



**UNIVERSITE MOHAMED BOUDIAF - M'SILA**  
**FACULTE DE MATHEMATIQUES ET**  
**D'INFORMATIQUE**



**DEPARTEMENT D'INFORMATIQUE**

**MEMOIRE de fin d'études**  
**Présenté pour l'obtention du diplôme de MASTER**  
**Domaine : Mathématiques et Informatique**  
**Filière : Informatique**  
**Spécialité : Réseaux**

**Par: HASSANI khaoula**

**SUJET**

**Impact du scaling et du sharing sur l'efficacité des  
algorithmes génétiques**

**Soutenu publiquement le :     /     /2016 devant le jury composé de :**

.....  
**HEMMAK Allaoua**  
.....  
.....

Université de M'sila  
Université de M'sila  
Université de M'sila  
Université de M'sila

**Président**  
**Rapporteur**  
**Examineur**  
**Examineur**

**Promotion : 2015 /2016**

## LISTE DES FIGURES

Figure 1.1 : Placement optimal des pièces .....	8
Figure 1.2 : Fonctionnement des algorithmes génétiques.....	13
Figure 1.3 : Codage des solutions.....	14
Figure 1.4 : Sélection par roulette .....	15
Figure 1.5 : Sélection par tournoi .....	16
Figure 1.6 : Croisement à un point .....	16
Figure 1.7 : Croisement à deux points .....	17
Figure 1.8 : Une mutation .....	17
Figure 1.9 : Exemple où les sélections classiques risquent de ne reproduire qu'un individu.....	19
Figure 1.10 : Fonction de scaling exponentielle .....	21
Figure 1.11 : Allure de l'évolution de k en fonction des générations .....	22
Figure 1.12 : Objectif du sharing.....	23
Figure 1.13 : Allure de la fonction de sharing en fonction de son intensité.....	24
Figure 3.14 : modélisation du graphe sous forme de matrice.....	41
Figure 3.15 : codage d'une solution (ensemble de villes) dans un tableau.....	42
Figure 3.16 : exemple de croisement.....	43
Figure 3.17 : exemple de mutation.....	44
Figure 4.1 : influence des paramètres de sharing et scalling.....	55
Figure 4.2 : génération des solutions optimales par la taille de population .....	56
Figure 4.3 : l'influence du sharing.....	57
Figure 4.4 : l'influence du scaling .....	58

## LISTE DES TABLEAUX

Tableau 2.1 : Récapitulatif des travaux sur les AGs.....	34
Tableau 3.1 : Nombre de solutions possibles.....	48
Tableau 3.2: Nombre de possibilités de chemins et temps de calcul en fonction du nombre de villes.....	48
Tableau 4.1 : Expérimentation de sharing =100 et scaling =1.....	53
Tableau 4.2 : Expérimentation sans scaling et sans sharing.....	54

## Table des matières

<b>Table des matières.....</b>	<b>i</b>
<b>Liste des figures.....</b>	<b>i</b>
<b>Liste des tableaux.....</b>	<b>i</b>
<b>INTRODUCTION GENERALE.....</b>	<b>2</b>

### CHAPITRE 1 : Théorie des algorithmes génétiques

1. Introduction .....	5
2. Problème d'optimisation combinatoire.....	5
2.1. Définition : .....	5
2.2. Résolution d'un problème d'optimisation combinatoire :.....	5
3. Les méthodes d'optimisation combinatoire : .....	6
3.1. Les méthodes exactes :.....	6
3.1.1. La programmation dynamique : .....	6
3.1.2. La programmation linéaire : .....	7
3.1.3. La méthode de branch and bound : .....	7
3.2. Les méthodes approchées:.....	7
3.2.1. Les méthodes Heuristiques : .....	8
3.2.2. Les Métaheuristiques : .....	9
A. Les méthodes de voisinage : .....	9
A.1. Le recuit simulé : .....	9
A.2. La recherche Tabou : .....	10
B. Algorithmes évolutionnaires : .....	10
B.1 Algorithmes génétiques (AGs) .....	11
B.2 Programmation génétique (PG) : .....	11
B.3 Stratégies d'évolution (SE) : .....	12
B.4 Programmation évolutive (PE) : .....	12
4. Les algorithmes génétiques : .....	12
4.1 Présentation: .....	12
4.2 Codage des individus : .....	14
4.3 L'opérateur de sélection : .....	14
4.4 L'opérateur de croisement:.....	16

4.5	L'opérateur de Mutation :	17
4.6	L'opérateur de remplacement :	17
4.7	Les paramètres d'un AG :	18
4.8	Avantages et inconvénients des algorithmes génétiques :	18
5.	Améliorations classiques :	19
5.1.	Scaling :	20
5.1.1.	Scaling linéaire :	20
5.1.2.	Scaling exponentiel :	20
5.2.	Sharing :	22
6.	Conclusion	25

## CHAPITRE 2 : L'état de l'art

I.	Introduction.....	27
2.	Scaling des populations d'un algorithme génétique pour problèmes Job Shop_Scheduling utilisant MapReduce.....	27
3.	Fitness sharing et méthodes Niching revisité : .....	28
4.	Algorithmes génétiques pour le réseau bayésien: Le rôle de scaling et niching.....	29
5.	Amélioration de l'efficacité d'un algorithme génétique appliquée à l'opération de Tactic Multi-Robot : .....	30
6.	Améliorer la diversité des algorithmes génétiques pour la sélection d'entité _améliorée:.....	30
7.	Algorithmes Génétiques: Qu'est-ce que fitness scaling est optimal?. .....	31
8.	Comparaison des fonctions de fitness scaling dans les algorithmes génétiques avec _des applications à traitement optique : .....	32
9.	Transformer Classement: une nouvelle méthode de fitness scaling dans les _algorithmes génétiques :.....	33
II.	Conclusion: .....	
	<b>36Erreur ! Signet non défini.</b>	

## CHAPITRE 3 : Conception de l'approche

1.	Introduction .....	38
2.	Démarche .....	38
3.	Les étapes de conception .....	38
3.1.	L'algorithme génétique .....	38
3.2.	Scaling .....	39

3.3. Sharing .....	39
4. Problème du voyageur de commerce : .....	40
5. Conclusion : .....	49

## **CHAPITRE 4 : Résultats et discussions**

1. Introduction : .....	51
2. les outils de developpement : .....	51
2.1 Fortran : .....	51
2.2 Gnuplot : .....	51
3. Expérimentation : .....	52
4. Conclusion : .....	58
<b>CONCLUSION GENERALE.....</b>	<b>60</b>

# INTRODUCTION GENERALE

L'optimisation est l'une des branches les plus importantes des mathématiques appliquées, et de nombreuses recherches, à la fois pratiques et théoriques, lui sont consacrées. Il existe deux grandes approches de l'optimisation. L'une est dite déterministe : les algorithmes de recherche utilisent toujours le même cheminement pour arriver à la solution, et on peut donc déterminer à l'avance les étapes de la recherche. L'autre est aléatoire : pour des conditions initiales données, l'algorithme ne suivra pas le même cheminement pour aller vers la solution, et peut même proposer différentes solutions. C'est vers cette seconde approche, que va s'orienter notre travail, et plus particulièrement vers un type bien précis d'algorithmes de recherche aléatoire, les algorithmes dits évolutionnaires.

Les algorithmes évolutionnaires représentent un outil important pour la résolution des problèmes d'optimisation. D'ailleurs, ils sont de plus en plus utilisés dans de multiples domaines. Ils sont faciles à mettre en œuvre et fournissent d'excellentes performances à de faibles coûts.

Les algorithmes génétiques font partie de cette famille, ils permettent d'explorer des domaines possédant de très nombreuses solutions, et leur efficacité pratique a été prouvée bien avant que les résultats de convergence théorique soient établis. Toutefois le choix des nombreux opérateurs génétiques, intervenant dans la mise en place de l'algorithme reste à l'appréciation de l'utilisateur, c'est pour cela qu'un domaine de recherche très actif est consacré à l'étude de ces derniers et à la mise en place de nouvelles techniques. Surtout que l'utilisation de ces algorithmes est souvent coûteuse en temps de calculs et les performances de ces algorithmes dépendent beaucoup des différents opérateurs génétiques.

Dans le cadre de notre thème de fin d'études, nous nous sommes intéressés à l'étude de l'impact du scaling et du sharing sur l'efficacité des algorithmes génétiques. En dépit de leur évolution, les algorithmes génétiques demeurent révéler deux handicaps majeurs : la convergence prématurée et le temps de calcul.

L'objectif de ce mémoire est d'implémenter un algorithme génétique en y intégrant les deux opérateurs : le scaling et le sharing afin de remédier à ces deux handicaps, nous nous sommes intéressés au PVC (Problème du Voyageur de Commerce) pour en mesurer l'efficacité.

Afin de bien présenter notre travail nous nous sommes optés pour la structure suivante :

- La première partie de notre travail est consacrée à l'exposition des techniques d'optimisation capables de résoudre un certain type de problèmes. Deux grandes classes de méthodes sont présentées : les méthodes exactes qui consistent généralement à énumérer, de manière implicite, l'ensemble des solutions de l'espace de recherche et garantissent de trouver une solution optimale, et les méthodes approchées qui traitent généralement des problèmes de grande taille, elles n'assurent pas de trouver la solution optimale mais sont efficaces. Les méthodes les plus connues et les plus utilisées vont être détaillées dans notre travail. Puis nous allons exposer les améliorations classiques (scaling et sharing) pour remédier aux deux handicaps majeurs : la convergence prématurée et le temps de calcul.
- Le deuxième chapitre présente un état de l'art concernant le scaling et le sharing sur l'efficacité des algorithmes génétiques.
- Le troisième chapitre décrit la démarche suivie pour la conception de notre approche en l'appliquant au PVC pour en mesurer l'efficacité.
- Enfin le quatrième chapitre présente les tests où nous présentons les différents résultats obtenus sur la modélisation du PVC à partir d'algorithmes génétiques et en fin pour terminer, une conclusion générale de manuscrite, nous évaluons notre travail et nous présentons des perspectives pour les prochaines études.



## **1. Introduction :**

L'optimisation combinatoire est un outil indispensable combinant diverses techniques des mathématiques discrètes et de l'informatique afin de résoudre des problèmes d'optimisation combinatoire de la vie réelle [1].

Un problème d'optimisation combinatoire consiste à trouver la meilleure solution dans un ensemble discret de solutions appelé ensemble des solutions réalisables. En général, cet ensemble est fini mais de cardinalité très grande [1].

Il s'agit, en général, de maximiser ou de minimiser une fonction objectif sous certaines contraintes afin de trouver une solution optimale en un temps d'exécution raisonnable.

Dans ce chapitre, nous allons présenter la méthode des algorithmes génétiques en tant que l'une des méthodes les plus représentatives, développées pour résoudre les problèmes d'optimisation combinatoire.

## **2. Problème d'optimisation combinatoire :**

### **2.1. Définition :**

L'optimisation combinatoire consiste à minimiser ou maximiser une fonction souvent appelée fonction coût, d'une ou plusieurs variables soumises à des contraintes. Elle occupe une place très importante en recherche opérationnelle, en mathématiques discrètes et en informatique. Son importance se justifie d'une part, par la grande difficulté des problèmes d'optimisation, et d'autre part par de nombreuses applications pratiques pouvant être formulées sous la forme d'un problème d'optimisation combinatoire [24]. Bien que les problèmes d'optimisation combinatoire soient souvent faciles à définir, ils sont généralement difficiles à résoudre. En effet, la plupart de ces problèmes appartiennent à la classe des problèmes NP-difficiles et ne possèdent donc pas à ce jour de solution algorithmique efficace valable pour toutes les données [1].

### **2.2. Résolution d'un problème d'optimisation combinatoire :**

Résoudre un problème d'optimisation combinatoire nécessite l'étude de trois points particuliers :

- La définition de l'ensemble des solutions réalisables,
- L'expression de l'objectif à optimiser,

- Le choix de la méthode d'optimisation à utiliser,

Les deux premiers points relèvent de la modélisation du problème, le troisième de sa résolution. Afin de définir l'ensemble des solutions réalisables, il est nécessaire d'exprimer l'ensemble des contraintes du problème. Ceci ne peut être fait qu'avec une bonne connaissance du problème sous étude et de son domaine d'application [32].

### **3. Les méthodes d'optimisation combinatoire :**

Les méthodes d'optimisation peuvent être réparties en deux grandes classes de méthodes pour la résolution des problèmes :

- Les méthodes exactes,
- Les méthodes approchées,

#### **3.1. Les méthodes exactes :**

Le principe des méthodes exactes consiste à rechercher, souvent de manière implicite, une solution, la meilleure solution ou l'ensemble des solutions d'un problème. L'optimisation exacte concerne toutes les méthodes permettant d'obtenir un résultat dont on sait qu'il est optimal à un problème précis. Cela va des méthodes du simplex aux méthodes de Lagrangien en passant par la programmation dynamique. On peut classer les méthodes exactes en quatre grandes classes [1]:

- La programmation dynamique,
- La programmation linéaire continue ou en nombres entiers,
- La programmation non linéaire avec ou sans contraintes,
- Les méthodes de recherche arborescente (Branch & Bound).

##### **3.1.1. La programmation dynamique :**

La programmation dynamique est une méthode utilisée pour résoudre des problèmes où une séquence de décisions optimale doit être trouvée. L'idée de base est que l'on peut déduire une ou la solution optimale d'un problème en combinant des solutions optimales d'une série de sous-problèmes consistant à choisir des séquences plus courtes de décisions. Les solutions des problèmes sont calculés de manière ascendante, c'est-à-dire qu'on débute par les solutions des sous-problèmes les plus petits pour ensuite déduire progressivement les solutions de l'ensemble [33].

### **3.1.2. La programmation linéaire :**

La programmation linéaire (PL) est une branche de l'optimisation permettant de résoudre de nombreux problèmes économiques et industriels. La programmation linéaire désigne la manière de résoudre les problèmes dont la fonction objective et les contraintes sont toutes linéaires [8]. Si l'ensemble de solutions possibles  $S$  est formulé comme un ensemble de variables à valeurs dans l'ensemble des réels  $R$  et si on a des contraintes à satisfaire des inégalités linéaires et si  $f$  est une fonction linéaire en ces variables, on parle alors d'un problème de programmation linéaire (PL). Plusieurs problèmes réels de recherche opérationnelle peuvent être exprimés comme un problème de PL. Pour cette raison, un grand nombre d'algorithmes pour la résolution d'autres problèmes d'optimisation sont fondés sur la résolution de problèmes linéaires [1].

### **3.1.3. La méthode de branch and bound :**

La méthode de branch and bound (procédure par évaluation et séparation progressive) consiste à énumérer ces solutions d'une manière intelligente en ce sens que, en utilisant certaines propriétés du problème en question, cette technique arrive à éliminer des solutions partielles qui ne mènent pas à la solution que l'on recherche. De ce fait, on arrive souvent à obtenir la solution recherchée en des temps raisonnables. Bien entendu, dans le pire de cas, on retombe toujours sur l'élimination explicite de toutes les solutions du problème [41].

