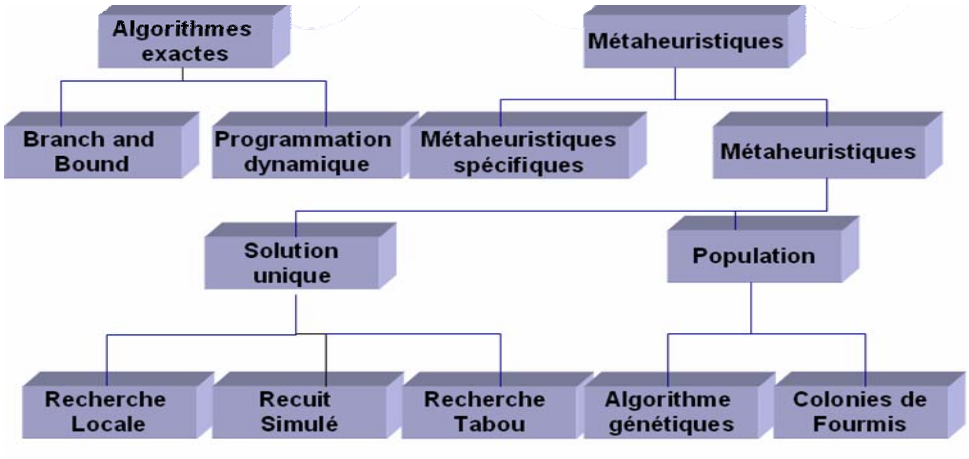


Figure 20 : Le schéma des méthodes basées sur les métaheuristiques. [14]

VII.5.2 Généralités :

Si l'on considère l'ensemble des méthodes de recherche d'optimum, on constate que celles-ci se réunissent autour de quelques concepts généraux. Par exemple, on trouvera beaucoup de méthodes (méthode de Newton, quasi-Newton, etc.) articulées autour de la descente de gradient. Le concept général de cette famille sera la descente de gradient, et ce concept général sera appelé métaheuristique.

Une métaheuristique est donc une méthode très générale, qui nécessite quelques transformations (mineure en générale) avant de pouvoir être appliquée à la résolution d'un problème particulier.

Si l'on considère les différentes métaheuristiques, on constate que celles-ci se répartissent en trois familles :

- **Les méthodes déterministes de recherche d'optimum local** :

Ces méthodes convergent rapidement mais, la plupart de temps, elles ne trouvent pas l'optimum global. Elles se contentent de trouver un optimum local et ne reposent pas sur un processus stochastique pour la recherche de l'optimum (voir 21).

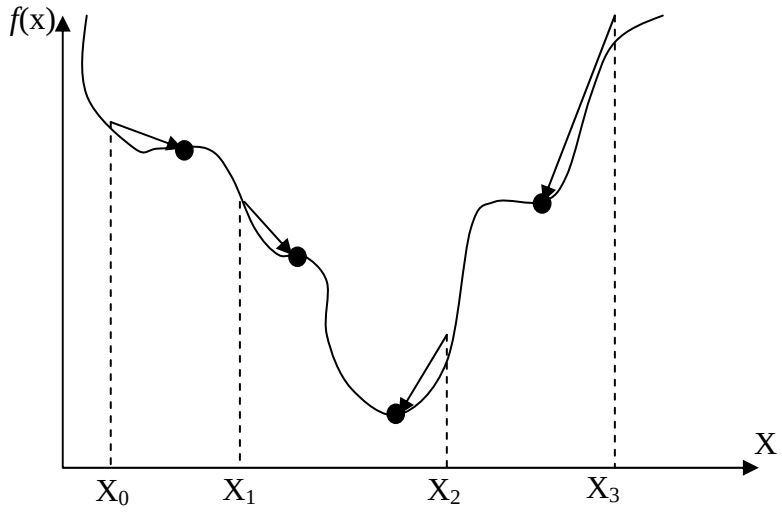


Figure 21 : Méthode déterministe de recherche de l'optimum local à partir de différentes solutions [02]

- **Les méthodes déterministes de recherche de l'optimum global** :

Ces méthodes permettent de trouver un optimum global rapidement et ne reposent pas sur un processus stochastique pour la recherche de l'optimum (voir 22).

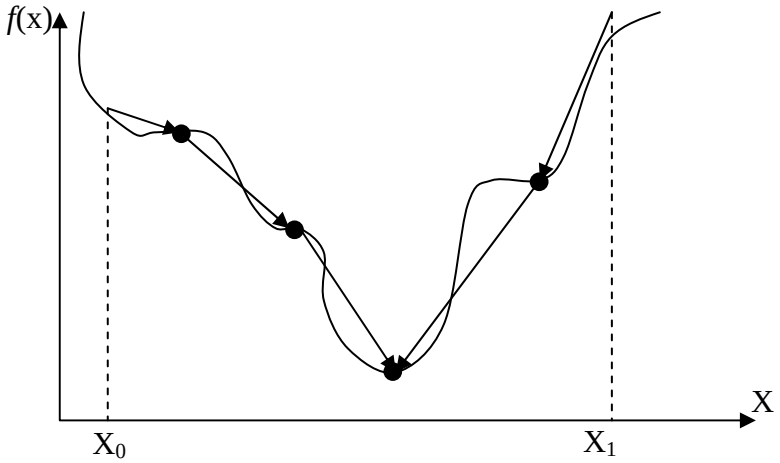


Figure 22 : Méthode déterministe de recherche d'optimum global [02]

- **Les méthodes stochastiques de recherche de l'optimum global** :

Ces méthodes reposent sur un processus stochastique chargé d'effectuer la recherche de l'optimum. Elles sont moins performantes (du point de vue rapidité) que les méthodes déterministes, mais elles peuvent trouver, en principe, un optimum global difficile à atteindre (voir 23).

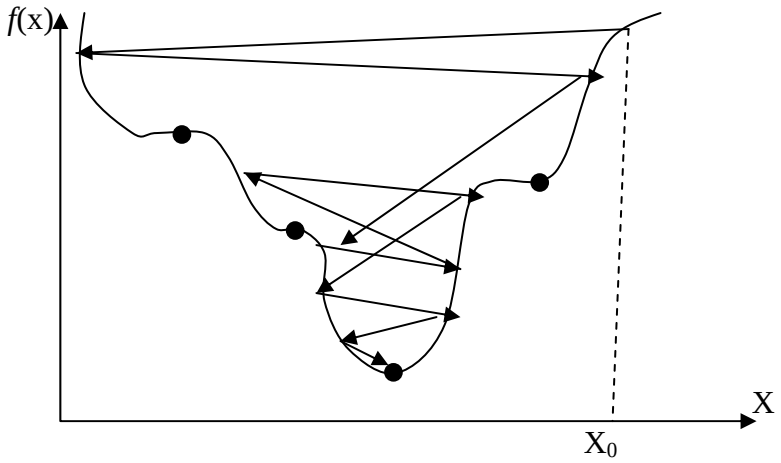


Figure 23 : Méthode stochastique de recherche d'optimum global [02]

VII.5.3 Le recuit simulé

Cette méthode de recherche a été proposée par des chercheurs d'IBM qui étudiaient les verres de spin. Ici, on utilise un processus métallurgique (le recuit) pour trouver un minimum. En effet, pour qu'un métal retrouve une structure proche du cristal parfait (l'état cristallin correspond au minimum d'énergie de la structure atomique du métal), on porte celui-ci à une température élevée, puis on le laisse refroidir lentement de manière à ce que les atomes aient le temps de s'ordonner régulièrement [47].

Ce processus métallurgique a été transposé à l'optimisation et a donné une méthode simple et efficace. Le pseudo code du recuit simulé est représenté dans l'algorithme 1.

Le fonctionnement de cet algorithme est le suivant :

- On commence par choisir un point de départ au hasard :
- On calcule un voisin de ce point ($\gamma = V$*oisin (x)).*
- On évalue ce point voisin et on calcule l'écart par rapport au point d'origine $(\Delta C = C(\gamma) - C(x))$.
- Si cet écart est négatif, on prend le point γ comme nouveau point de départ, s'il est positif on peut quand même accepter le point γ comme nouveau point de départ, mais avec une probabilité $e^{-\frac{\Delta C}{T}}$ (qui varie en sens inverse de la température *T).*

- Au fur et à mesure du déroulement de l'algorithme, on diminue la température $T(T = \alpha(T))$, souvent par paliers.
- On répète toutes ces étapes tant que le système n'est figé (par exemple, tant que la température n'a pas atteint un seuil minimal).

Une recherche de recuit simulé acceptera n'importe quelles nouvelles solutions qui sont évaluées comme des solutions supérieures, mais elle acceptera aussi les changements négatifs de qualité avec une probabilité qui dépend de la taille de diminution dans la qualité et la valeur courante de la température. La température commence à une haute valeur, idéalement assez haute que n'importe quelle solution inférieure aura presque une chance de 100 pourcent d'être acceptée, et les diminutions comme un processus exécute ces parcours. La chance d'une solution négative pouvant être acceptée diminue, durant il y a une chance de zéro n'importe quel changement négatif de qualité peut être alloué. A ce point « refroidissement » est dit d'avoir lieu, et le système doit avoir convergé à la solution optimale. [61]

Algorithme 1 Recuit simulé.

Init T (température initial)
Init x (point de départ)
Init ΔT (temperature)
While (not (end))
y =V*oisin* (x)
$\Delta C = C$ (y)-C (x)
if $\Delta C < 0$ then $y = x$
else if alea (0,1) < $e^{-\frac{\Delta C}{T}}$ then $y=x$
$T=\alpha$ (T)
if $T < \Delta T$ then end (while)
Repeat (while)

Quelques applications du recuit simulé :

- ✓ Traitement d'images.
- ✓ Problèmes d'ordonnancement.

- ✓ Conception des circuits électroniques (Problème de placement et de routage).
- ✓ Organisation du réseau informatique du Loto (France).
- ✓ Collecte des ordures ménagères.
- ✓ Problème du voyageur de commerce (10000 villes). La longueur de la tournée excède de moins de 2% celle de la tournée optimale.

Avantages de la méthode

- ✓ Solution de bonne qualité.
- ✓ Méthode générale et facile à programmer.
- ✓ Souplesse d'emploi: De nouvelles contraintes peuvent être facilement incorporées

Inconvénients:

- ✓ Nombre important de paramètres.
- ✓ Temps de calcul: excessif dans certaines applications. [62]

Dans la littérature des méthodes d'optimisation multiobjectifs, on trouve deux importantes extensions de la méthode Recuit Simulé basées sur les frontières de Pareto.

➔ **La méthode P.A.S.A (*Pareto Archived Simulated Annealing)***

Cette méthode, développée à EDF [48] utilise une fonction d'agrégation des fonctions objectives couplées avec un système d'archivage de solutions non dominées.

Algorithme 2 La méthode PASA.

1. Choix de P_0, une population d'archive initiale, éventuellement réduite à un seul élément x_0, et choix de T_q initiale.
2. Soit $\vec{X'}_n$ un voisin de $\vec{X}_n$
3. Calcul de $G(\vec{X'}_n)$ et de $\Delta G = G(\vec{X'}_n) - G(\vec{X}_n)$
4. Si $\Delta G < 0$ alors $\overrightarrow{X_{n+1}} = \vec{X'}_n$
5. Si $\Delta G > 0$ alors soit $p = \exp\left(-\frac{\Delta G}{T_q}\right)$
   - $\overrightarrow{X_{n+1}} = \vec{X'}_n$ avec une probabilité p
   - $\overrightarrow{X_{n+1}} = \vec{X}_n$ avec une probabilité 1-p
6. Si on sort de la boucle interne, alors $T_{q+1} = \alpha.T_q$ avec $\alpha<1$

7. Principe de non-dominance pour l'archivage de $\overrightarrow{x_{n+1}}$
   - Si $\overrightarrow{x_{n+1}}$ domine une des solutions archivées $\vec{y}_p$, $\overrightarrow{x_{n+1}}$ remplace $\vec{y}_p$ dans la population archivée ;
   - Si $\overrightarrow{x_{n+1}}$ est dominée par au moins une des solutions archivées, alors on ne l'archive pas ;
   - Si $\overrightarrow{x_{n+1}}$ n'est pas dominée par la population archivée alors on l'ajoute à la population archivée, et on retire de celle-ci les solutions dominées par $\overrightarrow{x_{n+1}}$.
8. *Return to base :* lorsque c'est nécessaire, $\overrightarrow{x_{n+1}} = \vec{y}_i$ (i choisi aléatoirement dans la population archivée)
9. Si (non convergence) alors retour en 2.

➔ **La méthode M.O.S.A (*Multiple Objective Simulated Annealing)***

Cette méthode a été proposée dans [49]. Elle utilise le recuit simulé pour rechercher la surface de compromis. Cette méthode fonctionne bien car, à haute température, le recuit simulé répartis les individus sur toute une surface, et non sur un point. A la fin de l'optimisation, la température de fonctionnement est encore suffisamment élevée pour qu'un effet stochastique distribue les solutions potentielles sur un front.

VII.5.4 La Recherche Tabou :

Cette méthode, mise au point par F. Glover [50] est conçue en vue de surmonter les minimax locaux de la fonction objective. C'est une technique d'optimisation combinatoire que certains présentent comme une alternative au recuit simulé.

A partir d'une configuration initiale quelconque, Tabou engendre une succession de configurations qui doivent aboutir à la configuration optimale. A chaque itération, le mécanisme de passage d'une configuration, soit x_n, à la suivante, soit x_{n+1}, est le suivant :

- on construit l'ensemble des « voisins » de x_n, c'est-à-dire l'ensemble des configurations accessibles en un seul « mouvement » élémentaire à partir de x_n (si cet ensemble est trop vaste, on en extrait aléatoirement un sous-

ensemble de taille fixée) : soit *Voisinage* (x_n) l'ensemble (ou le sous-ensemble) envisagé ;

- on évalue la fonction objective *f* du problème de chacune des configurations appartenant à *Voisinage*(x_n). La configuration x_{n+1}, qui succède à la configuration x_n dans la chaîne de Markov construite par Tabou, est la configuration de *Voisinage* (x_n) en laquelle *f* prend sa valeur minimale.

Notons que la configuration x_{n+1} est adoptée même si $f(x_{n+1}) > f(x_n)$: c'est grâce à cette particularité que Tabou permet d'éviter les minimax locaux de *f*.

Cependant, telle quelle la procédure ne fonctionne généralement pas, car il y a un risque important de retourner à une configuration déjà retenue lors d'une itération précédente, ce qui provoque l'apparition d'un cycle. Pour éviter ce phénomène, on tient à jour, à chaque itération, une « liste Tabou » de mouvements interdits ; cette liste -qui a donné son nom à la méthode- contient les mouvements inverses ($x_{n+1} \rightarrow x_n$) des m derniers mouvements ($x_n \rightarrow x_{n+1}$) effectués (typiquement m=7). La recherche du successeur de la configuration courante x_n est alors restreinte aux voisins de x_n qui peuvent être atteints sans utiliser un mouvement de la liste tabou. La procédure peut être stoppée dès que l'on effectué un nombre donné d'itérations, sans améliorer la meilleure solution atteinte jusqu'ici.

L'algorithme ainsi décrit est dit « Tabou simple ». Selon [51], il serait plus efficace que le recuit simulé pour le problème modèle du « coloration d'un graphe ». Cependant, le mode de construction de la liste tabou – qui, pour une simple raison d'économie de place mémoire, contient des mouvements interdits, et non des configurations interdites – peut bloquer l'accès à certaines solutions, pourtant non encore visitées. Pour éviter cet inconvénient, on peut employer la méthode plus complexe dite « tabou généralisée », qui prévoit la possibilité d'annuler le statut tabou d'un mouvement, lorsque le bénéfice escompté est « suffisant » (cette circonstance est appréciée à l'aide de la notion de « niveau d'aspiration ».

Le pseudo-code de cette méthode est représenté dans l'algorithme 3.

Algorithme 3 Recherche Tabou.

Init L=Ø (liste des points tabous)

Init x (point de départ)

Init k_{fin} (nb d'itérations max total)

Init k=0 et l=0

While (not (end))

y =V*oisinage* (x)–L (on prend un voisinage sans points tabous)

$\Delta C = C\ (y)$-C (x)

$x = y$ avec $y \in$ *Voisinage (x)* – *L* tel que min *C(y)* – *C(x)*

L = L+ {x}

k = k + 1

Update L (on retire des mouvements tabous de L suivant des critères d'aspiration)

if $k = k_{fin}$ then end (while)

Repeat

VII.5.5 La méthode GRASP:

La métaheuristique GRASP a été proposée initialement par Féo et Resende [60]. C'est une méthode multi-départ en deux phases pour la résolution approchée de problèmes difficiles de l'optimisation combinatoire. La première phase correspond à la construction d'une solution initiale à l'aide d'une procédure gloutonne aléatoire. L'incorporation d'une composante aléatoire permet d'obtenir des solutions dans des zones diverses de l'espace des solutions. La seconde phase correspond à une recherche locale améliorant ces solutions. Ce processus est répété un grand nombre de fois.

De plus, plusieurs nouveaux composants sont venus de compléter le schéma de base de GRASP comme par exemple :

– l'évolution, appelée réactive GRASP, permettant de déterminer de manière dynamique la valeur du paramètre α pilotant l'importance de la composante aléatoire,

– l'utilisation de techniques, à base de mémoire et d'apprentissage, pour profiter de l'information apportée par les solutions générées aux itérations précédentes,

– le couplage avec des méthodes de path relinking pour rechercher des solutions de bonne qualité en recomposant des chemins entre les solutions trouvées par GRASP.

En outre, de nombreuses hybridations avec d'autres métaheuristiques ont été proposées.

Une implémentation de GRASP pour un problème particulier va être caractérisée par six éléments principaux :

– l'algorithme glouton utilisé,

– l'importance de la composante aléatoire (fixée par un paramètre $\alpha \in [0; 1]$, une valeur $\alpha = 1$ correspondant à un glouton pur et une valeur $\alpha = 0$ correspondant à un algorithme aléatoire),

– le type de recherche locale et le voisinage considéré,

– la phase d'intensification éventuelle,

– le critère d'arrêt,

– et la phase de post-optimisation éventuelle.

L'adaptation de GRASP pour résoudre un problème particulier est assez facile lorsqu'ils existent des algorithmes de construction et de recherche locale pour ce problème. GRASP a ainsi été appliqué avec succès à un grand nombre de problèmes d'optimisation comme des problèmes d'ordonnancement, de routage ou encore des problèmes théoriques dans les graphes. [59]

VII.5.6 La méthode de Colonie de Fourmis :

Les insectes sociaux, comme les fourmis, les abeilles ou les termites sont généralement imaginées d'une manière simple, des animaux non intelligents. Néanmoins, ils exhibent collectivement des compétences impressives pour résoudre les problèmes. Inspirant de ces insectes, les recherches dans les décennies passées ont guidé en quelques progrès fascinant dans le champ d'algorithmes naturels. Ces algorithmes imitent la nature d'une manière ou d'une autre. Les réseaux de neurones imitent la structure de notre cerveau humain et les algorithmes génétiques simulent l'évolution. Ils sont caractérisés par

le parallélisme inhérent, l'adaptation, le feed-back positif et quelques éléments de l'aléatoire. [52]

La recherche guidée par Marco Dorigo produisit un nouveau membre de cette classe d'algorithmes : l'algorithme de « **le système de fourmis** ». Ici, la manière dont ces insectes trouvent le chemin le plus court à partir d'une ressource alimentaire à leur nid est copiée dans une tentative de résoudre les problèmes durs d'optimisation combinatoire [52]. A l'origine, l'optimisation par colonie de fourmis a été conçue pour résoudre le problème du voyageur de commerce (PVC). L'objectif de ce problème est de trouver la tournée la plus courte pour un ensemble de villes données. Une matrice fournissant les distances, d_{ij}, entre toutes les paires de villes est utilisée pour estimer la longueur d'une tournée. [53]

Les colonies de fourmis, et plus généralement les insectes sociaux, sont des systèmes distribués qui, dans l'aspect de la simplicité de leurs individus, présentent une forte structure d'organisation sociale. Comme un résultat de cette organisation, les colonies de fourmis peuvent accomplir les tâches complexes qui sont en quelques sortes loin dépasser les capacités d'individu pour une seule fourmi.

Le champ des algorithmes de fourmis étudie les modèles dérivés à partir de l'observation du comportement réel de fourmis, et utilise ces modèles comme une ressource d'inspiration pour la conception de nouveaux algorithmes pour la solution pour les problèmes d'optimisation de contrôle distribué. [54]

L'idée principale c'est que les principes de l'auto organisation qui allouent les coordinations de comportement des vraies fourmis peuvent être exploités pour coordonner la population d'agents artificiels qui collaborent pour résoudre les problèmes de l'ordinateur. Plusieurs aspects différents pour le comportement de colonies de fourmis ont inspirés des natures différentes d'algorithmes de fourmis. Des exemples sont le fourrage, division de travailliste, triage de nichée, et le transport coopérative. Dans tous ces exemples, les fourmis coordonnent leurs activités à partir de la stigmergie, une forme depuis une communication indirecte s'interposa par des modifications de

l'environnement. Par exemple, une fourmi de fourrage dépose un produit chimique sur le terrain qui monte la probabilité que les autres fourmis suivront le même chemin. Les biologistes ont démontré que quelques comportements de niveaux de colonies dans les insectes sociaux peuvent être expliqué à partir plutôt des modèles simples dont seulement la communication stigmergique est présente. Dans d'autres termes, les biologistes ont démontré qu'il est souvent suffisant de considérer la communication indirecte pour expliquer comment les insectes sociaux peuvent atteindre l'auto-organisation. L'idée derrière les algorithmes de fourmis est alors d'utiliser une forme de stigmergie artificielle pour coordonner les sociétés des agents artificiels. [54]

Un parmi les plus exemples qui ont un succès des algorithmes de fourmis est connu sous le nom « **Ant Colony Optimization** », ou ACO. ACO est inspiré par le comportement de colonies de fourmis, et fixe les problèmes d'optimisation discrets. [54]

L'optimisation par colonie de fourmis (OCF) est une métaheuristique qui a été utilisée avec succès pour résoudre plusieurs problèmes d'optimisation combinatoires. Toutefois, la plupart des travaux ont consiste à solutionner des problèmes a objectif unique. Dans les travaux utilisant l'OCF dans un contexte multiobjectifs, [55] ont présenté une approche qui favorise I'échange d'information entre des familles d'agents travaillant sur des objectifs différents pour la conception d'un réseau de distribution d'eau. Leur approche vise a générer I'ensemble Pareto-optimal. [56] ont développé un OCF bi-objectifs ou la meilleure solution globale est partagée par deux colonies pour solutionner un problème de tournées de véhicules. Dans cette approche, une colonie cherche à abaisser la meilleure solution connue sur le premier objectif tandis que d'autre cherche à optimiser le deuxième objectif tout en préservant la solution sur le premier. [57] proposent, pour leur part, une approche bi-objectifs de I'OCF basée sur la coopération de plusieurs colonies pour un problème d'ordonnancement de machine unique avec réglages dépendant de la séquence. Chacune des colonies possède deux matrices de trace de phéromone utilisées de manière différente par chacune des fourmis. [58]

VII.5.7 Monté Carlo :

Les méthodes Monté Carlo consistent en des simulations expérimentales ou informatiques de problèmes mathématiques ou physiques, basées sur le tirage de nombres aléatoires. Généralement on utilise en fait des séries de nombres pseudo-aléatoires générées par des algorithmes spécialisés. Les propriétés de ces séries sont très proches de celles d'une véritable suite aléatoire.

La méthode Monté Carlo est également utilisée dans le domaine pharmaceutique : on génère in vitro de très nombreuses molécules aléatoires, puis on les passe au crible en testant leur effet sur tel ou tel cible. On repère ainsi des molécules intéressantes qui après une étude et une modification pourront donner naissance à de nouveaux médicaments. L'orientation actuelle est même de réaliser la même chose in silico, c'est-à-dire de modéliser et de tester ces molécules dans un ordinateur.

Le grand avantage de cette méthode est sa simplicité. Elle permet entre autres de visualiser l'effet de différents paramètres et de donner ainsi des orientations, d'étudier des structures intéressantes qui auraient été a priori écartées et de trouver facilement des structures que l'on n'aurait pas aussi bien optimisées « à la main ». [61]

Les méthodes de types Monté Carlo recherchent l'optimum d'une fonction en générant une suite aléatoire de nombres en fonction d'une loi uniforme.

Algorithme :

- On génère un point initial x dans l'espace d'état, considéré comme solution courante.
- On génère aléatoirement un point x'.
- Si x' est meilleur que x alors x' devient la solution courante.
- Si le critère d'arrêt est satisfait alors fin sinon retour en à la deuxième étape [04]

VII.5.8 Particle Swarm Optimization « PSO »

L'optimisation par particule d'essaim (PSO) c'est un algorithme d'intelligence d'essaim, inspiré par le social dynamique et le comportement émergeant qui surgissent dans les colonies socialement organisées, PSO est un algorithme basé sur la population, c'est-à-dire il exploite une population d'individus pour explorer les régions prometteuses

de l'espace de recherche. Dans ce contexte, la population est nommée les essaims et les individus (c'est-à-dire les point de recherche) sont nommés les particules. Chaque particule meut avec une vitesse adaptable avec l'espace de recherche, et garde une mémoire de la meilleure position dont elle rencontre. Dans les variant global de PSO, la meilleure position qui est retenue par les individus de l'essaim est communiquée aux autres particules. Dans le variant local, chaque particule est assignée à un voisinage topologique consisté d'un nombre pré-spécifié de particules. Dans ce cas, la meilleure position retenue par les particules qui comprend le voisinage est communiquée entre eux. [63]

VII.5.9 Les algorithmes évolutionnaires :

Les algorithmes évolutionnaires sont parmi les métaheuristiques à base de population, ils sont inspirés de la biologie toute en introduisant le théorème de Darwin dans l'évolution des espèces. On va présenter en détail cette tendance d'algorithme dans le chapitre suivant.

VII.5.10 Les méthodes hybrides :

Clairement les différentes méthodes ont différents avantages et il est donc attractif de produire des méthodes hybrides. [64] classifie les algorithmes génétiques hybrides en quatre catégories, qui vont pour classifier d'autres hybridations comme suit :

1/ Les méthodes hybrides en pipe-line.

2/ Les méthodes hybrides asynchrones

3/ Les méthodes hybrides hiérarchiques

4/ Les opérateurs additionnels

Les méthodes hybrides en pipe-line :

C'est la plus simple et la plus directe manière pour l'implémentation hybride, premièrement, celui commence par explorer tout l'espace de recherche avec une méthode comme l'identification de l'optimum global mais peut être avec une convergence lente. Après l'identification des régions prometteuses, celui peut reporter à une méthode avec une haute convergence tarif pour faire accélérer la recherche. Ça peut être accomplis par

une combinaison d'une instance d'un algorithme génétique avec un algorithme de gradient.

Les méthodes hybrides asynchrones

Les différentes méthodes hybrides asynchrones travaillent en différents sous-ensembles de solutions d'espace. Les différentes méthodes peuvent travailler sur un sous-ensemble d'une population partagée pour quelques itérations. Si les individus pour les quels un sous-ensemble être plus performant que les autres dans la population partagée, ils ont alloués pour immigrer vers eu. Avec cette approche, une méthode qui converge lentement peut être combinée avec une qui converge rapidement, ou une méthode qui performe bien sur un sous-ensemble de l'espace de recherche peut être combinée avec une méthode qui performe bien un autre sous-ensemble. La même idée est employée dans les algorithmes génétiques avec des populations multiples, alors pour trouver des solutions multiples dans un espace multimodal.

Les méthodes hybrides hiérarchiques

Les méthodes hybrides hiérarchiques, des méthodes d'optimisation différentes sont appliquées dans des niveaux différents de l'optimisation. Dans un niveau global, vous pouvez appliquer une stratégie robuste d'optimisation pour trouver une disposition optimale. Dans les niveaux inférieurs, où le système doit être moins complexe et sensible il est plus approprié d'employer pour une instance de méthode purement analytique.

Les opérateurs additionnels

Dans la littérature il y a quelques exemples d'hybridation entre les différentes méthodes d'optimisation où des opérateurs d'une méthode sont ajoutés à ces méthodes ou même sont remplacés les opérateurs standard des autres méthodes. Par exemple, Yen et al. [64] ont introduis une méthode de simplexe comme une nouvelle manière de générer les enfants dans un algorithme génétique. Dans chaque itération un certain pourcentage de la population est généré par employer une méthode de simplexe sur un groupe d'individus prometteurs. Le reste de la population est généré en utilisant les opérateurs usuels génétiques. Ça est améliore la vitesse de convergence aussi que la compétence de trouver l'optimum absolue. [65]

VIII. Conclusion :

Dans ce chapitre, nous avons essayé de rassembler un état de l'art sur les problèmes d'optimisation multiobjectifs, tout en montrant la manière de définir un problème pareil, ses contraintes, ses objectifs ainsi que les méthodes utilisées pour le résoudre tout en respectant le concept de Compromis et les frontières de Pareto. Nous avons parlé sur les méthodes utilisées dans le domaine de l'optimisation multiobjectifs soit les problèmes mathématiques ou les métaheuristiques, parmi les métaheuristiques existant, nous trouvant les algorithmes évolutionnaires qui exercent le principe de la génétique pour résoudre ces problèmes et ils ont démontrés leur efficacité, pour cette raison et puisque notre travail se base sur l'un des algorithmes génétiques nous parleront dans le chapitre suivant en détails sur ces algorithmes.

