

الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'enseignement Supérieur et de la Recherche scientifique



Université Mohamed Khider Biskra
Faculté des Sciences et de la Technologie
Département de Génie Electrique
Filière : Electronique

Option : signaux et télécommunications

Réf:

Mémoire de Fin d'Etudes
En vue de l'obtention du diplôme:

MASTER

Thème

Application de la méthode GRASP (Greedy Randomized Adaptive Procédure) dans problème d'optimisation

Présenté par :

Ben Belabbes Fatima

Soutenu le : 05 Juin 2013

Devant le jury composé de :

Mr torki najat

Mr TOUMI ABIDA

Mr MEDOUAKH .SAADIA.

M.C.A

M.C.B

M.A.B

President

Encadreur

Examineur

Année universitaire: 2012 / 2013

الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'enseignement Supérieur et de la recherche scientifique



Université Mohamed Khider Biskra
Faculté des Sciences et de la Technologie
Département de Génie Electrique
Filière : Electronique
Option : signaux et télécommunications

Mémoire de Fin d'Etudes
En vue de l'obtention du diplôme:

MASTER

Thème

*Application de la méthode GRASP (Greedy Randomized Adaptive
Procédure) dans problème d'optimisation*

Présenté par :
Ben Belabbes Fatima

Avis favorable de l'encadreur :
Mr. TOUMI ABIDA

Avis favorable du Président du Jury
torki najat

Cachet et signature

الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'enseignement Supérieur et de la Recherche scientifique



Université Mohamed Khider Biskra
Faculté des Sciences et de la Technologie
Département de Génie Electrique
Filière : Electronique
Option : signaux et télécommunications

Thème :

*Application de la méthode GRASP (Greedy Randomized Adaptive
Procédure) dans problème d'optimisation*

Proposé par : Ben Belabbes Fatima.

Dirigé par : TOUMI ABIDA

RESUMES (Français et Arabe)

Un voyageur de commerce doit visiter n villes données en passant par chaque ville exactement une fois.

Il commence par une ville quelconque et termine en retournant à la ville de départ. Les distances entre les villes sont connues. Il faut trouver le chemin qui minimise la distance parcourue. le but du mémoire est

d'appliquer la méthode GRASP et algorithmes génétiques pour trouver le meilleur solution de coût minimal.

مندوب مبيعات يجب عليه زيارة المدن البيانات من خلال كل مدينة مرة واحدة بالضبط. وهي تبدأ مع بعض المدن وينتهي بالعودة الى مدينة المغادرة. من المعروف أن المسافات بين المدن. تأخذ وقت لذا يجب علينا أن نجد الطريق الذي يقلل من المسافة. والهدف من هذه الرسالة هو تطبيق طريقة GRASP والخوارزميات الوراثية لإيجاد أفضل الحلول بأقل مسافة و أقل تكلفة ممكنة.

Dédicace

Après l'élaboration de notre travail, nous trouvons que dédier ce Présent et modeste travail à mes parents, pour leur aide et encouragement.

A mes collègues qui m'ont donné le nécessaire et n'ont jamais tardé.

A mes amies qui sont toujours à coté de moi dans les moments difficiles.

Remerciement

Tenons à remercier infiniment avant tout DIEU tout puissant qui nous a donné toute la volonté, la santé et la paissance pour élaborer notre travail.

Et merci à notre encadreur Mme Toumi Abida Qui nous a aidés beaucoup et que à donné des conseils pratiques durant cette année.

Je remercie tout le département d'électronique de l'enseignant qui a beaucoup contribué à notre formation. Tenons à remercier tout les membres du jury qui ont accepté de gérer notre travail.

Merci Un grand merci à mon ami Azzedine Issawi

Enfin, je remercie ma famille, petits et grands, et mes amis qui ont toujours été à mes côtés.

Liste de figures

Figure. I.1 Présente les différentes méthodes d'optimisation.....	5
Figure. I.2 Représentation du minimum local et global d'une fonction.....	9
Figure. I.3 Les catégories des métaheuristiques.....	11
Figure. I.4 Algorithme de la méthode de descente.....	12
Figure. I.5 Algorithme de la méthode du recuit simulé.....	14
Figure. I.6 Algorithme de la recherche tabou.....	17
Figure. I.7 Algorithme de la méthode GRASP.....	19
Figure. I.8 Principe général d'un algorithme génétique.....	21
Figure. I.9 Algorithme de l'évolution différentielle.....	22
Figure. I.10 Algorithme de colonies de fourmis.....	23
Figure. I.11 Mécanisme de déplacement des essaims de particules.....	25
Figure. I.12 Algorithme des essaims de particules.....	25
Figure II.1. organigramme générale de GRASP	32
Figure II.2 L'influence du paramètre.....	33
Figure II.3. Pseudocode de la phase de construction.....	34
Figure II. 4 : Pseudocode de la recherche locale.....	35
Figure II. 5 : TSP avec 4 villes.....	35
Fig. III.1 : Illustration d'un problème multimodal.....	44
Fig. IV.1 la carte de la zone villes.....	51.
Fig. IV.2 le graphique montre l'emplacement des villes en cercles bleus.....	52
Fig. VI. 3. Principe général d'un algorithme génétique.....	53
Fig. VI. 4 Organigramme d'un Algorithme génétique	54

Fig. IV.5 résultats l’algorithme génétique si nombre de villes (n=10).....	56
Fig. IV.6 résultats l’algorithme génétique si nombre de villes (n=40).....	57
Fig. IV.7 résultats l’algorithme génétique si nombre de villes (n=100).....	57
Fig. VI.8 : organigramme générale de GRASP.....	59
Fig.VI .9: Procédure «Construction».....	60
Fig.VI .10: Organigramme Phase de Construction.....	62
Fig. IV.11 : résultats phase de construction si nombre de villes (n =10).....	63
Fig. IV.11 : résultats phase de construction si nombre de villes (n =40).....	63
Fig. IV.11 : résultats phase de construction si nombre de villes (n =10).....	64
Fig.VI.14 : Procédure «Recherche locale».....	65
Fig. VI. 15 : Organigramme phase de recherche locale.....	66

Liste de tableaux

Tab III .1 : montre l’explosion combinatoire du PVC.....	47
Tab.III.2 montre le gain pour chaque étape.....	50

Abréviations

AC (Ant Colony) : algorithme à colonie de fourmis.

GA (Genetic Algorithm) : algorithme génétique.

GRASP (Greedy Randomized Adaptive Search Procedure) : procédure de recherche Adaptative gloutonne et randomisée.

LS (Local Search) : recherche locale.

SA (Simulated Annealing) : algorithme de type recuit simulé.

TS (Tabu Search) : algorithme de recherche taboue.

TSP (Traveling Salesman Problème) : Le problème du voyageur de commerce

Résumé

Un voyageur de commerce doit visiter n villes données en passant par chaque ville exactement une fois. Il commence par une ville quelconque et termine en retournant à la ville de départ. Les distances entre les villes sont connues. Il faut trouver le chemin qui minimise la distance parcourue. le but du mémoire est d'appliquer la méthode GRASP et algorithmes génétiques pour trouver le meilleur solution de coût minimal.

Mots clés : TSP, GRASP, algorithmes génétiques.

ملخص

مندوب مبيعات يجب عليه زيارة المدن البيانات من خلال كل مدينة مرة واحدة بالضبط. وهي تبدأ مع بعض المدن وينتهي بالعودة الى مدينة المغادرة. من المعروف أن المسافات بين المدن تأخذ وقت لذا يجب علينا أن نجد الطريق الذي يقلل من المسافة. والهدف من هذه الرسالة هو تطبيق طريقة GRASP والخوارزميات الوراثية لإيجاد أفضل الحلول بأقل مسافة و أقل تكلفة ممكنة.

المفاتيح: TSP , GRASP , الخوارزميات الوراثية

Sommaire

Dédicace	I
Remerciements	II
Liste des figures	IV
Liste des tableaux	VI
Introduction générale	1

Chapitre I : Généralités sur les métaheuristiques

1- Introduction	5
I. 2. Méthodes de résolution des problèmes d'optimisation mono-objectif.....	5
I. 2.1 Les méthodes exactes	6
I. 2.2 Les méthodes approchées	8
I. 2.3 Les méthodes à base de voisinage.....	8
I.3 Les métaheuristiques	9
I.3.1 La méthode de descente	12
I.3.2 La méthode du recuit simulé	13
I.3.3 Dans la recherche tabou	16
I.3.4 La méthode GRASP	19
I.3.5 L'algorithme de l'Evolution Différentielle (ED).....	21
I.3.6 L'algorithme des colonies de fourmis	22
I.3.6 L'optimisation par les essais de particules.....	24
I.4 Conclusion	26

Chapitre II : Méthode GRASP

II.1 Introduction	28
--------------------------------	-----------

II.2	Présentation générale du GRASP.....	28
II.2.1	Version de base.....	28
II.2.2	Versions hybrides.....	29
II.3	Définition et Historique.....	30
II.3.1	Définition.....	30
II.3.2	Historique.....	30
II.4.	principe de GRASP.....	31
II.4.1	Fonctionnement	31
II.4.2	Phase de Construction.....	32
II.4.3	Phase de Recherche locale.....	34
II.5	Exemple d'application (TSP)	35
II.5.1	Modélisation.....	35
II.5.2	Domaines d'Application	36
II.5.3	Avantages et Inconvénients	37
II. 6	Variation de GRASP.....	38
II.7	conclusion	38

Chapitre III : Le problème de voyageur de commerce

III.1	Présentation.....	40
III .2	Définition et Historique du TSP.....	41
III .2.1	Définition	41
III .2.2	Historique.....	42
III .3	Formulation mathématique	42
III .4	Complexité du problème.....	44
III .5	Familles d'algorithmes.....	44

III .5.1 Les algorithmes déterministes.....	45
III.5.1.1 La méthode de séparation et 'évaluation.....	45
III.5.1.2 La méthode du glouton.....	45
III.5.2 Les algorithmes d'approximation permettent de trouver une solution.....	45
III. 5.2 .1 La méthode du recuit-simule.....	45
III. 5.2 .1.2 Les algorithmes génétiques.....	46
III. 5.2 .1.3 La méthode du "2-opt amélioré".....	46
III. 6 Bilan.....	47
III. 6.1 Analyse sur un exemple.....	47
III. 6.2 Le d'efi des 250 villes.....	47
III. 7 Intérêt et applications.....	48
III. 8 Conclusion	48

Chapitre IV : Résultats et discussions

IV.1 Introduction	50
IV.2 Identification de l'ordinateur	50
IV.3 Détaille du matlab utilisé	50
IV.4 Le problème du voyageur de commerce.....	50
IV.4.1 Définition	50
VI.5 Utilisation des algorithmes génétiques.....	53
VI.5 .1 Définition Les algorithmes génétiques	53
VI.5.2 Organigramme d'un algorithme génétique.....	54
VI.5.3 résumé de l'algorithme génétique de base	54
IV.6 Utilisation des GRASP	58
IV.6 .1 Définition.....	58

IV.6 .2	principe de GRASP.....	58
IV.6 .1 .1	Fonctionnement	58
IV.6 .1.2	Phase de Construction	59
IV.6 .1.3	Organigramme Phase de Construction.....	62
IV.7	Résultats et discussions de GRASP.....	63
IV.7 .1	Phase construction	63
IV.7 .2	Discussion.....	64
IV.7 .3	.Compression avec phase construction et les algorithmes génétique	
IV.8	Phase de recherche locale.....	64
IV.8 .1	Définition.....	65
IV.8 .2	Organigramme Phase de recherche locale.....	66
IV.8 .3	Résultats et discussions Phase de recherche locale.....	67
IV.8 .5	Compression avec Phase de recherche locale et phase construction.....	68
IV.8 .6	Compression avec Phase de recherche et les algorithmes génétique.....	69
IV.8 .7	Compression avec GRASP et les algorithmes génétique.....	69
IV.9	Conclusion	69

Conclusion générale et perspectives

Références bibliographiques

Annexe A

Annexe B

Annexe C

Introduction générale

Introduction générale

Dans divers domaines économiques, le besoin croissant d'efficacité et de rentabilité a conduit les décideurs à solutionner leurs problèmes quelle que soit la difficulté en introduisant des méthodes d'optimisation. Cependant, la compétition entre les méthodes de recherche pour arriver aux meilleurs résultats en termes de qualité et de temps de traitement a conduit à distinguer un type de problèmes classés difficiles. Lorsque le problème à résoudre est NP-difficile, l'utilisation d'une méthode exacte qui garantit l'obtention de la meilleure solution nécessite un temps de calcul très long qui croît de façon exponentielle avec la taille du problème traité. Dans ce cas, l'utilisation de méthodes qui offrent une solution de bonne qualité, quasi optimale, mais en un temps raisonnable s'avère intéressante. Ces méthodes sont les métaheuristiques.

Le premier chapitre et les métaheuristiques forment un ensemble d'algorithmes permettant de trouver la solution la plus rapide et la plus efficace pour une large gamme de problèmes d'optimisation difficile et pour lesquels on ne connaît pas de méthode classique plus efficace.

Le deuxième chapitre la méthode **GRASP** : c'est un processus multi-Start ou itérative, chaque itération consiste de 2 phases principales : une phase de construction ou une solution réalisable est produite, et une phase de recherche locale où un optimum sera trouver dans le voisinage de l'élément construit.

Le troisième chapitre Un voyageur de commerce doit visiter n villes données en passant par chaque ville exactement une fois. Il commence par une ville quelconque et termine en retournant à la ville de départ. Les distances entre les villes sont connues.

Le quatrième chapitre résultats et discussion Le problème du voyageur de commerce est un problème d'optimisation où il y a un nombre fini de villes, et le coût

Introduction générale

du voyage entre chaque ville est connue. Le but est de trouver un ensemble ordonné de toutes les villes pour le vendeur à visiter tels que le coût est minimisé. Pour résoudre le problème du voyageur de commerce, et Compression avec GRASP et les algorithmes génétique.