Pour ce faire, cette méthode se dote d'une fonction qui permet de mettre une borne sur certaines solutions pour, soit les exclure soit les maintenir comme des solutions potentielles. Bien entendu, La performance d'une méthode de branch and bound dépend, entre autres, de la qualité de cette fonction [1].

## **3.2. Les méthodes approchées:**

Les méthodes approchées fournissent une solution approchée au problème traité. Elles sont en général conçues de manière à ce que la solution obtenue puisse être proche de la valeur optimale : de telles méthodes permettent d'obtenir des bornes inférieures ou supérieures de la valeur optimale telles que [21]:

- Méthodes Heuristiques
- Méthodes Méta heuristiques

### 3.2.1. Les méthodes Heuristiques :

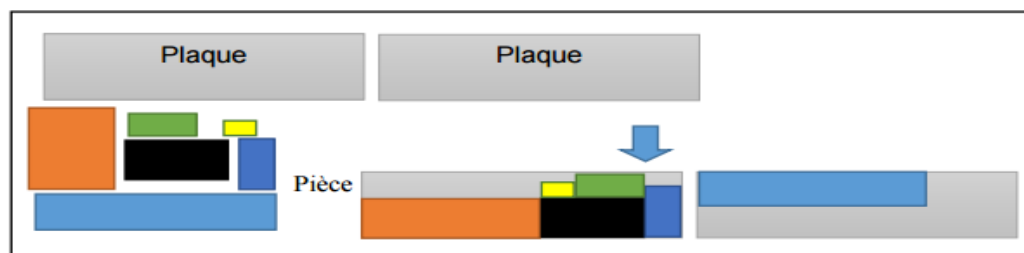
Du grec *heuriskein* : trouver/découvrir (*heureka*). Une heuristique est plutôt une méthode qui cherche (une stratégie) sans garantir le résultat. Destiné à un problème spécifique, le temps de calcul est raisonnable sans garantir la faisabilité ou l'optimalité [9].

En recherche opérationnelle, les heuristiques sont des règles empiriques simples qui ne sont pas basées sur l'analyse scientifique (différents algorithmes). Elles sont basées sur l'expérience, les résultats déjà obtenus et sur l'analogie pour optimiser les recherches suivantes. Généralement, on n'obtient pas la solution optimale mais une solution approchée [12]. Parmi les heuristiques, on peut citer l'algorithme glouton).

**L'Algorithme glouton :** Les algorithmes gloutons sont ceux qui trouvent une solution parmi plusieurs possibles, comparables selon un critère global. A chaque fin d'étape de l'algorithme, on choisit l'étape suivante en fonction des informations locales. La méthode gloutonne ne trouve pas forcément la meilleure solution [21]. L'idée générale est :

- Générer une solution initiale
- Modifier légèrement la structure de cette solution pour en obtenir une meilleure
- Répéter, tant qu'on trouve des améliorations

**Exemple :** On dispose de plaques rectangulaires toutes identiques dans lesquelles on veut placer des pièces rectangulaires sans chevauchement. Les pièces à placer ont des dimensions diverses [39].



**Figure 1.1 :** Placement optimal des pièces [39].

On veut trouver le placement pour minimiser le nombre de plaques utilisées.

Algorithme glouton : trier les pièces en fonction de leur taille et placer d'abord les pièces les plus grandes [39].

### 3.2.2. Les Métaheuristiques :

Le mot métaheuristique est dérivé de la composition de deux mots grecs [23]:

- heuristique qui vient du verbe heuriskein et qui signifie ‘trouver’.
- méta qui est un suffixe signifiant ‘au -delà’, ‘dans un niveau supérieur’.

Les métaheuristiques sont des méthodes inspirées de la nature, ce sont des heuristiques modernes dédiées à la résolution des problèmes et plus particulièrement aux problèmes d'optimisation, qui tentent de trouver l'optimum global (plusieurs solutions) ou l'optimum local (une seule solution) d'un problème d'optimisation difficile.

Les métaheuristiques se subdivisent en deux sous-classes [38] :

- Les méthodes de voisinage.
- Les algorithmes évolutionnaires

#### A. Les méthodes de voisinage :

Ces méthodes partent d'une solution initiale (obtenue de façon exacte, ou par tirage aléatoire) et s'en éloignent progressivement, pour réaliser une trajectoire, un parcours progressif dans l'espace des solutions. Dans cette catégorie, se rangent [38] :

- le recuit simulé ;
- la méthode Tabou le terme de **recherche locale** est de plus en plus utilisée pour qualifier ces méthodes.

##### A.1 . Le recuit simulé :

L'idée est d'effectuer un mouvement selon une distribution de probabilité qui dépend de la qualité des différents voisins [12]:

- Les meilleurs voisins ont une probabilité plus élevée.
- Les moins bons ont une probabilité plus faible.

On utilise un paramètre, appelé la température (notée T) :

- T élevée : tous les voisins ont à peu près la même probabilité d'être acceptés.
- T faible : un mouvement qui dégrade la fonction de coût a une faible probabilité d'être choisi.

- $T=0$  : aucune dégradation de la fonction de coût n'est acceptée. La température varie au cours de la recherche :  $T$  est élevée au début, puis diminue et finit par tendre vers 0.

### **A.2. La recherche Tabou :**

La recherche Tabou a été proposée par F.Glover en 1986. L'algorithme se nomme comme cela parce qu'il y a interdiction de reprendre des solutions récemment visitées.

Depuis cette date, la méthode est devenue très populaire, grâce aux succès qu'elle a remportés pour résoudre de nombreux problèmes. Cet algorithme introduit une notion de mémoire dans la stratégie d'exploration de l'espace de recherche [20].

### **B. Algorithmes évolutionnaires :**

Les algorithmes évolutifs (Evolutionary Algorithms) sont des techniques de recherche inspirées de l'évolution biologique des espèces, apparues à la fin des années 50. Parmi plusieurs approches [14], les algorithmes génétiques (AG) constituent certainement les algorithmes les plus connus [16].

Le principe d'un algorithme évolutionnaire est très simple. Un ensemble de  $N$  points dans un espace de recherche, choisi à priori au hasard, constituent la population initiale; chaque individu  $x$  de la population possède une certaine performance, qui mesure son degré d'adaptation à l'objectif visé : dans le cas de la minimisation d'une fonction objectif  $f$ ,  $x$  est d'autant plus performant que  $f(x)$  est plus petit. Un AE consiste à faire évoluer progressivement, par générations successives, la composition de la population, en maintenant sa taille constante. Au cours des générations, l'objectif est d'améliorer globalement la performance des individus; le but étant d'obtenir un tel résultat en imitant les deux principaux mécanismes qui régissent l'évolution des êtres vivants, selon la théorie de Darwin [5]:

- la sélection, qui favorise la reproduction et la survie des individus les plus performants,
- la reproduction, qui permet le brassage,
- la recombinaison et les variations des caractères héréditaires des parents, pour former des descendants aux potentialités nouvelles.

En fonction des types d'opérateurs, i.e., sélection et reproduction génétique, employés dans un algorithme évolutif, quatre approches différentes ont été proposées [5] :

- les algorithmes génétiques (AG)
- la programmation génétique (PG)
- les stratégies d'évolution (SE)
- la programmation évolutive (PE).

L'algorithme évolutionnaire peut être décrit généralement, comme suit :

- construction et évaluation d'une population initiale jusqu'à atteindre un critère d'arrêt :
- sélection d'une partie de la population,
- reproduction des individus sélectionnés,
- mutation de la descendance,
- évaluation du degré d'adaptation de chaque individu,
- remplacement de la population initiale par une nouvelle population.

### **B.1 Algorithmes génétiques (AGs) :**

Les AGs ont été conçus comme outil d'optimisation stochastique. Ils sont inspirés des mécanismes de la génétique et de l'évolution naturelle des êtres vivants.

Ils cherchent la solution globale d'une manière aléatoire. Ces algorithmes sont les plus connus et utilisés des algorithmes évolutionnaires [26].

### **B.2 Programmation génétique (PG) :**

La programmation génétique est une variante des algorithmes génétiques, destinée à manipuler des programmes pour implémenter un modèle d'apprentissage automatique. Les programmes sont généralement codés par des arbres qui peuvent être vus comme des chaînes de bits de longueur variable. Une grande partie des techniques et des résultats concernant les algorithmes génétiques peuvent donc également s'appliquer à la programmation génétique [40].

### **B.3 Stratégies d'évolution (SE) :**

Sont des algorithmes itératifs dans lesquels un parent génère un enfant (1+1). Le meilleur des deux survit et devient le parent de la génération suivante. La généralisation de ce processus a donné les algorithmes  $(u + \lambda)$  dans lesquels  $u$  parents génèrent  $\lambda$  enfants. Les  $u$  meilleurs survivent [25].

### **B.4 Programmation évolutive (PE) :**

La programmation évolutive a été introduite par Laurence Fogel en 1966 [40] dans la perspective de créer des machines à état fini (Finite State Machine) dans le but de prédire des événements futurs sur la base d'observations antérieures.

La programmation évolutive suit le schéma classique des algorithmes évolutifs de la façon suivante :

- ❖ on génère aléatoirement une population de  $n$  individus qui sont ensuite évalués;
- ❖ chaque individu produit un fils par l'application d'un opérateur de mutation suivant une distribution normale;
- ❖ les nouveaux individus sont évalués et on sélectionne de manière stochastique une nouvelle population de taille  $n$  (les mieux adaptés) parmi les  $2m$  individus de la population courante (parents + enfants);
- ❖ on réitère, à partir de la deuxième étape, jusqu'à ce que le critère d'arrêt choisi soit valide.

La programmation évolutive partage de nombreuses similitudes avec les stratégies d'évolution : les individus sont, a priori, des variables multidimensionnelles réelles et il n'y a pas d'opérateur de recombinaison [40].

## **4. Les algorithmes génétiques :**

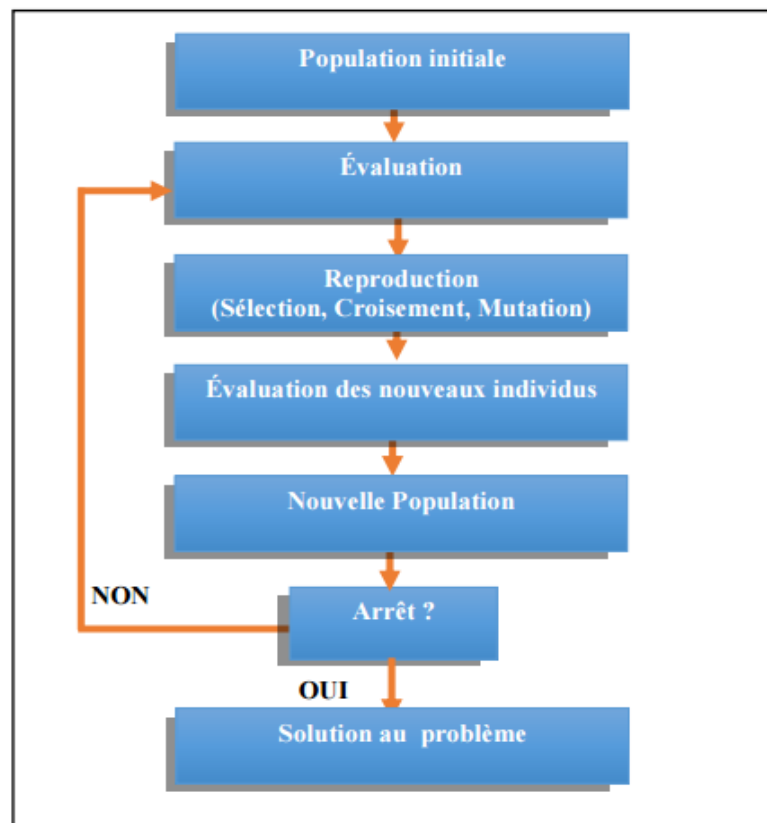
### **4.1 Présentation :**

Les algorithmes génétiques sont inspirés de la théorie de l'évolution et des processus biologiques qui permettent à des organismes de s'adapter à leur environnement. Ils ont été inventés dans le milieu des années 60 (Holland, 1962; Rechenberg, 1965; Fogel et al, 1966) [3].



La sélection naturelle que Darwin appelle l'élément "propulseur" de l'évolution, favorise les individus d'une population qui sont le mieux adaptés à un environnement. La sélection est suivie de croisements et de mutations au niveau des individus, constitués d'un ensemble de gènes. Ainsi, deux individus "parents" qui se croisent, transmettent une partie de leur patrimoine génétique à leurs descendants. L'individu enfant fait que celui-ci est plus ou moins adapté à l'environnement. S'il est bien adapté, il a une plus grande chance de procréer dans la génération future. Au fur et à mesure des générations, sont sélectionnés les individus les mieux adaptés, et l'augmentation du nombre des individus bien adaptés fait évoluer la population entière [31].

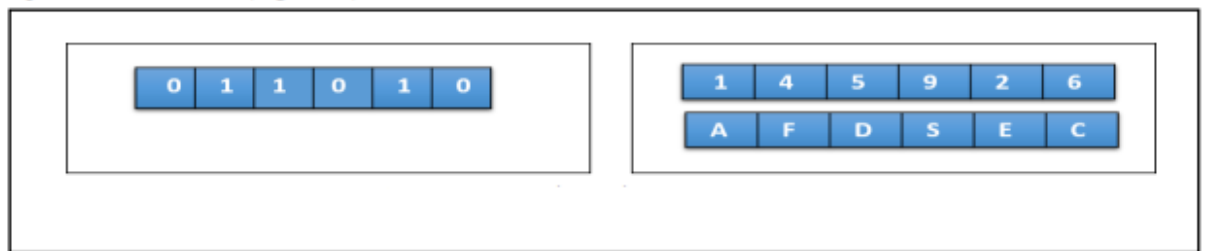
La mise en œuvre des algorithmes génétiques nécessite plusieurs étapes à détailler. La première est le codage d'un individu représenté par un chromosome. La seconde, est le calcul de la qualité. La troisième est de définir les opérateurs de reproduction [38].