Chapitre II

Les Algorithmes Evolutionnaires

I Introduction :

Le chapitre précédent discute les problèmes combinatoires, des exemples de ces derniers ainsi que les méthodes utilisées pour trouver des solutions adéquates. Parmi ces méthodes on trouve les plus répondues sont les algorithmes évolutionnaires inspirées de la biologie, ses concepts et ses bases dans l'évolution.

L'évolution biologique a engendré des systèmes vivants extrêmement complexes. Elle est le fruit d'une altération progressive et continue des êtres vivants au cours des générations et s'opère en deux étapes : la sélection et la reproduction.

* La sélection naturelle est le mécanisme central qui opère au niveau des populations, en sélectionnant les individus les mieux adaptés à leur environnement.

* La reproduction implique une mémoire : l'hérédité, sous la forme de gènes. Ce matériel héréditaire subit, au niveau moléculaire, des modifications constantes par mutations et recombinaisons, aboutissant ainsi à une grande diversité.

Ces principes, présentés pour la première fois par Darwin, ont inspiré bien plus tard les chercheurs en informatique. Ils ont donné naissance à une classe d'algorithmes regroupée sous le nom générique d'Algorithmes Evolutionnaires (ou Evolutionary Algorithms (EA)). [67]

II Historique et définitions :

II.1 Historique :

Le terme Algorithme évolutionnaire (EA) porte sur des méthodes d'optimisation stochastiques qui simulent le processus de l'évolution naturelle. Les origines d'AE peuvent être rendu aux années 1950, et depuis les années 1970 beaucoup de méthodologies évolutionnaires ont été proposées. [71]

En **1860** 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 rejette 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. La sélection naturelle : Sélection des individus les mieux « adaptés » à un milieu donné et qui auront une plus grande faculté de reproduction que les autres. La sélection naturelle soutient donc que les êtres vivants qui s'adaptent le mieux aux conditions naturelles de leur environnement vaincront et survivront. [69]

Dans les années 60 et 70, les premiers travaux sur l'évolution artificielle ont concerné les AGs, les stratégies d'évolution (SE) et la programmation évolutive. Ces trois types d'algorithmes ont utilisé des principes globalement communs car ils se sont tous inspirés des mêmes principes du néo-darwinisme: utilisation d'une population d'individus, évaluation des individus par une fonction, sélection des meilleurs et génération d'une nouvelle population avec des opérateurs de croisement et de mutation. Cependant, des choix méthodologiques ont initialement opposé ces méthodes. Ainsi, les premiers AG utilisaient plutôt un codage binaire des individus alors que les SE utilisaient un codage en nombre réel.

Au **20ème siècle,** Mise en évidence de l'existence de mutations génétiques. 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.

En **1966** L. J. Fogel introduisit le concept de la programmation évolutionnaire.

En **1973** I. Rechenberg donna lieu aux stratégies d'évolution.

J.H. Holland, professeur à l'université du Michigan, entreprit avec ses étudiants, **en 1975**, une vaste étude qui permit de poser les fondements des AG en calquant les principes de Darwin (sélection, croisement, mutation, chromosome, gènes). Il parvint alors, à mettre

au point les étapes de l'algorithme et ses principes de codage. Il esquissa aussi les grandes perspectives d'application des AGs. Ces travaux ont suscité un intérêt sans cesse croissant pour les mathématiciens dont Koza qui valida rigoureusement leurs mécanismes. Ensuite, dans les années 90 est apparue la programmation génétique qui introduit notamment des représentations arborescentes. [70]

En **1989** David Goldberg publie un ouvrage de vulgarisation des algorithmes génétiques.

Aux **années 1990**, une programmation d'une panoplie d'algorithmes génétiques transcrit en C++, appelée GAlib. Cette librairie contient des outils pour des problèmes d'optimisation en utilisant les AG. Elle est conçue pour servir de support de programmation. [75]

Aucun ne peut réellement dit qu'il a imaginé le premier algorithme évolutionnaire. Est il Fraser, Friedberg, Anderson, Bremermann ou un autre ? C'est sure, néanmoins, que aujourd'hui il y a trois différentes écoles dont ses racines ont été développées indépendamment de quelqu'un autre. Mais récemment ils ont trouvé ensemble et accepté d'un commun accord le dénominateur commun EC et EA pour :

- Les algorithmes génétiques (Gas),
- La programmation évolutionnaire (EP),
- Les stratégies d'évolution (ESs). [68]

Plus tard, une autre classe d'algorithmes évolutionnaires vit le jour, **la programmation génétique** (GP) de J. Koza qui était auparavant un sous groupe des Algorithmes Génétiques.

Ces différentes classes d'algorithmes évolutionnaires ne diffèrent que sur les détails d'implantation des opérateurs et sur les procédures de sélection et remplacement de la population. Malgré que leur but soit différent à l'origine, ils sont maintenant surtout utilisés pour résoudre des problèmes d'optimisations. [77]

II.2 Définition :

- Algorithmes Evolutionnaires : L'appellation Algorithmes Evolutionnaires (EAs) est un terme générique pour décrire les approches informatiques reposant sur les principes de l'évolution comme éléments clefs de leurs fonctionnements.
- Il existe un grand nombre d'algorithmes évolutionnaires : toutes ces approches ont en commun dans la simulation de l'évolution de structures individuelles au travers de méthodes de sélection, mutation et reproduction. Ces processus reposent sur la performance des structures individuelles à leur environnement.
- Plus précisément : Les EAs maintiennent une population de structures, qui évoluent en accord avec des règles de sélection et d'autres opérateurs, appelés opérateurs de recherche.
- Chaque individu possède une mesure de son adéquation avec l'environnement.
- La reproduction : s'appuie sur cette mesure pour favoriser la prolifération des individus les mieux adaptés.
- La recombinaison et la mutation : perturbent ces individus en fournissant par ce biais des heuristiques pour l'exploration de nouvelles solutions.
- Bien que relativement simples ces algorithmes sont suffisants pour assurer des mécanismes de recherche robustes et puissants. [72]

II.2.1 La diversité génétique :

Le terme *diversité génétique* désigne la variété des individus présents dans la population. Elle devient nulle lorsque tous les individus sont identiques. Mais il est important de savoir que lorsque la diversité génétique devient très faible, il y a très peu de chances pour qu'elle augmente à nouveau. Et si cela se produit trop tôt, la convergence a lieu vers un optimum local, on parle alors de *convergence prématurée*. Il faut donc préserver la diversité génétique, sans pour autant empêcher la convergence. [73]

II.2.2 Le dilemme exploration - exploitation

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

Typiquement, les opérations de sélection et de croisement sont des étapes d'exploration, alors que l'initialisation et la mutation sont des étapes d'exploitation. On peut ainsi régler les parts respectives d'exploration et d'exploitation en jouant sur les divers paramètres de l'algorithme (probabilités d'application des opérateurs, pression de sélection, ...). Malheureusement, il n'existe pas de règles universelles de réglages et seuls des résultats expérimentaux donnent une idée du comportement des divers composantes des algorithmes. [73]

La variation d'algorithmes évolutionnaires qui a été proposée (Algorithme Génétique, Programmation Evolutionnaire, Les Stratégies d'évolution, Programmation Génétique) partagent tous une base conceptuelle commune pour simuler l'évolution de structure d'individu via les processus de sélection, mutation, et reproduction. Les processus dépendent sur la performance perçue des structures d'individu comme les définit l'environnement.

Précisément, les algorithmes évolutionnaires maintiennent une population de structures, qui évoluent selon les règles de ces opérateurs, appelés « opérateurs de recherche », (ou opérateurs génétiques). D'après le point de vue simple des biologistes, ces algorithmes sont suffisamment complexes pour fournir des mécanismes de recherche robustes et d'une forte adaptabilité. [74]

II.2.3 Principes généraux :

Les EA sont une classe d'algorithmes d'optimisation par recherche probabiliste basés sur le modèle de l'évolution naturelle. Ils modélisent une population d'individus par des points dans un espace. Un individu est codé dans un génotype composé de gènes correspondant aux valeurs des paramètres du problème à traiter. Le génotype de l'individu

correspond à une solution potentielle au problème posé, le but des EA est d'en trouver la solution optimale.

La plupart des problèmes peuvent être résolus par une méthode de type recherche locale. Cette solution est optimale localement mais peut ne pas correspondre à un optimal global car il est possible qu'il existe une meilleure solution qui n'est pas dans le voisinage de la configuration initiale.

Les EA ont montré leur capacité à éviter la convergence des solutions vers des optima locaux, aussi bien lorsqu'ils sont combinés avec des méthodes de recherche locale comme la rétro-propagation du gradient [75] que lorsqu'ils sont seuls [76].

Quel que soit le type de problème à résoudre, les EA opèrent selon les principes suivants : la population est initialisée de façon dépendante du problème à résoudre (l'environnement), puis évolue de génération en génération à l'aide d'opérateurs de sélection, de recombinaison et de mutation. L'environnement a pour charge d'évaluer les individus en leur attribuant une performance (ou fitness). Cette valeur favorisera la sélection des meilleurs individus, en vue, après reproduction (opérée par la mutation et/ou recombinaison), d'améliorer les performances globales de la population.

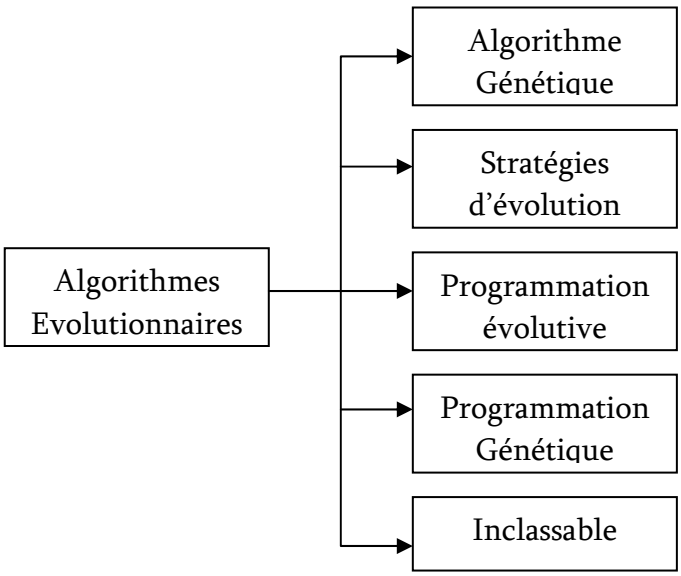


Figure 24 : Les différentes branches des algorithmes évolutionnaires [67]

De ces quatre méthodes classiques ont dérivé différentes techniques mélangeant les méthodes d'évolution des unes et des autres. Impossible à classer dans l'une des quatre familles citées ci-dessus, elles sont néanmoins considérées comme des EA. [67]

III Types d'algorithmes évolutionnaires :

On distingue quatre grandes familles historiques d'algorithme et les différences entre elles ont laissé des traces dans le paysage évolutionnaire actuel, en dépit d'une unification de nombreux concepts.

Algorithmes Génétiques (GA), élaborés par J. Holland, 1975 et D.E. Goldberg, 1989, Michigan, USA. Les GA ont été imaginés comme outils de modélisation de l'adaptation. Ils travaillent dans l'espace des chaînes de bits $[0,1]^n$. Ce sont les plus connus des algorithmes évolutionnaires, et (malheureusement ?) souvent les seules variantes connues des chercheurs des autres disciplines.

Stratégies d'évolution (ES), inventées par I. Rechenberg et H.P. Schwefel, 1965, Berlin. Les ES ont été mises au point par deux jeunes ingénieurs travaillant sur des problèmes numériques. Un énorme progrès a été apporté par les techniques adaptatives d'ajustement des paramètres de mutation, et ce sont sans contestation les meilleurs algorithmes pour les problèmes purement numériques.

Programmation évolutionnaire (EP), inventée par L.J. Fogel, 1966 et D.B. Fogel, 1991, 1995, Californie, USA. Imaginée par Larry Fogel pour la découverte d'automates à états finis pour l'approximation de séries temporelles, EP a rapidement travaillé sur des espaces de recherche très variés.

Programmation génétique (GP), élaborée par J. Koza, 1990, Californie, USA. Apparue initialement comme sous-domaine des GAs, GP est devenu une branche à part entière (conférence, journal, ...). La spécificité de GP est l'espace de recherche, fait d'arbres représentant des programmes complets. GP cherche à atteindre (et réussit souvent!) un des vieux rêves des programmeurs, faire écrire le programme par un autre programme. Et les tendances récentes sont pour GP ... le parallélisme systématique et sur

de grosses grappes de stations. Ainsi, les résultats récents les plus spectaculaires obtenus par Koza l'ont été avec 64 populations de 10000 individus, ou encore avec 1000 populations de 500 individus, utilisant le modèle en îlots sur des grappes Beowulf de stations Alpha ou Pentium sous Linux. [73]

III.1 Les algorithmes génétiques :

III.1.1 De la génétique à l'algorithmique :

Les théories de l'évolution montrent que les êtres vivants évoluent sous l'effet du milieu : les mieux adaptés ont plus de chances de survivre et donc de se reproduire. De génération en génération, les caractéristiques des individus les mieux adaptés on par conséquent plus de chances de se reproduire dans la population. *La génétique*, dont l'objet est d'étudier les mécanismes de l'hérédité, propose quant à elle un modèle permettant d'expliquer la transmission de ces caractéristiques d'une génération à l'autre.

Il existe en effet des entités responsables de la production des caractères héréditaires : ces entités sont appelées gènes et l'ensemble des gènes d'un individu définit son génotype. Par opposition, le phénotype d'un individu correspond à son apparence physique, ou plus précisément à un ensemble de caractéristiques que l'on peut observer, mesurer ou qualifier chez lui (certaines pouvant nécessiter des investigations complexes, comme la mesure de taux de globules rouges dans un système sanguin) ; le phénotype est susceptible de varier au cours du temps par l'effet du milieu dans lequel il évolue, mais ces variations ne peuvent pas être transmises. Un gène est en fait un segment de chromosome, long filament d'ADN (acide désoxyribonucléique), qui est le matériel génétique de toutes les cellules. Les chromosomes sont situés dans le noyau des cellules. Toutes les cellules, sauf les cellules sexuelles, possèdent le même nombre de chromosomes, présents sous forme de paires : dans chaque paire, un des chromosomes vient du père et l'autre de la mère ; les chromosomes d'une même paire sont appelées homologues. Dans les cellules sexuelles, on n'observe que la moitié des chromosomes, chaque paire ne fournissant qu'un seul de ses chromosomes homologues.

Deux mécanismes permettent de fabriquer de nouvelles cellules. Le premier procède par division cellulaires : une cellule duplique son matériel génétique avant de se couper en deux copies conformes. Ou presque conformes : une erreur de reproduction peut se produire, affectant un gène ; il y a alors mutation de ce gène. Ce mécanisme est mis en œuvre dans la *reproduction asexuée*. Le second fait intervenir deux parents pour fabriquer un enfant : c'est celui qui intervient dans la *« reproduction » sexuée*, qu'il convient plutôt d'appeler procréation, puisque l'individu créé n'est pas une reproduction de ses parents. Dans ce mécanisme, les cellules sexuelles transmises par les parents apportent chacune la moitié des chromosomes de l'enfant, ce qui lui redonne bien le nombre voulu de chromosomes.

A défaut de pouvoir créer de nouveaux gènes, la procréation favorise un brassage génétique, puisque le patrimoine génétique de l'enfant est issu à l'égalité des deux parents (alors que la reproduction conforme n'évolue que du fait des mutations qui se produisent aléatoirement et rarement, du moins dans des conditions normales). Deux opérations viennent s'ajouter à ce brassage de la procréation pour accroître encore la diversité qui en résulte : *les mutations*, déjà évoquées, et *le crossing-over* (ou enjambement). Le crossing-over a été introduit pour expliquer comment des gènes situés sur un même chromosome (gènes liés) peuvent ne pas subir le même sort : l'un est transmis et l'autre non. Il arrive en effet, lors de la production des chromosomes qu'un parent va transmettre à l'enfant par l'intermédiaire des cellules sexuelles qu'il fabrique, qu'un échange s'opère, chez ces parent, entre chromosomes homologues : une séquence de l'un se substitue à une séquence de l'autre : cet échange s'appelle crossing-over. Le chromosome qui reçoit l'enfant est alors constitué de morceaux des deux chromosomes homologues détenus par la cellule parentale et provenant donc des grands-parents de cet enfant. On remarquera que deux gènes proches sur un chromosome ont une probabilité plus faible d'être séparés par un crossing-over que deux gènes éloignés.

Les algorithmes génétiques vont s'inspirer, de façon plus ou moins fidèle, de ces mécanismes, sans pour autant en épuiser la complexité. Leurs prémices remontent au début des années soixante, lorsque J. H. Holland étudie les systèmes adaptatifs, dès 1962, à

l'université du Michigan, mais l'expression « algorithmes génétiques » n'apparaît qu'en 1967. Vers 1975, J. H. Holland propose « la théorie des schémas », tentative de formalisation et d'explication théorique du comportement des algorithmes génétiques. Sans doute suscitées par le temps de calcul important que nécessitant les algorithmes génétiques, les études sur le parallélisme de ces méthodes débutent aussi vers cette époque. Comme les algorithmes génétiques sont de gros consommateurs de temps, ce qui ne leur permet pas toujours d'atteindre des solutions convaincantes en des temps acceptables, il faut espérer que ces traitements en parallèle, pour lesquels les algorithmes génétiques semblent particulièrement adaptés, permettront de réduire les temps de calcul à des valeurs raisonnables et de rendre ainsi les algorithmes génétiques plus facilement applicables en pratique. [78]

III.1.2 Définition :

Les algorithmes génétiques sont des algorithmes d'exploration fondés sur les mécanismes de la sélection naturelle et de la génétique. Ils sont basés sur les principes de survie de structures les mieux adaptées et les échanges d'information.

À chaque génération, un nouvel ensemble de créatures artificielles (codées sous forme de chaînes de caractères) est construit à partir des meilleurs éléments de la génération précédente. Bien que reposant fortement sur le hasard (et donc sur un générateur de nombres aléatoires) ces algorithmes ne sont pas purement aléatoires. [72]

III.1.3 Les éléments d'un algorithme génétique :

Les algorithmes génétiques sont inspirés de la génétique classique [79] : on considère une « population » de points répartis dans l'espace. Avant d'expliquer en détail le fonctionnement d'un algorithme génétique, nous allons présenter quelques mots de vocabulaire relatifs à la génétique. Ces mots sont souvent utilisés pour décrire un algorithme génétique :

Génotype ou **Chromosome** :

Le génotype est constitué de gènes situés sur des chromosomes stockés dans le noyau des cellules sous la forme d'une longue chaîne d'acide désoxyribonucléique (ADN).

Dans la nature, l'ADN est un polymère constitué par l'enchaînement de quatre molécules, les nucléotides adénine (A), cytosine (C), guanine (G) et la thymine (T).

On peut donc décrire l'ADN par des chaînes de quatre caractères ACGT. L'ADN constitue l'ensemble des chromosomes, ou le génome d'un individu [72]. C'est une autre façon de dire « **Individu** ».

Gène :

Un chromosome est composé de gènes. Dans le codage binaire, un gène vaut soit 0 soit 1. [02]

Phénotype :

Le phénotype est l'ensemble des protéines et des enzymes qui peuvent être fabriqués à partir de l'ADN. En fait, l'ADN est copiée par un messager (ARN) qui au niveau du ribosome, se traduit en chaînes d'acides aminés formant les protéines et les enzymes.

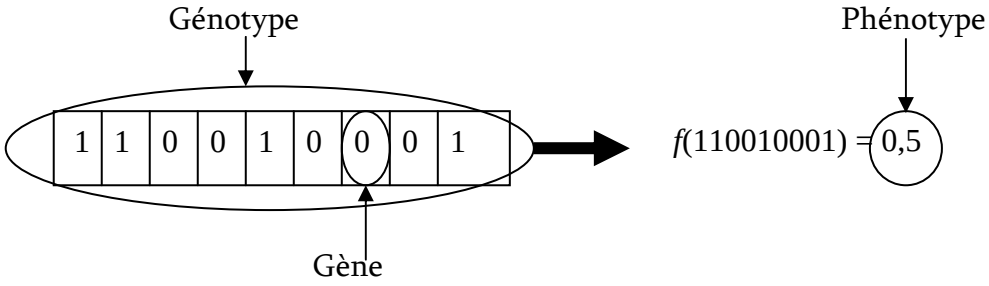


Figure 25 : Le vocabulaire des algorithmes génétiques. [02]

En général, on compte une protéine (un enzyme) par gène. Ce sont les protéines et les enzymes qui dictent la structure et le comportement des cellules qui permettent à un individu de :

- Réaliser des tâches dans son environnement,
- Survivre,
- Reproduire à des taux différents. [72]

Chaque génotype représente une solution potentielle à un problème d'optimisation. La valeur de cette solution potentielle est appelée **le phénotype**. [02]

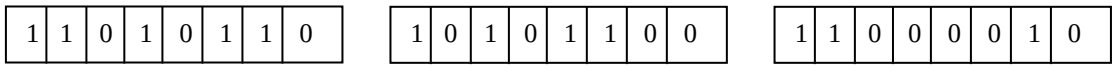
La fitness ou fonction d'évaluation :

Cette fonction est déterminée en fonction du problème à résoudre et du codage choisi pour les chromosomes. À un chromosome particulier, elle attribue une valeur numérique, qui est supposée proportionnelle à l'intérêt de l'individu en tant que solution. L'évaluation d'un individu ne dépendant pas de celle des autres individus, le résultat fournit par la fonction d'évaluation va permettre de sélectionner ou de refuser un individu pour ne garder que les individus ayant le meilleur coût en fonction de la population courante.

Cette méthode permet de s'assurer que les individus performants seront conservés, alors que les individus peu adaptés seront progressivement éliminés de la population. [72]

Conclusion :

Chaque individu va être « codé » à la manière d'un gène. Par exemple, le plus souvent, on fait correspondre une chaîne binaire à l'individu (cette chaîne binaire est une image de la position de l'individu dans l'espace de recherche). Par exemple, trois individus (de 8 bits chacun) sont représentés à la figure 26 :



(a) Individu 1. (b) Individu 2. (c) Individu 3.

Figure 26 : Trois individus. [80]

A chaque individu on associe une efficacité (on appelle aussi cette valeur l'« adaptation »). Cette efficacité va correspondre à la performance d'un individu dans la résolution d'un problème donné. Par exemple, si l'on considère un problème de maximisation d'une fonction, l'efficacité de l'individu croîtra avec sa capacité à maximiser cette fonction.

Si l'on prend l'exemple de la maximisation d'une fonction $f(x)$, on a le tableau 1 :

f(x)	10	20	5
Efficacité	2	4	1
Individu	1	2	3

Tableau 2 : Un exemple de valeurs d'efficacité. [02]

Comme on peut le voir sur cet exemple, l'individu 1 est moins efficace que l'individu 2. En effet, il maximise moins *f*(x) que l'individu 2.

Après avoir déterminé l'efficacité des individus, on opère une reproduction. On copie les individus proportionnellement à leur efficacité. Par exemple, si l'individu 1 est reproduit 2 fois, l'individu 2 sera reproduit quatre fois et l'individu 3 est reproduit une fois.

Ce schéma de fonctionnement est à l'image de la vie réelle. Plus un individu est adapté à son milieu (il se défend mieux contre ses prédateurs, il mange mieux) plus sa capacité à se reproduire sera grande.

Une fois que l'on a déterminé la fonction d'efficacité et on a opéré la reproduction, on effectue des « croisements » entre individus. Plus un individu sera efficace dans la résolution du problème, plus celui-ci se croisera avec un grand nombre d'autres individus. Pour cela, on utilise une roue sur laquelle un individu occupe une place qui est proportionnelle à son efficacité (roulette wheel selection en anglais). Cette roue, dans le cas de l'exemple que l'on a traité ci-dessus, est représentée à la figure 27.

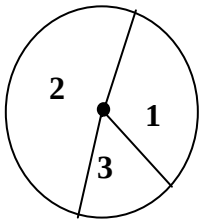


Figure 27 : Roue servant à sélectionner un individu pour le croisement. [80]

Pour notre exemple, on aura le tableau suivant :

Sélection pour le croisement	
« Mâle »	« Femelle »
Individu sélectionné suivant l'efficacité	Individu sélectionné aléatoirement
1	2
1	3
2	2
2	2
2	3
2	1
3	2

Tableau 3 : Le croisement, première étape. [02]

On va maintenant fusionner l'individu de départ avec l'individu associé. Cette opération s'appelle le croisement. Pour cela, on sélectionne aléatoirement une position dans le codage de l'individu. A cette position, on sépare le codage puis on échange les morceaux de droite entre les deux individus. Un exemple de croisement est représenté à la figure 28.

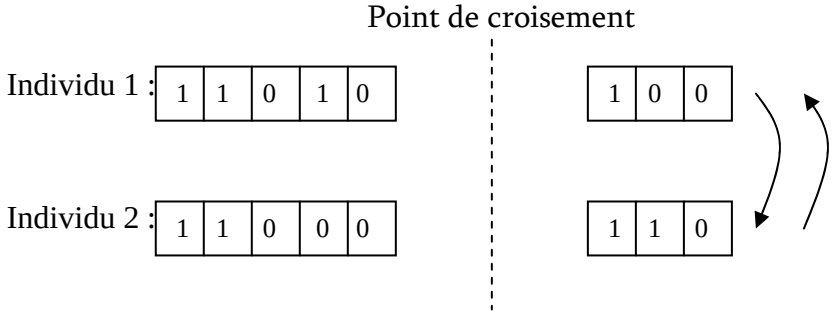


Figure 28 : Le croisement, deuxième étape. [80]

On peut répéter cette opération pour tous les individus du tableau précédent.

Ici, on constate que l'on a obtenu deux nouveaux éléments comme résultat. Notre population s'est diversité. A ce moment, soit on conserve toute la population (mais on supprime les doublons), soit on supprime les éléments les moins performants, de manière à conserver le même nombre d'individus dans la population. D'autres méthode de réduction de population peuvent être mises en œuvre ici (par exemple, on sélectionne au hasard les individus qui vont être supprimés).

Pour finir, on peut procéder à une mutation au niveau des gènes d'un individu. Pour cela, on commence par choisir au hasard un certain nombre d'individus (en général, la probabilité de sélection pour la mutation est faible).

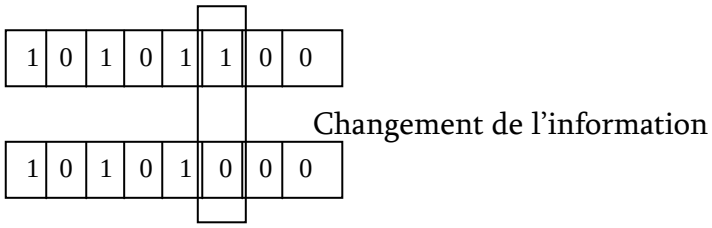


Figure 29 : Opérateur de mutation appliqué à deux chromosomes codés sur 8 bits. [07]

Ensuite, pour chaque individu sélectionné, on choisit au hasard le gène où aura lieu la mutation.

Finalement, à cette position, on change le 0 en 1 et réciproquement.

On recommence ce procédé jusqu'à ce que l'on atteigne un critère d'arrêt (par exemple, un nombre maximal d'itérations).

Le pseudo code d'un algorithme génétique est donné dans cet algorithme :

Algorithme 4 Un algorithme génétique [02].

```
Initialisation de la population
Evaluation de fonctions objectives
Calcul de l'efficacité
For I = 1 to MaxIter
    Sélection aléatoire,
    Sélection proportionnelle à l'efficacité
    Croisement,
    Mutation,
    Evaluation des fonctions objectives,
    Calcul de l'efficacité
End For
```

III.1.4 Les opérateurs d'un algorithme génétique :

Trois opérateurs caractérisent les algorithmes génétiques et rappellent l'origine de ces méthodes, ils vont permettre à la population d'évoluer, par la création d'individus nouveaux construits à l'aide des individus anciens. Plus précisément, on prélève, dans certains individus de la population courante, une partie de leurs caractéristiques en choisissant certaines parties des chromosomes qui les représentent ; puis on recombine ces différentes parties pour constituer les individus de la nouvelle population. La phase de sélection indique dans quelles configurations de la population courante on va prélever des morceaux de chromosomes ; la phase de croisement prélève ces morceaux de chromosomes et les recombine pour former les configurations de la population suivante ; la phase de mutation s'applique à la nouvelle population en changeant éventuellement certains gènes de certains chromosomes obtenus à la fin de la phase de croisement. Une succession des trois opérations de sélection, de croisement et de mutation constitue une génération, et les algorithmes génétiques consistent donc à faire évoluer une population initiale pendant un certain nombre de générations, nombre déterminé par l'utilisateur. Par ailleurs, comme au cours de cette évolution le meilleur individu calculé depuis le début risque de disparaître, il faut le conserver en mémoire afin de pouvoir le restituer à l'utilisateur à la fin du processus. Dans la suite, on remarquera que, sauf dans la phase de sélection, il n'est pas besoin de connaître la fonction à optimiser pour mettre en œuvre un algorithme génétique. [78]

➔ *Le codage :*

Le choix du codage est important est souvent délicat. L'objectif est bien sûr d'abord de pouvoir coder n'importe quelle solution. Mais il est souhaitable, au-delà de cette exigence, d'imaginer soit un codage tel que toute chaîne de caractères représente bien une solution réalisable du problème, soit un codage qui facilite ensuite la conception du croisement de telle sorte que les « enfants » obtenus à partir de la recombinaison de leurs « parents » puissent être associés à des solutions réalisables, au moins pour un grand nombre d'entre eux. [78]

➔ *La sélection :*

La sélection des individus sur lesquels on va appliquer le croisement fait intervenir la fonction *f* à optimiser : la probabilité de choisir l'individu X_i sera d'autant plus grande que $f(X_i)$ sera faible.