Chapitre I

Généralités sur les métaheuristiques

I.1. Introduction

Dans la vie courante, nous sommes, souvent, confrontés à des problèmes qui peuvent être décrits sous forme d'un problème d'optimisation, comme le fait de minimiser les coûts de production pour accroître les bénéfices dans le milieu industriel. Dans ce type de problème, ceci a été rendu possible grâce, aussi, à la croissance de la puissance de calcul des ordinateurs produits ces vingt dernières années. L'optimisation combinatoire permet de minimiser ou maximiser des fonctions dans des systèmes dans lesquels peut intervenir un grand nombre de paramètres. Pour résoudre de tels problèmes, on définit un espace de recherche S fini et une fonction objectif f appelée aussi fitness ou fonction de coût qu'il s'agit d'optimiser, *ie.* Minimiser ou maximiser. Cependant, pour les problèmes dits difficiles, on ne connaît pas d'algorithmes exacts rapides permettant de les résoudre. Même pour les autres, dits à variables continues, il n'existe pas, non plus, d'algorithmes permettant de repérer un optimum global à coup sûr et en un nombre fini de calculs. En effet, en optimisation, le problème peut se présenter sous différentes appellations, selon sa complexité. Il peut être de classe P, s'il contient tous les problèmes relativement faciles c'est à dire ceux pour lesquels on connaît des algorithmes efficaces. Plus formellement, ce sont les problèmes pour lesquels on peut construire une machine déterministe (e.g. une machine de Turing²) dont le temps d'exécution est de complexité polynomiale (le sigle P signifie « Polynomial time »). S'il est facile ou résolu par l'application des méthodes classiques. Par contre, dans le cas d'existence de discontinuité, de fonction non dérivable et de présence de bruits, on parle de problèmes d'optimisation difficile. Ces derniers sont de deux types, à savoir, les problèmes combinatoires (ou problèmes discrets) comme le voyageur de commerce et les problèmes à variables continues. Dans le cas de méthodes linéaires à variables continues, la programmation linéaire est utilisée. Dans le cas où elles sont non linéaires, il s'agit d'une optimisation difficile continue. Notons, toutefois que, jusqu'à présent, ce sont, plutôt, les méthodes combinatoires ou discrètes regroupant les méthodes exactes comme la méthode du simplexe [1], de Branch and Bound [2] et la programmation dynamique [3] qui sont utilisées. Néanmoins, lorsqu'on veut résoudre un problème complexe, ces méthodes prennent un temps de calcul qui croît exponentiellement avec la taille des instances du problème. Ceci conduit souvent à une explosion combinatoire. Pour éviter ce type de problème, on fait appel aux méthodes approchées qui ne cherchent pas forcément l'optimum absolu mais donnent une solution très proche de l'optimum absolu montrant l'inexistence d'une solution sensiblement meilleure et largement acceptée. Ceci est obtenu en utilisant les méthodes regroupant les

heuristiques et les métaheuristiques. Nous rappelons que la plupart des heuristiques sont spécifiques à un problème donné et ne s'appliquent, en général, qu'à des problèmes discrets. Elles sont de complexité raisonnable, autrement dit, idéalement polynomiales mais efficaces et simples à mettre en œuvre. Quant aux métaheuristiques, ce sont des algorithmes stochastiques qui tentent de trouver l'optimum global d'un problème d'optimisation difficile en un Temps raisonnable sans être piégé dans des optima locaux.

I. 2. Méthodes de résolution des problèmes d'optimisation mono-objective:

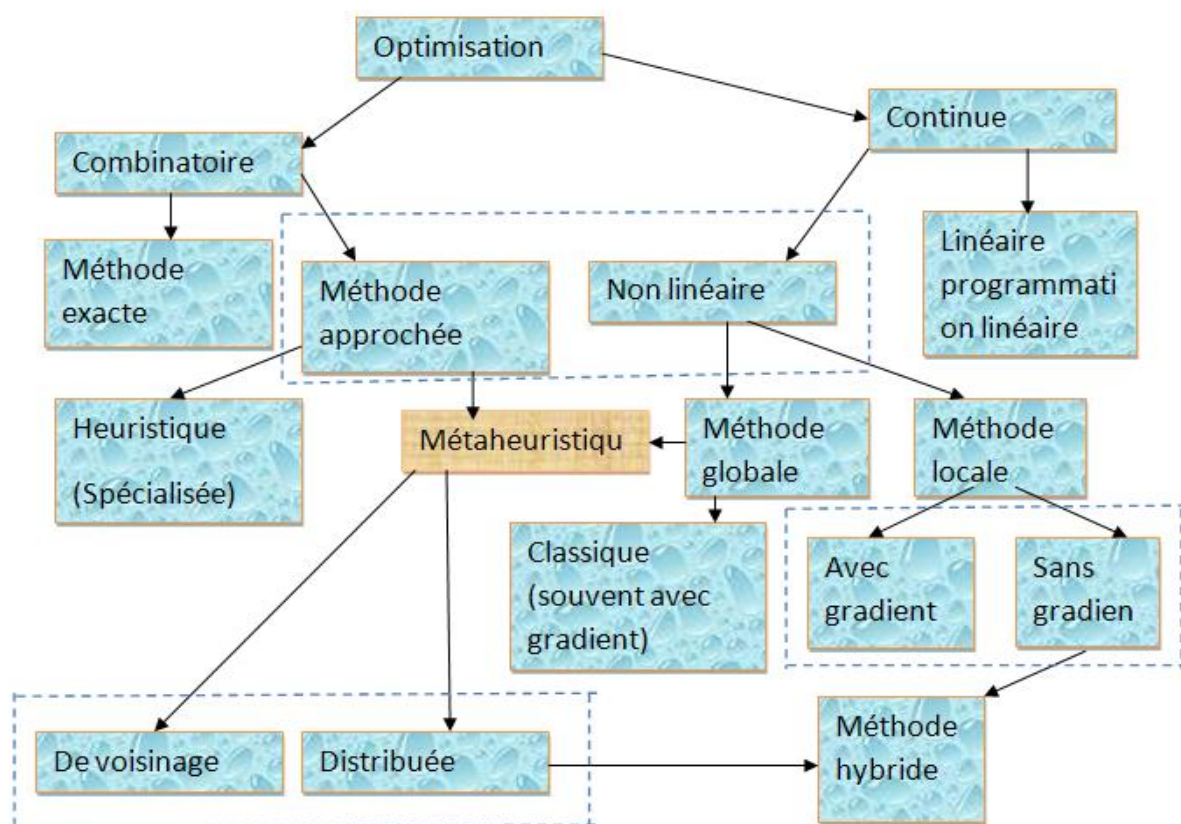


Fig. I.1 Présente les différentes méthodes d'optimisation [4]

Le schéma précédent présente une classification générale des méthodes de résolution des problèmes d'optimisation mono-objectif. On distingue en premier lieu l'optimisation continue de l'optimisation combinatoire. Pour l'optimisation continue, on sépare le cas linéaire du cas non linéaire, où l'on retrouve le cadre de l'optimisation difficile, dans ce cas on fait appel à une méthode locale qui exploite ou non les gradients de la fonction objectif. Si le nombre de minimums locaux est très élevé, le recours à une méthode globale s'impose : on retrouve alors les métaheuristiques. Pour l'optimisation combinatoire, on utilise les méthodes exactes.

Lorsqu'on est confronté à un problème difficile on a recours aux méthodes approchées, dans ce cas le choix est parfois possible entre une heuristique spécialisée, dédié au problème considéré, et une métaheuristique. Parmi les métaheuristiques, on peut différencier les métaheuristiques à base de voisinage, et les métaheuristiques à base de population. Enfin les méthodes hybrides associent souvent une métaheuristique et une méthode locale.

I.2.1 Les méthodes exactes :

Les méthodes exactes sont des méthodes qui garantissent la complétude de la résolution autrement dit ces méthodes donnent à tous les coups la solution optimale. Le temps de calcul nécessaire de telles méthodes augmente en général exponentiellement avec la taille du problème à résoudre. On distingue dans ce cas l'approche constructive qui est probablement la plus ancienne et occupe traditionnellement une place très importante en optimisation combinatoire. Une méthode constructive construit pas à pas une solution de la forme :

$s = (\langle V_1, v_1 \rangle \langle V_2, v_2 \rangle \dots \langle V_n, v_n \rangle)$ en partant d'une solution partielle initialement vide $s = 0$, elle cherche à étendre à chaque étape la solution partielle $s = (\langle V_1, v_1 \rangle \dots \langle V_{i-1}, v_{i-1} \rangle)$ ($i \leq n$) de l'étape précédente. Pour cela, elle détermine la prochaine variable V_i , choisit une valeur v_i dans D_i et ajoute $\langle V_i, v_i \rangle$ dans s pour obtenir une nouvelle solution partielle $s = (\langle V_1, v_1 \rangle \dots \langle V_n, v_n \rangle \langle V_i, v_i \rangle)$. Ce processus se répète jusqu'à ce que l'on obtienne une solution complète. Durant la recherche d'une solution, la méthode constructive fait intervenir des heuristiques pour effectuer chacun des deux choix : le choix de la variable suivante et le choix de la valeur pour la variable. Les méthodes de cette classe diffèrent entre elles selon les heuristiques utilisées. En général, les heuristiques portent plus souvent sur le choix de variables que sur le choix de valeurs car les informations disponibles concernant le premier choix semblent souvent plus riches. La performance de ces méthodes dépend largement de la pertinence des heuristiques employées, c'est à dire, de leur capacité d'exploiter les connaissances du problème.

Un premier type de méthodes constructives est représenté par les méthodes Gloutonnes. Ces méthodes consistent à fixer à chaque étape la valeur d'une variable sans remettre en cause les choix effectués précédemment.

Un deuxième type de méthodes constructives est représenté par les méthodes avec retour arrière. Ces méthodes de retour arrière avec une stratégie de recherche en profondeur d'abord consistent à fixer à chaque étape la valeur d'une variable. Aussitôt qu'un échec est détecté, un

retour arrière est effectué, c'est à dire, une ou plusieurs instanciations déjà effectuées sont annulées et de nouvelles valeurs recherchées. Les méthodes avec retour arrière sont en général complètes et de complexité exponentielle. Pour réduire le nombre de retour arrière (et le temps de recherche), on utilise des techniques de filtrage afin d'anticiper le plus tôt possible les échecs. Par exemple : ALICE, PROLOG III sont des systèmes de programmation sous contraintes fondés sur le principe de retour arrière.

Un troisième type de méthodes constructives est représenté par de nombreux algorithmes basés sur le principe de séparation et évaluation progressive, qui ont pour principe la construction d'un arbre de recherche dont le problème initial (problème de minimisation) est la racine. On divise le problème en sous problèmes (en deux ou plus) en introduisant par exemple une contrainte supplémentaire, qui peut être satisfaite ou non. L'optimum peut appartenir à l'un quelconque de ces sous problèmes. Tout sous problème infaisable sera éliminé. Si possible on calcule la solution du problème, sinon, on calcule une borne inférieure, si elle est supérieure de la meilleure solution déjà obtenue on élimine le sous-problème. Dans le cas restant, on subdivise à nouveau le sous problème. Pour améliorer l'efficacité de la recherche, on utilise des techniques variées pour calculer des bornes permettant d'élaguer le plus tôt possible des branches conduisant à un échec. Parmi ces techniques on peut citer : la relaxation de base en programmation linéaire et la relaxation lagrangienne.

Une autre méthode exacte, la méthode de programmation dynamique, c'est une méthode découverte par Bellman en 1956, elle s'est avérée une méthode très efficace pour la résolution des problèmes d'optimisation combinatoire, elle est conçue sur le modèle de l'algorithme du plus court chemin dans un graphe. Elle consiste à décomposer la résolution du problème initial en une suite de problèmes plus simples, la résolution du n ème se déduisant de celle du $(n-1)$ ème par une équation récurrence[5].

1.2.2 Les méthodes approchées

Contrairement aux méthodes exactes, les méthodes approchées ne procurent pas forcément une solution optimale, mais seulement une bonne solution (de qualité Une partie importante des méthodes approchées est désignée sous le terme de métaheuristiques. Plusieurs définitions d'une métaheuristique ont été proposées dans la littérature[6], cette définition est celle adoptée par le [7] Plusieurs classifications des métaheuristiques ont été proposées ; la plupart distinguent globalement deux catégories : les méthodes à base de solution courante unique, qui travaille sur un seul point de l'espace de recherche à un instant donné, appelées méthodes à base de voisinage comme les méthodes de recherche

locale (méthode de la descente), de recuit simulé et de recherche tabou, et les méthodes à base de population, qui travaillent sur un ensemble de points de l'espace de recherche, comme les algorithmes évolutionnaires et les algorithmes de colonies de fourmis.

I.2.3. Les méthodes à base de voisinage

Dans les problèmes d'optimisation où l'on cherche à optimiser une fonction objectif sur un espace de recherche donné, une petite perturbation sur un point de cet espace induit souvent une petite variation des valeurs de la fonction objectif en ce point. On déduit que les bonnes solutions ont tendance à se trouver à proximité d'autres bonnes solutions, les mauvaises étant proches d'autres mauvaises solutions. D'où l'idée qu'une bonne stratégie consisterait à se déplacer à travers l'espace de recherche en effectuant de petits pas (petits changements sur le point courant) dans des directions qui améliorent la fonction objectif. Cette idée est la base d'une grande famille d'algorithmes appelée méthodes à base de voisinage ou de recherche locale. Les méthodes de voisinage (ou méthodes de recherche locale) s'appuient toutes sur un même principe : elles résolvent le problème d'optimisation de manière itérative. Elles débutent avec une configuration initiale (souvent un tirage aléatoire dans l'espace des configurations), et réalisent ensuite un processus itératif qui consiste à effectuer un mouvement³ choisi par le mécanisme d'exploration⁴ en tenant compte de la fonction de coût. ce processus s'arrête et retourne la meilleure configuration trouvée quand la condition d'arrêt est réalisée. Cette condition d'arrêt peut porter sur le nombre d'essais effectués, sur une limite temporelle ou sur le degré de qualité de la meilleure Configuration courante. Cette versatilité permet de contrôler le temps de calcul, la qualité de la solution optimale trouvée s'améliorant au cours du temps. De manière générale les opérateurs de recherche locale s'arrêtent quand une solution localement optimale est trouvée, c'est à dire quand il n'existe pas de meilleure solution dans le voisinage. Les méthodes de voisinage diffèrent essentiellement entre elle par le voisinage utilisé et la stratégie de parcours de ce voisinage. La version la plus simple des méthodes de recherche locale est la méthode de descente.

I. 3 Les métaheuristiques

Apparues dans les années 1980, les métaheuristiques forment un ensemble d'algorithmes permettant de trouver la solution la plus rapide et la plus efficace pour une large gamme de problèmes d'optimisation difficile et pour lesquels on ne connaît pas de méthode classique plus efficace. Comme le montre la figure 1.2, il s'agit de trouver l'optimum global G sans être piégé par les autres optima locaux tel que le point L.

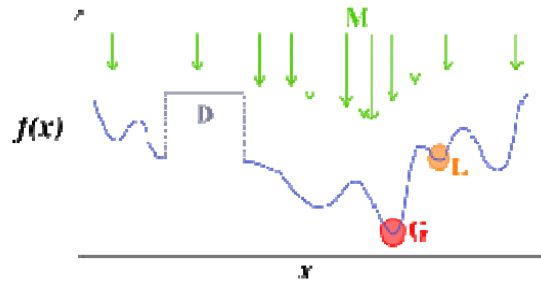


Fig. I.2 Représentation du minimum local et global d'une fonction

Les métaheuristiques fonctionnent selon un comportement itération c'est-à-dire que le même schéma se reproduit un certain nombre de Fois au cours de l'optimisation, généralement, elles s'articulent autour des trois notions suivantes :

- la diversification (exploration)
- l'intensification (exploitation)
- la mémorisation (apprentissage)

La diversification ou exploration désigne le processus qui procède pour récolter de l'information sur le problème à optimiser. La stratégie de diversification la plus simple consiste à redémarrer périodiquement le processus de recherche à partir d'une solution générée aléatoirement ou choisie judicieusement dans une région non encore visitée de l'ensemble des solutions admissibles. Pour sa part, l'intensification ou exploitation utilise l'information déjà récoltée pour explorer, en détail, les zones jugées prometteuses dans l'espace de recherche. Sa mise en œuvre réside, le plus souvent, dans l'élargissement temporaire du voisinage de la solution courante. Quant à la mémorisation, elle est le support de l'apprentissage qui permet à Sa mise en œuvre réside, le plus souvent, dans l'élargissement temporaire du voisinage de la solution courante. Quant est le support de l'apprentissage qui permet à l'algorithme de ne tenir compte que des zones où l'optimum global est susceptible de se trouver, évitant ainsi, les optima locaux qui sont de bonnes solutions, mais qui ne sont pas les meilleures des solutions possibles. Ainsi, en alternant l'intensification, la diversification et la mémorisation, le fonctionnement des métaheuristiques est progressif et itératif. L'étape initiale est souvent choisie de façon aléatoire et L'étape d'arrêt est souvent fixée à l'aide d'un critère d'arrêt. Toutes les métaheuristiques s'appuient sur l'équilibre entre l'intensification et la diversification de la recherche. Sinon, on assistera à une convergence trop rapide vers des minima locaux par manque de diversification ou à une exploration trop longue par manque d'intensification.