**Figure 1.2:** Fonctionnement des algorithmes génétiques [38].

## 4.2 Codage des individus :

Cette étape associe à chacun des points de l'espace d'état une structure de données. Elle se place généralement après une phase de modélisation mathématique du problème traité. La qualité du codage des données conditionne le succès des algorithmes génétiques. Les codages binaires ont été très utilisés à l'origine. Les codages réels sont désormais largement utilisés, notamment dans les domaines applicatifs pour l'optimisation de problèmes à variables réelles [38]. Des exemples du codage sont présentés dans figure suivante :



**Figure 1.3:** Codage des solutions [38].

Les algorithmes génétiques utilisent trois opérateurs pour générer de nouvelles solutions :

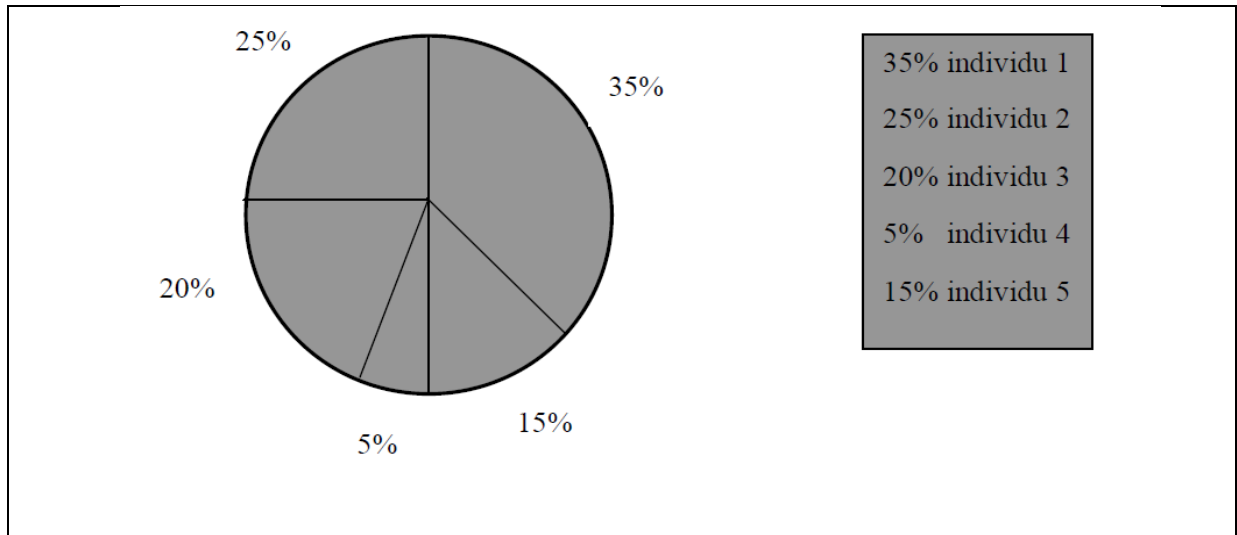
- L'opérateur de sélection qui permet de choisir des solutions parentes sur lesquelles la reproduction va être faite pour générer de nouvelles solutions.
- L'opérateur de croisement qui permet de croiser les deux solutions parentes et créer de nouvelles solutions.
- L'opérateur de mutation qui permet de diversifier les nouvelles solutions afin qu'elles ne ressemblent pas trop aux solutions parentes.

## 4.3 L'opérateur de sélection :

La sélection consiste à choisir des individus qui permettront de générer de nouveaux individus. Plusieurs méthodes existent pour sélectionner des individus destinés à la reproduction. On citera les deux méthodes classiques les plus utilisées.

**La sélection par roulette :** la population est représentée comme une roue de roulette, ou chaque individu est représentée par une portion qui correspond proportionnellement à sa valeur de fitness. La sélection d'un individu se fait en tournant la roue en face d'un pointeur fixe. Cette procédure est répétée jusqu'à constitution

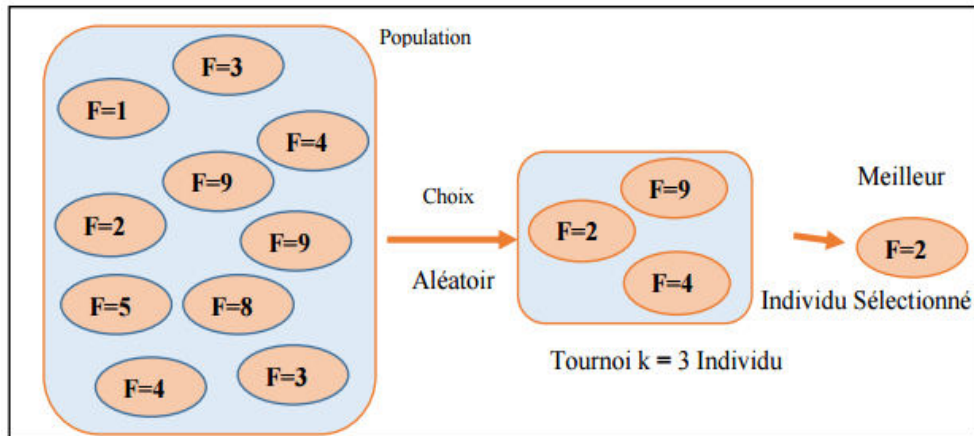
complète de la population des parents. L'un des inconvénients de ce type de sélection est de choisir presque toujours le même individu s'il en existe un bien meilleur que les autres, ce qui cause une perte de diversité dans la population [18]. La figure 1.4 illustre une population de 5 individus dont les performances sont représentées en roulette.



**Figure 1.4:** Sélection par roulette [18].

**La sélection par tournoi :** La sélection par tournoi est l'une des sélections les plus utilisées dans les algorithmes évolutionnaires. Le principe consiste à choisir aléatoirement un sous-ensemble d'individus ( $K$  individus) dans la population, puis à sélectionner le meilleur individu dans ce groupe en fonction de sa fitness. Ce processus est répété jusqu'à l'obtention du nombre d'individus requis.

Le nombre de participants à un tournoi ( $K$ ), appelé la taille du tournoi, est utilisé pour faire varier la pression de cette sélection. Si ce nombre est grand, alors la pression sera forte et les faibles individus auront une petite chance d'être choisis. En général, un seul gagnant est choisi parmi les participants à un tournoi. Ce gagnant peut être choisi d'une façon déterministe ou probabiliste. Dans le cas déterministe, qui est pratiquement le plus utilisé, le gagnant est l'individu de meilleure qualité (meilleure fitness). Dans le cas probabiliste, chacun des participants peut être choisi en tant que gagnant avec une probabilité proportionnelle à sa fitness [18]. Cette méthode est en général satisfaisante.

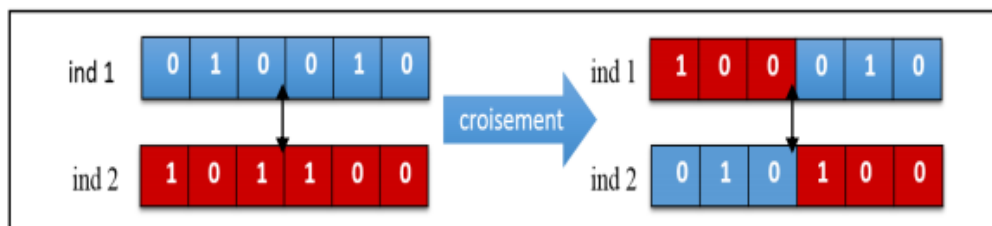


**Figure 1.5:** Sélection par tournoi [18].

#### 4.4 L'opérateur de croisement:

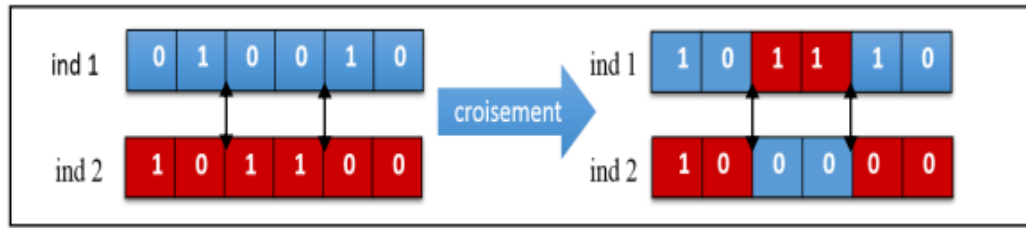
Les croisements permettent de simuler des reproductions d'individus dans le but d'en créer de nouveaux. Il est tout à fait possible de faire des croisements aléatoires. Toutefois, une solution largement utilisée est d'effectuer des croisements multi-points [18].

**Le croisement à un point :** Il a été initialement défini pour le codage binaire. Le principe consiste à tirer aléatoirement une position pour chaque parent et à échanger les sous-chaines des parents à partir des positions tirées ; ce qui donne naissance à deux nouveaux individus ind1 et ind2 [18].



**Figure 1.6:** Croisement à un point [18].

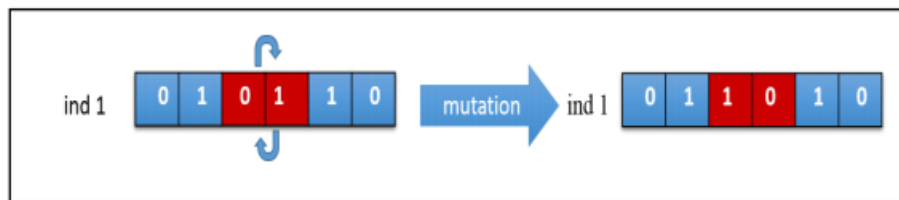
**Le croisement multi-points :** Cette méthode reprend le mécanisme de la méthode de croisement à un point en généralisant l'échange à 3 ou 4 sous-chaines.



**Figure 1.7 :** Croisement à deux points [18].

#### 4.5 L'opérateur de Mutation :

L'opération de mutation protège les algorithmes génétiques des pertes prématurées d'informations pertinentes. Elle permet d'introduire une certaine information dans la population, qui aurait pu être perdue lors de l'opération de croisement. Ainsi, elle participe au maintien de la diversité, utile à une bonne exploration du domaine de recherche [18].



**Figure 1.8:** Une mutation [18].

#### 4.6 L'opérateur de remplacement :

Cet opérateur permet de réintroduire les descendants obtenus par applications successives des opérateurs de sélection, de croisement et de mutation (la population P') dans la population de leurs parents (la population P). On trouve des méthodes de remplacement différentes ; par exemple, la méthode de remplacement stationnaire [34].

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

- La première se contente de remplacer la totalité de la population P par la population P', cette méthode est connue sous le nom de remplacement générationnel.
- La deuxième méthode consiste à remplacer les mauvais individus trouvés dans la population P par les meilleurs de la population P' [38].

#### 4.7 Les paramètres d'un AG :

Les paramètres qui conditionnent la convergence d'un algorithme génétique sont :

- la taille de la population d'individus ;
- le nombre maximal de générations ;
- la probabilité de croisement ;
- la probabilité de mutation.

Les valeurs de tels paramètres dépendent fortement de la problématique étudiée. Ainsi, il n'existe pas de paramètres qui soient adaptés à la résolution de tous les problèmes qui peuvent être posés à un algorithme génétique. Cependant, certaines valeurs sont souvent utilisées (définies dans la littérature) et peuvent être de bons points de départ pour démarrer une recherche de solutions à l'aide d'un AG [26].

- la probabilité de croisement est choisie dans l'intervalle [0.7, 0.99] ;
- la probabilité de mutation est choisie dans l'intervalle [0.001,0.01].

Trouver de bonnes valeurs à ces paramètres est donc un problème parfois délicat.

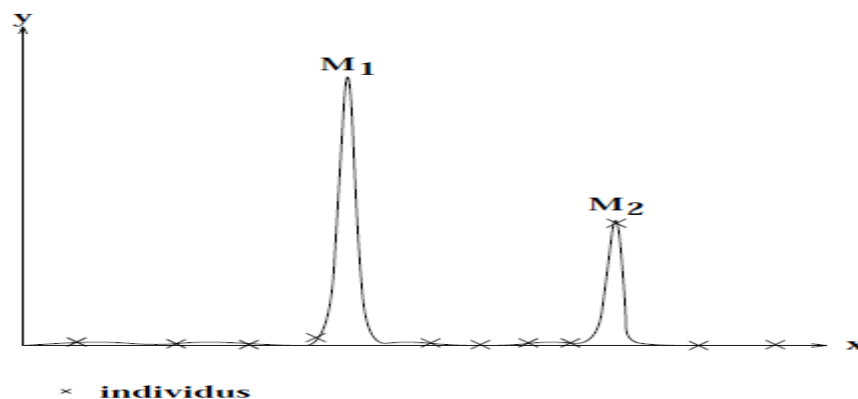
#### 4.8 Avantages et inconvénients des algorithmes génétiques :

- **Les avantages des AGs [36]:**
  - Les AGs opèrent au niveau du codage des paramètres sans se soucier de leur nature, donc ils s'appliquent à de nombreuses classes de problèmes, qui dépendent éventuellement de plusieurs paramètres de natures différentes (booléens, entiers, réels, fonctions...);
  - Pour les mêmes raisons un AG est dans l'idéal totalement indépendant de la nature du problème et de la fonctionnelle à optimiser, car il ne se sert que des valeurs d'adaptation, qui peuvent être très différentes des valeurs de la fonction à optimiser, même si elles sont calculées à partir de cette dernière;

- Potentiellement les AGs explorent tous l'espace des points en même temps, ce qui limite les risques de tomber dans des optimums locaux;
  - Les AGs présentent une grande robustesse c'est-à-dire une grande capacité à trouver les optimums globaux des problèmes d'optimisation.
- **Les inconvénients des AGs [36]:**
    - Les AGs ne sont encore actuellement pas très efficaces en coût (ou vitesse de convergence), vis-à-vis de méthodes d'optimisation plus classiques;
    - Parfois les AGs convergent très vite vers un individu particulier de la population dont la valeur d'adaptation est très élevée;
    - Le respect de la contrainte de domaine par la solution codée sous forme de chaîne de bits pose parfois problème. Il faut bien choisir le codage, voir modifier les opérateurs;
    - L'utilisation d'un AG ne garantit pas le succès de l'optimisation;

## 5. Améliorations classiques :

Les processus de sélection présentes sont très sensibles aux écarts de fitness et dans certains cas, un très bon individu risque d'être reproduit trop souvent et peut même provoquer l'élimination complète de ses congénères; on obtient alors une population homogène contenant un seul type d'individus. Ainsi, dans l'exemple de (la figure 1.9) le second mode M2 risque d'être le seul représentant pour la génération suivante et seule la mutation pourra aider à atteindre l'objectif global M1 au prix de nombreux essais successifs [19].



**Figure 1.9:** Exemple où les sélections classiques risquent de ne reproduire qu'un individu [19].

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

Enfin, on peut également introduire des recherches multi-objectifs, en utilisant la notion de dominance lors de la sélection [19].

## 5.1. Scaling :

Le scaling ou mise à l'échelle, modifie les fitness afin de réduire ou d'amplifier artificiellement les écarts entre les individus. Le processus de sélection n'opère plus sur la fitness réelle mais sur son image après scaling. Parmi les fonctions de scaling, on peut envisager le scaling linéaire et le scaling exponentiel. Soit  $f_r$  la fitness avant scaling et  $f_s$  la fitness modifiée par le scaling [27].