Pour cette phase, nous supposons connu : les individus de la population au temps *t*, la fonction d'évaluation. Il va falloir trouver une manière de choisir les individus répondant le mieux à notre problème. Il s'agit donc de privilégier les individus ayant un score au-dessus de la moyenne, et de pénaliser ceux en dessous de cette moyenne [72]. Plusieurs approches sont possibles :

La sélection par roulette (Wheel) :

Les parents sont sélectionnés en fonction de leurs performances. Meilleur est le résultat codé par un chromosome, plus grandes sont ses chances d'être sélectionné. Il faut imaginer une sorte de *roulette de casino* sur laquelle sont placés tous les chromosomes de la population, la place accordée à chacun des chromosomes étant en relation avec sa valeur d'adaptation.

Ensuite, *la bille* est lancée et s'arrête sur un chromosome. Les meilleurs chromosomes peuvent ainsi être tirés plusieurs fois et les plus mauvais ne jamais être sélectionnés.

Cela peut être simulé par l'algorithme suivant :

- On calcule la somme S_1 de toutes les fonctions d'évaluation d'une population.
- On génère un nombre *r* entre *0* et S_1.
- On calcule ensuite une somme S_2 des évaluations en s'arrêtant dès que *r* est dépassé.
- Le dernier chromosome dont la fonction d'évaluation vient d'être ajoutée est sélectionné.

La sélection par roulette rencontre des problèmes lorsque la valeur d'adaptation des chromosomes varie énormément. Si la meilleure fonction d'évaluation d'un chromosome

représente 90% de la roulette alors les autres chromosomes auront très peu de chance d'être sélectionnés et on arriverait à une stagnation de l'évolution. [72]

Sélection par rang :

La sélection par rang trie d'abord la population par fitness. Chaque chromosome se voit associé un rang en fonction de sa position. Le plus mauvais chromosome aura le rang 1, le suivant 2, et ainsi de suite jusqu'au meilleur chromosome qui aura le rang N (pour une population de N chromosomes). La sélection par rang d'un chromosome est la même que par roulette, mais les proportions sont en relation avec le rang plutôt qu'avec la valeur de l'évaluation. Avec cette méthode de sélection, tous les chromosomes ont une chance d'être sélectionnés. Cependant, elle conduit à une convergence plus lente vers la bonne solution. Ceci est dû au fait que les meilleurs chromosomes ne diffèrent pas énormément des plus mauvais. [72]

Sélection Steady-State

L'idée principale est qu'une grande partie de la population puisse survivre à la prochaine génération. L'algorithme génétique marche alors de la manière suivante.

- A chaque génération sont sélectionnés quelques chromosomes (parmi ceux qui ont le meilleur coût) pour créer des chromosomes fils.
- Les chromosomes les plus mauvais sont retirés et remplacés par les nouveaux.
- Le reste de la population survie à la nouvelle génération. [72]

➔ *Le croisement :*

L'opérateur de croisement permet la création de nouveaux individus selon un processus fort simple. Il permet donc l'échange d'information entre les chromosomes (individus) par le biais de leur combinaison. La population qui résulte de la sélection est divisée en deux sous population de taille et chaque couple formé par un membre provenant de chaque sous population participe à un croisement avec une probabilité donnée (la probabilité de croisement, souvent supérieur à 60%). Si le croisement a lieu, sa localisation entre la position 1 et la position l, dans le cas du codage binaire, est tirée selon

une loi uniforme et les deux individus échangent leurs gènes des deux cotés de cette localisation.

Deux nouveaux individus ont été créés par ce biais. Toutefois, un individu sélectionné lors de la reproduction ne subit pas nécessairement l'action d'un croisement. Ce dernier ne s'effectue qu'avec une certaine probabilité. Plus cette probabilité est élevée et plus la population subira de changement. On peut aussi imaginer que le croisement ait lieu entre plusieurs zones différentes des chaînes (croisement multiple). [80]

➔ *La mutation :*

Le rôle de cet opérateur est de modifier aléatoirement, avec une certaine probabilité, la valeur d'un composant de l'individu. Dans le cas du codage binaire, chaque bit $a_i \in \{01\}$, est remplacé selon une probabilité p_m par son complémentaire : $\tilde{a}_i = 1 - a_i$.

La mutation est traditionnellement considérée comme un opérateur intervenant à la marge dans la mesure où sa probabilité est en général fixée assez faible (de l'ordre de 1%). Mais elle confère aux algorithmes génétiques une propriété très importante : tous les points de l'espace de recherche peuvent être atteints. Cet opérateur est donc d'une grande importance et il est loin d'être marginal. Il a de fait un double rôle : celui d'effectuer une recherche locale et/ou de sortir d'une trappe (optima locaux). [80]

III.1.5 L'optimisation multiobjectifs et les algorithmes génétiques :

Adapter un algorithme génétique à un problème d'optimisation combinatoire revient à spécifier les cinq composantes suivantes:

1. une représentation génétique des solutions du problème,

2. une façon de créer une population initiale de solutions, une fonction d'évaluation jouant le rôle de l'environnement, distinguant les solutions en termes de leur adaptation,

4. des opérateurs génétiques qui modifient la composition des enfants pendant la reproduction,

5. des valeurs pour les divers paramètres que l'algorithme génétique utilise (taille de la population, probabilités d'application des opérateurs génétiques, etc.) [86]

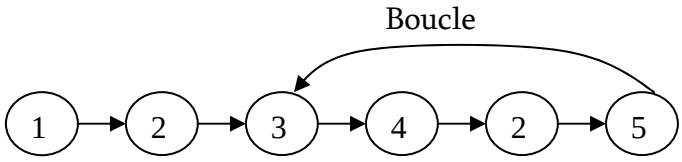


Figure 30 : Le fonctionnement d'un algorithme génétique multiobjectifs. [80]

Les algorithmes génétiques sont très bien adaptés au traitement d'un problème d'optimisation multiobjectifs. En témoigne le nombre important d'articles qui ont été publiés sur ce sujet. De plus, ce domaine est très dynamique et ne cesse de se développer.

1 Initialisation de la population.

2 Evaluation de l'efficacité des individus de la population.

3 Croisement.

4 Mutation.

5 Sélection.

Nous allons maintenant nous intéresser aux diverses méthodes qui ont été mises en œuvre pour le traitement de problèmes d'optimisation multiobjectifs. Ces méthodes se décomposeront en deux familles :

- ➔ Une première famille comportera les méthodes dites « non agrégatives ». Dans ces méthodes, il n'y a pas fusion des différentes fonctions objectives pour se ramener à un problème d'optimisation mon-objectif.
- ➔ La deuxième famille comportera les méthodes « agrégatives ». Le but de ces méthodes est de se ramener à un problème d'optimisation mono-bjectif en utilisant une fonction objective équivalente.

III.1.6 Les méthodes non agrégatives :

La méthode Vector Evaluated Genetic Algorithm (VEGA) :

Cette méthode permet de traiter un problème d'optimisation multiobjectifs sans avoir à agréger les fonctions objectives en une seule fonction [38] [81].

L'algorithme VEGA considère une population de *N* individu. Ces *N* individus sont répartis en *k* groupes (*k* étant le nombre de fonctions objectives de notre problème) de *N/k* individus (avec *N* multiple de *k*). À chaque groupe, on associe une fonction objective permet de déterminer l'efficacité d'un individu au sein du groupe. Ensuite, les individus sont mélangés et les croisements sont opérés en tenant compte de l'efficacité de chaque individu.

La figure 30 représente les différentes séquences de fonctionnement de la méthode. Sur cette figure, on a représenté les différents types de populations que l'on traite au cours du déroulement de la méthode VEGA (soit un ensemble d'individus, soient des groupes d'individus).

Etape 1 : Itération *i*: Initialisation d'une population de taille *N*.

Etape 2 : Création de *k* groupes (sous population).

Etape 3 : Calcul des efficacités.

Mélange des individus.

Etape 4 : On applique l'algorithme génétique classique (croisement, mutation et sélection). Puis on passe à l'itération suivante (i + 1).

III.1.7 Les méthodes agrégatives :

La méthode Multiple Objective Genetic Algorithm (MOGA) :

Cette méthode est présentée dans [81]. Elle utilise la relation de dominance au sens de Pareto pour déterminer l'efficacité d'un individu. Ici, le « rang » d'un individu (numéro d'ordre qui permet de classer un individu par rapport aux autres) est donné par le nombre d'individus qui dominent l'individu considéré.

Par exemple, si l'on considère un individu x_i à la génération *t* qui est dominé par $p_i^{(t)}$ individus, le rang de l'individu considéré est donné par :

$$rang(\chi_i, t) = 1 + p_i^{(t)}$$

A tous les individus non dominés on affecte le rang 1. Les individus dominés se voient donc affectés à un rang important (une forte pénalité donc).

Pour le calcul de l'efficacité, on peut suivre les étapes suivantes :

1. Classer les individus en fonction de leur rang.

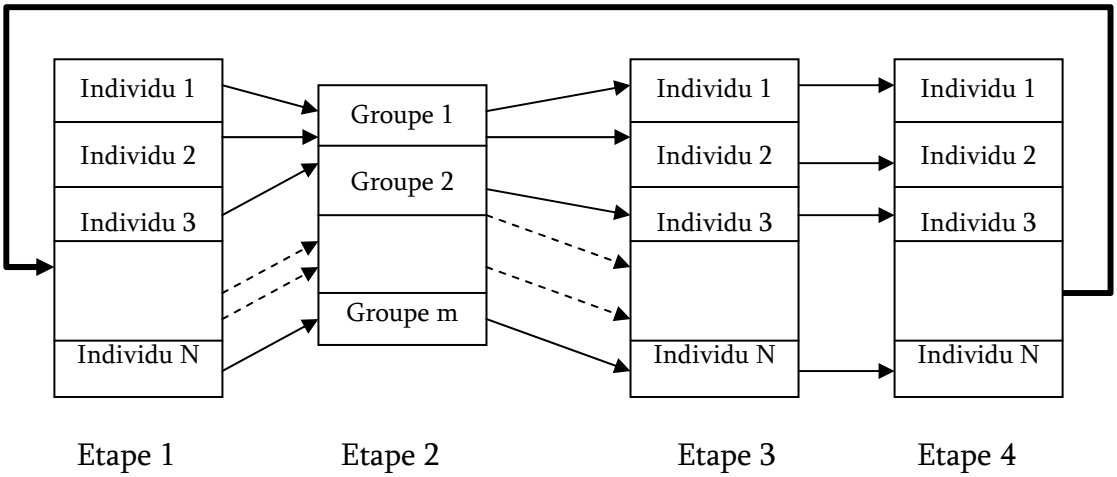


Figure 31 : Principe de l'algorithme VEGA [80]

2. Affecter une efficacité à un individu en interpolant à partir du meilleur (rang 1) jusqu'au plus mauvais (rang n), en utilisant une fonction *f* (rang). Cette fonction est, le plus souvent, linéaire (voir la figure 32).

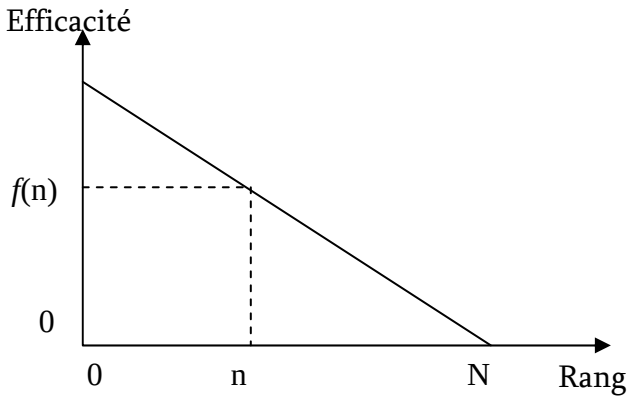


Figure 32 : Un exemple de fonction d'efficacité [80]

Le pseudo code de cette méthode est représenté dans l'algorithme 5.

Algorithme 5 Algorithme MOGA [02].

Initialisation de la population

Evaluation de fonctions objectives

Assignation d'un rang basé sur la dominance
Assignation d'une efficacité à partir du rang
For I = 1 to G
Sélection aléatoire proportionnelle à l'efficacité,
Croisement,
Mutation,
Evaluation des fonctions objectives,
Assignation d'un rang basé sur la dominance
Assignation d'une efficacité à partir du rang
End For

La méthode Non Dominated Sorting Genetic Algorithm (NSGA) :

Cette méthode reprend les grandes lignes de la méthode MOGA précédemment décrite. La différence principale intervient lors du calcul de l'efficacité d'un individu [82].

Cette méthode est basée sur une classification en plusieurs niveaux des individus. Dans un premier temps, avant de procéder à la sélection, on affecte à chaque individu de la population un rang (en utilisant le rang de Pareto). Tous les individus non dominés de même rang sont classés dans une catégorie considérée.

On veut maintenant que les individus de la catégorie considérée se répartissent de manière uniforme au sein de celle-ci. On veut donc une bonne représentation, ou encore on veut qu'il y ait une diversité de solutions.

Pour maintenir la diversité de la population, ces individus classés se voient affectés d'une nouvelle valeur d'efficacité. Pour cela, on utilise la formule suivante :

$$m_i = \sum_{j=1}^{k} Sh(d(i,j))$$

$$Sh(d(i,j)) = \begin{cases} 1 - \left(\dfrac{d(i,j)}{\sigma_{share}} \right)^2 & si\ d(i,j) < \sigma_{share} \\ 0 & sinon \end{cases}$$

Ici, *k* désigne le nombre d'individus dans la catégorie considérée et *d(i,j)* est la distance entre l'individu *i* et l'individu *j*. Il est toujours possible d'utiliser la distance

classique à la place de cette dernière. σ_{share} est la distance d'influence. Comme on peut le voir dans l'expression ci-dessus, tous les individus qui sont suffisamment proches (dont la distance *d(i,j)* est inférieure à σ_{share}) seront pris en compte dans le calcul de m_i. Les autres seront ignorés.

La valeur d'efficacité de l'individu *i* au sein de la catégorie considérée sera alors :

$$f_i = \frac{F}{m_i}$$

Où *F* est la valeur d'efficacité affectée à la catégorie à laquelle appartient l'individu. La figure 10 montre les différents paramètres qui interviennent dans le calcul de l'efficacité d'un individu.

Comme on voit à la figure 33 le paramètre σ_{share} permet de définir une zone d'influence pour le calcul de l'efficacité d'un individu.

Ensuite, on ignore les individus de ce groupe. Le processus reprend alors avec les individus restants, où l'on opère une nouvelle classification, une nouvelle catégorisation et une nouvelle opération de partage. Ce processus est répété jusqu'à ce que tous les individus aient été traités.

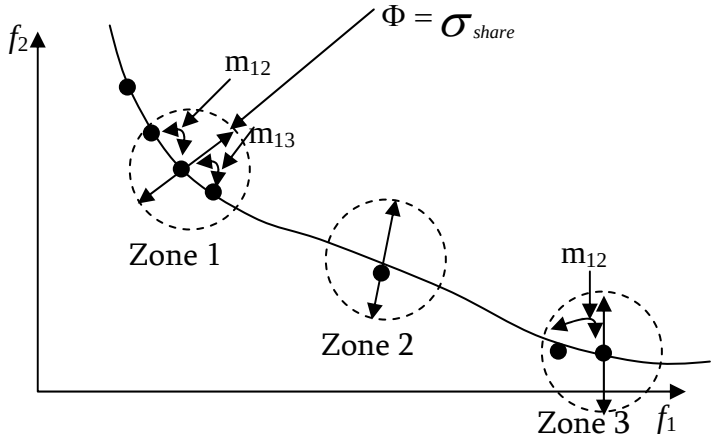


Figure 33 : Le calcul d'efficacité [80].

Comme les individus qui ont un rang de Pareto de valeur 1 ont une meilleure efficacité, ils sont reproduits en plus nombre que les autres. Cette propriété permet

d'obtenir une convergence plus rapide vers la surface de compromis. Le partage, lui, permet de maintenir une répartition « uniforme » sur la surface de compromis.

Le pseudo-code de cette méthode est donné dans l'algorithme 6.

Algorithme 6 Algorithme NSGA [02].

```
Initialisation de la population
Evaluation des fonctions objectives
Assignation d'un rang basée sur le rang de dominance sur chaque surface de compromis
Calcule du compte des voisins
Assignation d'une efficacité partagée
For i=1 to G
    Sélection aléatoire proportionnelle à l'efficacité
    Croisement mutation
    Evaluation des fonctions objectives
    Assignation d'un rang basée sur le rang de dominance sur chaque surface de compromis
End for
```

La méthode Niched Pareto Genetic Algoithm (N.P.G.A.)

Cette méthode reprend les grandes lignes de la méthode N.S.G.A précédemment décrite. La différence principale intervient lors du processus de sélection [83]

Dans l'algorithme génétique classique, la méthode de sélection entre deux individus utilise la roue de sélection telle qu'elle a été décrite précédemment. Dans cette méthode, seule la manière de sélectionner les individus change. Au lieu de comparer deux individus, on utilise un groupe de I individus (typiquement dix). Si les individus sont soit dominés, soit non dominés, on utilise le partage de la valeur d'efficacité.

Le pseudo-code de la fonction sélection pour un problème multiobjectifs ou tous les objectifs doivent être maximisés est représenté dans l'algorithme 7. Son implémentation dans un algorithme génétique est représentée dans l'algorithme 8.

Algorithme 7 Détail de la fonction de sélection [02].

```
Fonction sélection
   Mélange (random_pop_index)
   Candidat_1 = random_pop_index[1]
   Candidat_2 = random_pop_index[2]
   Candidat_1_dominé = false
   Candidat_2_dominé = false
   /* sélectionne t_dom individus au hasard dans S */
   begin
   Indiv_comp=random_pop_index[Ensemble_index_comp]
   If S[Indiv_comp] domine S[Candidat_1]
      Then Candidat_1_domine = true
   If S[Indiv_comp] domine S[Candidat_2]
      Then Candidat_2_domine = true
   End
If (Candidat_1_dominé AND NOT Candidat82_dominé)
   Then return Candidat_2
   Else if (NOT Candidat_1_dominé AND Candidat_2_dominé)
   Then return Candidat_1
   Else
   Do partage
 End
```

La fonction de sélection choisit un individu de la population S.

Les valeur de t_{dom} et doivent être fournies par l'utilisateur.

Algorithme 8 Algorithme N.P.G.A. [02]

```
Initialisation de la population
Evaluation des fonctions objectives
For i=1 to G
```

Sélection par tournoi entre deux individus

(Utilisation de t_{dom} *)*

Seul le candidat 1 est dominé : Sélectionner le candidat 2

Seul le candidat 2 est dominé : Sélectionner le candidat 1

Les deux candidats sont dominés ou ne pas dominés :

Effectuer un partage d'efficacité

Sélectionner le candidat avec le plus petit nombre de Voisins (utilisation de σ_{share} *)*

Croisement

Evaluation des fonctions objectives

End for

Mais seulement sur une partie à chaque exécution, elle est très rapide. Cependant, elle requiert de la part de l'utilisateur les paramètres t_{dom} et σ_{share} , dont dépendant les performances de l'algorithme. Dans [84], on trouvera des techniques permettant de régler « automatiquement » ces paramètres.

La méthode Weighted Average Ranking Genetic Algorithm (W.A.R.G.A)

Cette méthode s'inspire de la méthode MOGA. La différence principale dans la manière dont la relation de dominance est établie entre deux solutions.

Dans cette méthode, le rang de dominance est calculé de la manière décrite dans l'algorithme 9, nommée W.A.R. (Weighted Average Ranking) :

Algorithme 9 la relation W.A.R.[02]

A solution à comparer (vecteur de dimension n).

S ensemble de solution (S ne contient pas A).

Repeat

i = 1

Soit A, la variable i du point A.

N_i *= nombre de point de S meilleurs que* A_i *du point de vue de la variable i.*

i= i+1

until i>n

Le rang de A vaut N= $N = \sum_{i=1}^{n} N$

Ensuite, l'efficacité d'un individu sera calculée de la même manière que pour la méthode MOGA.

Prenons un exemple. A la figure 34 (f_1 et f_2 sont à minimiser). On a :

-Vis-à-vis de f_1, A est dominé 4 points.

Vis-à-vis de f_2 , A est dominé 0 points.

Donc, N (A) = 4+0 = 4

-Vis-à-vis de f_1, B est dominé 1 points.

Vis-à-vis de f_2, B est dominé 3 points.

Donc, N (B) = 1+3 = 4

Le Strength Pareto Evolutionary Algorithm (SPEA)

La méthode SPEA qui est implémentée par Zitzler et Thiele [90] est l'illustration même d'un algorithme évolutionnaire élitiste. Pour réaliser cet élitisme, SPEA maintient

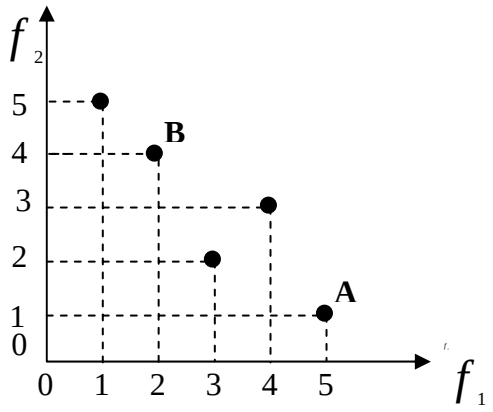


Figure 34 : La relation WAR [80].

une archive externe contenant le meilleur front de compromis rencontré durant la recherche.

Fonctionnement général

La première étape consiste à créer une population initiale (constituée par exemple d'individus générés aléatoirement). L'archive externe, au départ initialisée à l'ensemble vide, est mise à jour régulièrement en fonction des individus non dominés de la population.

A chaque itération de l'algorithme, on retrouve les étapes classiques sélection, croisement, mutation. Puis les nouveaux individus non dominés découverts viennent s'ajouter à l'archive, et les individus de l'archive dominés par le nouvel arrivant sont supprimés. Si l'archive vient à excéder une certaine taille (fixée au départ), alors une phase de clustériser (Clustering) est appliquée dans le but de garder les meilleurs représentants.

La figure 35 (extraite de la thèse de Zitzler [91]) et l'algorithme 7 illustrent le schéma général de fonctionnement de l'algorithme.

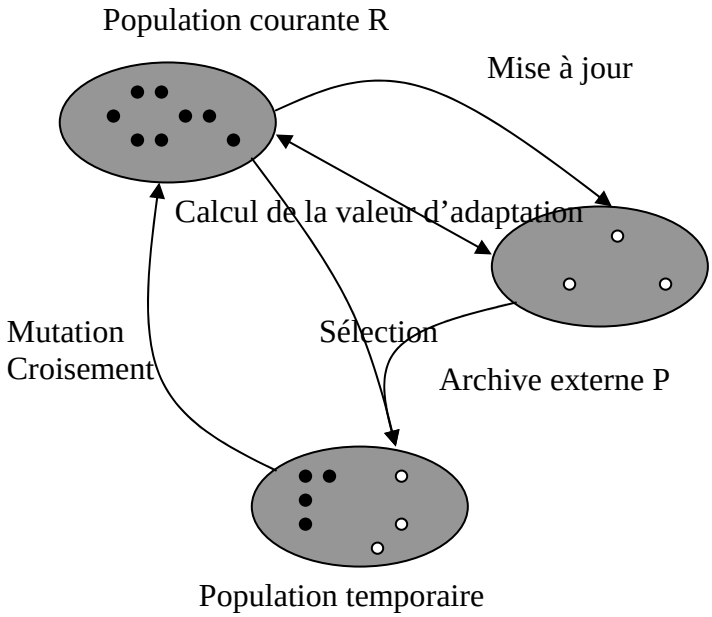


Figure 35 : Fonctionnement général de l'algorithme SPEA. [91]

III.2 La programmation évolutionnaire :

La programmation évolutionnaire, originellement conçue par Lawrence J. Fogel en 1960, est une stratégie stochastique d'optimisation similaire aux algorithmes génétiques, mais au lieu de l'emphase des places dans la liaison comportementale entre les parents et leurs progénitures, plutôt que la recherche d'émuler les opérateurs génétiques spéciaux

comme il est observé dans la nature. La programmation évolutionnaire est similaire aux stratégies d'évolution, quoique les deux approches soient développées indépendamment.

Comme les deux les stratégies évolutionnaires et les algorithmes génétiques, la programmation évolutionnaire est une méthode utile pour l'optimisation quand aux autres techniques comme la descente de gradient ou descente directe, ne sont pas possible. Les fonctions d'optimisation combinatoires et les valeurs réelles sont bien adaptées pour la programmation évolutionnaire. [74]

Le livre de 1966, « Artificial Intelligence Through Simulated Evolution » par Fogel, Owens et Walsh est la publication borne des applications de programmation évolutionnaire, quoique quelques autres articles ont apparu plutôt. Dans le livre, un automate d'état fini fut évolué pour prédire les symboles de chaînes de caractère générées à partir du processus de Markov et les séries de temps non-stationnaires. Comme la prédiction évolutionnaire est motivée par une reconnaissance dont la prédiction est une clé de comportement intelligent. [74]

III.2.1 Le processus :

Pour la programmation évolutionnaire, comme les AGs, il y a une hypothèse sous-jacente dont la fitness peut être caractérisée aux termes des variables, et ici existe une solution optimale (ou multiple comme les optima) aux termes de ces variables. Par exemple, si un qui essaie de trouver le chemin le plus court dans le problème de voyageur de commerce, chaque variable doit être une solution. La longueur du chemin peut être exprimé sous forme d'un nombre, qui doit servir la fitness de solution. La fitness pour ce problème peut être caractérisée comme une hypersurface proportionnelle pour les longueurs du chemin dans un espace des chemins possible. Le but peut doit être de trouver le chemin globalement le plus court dans cet espace, ou plus pratiquement, de trouver les tours les plus courtes rapidement.

La méthode de programmation évolutionnaire de base comporte 3 étapes (répéter tant que la limite d'itération n'est pas dépassée ou une solution adéquate n'est pas obtenue) :

- ✓ Choisir une population initiale de solutions aléatoires. Le nombre de solutions dans la population est très approprié à la vitesse de l'optimisation, mais pas de réponses définitives sont valables de combien de solutions sont appropriées et combien de solutions sont juste dilapidatrice.
- ✓ Chaque solution est reproduite vers une nouvelle population. Chacune de ces solutions filles sont mutées suivant une distribution de types de mutation, rangeant du mineur vers l'extrême avec un continuum entre les types de mutation. La sévérité de mutation est jugée dans la base des échanges fonctionnels imposés sur les parents.
- ✓ Chaque solution fille est évaluée par sa fitness. Typiquement, un tournoi stochastique est conservé pour déterminer N solutions d'être retenues pour la population de solutions, quoique ce soit représenté occasionnellement d'une manière déterministe. Il n'y a pas d'exigence que la taille de la population être conservée constante, toutefois, ni seulement qu'un fils singulier être généré à partir de chaque parent. [74]

III.3 Les stratégies d'évolution :

En 1963 deux étudiants à l'université technique de Berlin (TUB) rencontrèrent et furent plutôt collaborer des expérimentations qui utilisent le tunnel de vent de l'institue de Flow Engineering. Durant la recherche des formes optimales des corps dans le flux, qui fut ensuite une matière de l'expérimentation intuitive et laborieuse, l'idée était conçue du procédé stratégique. Néanmoins, des tentatives avec la coordination et les stratégies de gradient simple étaient infructueuses. Ensuite un des étudiants, Ingo Rechenberg, aujourd'hui Professeur de Bionique et l'ingénierie évolutionnaire, rencontre l'idée d'essayer les échanges aléatoires dans les paramètres définissant la forme, suivant l'exemple de mutations naturelles. La stratégie d'évolution était née. Un troisième étudiant, Peter Bienert, les joignit et commença la construction d'un automate expérimentateur, qui doit travailler suivant les règles simples de mutation et de sélection. Le deuxième étudiant, Hans-Paul Schwefel, posa pour tester l'efficacité des nouvelles méthodes avec l'aide de l'ordinateur Zuse Z23. [74]

Les stratégies d'évolution sont des stratégies d'optimisation et similaire aux AGs, ils sont basés sur le mécanisme des génétiques naturelles, qui permettent le développement des espèces. Ils sont fondés sur trois principes de bases :

1/ recombinaison ;

2/ la sélection naturelle ;

3/ la diversité par variation.