Le fonctionnement des métaheuristiques est généralement inspiré à partir de systèmes physiques comme le recuit simulé, de systèmes biologiques comme les algorithmes évolutionnaires, de systèmes ethnologiques comme les algorithmes de colonies de fourmis ou de l'optimisation par essaims particulaires etc. Ainsi, les métaheuristiques peuvent être réparties en deux catégories à savoir, les méthodes à base de voisinage correspondant à la recherche locale et les méthodes à base de population correspondant à la recherche globale (fig.1.3).

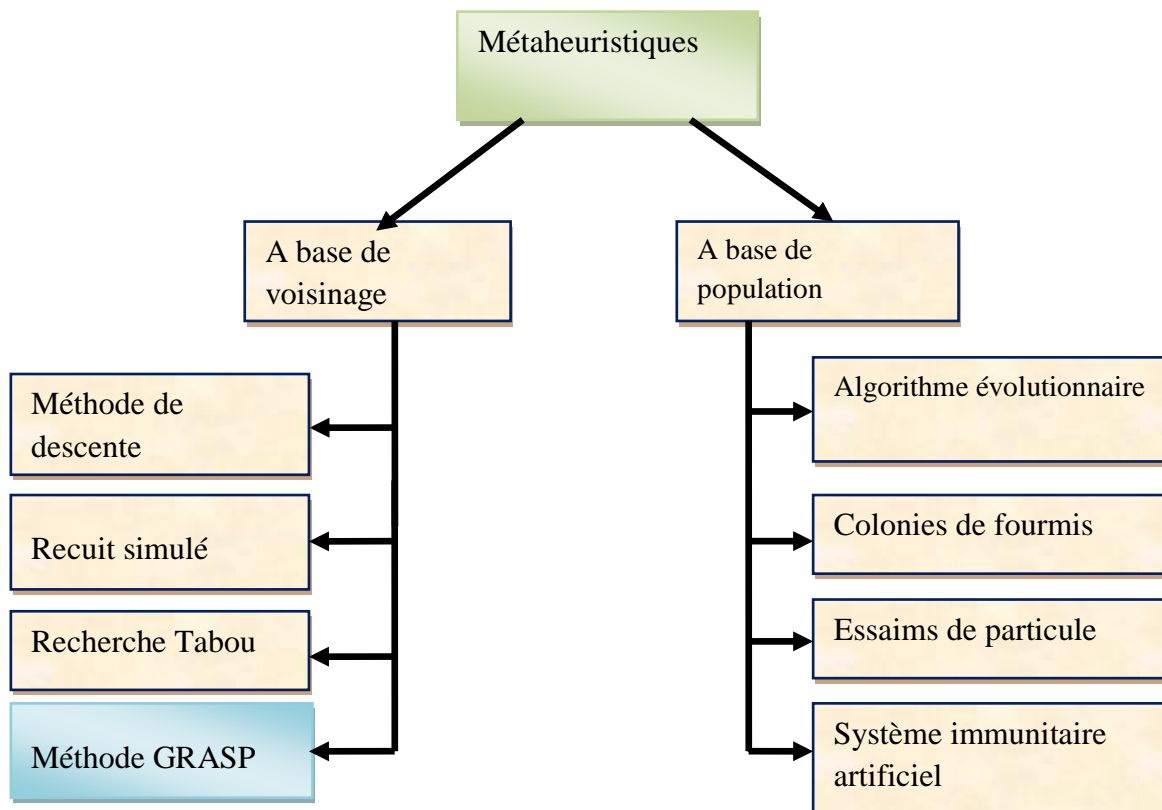


Fig. I.3 Les catégories des métaheuristiques.

Les métaheuristiques à base de voisinage ou à base de solution unique sont les méthodes de recherche locale. Elles sont itératives, ie. À partir d'une solution unique x_0 considérée comme point de départ, la recherche consiste à passer d'une solution à une solution voisine par déplacements successifs dans un voisinage constitué de l'ensemble des solutions. Souvent, les opérateurs de recherche locale s'arrêtent quand une solution localement optimale est trouvée. Mais accepter uniquement ce type de solution n'est pas toujours satisfaisant. Il est alors important de sortir de ces minima locaux en permettant à l'opérateur de recherche locale de trouver des points pour lesquels la nouvelle solution retenue sera de qualité meilleure que la

précédente. C'est le principe adopté pour la méthode de descente, le recuit simulé, la recherche tabou et la méthode GRASP.

I.3.1 La méthode de descente

Appelée aussi méthode d'amélioration itérative ou Hill Climbing en anglais est assez ancienne [8]. Sa rapidité et sa simplicité font d'elle la méthode la plus utilisée. Elle part d'une solution initiale X_i d'un ensemble de recherche S et progresse vers une solution voisine X_{i+1} de meilleure qualité quel que soit i . Son algorithme général pour un problème de minimisation est donné à la figure 1.4.

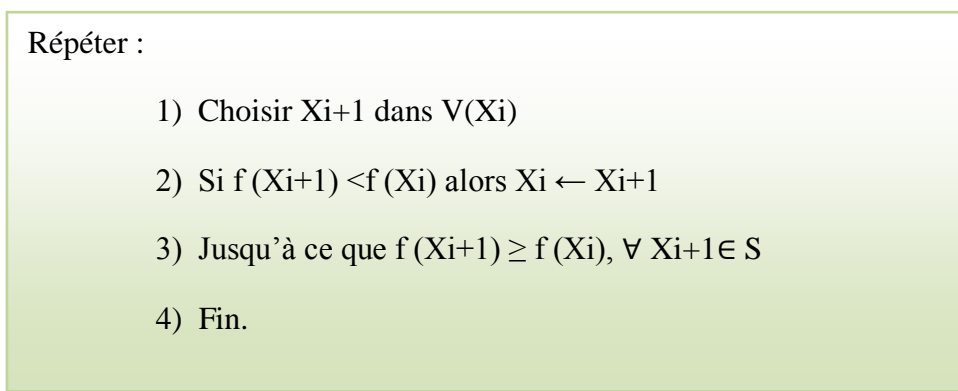


Fig. I.4 Algorithme de la méthode de descente.

Pour appliquer une descente, il faut bien choisir la solution initiale (X_i) qui est généralement aléatoire ou provenue d'une méthode approchée. Quand cette solution est aléatoire, on pourra appliquer plusieurs fois la descente en changeant à chaque fois la solution initiale et ne mémorisant que celle donnant le meilleur résultat. La méthode de descente est très facile à programmer. Son inconvénient réside dans le fait qu'elle calcule une solution locale car elle s'arrête au premier optimum rencontré. La répétition de déclaration de la solution initiale permet d'atténuer cet inconvénient.

➤ Avantages et inconvénient

En général, l'efficacité des méthodes de recherche locale simples (descente, ou plus grande descente) est très peu satisfaisante. D'abord, par définition, la recherche s'arrête au premier minimum local rencontré, c'est là leur principal défaut. Pour améliorer les résultats, on peut lancer plusieurs fois l'algorithme en partant d'un jeu de solutions initiales différentes, mais la performance de cette technique décroît rapidement. En revanche, autoriser de temps à autre une

certaine dégradation des solutions trouvées, afin de mieux explorer tout l'espace des configurations, a conduit au développement des deux méthodes que nous explorons aux paragraphes suivants, à savoir le recuit simulé et la méthode Tabou.

Le principal avantage de la recherche locale simple est évidemment sa grande simplicité de mise en œuvre: la plupart du temps, elle ne fait que calculer $f(s+i)-f(s)$, où i correspond à un déplacement élémentaire, et si cette expression peut se simplifier algébriquement, alors on pourra évaluer très rapidement cette différence. Il est important de remarquer également l'importance du choix de la fonction de voisinage N : un minimum local pour une certaine structure de voisinage ne l'est pas forcément pour une autre. C'est d'ailleurs ce constat qui est à l'origine de la méthode dite de recherche par voisinage variable, qui repose sur la construction de solutions s parmi plusieurs voisinages N_i , plutôt que dans un seul.

I.3.2 La méthode du recuit simulé (Simulated Annealing)

A été introduite par Kirkpatrick, Gelatt, et Vecchi en 1983 [9]. Cette métaheuristique s'inspire du recuit thermique des métaux en métallurgie. En effet, le refroidissement brutal d'un métal donne une structure amorphe, état métastable qui correspond à un optimum local pour un problème d'optimisation combinatoire. A l'opposé, si l'on le refroidit lentement, on obtient un état stable correspondant à un optimum global. Par analogie, on prend l'énergie du métal comme fonction objectif du problème combinatoire et les états physiques comme configurations aléatoires. La température est simulée par un paramètre de contrôle dépendant du problème traité. La méthode du recuit simulé autorise des variations de la configuration initiale ou solution initiale afin de s'échapper aux minima locaux dans lesquels le système peut être piégé lorsqu'on utilise la méthode de descente. Au lieu de refuser les solutions dégradant la fonction objectif, on les accepte avec une certaine probabilité en se basant sur le critère de Métropolis. Pour un problème de minimisation, à partir de la configuration courante i , on effectue une modification aléatoire j , puis on calcule l'énergie $\Delta E_{ij} = E_i - E_j$. Si cette énergie est augmentée, on accepte automatiquement cette modification sinon on tire aléatoirement un nombre entre 0 et 1 et on calcule la probabilité $\exp(-\Delta E_{ij}/T)$. Si cette probabilité est supérieure ou égale à ce nombre, on accepte la modification effectuée sinon elle est rejetée. Lorsque l'énergie reste stationnaire on diminue la température et on reprend le processus de calcul de l'énergie.

S'il s'agit d'un problème de maximisation, il suffit de prendre l'opposé de l'énergie ($-\Delta E_{ij} = E_i - E_j$) comme fonction objectif. L'algorithme du recuit simulé pour un problème de minimisation est donné à la figure (1.5).

- 1) Choisir une solution initiale $s \in S$
- 2) $T \leftarrow T_0$ {initialiser la température courante} ;
- 3) Répéter
- 4) Choisir aléatoirement $s' \in V(s)$
- 5) Calculer $\Delta \leftarrow f(s') - f(s)$;
- 6) Si $(\Delta < 0)$ alors
- 7) $s \leftarrow s'$;
- 8) Sinon r nombre aléatoire de $[0,1]$;
- 9) Si $(r < \exp(\Delta/T))$ alors
- 10) $s \leftarrow s'$;
- 11) $T \leftarrow 0.9 * T$ {réduire la température courante} ;
- Fin si
- Fin si
- 12) Jusqu'à conditions d'arrêt satisfaites

Fig. I.5 Algorithme de la méthode du recuit simulé.

La méthode du recuit simulé donne une solution de bonne qualité. Elle est facile à programmer et applicable à tous les problèmes ayant la caractéristique itérative tout comme elle offre une souplesse d'emploi car de nouvelles contraintes peuvent être incorporées facilement dans le programme. Néanmoins, cette méthode présente des inconvénients qui apparaissent dans le nombre important de paramètres tels la température initiale, le taux de décroissance de la température, la durée du palier de température, le critère d'arrêt, etc. Ainsi que le temps de calcul qui est excessif dans certaines applications.

Et voici l'interprétation de son fonctionnement :

✚ Si $f(s') < f(s)$ alors $e^{\frac{f(s)-f(s')}{T}} > 1$, donc r est toujours inférieur à cette valeur, et on accepte la solution s' (une meilleure solution est donc toujours acceptée, ce qui paraît logique).

✚ Si $f(s') > f(s)$ et T est très grand, alors $e^{\frac{f(s)-f(s')}{T}} < 1$, et on il y a de fortes chances d'accepter s' (bien que la solution s' soit plus « mauvaise » que s !)

✚ Si $f(s') > f(s)$ et T est très petit, alors $e^{-\frac{f(s)-f(s')}{T}} < 0$, et on va donc probablement refuser s .

Dans un premier temps, T étant généralement choisi très grand, beaucoup de solutions, même celles dégradant la valeur de f , sont acceptées, et l'algorithme équivaut à une visite aléatoire de l'espace des configurations. Mais à mesure que la température baisse, la plupart des mouvements augmentant l'énergie sont refusés, et l'algorithme se ramène à une amélioration itérative classique. A température intermédiaire, l'algorithme autorise de temps en temps des transformations qui dégradent la fonction objective. Il laisse ainsi une chance au système de s'extraire d'un minimum local. Cet algorithme est parfois amélioré en ajoutant une variable qui mémorise la meilleure valeur rencontrée jusqu'à présent (sans cela, l'algorithme pourrait converger vers une certaine solution s , alors qu'on avait visité auparavant une solution s' de valeur inférieure à $f(s)$!).

➤ Avantages et inconvénients

Le recuit simulé présente l'avantage d'offrir des solutions de bonne qualité, tout en restant simple à programmer et à paramétrer. Il offre autant de souplesse d'emploi que l'algorithme de recherche local classique : on peut inclure facilement des contraintes dans le corps du programme. Aarts, Korst et Laarhoven (1997) ont par ailleurs démontré que, sous certaines conditions de décroissance de la température, l'algorithme du recuit simulé converge en probabilité vers un optimum global lorsque le nombre d'itérations tend vers l'infini. L'un des inconvénients du recuit simulé est qu'une fois l'algorithme piégé à basse température dans un minimum local, il lui est impossible de s'en sortir tout seul. Plusieurs solutions ont été proposées pour tenter de résoudre ce problème, par exemple en acceptant une brusque remontée de la température, de temps en temps, pour relancer la recherche sur d'autres régions plus éloignées. Il est également possible d'empêcher la température de descendre trop bas : on lui donne une valeur minimale au-delà de laquelle on ne change plus de palier de température. Mais si cette valeur est trop grande, l'algorithme passera son temps à augmenter et diminuer son énergie car il acceptera trop de perturbations dégradantes et il n'arrivera pas à explorer à fond une vallée.