### 5.1.1. Scaling linéaire :

Dans ce cas, la fonction de scaling est définie de la façon suivante :  $f_s = a f_r + b$

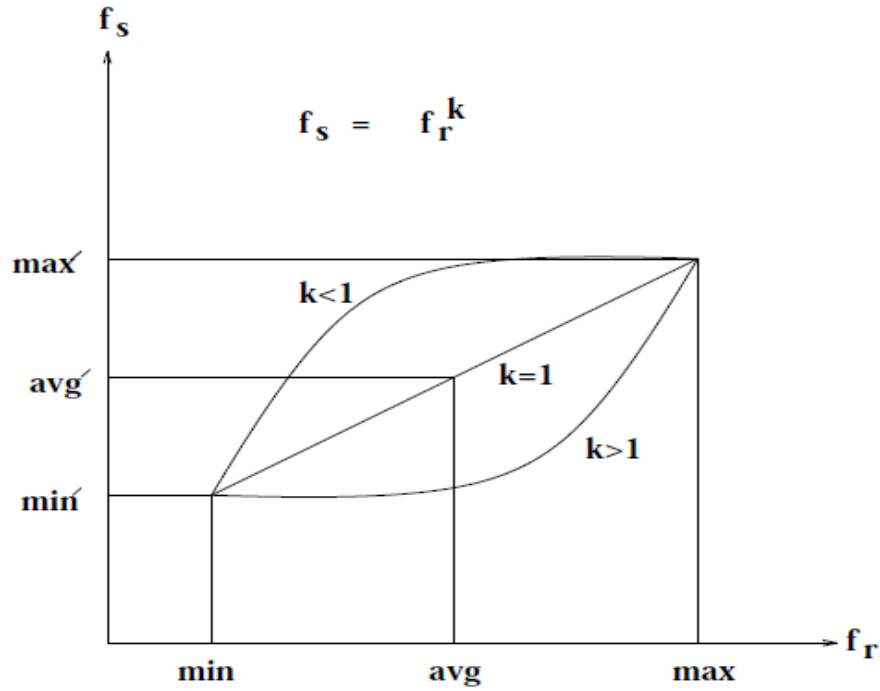
$$a = \frac{\max' - \min'}{\max - \min} ; \quad b = \frac{\min' \cdot \max - \min \cdot \max'}{\max - \min} \quad (1)$$

En règle générale,  $a < 1$ , ce qui permet de réduire les écarts de fitness et donc de favoriser l'exploration de l'espace. Ce scaling est statique par rapport au numéro de génération et pénalise la fin de convergence lorsque l'on désire favoriser les modes dominants [19].

### 5.1.2 Scaling exponentiel :

Il est défini de la façon suivante (voir figure 1.10) [19] :  $f_s = (f_r)^{k(n)}$





**Figure 1.10:** Fonction de scaling exponentielle [19].

Où  $n$  est la génération courante.

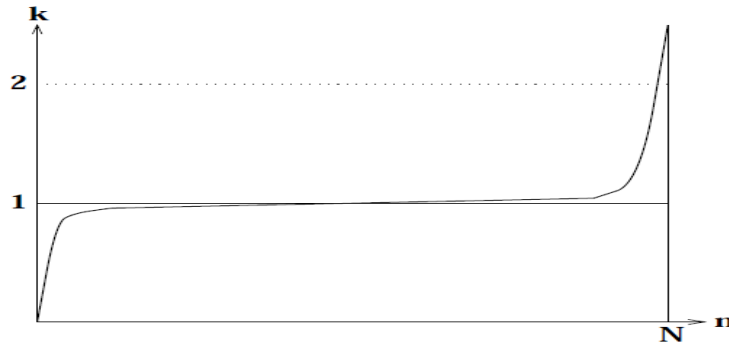
- Pour  $k$  proche de zéro, on réduit fortement les écarts de fitness ; aucun individu n'est vraiment favorisé et l'algorithme génétique se comporte comme un algorithme de recherche aléatoire et permet d'explorer l'espace.
- Pour  $k$  proche de 1 : le scaling est inopérant.
- Pour  $k > 1$  les écarts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

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

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

- $n$  étant la génération courante,
- $N$  le nombre total de générations prévues,
- $p$  un paramètre à choisir.

Le choix de  $p=0.1$  s'est avéré pertinent dans les applications (Nous ne détaillerons cependant pas ici les mesures de sensibilité faites sur quelques exemples pour évaluer ce choix) [27]. L'évolution de  $k$  en fonction de la génération  $n$  est donnée par la figure 1.11.

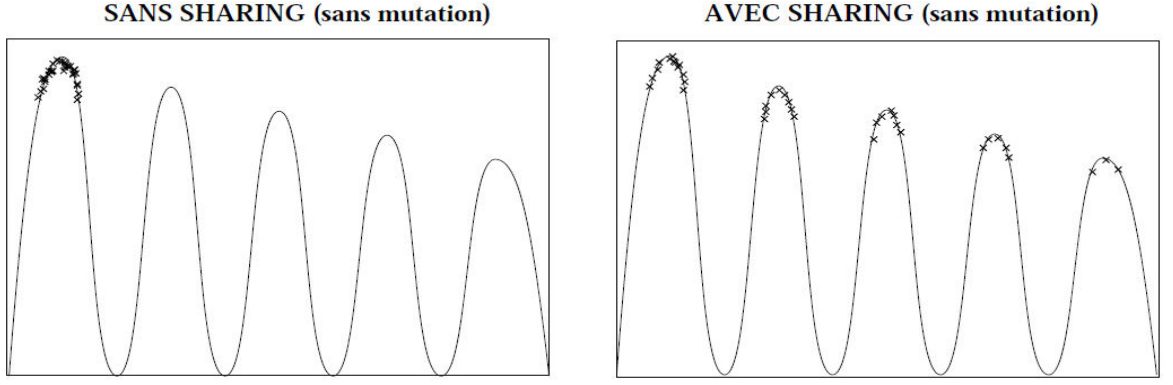


**Figure 1.11:** Allure de l'évolution de  $k$  en fonction des générations [27].

Ce dernier principe de scaling donne effectivement de meilleurs résultats sur nos problèmes que le scaling linéaire et sera donc systématiquement utilisé. Dans le cas des fonctions objectives multi-modes présentant des optima quasi-équivalents, cette technique de scaling, en amplifiant les écarts de fitness en fin de convergence, va effectivement favoriser le mode dominant mais aussi masquer les modes sous-optimaux qui peuvent tout de même présenter un intérêt. Le scaling permet donc une bonne exploration de l'espace d'états mais ne favorise pas la répartition des individus sur les différents modes de la fonction objective [19].

## 5.2. Sharing :

L'objectif du sharing est de répartir sur chaque sommet de la fonction à optimiser un nombre d'individus proportionnel à la fitness associée à ce sommet. La figure 1.12 présente deux exemples de répartitions de populations dans le cas d'une fonction à cinq sommets : le premier sans sharing, le second avec sharing [19].



**Figure 1.12:** Objectif du sharing [19].

De la même façon que le scaling, le sharing modifie l'adaptation des individus. Ce dernier pénalise un individu en fonction du taux d'agrégation de la population dans son voisinage.

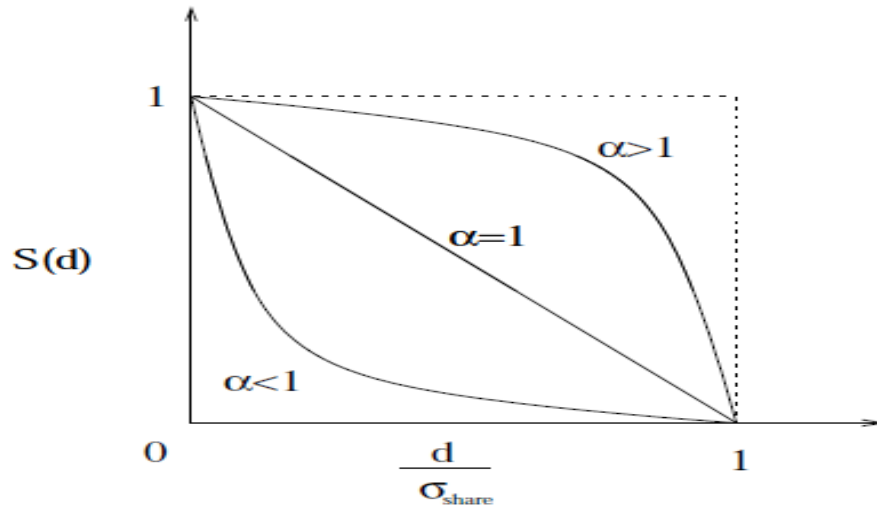
Pour cela, une distance  $d$  représentative des différences entre individus est nécessaire. La nouvelle fonction d'adaptation  $f_{sh}$  d'un individu  $i$  est donnée par [27]:

$$f_{sh}(i) = \frac{f(i)}{\sum_{j=1}^{N_{pop}} S(d(x_i, x_j))} \quad (3)$$

$$\text{Avec : } \begin{cases} S(d) = 1 - \left(\frac{d}{\sigma_{share}}\right)^\alpha & \text{si } d < \sigma_{share} \\ S(d) = 0 & \text{si } d > \sigma_{share} \end{cases}$$

Le paramètre  $\sigma_{share}$  définit la zone d'influence des individus : seuls les individus dont la distance est inférieure à  $\sigma_{share}$  se pénalisent mutuellement. Sa valeur doit être déterminée en fonction du problème traité et de la distance définie sur l'espace de recherche [27]. Il est souvent utile, pour ce réglage, de normaliser les distances (entre 0 et 1 par exemple).

- Le paramètre  $\alpha$  fixe l'intensité du sharing : plus  $\alpha$  est grand, plus les groupes d'individus agglomérés sont pénalisés (figure 1.13).



**Figure 1.13 :** Allure de la fonction de sharing en fonction de son intensité [27].

Dans la pratique, ce type de sharing donne de bons résultats mais sa complexité en

$O(N_{pop}^2)$  par rapport au nombre d'individus peut s'avérer pénalisante comparée aux autres opérations de l'algorithme génétique qui s'effectuent généralement en  $O(N_{pop})$ . C'est pourquoi le sharing clustérisé lui est souvent préféré [35].

### Sharing clustérisé :

Le sharing clustérisé permet de réduire la complexité du sharing en répartissant les individus de la population par groupes de proximité appelés clusters.

Deux paramètres  $d_{min} < d_{max}$  définissent la manière avec laquelle sont construits ces clusters [35]:

- Initialement, chaque individu de la population est considéré comme le centre d'un cluster dont il est l'unique élément.
- Si deux centres sont à une distance inférieure à  $d_{min}$ , les deux clusters correspondants sont réunis dans un unique cluster dont le centre est le milieu des deux centres initiaux.
- Si la distance d'un individu au centre du cluster le plus proche est inférieur à  $d_{max}$ , l'individu est ajouté au cluster et le centre de ce dernier devient le barycentre entre l'individu et l'ancien centre du cluster.

Ce type de sharing s'avère aussi efficace que celui présenté précédemment. La méthode nécessite cependant le calcul de barycentres entre les points de l'espace de recherche, ce qui peut être difficile en fonction du codage utilisé pour les individus [35].

## **6. Conclusion :**

Dans ce chapitre, nous avons présenté la théorie des algorithmes génétiques. En premier lieu, nous avons introduit un rappel sur le problème d'optimisation combinatoire. Puis nous avons essayé d'exposer quelques méthodes d'optimisation combinatoire en allant des méthodes exactes aux méthodes méta-heuristiques. Ce chapitre nous a aussi permis d'avoir une vue générale sur les concepts des AG, leurs paramètres et leurs mécanismes de fonctionnement. Une de ces applications est l'optimisation. Enfin, nous avons défini les améliorations classiques (scaling et sharing).

## **I. Introduction :**

Les algorithmes génétiques ont été largement utilisés pour aborder plusieurs types de problèmes en y intégrant les deux nouvelles techniques, en l'occurrence, le scaling et le sharing. Leur performance dans le champ d'optimisation et leur facilité d'utilisation ont été mises en avant. Dans ce chapitre, nous allons présenter l'état de l'art de ces applications.

Rappelons que les algorithmes génétiques sont des algorithmes évolutionnaires de recherche combinatoire qui se basent sur la fonction appelée fitness. Leur inconvénient majeur est la convergence prématurée. Les notions de scaling et de sharing ont été instaurées pour pallier à ce handicap et sont présentées dans les travaux suivants :

### **1. Aperçu sur les méthodes de maintien de la diversité dans les algorithmes génétiques :**

Deepti et al. (2012) [10] montrèrent que l'algorithme génétique est une méthode d'optimisation basée sur le principe de Darwin de la survie du plus fort. C'est une abstraction de la génétique naturelle complexe et un processus de sélection naturelle. L'algorithme génétique se fonde sur le principe de la sélection naturelle pour la reproduction de diverses opérations évolutives comme le croisement et la mutation. Deux facteurs de contrôle qui doivent être équilibrés dans le processus de sélection sont la diversité génétique et la pression sélective.

La diversité de la population peut être contrôlée par le moyen de fitness sharing, le surpeuplement déterministe et tant d'autres.

Deepti et al. (2012) [10] nous fournirent une brève connaissance de la variété des méthodes de maintien de la diversité de la population.

### **2. Scaling des populations d'un algorithme génétique pour problèmes Job Shop Scheduling utilisant MapReduce:**

Di-Wei Huang et al. (2010) [11] présentèrent un AG avec les populations massives pour résoudre le problème Job Shop Scheduling (JSSC) est implémenté en utilisant MapReduce. Le AG est non-trivial en ce qu'il comprend le codage / décodage des chromosomes, la

construction d'horaires, effectuer des recherches locales, la manipulation des sélections du tournoi-base, et le traitement de liaison non aléatoire. Comme il a été suggéré par la recherche théorique que AG avec de grandes tailles de population sont avantageux dans la résolution de problèmes difficiles, leurs AG pour JSSC est donnée populations massives et exécuté sur un cluster de 414 machines.

Di-Wei Huang et al (2010) [11] ne mirent pas l'accent sur proposant des algorithmes innovants ou surpassant d'autres solutions à JSSC, mais montre les effets d'un AG en cours d'exécution de grandes populations en parallèle, comme une amélioration potentielle aux solutions existantes. Deux expériences sont effectuées. La première expérience montre comment la taille des populations affecte l'AG à l'approche d'une bonne solution; le second montre comment le temps de fonctionnement peut être réduit par scaling de la taille d'un cluster.

### **3. Fitness sharing et méthodes Niching revisité :**

Bruno Sareni et al. (1998) [7] examinèrent l'intérêt pour la fonction d'optimisation multimodale se développe rapidement depuis les problèmes d'optimisation du monde réel nécessitent souvent l'emplacement de plusieurs optima en l'espace de recherche.