Au contraire aux autres algorithmes naturels, les stratégies d'évolution utilisent l'opérateur de mutation comme un opérateur essentiel, où la fréquence d'application dépend sur certains paramètres, assumant les différentes valeurs durant le processus de recherche.

La recombinaison a rapport mineur et peut disparaître. L'opérateur de mutation est un opérateur de diversification important allouant des petites perturbations sur la solution courante.

Encore, les ES ont quelques autres options comparées aux autres traditionnels algorithmes naturels, comme un nombre libre de parents entraîné dans la reproduction. [85]

IV L'optimisation multiobjectifs évolutionnaire :

Dans cette section, on va parler sur les techniques ajoutées au principe de fonctionnement des algorithmes évolutionnaires pour mieux avoir des résultats réalisables, ces techniques essaient de garder les meilleurs solutions dans toutes les populations pour ne les pas perdre (l'élitisme), ainsi que de mieux maintenir la diversité de la population pour garder la convergence de l'algorithme et l'exploration de tout l'espace de recherche.

IV.1 Maintenir la diversité !

La diversité est une notion déjà importante lors de l'optimisation de problèmes à un objectif, elle devient prépondérante lorsque l'on traite de problèmes multiobjectifs.

Maintenir un certain degré de diversité dans la population d'un algorithme évolutionnaire consiste à éviter que la population ne converge prématurément vers une petite zone de l'espace de recherche ou de l'espace des objectifs. En effet, s'il n'existe pas de mécanisme de contrôle de la diversité, les opérations de sélection vont privilégier trop vite certains individus meilleurs à cette étape de la recherche. Cette convergence prématurée a comme effet de limiter la recherche à un sous-ensemble plus restreint de l'espace de recherche, qui peut ne contenir aucune solution optimale. Dans le cas de problèmes multiobjectifs, converger prématurément rend impossible la découverte de l'intégralité des frontières de Pareto.

En effet, les individus de la population se focaliseront sur une partie des frontières de Pareto, et ne se répartiront pas sur la totalité de ces frontières. Pour remédier à ce problème, de nombreuses techniques ont été développées. Celles-ci nuent sur la pression de sélection, afin de privilégier certains individus, permettant d'obtenir une population plus diversifiée. Cependant, ces techniques ajoutent un coût calculatoire non négligeable pour l'algorithme, elles doivent donc être choisies avec soin.

Quelques techniques utilisées pour maintenir la diversité sont :

IV.1.1 Le sharing :

Le sharing consiste à modifier la valeur de coût d'un individu (calculée uniquement à partir de la fonction objective du problème). C'est cette nouvelle valeur qui sera utilisée comme valeur d'adaptation par l'opérateur de sélection. Cette technique, introduite dans [87], est largement utilisée aujourd'hui.

Pour éviter qu'un trop grand nombre d'individus ne se concentrent autour d'un même point, il faut pénaliser la valeur d'adaptation en fonction du nombre d'individus au voisinage du regroupement : plus les individus sont regroupés, plus leur valeur d'adaptation est faible, et des individus proches les uns des autres doivent partager leur valeur d'adaptation. Dans la pratique, on estime ce taux de concentration en ouvrant un domaine autour d'un individu, puis on calcule les distances entre les individus contenus dans ce domaine.

Pour déterminer les bornes du domaine ouvert autour de l'individu choisi, on définit une distance maximale, appelée σ_{share}, au delà de laquelle les individus ne seront plus considérés comme faisant parti du domaine ouvert. La distance séparant deux individus *i* et *j* est calculée grâce à la fonction *d(i;j)*. La valeur d'adaptation *F(i)* d'un individu $i \in P$ (population) est égale à son coût *F'(i)* divisé par sa valeur de niche :

$$F(i) = \frac{F'(i)}{\sum_{j \in p} Sh(d(i,j))}$$

Où la fonction Sh est définie comme suit :

$$Sh(d(i,j)) = \begin{cases} 1 - \left(\dfrac{d(i,j)}{\sigma_{share}} \right)^2 & si\ d(i,j) < \sigma_{share} \\ 0 & sinon \end{cases}$$

La fonction *d(i;j)* de calcule de distance peut être définie dans l'espace de recherche, par exemple à l'aide d'une distance de Hamming, ou dans l'espace objectif. Ce choix dépend souvent du problème, car le maintient de la diversité dans l'espace objectif, bien qu'il soit souvent plus simple à réaliser, n'assure pas forcément le maintient de la diversité dans l'espace de recherche. [07]

IV.1.2 La réinitialisation :

La réinitialisation est une technique largement utilisée par toutes les métaheuristiques, les algorithmes évolutionnaires n'échappant pas à la règle. Lors d'approches par population, elle consiste à réinitialiser un certain nombre d'individus de la population, par exemple de manière aléatoire. En introduisant de manière régulière certains individus générés aléatoirement, de nouvelles zones de l'espace de recherche, peut-être inexplorées jusqu'ici, peuvent être découvertes. [07]

IV.1.3 Le crowding :

L'approche par « crowding » consiste à déterminer un représentant par niche découverte. A la différence du « sharing », où tous les individus sont susceptibles d'être sélectionnés et de participer aux phases de croisement, mutation et sélection, avec le « crowding », seuls les représentants participeront aux différentes étapes de l'algorithme.

Le « crowding » fut introduit par De Jong [88] et fut adapté notamment au travers des travaux de Blickle [89].

IV.2 L'élitisme :

L'idée générale qui ressort de ces réflexions sur l'élitisme est qu'il est important de ne pas oublier de conserver, d'une manière ou d'une autre, les bonnes solutions obtenues génération après génération.

En optimisation mono-objectif, on a pour habitude de sauvegardes la meilleure solution obtenue au cours du processus d'optimisation. En optimisation multiobjectifs, un processus similaire consiste à placer dans une archive tous les individus non dominés obtenue en cours d'optimisation. Cette archive représente à tout moment la meilleure approximation de la surface de compromis que l'on a obtenue jusqu'ici. De plus, comme cette archive ne sert qu'à sauvegarder les individus non dominés, elle ne modifie pas le comportement de l'algorithme génétique. Une des premières implémentations de ce mécanisme dans un algorithme génétique est présentée dans [88].

Conserver ces solutions pour les générations futures permet d'améliorer les performances des algorithmes sur certains problèmes. En effet, dans ses expérimentations, De Jong a observé qu'un algorithme génétique élitiste améliorait significativement les résultats sur des fonctions unimodales. D'un autre côté, l'élitisme peut causer une convergence prématurée sur des fonctions multimodales.

Réaliser un algorithme élitiste dans le cadre des problèmes multiobjectifs est plus difficile que pour les problèmes à un objectif. En effet, la meilleure solution n'est plus un unique individu, mais tout un ensemble dont la taille peut aller jusqu'à dépasser la taille maximale de la population. Deux adaptations du mécanisme élitiste sont considérées : la première approche regroupe les algorithmes, fondés sur les travaux de De Jong, qui conservent pour les générations futures les k meilleurs individus [92].

Les approches récentes (parmi [93]) tendent à utiliser une population externe d'individus dans laquelle est stocké le meilleur ensemble des points non dominés découverts jusqu'ici. Cet ensemble est mis à jour continuellement pendant la recherche, et

les individus stockés continus à pouvoir être choisis par l'opérateur de sélection. Ils peuvent ainsi se reproduire et transmettre leurs caractéristiques aux générations suivantes.

Actuellement, les algorithmes élitistes obtiennent de meilleurs résultats sur un grand nombre de problèmes multiobjectifs [51; 24].

V Conclusion :

Voilà dans ce chapitre, nous avons présenté un état de l'art sur les algorithmes évolutionnaires d'une manière générale, ensuite nous avons présenté le fonctionnement d'un algorithme évolutionnaire dans l'un de ses types appelé les algorithmes génétiques. Nous avons présenté dans ce chapitre comment les algorithmes évolutionnaires exploitent le principe de la génétique de Darwin pour résoudre les problèmes NP-Complet d'une manière générale et les problèmes d'optimisation multi-objectifs d'une manière spéciale. Dans le chapitre suivant, nous allons parler sur la programmation génétique l'un des algorithmes évolutionnaires en détails puisque nous allons l'utiliser dans notre recherche.

Chapitre III

La Programmation Génétique

I. Introduction :

Dans le chapitre précédent, nous avons décrit la notion des méthodes évolutionnaires pour résoudre les problèmes d'optimisation multiobjectifs, et spécialement, nous avons détaillé tout ce qui concerne les algorithmes génétiques, ainsi nous avons parlé des stratégies d'évolution et la programmation évolutionnaire. Aussi nous avons su que parmi les méthodes évolutionnaires, on trouve la programmation génétique, la méthode responsable de faire évoluer les programmes machines pour donner un programme optimale capable de résoudre un problème quelconque.

La plupart des algorithmes évolutionnaires sont consacrés d'évoluer des solutions dans des problèmes paramétrés. En contraste, la programmation génétique (GP) est une communauté de recherche évolutionnaire qui ont comme défi : évoluer les programmes d'ordinateurs actuels pour résoudre quelques tâches. [103]

Puisque ce mémoire est intitulé L'optimisation multiobjectifs par la programmation génétique, nous allons consacrer ce chapitre pour décrire en détail cette méthode.

II. Historique :

Le défi :

« Est qu'on peut laisser les ordinateurs apprendre comment résoudre les problèmes sans être explicitement programmés ? Dans un autre terme, comment les ordinateurs peuvent être réalisés pour réaliser ce qui est nécessaire de faire, sans être dites comment le faire ? » Arthur Samuel (1956).

Pour le succès :

« L'objectif [est] … d'avoir des machines pour présenter (exhiber) le comportement, comme s'il est donné par les humains, doit être assumé pour impliquer l'utilisation de l'intelligence. » Arthur Samuel (1983). [105]

III. Définition :

La programmation génétique [94] est une technique qui permet aux ordinateurs de résoudre les problèmes sans être explicitement programmés. Cet algorithme travaille en utilisant le même principe des algorithmes génétiques pour automatiquement générer des programmes d'ordinateur. [95]

Dans la programmation génétique (PG) les individus dans la population sont des programmes d'ordinateur. Pour détendre le processus de création de nouveaux programmes à partir de deux programmes parents, les programmes sont écrits sous forme d'arbres. Les nouveaux programmes sont produis en supprimant des branches à partir d'un arbre et l'insérer à un autre. Ce processus simple, connu croisement, assure que le nouveau programme est aussi un arbre et syntaxiquement valide. Cette programmation génétique est fondamentalement différente de lignes de Fortran ou code machine.

La programmation génétique est une méthode systématique pour laisser les ordinateurs automatiquement résoudre un problème débutant par une instruction de haut niveau de quoi il est en besoin pour le faire. Spécialement, la programmation génétique transforme itérativement une population de programmes d'ordinateur vers une nouvelle génération de programmes tout en appliquant les opérateurs génétiques inspirants de la nature. [113]

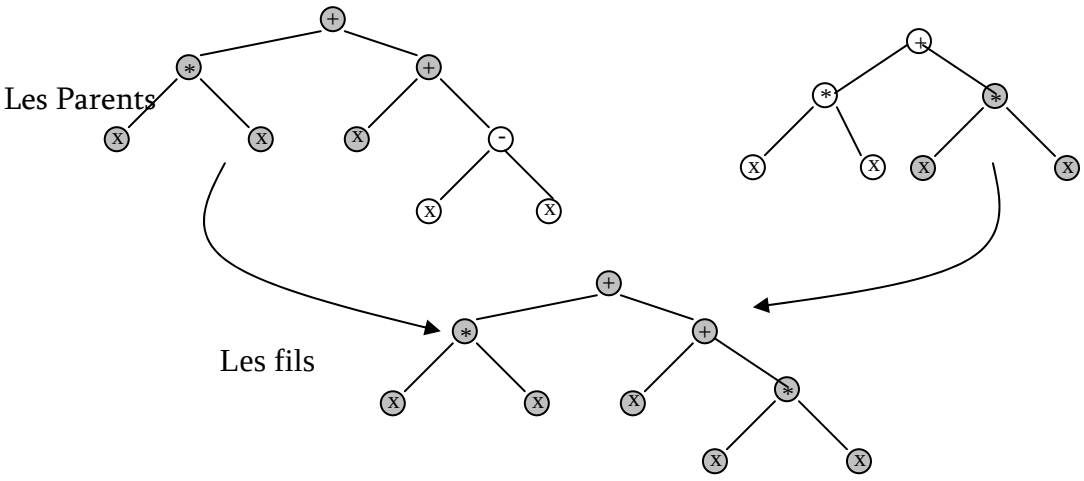


Figure 36 : Le crossover de la programmation génétique :
$x^2 + (x+(x-x))$ croisé avec $2x^2$ pour produire $2x^2 + x$ [95]

Les opérateurs utilisés dans le processus de la programmation génétique sont les mêmes utilisés dans les algorithmes génétiques avec le même principe : le croisement, la mutation, la reproduction, la duplication génétique et la sélection. La seule différence est que les chromosomes ici sont les programmes d'ordinateurs.

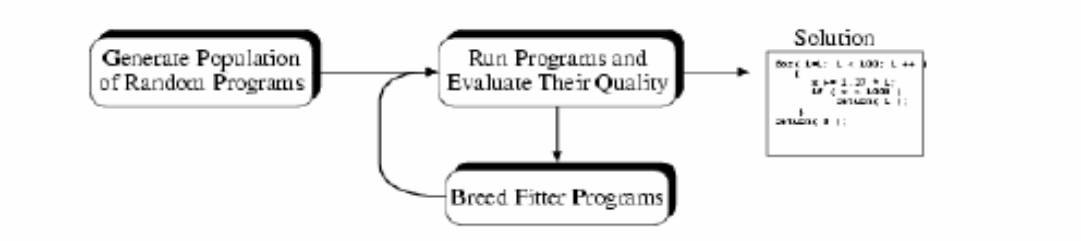


Figure 37 : la boucle principale de la programmation génétique. [113]

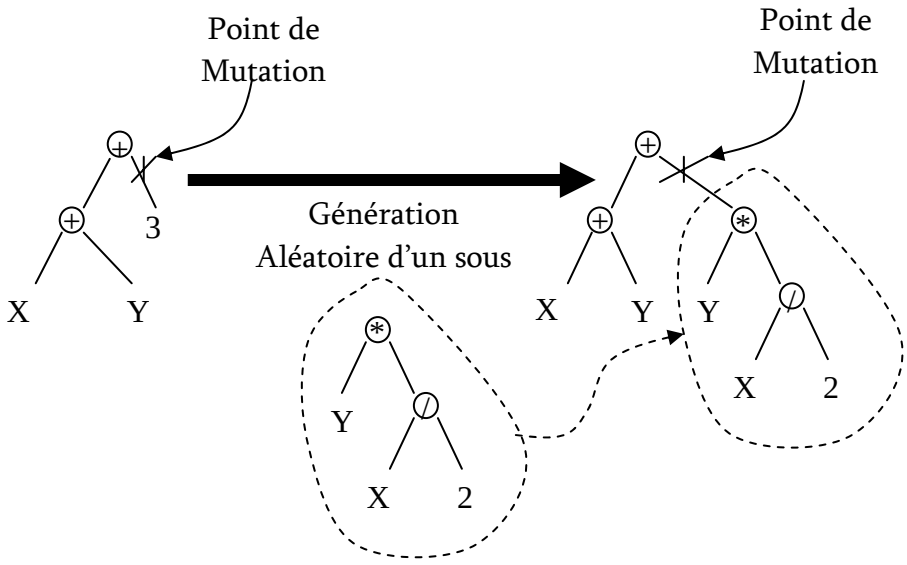


Figure 38 : un exemple de mutation d'un sous arbre [59].

IV. Concepts de base de GP :

Les individus de la programmation génétique :

Dans les algorithmes génétiques, la population est un ensemble d'individus, dont chaque individu forme une solution (appartenant à l'espace de solutions du problème étudié), cette solution est présentée sous forme de nombre binaire simple et elle est traitée en utilisant les opérateurs génétiques. Aussi que dans la programmation génétique, la population est un ensemble d'individus, mais ici les individus sont des programmes machines choisis aléatoirement au début du processus en combinant un ensemble de terminaux avec un ensemble de fonctions.

L'un de ces programmes est écrit dans la forme d'un arbre d'une façon à respecter l'ordre de l'exécution de son expression, cette dernière est écrite dans sa notion préfixe pour résulter l'arbre de programme.

Les étapes de base :

La préparation de la programmation génétique pour résoudre un problème quelconque passe par un ensemble d'étapes, elles sont :

- Il faut déterminer l'ensemble de terminaux ;
- Il faut déterminer l'ensemble de fonctions ;
- Il faut déterminer la mesure de fitness ;
- Il faut déterminer des paramètres pour l'exécution ;
  - ✧ La taille de la population,
  - ✧ Le nombre de générations,
  - ✧ Les paramètres concernant les opérateurs génétiques (la sélection, la reproduction, le croisement et la mutation),
- Il faut déterminer une méthode pour la conception de résultat et le test d'arrêt. [105]

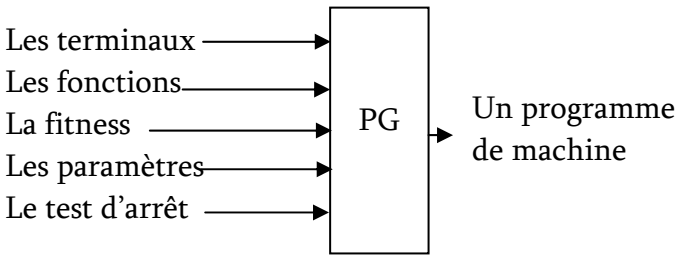


Figure 39 : Les étapes pour concevoir un PG d'un problème spécifique.

[96] dit qu'il y a six étapes préliminaires pour résoudre un problème utilisant la programmation génétique. Les terminaux sont choisis (1), les fonctions (2), la fonction de fitness (3), les paramètres de contrôle (4), le critère d'arrêt (5) et déterminant l'architecture de programmes (6).

Dans la terminologie de Koza, les terminaux et les fonctions sont des composants des programmes. [95]

Le choix des composants de programme (c'est-à-dire les terminaux et les fonctions) et le choix de la fonction de fitness largement détermine l'espace dont la programmation génétique cherche, et par conséquent quelle difficulté est la recherche, quel succès on va obtenir.

Le contrôle des paramètres inclus la taille de la population, le taux de croisement et de mutation, le nombre de génération, la méthode de sélection et est ce que le processus soit élitiste ou non, …etc. le critère d'arrêt est simplement une règle pour arrêter l'exécution du PG. Typiquement la règle est d'arrêter c'est de trouver un programme qui résout le problème ou bien après un nombre donné de générations. Dans la dernière étape l'architecture des programmes évolués est choisie. [95]

Le choix des terminaux et des fonctions :

La première exigence c'est quand on décide les terminaux et les fonctions dont les programmes évolués seront composés par, c'est d'assurer qu'ils sont capables d'exprimer la solution du problème. Koza [94] appelle ça la propriété de quantité suffisante.

Les terminaux et les fonctions sont à partir du langage qui représente les solutions probatoires. Les opérateurs génétiques (croisement, mutation, etc.), ADFs, la fonction de fitness et la sélection de schème combinés pour une transmission de fonction qui transforment la population d'individus écrite dans un langage de représentation. [95]

La fermeture (Closure) :

Koza [96] définit la fermeture d'être satisfait quand n'importe quelle fonction est capable d'accepter comme ses arguments n'importe quelle valeur ou quel type de données qui peuvent être possiblement retournés par n'importe quelle fonction (incluant elle-même) ou être prisent par n'importe quel terminal. Si les terminaux et les fonctions ont cette propriété alors de nouveaux arbres résultats peuvent être créés par le croisement. Dans les algorithmes génétiques traditionnels, la fermeture est c'est que le chromosome n'est pas besoin d'être traité comme un programme exécutable.

La fermeture est souvent réalisée par l'exigence de tous les terminaux, les fonctions et les arguments des fonctions d'être de même type, par exemple, les entiers.

La fonction de fitness :

En fin de compte la fonction de fitness guide l'évolution d'une population de la programmation génétique. La fonction de fitness ne donne pas seulement une grande récompense à la solution correcte, mais aussi elle récompense préférentiellement les solutions improuvées pendant l'exécution de la GP, commençant par la création de la population initiale jusqu'à la découverte de la solution finale.

Contrôle de paramètres :

Un ensemble de paramètres doivent être ajustés lors de la conception d'une programmation génétique sont : la taille de la population à évoluer, le nombre maximum de générations à atteindre, les opérateurs génétiques utilisés, les probabilités de croisement et de mutation, la méthode de sélection utilisée (tournoi, la roulette, …), …etc. Ces paramètres peuvent influer directement ou indirectement sur l'évolution de programmes ainsi que la convergence du système vers le programme optimal.

Le critère d'arrêt :

Le critère d'arrêt le plus commun est d'arrêter l'évolution de programmes génétiques qui est entre autre une solution exacte ou approximative est trouvée ou bien la génération 50 (par exemple) est atteintes. La motivation pour ces remarques pour l'observation qui est faite sur quelques problèmes de PG semble à « élonger de vapeur » après la 50^{ième} génération.

Les fonctions automatiquement définies (ADFs) :

« La fonction automatiquement définie (ADF) est une fonction (c'est-à-dire un sous-programme, une procédure, un module) qui est dynamiquement évoluée durant l'exécution de la programmation génétique et qui peut être appelée par un programme appelant (c'est-à-dire le programme principale) qui est simultanément été évolué » [96].

Koza et Rice introduisent les ADFs pour attaquer le problème de convergence, c'est-à-dire d'aider la GP de converger et de résoudre plus de problèmes complexes.

V. Autres paramètres de GP :

Le tournoi de sélection :

Dans les algorithmes évolutionnaires, il y a quelques différentes techniques dans l'utilisation pour décider quels individus vont être reproduire, combien d'enfants ils auront et quels individus vont être mourir (c'est-à-dire être supprimé de la population). La caractéristique générale est de récompenser les bonnes solutions avec plus de résultats (et possiblement aussi avec une longue vie).

Dans le tournoi de sélection, un nombre d'individus (la taille de tournoi) sont choisis dans l'aléatoire (avec re-sélection) de la population engendrée. Tout individu est comparé avec chacun des autres individus et le meilleur d'eux est choisi. Quand le nombre de candidats dans le tournoi est petit, les comparaisons ne sont pas chères. Un élément est inhérent dans le tournoi de sélection due à la sélection aléatoire des candidats. Quelques autres sélections de schèmes sont aussi stochastiques (c'est-à-dire contient un élément de chance) [95]

L'état stable des populations :

Dans l'algorithme général traditionnel des algorithmes évolutionnaires [97], l'évolution procède via une séquence de générations discrètes, ça ne résulte pas un chevauchement. Un individu existe seulement dans une seule génération, il peut seulement influer les générations ultérieurs à travers ses enfants. Ça est comme quelques espèces de plantes (et animaux) qui vivent seulement une année. Dans le printemps ils engendrent à partir des grains, évoluent durant l'été et produisent leurs grains dans l'automne. Ils survivent l'hiver par un sommeil gisant, mais leurs parents meurent. Ces nouveaux individus commencent à évoluer encore dans le printemps prochain. Ces espèces forment un ensemble continue pendant quelques années mais pas d'un seul individu qui vie plus qu'une année.

Dans les algorithmes génétiques à l'état stable [98], de nouveaux enfants sont continuellement ajoutés à la population et peut immédiatement être sélectionnée comme des parents pour les nouveaux individus. Usuellement quand à chaque individu est ajouté à la population un membre existant de la population est supprimé d'elle. Ça assure que la population reste à la taille constante. Les populations à l'état stable sont de plus en plus populaires. [95]

L'optimalité de Pareto :

Les GPs existants utilisent une fonction de fitness scalaire où chaque individu est donné une mesure singulière de leurs utilités. Une alternative est d'utiliser une mesure de fitness multidimensionnelle où chaque dimension de fitness réfère à un aspect différent de solution probatoire. [95]

L'optimalité de Pareto [11, page 197] offre une manière de comparaison entre les individus dans la population en utilisant des critères multiples sans introduire des moyens arbitraires pour les combinés en une seule fitness. Une preuve de l'efficacité de Pareto tournoi de sélection est donnée dans [99] qui contient un rapport des techniques d'optimisation multiobjectifs. Dans l'approche de Pareto, les valeurs de la fitness sont comparées dimension par dimension. Si une valeur de fitness est non plus mauvaise que les autres dans chaque dimension et meilleur dans une dimension au moins alors il est dit qu'il domine les autres.

VI. La programmation génétique et le phénomène de Bloating :

La programmation génétique est le plus courant algorithme évolutionnaire de variable longueur. L'objectif de la programmation génétique est usuellement de trouver le programme d'ordinateur le plus performant pour une tâche donnée. Comme un programme peut être de n'importe quelle taille et forme, avec les contraintes de programme à la main. Pour qu'un système de programmation génétique trouve un programme pareil, l'expérimentation a besoin de fournir seulement deux choses: une

description de comment former les programmes candidats, et une fonction qui estime les programmes candidats et les attribue des points. [103]

Dans la programmation génétique se trouvent des difficultés quand: les solutions candidates qui ont considérables tendent d'accroître d'une manière monstre et illimitée, et indépendante de haute qualité. Par exemple, ce n'est pas rare pour la programmation génétique de commencer en dehors des solutions candidates seulement cinq longues unités, et aussitôt remonter les candidates considérables pour n'importe quelles milliers de longues unités. [103]

Dans le monde de la programmation génétique, ce problème est connu par le Bloating (la croissance de code). Ce phénomène représente un problème sérieux dans la programmation génétique escaladée pour les problèmes larges et difficiles. Premièrement, cette croissance monstre de code consomme les ressources d'ordinateur, faisant le processus de recherche lent et plus lent, et éventuellement le forçant de stopper quand les ressources valables ont été épuisées. Deuxièmement, les solutions candidates gonflées sont souvent plus difficiles de les modifier dans les cas sémantiques, gênant la capacité de PG de se reproduire et de découvrir les meilleures solutions. Troisièmement, le bloating peut ralentir le processus d'évaluation de qualité. Dans un sens réel, le bloating rend la programmation génétique dans une course avec le temps, pour trouver la meilleure solution possible avant que ce phénomène met un arrêt effectif à la recherche. [103]

Mécanismes de ce phénomène :

Dans le code croissant examiné, c'est informatif de considérer la taille du programme moyen et la taille du grand programme. Dans GP standard, seulement que les deux opérateurs « Croisement et Sélection » affectent la population. Le croisement change les individus du programme dans la population et la sélection change l'ensemble des programmes participant dans la population.

L'opérateur de croisement entre deux programmes rend un programme plus large avec le même montant dont il rend les autres programmes plus petits. Ainsi, quoique le croisement puisse accroître ou décroître la taille du programme, il ne change pas la taille

moyenne de la population. L'opérateur de sélection peut facilement changer la taille moyenne de la population par la préférence de sélection des programmes larges ou petits, mais il ne peut pas changer la taille de n'importe quel programme. Ainsi, seule la sélection peut accroître la taille moyenne d'un programme au point où elle égale le plus large programme et seulement par la sélection de N copies des programmes les plus larges.

A cause des effets stricts des opérateurs de croisement et de sélection, la croissance du code soutenue vue dans la plupart des expérimentations de PG peut seulement causé par l'interaction entre ces deux opérateurs. En particulier, la sélection doit préférentiellement sélectionner le plus large programme créé par le croisement. Néanmoins, ce n'est pas évident, la raison inhérente de la sélection pour choisir les programmes les plus larges, ni le croisement fait préférentiellement crée les programmes les plus larges. Ainsi, il faut avoir un autre facteur qui pousse ces deux opérateurs envers les programmes les plus large.

Langdon et Poli ont démontré que pour au moins un problème, l'inclusion de la fitness cause la croissance de code [106]. La sélection aléatoire ne cause pas la croissance du code. Néanmoins, comme il est toujours noté, la croissance ne contribue pas directement à la fitness. Alors la question devient : c'est quoi avec la fitness basée sur la sélection qui, en conjonction avec le croisement, cause la croissance du code ? [107]

Sur cette question, plusieurs théories essayent d'expliquer les causes de la croissance du code dans la programmation génétique :

- La théories des « Introns » énonce que la croissance du code joue le rôle d'un mécanisme protection pour éviter les effets destructifs des opérateurs, une raison que seulement les solutions appropriées sont trouvées ([108], [109], [110]). Les Introns sont des pièces de code n'ont pas d'influence sur la fitness : sont les sous programmes qui ne sont jamais exécutés, ou les sous programmes qui n'ont aucun effet.
- La théorie « la fitness cause la croissance de code » : cette théorie repose sur l'hypothèse qu'il y a une grande probabilité de trouver le plus grand programme

avec le même comportement que de trouver le plus petit. Ainsi, une seule meilleure solution est trouvée, les programmes naturellement tendent d'accroître à cause de la pressure de la fitness [106]. Cette théorie énonce que la croissance du code est indépendante des opérateurs et peut avoir lieu pour n'importe quelle variable de longueur. Comme une conséquence, la croissance de code ne peut pas être limitée dans une population basée sur un algorithme stochastique (GP).