Ainsi, il est fort possible que l'algorithme arrive à "trouver" la vallée dans laquelle se cache un minimum global, mais il aura beaucoup de mal à l'explorer et donc risque de s'en éloigner sans avoir décelé la solution au problème. C'est là le principal problème du recuit : le paramétrage de la température peut être rebutant et très empirique. On constate également qu'il faut une diminution de la température « suffisamment lente », donc un certain nombre

d'itérations, pour voir la solution s'améliorer. Corrélativement, le temps de calcul devient parfois excessif avec certaines applications, et l'algorithme exige en particulier le calcul d'une exponentielle.

➤ Applications

L'algorithme du recuit simulé a montré son efficacité sur les problèmes combinatoires classiques, surtout sur les échantillons de grande taille. Par exemple, l'expérience a montré qu'il ne devenait vraiment efficace sur le problème du voyageur de commerce qu'au-delà d'environ 800 villes. Il a été testé avec succès sur le problème du partitionnement de graphe. Des analyses ont montré qu'il était efficace avec certaines catégories de problème où l'ensemble des solutions possèdent certaines propriétés particulières. Ceci expliquerait le succès du recuit simulé dans le domaine du placement des circuits électroniques, où il est très employé.

I.3.3 Dans la recherche tabou développée par Glover [10]

Le principe est de passer à la meilleure solution voisine de la solution initiale à chaque itération même si celle-ci est plus mauvaise que la solution initiale. Ce critère autorise la dégradation de la fonction objectif et évite ainsi le blocage dans un minimum local. Le déplacement s'effectue de solution en solution en s'interdisant de revenir à une solution déjà rencontrée. Au cours de la recherche, ces solutions visitées sont mises dans une liste interdite appelée liste taboue. Cette liste de longueur finie k mémorise les k dernières solutions rencontrées ou les mouvements donnant ces solutions. Durant un nombre d'itération égal à k , il est interdit d'effectuer une transformation inverse à une transformation récemment effectuée c'est-à-dire ajouter une solution récemment supprimée ou supprimer une solution récemment ajoutée. Interdictions sont parfois inutiles. Par exemple, un mouvement qui mène à une solution meilleure que toutes celles visitées n'a aucune raison d'être interdit. Dans ce cas, le critère d'aspiration est introduit ; il s'agit de lever le statut tabou et d'exécuter cette solution. L'algorithme général de la recherche tabou pour un problème de minimisation est illustré sur la figure suivante (fig.1.6).

La recherche tabou présente l'avantage de posséder une mémoire sous la forme de la liste tabou. Cette mémoire peut être à court terme si on.

```

1) Poser  $T \leftarrow \emptyset$  {initialiser une liste taboue vide}
   2) Générer une solution initiale  $s$ 
   3) Mettre  $s^* \leftarrow s$  { $s^*$  est la meilleure solution}
   4) Répéter
   5) Générer  $N(s)$ 
   6) Choisir  $s'$  qui minimise  $f(s')$  dans  $N(s)$ 
   7) Si  $f(s') < f(s^*)$  alors
   8) Poser  $s^* \leftarrow s'$  {condition d'aspiration : si la solution courante améliore
la meilleure solution ; accepter la même si elle est taboue}
   9) Poser  $s \leftarrow s'$  {mettre à jour la solution courante}
  10)  $T \leftarrow T + s'$  {mettre à jour la liste tabou}
  11) sinon si  $s' \in NT(s)$  alors
  12) Poser  $s \leftarrow s'$  {mettre à jour la solution courante si elle n'est pas taboue}
  13) Si  $f(s') < f(s^*)$  alors
  14) Poser  $s^* \leftarrow s'$ 
  15)  $T \leftarrow T + s'$ 
Fin.

```

Fig. I.6 Algorithme de la recherche tabou.

Mémorise la solution telle qu'elle est et à long terme si on mémorise la fréquence d'apparition au lieu d'interdire complètement le mouvement introduisant cette solution. Ainsi, les paramètres à fixer sont moins nombreux. Il s'agit de la taille de la liste tabou qui peut être statique déterminée à l'avance et constante durant tout le processus de résolution ou dynamique variante durant la résolution et du nombre total d'itérations. Cependant, dans plusieurs cas d'optimisation, la recherche tabou demande de lui ajouter les mécanismes d'intensification et de diversification qui introduisent de nouveaux paramètres à régler.

Avantages et inconvénients

La méthode Tabou est une méthode de recherche locale, et la structure de son algorithme de base est finalement assez proche de celle du recuit simulé, donc on passera donc de l'un à l'autre facilement, avec l'avantage, par rapport au recuit simulé, d'avoir un paramétrage simplifié : dans un premier temps, le paramétrage consistera d'abord à trouver une valeur indicative t d'itérations pendant lesquelles les mouvements sont interdits. Il faudra également décider d'une stratégie de mémorisation à long terme – sur la qualité des solutions, sur leur récurrence, ou sur leur qualité...., En revanche, la méthode Tabou exige une gestion de la mémoire de plus en plus lourde à mesure que l'on voudra raffiner le procédé en mettant en place des stratégies de mémorisation complexe. L'efficacité de la méthode Tabou fait qu'elle est largement employée dans les problèmes d'optimisation combinatoire : elle a été testée avec succès sur les grands problèmes classiques (voyageur de commerce, ordonnancement d'ateliers) et elle est fréquemment appliquée sur les problèmes de constitution de planning, de routage, d'exploration géologique, etc.

I.3.4 La méthode GRASP (Greedy Randomised Adaptive Search Procedure)

Qui signifie procédure de recherche adaptative aléatoire gloutonne est une métaheuristique introduite par FEO et RESENDE en 1995 [11]. Cette méthode répète un processus composé de deux étapes à chaque itération : la construction d'une solution par une méthode gloutonne et l'amélioration de cette solution par une méthode de recherche locale. Durant l'étape de construction, on choisit toutes les solutions réalisables, on prend les meilleures selon la fonction objectif puis on tire au hasard pour former la liste des candidats restreinte (LCR). Cette liste est mise à jour après chaque itération de construction ce qui donne l'aspect adaptatif de l'algorithme. On sélectionne au hasard une solution dans LCR qui sera améliorée en lui appliquant une méthode de recherche locale. On répète itérativement ces étapes jusqu'à l'obtention de la meilleure solution. L'algorithme de GRASP pour un problème de minimisation est le suivant (fig.1.7).

```

T = dimension du problème ;
Pour k=1 jusqu'à itération_max faire
Construction :
Solution=0 ;
Evaluer le cout incrémental des éléments candidats ;
Tant que Dimension (solution) ≠ T : Construire LCR : la liste de candidats
restreinte ;
Sélectionner au hasard une solution s de LCR : solution = Ajouter (solution, s)
Fin
Recherche locale :
Tant que solution non optimale localement :
Trouver s' ∈ Voisinage (solution) tel que  $f(s') < f(solution)$ 
    Solution = s'
Fin
meilleure_solution = min (solution, meilleure_solution) ;
Fin
Retourner meilleure_solution

```

Fig. I.7 Algorithme de la méthode GRASP.

L'aspect le plus intéressant de GRASP est sa facilité d'implémentation ainsi que le nombre restreint de paramètres à régler ; il suffit de fixer la longueur de la liste LCR et le nombre maximum d'itérations. Cependant, cette méthode est moins performante par rapport aux autres métaheuristiques du fait de l'absence de mémoire ce qui implique l'obtention de la même solution en plusieurs reprises.

Contrairement aux méthodes précédentes, **les méthodes de recherche à base de population** se basent sur une population d'individus qui interagissent entre eux afin de s'améliorer ou de produire de nouveaux individus plus performants. Elles travaillent sur un ensemble de points de l'espace de recherche. Ces métaheuristiques sont inspirées de la biologie et utilisant des phénomènes d'auto organisation. Parmi ces algorithmes à population on trouve, les algorithmes évolutionnaires comme les Algorithmes Génétiques, les colonies de fourmis, les essaims de particules etc.

➤ **Avantages et inconvénients**

La méthode GRASP est relativement simple à programmer, et permet d'améliorer les résultats d'une simple recherche locale. Elle permet également de faire intervenir une heuristique spécialisée. Elle ne nécessite que peu de paramétrage : la taille de la liste, qui permet d'équilibrer la quantité d'adaptabilité (l'heuristique) et le facteur stochastique. Elle nécessite de plus peu de calculs supplémentaires par rapport à une recherche locale simple. Elle est en revanche moins performante que d'autres métaheuristiques, essentiellement du fait de son absence de mémoire : comme avec une recherche locale simple, rien ne garantit que l'algorithme ne va pas, à plusieurs reprises, explorer des zones très similaires et retomber sur les mêmes minima locaux.

❖ **Les algorithmes génétiques**

S'inspirent des principes de la génétique. Pour cette raison, ces algorithmes accordent une grande importance à la distinction entre la représentation génétique d'un individu (génotype) et sa représentation réelle (phénotype). L'opérateur principal utilisé par les algorithmes génétiques pour la construction de nouvelles solutions est l'opérateur de recombinaison appelé aussi croisement. Cet opérateur récupère des parties du génotype de deux ou plusieurs solutions ou parents qu'il combine pour construire un ou plusieurs nouveaux génotypes ou enfants, héritant ainsi de certaines de leurs caractéristiques. L'utilisation de la recombinaison, seule, ne permet pas d'introduire du matériel génétique nouveau puisque cet opérateur combine le matériel déjà présent dans la population. Pour remédier à ce problème, les algorithmes génétiques utilisent la mutation, comme opérateur secondaire permettant d'introduire de nouveaux gènes inexistants dans la population. Le principe général d'un algorithme génétique est présenté dans la figure 1.8 (fig. 1.8).

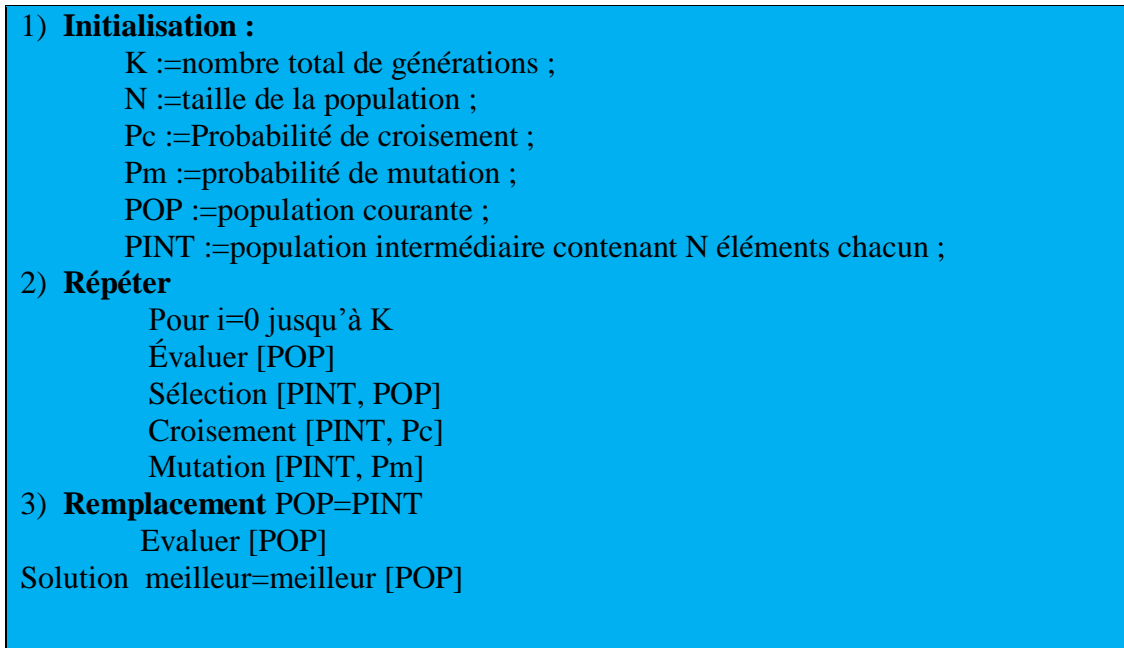


Fig. I.8 Principe général d'un algorithme génétique.

I.3.5 L'algorithme de l'Evolution Différentielle (ED) (Differential Evolution) a été développé par Price et Storn en 1995 [12] pour résoudre les problèmes d'optimisation combinatoires. Cet algorithme utilise les mêmes opérateurs évolutionnaires que l'algorithme génétique (sélection, croisement et mutation) mais le fonctionnement est différent (fig.1.9). En effet, cette méthode crée de nouveaux individus. Une nouvelle solution est créée à partir des différences entre deux individus de la population. Si on considère un problème à D dimensions avec une population de N individus évoluant à chaque génération t, l'algorithme de la méthode ED pour un problème de minimisation est donné à la figure 1.8. Dans cet algorithme F est un nombre réel appartenant à l'intervalle [0,2] appelé facteur d'amplification. De nombreuses améliorations ont été proposées pour le choix des paramètres. Cette métaheuristique a été utilisée avec succès dans le cas des problèmes réels notamment pour résoudre des problèmes continus.

```

1) Générer une population initiale
2) Pour  $t = 0, \dots, T$ 
3) Pour  $i = 0, \dots, N$ 
4) Générer aléatoirement
    $r1, r2, r3$  avec  $r1 \neq r2 \neq r3$ 
   Choisir aléatoirement trois individus
    $xr1, t, xr2, t, xr3, t$ 
   Mutation : Créer un mutant  $vi, t$  à partir de
   l'individu  $xi, t$  et de  $xr1, t, xr2, t, xr3, t$ 
    $vi, t + 1 = xr1, t + F * (xr2, t - xr3, t)$ 
   Croisement : Créer l'individu test  $ui, t$  à
   partir du mutant  $vi, t$  et de  $xi, t$ 
   Sélection:
   Si  $f(ui, t) < f(xi, t)$  alors  $xi, t \leftarrow ui, t$ 
   Sinon  $xi, t + 1 \leftarrow xi, t$ 
5) Fin pour
6) Fin pour

```

Fig. I.9 Algorithme de l'évolution différentielle.

I.3.6 L'algorithme des colonies de fourmis (Ant Colony Optimisation)

Est un algorithme d'intelligence en essaim dont le principe est basé sur la manière dont les fourmis cherchent leurs nourritures et retrouvent leur chemin pour retourner dans la fourmilière (fig.1.10). Initialement, les fourmis explorent les environs de leur nid de manière aléatoire [10] Sitôt qu'une source de nourriture est repérée par une fourmi, son intérêt est évalué (quantité et qualité) et la fourmi ramène un peu de nourriture au nid. Les fourmis peuvent déposer des phéromones au sol, grâce à une glande située dans leur abdomen et former, ainsi, des pistes odorantes qui pourront être suivies par leurs congénères. Les traces laissées s'accumulent au fur et à mesure que la piste est rejointe par plus de congénères. Les phéromones ont comme caractéristique l'évaporation en fonction du temps. Les pistes les plus longues seront donc abandonnées au profit de la plus courte. Ainsi, pour illustrer le principe de cette méthode pose A un ensemble de K fourmis. Une fourmi de cet ensemble va démarrer de la ville i à l'instant t vers la ville j , la destination est choisie en fonction de la visibilité η_{ij} (l'inverse de la distance) et de la quantité de phéromones $\tau_{ij}(t)$ déposée entre les deux villes.