Dans ce contexte, méthodes niching étendent simple AG par la promotion de la formation de sous-populations stables dans le voisinage des solutions optimales. Méthodes niching ont été développées pour réduire l'effet de la dérive génétique résultant de l'opérateur de sélection dans le AG standard. La méthode de sharing est probablement le plus connu et aussi utilisé parmi les techniques niching. Ils présentent (1998) [7] les grands principes de fitness sharing et examine l'évolution récente de cette technique. Et ils consacrent (1998) [7] à une importante variété d'autres méthodes de nichage ont été rapporté dans le document. Ils concentrent sur le surpeuplement des techniques et explorent une récente méthode de compensation niching prometteuse appelée clearing. Certains résultats empiriques sont présentés pour une grande et un nombre limité d'évaluations de la fonction de fitness. Enfin, ils comparent (1998) [7] la méthode du sharing avec d'autres techniques de nichage.

#### **4. Algorithmes génétiques pour le réseau bayésien: Le rôle de scaling et niching :**

Les algorithmes génétiques (AG) sont inspirés du phénomène naturel d'évolution des espèces et sont utilisés pour l'adaptation, recherche, l'optimisation et l'apprentissage dans des environnements complexes. Le réseau bayésien (BN) sont des modèles graphiques qui sont basés sur la théorie des probabilités et la théorie des graphes, et sont utilisés pour raisonner et apprendre dans l'incertitude. Ole J. Mengshoel et al. [28] considèrent dans leur article comment utiliser la capacité d'optimisation de la fonction d'un AG comme un algorithme d'inférence BN.

Ole J. Mengshoel et al. (1998) [28] examinent le rôle de nichage et mise à l'échelle dans un AG utilisé pour rechercher les explications les plus probables dans un BN. La contribution principale de ce document est qu'il identifie que BN non triviaux sont des fonctions de fitness multi-modales où les diversités préservant technique de nichage est de l'aide. Le rôle de scaling en utilisant AG pour BN inférence a été exploré dans les recherches antérieures.

Cependant, Ole J. Mengshoel et al. (1998) [28] ajoutent que la recherche en suggérant une nouvelle fonction de scaling et aussi en constatant que le détartrage et le nichage des résultats sensiblement meilleurs que soit technique seule.

Ole J. Mengshoel et al. (1998) [28] concentrent sur la technique connue sous le nom nichage. Niching est basé sur la métaphore de la nature que les différentes espèces ou sous-populations ont différentes niches. Il n'y a pas de concurrence entre les niches, mais dans un créneau il y a compétition. Dans AG, niching permet différentes parties de la fonction fitness à être explorées en parallèle, avec la convergence à plusieurs plutôt qu'une seule fonction maximum.

Enfin Ole J. Mengshoel et al. (1998) [28] présentent des résultats théoriques et empiriques liés à la fonction de fitness sharing et scaling. En particulier, ils montrent que nichage combiné avec scaling de manière significative améliore la performance d'un algorithme génétique pour le réseau bayésien.



## **5. Amélioration de l'efficacité d'un algorithme génétique appliquée à l'opération de Tactic Multi-Robot :**

Gustavo et al. (2010) [17] ont examiné un algorithme génétique pour accomplir la formation d'une équipe robotique qui doit effectuer une tâche de lutte contre l'incendie; ils ont [17] évalué les caractéristiques comme la structure des chromosomes, le taux de mutation, les types de mutation, nombre de personnes et le taux de recouvrement.

Cependant, deux facteurs qui a une influence fortement l'efficacité de la recherche génétique n'a pas été examinée, qui est : chevauchement des populations et de fitness scaling. La tâche robotique est effectuée sur un scénario de catastrophe naturelle (Une simulation d'incendie de forêt). La mission robot squad entoure la propagation de l'incendie et d'éviter le feu sur la base de la stratégie proposée par l'AG. Des simulations ont été réalisées avec plusieurs paramètres GA (Plusieurs types de scaling et différents degrés de chevauchement) afin d'obtenir l'optimisation la plus efficace pour la formation de groupe et de l'exécution des tâches. Simulations résultats montrent que l'utilisation de la population qui se chevauchent et de fitness scaling actuelle mieux.

## **6. Améliorer la diversité des algorithmes génétiques pour la sélection d'entité améliorée :**

Akram et al. (2010) [2] montrent que convergence prématurée à une grande relation avec la diversité de la population. Simple recherche GA commence habituellement avec la population diversifiée mais après un nombre d'itérations, de nombreux membres de la population ont tendance à porter des solutions similaires et convergent vers un certain point. Selon certaines études, les tentatives pour résoudre ce problème par une des opérations suivantes Techniques: (i) fitness scaling et sélection de parent, et (ii) opérateurs de croisement et de mutation améliorés et l'introduction recherche hybride.

Relative fitness scaling Algorithme Génétique(GARF) est inspirée par la loi de puissance inverse relation qui se produit dans la science physique et sociale, représentée dans l'équation suivante :

$$f(x) = \frac{x}{\sqrt{1 - \frac{x^2}{c^2}}} \quad (5)$$

Où  $x$  représente fitness de la population et  $c$  est le facteur de plafond qui peut prendre des valeurs supérieures ou égale à la remise en forme maximale possible. A noter que lorsque  $c$  est très élevée, l'effet du facteur d'échelle est minimale ( $f(x) \approx x$ ). De cette façon, relative scaling donne plus les individus d'une petite ou moyenne fitness et un conditionnement physique très élevé valeur à quelques individus. Cette scaling préserve la population la diversité en créant de petites différences entre petits et grands les membres de la valeur de fitness, et évite donc de détruire impropres individus au début du processus. Une fois les solutions candidats abordent une valeur de remise en forme à proximité de  $c$ , ils vont dominer la roue de roulette.

Et ils [2] présentent un nouvel algorithme génétique modifié basé sur le renforcement de la diversité de la population, la sélection des parents et des opérateurs génétiques améliorés. Les résultats pratiques indiquent l'importance de la variante AG proposée par rapport à beaucoup d'autres algorithmes de la littérature sur les différents ensembles de données.

## **7. Algorithmes Génétiques: Qu'est-ce que fitness scaling est optimal?**

Vladik et al. (2010) [37] Supposent de maximiser une objective fonction  $J(x)$ . Ils choisissent en quelque sorte la génération première des « individus »  $x_1; x_2; \dots; x_n$  (i.e., possible les valeurs de  $x$ ), et calculer « le fitness »  $J(x_i)$  de toutes ces personnes. Pour chaque individu  $x_i$ , nous attribuer une probabilité  $p_i$  de survie qui est proportionnelle à son aptitude physique. Afin d'obtenir la prochaine génération, nous répétons alors les temps procédure  $k$  suivantes: prendre deux individus au hasard (i.e.,  $x_i$  avec la probabilité  $p_i$ ) et « combiner » eux selon une règle. Pour chaque individu de cette nouvelle génération, ils calculent aussi son fitness (et la probabilité de survie), « combiner » eux pour obtenir la troisième génération, etc. Dans certaines conditions raisonnables, la valeur de la fonction objective augmente de génération en génération et converge vers une valeur maximale.

L'utilisation  $f(J(x_i))$  à la place de  $J(x_i)$  en tant que valeur fitness, où  $f(x)$  est une fonction déterminée qui est appelé une fonction de scaling. L'efficacité de fitness scaling dépend essentiellement du choix de  $f$ .

Les mécanismes de scaling utilisés maintenant :

1) Scaling linéaire, où  $f(z) = az + b$ . Les constantes  $a$  et  $b$  sont choisis soit au début, de la procédure, ou recalculée pour chaque génération. Pour des problèmes d'ordre une telle procédure a été proposée et utilisée dans [6].

2) Power law scaling, où  $f(z) = z^\alpha$  pour une constante  $\alpha$ . Ce type de mise à l'échelle a été proposé par [15], qui a montré que, pour les applications de machine vision la meilleure valeur pour  $\alpha$  est 1.005; pour d'autres problèmes d'autres valeurs de  $\alpha$  peut conduire à de meilleurs résultats

3) Scaling exponentiel, où  $f(z) = \exp(-\beta z)$  pour certains  $\beta$ . Ce type de scaling est motivé par une analogie avec le recuit simulé. Ce type de scaling est le plus couramment utilisé en robotique.

Ils formulent le problème du choix  $f$  comme une optimisation mathématique et le résoudre sous différents critères d'optimalité. En conséquence, ils obtiennent une liste de fonctions  $f$  qui sont optimales selon ces critères. Cette liste inclut les fonctions qui ont été empiriquement avéré être le meilleur pour certains problèmes, et quelques nouvelles fonctions qui peuvent être la peine d'essayer.

## **8. Comparaison des fonctions de fitness scaling dans les algorithmes génétiques avec des applications à traitement optique :**

Farzad A.Sadjadi (2004) [13] explore de nombreuses tâches optiques ou d'images réduisent à l'optimisation de certain ensemble de paramètres. Les algorithmes génétiques peuvent optimiser ces paramètres, même lorsque les fonctions qu'ils tracent sont assez compliqués, mais ils ne peuvent le faire le point où les fonctions de remise en forme, ils sont donnés peuvent se différencier entre les bons résultats et le meilleur résultat. Cela peut se produire lorsque le point optimal se trouve dans une zone (dans un exemple tridimensionnel), tel qu'un plateau, où tous les points environnants sont de très près la même forme physique. S'il y a plusieurs pics à proximité, tous à peu près la même forme physique, mais avec divise très profondes, l'algorithme aura du mal à «sauts» de l'un à l'autre. Une façon de surmonter ces obstacles est à l'échelle des valeurs de conditionnement physique donnés par la fonction de fitness, ce qui modifie légèrement la fonction de fitness du point de vue de l'algorithme, récompensant ainsi les solutions plus aptes à une précision supérieure à celle qui naturellement se produire.

Quatre de ces méthodes de scaling sont comparées en fonction de leur manutention d'un ensemble de données de traitement optique de l'échantillon. Le succès sera déterminé en comparant la variance dans le temps, la pression de sélection au cours du temps, et le meilleur des graphes de génération.

## 9. Transformation ranking: une nouvelle méthode de fitness scaling dans les algorithmes génétiques :

A.A. Hopgood et al. (2009) [4] présentent la première évaluation systématique des effets des six formes existantes de fitness scaling dans les algorithmes génétiques :

- **Linéaire générique scaling:**

Ceci est une relation linéaire simple entre fitness scaling,  $s_i$ , et raw fitness  $f_i$

- **Sigma scaling :**

Est une variante de scaling linéaire où l'aptitude d'un individu est mise à l'échelle en fonction de son écart par rapport à la fitness de la population, mesurée en les écarts-types (à savoir, 'sigma',  $\sigma$ ).

- **Boltzmann scaling:**

Est une méthode non linéaire qui utilise l'idée d'une "température",  $T$ , lentement que les gouttes de génération en génération.

- **Linéaire rang scaling:**

En linéaire rang scaling, les fitnesses scaling sont uniformément réparties basées sur le rang ordonnancement des chromosomes du plus fort au moins en forme.

- **Non linéaire rang scaling:**

Ceci est une forme non linéaire de rang scaling qui augmente la pression de sélection.

- **Probabilistes non linéaire rang scaling :**

Ceci est une forme intégré le rang non linéaire scaling dans la sélection de la roue de la roulette et SUS, plutôt que de le traiter comme une première étape distincte.

Les fonctions de test choisies étaient les fonctions à deux dimensions Schwefel et Griewank.

Ils ont proposé [4] une nouvelle forme de rang scaling, transformation de ranking, qui progresse de plus en plus à peu près de linéaire à non linéaire.

Ils ont fourni [4] la meilleure amélioration de la qualité de la recherche pour la fonction Griewank, et était deuxième à l'échelle probabiliste de rang non linéaire pour la fonction Schwefel. Sélection du tournoi, par comparaison, était toujours l'option de calcul moins cher, mais ne trouve pas nécessairement les meilleures solutions.

Dans le tableau (Tableau 2.1) ci-dessous nous présentons quelques œuvres réalisées en la matière :

L'auteur et l'année	L'objet	Commentaire
Deepti et al. (2012) [10]	Un aperçu des méthodes de maintien de la diversité dans les algorithmes génétiques	AGs est appliquée à une grande variété de recherche et d'optimisation des problèmes dans divers domaines de la science, de l'ingénierie et de la technologie.
Di-Wei Huang et al. (2010) [11]	Scaling des populations d'un algorithme génétique pour problèmes Job Shop Scheduling utilisant MapReduce	AGs avec les populations massives fournir une nouvelle possibilité en vue de résoudre les problèmes difficiles, et il peut être obtenu en utilisant MapReduce en cours d'exécution sur un cluster de marchandises Matériel.
Bruno Sareni et al. (1998) [7]	Fitness sharing et méthodes Niching revisité	méthodes niching ont été développés pour réduire l'effet de la dérive génétique résultant de l'opérateur de sélection dans le AG standard.
Ole J. Mengshoel et al. (1998) [28]	Algorithmes génétiques pour le réseau bayésien: Le rôle de scaling et niching	Bien que le AGs sont particulièrement adaptés pour le calcul des explications les plus probables dans un BN avec des nœuds discrets, d'autres formes de calcul évolutif pourrait être adapté pour des tâches d'inférence et d'apprentissage étroitement liées

Gustavo et al. (2010) [17]	Amélioration de l'efficacité d'un algorithme génétique appliquée à l'opération de Tactic Multi-Robot	Les résultats des simulations montrent que l'utilisation de la population de chevauchement et la forme physique mise à l'échelle présente de meilleurs résultats que la population non-chevauchement et conditionnement physique unscaled.
Akram et al. (2010) [2]	Améliorer la diversité des algorithmes génétiques pour la sélection d'entité améliorée	algorithme DGA sera encore développé à l'avenir pour améliorer la sélection des parents procédure et de fusionner une meilleure option en cas de besoin.
Vladik et al. (2007) [37]	Algorithmes Génétiques: Qu'est-ce que fitness scaling est optimal?	si l'on applique un algorithme génétique avec deux équivalents battitures, on obtient le même résultat dans les deux cas.
Farzad A.Sadjadi (2004) [13]	Comparaison des fonctions de fitness scaling dans les algorithmes génétiques avec des applications à traitement optique	Fitness scaling peut offrir le plus d'avantages à une tâche générique de traitement
A.A. Hopgood et al. (2009) [4]	Transformer Classement: une nouvelle méthode de fitness scaling dans les algorithmes génétiques	transformation de classement, progresse de plus en plus à peu près linéaire à non linéaire

**Tableau 2.1** : Récapitulatif des travaux sur les AGs.

## **II. Conclusion:**

Afin de situer notre travail, nous avons présenté dans ce chapitre un bref état de l'art des algorithmes génétiques et de l'intégration du scaling et du sharing, ainsi que leurs cas d'applications. Il est à signaler que les cas d'utilisation de ces processus sont nettement rares malgré leur apport en matière de qualité de solutions. Cette pénurie en la matière est peut être due d'une part au temps de calcul engendrée par l'ajout de ces deux opérateurs et d'autre part par l'avènement d'hybridation d'algorithmes génétiques avec d'autres méthodes qui s'impose de plus en plus dans l'optimisation combinatoire.