- La théorie de tendance de suppression : cette théorie énonce que la suppression des plus grand sous programmes et plus vulgaire que la suppression des plus petits (à cause des conséquence destructive possible), aussi, il y a une tendance naturelle qui bénéfice pour la préservation des programmes longs [111]. [112]

Mais on peut dire qu'aucune théorie donne une explication définitive sur les causes de croissance de code, ni une bonne solution pour éviter ce phénomène qui éloigne la convergence vers la solution optimale.

VII. *La programmation génétique et les problèmes complexes :*

Koza a étudié un ensemble de problèmes complexes en utilisant la programmation génétique, et il a démontré les avantages de cet algorithme, voila quelques problèmes :

- *La régression symbolique.*
- *Les spirales entrelacées.*
- *Wall following.*
- *Box moving.*
- *Les fourmis artificielles.*
- *La coévolution et les stratégies de jeux.*
- *La compression d'image programmatique.*
- *L'optimisation.*
- *Le problème de multiplier.*

- *N-parity.*

- *L'information quantique :*

[104] utilise la programmation génétique pour évoluer les algorithmes quantiques. L'information quantique prend avantage des théories de physique quantique pour construire les unités d'ordinateur qui, dépendant de l'algorithme, ont une meilleure complexité de calcul que les mieux algorithmes classiques possible. La donnée dans les ordinateurs quantiques n'est pas stockée avec les bits mais avec les qubits, les vecteurs de base représentant les états quantiques possibles de donnée. Les algorithmes quantiques prennent souvent la forme de réseaux d'avant alimentation de « circuits quantiques », des matrices spéciales qui opèrent sur ces qubits. L'information quantique est plus compliquée, mais [104] fut capable d'appliquer PG pour trouver un nombre intéressant d'algorithmes quantiques. Ça inclut les algorithmes quantiques connus (en avance le problème prometteur de Deutch et le problème de recherche dans une base de données de Grover), trouvant un algorithme quantique équivalent aux majorités de problèmes classiques, et découvrant les résultats quantiques dont un expert a pensée n'existe pas. [103]

- *… etc.*

VIII. Multiobjective genetic programming :

L'optimisation multiobjectifs a été devenue récemment en popularité et en utilité [100], mais elle a uniquement commencé à appliquer au programmation génétique dans les dernières années [101]. L'optimisation multiobjectifs permet d'utiliser multiples fonctions de fitness. Au contraire au traditionnel algorithme évolutionnaire, où les individus dans une population peuvent être classées dans l'ordre, l'optimisation multiobjectifs utilise un tri non dominé pour classer les individus. En plus de classement relatif de l'individu, les algorithmes multiobjectifs souvent mesure la diversité des individus dans la population. [102]

Plusieurs techniques qui forment les frontières de Pareto appliquées d'une manière générale aux algorithmes évolutionnaires sont appliquées spécialement à la

programmation génétique. Ces techniques qui évoluent un nombre fini de programmes machines en combinant des fonctions objectives indépendantes, et essayent de trouver tous les programmes dominants (qui forment les frontières de Pareto). Nous avons essayé de démontrer toutes ces épreuves dans ce mémoire.

IX. Exemple de Problème pour la programmation génétique :

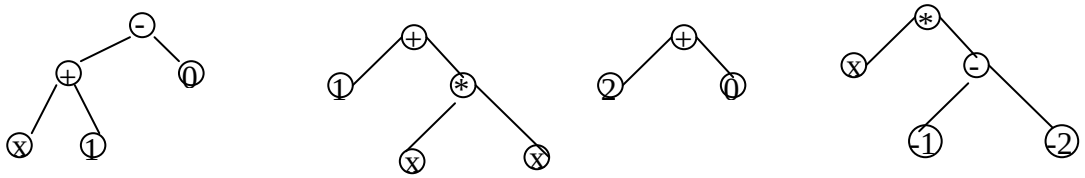
Un exemple de problème pour l'illustration du principe de la programmation génétique est « la régression symbolique »:

La régression symbolique est un exemple canonique pour la programmation génétique. Son objectif est de trouver une fonction symbolique qui mieux adapte un ensemble de points de données de (x, y) dans le plan réel cartésien. La régression symbolique diffère de la régression classique dans les statistiques dont fonction de l'ensemble inclut les fonctions transcendantales (sinus, cosinus, logarithme naturel).

Voilà un tableau résumant la conception de ce problème par la programmation génétique tout en exploitant l'ensemble de paramètres adéquats :

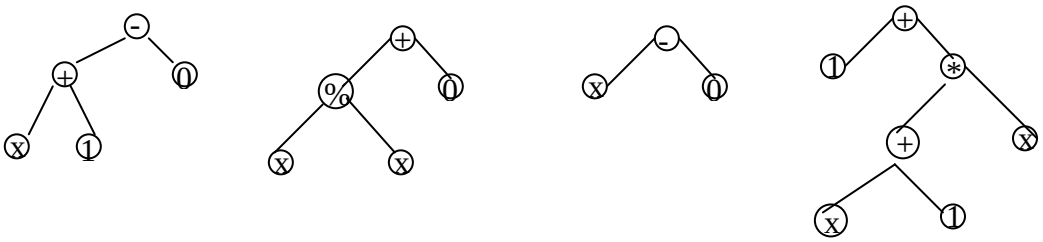
	Objective :	Trouver un programme d'ordinateur avec une entrée (variable indépendante x), et une sortie égale à la valeur de la polynomiale quadratique $x^2 + x + 1$ dans un intervalle entre -1 et +1.
1	L'ensemble de terminaux :	T = {x}.
2	L'ensemble de fonctions :	F = {+, -, *, %} Remarque : la fonction de division protégée % retourne la valeur 1 si une division par 0 est atteinte (incluant 0 divisé par 0).
3	La fitness :	La somme de la valeur absolue des différences (erreurs), calculés (en quelque manière) autour des valeurs de la variable indépendante de x à partir de -1.0 jusqu'à +1.0, entre la sortie de programme et le polynomiale quadratique $x^2 + x + 1$.

4	Paramètres :	La taille de la population M=4.
5	Test d'arrêt :	Un individu émergé dont la somme des valeurs absolues (erreurs) est moins que 0.1.



X + 1, F = 0.67 | X^2 + 1, F = 1.00 | 2 F = 1.70 | X F = 2.67

Figure 40: Quelques individus de la génération initiale et leurs fitness. [105]



X + 1 | 1 | X | X^2 + X + 1

Figure 41 : Quelques individus de la génération 1 [49].

X. *Conclusion :*

La programmation génétique est l'un des algorithmes évolutionnaires les plus répondus dans le domaine de la création de programmes machines à cause de ses capacités de créer des programmes aptes à résoudre le problème conçu. La voie de la programmation génétique a trouvé son succès surtout dans la création des programmes responsables de faire fonctionner des circuits électroniques répondant aux exigences de l'informatique quantique et autres.

Chapitre IV

Optimisation MultiObjectifs par Programmation Génétique

I. Introduction :

Dans les chapitres précédents, on a étudié les problèmes d'optimisation multiobjectifs, quelques exemples ainsi que les méthodes utilisées pour résoudre ce genre de problèmes. On a constaté que les méthodes les plus répondues sont les algorithmes évolutionnaires à cause de leurs avantages, surtout qu'elles n'étudient plus la nature et la définition du problème mais l'ensemble de ses solutions qui sont représentées d'une manière très simple (binaire, réel, ou programme). Ces méthodes qui favorisent la manipulation de ces solutions par des opérateurs génétiques, on a trouvé que les plus utilisées dans l'optimisation multiobjectifs sont les algorithmes génétiques tout en exploitant un ensemble d'approches, chacune a son principe d'évaluation, d'enregistrement des meilleures solutions, … ; aussi on a trouvé que beaucoup de travaux sont réalisés dans l'hybridation des algorithmes génétiques avec les problèmes d'optimisation multiobjectifs.

Parmi ces méthodes, on trouve aussi la programmation génétique, cette méthode a un énorme succès dans le domaine de la création des programmes d'ordinateurs ou de circuits logiques ou électroniques (surtout la voie de l'information quantique), mais peu (autrement dit, moins de dix) de travaux sont réalisés en hybridant la programmation génétique avec l'optimisation multiobjectifs. Pour cette raison, on a décidé de faire cette hybridation pour démontrer la souplesse et l'efficacité de cette méthode pour ce genre de problèmes.

Alors, dans ce chapitre, on va appliquer la programmation génétique à quelques problèmes NP-hard mono-objectifs, mais le problème rencontré est le phénomène de bloating (croissance de code d'une manière monstre). Pour résoudre ce problème on va exploiter la définition de l'optimisation multiobjectifs, c'est-à-dire, on va définir ce problème comme un deuxième objectif à atteindre contre la définition d'un problème NP-hard étudié. Ce travail est basé sur une approche célèbre des algorithmes évolutionnaires multiobjectives qui est SPEA2 (Strength Pareto Evolutionary Algorithms).

II. La résolution d'un problème d'optimisation par GP :

Comme on a vu précédemment, le principe de la programmation génétique est de faire trouver le meilleur programme de machine pour un problème spécifique, en se basant sur la définition du problème. Cette technique qui a comme objectif de laisser la machine apprendre toute seule à résoudre les problèmes sans une programmation humaine.

Mais dans le domaine de l'optimisation multiobjectifs, peu de travaux qui permettent de résoudre ces problèmes en utilisant la programmation génétique. Et à cause de la nature des problèmes d'optimisation multiobjectifs, on trouve plusieurs objectifs à atteindre d'une manière de trouver les meilleures solutions, autrement dit, trouver les frontières de Pareto.

Pour cela, l'utilisation d'un algorithme évolutionnaire veut dire d'essayer de maximiser ou de minimiser plusieurs fonctions objectives, c'est-à-dire plusieurs fitness, selon le nombre des objectifs qu'on a, et de trouver les frontières de Pareto (l'ensemble de solutions optimisant tous ces objectifs). Cette illustration est la même pour tous les types des algorithmes évolutionnaires, notamment la programmation génétique. L'utilisation de cette technique implique de trouver le meilleur programme dans les frontières de Pareto satisfaisant tous les objectifs.

Nous débutons notre conception pour trouver ce type de programmes par l'étude de la programmation génétique dans les problèmes d'optimisation, spécifiquement les problèmes NP-hard mono-objectifs, tout en traitant quelques exemples en utilisant l'environnement de programmation MATLAB.

III.1 Le choix de paramètres :

Comme tout algorithme évolutionnaire, on a un ensemble de paramètres à ajuster pour bien appliquer la programmation sur un problème. Ces paramètres permettent d'aider l'algorithme à converger d'une manière efficace et rapide vers la bonne solution. Et comme on a vu dans le chapitre précédent, en plus des paramètres des algorithmes

génétiques, la programmation génétique a d'autres paramètres à définir et régler comme les fonctions et les terminaux, …

Ces paramètres se basent dans une première phase sur la définition du problème. C'est-à-dire, à partir de la définition du problème, on va faire sortir l'ensemble de fonctions et de terminaux ainsi que la fonction de fitness. Ensuite, dans la deuxième phase, on règle l'ensemble des paramètres concernant les opérateurs génétiques (la reproduction, la sélection, le croisement et la mutation), ainsi que l'ensemble des paramètres concernant la population (taille, nombre d'individus, …). Dans cette phase, on décide aussi si notre algorithme soit élitiste ou non.

Avec cette démarche, on va décrire comment nous ajustant les paramètres d'un problème d'optimisation mono-objectifs (quelles sont les paramètres et quelles sont leurs valeurs) dans cette conception.

II.1.1 Les fonctions

La fonction dans la programmation génétique c'est l'élément qui permet de modifier des entrés pour avoir des résultats, c'est elle qui permet de faire dynamiser le comportement d'un programme machine. Cette notion de fonction dans la programmation génétique est, dans la plupart des cas, une fonction de base, c'est-à-dire une fonction élémentaire.

Exemple : l'addition, la soustraction, NOR, OR, AND, incrémentation, décalage, …

Mais aussi, elle peut être complexe (composée), d'une autre manière, elle se compose d'autres fonctions élémentaires et le traitement au sein de ce type de fonctions est un peu compliqué.

Exemple :

- Une fonction qui permet de trouver si un nombre entier soit un nombre premier ou non.
- Une fonction qui permet de trier deux nombre.
- …. Etc.

Dans notre conception, nous avons utilisé les deux cas, c'est-à-dire, nous avons des problèmes dans leurs définitions, nous trouvons seulement les fonctions élémentaires, d'autres problèmes nous trouvons les fonctions composées, ou bien les deux cas ensemble. Et puisque notre conception accepte tout genre de problème d'optimisation, nous pouvons ajouté n'importe quelle type de fonction.

Une fonction au sein de la programmation génétique, contient un ensemble d'entrées (arities) et une sortie (résultat), d'un autre terme les entrées présentent l'ensemble de ses opérandes qui doivent être des terminaux ou des résultats d'autres fonctions (ça dépend de la position de la fonction dans le programme), par exemple, si nous avons la fonction d'addition :

Elle a deux opérandes *a + b* et un résultat *c*, *a* et *b* peuvent être :

- Des terminaux : a = 3, b = 5
- Des résultats d'autres fonctions.

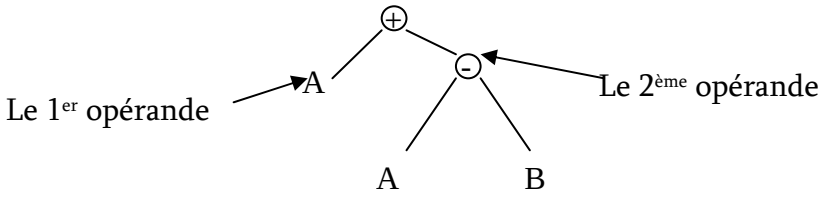


Figure 42 : la définition d'une fonction.

Le résultat d'une fonction fille est transmet à une fonction père comme un opérande de cette dernière :

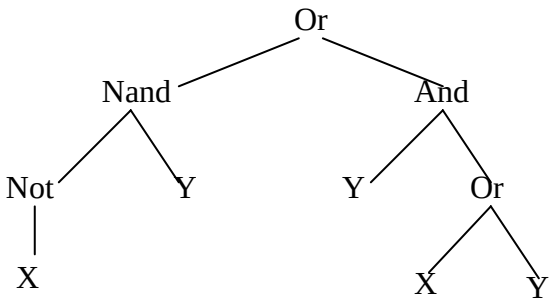


Figure 43 : 2ème exemple de la définition d'une fonction.

La fonction	Arity	Opérande
Not	1	X
Nand	2	Not(X), Y
Or	2	X, Y
And	2	Y, or(X, Y)
Or	2	Nand(not(X),Y), and((Y, or(X, Y))

II.1.2 Les terminaux

Se sont les données de programme, ils permettent de donner la valeur finale d'un programme en passant par l'ensemble de fonctions. Ils constituent les opérandes des fonctions vues dans la partie précédente. Les terminaux ont des valeurs fixes au long de déroulement d'un programme machine (c'est-à-dire des constantes).

Un terminal peut être une constante ou bien un résultat d'une fonction. Dans ce qui concerne le premier cas, les terminaux sont fixés au début de la programmation génétique, c'est-à-dire dans la phase d'ajustement des paramètres. Mais, le deuxième cas consiste que les résultats des fonctions filles au cours de déroulement d'un individu (programme) forment des terminaux ou des opérandes pour la fonction mère.

Exemple : on la forme : X + Y = C, on remarque que X, Y, C sont des terminaux.

Les terminaux aussi sont choisit d'une manière dépendante du problème, par exemple le problème de voyageur de commerce, les terminaux consistent: **les villes** à visiter, une structure qui contient **les distances** entre les villes et **la longueur** du chemin en cours.

II.1.3 La fonction de fitness (définition de problème)

Après avoir fixer les fonctions et les terminaux, la programmation génétique a besoin d'une fonction qui va décider quel individu va présenter le meilleur programme, et quels sont les individus qui vont participer dans tout le processus de la programmation génétique.

Cette fonction c'est la fonction de fitness, dont nous allons formuler selon la définition du problème en cours en se basant sur le résultat final d'un programme machine.

II.1.4 Le réglage des opérateurs (reproduction, croisement, mutation)

Dans la programmation génétique, sont les opérateurs génétiques qui jouent le rôle d'influer sur l'opération de choix des meilleures solutions de la population par :

- Reproduire les meilleurs individus sélectionnés par leurs valeurs de fitness.
- Croiser les individus pour avoir de nouveaux individus, tout en gardant la diversité dans la population.
- Faire changer un point dans un individu par l'opération de mutation.

Ces trois opérateurs ont en besoin de quelques paramètres pour que l'utilisateur puisse les contrôler et pour que l'exécution puisse converger facilement à la meilleure solution, parmi ces paramètres on trouve que le plus important c'est la probabilité d'appliquer un opérateur quelconque. D'un autre terme, il faut savoir la probabilité qu'un individu subi l'opération de croisement ou la mutation dans une génération, ou bien le nombre de fois qu'un meilleur individu soit reproduit pour composer la génération suivante.

Cette probabilité dans notre conception est laissée au choix de l'utilisateur, dont il peut la mettre :

- Fixe : elle est fixée au long de l'exécution, et elle a les valeurs (le croisement = 0.5 et la mutation = 0.5) comme dans la figure suivante :

- Variable : elle se change durant le processus de la programmation génétique selon l'état génétique pour chaque génération.

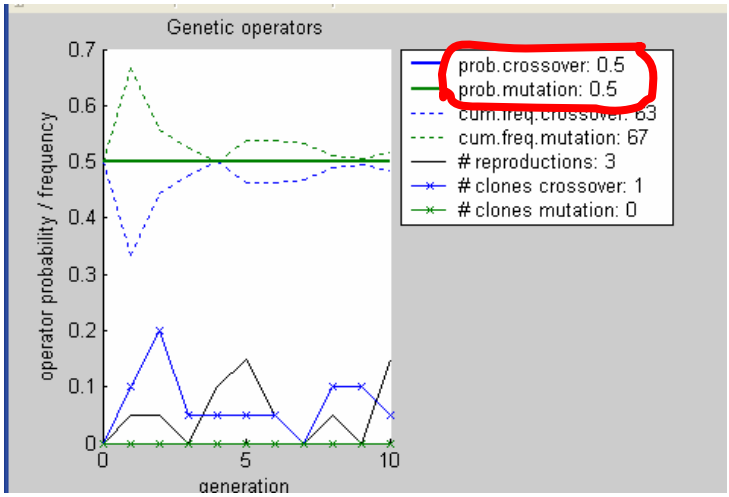


Figure 44 : Les probabilités de Mutation et de Croisement sont fixes.

La mutation se fait sur un seul individu et son résultat aussi est un seul individu, mais le croisement se fait sur deux individus, par contre son résultat est un seul individu. Nous avons affecté à la reproduction une probabilité de 0.1, aussi nous avons mis la probabilité minimale 0.1 pour les trois opérateurs. Pour le nombre d'individus qui se reproduisent, nous avons déduit qu'il va être variable d'une génération à une autre, ça dépend du nombre d'individus meilleurs.

II.1.5 L'initialisation de la population :

La population initiale est introduite en utilisant plusieurs méthodes :

- La méthode de « **Full** » : dans cette méthode, chaque nouveau arbre reçoit les nœuds qui ne sont pas des terminaux dans l'initialisation des niveaux de la profondeur, excepté le dernier niveau qui est constitué seulement des terminaux. Cette méthode est perfectement balance tous les branches de la même longueur.

- La méthode de « **Grow** » : chaque nouveau nœud est choisi entre les terminaux et les non terminaux, mais les nœuds du niveau initial sont tous des terminaux. Cette méthode ne garantie pas la balance des branches (quelques un sont plus longs que les autres).

- La méthode de « **Ramped** » : c'est la méthode « Ramped Half and Half » dont un nombre égale d'individus est initialisé pour chaque profondeur entre 2 et la valeur initiale de la profondeur pour chaque niveau où la moitié d'individus est initialisé en utilisant la méthode de « Full » et l'autre moitié est initialisé par la méthode de « Grow ». la population résultante de cette méthode est plus diverse entre les arbres balancés et les arbres non balancés.

II.1.6 Les limites de la taille et la profondeur des arbres :

La taille d'un arbre c'est le nombre de nœuds contenant de cet arbre et la profondeur c'est le nombre de niveaux existants dans l'arbre, les limites de ces deux notions sont rétablies d'une manière dynamique selon le problème et le résultat des opérateurs génétiques.

II.1.7 Le meilleur individu :

La sélection des individus dans chaque population est faite par une des méthodes suivantes :

- Roulette : cette méthode agit comme une roulette avec des pointeurs aléatoires, chaque individu possède une portion de la roulette qui correspond à son nombre d'enfants attendus.
- Sus : cette méthode est reliée à la méthode de « Roulette » seulement les pointeurs sont espacés d'une manière égale.
- Tournament : cette méthode choisit chaque parent par dessiner aléatoirement le nombre d'individus de la population et elle sélectionne le mieux entre eux.
- Lexictour : cette méthode implémente « The Lexicographic Parsimony Pressure » elle a le même principe que le Tournament mais si deux individus ont la même fitness elle choisi l'individu le plus petit.

II.1.8 Le calcul de la complexité et la diversité :

Dans cette conception, pour savoir si la population courante est complexe et elle maintient la diversité, on utilise des fonctions qui calculent ces deux notions pour aider le processus à converger vers le meilleur individu.

II.1.9 Quels sont les individus qui survivent ?

Pour que les meilleurs individus survivent pour participer aux populations prochaines, il faut exploiter l'un des principes de l'élitisme :

- Replace : les enfants remplacent la population parente complètement, cette méthode ne garde pas l'élitisme.
- Keepset : l'individu meilleur du deux parents et leurs enfants est maintenu pour la nouvelle population indépendamment de son état (parent ou fils).
- Halfelitism : la moitié de la nouvelle population va être occupé par les meilleurs individus choisis.
- Totalelitism : les meilleurs individus des deux parents et les enfants sont choisis pour remplir la nouvelle population.

II.1.10 La représentation graphique des résultats :

Pour bien interpréter les résultats et le processus de l'exécution, on a utilisé des représentations graphiques de :

- La valeur de fitness : dans ce graphe on montre l'évolution de la valeur : maximal, moyenne, médique pour le meilleur individu.
- La complexité : ce graphe représente l'évolution de la taille, la profondeur de l'arbre et le pourcentage des introns durant l'exécution.
- La diversité : ce graphe montre l'évolution de la diversité de la population.
- Les opérateurs : ce graphe montre l'évolution des probabilités des opérateurs et les fréquences des occurrences opérés, ainsi que le nombre de reproductions des individus de la génération courante.

- Les frontières de Pareto : on montre la meilleur fitness trouvée pour chaque arbre, les frontières de Pareto pour un ensemble de solutions ainsi que la taille et la profondeur de la population courante.
- Visualisation de l'arbre : ce graphe visualise l'arbre représentant le meilleur individu trouvé durant l'exécution (terminaux et fonctions).

III.2 Exemple « TSP » :

Le premier exemple dont nous avons appliqué dans cette conception c'est le problème NP-Hard Voyageur de commerce (TSP), ce problème qu'on a étudié on détaille dans le premier chapitre. La seule application de la programmation génétique pour ce problème est le travail de Bretton Swope intitulé **« Evolution of a Path Generator for a Round-Trip Symmetric Traveling Salesperson Problem Using Genetic Programming »** [114], le but de ce travail est de trouver le meilleur programme permettant de résoudre le problème de TSP, c'est-à-dire :

- Trouver le chemin le plus court reliant un ensemble de villes à partir de la première ville et revenant à cette dernière.
- Pas de répétition de ville, c'est-à-dire la visite d'une ville se fait une et une seule fois.
- Visiter toutes les villes.
- Le problème ici est un problème mono-objectif.

Comme on a vu précédemment que pour réaliser la programmation génétique pour un problème spécifique, il faut passer par un ensemble d'étapes :

L'ensemble de fonctions : d'après la définition de problème, nous avons essayer de créer les fonctions qui vont former notre PG, nous avons construit deux fonctions :

- ✓ CombinePath qui a deux fils (x, y) qui sont en général soit deux villes ou bien des résultats d'autre fonctions, et elle retourne à ces parents **le chemin** reliant les villes visitées et **la distance** obtenue.

- ✓ SwapAndShift qui a un seul fils (x) qui est le chemin et la distance entre quelques villes, elle retourne à ces parents, après quelques opérations faites sur le chemin obtenu, le chemin actuel et sa distance.

Le principe de ces deux fonctions est présenté dans les figures

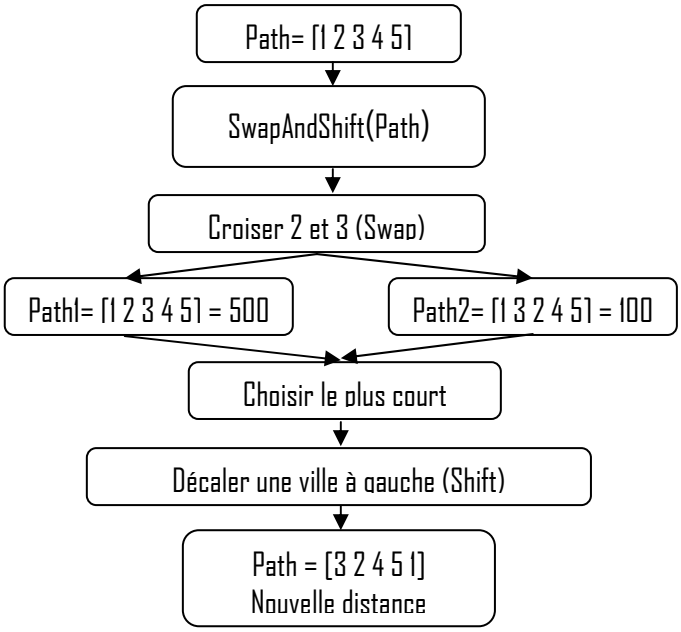


Figure 45 : La fonction SwapAndShift.

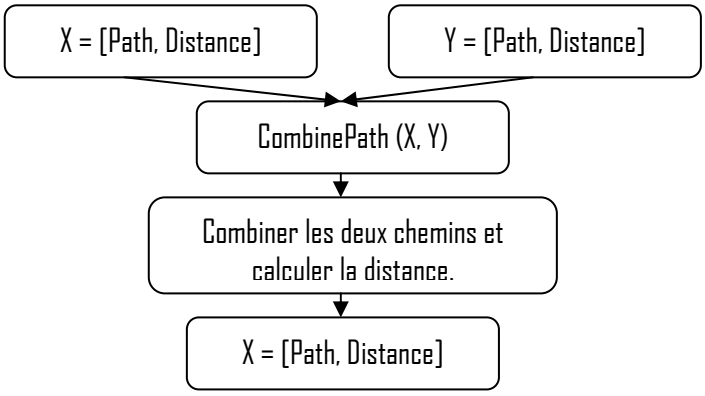


Figure 46 : La fonction CombinePath.

La fonction	Arity
CombinePath	2
SwapAndShift	1

L'ensemble de terminaux : les terminaux ici sont les villes à visiter *City1, City2, …, City10*, nous avons choisi 10 villes où ces villes ont des distances entre eux comme dans le tableau suivant :

	1	2	3	4	5	6	7	8	9	10
1	0	921	153	690	579	536	575	415	308	327
2	921	0	1048	396	848	554	628	805	1029	1128
3	153	1048	0	782	579	611	629	431	387	202
4	690	396	782	0	455	185	237	440	904	806
5	579	848	579	455	0	303	220	167	881	489
6	536	544	611	185	303	0	98	257	784	622
7	575	628	629	237	220	98	0	225	844	608
8	415	805	431	440	167	257	225	0	715	384
9	308	1029	387	904	881	784	844	715	0	588
10	327	1128	202	806	489	622	608	384	588	0

La fonction de fitness : puisque le problème ici est mono-objectif, alors on a une seule fitness à représenter qui reflète toutes les exigences de problème :

$$\text{Fitness} = 1000*(Nv-Nc) + (Opd - Ipd)$$

Tel que : Nv = nombre de villes à visiter = 10,

Nc = nombre de villes déjà visitées,

Opd = une distance optimale = 4375, 6000.

Ipd = la distance de chemin de l'individu actuel.

Les opérateurs : sont la mutation, le croisement et la reproduction, avec une probabilité variable durant l'exécution.