Répéter :
 Pour $i=1$ à k faire
 Construire le Trajet (i)
 Fin Pour
 Mettre à Jour Phéromones O
 Jusqu'à ce que le critère de terminaison soit satisfait.
 Dans la procédure Construire le Trajet (i), chaque fourmi construit un trajet selon la probabilité

$p^{kij}(t)$ telle que :

$$p^{kij}(t) = \left[(\tau_{ij}(t)) \right]^{\alpha^*} (\eta_{ij})^\beta \left[\sum_{s \in jik} (\tau_{ij}(t))^{\alpha^*} (\eta_s)^\beta \right] \text{ si } s \in jik$$

sinon $p^{kij}(t) = 0$

Fig. I.10 Algorithme de colonies de fourmis.

α et β sont deux paramètres contrôlant respectivement l'influence du taux de phéromone sur le trajet (ij) et l'influence de la distance sur le trajet (ij). La procédure mettre à jour Phéromone O consiste à mettre à jour le taux de phéromones (ij) pour la fourmi k en appliquant La formule suivante :

$$\tau_{ij}(t+1) = (1 - \rho) * \tau_{ij}(t) + \Delta\tau_{ij}(t) + \Delta\tau_{ij}(t) \quad (1.1)$$

$$\text{Avec } \Delta\tau_{ij}(t) = \sum_{x=1}^k (\Delta\tau_{ij}^x(t)) \text{ et } \Delta\tau_{ij}^k(t) = \frac{Q}{Lk(T)}$$

Q est une constante et Lk(t) est la longueur totale de la tournée de la fourmi k à l'instant t. Pour éviter d'être piégé dans des optima locaux, les algorithmes de colonies de fourmis utilisent la stratégie d'évaporation des pistes de phéromones dont la solution est mauvaise [14]. Les algorithmes de colonies de fourmis sont très efficaces en optimisation combinatoire grâce à leur robustesse i.e. La recherche reste efficace même si certains de ses individus sont défaillants, et à sa décentralisation, i.e. les fourmis n'obéissent pas à une autorité centralisée, comme ils ont l'avantage d'être exécutés en parallèle. Néanmoins, cette méthode présente l'inconvénient de réglage d'un nombre important de paramètres (nombre de fourmis, visibilité, quantité de phéromones etc.).

I.3.7 L'optimisation par les essais de particules (Particle Swarm Optimisation ou PSO)

Est issue d'une analogie avec les comportements collectifs de déplacements chez certains groupes d'animaux, commères bancs de poissons, les nuées des oiseaux, les araignées etc. [15]. En effet, tout comme ces animaux se déplacent en groupe pour trouver de la nourriture ou éviter les prédateurs, les algorithmes à essaim de particules recherchent des solutions pour un problème d'optimisation. L'essaim particulaire est constitué d'une population de particules. Chaque particule i correspond à une solution du problème traité. Elle est caractérisée par une position \vec{x}_i et une vitesse (vélocité) de changement de position (\vec{v}_i et garde en mémoire sa meilleure position atteinte qui peut être parfois la position courante. À chaque itération t , la position et la vitesse sont mises à jour en tenant compte de la meilleure position \vec{p}_i de la particule et de la meilleure position \vec{p}_g de ses voisins suivant la formule citée ci-dessous :

$$\begin{cases} \vec{v}_i(t) = \omega * \vec{v}_i(t-1) + c1 * \rho1 * (\vec{p}_i - \vec{x}_i(t-1)) + c2 * \rho2 * (\vec{P}_g - \vec{x}_i(t-1)) \\ \vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t) \end{cases} \quad (1.2)$$

ω est une constant appelée coefficient d'inertie. Elle joue un rôle important dans la convergence. Une faible valeur de ω nous guide à une recherche locale tandis qu'une grande valeur nous permet une exploration globale de l'espace de recherche [16]. $c1$ représente l'attraction de la particule vers sa meilleure position ; c' est la mémoire propre à la particule, tandis que $c2$ représente l'attraction vers la meilleure de ses voisines engendrant la mémoire partagée. $\rho1$ et $\rho2$ sont des variables aléatoires tirés de l'intervalle $[0,1]$ à chaque itération. Eberhart et Shi [17] ont suggéré d'utiliser $c1=c2=2$.

La figure (1.11) montre le mécanisme de déplacement des essaims de particules.

Maurice et Clerc [15,16] ont introduit un nouveau paramètre appelé coefficient de Constriction 'k' qui est calculé comme suit :

$$K = \frac{2}{1.2 - \phi - \sqrt{\phi * \phi - 4 * \phi}} \quad \text{avec } \phi = C_1 + C_2 \text{ et } \phi > 4$$

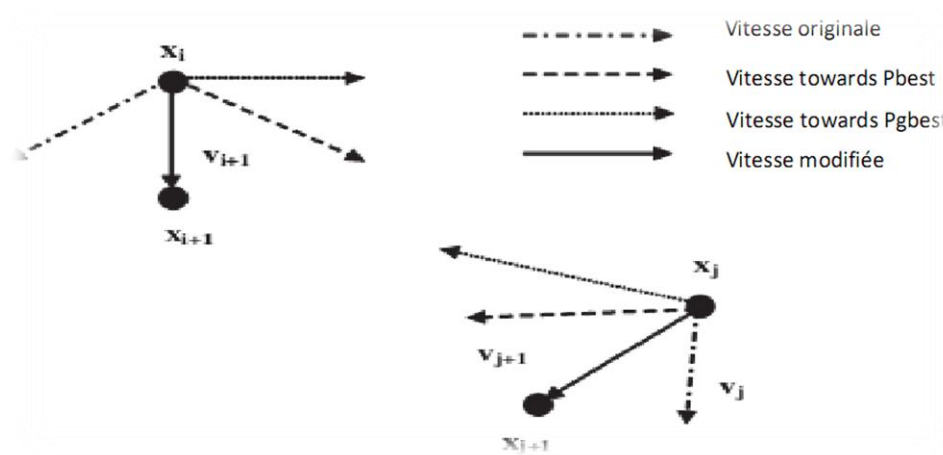


Fig. I.11 Mécanisme de déplacement des essais de particules.

La formule de PSO devient

$$\begin{cases} \vec{v}_i(t) = \omega * \vec{v}_i(t-1) + c1 * \rho1 * (\vec{p}_i - \vec{x}_i(t-1)) + c2 * \rho2 * (\vec{P}g - \vec{x}_i(t-1)) \\ \vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t) \end{cases} \quad (1.3)$$

Ce coefficient a le même rôle que le coefficient d'inertie et quand il est Utilisé, Φ est fixé à 4.1 et $c1=c2=2.05$. Les étapes principales de l'algorithme OEP sont présentées sur la figure (1.12).

- 1) $N :=$ nombre d'individus ;
- 2) $\vec{x}_i :=$ position de la particule P_i ;
- 3) $\vec{v}_i :=$ vitesse de la particule P_i ;
- 4) $P_{best_i} :=$ meilleure fitness obtenue pour la particule P_i ;
- 5) $\vec{x}_{p_{best_i}} :=$ position de la particule P_i pour meilleure fitness ;
- 6) $\vec{x}_{g_{best}} :=$ position de la particule ayant la meilleure fitness de tout le voisinage ;
- 7) Initialiser aléatoirement la population ;
- 8) Répéter
- 9) Pour i de 1 à N faire
- 10) Si $(f(\vec{x}_i) < P_{best_i})$ alors $f(\vec{x}_i) = P_{best_i}$ et $\vec{x}_{p_{best_i}} = \vec{x}_i$
- 11) Fin Si
- 12) Fin Pour
- 13) Pour i de 1 à N Faire

$$\begin{cases} \vec{v}_i(t) = \omega * \vec{v}_i(t-1) + c1 * \rho1 * (\vec{p}_i - \vec{x}_i(t-1)) + c2 * \rho2 * (\vec{P}_g - \vec{x}_i(t-1)) \\ \vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t) \end{cases}$$

- 14) Fin Pour
- 15) Jusqu' à ce que le d'arrêt soit atteint.

Fig. I.12 Algorithme des essais de particules.

La méthode des essais particulaires est très puissante en optimisation combinatoire. Elle a rencontré un vif succès depuis sa création et ça grâce à sa simplicité et son efficacité. Notons qu'il existe d'autres métaheuristiques qui sont moins populaires et peu efficaces. Parmi ces méthodes, on cite quelques variantes du recuit simulé, les algorithmes à estimation de distribution, les systèmes immunitaires artificiels, les algorithmes inspirés des insectes sociaux, la méthode du bruitage, etc. Une étude plus détaillée se trouve dans l'ouvrage de Johann Dréo et al. [4].

I.4 Conclusion

Dans ce chapitre, nous avons présenté quelques méthodes d'optimisation combinatoires en s'appuyant sur les caractéristiques principales des métaheuristiques. Ces dernières sont très efficaces en optimisation difficile sans avoir besoin de modifier la structure de base de l'algorithme utilisé. Elles sont devenues très populaires grâce à leur simplicité d'emploi dans différents domaines. Il est à noter qu'une bonne performance nécessite souvent une formalisation adéquate du problème posé et une adaptation intelligente d'une métaheuristiques. Malgré le succès remarquable de leur démarche, les métaheuristiques présentent des difficultés à lesquelles est confronté l'utilisateur dans le cas d'un problème concret comme le choix d'une méthode efficace pour avoir une solution optimale et le réglage des paramètres qui peut être réalisable en théorie mais irréalisable en pratique. Les chercheurs visent à surpasser ces difficultés en proposant des techniques d'amélioration dont on cite l'hybridation des métaheuristiques. Cette hybridation exploite la puissance de plusieurs algorithmes, et les combine en un seul méta-algorithme. Le chapitre suivant sera consacré à l'étude des métaheuristiques GRASP.

Chapitre II

Méthode GRASP

II.1 Introduction

Les outils et le matériel informatique sont en constante évolution, chaque année des nouvelles méthodes et technologies apparaissent, afin de faciliter les tâches difficiles à traiter par un être humain ; ces tâches font aussi l'objet d'une évolution assez rapide que celle des outils de résolution, et cela en matière de complexité et des nouvelles exigences de l'être humain, ce qui nous amène à développer de nouvelles techniques qui prennent en compte les exigences en termes de complexité et de fiabilité.

GRASP appartient à la famille des méthodes metaheuristiques, il permet d'obtenir une solution à un problème d'optimisation combinatoire, et cela par l'application d'opérations de construction et de la recherche locale pour plusieurs itérations.

Dans cet chapitre on va définir la méthode GRASP et expliquer son fonctionnement général, ensuite on va illustrer par un exemple de résolution par GRASP, puis on cite quelques aspects importants de la méthode.

II.2 Présentation générale du GRASP

II.2.1 Version de base

Le GRASP (Greedy Randomized Adaptive Procédure) crée une nouvelle solution à chaque itération. Il s'apparente à un échantillonnage de l'espace des solutions. Il peut fournir de bons résultats pour divers problèmes combinatoires. Des techniques plus élaborées peuvent être introduites pour améliorer les performances de la version de base comme l'indique l'état de l'art effectué. Les solutions produites sont indépendantes et la meilleure est restituée à la fin. Une itération est composée de deux phases :

- ❖ la construction d'une solution par une heuristique gloutonne randomisée;
- ❖ l'amélioration de celle-ci par une recherche locale.

En et, une heuristique déterministe donnerait toujours le même résultat à chaque itération. Il est donc nécessaire d'introduire des aléas au cours de la recherche. Le but n'est pas de fournir des solutions aléatoires à la recherche locale, car un tel principe engendrerait en général des coûts assez élevés, mais de calculer des solutions à la fois variées et de bonne qualité. La randomisation est basée sur une liste restreinte (Restricted Candidate List - RCL) de choix possibles à chaque étape de la construction de la solution.

L'ensemble de ces éléments donne son nom à la méthode. En eet, un algorithme glouton (greedy search) est utilisé avec l'introduction d'aléas (Randomized) au cours de la production des solutions

différentes à chaque itération. Ces aléas sont représentés par une liste de choix possibles (RCL) qui change à chaque étape de la construction d'une solution (adaptive). Deux politiques sont envisageables. La première est de remplir la liste des décisions possibles, sélectionnées au hasard, puis de choisir la meilleure selon le critère glouton. La seconde est d'y placer les meilleures décisions trouvées par l'algorithme glouton et d'en choisir une au hasard. Ici, la deuxième approche est employée.

La conception d'un GRASP repose sur des principes simples. Le vrai point délicat est de randomiser l'algorithme glouton en préservant la qualité des solutions. En particulier, il faut déterminer la taille de la RCL. Cette dernière peut être étagée par une cardinalité α ou selon la qualité de ses éléments. Dans le premier cas, le remplissage de la liste se fait simplement en y introduisant α élément-décisions. Dans le second, une décision possible est insérée seulement si le coût qui lui est associé est inférieur à un certain seuil déterminé comme suit. Soit e un élément et $c(e)$ son coût associé. Si c_{min} et c_{max} sont respectivement le plus petit et le plus

Grand coût possible ajouté, alors e est inséré si et seulement si $c(e) \leq c_{min} + \beta(c_{max} - c_{min})$ avec $\beta \in [0;1]$. Quand $\beta = 1$, c'est une construction aléatoire qui opère, et quand $\beta = 0$, c'est l'algorithme glouton pur qui est utilisé. Il est difficile de déterminer la valeur de β et elle est souvent choisie aléatoirement.

II 2.2 Versions hybrides

Des versions hybrides du GRASP peuvent également être implémentées, comme par exemple la combinaison avec l'utilisation d'un pathre linking (PR). Le PR est une méta- heuristique qui a été proposée pour la première fois par Glover pour intensifier un TS ou un SS par l'exploration de trajectoires reliant des solution-élites [18]. La trajectoire d'une solution U vers une solution-guide V se parcourt en introduisant dans U des attributs présents dans V. Plusieurs alternatives sont possibles pour un couple (U;V), par exemple :

- ❖ PR unidirectionnel : réalisé uniquement de la solution U à la solution V ;
- ❖ PR bidirectionnel : réalisé de U vers V puis de V vers U;
- ❖ PR randomisé : le choix de l'attribut introduit dans U se fait aléatoirement parmi ceux Permettant de se rapprocher de la solution guide V ;
- ❖ PR tronqué : la transformation de U en V est seulement partielle et la trajectoire n'est donc qu'en partie explorée.

Un PR est rarement utilisé seul. Il sert généralement de post-optimisation à une autre métaheuristique. Dans le cadre du GRASP, il fut introduit par [19], et les utilisations sont soit en post-optimisation, soit en intensification pendant les itérations du GRASP. En intensification [19], un groupe de solutions-élites est constitué au cours des itérations en y introduisant des solutions sous les conditions suivantes : elles doivent à la fois être de bonne qualité, et savamment différentes les unes par rapport aux autres. Une mesure de distance est donc nécessaire. Le PR est alors réalisé à chaque itération entre la solution courante U et une solution-élite V choisie aléatoirement dans le groupe. Cependant, il est possible de biaiser ce choix en faveur des plus distantes de U , ce qui donne habituellement de meilleurs résultats [20]. En tant que post-optimisation, un groupe de solution-élites est également constitué.