## 1. Introduction :

Après avoir présenté certaines méthodes pour résoudre les problèmes d'optimisation combinatoire et l'état de l'art dans les chapitres précédents, Nous arrivons dans le présent chapitre à la proposition de notre conception, en utilisant les méthodes de résolutions de problème les algorithmes génétiques et les algorithmes de scaling et de sharing. Nous commençons la description de notre conception par la démarche de notre travail. Ensuite, nous présenterons les étapes de la conception proposée. Puis nous intéressons à adapter les algorithmes génétiques pour résoudre le problème de voyageur de commerce. Enfin, la conclusion.

## 2. Démarche

La section qui suit décrit les divers éléments de conception de l'approche évaluée dans le cadre de la présente étude. La première sous-section décrit l'algorithme génétique. La seconde sous-section décrit l'algorithme de scaling. La troisième sous-section décrit l'algorithme de sharing. Nous fournissons une spécification puis une formulation mathématique du PVC afin qu'on puisse mesurer l'efficacité de notre approche.

## 3. Les étapes de conception :

### 3.1 l'algorithme génétique :

Le programme d'évolution est un algorithme probabiliste qui maintient une population d'individus,  $P(t) = \{x_1^t, \dots, x_n^t\}$  pour l'itération  $t$ . Chaque individu représente une solution potentielle au problème à portée de main, et dans tout programme d'évolution, est mis en œuvre comme certains (éventuellement complexe) structure de données  $S$ . Chaque solution  $x_i^t$  est évaluée pour donner une mesure de sa "fitness". Ensuite, une nouvelle population (itération  $t + 1$ ) est formée en sélectionnant les individus plus en forme (select étape). Membre de la nouvelle population subissent des transformations (modifier l'étape) par l'intermédiaire d'opérateurs «génériques» pour former de nouvelles solutions.

Structure de base d'un algorithme génétique :

Algorithme Génétique

Debut

Initialize (iter)

Evaluation (iter)

pour iter = 1 à niter



```
Selection (iter)
TSPCross (iter)
Mutation (iter)
Fin pour
Fin
```

**Algorithme 3.1.** Algorithme génétique.

**3.2 Scaling :**

La technique de scaling (ou mise à l'échelle) permet de diminuer ou d'augmenter artificiellement les écarts d'évaluation entre les individus afin d'ajuster la pression sélective des techniques de sélection.

```
Fonction scaling ()
Debut
Si iter mod 50=0 alors
    sum_fitness = 0
    ks = Math.Tan (Math.PI * iter * 0.5 / (niter + 1)) ^ 0.1
    pour i = 1 à taille
        Fitness (i) = fitness(i) ^ ks
        sum_fitness = sum_fitness + fitness (i)
    Finpour
Fin Si
Fin
```

**Algorithme 3.2.** Algorithme de Scaling.

**3.3 . Sharing :**

Le sharing modifie l'adaptation des individus. Ce dernier pénalise un individu en fonction du taux d'agrégation de la population dans son voisinage.

```
Fonction sharing ()
Debut
Si d >= n alors
```

```
    sh = 0
Sinon
    sh = 1 - (d / n) ^ 0.9
FinSi
Fin
Fonction distance ()
Debut
    s = 0
    Pour i = 1 à n
        s = s + Math.Abs (TSP(x1, i) - TSP(x2, i))
    distance = s
Fin
Fonction mi ()
Debut
    s = 0
    Pour i = 1 à taille
        s = s + sharing (distance (i, j))
    Fin pour
Fin
```

**Algorithme 3.3.** Algorithme de sharing

#### 4. Problème du voyageur de commerce :

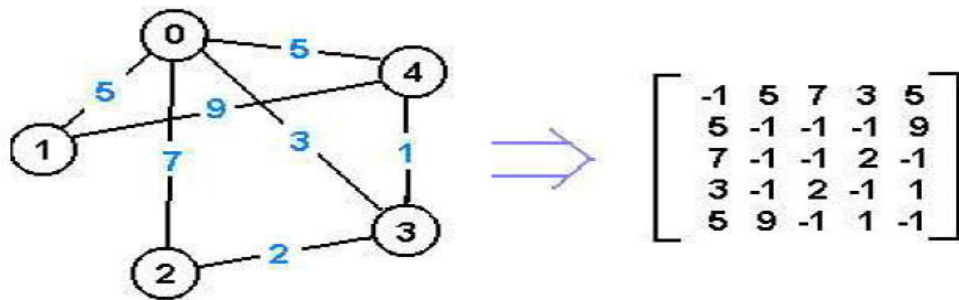
Nous allons maintenant nous intéresser à une application plus concrète : le problème du voyageur de commerce. Cet exemple est un classique appartenant à la classe des problèmes NP-complets.

Il consiste à visiter un nombre  $N$  de villes en un minimum de distance sans passer deux fois par la même ville. Il s'agit donc d'optimiser le coût d'un parcours dans un graphe complet possédant un certain nombre de sommets, en passant une et une seule fois par chacun. Des méthodes déterministes existent déjà pour résoudre le problème, mais le temps de calcul est très long : elles reviennent à parcourir toutes les solutions possibles et à déterminer la moins coûteuse [34].

Le but sera ici de montrer comment modéliser le problème à partir d'algorithmes génétiques et des divers opérateurs que nous avons à disposition, qui ont été définis antérieurement [34].

Formellement, à partir d'une matrice  $C = cij$  où  $cij$  représente le coût du déplacement de la ville  $i$  à la ville  $j$   $1 \leq i, j \leq n$ , il faut trouver une permutation  $\begin{pmatrix} 1 & 2 & \dots & n \\ \sigma_1 & \sigma_2 & \dots & \sigma_n \end{pmatrix}$  qui minimise la somme  $\sum_{i=1}^{n-1} c\sigma(i+1) + c\sigma(n)\sigma(1)$  (Le coût d'une tournée est égal à la somme des coûts de tous les arcs appartenant à la tournée) [34].

**Exemple :**



**Figure 3.14 :** modélisation du graphe sous forme de matrice

**L'espace de recherche :**

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

**Codage des points de l'espace de recherche :**

Une première idée serait de coder chaque permutation (i.e : chaque point de l'espace de recherche) par une chaîne de bits [34].

Par exemple, pour  $n = 8$  :

0011 0111 0000 0100 0001 0010 0101 0110

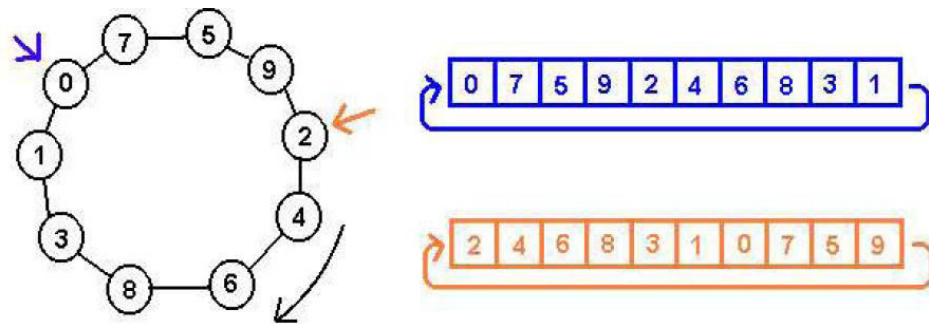
Représente la permutation :

3 7 0 4 1 2 5 6

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

### Représentation d'une solution :

Comme nous l'avons déjà dit le voyageur de commerce doit revenir à son point de départ et passer par toutes les villes une fois et une seule. Nous avons donc codé une solution par une structure de données comptant autant d'éléments qu'il y a de villes, c'est à dire une permutation. Chaque ville y apparait une et une seule fois. Il est alors évident que selon la ville de départ que l'on choisit on peut avoir plusieurs représentations différentes du même parcours [34].



**Figure 3.15 :** codage d'une solution (ensemble de villes) dans un tableau.

### Sélection :

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

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

Où  $P_i$  est l'individu  $i$  de la population et  $m$  la taille de la population. La place d'un individu  $P_i$  dans la roulette est proportionnel à  $\frac{f(P_i)}{\bar{f}}$ . On sélectionne alors  $m/2$  individus pour la reproduction. Il y a aussi la possibilité d'avoir une politique d'«élitisme». C'est à dire qu'à chaque étape de sélection le meilleur chromosome est automatiquement sélectionné [34].

### Croisement :

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

Nous pouvons suivre la méthode suivante :

1. On choisit aléatoirement deux points de découpe.
2. On interverti, entre les deux parcours, les parties qui se trouvent entre ces deux points.
3. On supprime, à l'extérieur des points de coupe, les villes qui sont déjà placées entre les points de coupe.
4. On recense les villes qui n'apparaissent pas dans chacun des deux parcours.
5. On remplit aléatoirement les trous dans chaque parcours [34].

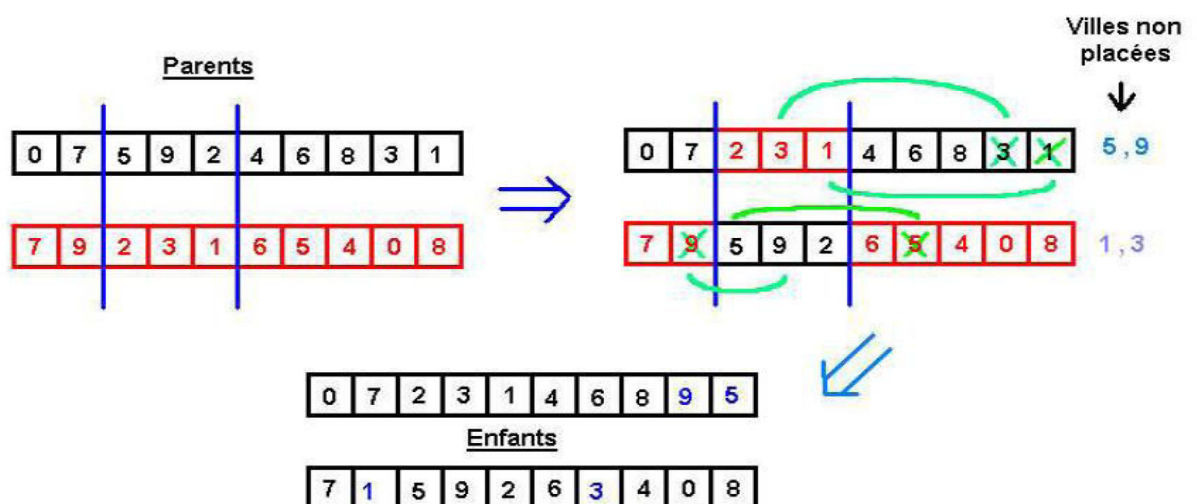
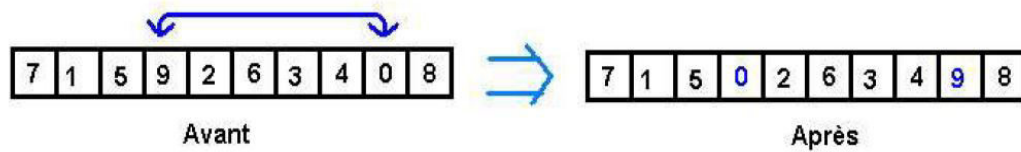


Figure 3.16 : exemple de croisement.

### Mutation :

Il s'agit ici de modifier un des éléments d'un point de l'espace de recherche, soit d'une permutation.

Dans notre cas, cela correspond donc à une ville. Quand une ville doit être mutée, on choisit aléatoirement une autre ville dans ce problème et on intervertit les deux villes.



**Figure 3.17 :** exemple de mutation

### Calcul des fitness :

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

Lorsque cette dernière éventualité se produit, il faut que le comportement suivant soit respecté, à savoir : plus notre chemin sera court, plus la fitness qui lui est associée sera forte.

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

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

**ALGORITHME ABSTRAIT :**

Il s'agit dans un premier temps d'établir un ensemble de gènes. Ceci étant fait, il faut alors définir deux fonctions, qui respectivement permettront de croiser deux ensembles de gènes et de réaliser une mutation sur deux ensembles de gènes [34].

Fonction croisement ()

Begin

Pour  $i = 1$  à taille

$r = \text{rnd}()$

Si  $r \leq pc$  alors

$j = 1$

tantque  $j < p$

$x = \text{pool}(i, z + j + 1)$

$\text{Pool}(i, z + j + 1) = \text{pool}(i, z + j)$

$\text{Pool}(i, z + j) = x$

$j = j + 2$

Fin tant que

Fin si

Fin

**Algorithme 3.4.** Algorithme de croisement

Et ci-dessous la procédure de mutation :

Fonction mutation ()

Debut

Pour  $i = 1$  à taille

$r = \text{rand}()$

Si  $(r \leq pm)$  alors

```
z1 = (rand ()*w) +1

z2 = z1 + 2

Si (z2 >= w) alors

    z2 = 1

Fin Si

x = TSP (m, z1)

TSP (m, z1) = TSP (m, z2)

TSP (m, z2) = x

Fin Si

Fin pour

Fin
```

### **Algorithme 3.5.** Algorithme de mutation

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

Voici donc une procédure sélection qui renvoie les éléments les plus intéressants d'un génome (ce sera dans le cas du voyageur de commerce, les éléments de fitness la plus grande). Le choix est sinon laissé au programmeur [34].on choisit la selection par roulette.

Fonction roulette()

```
debut
    s1 = 0
    r = Rnd() * sum_fitness
    i = 0
    tantque s1 < r
        i = i + 1
        s1 = s1 + fitness(i)
```



```
Fin tant que
roulette = i
End
```

Fonction Selection ()

Début

```
Pour i = 1 à taille
  x = roulette ()
  y = roulette ()
  Si fitness(x) > fitness(y) alors
    Pour j = 1 à n
      pool1 (i, j) = pool(x, j)
  Sinon
    Pour j = 1 à n
      pool1 (i, j) = pool(y, j)
  Fin SI
pour i = 1 à el
  Pour j = 1 à n
    pool1 (taille + i, j) = pool (indsol, j)
Fin
```

**Algorithme 3.6.** Algorithme de Sélection

**Comparaison de complexités :**

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

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

- les algorithmes déterministes qui trouvent la solution optimale
- les algorithmes d'approximation qui fournissent une solution presque optimale complexité via une méthode de résolution classique (méthode déterministe) :

Un calcul rapide de la complexité montre qu'elle est en  $O(n!)$  où  $n$  est le nombre de villes.