Les paramètres :

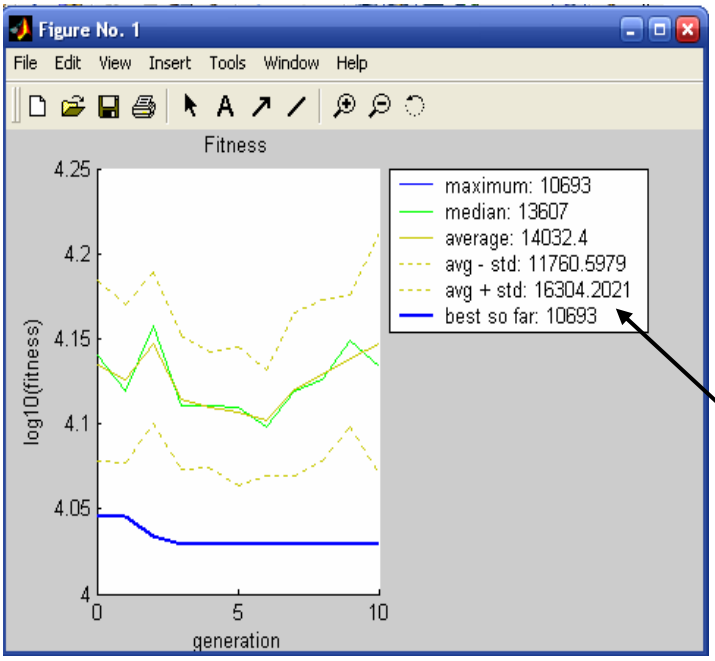
Objectif	Trouver le meilleur programme permettant de résoudre le problème de TSP, c'est-à-dire trouver le meilleur chemin.
L'ensemble de terminaux	City1, City2, City3, City4, City5, City6, City7, City8, City9, City10

L'ensemble de fonctions	CombinePath, SwapAndShift
La fitness brute	La différence entre le chemin optimal et le chemin retourné par l'individu plus une pénalité si le chemin ne comporte pas toutes les villes.
Probabilité de mutation	Variable
Probabilité de croisement	Variable
Nombre de copies à reproduire	Variable
Le succès	Le nombre de villes visitées (existant dans le chemin final)
La taille de population	5, 10, 20, 50
Le nombre de générations	5, 10, 20, 50
La population initiale	Full, Grow, Ramped
La méthode de sélection	Roulette, Tournament
L'élitisme	Replace, Keepset, Totalelitism
Les graphes	Les 5 types de graphes

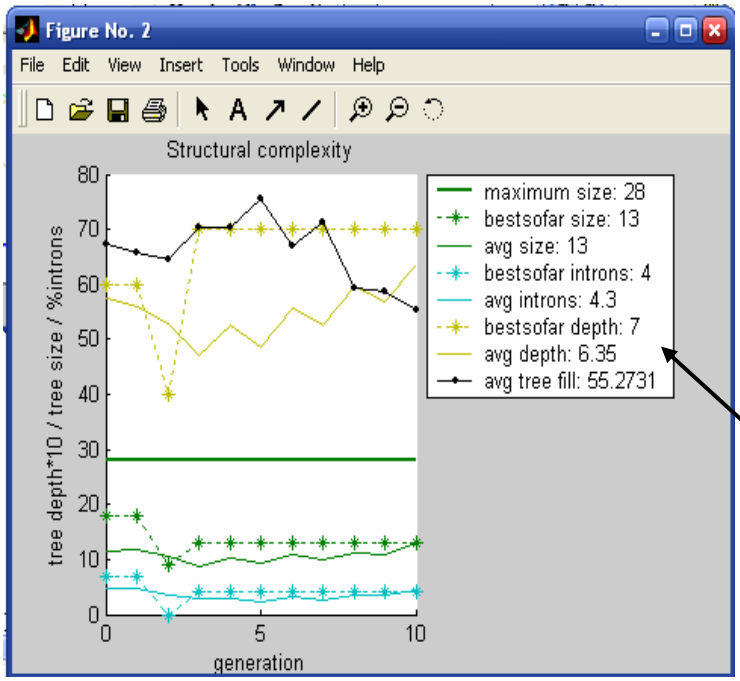
N.B : ces paramètres sont fixés dans plusieurs essaies.

Quelques résultats :

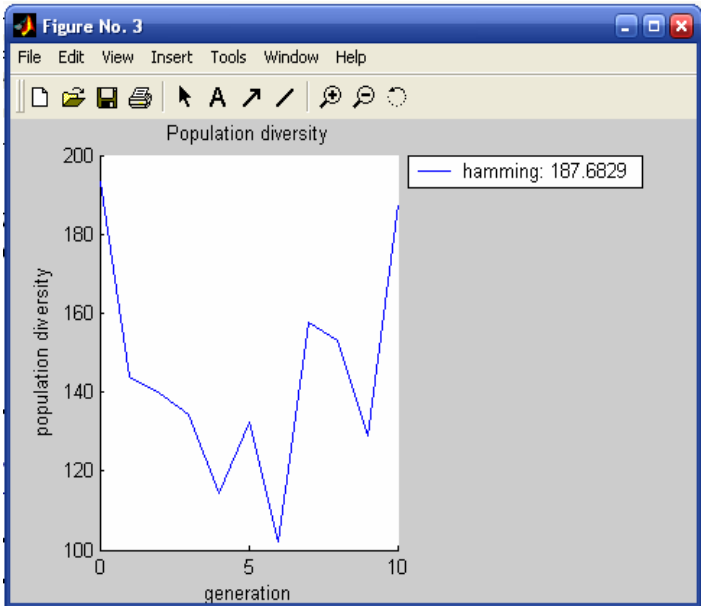
Voilà les résultats obtenus dans des exécutions différentes, la première est détaillée :



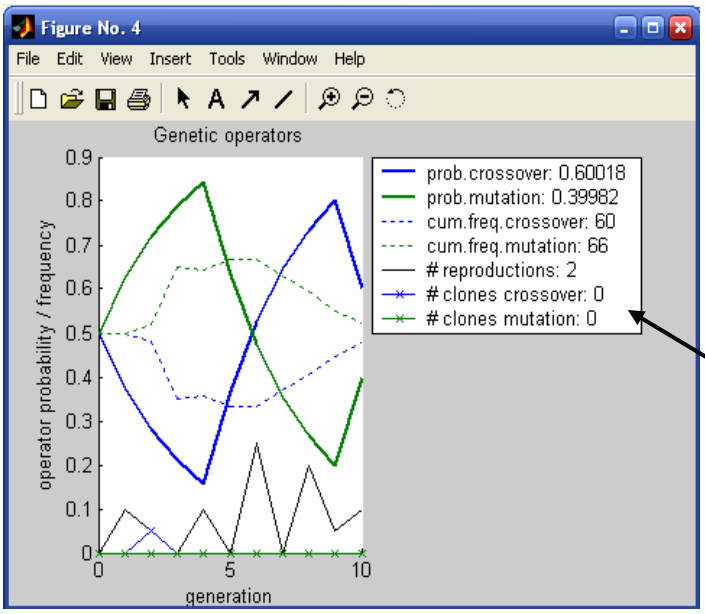
Les résultats de la fitness de la population courantes : Le maximum, la moyenne et le meilleur individu.



La taille maximale, la taille du meilleur individu, les introns, la profondeur moyenne du meilleur individu.

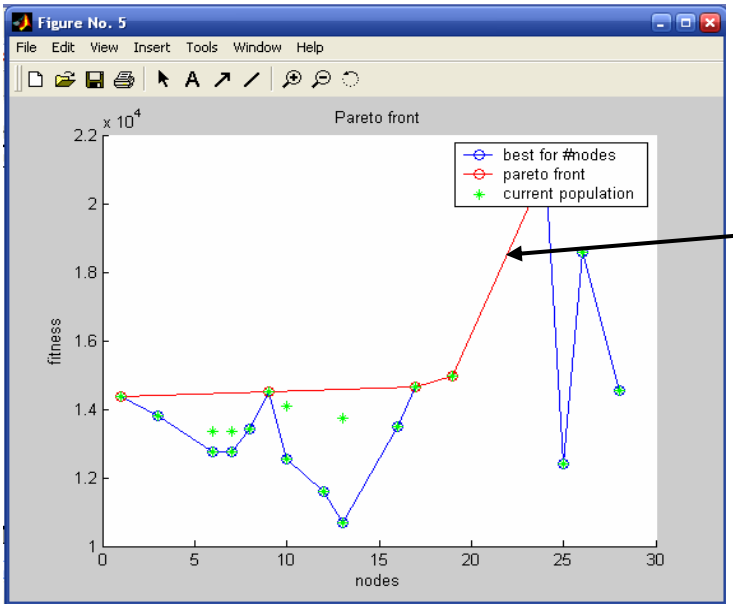


La diversité de la population.

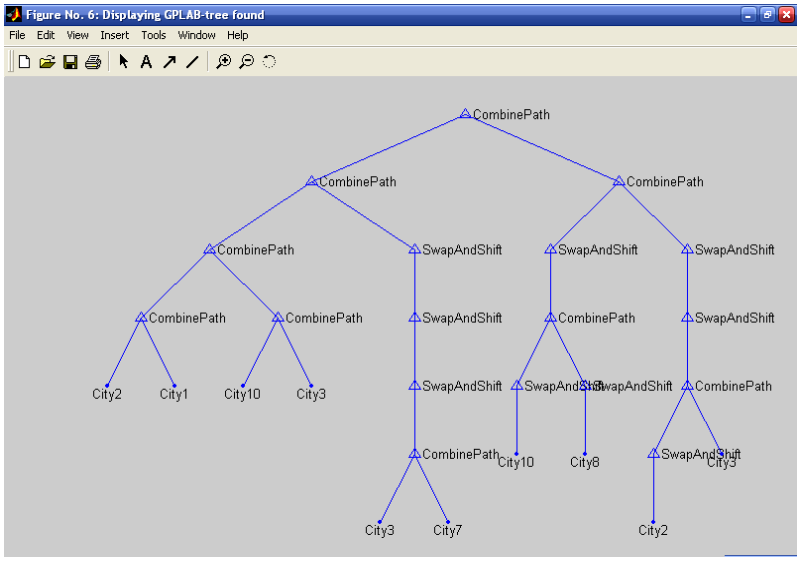


Des informations sur les opérateurs pour chaque population (probabilité, fréquence d'occurrences, …

Générations	1	2	3	4	5	6	7	8	9	10
Prob-Cross	0.625	0.28	0.21	0.15	0.36	0.52	0.64	0.73	0.80	0.60
Prob-Mutat	0.375	0.72	0.79	0.85	0.64	0.48	0.36	0.27	0.20	0.40
Fréq-Cross	6	12	15	20	24	28	36	43	52	60
Fréq-Mutat	6	13	28	36	48	56	61	63	54	66
Reproductoions	2	1	0	2	0	5	0	4	1	2



Les frontières de Pareto.



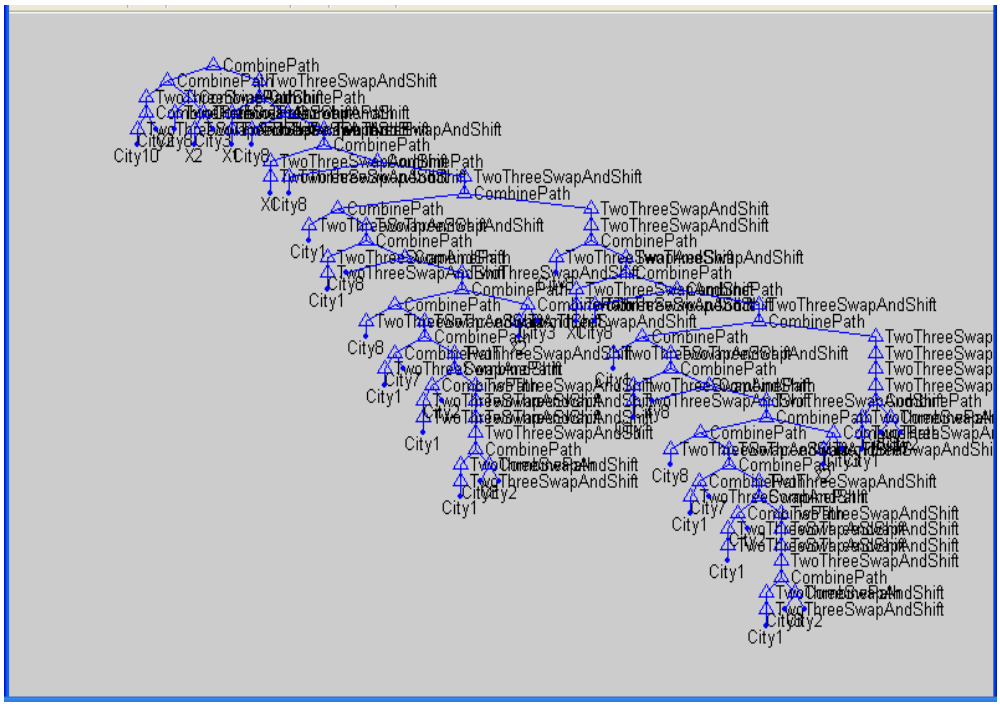
Le meilleur individu

Le chemin le plus court = 10693.

Le meilleur individu = 73.

La profondeur du meilleur individu = 20.

Le nombre de nœud = 13.



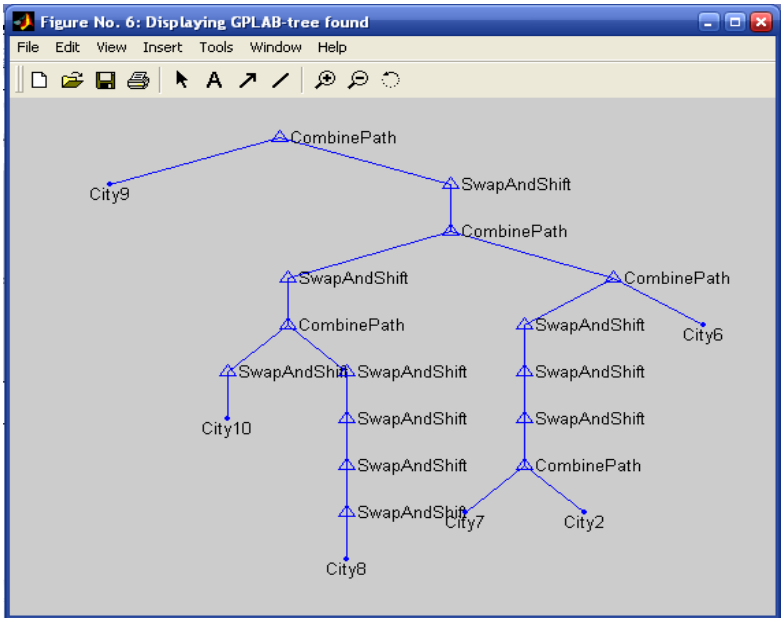
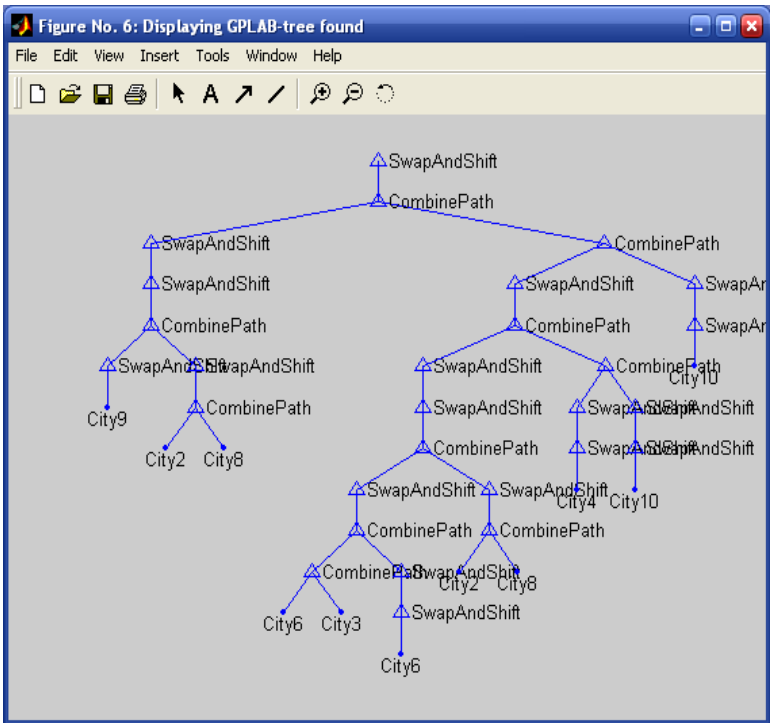
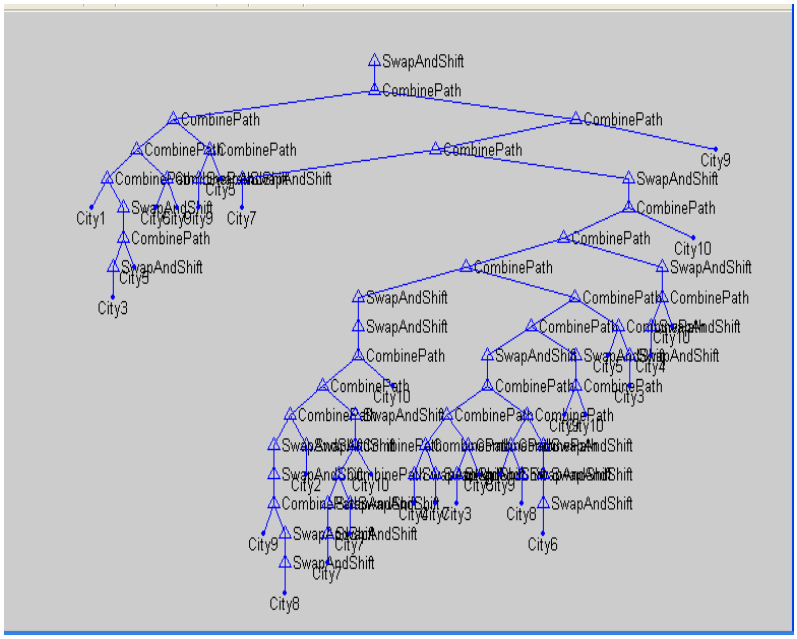


Figure 47 : Un arbre Monstre.



L'arbre après 3 générations de fitness de -679



II.2.1 Remarques sur les résultats :

D'après les résultats obtenus dans plusieurs essaies d'exécution de notre conception par les paramètres décrits dans les parties précédentes de ce chapitre, on remarque que :

- Les problèmes qu'on peut modéliser sont des problèmes NP-hard mais sont des problèmes monobjectif, et l'objectif de cette recherche est les problèmes multiobjectifs.
- Une seule fitness est conçue, ce que veut dire qu'on ne peut pas bien formuler les frontières de Pareto.
- Les arbres obtenus ont les inconvénients suivants :
  - ✓ Ile ne donnent pas la solution voulue qui répond à tous les exigences du problème (par exemple le problème de TSP, dans les arbres obtenus, on ne trouve pas toutes les villes et les villes aparaissent se répètent dans l'apparaition, …).
  - ✓ Se sont des arbres monstres c'est-à-dire, ils sont dans l'état de bloating dont on a parlé dans le chapitre précédent (la profondeur et la taille de l'arbre sont monstres).
- Les solutions obtenues ne convergent pas vers la meilleure solution à cause des inconvénients précédents.

Alors, il faut :

- ✓ Avoir une autre fitness pour que le problème devienne un problème multiobjectifs.
- ✓ Avoir une solution pour le phénomène de Bloating et générer des arbres de petite taille.

III. L'optimisation de l'arbre comme un 2ème objectif :

III.1 Formulation de problème

La tendance de la taille d'arbre qui s'accroît rapidement durant l'exécution de la PG est connue et elle influe sur la convergence vers la solution optimale. Plusieurs raisons pour lesquelles il faut prendre quelques solutions contre ce phénomène :

- ✓ Le coût d'utilisation de mémoire et de temps CPU.
- ✓ Les petites solutions généralisent les données mieux que les grandes.
- ✓ Quand la taille de l'arbre devient monstre, la fitness de la population ne peut pas être prouvée.

Pour ces raisons il faut essayer de limiter la taille de l'arbre, et puisque l'objectif de cette recherche est la résolution des problèmes d'optimisation multiobjectifs par la programmation génétique, on va utiliser ses techniques dans la programmation génétique pour réduire l'effet du bloating toute en définissant la taille du programme comme une deuxième fonction objective contre la fonction du problème.

Finalement, on va aboutir à une méthode utilisant la programmation génétique pour résoudre les problèmes d'optimisation multiobjectifs dont l'une des fonctions objectives c'est une solution contre l'accroissement des arbres d'une manière monstre.

Dans la section suivante, on va essayer de trouver comment formuler la deuxième fonction objective contre le bloating toute en décrivant les méthodes utiliser dans ce domaine.

III.2 Les méthodes de résolution de problème du Bloating

Beaucoup d'études ont examiné les raisons possibles de Bloating [02, 03, 04]. L'accroissance dans la taille de code est un effet appelé « Introns », une partie de l'arbre qui n'affecte pas la fonctionnalité des individus. Vers la fin de l'exécution du PG les introns s'accroissent rapidement et comprennent presque tout le code tant que le processus d'optimisation stagne (pas d'amélioration de fitness) [116]. Alors, la question c'est pourquoi l'évolution favorise les programmes avec des grandes sections du code non

fonctionnelle par rapport aux petites solutions. Une explication c'est que le croisement de PG est non homologue, c'est-à-dire il ne change pas les fragments de code qui ont la même fonctionnalité dans les deux parents. Donc, le croisement réduit le plus souvent la fitness des progénitures relatives à leurs parents en perturbant les segment valables de code ou en les plaçant dans des différents contextes. A cause que les points de croisement sont choisis aléatoirement dedans un individu, le risque de perturber les bloques de code fonctionnel peut être réduit substantiellement en ajoutant les introns.

Pour gêner cet effet d'utiliser beaucoup de ressources machine, il faut limiter la profondeur ou le nombre de nœuds ainsi que de trouver la limite raisonnable est difficile. Si la limite est faible, PG ne pourrait pas trouver la solution, si elle élevée, l'évolution serait freinée à cause des ressources immenses utilisées, c'est pour ça plusieurs stratégies sont adaptées pour résoudre ce problème :

- Des méthodes qui modifient la structure du programme et/ou les opérateurs génétiques pour supprimer ou réduire les facteurs influant sur ce phénomène comme la méthode d'ADF « Automatically Defined Functions »[117], EDI « Explicitly Defined Introns »[116] et Deleting Crossover [115].
- Des méthodes qui intègrent la taille des programmes comme un objectif « caché », par exemple : une contrainte (pour limiter la taille) ou bien un terme de pénalité « Parsimony Pressure » [118], et autres.

Ces deux approches ont des inconvénients : pour la 1ère approche, il faut reconnaître comment la structure et les opérateurs génétiques interagissent avec le phénomène. Et pour la 2ème approche comment on peut associer l'ensemble des paramètres optimaux entre eux, par exemple choisir le facteur de parcimonie approprié quand on applique la méthode de Constant Parsimony Pressure [118].

III.2.1 Constant Parsimony Pressure :

Un mécanisme pour limiter la taille de code est de pénaliser les grands programmes en ajoutant un terme dépendant de la taille à la fitness. Cette approche est nommée « Constant Parsimony Pressure » [118], la fitness d'un individu i est calculée en

ajoutant le nombre de tranches Ni, pondéré par un facteur de parcimonie α, à la fitness régulière :

$F_i = E_i + \alpha * N_i$, tel que Ei c'est une erreur de fitness.

Soule et Foster [118] rapportent que dans quelques exécutions, la pression de parcimonie guide toute la population à la taille minimale possible.

III.2.2 La méthode de deux étapes :

Autre alternative est d'optimiser la fonctionnalité ensuite la taille [119]. La formule de la fitness pour un individu i dépend sur ses propres performances. Il est nécessaire de mettre une erreur acceptable et maximal ε. Pour les problèmes discrets ε peut être mis à zéro. La population est divisée en deux groupes :

1- Les individus qui n'ont pas atteint encore une erreur égale ou moins de ε, obtiennent une fitness accordant à leur erreur Ei, sans aucune pression sur la taille :

$$F_i = E_i + 1 \qquad if \qquad E_i > \varepsilon$$

2- La fitness de l'individu qui a atteint une erreur égale ou moins de ε. La nouvelle fitness est calculée en utilisant la taille Ni de l'individu i :

$$F_i = 1 - \frac{1}{N_i} \qquad if \qquad E_i \leq \varepsilon$$

Un individu avec une grande taille d'arbre aura une fitness proche tandis que la fitness petite sera beaucoup proche du zéro.

Un avantage de cette méthode est que la pression sur la taille ne gênera pas PG de trouver les meilleures solutions parce que pas de pression est appliquée sauf que les individus qui ont atteint la performance aspirée.

III.2.3 Adaptive Parsimony Pressure :

Cette méthode est similaire à la précédente. Zhang et Mühlenbein ont proposé un algorithme qui varie le facteur de pression α durant l'évolution [120] :

$$F_i(g) = E_i(g) + \alpha(g) * C_i(g)$$

Ci(g) est la complexité de l'individu i dans la génération g. la complexité peut être définie en plusieurs manières [120], par exemple, le nombre de nœud dans l'arbre ou la taille normalisée en divisant la taille de l'individu par la taille maximale de la population [115]. Au contraire à la stratégie de deux étapes, la fonction de fitness ne dépend pas de la performance de l'individu mais de la meilleure performance de la population à la génération g. La pression de parcimonie utilisée pour calculer la fitness dans la génération g est accrue substantiellement si le meilleur individu dans la génération g = 1 a atteint une erreur ε.

Blickle [115] a rapporté des résultats supérieurs comparés à la méthode de Constant Parsimony Pressure quand on applique Adaptive Parsimony Pressure pour un problème continu et les mêmes résultats dans les problèmes discrets.

III.3 La définition du bloating comme une 2ème fonction objectif

Dans notre approche, on ne peut pas utiliser une seule fitness parce que l'objectif est de trouver une approche pour résoudre les problèmes d'optimisation multiobjectifs par la programmation génétique, et en même temps, on ne peut pas ignorer la fitness représentant la fonctionnalité du problème. Pour ces raisons, on va concevoir une approche d'optimisation bi-objectifs :

a. Le premier objectif c'est la définition du problème.

b. Le deuxième objectif c'est de réduire le bloating des arbres au cours d'exécution.

Alors, on a deux fitness à évoluer la première représente la définition du problème et elle est variée selon le problème à résoudre. La deuxième c'est de réduire le bloating contre la première et elle est **standard** pour tous les problèmes. Pour la 2ème fitness, on a choisi l'approche « Adaptive Parsimony Pressure » décrite par Byoung Tak-Zhang comme suit:

$$F_i(g) = E_i(g) + \alpha_i * C_i$$
$$E_i = \textit{une erreur}$$
$$C_i = \textit{la complexité du programme}(\textit{nombre de noeuds})$$

$$\alpha(G) = \begin{cases} \frac{1}{T^2} * \frac{E_{Best}(G-1)}{\widehat{C}_{Best}(G)} & si\ E_{Best}(G-1) > \varepsilon \\ \frac{1}{T^2} * \frac{1}{E_{Best}(G-1) * \widehat{C}_{Best}(G)} & \text{sinon} \end{cases}$$

E_{best} = l'erreur du meilleur individu

T = la taille de l'ensemble

$\widehat{C}_{Best}$ = l'estimation de la complexité du meilleur individu (programme)

$$\widehat{C}_{Best}(G+1) = C_{Best}(G) + \Delta C_{sum}(G)$$
$$\Delta C_{sum}(G) = \tfrac{1}{2}(C_{Best}(G) - C_{Best}(G-1) + \Delta C_{sum}(G-1)$$
$$\Delta C_{sum}(0) = 0$$

III.4 Les frontières de Pareto bi-objectifs

Naturellement, la plupart des problèmes d'optimisation impliquent des objectifs conflictuels qui ne peuvent pas être optimisés simultanément. Ce type de problème est souvent pris en trensformant les critères d'optimisation vers un seul objectif qui est ensuite optimisé en utilisant une méthode uni-objectif appropriée. Mais dans notre recherche, on a deux objectifs :

1/ la fonctionnalité du problème,

2/ et la taille du code.

Quand le deuxiée objectif est traditionnellement converti vers une contrainte en limitant la taille d'un programme, on va controler la taille du code en ajoutant un terme de pénalité « La pression de parcimonie » conrrespond à la somme des agrégation pondérés. Le classement des objectifs, c'est-à-dire optimiser en premier lieu la fonctionnalité du problème ensuite la taille du programme (la stratigé à deux étapes), est faiblement différent, mais reste en besoin d'incorporer les informations de performence comme dans les autres techniques.

Alternativement, il existe des méthodes qui traitent tous les objectifs d'une manière égale. A la place de restriction la recherche à priori à une solution comme dans les stratégies susmentionnées, elles essayent de trouver ou d'approcher ce qu'on appelle

« L'ensemble Optimale de Pareto » consisté des solutions qui ne peuvent pas être prouvées dans un seul objectif sans la dégradation vers un autre. Dans la dernière décennie, plusieurs algorithmes évolutionnaires sont développées pour ce scénario d'optimisation, et quelques études [120], [121] démontrent pour un ensemble de problèmes de test que ce type peut être supérieur, c'est-à-dire la somme des agrégation pondérés dans le terme del'effort et la qualité des calculs des solutions trouvées (dans le cas d'utilisation d'un algorithme évolutionnaire élitiste). C'est une motivation pour appliquer un algorithme évolutionnaire multiobjectif pour le problème de bloat dans la programmation génétique en considérant la fonctionnalité et la taille d'un programme comme des objectifs indépendants.