Le principe est d'appliquer le PR en n de GRASP entre les solution-élites uniquement. Ce groupe est actualisé en y remplaçant les moins bonnes solutions par les nouvelles obtenues par le PR si elles répondent au critère de qualité requis.

II.3 Définition et Historique

II.3.1 Définition

GRASP (Greedy Randomized Adaptive Search Procédure) est un algorithme métaheuristic et multi-start utilisé pour la résolution des problèmes d'optimisation combinatoire, chaque itération de cet algorithme se passe sur deux étapes, la première – étape de construction – essaie de construire une solution selon l'algorithme semi-glouton (semi-Greedy) et la deuxième – étape de recherche locale – essaie d'améliorer la solution construite dans la première étape, par une recherche locale. La solution finale sera la meilleure solution obtenue.

II.3.2 Historique

GRASP était proposé initialement par Thomas A. FEO (Université du Texas) et Mauricio G.C. RESENDE (Université de Californie) en 1989 dans un article du journal – Operations Research Letters. Un article visant à résoudre le problème de couverture (set covering problème) de manière efficace, en utilisant l'approche semi-Greedy de Chvatàl, et la recherche locale.

Après son introduction, GRASP était appliqué dans un nombre important de travaux scientifiques, notamment dans la recherche opérationnelle, et l'informatique théorique.

De nombreuses améliorations du GRASP de base, et hybridations avec d'autres metaheuristiques ont vu le jour depuis 1989. [21].

II.4. principe de GRASP

II.4.1 Fonctionnement

GRASP est un algorithme metaheuristiques basant dans son fonctionnement sur deux techniques d'optimisation largement connues, il s'agit de l'algorithme glouton (Greedy) et la recherche locale.

Pour un nombre donné d'itérations, on effectue la construction d'une solution suivant une heuristique semi-glouton (Randomized Greedy), puis on applique une recherche locale sur le voisinage de cette solution afin d'obtenir un optimum local, la solution finale sera la meilleur Solution obtenue parmi outes les solutions de chaque itération. On remarque l'indépendance des itérations dans GRASP. [21], [22].

La figure suivante montre l'organigramme général du déroulement de GRASP.

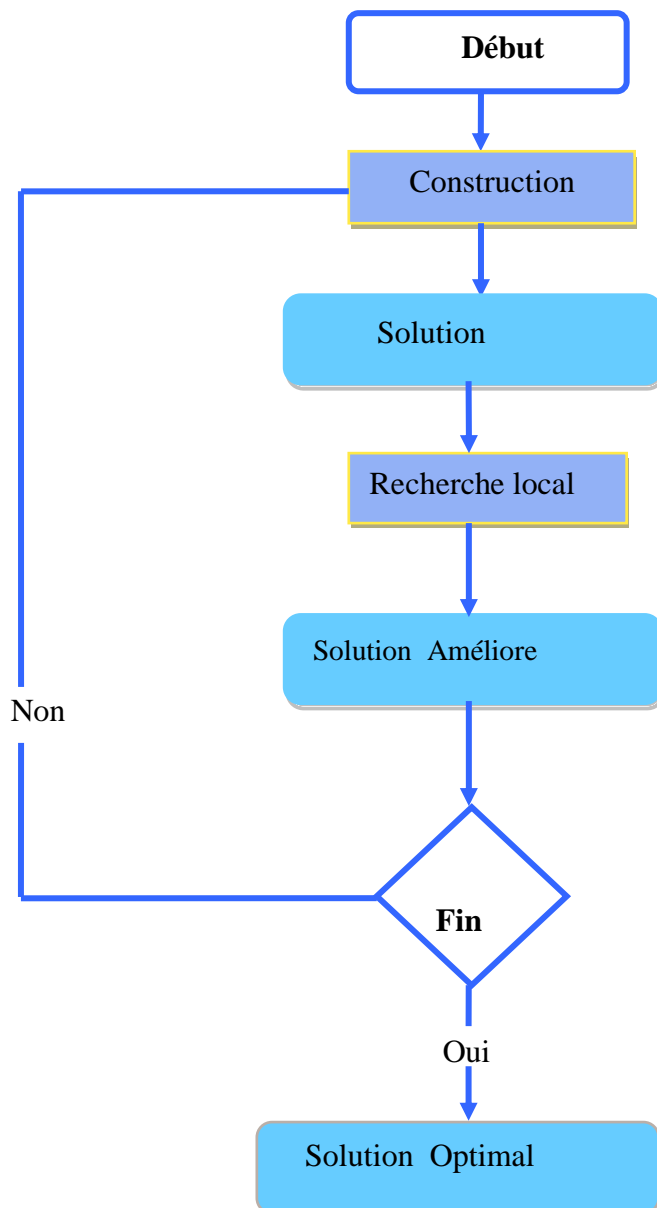


Figure II.1 organigramme générale de GRASP

II.4.2 Phase de Construction

On considère le problème de minimisation suivant : [23]

- $E = \{1 \dots n\}$: ensemble de base
- $F \subseteq 2^E$: ensemble de toutes les solutions possible
- $f: F \rightarrow \mathbb{R}$: la fonction objective (à minimiser)
- S^* : solution optimale, telle que : $f(S^*) \leq f(S) \quad \forall S \in F$

Cette étape est considérée la plus importante dans tout le processus GRASP une solution possible $s \in F$ est construite à partir des éléments $e \in E$;élément par élément suivant l'heuristique semi-glouton, qui est un algorithme glouton augmenté d'un aspect aléatoire. Avant de construire la solution, une liste de tous les composants ($e \in E$) possibles, doit être déterminé, c'est la liste des candidats (C) (candidate liste), le cout incrémental $c(e)$ pour chaque élément de C doit être évalué, ceci va nous permettre de sélectionner les meilleurs éléments et les mettre dans une autre liste appelée : liste limitée descandidats (Restricted Candidate List – RCL). [21], [24]

La construction de la RCL est gouvernée par un paramètre $\alpha \in [0,1]$ qui contrôle la qualité et le nombre d'éléments à inclure. Pour un problème de minimisation, RCL est construite comme suit :

$$\text{RCL} \leftarrow \{e \in C \mid c(e) \leq c_{Min} + \alpha(c_{Max} - c_{Min})\} \quad (2.1)$$

Avec :

c_{Min}, c_{Max} : Le cout incrémental minimal, resp maximal depuis la liste C .

$c(e)$: le cout incrémental de l'élément e .

Pour $\alpha = 0$, la RCL contiendra toujours le meilleur élément de C , pour tous les itérations, et cela rendra la recherche purement gloutonne (greedy), au Contraire, pour un $\alpha = 1$ la recherche sera purement aléatoire. C'est la Construction de la liste RCL qui donne l'aspect glouton (greedy) au GRASP.



Figure II.2. L'influence du paramètre α

Après avoir construit la RCL, on prend – aléatoirement - un élément de celle-ci et on l'incorpore à la solution en cours de construction – c'est l'aspect aléatoire (random) du GRASP. Ensuite, on met à jour l'ensemble C , les couts incrémental des éléments de C , et la RCL, ceci est l'aspect adaptatif (adaptive) de l'algorithme. La construction continue de cette

manière jusqu'à ce qu'on obtient une solution complète. Le pseudocode pour la construction est donné dans la figure suivante :

```

Procédure Greedy Randomized Construction ( $\alpha$ )
Solution  $\leftarrow \emptyset$ 
C  $\leftarrow E$ 
Evaluation du cout incrémental  $c(e) \ e \in C$ 
  Tant-Que  $C \neq \emptyset$  faire
     $C_{Min} \leftarrow \text{Min}\{c(e), e \in C\}$ 
     $C_{Max} \leftarrow \text{Max}\{c(e), e \in C\}$ 
     $RCL \leftarrow \{e \in C \mid c(e) \leq C_{Min} + \alpha (C_{Max} - C_{Min})\}$ 
    Sélection aléatoire de  $s$  depuis RCL
    Solution  $\leftarrow$  Solution  $\cup \{s\}$ 
    Mise à jour de C
    Réévaluation du cout incrémental pour  $e \in C$ 
  Fin Tant-Que
Retourne Solution

```

Figure II.3 Pseudocode de la phase de construction.

La solution construite par cette approche n'est pas nécessairement optimale, même dans un contexte local, et c'est pour ça qu'on est obligé d'effectuer une recherche locale.

II.4.3 Phase de Recherche locale

L'objectif de cette étape est l'amélioration de la solution obtenue dans l'étape antérieure afin d'obtenir un optimum local. L'amélioration se fait par l'application d'un algorithme de recherche local sur la solution. La qualité de la solution résultante dépend de la structure du voisinage, la technique de recherche, et la solution de départ elle-même. [25]

La recherche locale est un algorithme itératif qui va remplacer la solution actuelle par une meilleure solution dans le voisinage, l'algorithme s'arrête lorsqu'on ne trouve aucune meilleure solution (optimum local), deux stratégies de recherche sont envisagées : [22] , [26]

- ✚ Best-improving : on examine tous les voisins, et on remplace la solution par le meilleur voisin.
- ✚ First-improving : on remplace la solution initiale par le premier meilleur voisin qu'on trouve.

Dans la pratique ces deux approches aboutissent à la même solution, cependant, la recherche avec l'approche first-improving n'est pas coûteuse en temps de calcul.

La figure suivante illustre le pseudocode de la recherche locale.

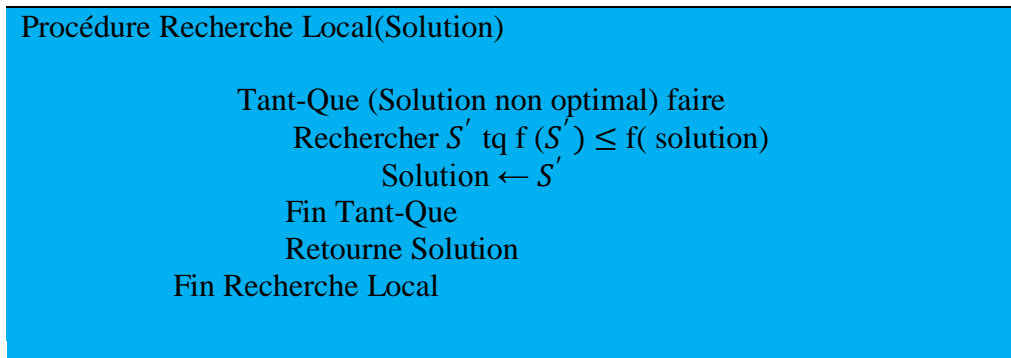


Figure II. 4 : Pseudocode de la recherche locale

II.5 Exemple d'application (TSP)

Soit le TSP (Traveling Salesman Problem) avec 4 villes suivant :

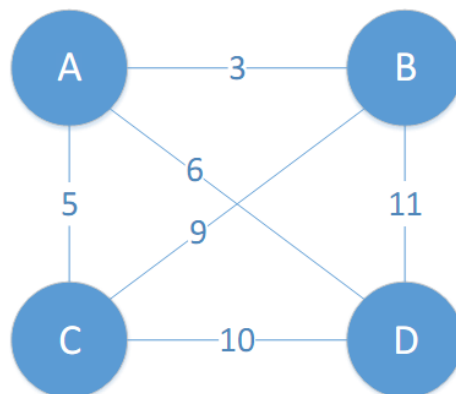


Figure II. 5 : TSP avec 4 villes

II.5.1 Modélisation

Soit E l'ensemble de tous les arcs, pour chaque arc $e \in E$, on associe un coût $C(e)$, $F \subseteq 2^E$ est l'ensemble de tous les chemins qui forme un cycle Hamiltonien (l'ensemble des solutions), pour une solution S la fonction objectif est la f est la somme des coûts des arcs composant la solution. On cherche s^* telle que $f(s^*) \leq f(S) \forall S \in F$, le cycle Hamiltonien de coût minimale -> Problème de minimisation.

Résolution par GRASP

Itération 1 : ($\alpha = 0.5$)

- **Construction**
 - $C \leftarrow \{AC, AB, AD, BC, BD, CD\}$ (TSP Symétrique)
 - Coûts (C) $\leftarrow \{5, 3, 6, 9, 11, 10\}$
- **Itération 1**
 - $C_{min} \leftarrow 3$; $C_{max} \leftarrow 11$
 - RCL $\leftarrow \{AC, AB, AD\}$
 - $S \leftarrow \{AC\}$ (sélection aléatoire)
 - $C \leftarrow \{BC, CL\}$
 - Coûts (C) $\leftarrow \{14, 15\}$
- **Itération 2**
 - $C_{min} \leftarrow 14$; $C_{max} \leftarrow 15$
 - RCL $\leftarrow \{BC\}$
 - $S \leftarrow \{AC, BC\}$
 - $C \leftarrow \{BD\}$
 - Coûts (C) $\leftarrow \{25\}$
 - etc...
 - Après 4 itérations on trouve la solution $S = (AC, AB, BD, AD)$ avec

$$f(S) = 31$$

Recherche Local

- Pour une structure de voisinage de type 2-opt on aura l'optimum Local suivant : $S' = (AC, CD, BD, AD)$ avec $f(S') = 29$

(Dans un voisinage 2-opt on coupe 2 arcs du graphe et on essaie de joindre le graphe pour optimiser le coût total.) [25]. Ce processus est répété pour un nombre MAX_ITER d'itérations, à la fin on prend la solution optimale parmi les MAX_ITER solutions obtenues.

II.5.2 Domaines d'Application

GRASP est destiné essentiellement pour l'utilisation dans la résolution des problèmes de la recherche opérationnelle, notamment le problème de couverture, qui a été le sujet de la

première publication définissant l'algorithme GRASP. Ceci n'a pas empêché les chercheurs à utiliser GRASP pour la résolution d'un éventail de problèmes, notamment : [26]

- Le routage
- SAT (Logique)
- L'optimisation des graphes.
- Ordonnancement, Production.
- Télécommunications (conception de réseau)
- Conception VLSI
- ...etc.

D'autres applications de GRASP dans des domaines différents ont vu le jour, on cite principalement l'utilisation de GRASP dans le problème de la prédiction de la structure de la protéine.

II.5.3 Avantages et Inconvénients

GRASP est l'une des metaheuristique qui possède des avantages prometteurs pour les problèmes d'optimisation combinatoires ; parmi lesquels on note : [24],

- **Nombre de paramètres à régler** : il n'y'a que deux paramètres manipuler et le nombre des itérations.
- **Temps de calcul** : par rapport à d'autres metaheuristique, l'algorithme consomme moins de temps CPU, et il est prédictible.
- **Parallélisassions** : peut-être trivialement implémenté en parallèle, donc un gain considérable en temps de calcul.
- **Implémentation** : relativement plus simple à implémenter.
- Implémentation simple et facile.
- Facile d'ajuster et modifier les paramètres.
- Possibilité d'introduire nouveaux concepts d'améliorations et d'ajustements.