En supposant que le temps pour effectuer un trajet est d' 1  $\mu$ s, le tableau 3.1 témoigne de l'explosion combinatoire [34].

Nb villes	Nb possibilités	Temps de calcul
5	120	120 $\mu$ s
10	181440	0.18 ms
15	43 MILLIARDS	12 h
20	60 E+15	1928 ans
25	310 E+21	9.8 Milliards A.

**Tableau 3.1 :** Complexité d'une méthode exacte.

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

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

Nb villes	Nb possibilités	Temps de calcul
5	25	120 $\mu$ s
10	100	100 $\mu$ s
15	225	225 $\mu$ s
20	400	400 $\mu$ s

**Tableau 3.2 :** Complexité d'un AG.

Les résultats entre la méthode déterministe et approximative ne sont pas comparables en soi, elles utilisent des méthodes différentes pour arriver à des résultats différents. Comme dis précédemment, dans un cas, on cherche la solution optimale, dans le second (tableau 3.2), une solution presque optimale.

On s'aperçoit bien à l'aide de ces calculs que pour parvenir à ces solutions on consomme respectivement un temps exponentiel (explosion combinatoire) et un temps polynomial (quadratique) par rapport au nombre de villes [34].

**Algorithme générale :**

Début

```
Initialisation()
Evaluation()
  Pour iter= 1 à niter faire
    Sélection ()
    Croisement ()
    Mutation()
    Si iter mod 50 alors scaling()
    Si iter mod 100 alors sharing ()
  Evaluation ()
Finpour
Fin
```

**Algorithme 3.7.** Algorithme générale

## 5. Conclusion :

Les algorithmes génétiques seuls ne sont pas très efficaces dans la résolution du PVC. Ils apportent cependant assez rapidement une solution acceptable, même dans le cas d'une instance du problème non-euclidienne. Néanmoins, il est possible de l'améliorer assez efficacement en le combinant avec un algorithme déterministe sur les chromosomes de la population.

## 1. Introduction :

L'objet de ce chapitre est de présenter les contributions de notre recherche tant sur un plan théorique que pratique à l'usage des gestionnaires de projet. Il finit par une discussion sur les limites de l'étude de cas et fournit quelques pistes de réflexions pour de futures recherches.

## 2. les outils de développement:

### 2.1. Fortran :

Fortran (*FORmula TRANslator*) est un langage de programmation utilisé principalement pour le calcul scientifique. Inventé en 1954, c'est le plus ancien langage de programmation de haut niveau, suivi notamment par Lisp (1958), Algol (1958) et COBOL(1959). Le nombre de bibliothèques scientifiques écrites en Fortran, et les efforts continus consacrés aux compilateurs pour exploiter au fil des décennies les nouvelles possibilités des calculateurs (vectorisation, coprocesseurs, parallélisme) ont maintenu l'usage de ce langage, non sans d'importantes évolutions [43].

Fortran 95 est une version du langage de programmation Fortran, Toutefois, ses utilisateurs ont voulu y ajouter au fil du temps les idées intéressantes utilisées dans d'autres langages. Cela a donné les versions Fortran 90, puis Fortran 95 [42].

Fortran 95 reprend les extensions déjà définies dans Fortran 90 et y ajoute :

- Ajout principal :
  - Ordre et construction *FORALL*
- Ajouts secondaires :
  - fonction CPU\_TIME
  - libération mémoire automatique des tableaux dynamiques
  - la fonction SIGN distingue entre +0 et -0

### 2.2. Gnuplot :

Gnuplot est un logiciel qui sert à produire des représentations graphiques en deux ou trois dimensions de fonctions numériques ou de données. Le programme fonctionne sur de nombreux ordinateurs et systèmes d'exploitation (Linux, Windows, OS/2,VMS...) et peut envoyer les graphiques à l'écran ou dans des fichiers dans de nombreux

formats. Gnuplot utilise également l'algorithme de Levenberg-Marquardt pour ajuster les paramètres d'une fonction numériques sur des données expérimentales [44].

Le programme est distribué sous une licence de logiciel libre qui permet de copier et de modifier le code source du programme. Les versions modifiées du programme ne peuvent être distribuées que sous forme de fichiers correctifs. Le programme n'a aucun raccordement avec le projet GNU et n'utilise pas la licence de copyleft GPL [44].

Le programme peut être utilisé interactivement, et est accompagné d'une aide en ligne. L'utilisateur entre en ligne de commande des instructions qui ont pour effet de produire un tracé. Il est aussi possible d'écrire des scripts gnuplot qui, lorsqu'ils sont exécutés, génèrent un graphique [44].

### 3. Expérimentations et analyse :

Pour montrer l'effet apporté par le scaling et le sharing à la performance des algorithmes génétiques, il faudrait en premier tester l'algorithme avec le sharing et le scaling, et en deuxième lieu sans les utilisés. Après la collections des données, on procède à les représenter avec des graphes adéquats, enfin, on analyse ces résultats et on les interprètes.

- **Test 01 :** Le but de voir l'effet de la taille de la population sur l'apport du sharing et scaling sur la performance, quand le sharing et scaling sont présents et quand ils ne sont pas utilisés.

Les paramètres sharing et scaling sont choisis comme suit : sh = 100 & scl = 1, les résultats sont reportés sur le tableau suivant (tableau 4.1,4.2) et montré sur le graphe figure 4.1 suivant :

Nombre de population	Solutions optimales/iteration	Solutions optimale trouvée
50	4386	4386.00000
100	4032	4032.00000
150	3813	3813.00000

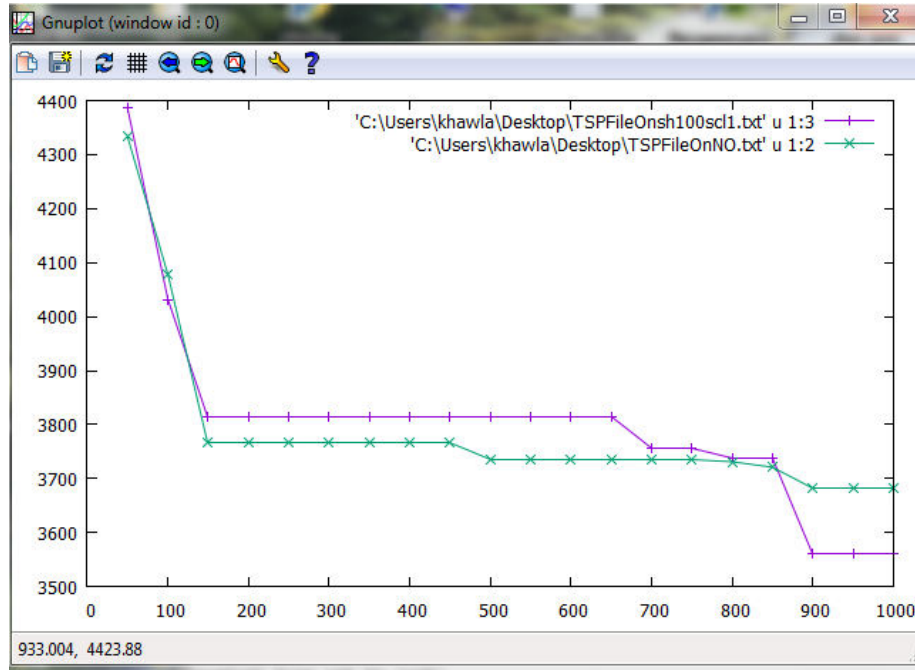
200	4486	3813.00000
250	4052	3813.00000
300	3855	3813.00000
350	4106	3813.00000
400	4176	3813.00000
450	3898	3813.00000
500	3918	3813.00000
550	3840	3813.00000
600	3931	3813.00000
650	3845	3813.00000
700	3755	3755.00000
750	3985	3755.00000
800	3738	3738.00000
850	3918	3738.00000
900	3562	3562.00000
950	3800	3562.00000
1000	3813	3562.00000

**Tableau 4.1** Expérimentation sharing=100 ,scaling=1.

<b>Nbr de generation</b>	<b>Solutions optimale trouvée</b>
50	4333.00000

100	4078.00000
150	3768.00000
200	3768.00000
250	3768.00000
300	3768.00000
350	3768.00000
400	3768.00000
450	3768.00000
500	3735.00000
550	3735.00000
600	3735.00000
650	3735.00000
700	3735.00000
750	3735.00000
800	3731.00000
850	3721.00000
900	3681.00000
950	3681.00000
1000	3681.00000

**Tableau 4.2** Expérimentation sans scaling et sans sharing.



**Figure 4.1 l'influence des paramètres de sharing et scalling**

On distingue trois parties sur le graphe qui définissent le comportement relatif des deux courbes des deux tests.

**Partie 01 :  $n \in [50, 150]$**

Les deux courbes ont la même allure avec une performance meilleure que pour la courbe du test quand les paramètres de sharing et scaling ne sont pas utilisés.

**Partie 02:  $n \in [150, 850]$**

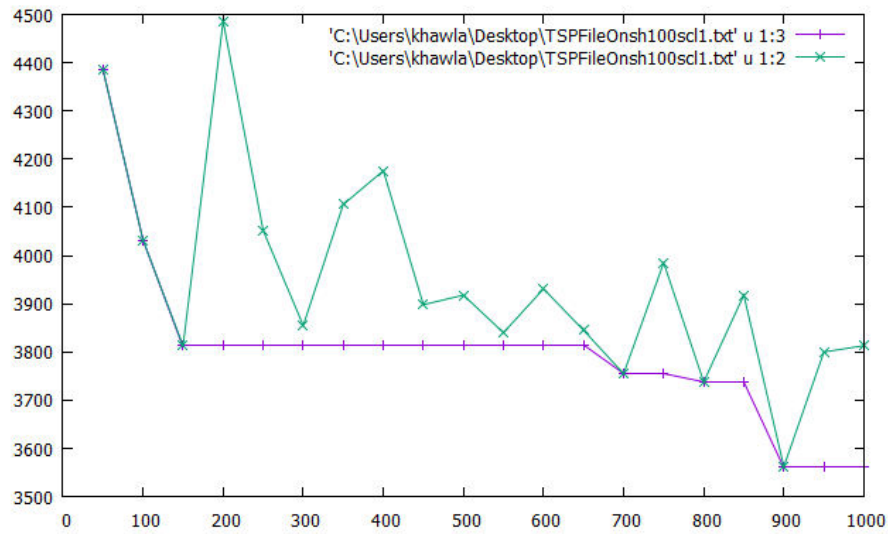
Même remarque dans cette zone, les deux courbes présentent une variation presque similaire en fonction du nombre des particules. On remarque toujours que la meilleure solution est donnée dans le cas où ni le sharing, ni le scaling sont utilisés.

**Partie 03:  $n \in [850, 1000]$**

Le point où la taille de la population est  $n=850$  particules est critique dans notre expérimentations. L'effet des paramètres sharing et scaling se manifeste et les deux courbes échange leur comportements, la courbe du sh\_scl présente une pente décroissante très prononcée qui donne le privilège au cas du sharing et scaling d'avoir la solution optimal jusqu'ici.



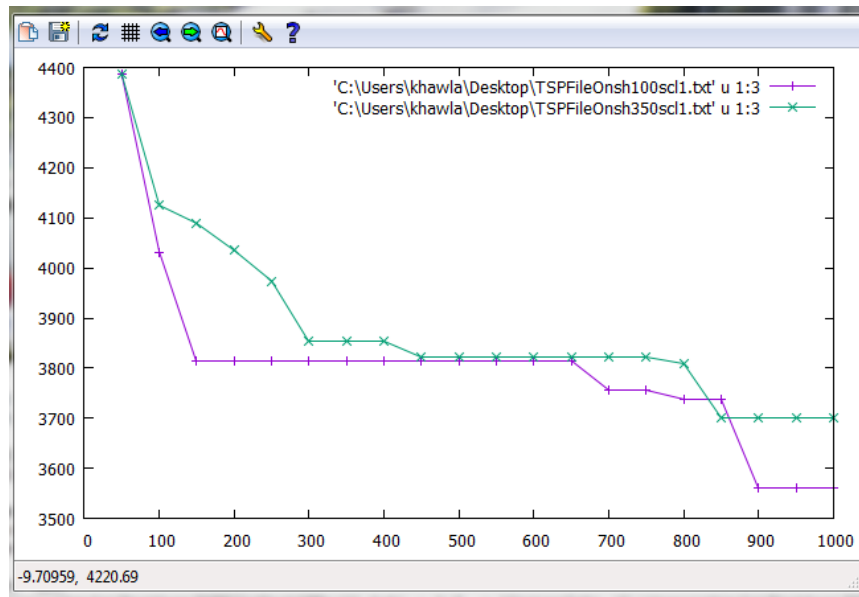
Pour mieux apprécier les solutions optimales trouvées pour chacune des populations considérés on présente le graphe suivant :



**Figure 4.2 génération des solutions optimales par la taille de population**

Les trois zones discutées juste au-dessus, sont bien visibles et bien expliquées dans ce graphe, où on remarque que : la première partie de la courbe présente une décroissance accentuée vers une solution qui restera stable car les populations dans la deuxième partie n'ont pas pu la dépasser, et ceci de la taille de la population à  $n=150$  à  $n=650$ , en  $n=700$  la solution est de nouveau améliorée et continue légèrement s'améliorer jusqu'à  $n=850$ . En  $n=900$  une brusque chute de la courbe améliore nettement la solution optimale.

- **Test 02 :** dans ce test on va explorer l'influence du sharing sur la performance de l'algorithme génétique pour trouver les solutions optimales suivant la taille de la population considérée. Pour cela on fixe le scaling à la valeur  $scl = 1$  et le sharing on le lui attribue les deux valeurs  $sh = 100$  et  $350$ . Le graphe suivant présente la variation des deux courbes suivants la taille croissante de la population.