Pour cette raison, on va étudier un problème d'optimisation **bi-objectifs** pour trouver les frontières de Pareto tout en maintenant la diversité de la population. Pour atteindre ces objectifs, on a choisi l'approche SPEA2 des algorithmes évolutionnaires multiobjectives.

III.5 L'approche SPEA2

Dans cette recherche, on va utiliser une version prouvée de l'algorithme « Strength Pareto Evolutionary Algorithm (SPEA) » pour l'optimisation multiobjectifs proposée dans [93]. En plus de la population SPEA maintient un ensemble externe d'individus (archive) qui contient les solutions non dominées parmi toutes les solutions considérées. Dans chaque génération l'ensemble externe est actualisé et si nécessaire taillé en utilisant la procédure de **clustering** (groupement). Ensuite, les individus de la population et de l'ensemble externe sont évalués interdépendemment, tel que les membres de l'ensemble externe ont les valeurs meilleures de la fitness par rapport aux membres de la population. Finalement, la sélection est effectuée sur l'union de la population et de l'ensemble externe ainsi que les opérateurs : croisement et mutation. Puisque l'approche SPEA a démontrée de bonne performance dans les études comparatives [120], [121], elle a été un point de référence dans les différentes investigations récentes [122]. De plus, elle a été utilisée dans des différentes applications [123].

☞ Principes

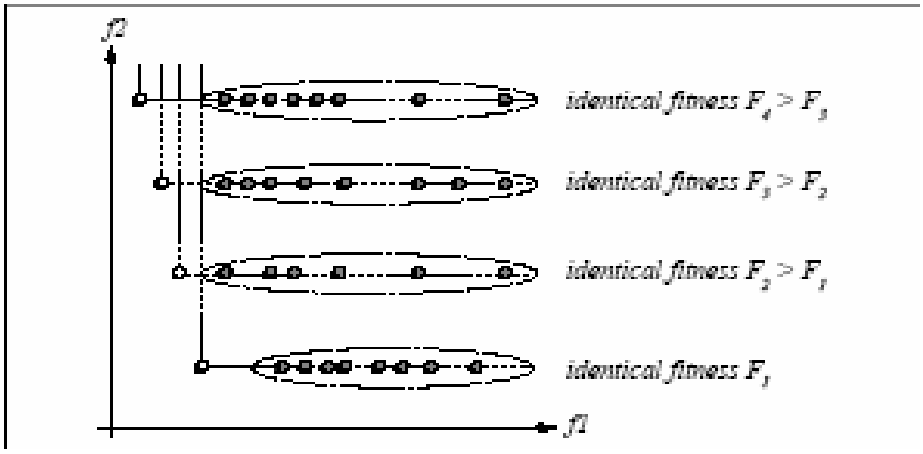


Figure 48 : SPEA2.

Le principe de SPEA2 est le suivant :

a/ Les entrées de l'algorithme sont : une population de taille N d'individus et un archive (ensemble externe) vide de taille N', aussi qu'il faut initialiser le nombre maximale de génération T.

b/ Les sorties de l'algorithme est un ensemble A des individus non dominés.

c/ Les étapes de l'algorithme :

a- Etape 1 : Initialisation de la population P0 et l'archive P'0 qui est vide au temps t=0.

b- Etape 2 : assignement de la fitness : évaluer les individus des deux ensembles (Pt et P't) au temps t.

c- Chaque individu i dans l'archive est assigné à la valeur S(i)∈ {1, 0}qui représente en même temps sa fitness F(i). S(i) = nombres de membres de la population j qui sont dominés ou égales à i toute en respectant la valeur de l'objective divisée par la taille de la population + 1.

d- La fitness F(j) d'un individu j dans la population est calculée par sommer les valeurs S(i) de tous les membres de l'archives i qui dominent ou égalent à j+1 à la fin.

e- Etape 3 : tous les membres non dominés de la population Pt et de l'archive P't sont copiés dans l'archive P't+1 et les membres dominés sont supprimés de l'archive.

f- Si la taille de l'archive P't+1 dépasse la limite prédéfinie N', alors les membres supplémentaires sont supprimés en utilisant la technique de Clustering toute en préservant les frontières de Pareto.

g- Sinon, si la taille de l'archive P't+1 est inférieur de N', alors remplir le reste de l'archive par des individus dominés de l'archive et de la population.

h- Etape 4 : si t est supérieur ou égale à T ou un autre critère d'arrêt, alors mettre P't+1 dans A.

i- Etape 5 : performer le tournoi binaire avec le remplacement dans P't+1 pour la reproduction.

j- Etape 6 : Appliquer le crossover et la mutation, et mettre P't+1 dans la population résultante. T=t+1 et aller à l'étape 2.

k- La fitness est calculée comme suit :

$S(i) = |\{j|j \in Pt + P't \text{ et } i \text{ domine } j\}|$ cardinalité

Fitness F(i) = somme S(j) tel que j domine i

$P't+1 = \{i|i \in Pt + P't \text{ et } F(i)<1\}$

☞ Algorithmes

Voilà dans l'organigramme suivant, le schéma de la méthode de SPEA2 :

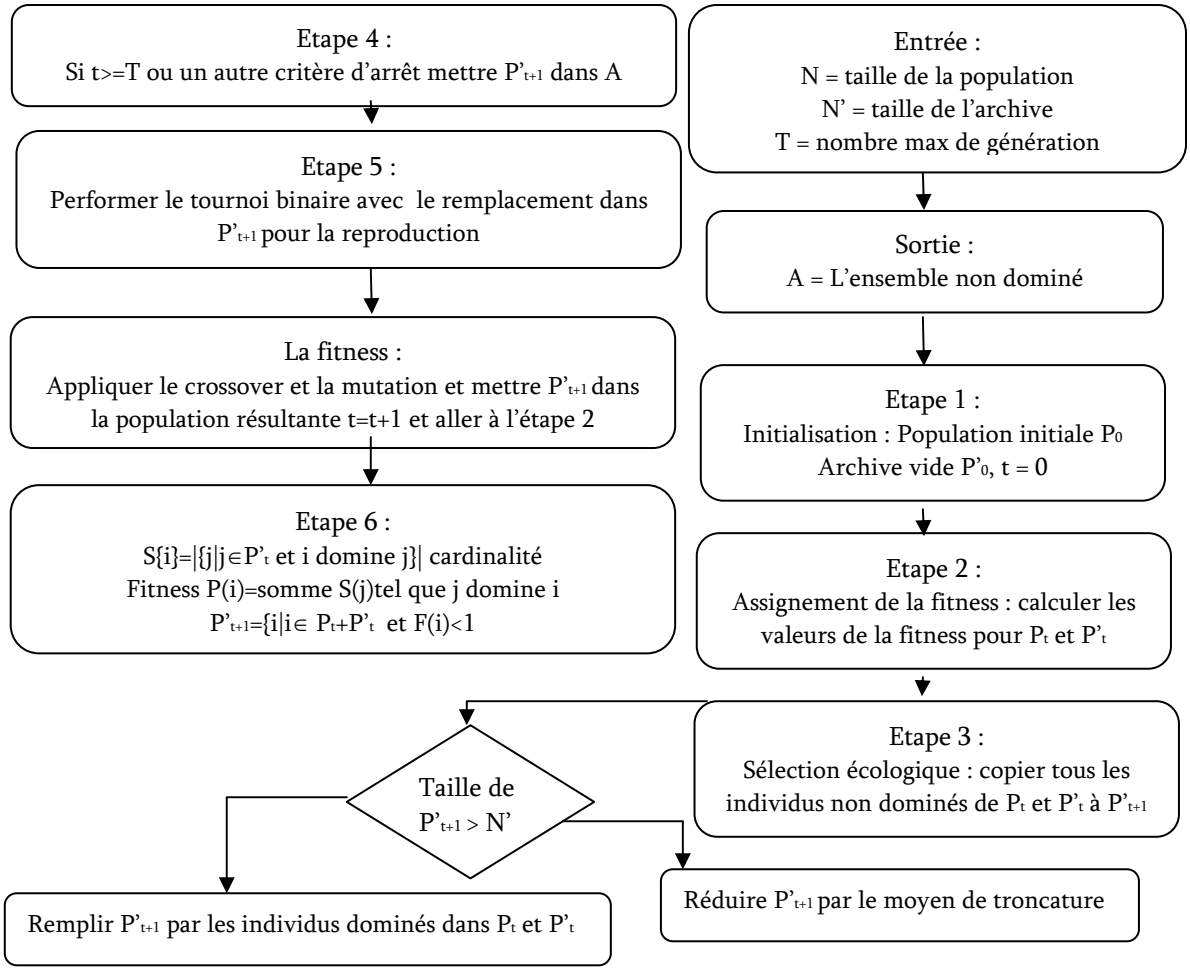


Figure 49 : Le Principe de SPEA2.

IV. Application de l'approche MOGP+SPEA2 :

IV.1 Les ressources utilisées :

Les ressources utilisées pour réaliser ce travail et pour démontrer l'utilité de la programmation génétique sur les problèmes d'optimisation multi-objectifs sont :

1- Un micro-processeur Pentium III de vitesse 600 MHertz,

2- Une RAM de 192 Moctet,

3- Le système d'exploitation XP Pack 2 (PNX),

Le langage de programmation utilisé est l'environnement de simulation « Matlab » à cause de sa capacité de simuler les cas de programmation aléatoire, aussi que sa manière de présenter les résultats sous forme de graphe.

IV.2 Le problème de TSP Bi-objectifs

Objectif	Trouver le meilleur programme permettant de résoudre le problème de TSP, c'est-à-dire trouver le meilleur chemin et réduisant le phénomène de Bloating.
L'ensemble de terminaux	City1, City2, City3, City4, City5, City6, City7, City8, City9, City10
L'ensemble de fonctions	CombinePath, SwapAndShift
La Première fitness.	La différence entre le chemin optimal et le chemin retourné par l'individu plus une pénalité si le chemin ne comporte pas toutes les villes.
La deuxième fitness	Réduire le Bloating : Adaptive Parsimony Pressure.
La fitness Brute	SPEA2 : le principe de dominance
Probabilité de mutation	Variable
Probabilité de croisement	Variable
Nombre de copies à reproduire	Variable
Le succès	Le nombre de villes visitées (existant dans le chemin final)
La taille de population	5, 10, 20, 50
Le nombre de générations	5, 10, 20, 50

La population initiale	Full, Grow, Ramped
La méthode de sélection	Roulette, Tournament
L'élitisme	Replace, Keepset, Totalelitism
Les graphes	Les 5 types de graphes

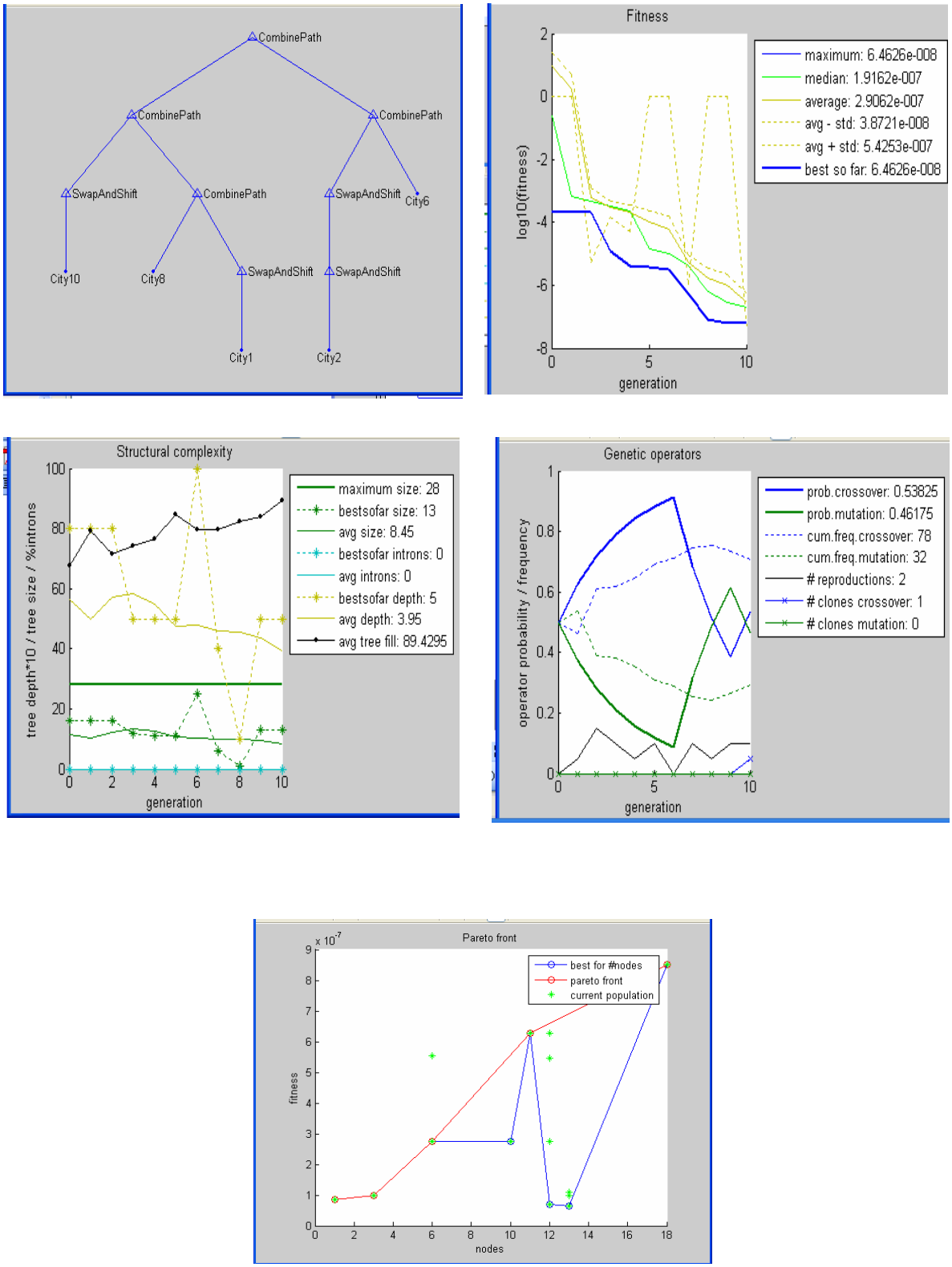
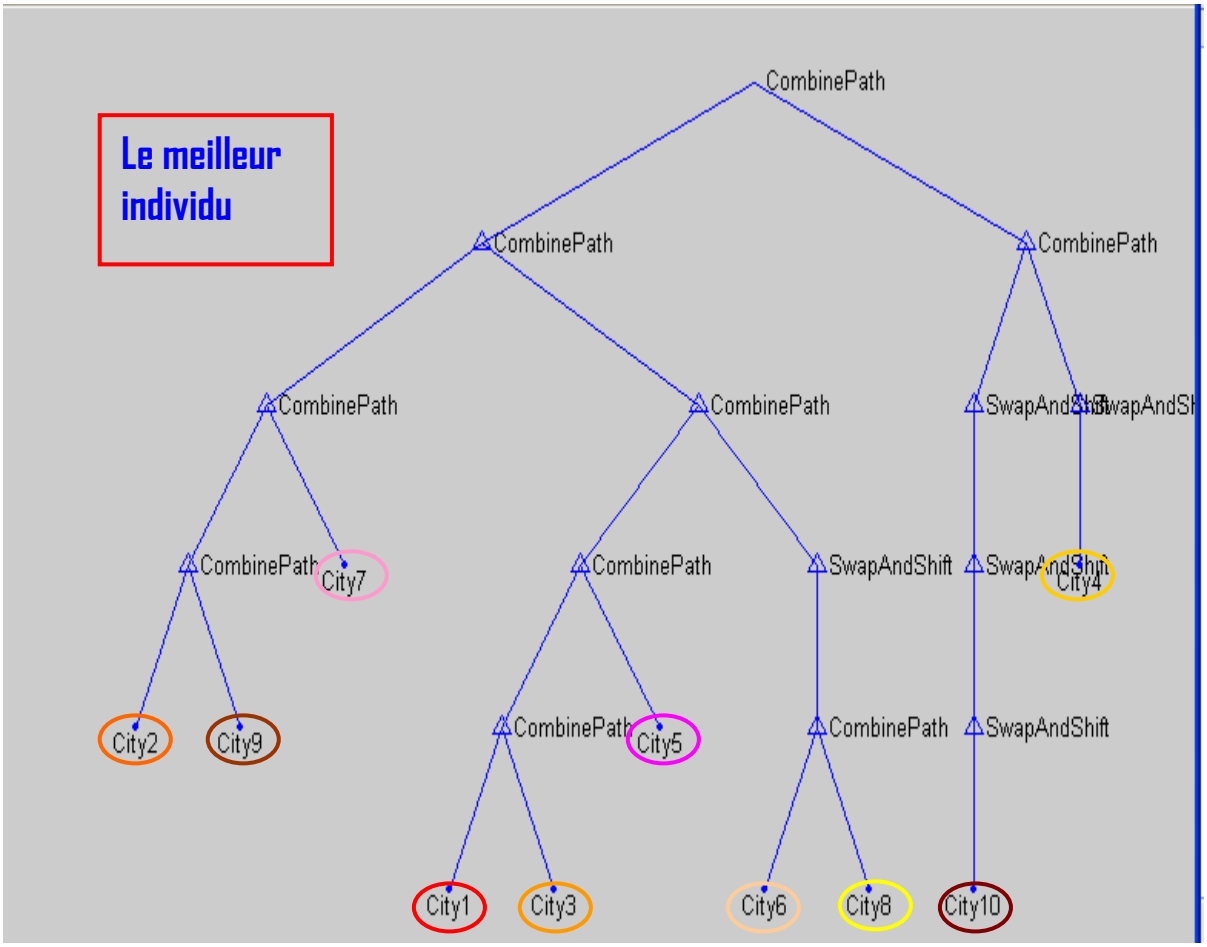


Figure 50 : Les résultats d'une exécution sur le problème de TSP.

Tout en appliquant ces paramètres au problème de TSP bi-objectifs, et après quelques éssaies on a abouti dans un essaie au meilleur programme dont on a :

- Le nombre de ville est exactement 10.
- Le meilleur chemin.
- Pas de phénomène de Bloating.



CombinePath(CombinePath(CombinePath(CombinePath(City2,Cyty3),City7),CombinePath(CombinePath(CombinePath(City1,City3),City5),SwapAndShift(CombinePath(City6,City8)))),CombinePath(SwapAndShift(SwapAndShift(SwapAndShif(City10),SwapAndShift(City4)))

IV.3 Le problème de « N-Parity »

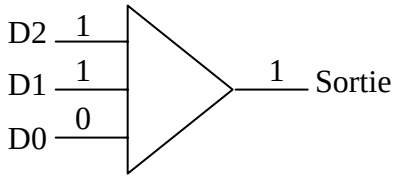
La fonction d'even-Parity est choisie parce qu'elle est utilisée généralement comme un problème de test de programmation génétique [117] et la complexity (arity =

nombre des entrées) peut être facilement adaptée pour les ressources machines valables ou pour les performances d'un algorithme.

Le problème de N-Parity est un problème complexe mais monobjectif, il est choisi parce qu'il utilisé comme un problème de test de PG par Koza et sa complexité peut être facilement adaptée soit pour les ressources machine valables soit pour les performance d'un algorithme.

La fonction booléenne even-k-parity pour k arguments booléens retourne « vraie » si un nombre pair de ces arguments booléens est « vraie » sinon elle retourne « Nil ». Les fonctions de parité sont souvent utilisées pour contrôler la précision des données binaires stockées ou transmettes dans les ordinateurs parce que un changement dans la valeur de n'importe de ses arguments perturbe la valeur de la fonction. A cause de la sensibilité de ses entrées, la fonction de parité est difficile à s'initier. L'ensemble d'entraînement consiste de tous les 2^k combinaisons possibles des entrées. L'erreur pour un individu est mesurée par le nombre des cas d'entrée pour lesquels, la valeur correcte de sortie n'est pas fournisse. La solution correcte pour le problème de la fonction de even-k-parity est trouvée quand l'erreur égale à zéro.

Le nombre d'arguments k choisi est 3 (3-Parity), dont les terminaux consistent de tous les entrées D0, D1 et D2, et pas de valeur constante utilisée. L'ensemble de fonctions consiste des quatre fonctions booléennes {AND, NAND, OR, NOR}.



Les cas de fitness dans le problème de 3-Parity :

Les cas de fitness	D2	D1	D0	Even 3-parity
0	NIL	NIL	NIL	T
1	NIL	NIL	T	NIL
2	NIL	T	NIL	NIL
3	NIL	T	T	T
4	T	NIL	NIL	NIL
5	T	NIL	T	T
6	T	T	NIL	T
7	T	T	T	NIL

Les paramètres

Objectif	Trouver le meilleur programme permettant de produire la valeur de la fonction booléenne 3-parity égale à la valeur des trois variables indépendantes en entrée, ainsi de réduire le phénomène de Bloating.
L'ensemble de terminaux	D0, D1, D2
L'ensemble de fonctions	And, Nand, Or, Nor
Les cas de la première fitness.	Toutes les combinaisons (2^3 = 8 cas) des trois arguments booléens
La première fitness	Le nombre de cas de fitness pour les quels la valeur retournée par le programme égale à la valeur correcte de la fonction booléenne de 3-parity ➔ la somme autour des 8 cas de l'erreur entre la valeur retournée par le programme et la valeur correcte de la fonction.
La deuxième fitness	Réduire le Bloating : Adaptive Parsimony Pressure.

La fitness Brute	SPEA2 : le principe de dominance
Probabilité de mutation	Variable
Probabilité de croisement	Variable
Nombre de copies à reproduire	Variable
Le succès	Un programme qui a le score maximal de nombre de cas de fitness égale aux valeurs exactes
La taille de population	5, 10, 20, 50
Le nombre de générations	5, 10, 20, 50
La population initiale	Full, Grow, Ramped
La méthode de sélection	Roulette, Tournament
L'élitisme	Replace, Keepset, Totalelitism
Les graphes	Les 5 types de graphes

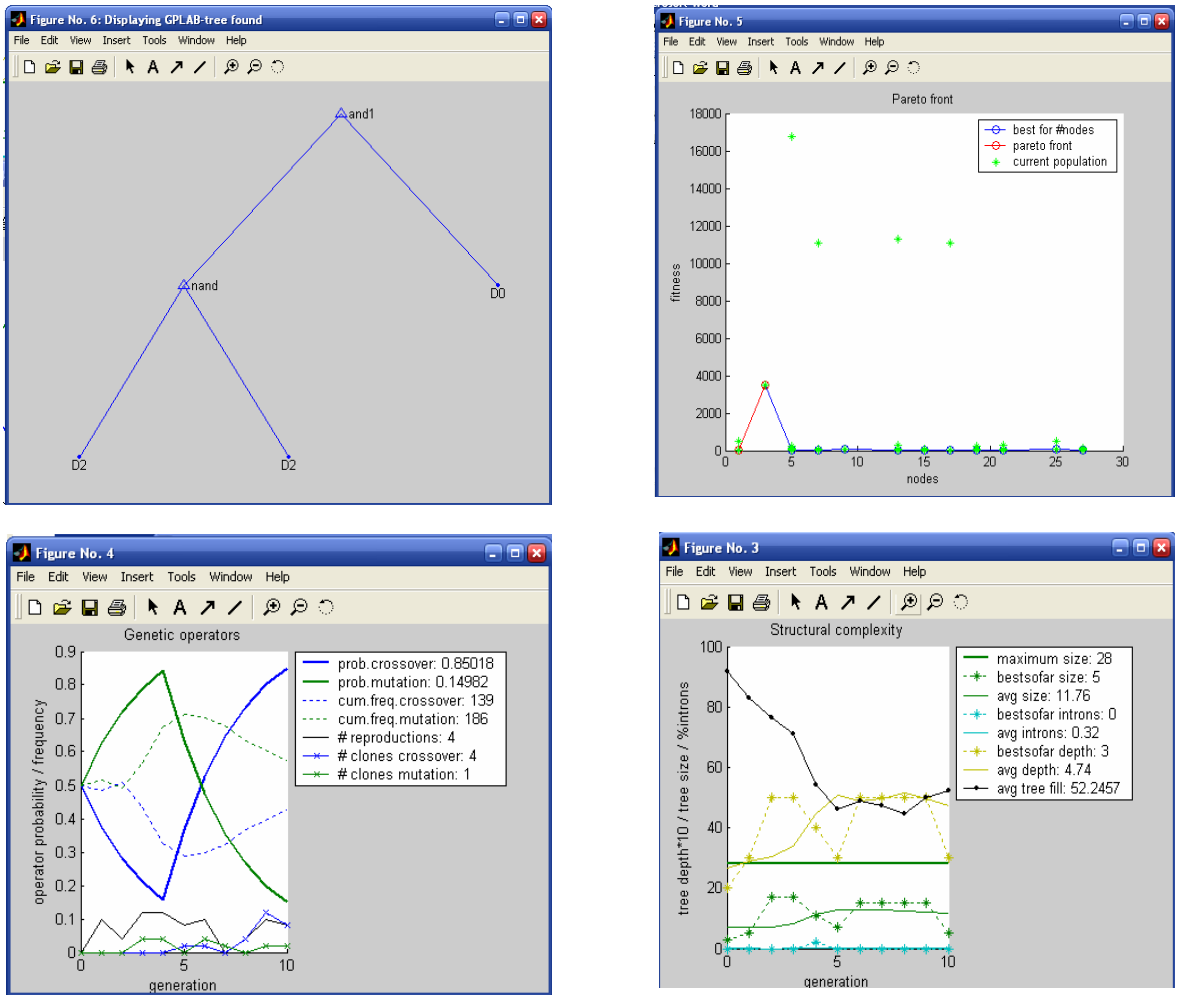
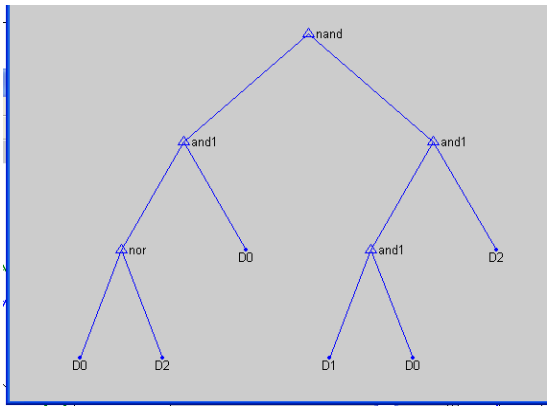
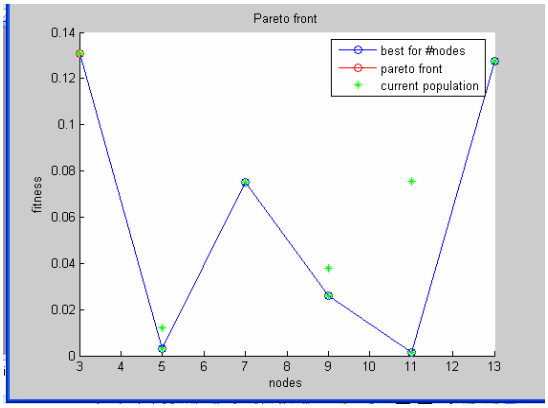


Figure 51 : Le résultat d'une exécution sur le problème 3-Parity.


nand
and1
and1
nor
D0
and1
D2
D0
D2
D1
D0


Pareto front
best for #nodes
pareto front
current population
fitness
nodes

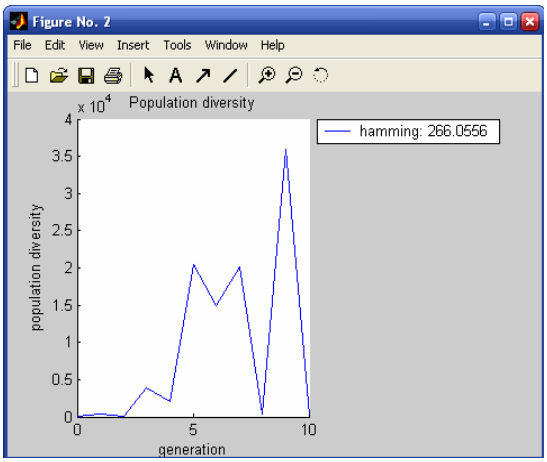


Figure No. 2
File Edit View Insert Tools Window Help
Population diversity
hamming: 266.0556
population diversity
generation

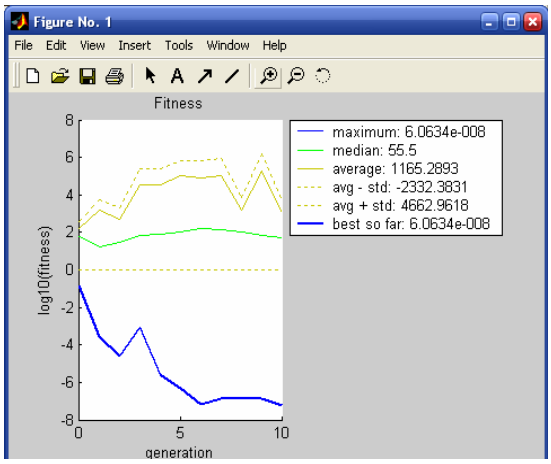


Figure No. 1
File Edit View Insert Tools Window Help
Fitness
maximum: 6.0634e-008
median: 55.5
average: 1165.2893
avg - std: -2332.3831
avg + std: 4662.9618
best so far: 6.0634e-008
log10(fitness)
generation

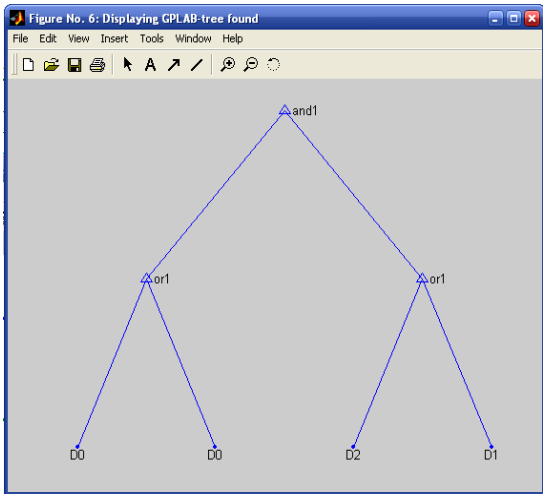


Figure No. 6: Displaying GPLAB-tree found
File Edit View Insert Tools Window Help
and1
or1
or1
D0
D0
D2
D1

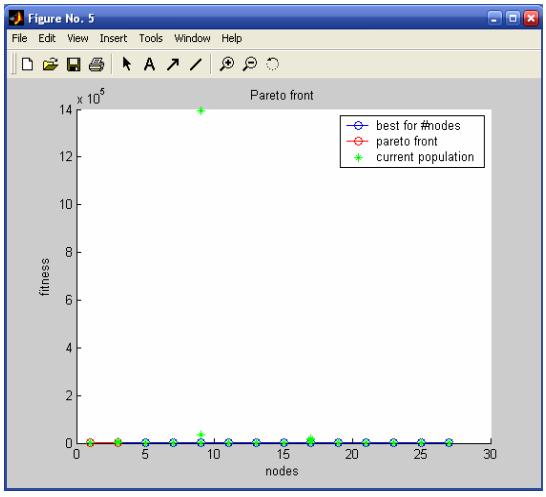


Figure No. 5
File Edit View Insert Tools Window Help
Pareto front
best for #nodes
pareto front
current population
fitness
nodes

V. Conclusion :

D'après les résultats expérimentaux, nous concluons que le phénomène de Bloating a un grand effet sur la convergence du problème vers les meilleures solutions (comme apparaît dans les arbres affichés dans la première partie de ce chapitre). Quand on a réglé ce problème, on obtient le résultat suivant : la convergence de problème vers les solutions optimales est mieux que dans le cas où l'arbre agrandisse d'une manière monstre.