Néanmoins, comme toute metaheuristique, GRASP est loin d'être une méthode parfaite : [24]

- ❖ **Incertitude sur la qualité de la solution** : la solution finale dépend toujours sur les paramètres de l'algorithme.
- ❖ **Construction sans mémoire** : Informations non sauvegardées d'une itération à l'autre.

II. 6 Variation de GRASP

Dans le but d'améliorer les différents aspects de l'algorithme, le GRASP basique faisait l'objet de plusieurs améliorations et modifications tentant d'augmenter la qualité du résultat final, et la performance de l'algorithme, parmi ces améliorations on cite : [26].

- * **Réactive GRASP** : vise à éliminer la dépendance sur le paramètre α ; où à chaque itération α est choisi automatiquement parmi un ensemble défini de valeurs possibles.

- * **Cost Perturbation** : Introduit la notion de bruit sur les coûts des composants de solutions afin d'accroître la variance des solutions construites lors de la première phase.

- * **Bias Function** : utilise une différente distribution pour la sélection aléatoire de l'élément du RCL, qui dirige la sélection vers un candidat particul

- * **Autres** :

- ❖ GRASP avec Construction intelligente.
- ❖ Proximate Optimality Principle
- ❖ Path-Relinking. ...etc. ier

II.7 conclusion

On a présenté le « GreedyRandomized Adaptive SearchProcedure » et on a vu que les résultats de GRASP sont compétitifs avec les autres méthodes heuristiques. Le GRASP est adaptatif avec des nouveaux problèmes et permis d'introduire des nouveaux concepts, qui le donne une caractéristique très satisfaisante et le rend performant et plus efficaces.

Dans cet chapitre on a donné une vue globale sur la méthode GRASP, ainsi que ses principaux aspects, il faut noter que GRASP reste une méthode relativement peu exploitée dans le domaine de l'intelligence artificielle, bien qu'elle donne un résultat comparable, voire meilleur que les méthodes classiques.

Chapitre III

*Problème Le voyageur de
commerce (TSP)*

III.1 Présentation

Le Problème du Voyageur de Commerce (PVC) est un problème classique En optimisation qui a intéressé les mathématiciens dès le 19^{ème} siècle.

Enonce Un voyageur de commerce doit visiter n villes données en passant par chaque ville exactement une fois. Il commence par une ville quelconque et termine en retournant à la ville de départ. Les distances entre les villes sont connues. Quel chemin faut-il choisir afin de minimiser la distance par courue ? La notion de distance peut-être remplacée par d'autres notions comme le temps qu'il met ou l'argent qu'il dépense : dans tous les cas, on parle de Coût.

Sa complexité n'a pas facilité son étude et si l'on analyse tous les par cours Possibles, pour n villes, le nombre de possibilités est de $(n-1)!$ Pour 6 villes nous avons 120 possibilités, pour 10 villes plus de 362 000 et pour 60 villes plus de 10^{80} soit le nombre d'atomes estimé dans l'univers.

Ceci peut expliquer pourquoi le problème n'a pas été étudié sérieusement Avant l'arrivée des ordinateurs dans les universités, mais, depuis 1954 de Nombreux chercheurs l'ont traité.

Pour ma part, n'ayant que la durée de mon stage pour découvrir le problème, mon objectif n'a pas été de trouver moi même des solutions efficaces pour le résoudre mais plutôt de faire un tour d'horizon des algorithmes existants, et d'en implémenter certains avec Matlab.

III .2 Définition et Historique du TSP

III .2.1 Définition

Un voyageur de commerce doit visiter n villes données en passant par chaque ville exactement une fois. Il commence par une ville quelconque et termine en retournant à la ville de départ. Les distances entre les villes sont connues. Il faut trouver le chemin qui minimise la distance parcourue. La notion de distance peut-être remplacée par d'autres notions comme le temps qu'il met ou l'argent qu'il dépense. Formellement, à partir d'une matrice $C = (c_{ij})$ où c_{ij} 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 $\sum_{i=1}^{n-1} c_{\sigma(i+1)\sigma(i)} + c_{\sigma(n)\sigma(1)}$ (le coût d'une tournée est égale à la somme des coûts de tous les arcs appartenant à la tournée). Autrement dit, il faut trouver un circuit (respectivement cycle) Hamiltonien de longueur minimale dans un graphe orienté (respectivement non orienté) valué complet. Si $c_{ij} = c_{ji}$ pour tout $i \in \{1, \dots, n\}$ le problème est dit symétrique sinon il est asymétrique. Le TSP peut être formulé comme un problème de programmation linéaire en nombre entier. Il a des applications directes dans le transport, la logistique, etc. Par exemple trouver le chemin le plus court pour les bus de ramassage scolaire ou, pour le camion de ramassage des ordures. Le TSP présente plusieurs caractéristiques communes aux problèmes d'optimisation combinatoire et fournit ainsi une plate-forme idéale pour étudier les méthodes générales applicables à un grand nombre de ces problèmes.

-
1. Une heuristique est une méthode, conçue pour un problème d'optimisation donné, qui produit une solution non nécessairement optimale lorsqu'on lui fournit une instance de ce problème.

III .2.2 Historique

- **19^{ième} Siècle** Les premières approches mathématiques exposées pour le PVC ont été traitées au 19^{ième} Siècle par les mathématiciens Sir William Rowan Hamilton et Thomas Penyngton Kirkman. Hamilton en a fait un jeu : Hamilton Icosian game. Les joueurs devaient réaliser une tournée passant par 20 points en utilisant uniquement les connections Préséniles.
- **Années 1930** Le PVC est traité plus en profondeur par Karl Menger ` Harvard. Il est ensuite d'enveloppe à Princeton par les mathématiciens Hassler Whitney et Merrill Flood.
- **1954** Solution du PVC pour 49 villes par Dantzig, Fulkerson et Johnson par la méthode du cutting – plane [27].
- **1975** Solution pour 100 villes par Camerini, Fratta et Maffioli.
- **1987** Solution pour 532, puis 2392 villes par Padberg et Rinaldi.
- **1998** Solution pour les 13 509 villes des Etats-Unis.
- **2001** Solution pour les 15 112 villes d'Allemagne par Applegate, Bixby Chvtal et Cook des universités de Rice et Princeton.

III .3 Formulation mathématique

Nous allons commencer par aborder le problème le plus simple, dans lequel une seule tournée doit être réalisée, le TSP. Il peut être vu comme un cas particulier du problème délocalisation-routage dont la somme des demandes n'excède pas la capacité d'un véhicule et où un seul dépôt est disponible. Le but est de trouver un cycle de longueur totale minimale qui passe exactement une fois par chaque point. En considérant un ensemble de nœuds V et d'arcs A de coût c_{ij} , le problème peut se formuler ainsi, avec les variables binaires x_{ij} qui valent 1 si l'arc $(i; j) \in A$ est utilisé (Nemhauser et Wolsey, 1999) :

$$\min \sum_{i \in V} \sum_{j > i} c_{ij} x_{ij} \quad (1)$$

Sous les contraintes :

$$\sum_{i:(i,j) \in A} x_{ij} = 1 \quad \forall j \in v \quad (2)$$

$$\sum_{j:(i,j) \in A} x_{ij} = 1 \quad \forall i \in v \quad (3)$$

$$\sum_{(i,j) \in A: i \in U, j \in V \setminus U} x_{ij} \geq 1 \quad \forall U \subset \text{ Avec } 2 \leq |U| < |V| - 2 \quad (4).$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in A \quad (5)$$

(1) est la fonction-objectif. Les contraintes (2) et (3) correspondent au degré des variables, assurant que pour chaque nœud, il y a un arc entrant et un sortant utilisé. Les contraintes (4) éliminent les sous-tours. En, les contraintes (5) assurent l'intégrité des variables. Bien que ce soit c été dédiées au TSP, comme par exemple Dantzig et al. (1954); Gomory (1958, 1963) ou Chvátal (1973). Pour un problème asymétrique ($c_{ij} = c_{ji}$), une borne inférieure peut être trouvée en résolvant un problème d'arctation, et elle est d'ailleurs optimale si aucun sous cycle n'apparaît dans la solution. Pour un problème symétrique, une borne inférieure peut être.

III.4 Complexité du problème

Ce problème est un représentant de la classe des problèmes NP-complets. L'existence d'un algorithme de complexité polynomiale reste inconnue. En supposant que le temps pour évaluer un trajet est de $1\mu s$, le tableau montre l'explosion combinatoire du PVC.

Nombre de villes	Nombre de possibilités	Temps de calcul
5	12	$12\mu s$
10	181 440	$0.18ms$
15	43 milliards	12 heures
20	60.10^{15}	1 928 ans
25	310.10^{21}	9,8 milliards d'années

Tab III .1 : montre l'explosion combinatoire du PVC

De plus, le PVC est un problème *multimodal*¹, c'est à dire qu'un minimum local n'est pas forcément un minimum global. Cette propriété justifie l'intérêt d'algorithme qui accepte des transformations n'améliorant pas forcément la longueur du trajet immédiatement comme l'algorithme de recuit- *simulé*².

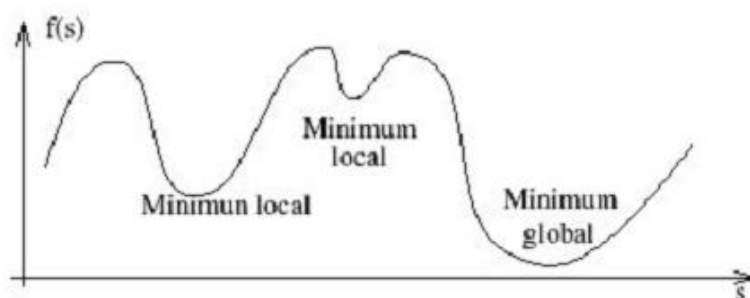


Fig. III.1 Illustration d'un problème multimodal [28].

III.5 Familles d'algorithmes

Il existe deux familles d'algorithmes permettant de résoudre le PVC :

- Les algorithmes déterministes qui trouvent la solution optimale

- Les algorithmes d'approximation qui fournissent une solution proche de la solution optimale.

III .5.1 Les algorithmes déterministes

Permettent de trouver la solution optimale, mais leur complexité est de l'ordre du factoriel. Ces algorithmes complexes ont un code de l'ordre de 10 000 lignes. De plus, ils sont très gourmands en puissance de calcul. Par exemple, il a fallu 27 heures à un puissant super-calculateur pour trouver la solution optimale pour 2392 villes.

III.5.1.1 La méthode de séparation et ´évaluation

Cette méthode est une méthode exacte mais contrairement a une méthode de recherche exhaustive, elle n énumère pas l'ensemble des solutions car elle évite des ensembles de solutions trop mauvaises. Cette méthode a permis de trouver la solution exacte à un problème à 13 509 villes. J'ai essayé de programmer cette méthode mais j ai manqué de temps pour y parvenir car elle c'est avérée difficile à implémenter.

III.5.1.2 La méthode du glouton

La méthode du glouton est probablement la plus ancienne, elle consiste `se déplacer systématiquement vers la ville la plus proche qui n'a pas encore été visitée. Elle permet rarement d'obtenir la solution optimale mais peut permettre de trouver un premier parcours de longueur acceptable. Le trajet dépendant du point initial, on peut améliorer la technique en effectuant tous les essais `a partir des différents points et choisir le meilleur. J'ai réussi à implémenter cet algorithme sur Matlab, ce qui a été mon point de départ.

III.5.2 Les algorithmes d'approximation permettent de trouver une solution dont la longueur est proche de celle de la solution optimale. Ils ont l'avantage de permettre, en un temps raisonnable, de trouver une solution. De ce fait, ils ne sont à utiliser que dans les cas où une solution approchée est acceptable.

III. 5.2 .1 La méthode du recuit-simulé

Le recuit est une technique utilisée en métallurgie qui consiste à faire fondre plusieurs fois un métal puis à le laisser lentement refroidir pour en améliorer les qualités mécaniques.

La fonction objective que l'on cherche à minimiser dans tout algorithme de résolution du PVC est la longueur du trajet. L'avantage de cette méthode est qu'elle accepte (avec une certaine probabilité) une augmentation de cette fonction objective. Cependant, un paramètre de contrôle

(classiquement la température) rend de moins en moins probable une transformation désavantageuse.

III. 5.2 .1.2 Les algorithmes génétiques

Les algorithmes génétiques sont des algorithmes d'optimisation s'appuyant sur des techniques dérivées de la génétique et des mécanismes de sélection de la nature : croisements, mutations, sélections, etc. Ils appartiennent à la classe des algorithmes évolutionnaires. Lorsque l'on utilise un algorithme génétique, contrairement aux méthodes précédentes, on ne travaille pas avec un seul trajet mais avec une population de trajets que l'on fait évoluer en croisant les meilleurs entre-deux et en les faisant muter de façon aléatoire.

III. 5.2 .1.3 La méthode du "2-opt amélioré"

C'est la solution que l'on choisit de programmer pour améliorer le parcours obtenu grâce à l'algorithme glouton, l'algorithme glouton elle est largement inspirée d'un algorithme vu sur internet utilisant la transformation du 2-opt [29]. Les explications de cette méthode seront illustrées, en annexe, avec un exemple à 30 villes.

- **l'algorithme glouton**⁴ était en premier lieu afin d'obtenir un premier trajet.
- **Décroissement** ensuite algorithme permettant d'insérer un point dans un segment plutôt que dans un autre dans le but de réduire la longueur totale du trajet a été utilisé.
- **Flip** Afin de ne pas rester bloqué dans un minimum local⁵, une fonction qui choisissait aléatoirement deux villes et qui inversait leurs positions a été utilisée. Après cette opération, il ne faut pas oublier de répéter les étapes de Décroissement et d'Insertion.

III. 6 Bilan

III. 6.1 Analyse sur un exemple

Dans cette partie, le même exemple que dans la section 3.3.5 (page 47) est traité avec la méthode du "2-opt amélioré" afin d'observer le gain pour chaque étape.

Etape de la méthode	Temps de calcul	Longueur	Pourcentage gagné
Algorithme glouton	<1s	4.8692	\
Algorithme de décroisement	<1s	4.4033	9%
Algorithme d'insertion	<1s	4.2971	2%
Record obtenu	\	4.2489	1%
Algorithme glouton	<1s	4.8692	\
Algorithme glouton amélioré	2s	4.3332	11%

Tab III .2: montre le gain pour chaque étape

On remarque que, pour un exemple de seulement 30 villes, les temps de calcul sont très courts, cependant on constate que ces durées augmentent de façon importante avec le nombre de villes. De plus, grâce à la colonne faisant apparaître le pourcentage de longueur gagnée entre deux étapes, on peut voir que plus on a amélioré le parcours, plus il est difficile de continuer à l'améliorer.

III. 7 Intérêt et applications

Le PVC fournit un exemple d'étude d'un problème NP-complet dont les méthodes de résolution peuvent s'appliquer à d'autres problèmes de mathématiques discrètes. De plus, voici une liste non-exhaustive d'applications du PVC :

- bus de ramassage scolaire
- conception de circuit électronique
- optimisation de parcours en robotique
- passage de train sur une voie [31].
- atterrissage d'avions [31].
- processus de fabrication en industrie chimique [31].
- production de cartes génétiques [32].