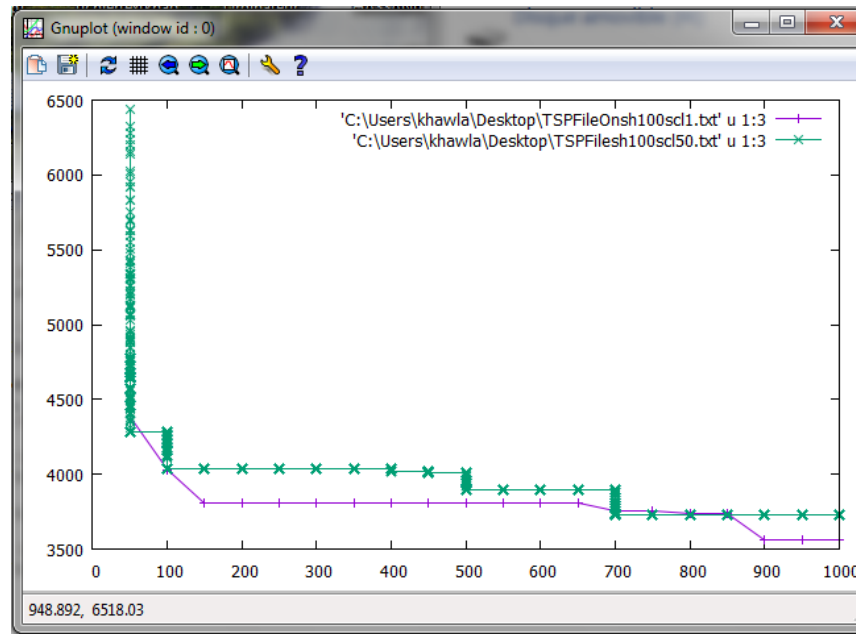


**Figure 4.3 l'influence du sharing**

On remarque que sur les trois zones du comportement trouvées dans l'étude précédente sont bien visible ici aussi, néanmoins, c'est la courbe du sharing  $sh = 100$  qui est meilleure dans les trois zones, surtout, dans la dernière, où la différence s'accroît en faveur du sharing 100. De ce fait, on peut comprendre que les valeurs larges du sharing sont moins intéressantes que des valeurs raisonnables telles que  $sh = 100$ .

- **Test 03 :**

Dans cette expérimentation on s'intéresse à l'impact du scalling et donc, on prend deux valeurs  $scl = 1$  et  $scl = 50$ , tout en fixant le sharing à la valeur  $sh = 100$ . Le graphe suivant présente les résultats trouvés.



**Figure 4.4 l'influence du scaling**

On remarque toujours le comportement des deux courbes qui se décrit en trois zones, seulement ici, la courbe du  $scl = 1$  comprend toujours les meilleure solutions optimales que celle de la courbe avec  $scl = 50$ . Donc, il faudrait avoir une valeur pas trop large mais raisonnable du scaling pour avoir l'influence voulue sur l'algorithme génétique.

#### **4. Conclusion :**

Nous avons mené une étude sur l'influence du sharing et de scaling sur la performance de l'algorithme génétique à trouver des solutions optimales au problème considéré qui est le TSP(Travel Salesman Problem). On n'est arrivée à la conclusion que le sharing est le scaling ont une influence sensible sur la performance de l'algorithme néanmoins leurs valeurs doivent être bien étudiées pour sélectionner les meilleures qui peuvent apporter plus de performance à l'algorithme génétique.

## **CONCLUSION GENERALE :**

Dans ce travail, nous avons présenté l'étude de l'impact du scaling et du sharing sur l'efficacité des algorithmes génétiques.

Le premier chapitre, nous avons introduit un rappel sur le problème d'optimisation combinatoire et ces méthodes. Nous avons vu les algorithmes génétiques qui sont des procédures assez robustes pour résoudre un problème d'optimisation. Néanmoins ils présentent certaines limites et difficultés qui influent fortement sur la convergence de l'algorithme. Ces difficultés reposent sur le choix des bons paramètres tels que : la taille de la population, le nombre de générations, les probabilités de croisement et de mutation et les méthodes des opérateurs de reproduction. Puis nous avons discuté les améliorations classiques (scaling et sharing).

Dans le deuxième chapitre, nous avons présenté un bref état de l'art des algorithmes génétiques et de l'intégration du scaling et du sharing, ainsi que leurs cas d'applications.

Dans le chapitre trois, nous avons exposé la proposition de notre conception, en utilisant les méthodes de résolutions de problème les algorithmes génétiques et les algorithmes de scaling et de sharing.

D'après les résultats obtenues dans le quatrième chapitre on peut conclure que le choix des paramètres tel que : le nombre de générations, probabilité de croisement, probabilité de mutation, le nombre de mutations ont une grande influence sur la convergence de l'algorithme. Cependant, notre intérêt était d'étudier l'influence des paramètres de sharing et de scaling sur la performance des algorithmes génétiques à trouver des solutions optimales, on a pu observer que ces paramètres peuvent avoir un impact assez important sur l'amélioration de la performance des algorithmes génétiques si leurs valeurs sont soigneusement choisies.

### **Perspectives :**

A partir du travail réalisé dans le cadre de ce mémoire, de futures recherches peuvent être faites pour élaborer une méthode plus évoluée pour l'extraction des paramètres de sharing et du scaling, afin d'avoir les meilleures valeurs pour augmenter la performance de l'algorithme génétique.

# Bibliographie

- [1] Abdeslam LAYEB, Utilisation des Approches d'Optimisation Combinatoire pour La Vérification des Applications Temps Réel. Thèse de Doctorat, Université Mentouri de Constantine 2010.
- [2] Akram AlSukker, Rami N. Khushaba, ET Ahmed Al-Ani. « Enhancing the Diversity of Genetic Algorithm for Improved Feature Selection » Systems Man and Cybernetics (SMC), IEEE International Conference on, 10-13 Oct. 2010, p.1325 – 1331.
- [3] Alain Hertz, 'L'optimisation Combinatoire', École Polytechnique, Canada, 2006
- [4] A. A. Hopgood et A. Mierzejewska. « Transform Ranking: a New Method of Fitness Scaling in Genetic Algorithms », Research and Development in Intelligent Systems XXV. London, Springer, (2009), p. 349-354.
- [5] Bäck T., Hammel U., Schwefel F-P., (1997). Evolutionary Computation: Comments on the history and current state. IEEE transactions on Evolutionary Computation, 1(1), 3-17
- [6] Baker, J. E. Adaptive selection methods for genetic algorithms, Proceedings of an International Conference on Genetic Algorithms and their Applications, Pittsburgh, PA, 1985.
- [7] Bruno Sareni and Laurent Krahenbuhl. « Fitness Sharing and Niching Methods Revisited »,IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 2, NO. 3, SEPTEMBER 1998.
- [8] Christelle Guéret, Christian Prins et Marc Sevaux, Programmation Linéaire, Eyrolles, 2000.
- [9] C.P.Vincent, « Heuristique - Création, intuition, créativité et stratégies d'innovation », BOD - Books on Demand France, 2012.
- [10] Deepti Gupta, Shabina Ghafir. «An Overview of methods maintaining Diversity in Genetic Algorithms », International Journal of Emerging Technology and Advanced Engineering Website: [www.ijetae.com](http://www.ijetae.com) (ISSN 2250-2459, Volume 2, Issue 5, May 2012)

- [11] Di-Wei Huang, Jimmy Lin, « Scaling Populations of a Genetic Algorithm for Job Shop Scheduling Problems using MapReduce », 2nd IEEE International Conference on Cloud Computing Technology and Science, 2010.
- [12] E.Taillard, «Introduction aux méta-heuristiques », Cours 3eme cycle informatique, Haute Ecole d'Ingénierie et de Gestion du Cantone De Vaud, Genève, 2007.
- [13] Farzad A. Sadjadi,« Comparison of fitness scaling functions in genetic algorithms with applications to optical processing », Proceedings of the SPIE, Volume 5557,(2004), p. 356-364 .
- [14] Fogel, L. J., Owens, A. J., Walsh, M. J., (1966). Artificial Intelligence through Simulated Evolution.
- [15] Gillies, A. M. Machine learning procedures for generating image domain feature detectors, Doctoral Dissertation, University of Michigan, Ann Arbor, MI, 1985.
- [16] Goldberg D. E., (1989a). Genetic Algorithms in search, optimization and machine learning. Addison-wesley Publishing Inc.
- [17] Gustavo Pessin, Fernando S. Osorio, Denis F. Wolf, ET Christiane R.S. «Improving Efficiency of a Genetic Algorithm Applied to Multi-robot Tactic Operation », IBERAMIA 2010, LNAI 6433, pp. 50–59.
- [18] HAJ-RACHID, Christelle BLOCH, Wahiba RAMDANE-CHERIF, Pascal CHATONNAY, « Différentes operateurs évolutionnaires de permutation: sélections, croisements et mutations » Thème 4 OMNI, Juillet 2010.
- [19] Jean-Marc Alliot, Nicolas Durand, «Algorithmes génétiques» March 14, 2005.
- [20] Lambert veller sylvain , lechevalier david,quirico tommy «Problème de ramassage dans une ville virtuelle - Algorithme Tabu Search »Université de Bourgogne 2010-2011
- [21] LAGGOUN ASSIA : développement d'une approche pour la résolution d'un problème de lot sizing avec transport. Thèse de magister, Université Hadj-Lakhdar – Batna.
- [22] Lambert veller sylvain, lechevalier david, quirico tommy «Problème de ramassage dans une ville virtuelle - Algorithme Tabu Search »Université de Bourgogne 2010-2011

- [23] Mohamed Amine ABID «Etude des algorithmes de recuit simule, de recherche tabou et genetique implemented dans un système de construction d'horaires de cours universitaires», Université de Sherbrooke(Quebec), CANADA, Decembre 2008.
- [24] Mostepha, R : Résolution de problèmes d'optimisation combinatoire par systèmes artificiels auto-organisés. Thèse de magister, Université Mentouri de Constantine ,2008.
- [25] M.Merdjaoui Brahim «Optimisation multi-objectif par algorithmes génétiques et approche Pareto des paramètres d'usinage sous contraintes des limitations de production »Université M'hamed BOUGARA Boumerdes, 2006.
- [26] Naima ZERARI: les algorithmes génétiques en maintenance, Ingénieur en Informatique, Université de Batna.
- [27] Nicolas DURAND «Algorithmes génétiques et autres outils d'optimisation appliqués à la gestion du trafic aérien » L'institut national polytechnique de Toulouse, 2004.
- [28] Ole J. Mengshoel and David C. Wilkins. « Genetic Algorithms for Belief Network Inference: The Role of Scaling and Niching», Proc. Seventh Annual Conference on Evolutionary Programming, 1998.
- [29] Omessaad, H : Contribution au développement de méthodes d'optimisation Stochastiques application à la conception des dispositifs électrotechniques, Thèse de Doctorat, Université De Lille France 2003.
- [30] Palpant M. : Recherche exacte et approchée en optimisation combinatoire : schémas d'intégration et applications. Thèse de Doctorat, Université d'Avignon, 2005.
- [31] Rachid Chelouah, 'L'optimisation combinatoire', INA Institut d'informatique appliquée, Suisse, 2003.
- [32] SAHA ADEL: Résolution des problèmes Multi Objectifs à Base de Colonies de Fourmi, Thèse de magister, Université De BATNA.
- [33] Sophie Jacquin: hybridation des métaheuristiques et de la programmation dynamique pour les problèmes d'optimisation mono et multi-objectif : application à la production d'énergie, Thèse de docteur de Lille 1,2015.
- [34] Souquet Amédée, Radet Francois-Gérard « Algorithme génétique ».

- [35] Thomas Rivière« Optimisation de graphes sous contrainte géométrique : création d'un réseau de routes aériennes pour un contrôle Sector-Less », L'institut national polytechnique de Toulouse, 2006.
- [36] Toufik BENDIB, «Modélisation et simulation du transistor DGMOSFET en utilisant les Algorithmes Génétiques», Thèse de magister, Université de Batna ,2008
- [37] Vladik kreinovich, Chris Quintana et Olac Fuentes. «Genetic algorithms: What fitness scaling is optimal», Cybernetics and Systems: An International Journal.Volume 24, 21 May 2007, p. 9-26.

## WEBGRAPHIE

- [38] <http://dspace.univ-biskra.dz:8080/jspui/bitstream/123456789/5235/2/mémoire.pdf>  
consulté le : 26/02/2016.
- [39] <http://iecl.univ-lorraine.fr/~Jean-Francois.Scheid/Enseignement/heuristiques.pdf> consulté  
le : 26/02/2016.
- [40] [http://www.memoireonline.com/10/12/6363/m\\_Contribution--loptimisation-complexe-par-des-techniques-de-swarm-intelligence8.html](http://www.memoireonline.com/10/12/6363/m_Contribution--loptimisation-complexe-par-des-techniques-de-swarm-intelligence8.html) consulté le : 28/02/2016.
- [41] <http://www.memoireonline.com/01/10/3084/Lutilisation-de-la-programmation-mathematique-pour-la-resolution-dun-probleme--car-sequenci.html> consulté le 16/03/2016.
- [42] [https://fr.wikipedia.org/wiki/Fortran\\_95](https://fr.wikipedia.org/wiki/Fortran_95) consulté le 22/04/2016.
- [43] <https://fr.wikipedia.org/wiki/Fortran> consulté le 22/04/2016.
- [44] <https://fr.wikipedia.org/wiki/Gnuplot> consulté le 22/04/2016.



### الملخص

الغرض من هذا البحث هو دراسة تأثير عمليتي سكالين و شارين على فعالية الخوارزميات الجينية . رغم تطورها، لا تزال الخوارزميات الجينية تتميز بمشاكلين أساسيين : التقارب المبكر وطول وقت الحساب. نستخدم الخوارزمية الجينية من خلال دمج هاتين العمليتين لمعالجة هذه العيوب عن طريق ثم نطبقها على PVC لقياس فعاليتها النتائج تبدو واعدة وتحديد وسائط هاتين العمليتين أمر بالغ الأهمية في جودة الحل.

**الكلمات المفتاحية:** الخوارزمية الجينية؛ رفع ومشاركة ؛ PVC ؛ الأمل اندماجي؛ الأدلة العليا.

### Résumé

L'objet de ce thème de recherche est l'étude de l'impact du scaling et du sharing sur l'efficacité des algorithmes génétiques. En dépit de leur évolution, les algorithmes génétiques demeurent révéler deux handicaps majeurs : la convergence prématurée et le temps de calcul. Nous utilisons un algorithme génétique en y intégrant les deux opérateurs : le scaling et le sharing afin de remédier à ces deux handicaps en l'appliquant au PVC pour en mesurer l'efficacité. Les résultats obtenus s'avèrent prometteurs et le paramétrage de ces deux procédés est nettement crucial.

**Mots clés:** Algorithme Génétique ; Scaling & Sharing ; PVC ; Optimisation Combinatoire; Méta Heuristiques.

### Summary

The purpose of this research theme is the study of scaling and sharing impact on the genetic algorithm effectiveness. Despite their evolution, genetic algorithms remain reveal two major handicaps: the premature convergence and computation time. We use a genetic algorithm by integrating the two operators: scaling and sharing to remedy these disadvantages and we apply this approach on TSP to measure its effectiveness. The results are promising and the parameters setting of these two processes is clearly crucial.

**Key words:** Genetic Algorithm; Scaling & Sharing; PVC; Combinatorial Optimization; Metaheuristics.

## **CHAPITRE 1**

# **THEORIE DES ALGORITHMES GENETIQUES**

## **CHAPITRE 2**

### **L'ETAT DE L'ART**

## **CHAPITRE 3**

### **CONCEPTION DE L'APPROCHE**

## **INTRODUCTION GENERALE**

## **CONCLUSION GENERALE**

## **CHAPITRE 4**

### **RESULTAT ET DISCUSSION**