Nous avons appliqué notre approche à deux grands problèmes NP-Hard « TSP et 3-Prity », mais cette approche est générale à tous les problèmes complexes quelque soit leurs définitions parce que la fitness qui contrôle le bloating est inchangeable et reste fixe dans tous les problèmes comme : le problème de Sac à Dos, Coloring Graph, N-Queens, Traitement d'image, …

Conclusion générale :

Dans ce mémoire, nous avons essayé de montrer que la programmation génétique est une bonne voie pour trouver des programmes machine capables de résoudre des problèmes complexes dans l'industrie, la robotique, les réseaux industriels, le traitement d'images, d'une autre façon, tout problème du monde réel simple ou complexe, soit d'optimisation ou de simulation.

L'avantage qui a laissé ce type d'algorithme d'avoir cette utilité c'est que la programmation génétique produit des programmes machine non des solutions fixes, c'est-à-dire pour un problème quelconque on cherche de trouver un programme qui donne des solutions à l'opposé des autres approches qui nécessitent parmi ses entrées la connaissance de programme à suivre pour trouver les solutions (d'autre terme, il faut offrir le programme et les données initiales pour avoir les résultats). Avec la programmation génétique, il faut offrir les opérations de bases ainsi que les données nécessaires et le processus de PG va générer un programme à partir de ces entrées, ce programme à son tour va présenter un programme machine qui va être exécuté pour nous générer les résultats voulus.

Parmi les problèmes réels, on trouve les problèmes d'optimisation multiobjectifs qui ont démontré leur difficulté d'être résolus, les chercheurs ont appliqué plusieurs types de méthodes (commençant par les méthodes mathématiques aux heuristiques). Nous dans ce mémoire, nous avons conçu une application (Matlab) permettant de résoudre ce genre de problèmes par la programmation génétique comme un premier pas vers une recherche vaste dans le domaine d'intelligence artificielle dont les chercheurs cherchent de concevoir un ordinateur intelligent qui programme lui-même sans le besoin de l'utilisateur ainsi que dans le domaine de l'informatique quantique dont aussi les utilisateurs cherchent de trouver des ordinateurs qui obéissent aux principes de la mécanique quantique pour être plus rapide et plus efficace.

Nous avons résolu le fameux problème rencontré dans l'utilisation de la programmation génétique connu par le Bloating, ce phénomène qui empêche la convergence du processus vers les meilleurs programmes tout en appliquant les principes de l'optimisation multiobjectifs.

Finalement, nous concluons par les perspectives suivantes : de généraliser cette application à tout genre de problèmes surtout les problèmes d'optimisation qui ont plus de deux objectifs, aussi que l'utiliser dans le domaine de l'informatique quantique pour créer des circuits électroniques quantiques.

Bibliographie

[01] : Rémy-Robert, Alexandre Joseph ; Systèmes Interactifs d'Aide à l'Élaboration de Plannings de Travail de Personnel Contraintes, Aide à la Décision, Représentation Combinatoire des Préférences, Équité et Résolution par Décomposition Arborescente et par Consistance ; 07 Novembre 2003 ; Université Joseph Fourier-Grenoble1, Science et Géographie ; Page 28.

[02] : Yann Collette, Patrick Siarry ; Optimisation Multiobjectif ; Septembre 2002 ; Edition Eyrolles, 75240 Paris Cedex 05, France ; Page 1, 1, 2, 17, 18, 19, 21, 22, 23, 41, 42, 43, 44, 46.

[03] : Inconnu.

[04] : Alain Berro ; Optimisation Multiobjectif et Stratégie d'évolution en environnement Dynamique (thèse de Doctorat) ; 18 Décembre 2001 ; Université des sciences sociales ToulouseI ; page 14, 27, 29.

[05] : M. Caldora Costa ; Optimisation de Diapositifs Electromagnétiques dans un Contexte d'analyse par la Méthode des Eléments Finis ; 17 Juillet 2001 ; L.E.G-Laboratoire d'électrotechnique de Grenoble ; Page 10.

[06] : Johan Andersson ; Multiobjective optimization in Engineering Design, Applications to Fluid Power System (thèse de Doctorat); 2001; Université de Linköping, Sweden; Page 29.

[07] : Vincent Barichard ; Approches Hybrides pour les Problèmes Multiobjectifs (thèse de Doctorat) ; 24 Novembre 2003 ; Université d'Angers ; Page 9, 11,37, 40.

[08] : Mostapha Benhenda ; http://www.eleves.ens.fr/home/benhenda.html

[09] : ????; http://www.al.lu/materials/pascal/chaps/chap02.html.

[10]: Tollari Sabrina ; 2003 ; http://sis.univ-tln.fr/~tollari/TER/AlgoGen1/node4.html

[11] : Vincent Barichard & Jin-Kao Hoa ; Un Algorithme Hybride pour le Problème de Sac à Dos Multi-objectifs ; 2002 ; Article d'Actes JNPC'02 ; Page 1.

[12] : I. Othmani ; Optimisation multicritère : Fondements et Concepts. Thèse de PHD ; Université de Grenoble ; 1998.

[13] : M. Ehrgott; Multicriteria optimization; In Lecture Notes in Economics and Mathematical Systems, volume 491; Springer; 2000.

[14] : Elghazali Talbi ; Méthodes d'Optimisation Avancée ; Page 18, 19.

[15] : S. Martello, P. Toth; An upper bound for the zero-one knapsack problem and a branch and bound algorithm; European Journal of Operational Research, Vol. 1, pp. 169-175; 1977.

[16] : Inconnu.

[17] : A. Ramani, F. A. Aloul, I. L. Markov et K. A. Sakallah ; Breaking Instance-Independent Symmetries in Exact Graph Coloring; 2004; Page 2.

[18] : G. J. Chaitin et. Al; Register allocation via coloring; in Computer Languages; 6:47-57; 1981.

[19] : M. Yokoo, T. Suyama, et H. Sawada ; Solving satisfiability problems using field programmable gate arrays: first results; In 2nd International Conference on Principles and Practice of Constraint Programming (CP'96); page 497–509, Springer; 1996.

[20] : T. Suyama, M. Yokoo, H. Sawada, et A. Nagoya ; Solving satisfiability problems using reconfigurable hardware; IEEE Transactions on VLSI Systems, 9(1):109–116; 2001.

[21] : P. Zhong, P. Ashar, S. Malik, et M. Martonosi ; Using reconfigurable computing techniques to accelerate problems in the CAD domain: a case study with Boolean satisfiability; In 35th Design Automation Conference (DAC), page 194–199; 1998.

[22] : M. Davis et H. Putnam ; A computing procedure for quantification theory; Journal of the ACM, (7):201–215; 1960.

[23] : P. Zhong, M. Martonosi, P. Ashar, et S. Malik ; Using configurable computing to accelerate Boolean satisfiability; IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, 18(6):861–868; 1999.

[24] : M. Platzner et G. D. Micheli ; Acceleration of satisfiability algorithms by reconfigurable hardware; In 8th International Workshop on Field-Programmable Logic and Applications (FPL'98), page 69–78; Springer; 1998.

[25] : O. Mencer, M. Platzner, M. Morf, et M. J. Flynn; Object-oriented domain specific compilers for programming FPGAs; IEEE Transactions on VLSI Systems, 9(1):205–210, 2001.

[26] : M. Abramovici et J. T. De Sousa ; A SAT solver using reconfigurable hardware and virtual logic; Journal of Automated Reasoning, 24(1–2):5–36; 2000.

[27] : M. Abramovici et D. Saab ; Satisfiability on reconfigurable hardware; In 7th International Workshop on Field-programmable Logic and Applications (FPL'97), page 448–456; Springer; 1997.

[28] : C. Blessl et M. Platzner ; Instance-Specific Accelerators for Minimum Covering ; In the Journal of Supercomputing , 26, 109-129; 2003; Page 110, 111, 112.

[29] : Artificial Intelligence: Introduction to local search; Page 9.

[30] : S. Prestwich et C. Quirke ; Boolean and Pseudo-Boolean Models for Scheduling; Page 1.

[31] : C. M. Li; Integrating Equivalency Reasoning into Davis-Putnam Procedure; *Seventeenth National Conference on Artificial Intelligence*; Austin, Texas, USA; 2000; page 291–296.

[32] : P. Baumgartner et F. Massacci; The Taming of the (X)OR; *First International Conference on Computational Logic, Stream on Automated Deduction: Putting Theory into Practice, Lecture Notes in Artificial Intelligence* vol. 1861; Springer-Verlag; 2000; page 508–522.

[33] : J. Whittemore, J. Kim et K. Sakallah; SATIRE: A New Incremental Satisfiability Engine; *Thirty-Eighth Design Automation Conference;* 2001; page 542–545.

[34] : M. R. Dransfield, V. W. Marek et M. Truszczynski. Satisfiability and van der Waerden Numbers; *Sixth International Conference on Theory and Applications of Satisfiability Testing;* Portofino, Italy; 2003; page 325-336.

[35] : H. E. Dixon, M. L. Ginsberg et A. J. Parkes; Likely Near-term Advances in SAT Solvers; *Workshop on Microprocessor Test and Verification*; Austin, Texas, USA; 2002.

[36] : S.-Y. Shin, B.-T. Zhang et S.-S. Jun; Solving Traveling Salesman Problems Using Molecular Programming; 1999.

[37] : D. A. Van Veldhuizen; Multiobjective Evolutionary Algorithms: Classifications, Analyses and New Innovation, Ph. D.; Graduate School of Engineering; Air Force Institute of Technology; Wright Patterson AFB, Ohio, USA; Janvier; 1999.

[38] : C. A. Coello Coello; An apdated survey of GA based multiobjective optimisation techniques; rapport technique; Lania-RD-98-08; Xalapa, Veracruz, Mexico; décembre;1998; http://www.lania.mx/~ccoello/EMOO.

[39] : L. R. Keeney et H. Raiffa; Decisions with multiple objectives: preferences and value tradeoff; edition Combridge; University Press; 1993.

[40] : K. M. Miettinen; Nonlinear Multiobjective Optimization; edition Kluwer; academic publisher; 1999.

[41] : Y. Y. Haimes, W. A. Hall et H. T. Freedman; Multiobjevtive optimization in Water Resouces Systems; editions Elsevier scientific publication company; 1975.

[42] : H. Eschnauer, J. Koski et A. Osyczka; Multicriteria design optimization: procedures and applications; Springer-Varleg; 1990.

[43] : J. A. Nelder et R. Mead; A Simpler Method For function minimiszation; Computer Journal, volume 7; page 308-313; 1965.

[44] : L. A. Zadeh, Fuzzy Sets; information and control; volume 8, pages 338-353; 1965.

[45] : M. Sakawa et H. Yano; an interactive fuzzy satisficing method for multiobective linear programming problems with fuzzy parameters; fuzzy sets and systems, volume 28, pages 129-144; 1988.

[46] : S. M. Sait et H. Youssef; iterative computer algorithms with applications in ingineering: solving combinatorial optimization problems, IEEE computers society, 1999.

[47] : E. Bonomi et J.-L. Lutton ; le recuit simulé pour la science ; numéro 129, pages 68-77 ; Juillet 1988.

[48] : P. Angrand et X. Mouny ; une méthode originale d'optimisation multiobjectif ; Note interne EDF-DER numéro HT-14/97/035/A ; mars ; 1998.

[49] : E. L. Ulunga, J. Taghem, P. H. Fortemps et D. Tuyttens; MOSA Method : A Tool For Solving Multiobjective Combinatoial Optimization Problems; Journal of Multicriteria Decision Analysis; volume 8, numéro 4, pages 221-236; 1999.

[50] : F. Glover; Artificial Intelligence, heuristic frameworks and tabu search; Managerial and decision economics; volume 11, pages 365-375; 1990.

[51] : D. De Werra; heuristics for graph coloring; computing Suppl; volume 7, pages 191-208; 1990.

[52] : D. Gaertner ; Natural Algorithms for Optimisation Problems, Outsourcing Report; 16 Janvier 2004.

[53] : C. Gagné, M. Gravel et W. L. Price ; Optimisation par Colonie de Fourmis pour un Problème d'Ordonnancement Industriel avec un Temps de Réglage Dépendant de la Séquence ; 3ème Conférence francophone de Modélisation et Simulation « Conception, Analyse et Gestion de Systèmes Industriels » ; MOSIM'01, Troyes, France ; 2001.

[54] : M. Dorigo ; From Real to Artificial Ants, Chapitre 1 ; 1998.

[55] : C. E. Mariano, et E, M. Morales; A Multiple Objective Ant-Q Algorithm for the Design of Water Distribution Irrigation Networks; Technical Report HC-9904; Mexico Institut.

[56] : L. M .Gambardella, E. Taillard et G. Agazzi; MACS-VRPTW: A multiple ant colony system for vehicule routing problems with time windows; *D. Corne. M. Dorigo et F. Glover Editors;* New ideas in optimization; McGraw-Hill, page 63-76.

[57] : S. Iredi, D. Merkle et M. Middendorf; *Bi-Criterion Optimizalion with Multi Colony Am Algorithms;* First International Conference (FMO'Ol); Zurich, Springer Verlag, page 359-372.

[58] : C. Gagné, M. Gravel et W. L. Price ; Optimisation Multi-objectifs à L'aide d'un Algorithme de Colonie de Fourmis ; Infor, Vol 42, no. 1 Février 2004.

[59] : X. Delorme ; Modélisation et résolution des Problèmes Liés à l'exploitation d'infrastructures ferroviaires, thèse de Doctorat ; l'université de Valenciennes et Hainaut-Cambrésis ;10 Décembre 2003.

[60] : Thomas A. Féo et Mauricio G.C. Resende ; A probabilistic heuristic for a computationally difficult set covering problem; Operations Research Letters, 8 :67–71; 1989.

[61] : M. O'Keeffe et M. Ô Cinnéide ; A Stochastic Approach to automated Design Improvement; 2003.

[62] : Le Recuit Simulé ; 17 Novembre 2004.

[63] : S. Kemmoé , L. Deroussi, M. Gourgand et A. Quilliot ; Des Essaims Particulaires Efficaces Pour l'optimisation Combinatoire ; Congrès du GDR MACS, Pôle STP ; Octobre 2004.

[64] : K. E. Parsopoulos, D. K. Tasoulis et M. N. Vrahatis ; Multiobjective Optimization Using Parallel Vector Evaluated Particule Swarm Optimization ; 2004.

[65] : J. Yen, J. Liao, B. Lee, et D. Randolph; A hybrid approch to modeling metabolic systems using a genetic algorithm and simplex method; *IEEE Transaction on systems, man and cybernetics*; vol. 28, page 173-191; 1998.

[66] : J. Andersson ; A Survey for Multiobjective Optimization in engineering Design; Rapport technique, LiTH-IKP-R-1097; 2000

[67] : Anne Spalanzani ; Algorithmes évolutionnaires pour l'étude de la robustesse des systèmes de reconnaissance automatique de la parole ; Thèse de doctorat, 28 Octobre 1999 ; Université Joseph Fourier, Grenoble I ; page 45, 47.

[68] : Hans-Paul Schwefel ; On the Evolution of Evolutionary Computer; Université de Dortmund, Allemagne; 1994.

[69] : Jean-Sébastien Lacroix, Stéphane Terrade ; Algorithmes Génétiques ; 17 Novembre 2004 ; Ecole Polytechnique Montréal.

[70] : Miroslav GREGAN ; Applications d'algorithmes génétiques (AGs) ; Cours au sein d'EIDV, Ecole d'ingénieurs du Cancon Du Vaud ; 13 Janvier 2005.

[71] : Eckart Zitzler ; Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications; Thèse de Doctorat; 11 Novembre 1999; Institut Fédéral de Technologie, Zurich, Suisse; Page 14.

[72] : Christophe Bontemps ; Principes Mathématiques et Utilisations des Algorithmes Génétiques ; 18 Novembre 1995.

[73] : inconnu. (**Web**)

[74] : J. Heitkötter, D. Beasley ; The Hitch-Hiker's Guide to Evolutionary Computation; FAQ for comp.ai.genetic; 20 Septembre 2000.

[75]: B. Belew, J. McInerney et N. Schraudolph; Evolving Networks : Using the Genetic Algorithms with Connectionist Learning; CSE Technical Report CS90-174; Computer Science, UCSD; 1990.

[76] : Goldberg D.E., Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley, Massachusetts, 1989.

[77] : Samuel BERNARD ; Algorithmes Evolutionnaires ; 2003. (Web).

[78] : I. Charon, A. Germa, O. Hudry, Méthodes d'Optimisation Combinatoire ; 1996.

[79] : D. Goldberg ; Algorithmes génétiques ; Editions Addison Wesly, Juin, 1994.

[80] : T. Vallée ; Présentation des algorithmes génétiques et de leurs applications en économie ; IFReDE-E3i. Université Montesquieu Bordeaux IV ; Décembre 2003.

[81] : K. Deb; An Overview of Multi-objective Evolutionnary Algorithms; Journée JET, Copie de Transparents; Mai 1999.

[82] : N. Srinivas, K. Deb; Multiobjective Optimization Using Non-Dominated Sorting in Genetic Algorithms; Rapport technique; Département d'Ingénierie Technique; L'Institut Indien de Technologie; Kanput, Inde; 1993.

[83] : J. Horn, N. Nafpliotis, D. E. Goldberg; Multiobjective Optimization Using the Niched Pareto Genetic Algorithm; Rapport technique; IlliGAI Report 93005; Université de L'illinois; Urbana, Illinois; 1993.

[84] : K. Deb; Multi-objective Using Evolutionary Algorithms; John Wiley & Sons; 2001.

[85] : A. Augugliaro, L. Dusonchet, E. Riva Sanseverino ; Multiobjective service restoration in distribution networks using an evolutionary approach and fuzzy sets; Electrical Power and Energy Systems 22 (2000) 103–110; 1999.

[86] : B. Ahiod, I. Berrada. I. Kassou; Deux Méthodes de Recherche Locale pour Résoudre un Problème d'Horaire du Personnel Infirmier dans un établissement Hospitalier ; 1998.

[87] : D.E. Goldberg, J. Richardson; Genetic Algorithms with Sharing for Multimodal Function Optimization; Dans Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms; pages 41-49; Lawrence Erlbaum; 1987.

[88] : K.A. De Jong; An analysis of the Behaviour of a Class of Genetic Adaptive Systems; Thèse de Doctorat; Universitie de Michigan; 1975.

[89] : T. Blickle; Theory of Evolutionary Algorithms and Application to System Synthesis; Thèse de Doctorat; L'institut Fédéral de Technologie de Suisse, Zurich; 1996.

[90] : E. Zitzler, L. Thiele; Multiobjective Optimization Using Evolutionary Algorithms: a Comparative Case Study; Dans la Lecture des notes des Sciences d'Ordinateur; pages 292-301; Printemps, 1998.

[91] : E. Zitzler; Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications; Thèse de Doctorat; L'institut Fédéral de Technologie de Suisse, Zurich; 1999.

[92] : K. Deb, S. Agrawal, A. Pratap, T. Meyarivan; A Fast and Elitist Multiobjective Genetic Algorithm for Multi-Objective Optimization: NSGA-II; Dans le Proceedings of the Parallel Problem Solving from Nature VI (PPSN-VI); pages 849-858; 2000.

[93] : E. Zitzler, L. Thiele; Multiobjective Evolutionary Algorithms: a Comparative Case Study And the Strength Pareto Approach; IEEE Transactions on Evolutionary Computation 3; 257-271; 1999.

[94] : John R. Koza; Genetic Programming: On the Programming of Computers by Natural Selection; MIT Press Cambridge, MA; USA 1992; page 86.

[95] : W. B. Langdon; genetic programming and data structures; Departement de science d'ordinateurs, université de Collège London, 27 septembre 1996.

[96] : J. R. Koza; Genetic programming II: automatic discovery of reusable programs; MIT press, Cambridge, Massachusetts; Mai 1994.

[97] : John H. Holland; Adaptation in Natural and Artificial Systems An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence; MIT Press, 1992; Première publication par l'université de Michigan Press, 1975.

[98]: G. Syswerda; Uniform crossover in genetic algorithms; In J. David Schaffer; Procédé de la troisième conférence international sur les algorithmes génétiques ; pages 2-9 ; université de George Mason ; 4-7 Juin, 1989 ; Morgan Kaufman.

[99] : Carlos M. Fonseca, Peter J. Fleming ; Genetic algorithms for multiobjective optimization : Formulation, discussion and generalization ; dans Stephanie Forrest, éditeur, Procédé de la 5ème conférence international sur les algorithmes

génétiques, ICGA-93, pages 416-423, université d'Illinois à Urbana-Champaign ; 17-21 Juillet 1993 ; Morgan Kaufman.

[100] : K. Deb, S. Agrawal, A. Pratap, et T. Meyarivan; A fast and elitist multiobjective genetic algorithm: NSGA-II; IEEE Transactions sur les algorithmes évolutionnaires, 6(2):182.197 ; Avril 2002.

[101] : K. Rodriguez-Vazquez, Carlos M. Fonseca, et Peter J. Fleming ; Multiobjective genetic programming: A nonlinear system identification application; dans Papersat de l'interruption retard de la conférence de Genetic Programming 1997, pages 207-212 ; 1997.

[102] : Gregory J. Barlow; design of autonomous navigation controllers for unmanned aerial vehicles using multi-objective genetic programming; Mars 2004.

[103] : S. Luke, Issues in scaling genetic programming: breeding strategies, tree generation and code bloat; 2000

[104] : L. Spector, H. Barnum, H. J. Bernstein et N. Swamy; Quantumcomputing applications of genetic programming. Dans L. Spector, W. B. Langdon, O'Reilly; U.-M. et Angeline, P. J., eds., Advances in Genetic Programming 3. Cambridge,MA, USA: MIT Press. 135–160; 1999.

[105] : John R. Koza, L. Spector; Automated invention by means of genetic programming ; aaai-2004 tutorial, san jose, Samedi, Juillet 25; 2004.

[106] : W. B. Langdon, R. Poli; Fitness Causes Bloat; Rapport technique CSRP-97-09; Université de Birmingham, UK ; 1997.

[107] : T. Soule ; Code Growth in Genetic Programming ; Mémoire Présenté dans la réalisation partielle des exigences pour le degré de docteur de philosophie ; Université de Idaho, 15 Mai 1998.

[108] : Complexity compression and evolution ; Dans L. ESHELMAN, Ed., les algorithms génétiques : Procédé de la 6ème conference international (ICGA95), pages 310– 317 ; Pittsburgh, PA, USA : Morgan Kaufmann ; 1995.

[109] : Accurate replication in genetic programming ; Dans L. ESHELMAN, Ed., les algorithms génétiques : Procédé de la 6ème conference international (ICGA95), pages 303–309, Pittsburgh, PA, USA : Morgan Kaufmann ; 1995.

[110] : Genetic programming and redundancy; Dans J. HOPF, Ed., *Genetic Algorithms Workshop à KI-94*, pages 33–38 : Max-Planck-Institut d'informatique; 1994.

[111] : Exons and code growth in genetic programming; Dans J. A. F. ET AL., Ed., *EuroGP 2002*, volume 2278 de *LNCS*, pages 142–151 : Springer-Verlag; 2002.

[112] : O. Teytaud, S. Gelly, N. Bredeche, M. Schoenauer; Apprentissage statistique et programmation génétique: la Croissance du Code est-elle inévitable ?; CAP2005 ; 2005.

[113] : John R. Koza, Riccardo Poli; A Genetic Programming Tutorial; Chapitre 8 d'un tutorial de PG, 2003.

[114] : B. Swope ; Evolution of a Path Generator for a Round-Trip Symmetric Traveling Salesperson Problem Using Genetic Programming; Stanford Mechanical Engineering Department Stanford University Stanford, California 94305.

[115] : T. Blickle et L. Thiele ; Genetic programming and redundancy ; Dans J. Hopf; éditeur; Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI-94, Saarbrücken); pages 33–38; 1994.

[116] : W. Banzhaf, Frank D. Francone, Robert E. Keller, et P. Nordin. Genetic Programming: An Introduction; Morgan Kaufmann, San Francisco; CA; 1998.

[117] : John R. Koza ; Genetic Programming II: Automatic Discovery of Reusable Programs ; MIT Press ; Cambridge ; Massachusetts ; 1994.

[118] : T. Soule et James A. Foster ; Effects of code growth and parsimony pressure on populations in genetic programming ; Evoluationary Computation ; 6(4):293–309 ; 1999.

[119] : T. Kalganova et J. F. Miller ; Evolving more efficient digital circuits by allowing circuit layout evolution and multiobjective fitness ; Dans A.Stoica, D. Keymeulen, et J. Lohn éditeurs ; Proceedings of the 1st NASA/DoD Workshop sur Evolvable Hardware (EH'99) ; pages 54–63 ; Piscataway ; NJ ; 1999, 1999. IEEE Computer Society Press.

[120] : Byoung-Tak Zhang et H. Mühlenbein ; Balancing accuracy and parsimony in genetic programming ; Evoluationary Computation ; 3(1):17–38 ; 1995.

[121] : E. Zitzler, K. Deb, et L. Thiele ; Comparison of multiobjective evolutionary algorithms: Empirical results ; Evolutionary Computation ; 8(2):173–195 ; 2000.

[122] : D. W. Corne, J. D. Knowles, et M. J. Oates ; The pareto envelope-based selection algorithm for multiobjective optimisation ; Dans M. Schoenauer et al., éditeur, PPSN VI ; pages 839– 848 ; Berlin, 2000 ; Springer.

[123] : M. Lahanas, N. Milickovic, D. Baltas, et N. Zamboglou ; Application of multiobjective evolutionary algorithms for dose optimization problems in brachytherapy ; Dans E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, et David W. Corne ; éditeurs, Proc. De la première international Conférence sur « evolutionary multi-criterion optimization (EMO'01) » ; volume 1993 de Lecture Notes in Computer Science ; pages 575–588 ; Berlin ; 2001. Springer-Verlag.