III. 8 Conclusion

On peut constater que le problème de voyageur de commerce est un plan standard dont il est sujet à d'application de plusieurs techniques d'optimisation stochastiques ou déterministe. C'est pourquoi on a choisi de l'utiliser dans notre application.

Chapitre IV

Résultats et discussion

IV.1 Introduction

TSP c'est problème métaheuristiques la méthode métaheuristiques utilise à base de voisinage ou à base de solution unique sont les méthodes de recherche locale. Et a base algorithme génétique. Notre objective c'est des résoudre le problème utilisaient c'est méthode GRASP et algorithme génétique

IV.2 Identification de l'ordinateur

Système d'exploration : Microsoft Windows XP proceessionnel (5, 1, Générer 2600),
Processeur : Intel (R) Pentium (R) 4CPU3.00GHz (2CPUs), Mémoire : 1014MB RAM.

IV.3 Détaille du matlab utilisé

Matlab 7.0.0.19920 (R14) May 06, 2004, License number : 247489

IV.4 Le problème du voyageur de commerce

IV.4.1 Définition

Le problème du voyageur de commerce est un problème d'optimisation où il ya un nombre fini de villes, et le coût du voyage entre chaque ville est connue. Le but est de trouver un ensemble ordonné de toutes les villes pour le vendeur à visiter tels que le coût est minimisé. Pour résoudre le problème du voyageur de commerce, nous avons besoin d'une liste des emplacements de la ville et des distances, ou des coûts, entre chacun d'eux. Notre vendeur est en visite villes des États-Unis. Vois-ci dessous la carte de la zone villes.

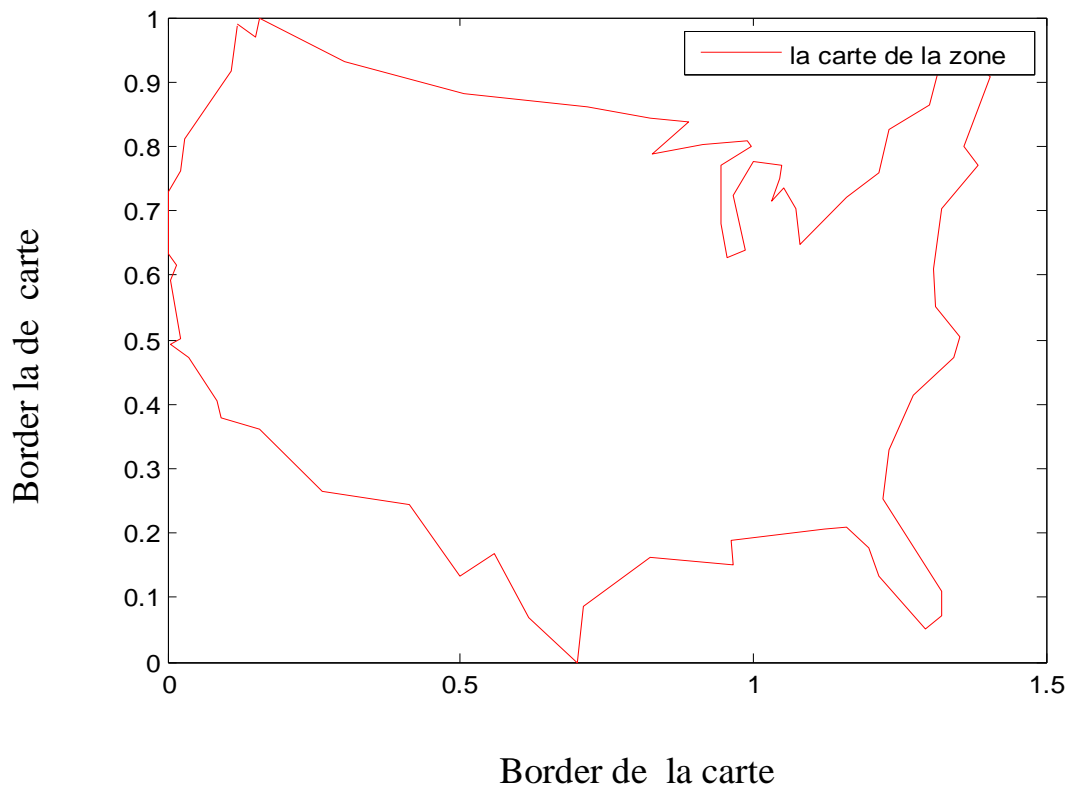


Fig. IV.1 la carte de la zone villes.

Nous allons générer des endroits aléatoires de villes à l'intérieur de la frontière de la zone pour s'assurer que toutes les villes sont à l'intérieur ou très près de la frontière de la zone. Les différentes trajectoires que le voyageur de commerce peut suivre pour atteindre sadistimation.

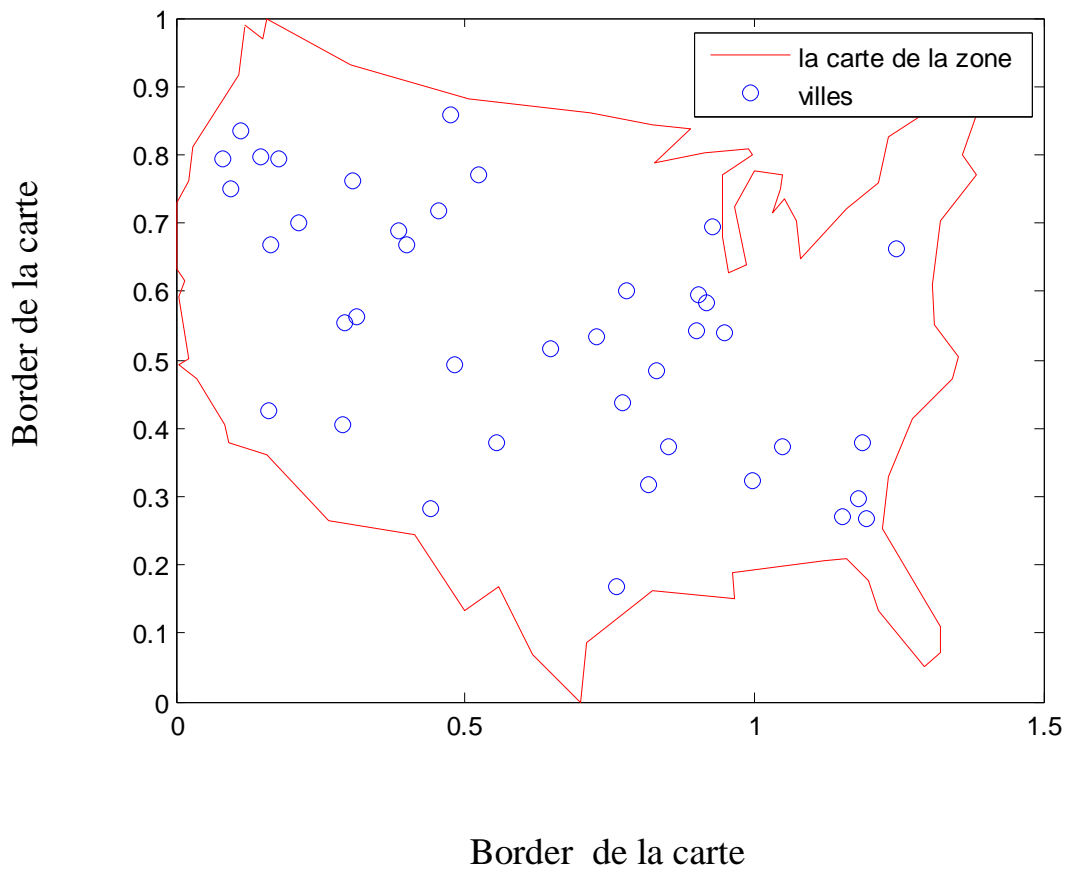


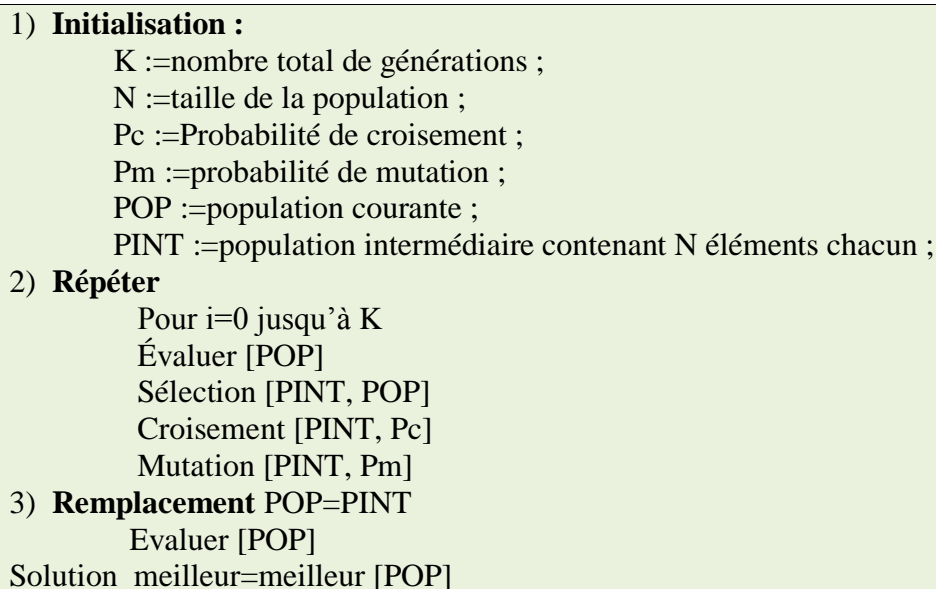
Fig. IV.2 le graphique montre l'emplacement des villes en cercles bleus.

Les cercles bleus représentent les emplacements des villes où le vendeur a besoin de se déplacer et de livrer ou de ramassage des biens. Par défaut, le solveur d'algorithme génétique permet de résoudre des problèmes d'optimisation basée sur les types de données doubles et chaîne binaire. Les fonctions de création, de croisement et mutation assument la population est une matrice de type double ou logique dans le cas des chaînes binaires. Pour résoudre ce problème qui est la détermination de trajectoire optimale que le voyageur de commerce. Doit suivre ou va utiliser deux techniques évolutionnaires qui sont : Les **l'algorithme génétique** et le **GRASP**. Ou Les l'algorithme génétique sont utilisés pour faire d'une comparaison avec les résultats de GRASP.

VI.5 Utilisation des algorithmes génétiques

VI.5.1 Définition Les l'algorithme génétique

S'inspirent des principes de la génétique. Pour cette raison, ces algorithmes accordent une grande importance à la distinction entre la représentation génétique d'un individu (génotype) et sa représentation réelle (phénotype). L'opérateur principal utilisé par les algorithmes génétiques pour la construction de nouvelles solutions est l'opérateur de recombinaison appelé aussi croisement. Cet opérateur récupère des parties du génotype de deux ou plusieurs solutions ou parents qu'il combine pour construire un ou plusieurs nouveaux génotypes ou enfants, héritant ainsi de certaines de leurs caractéristiques. L'utilisation de la recombinaison, seule, ne permet pas d'introduire du matériel génétique nouveau puisque cet opérateur combine le matériel déjà présent dans la population. Pour remédier à ce problème, les algorithmes génétiques utilisent la mutation, comme opérateur secondaire permettant d'introduire de nouveaux gènes inexistants dans la population. Le principe général d'un algorithme génétique est présenté dans la figure VI. 3 (fig. VI. 3).

- 
- ```

1) Initialisation :
 K :=nombre total de générations ;
 N :=taille de la population ;
 Pc :=Probabilité de croisement ;
 Pm :=probabilité de mutation ;
 POP :=population courante ;
 PINT :=population intermédiaire contenant N éléments chacun ;
2) Répéter
 Pour i=0 jusqu'à K
 Évaluer [POP]
 Sélection [PINT, POP]
 Croisement [PINT, Pc]
 Mutation [PINT, Pm]
3) Remplacement POP=PINT
 Evaluer [POP]
 Solution meilleur=meilleur [POP]

```

**Fig. VI. 3.** Principe général d'un algorithme génétique.

### VI.5.2 Organigramme d'un algorithme génétique

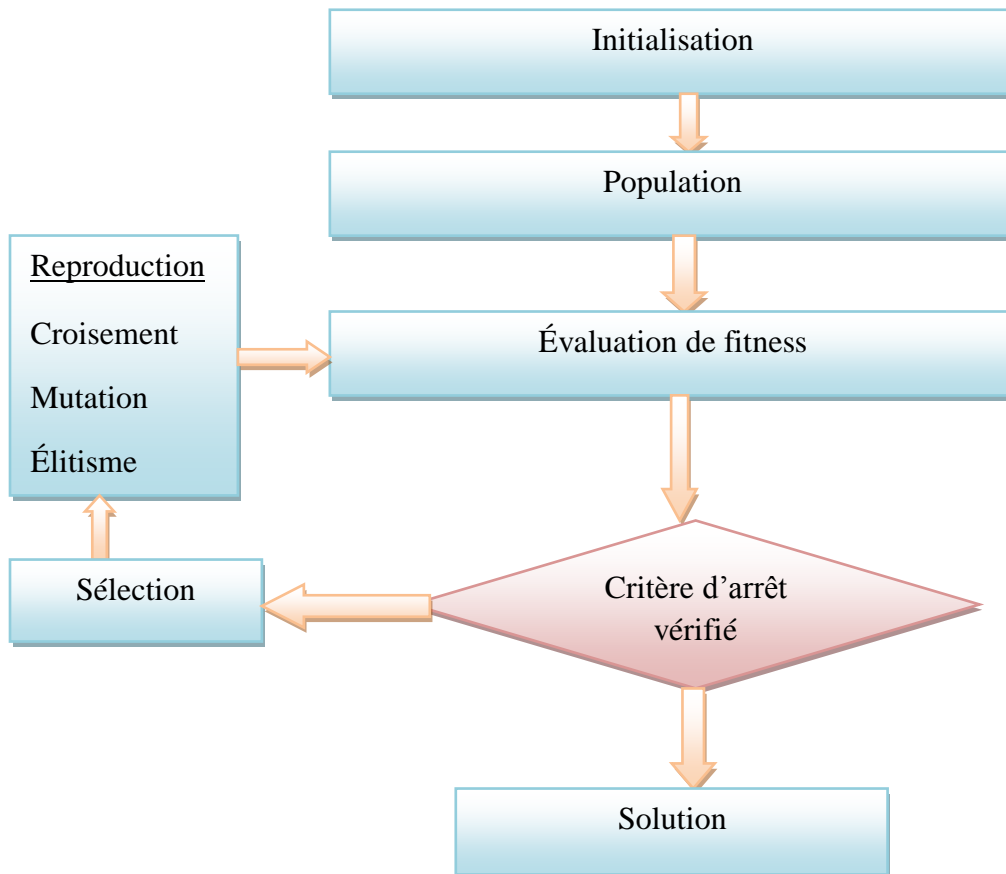


Fig. VI. 4 Organigramme d'un Algorithme génétique

### VI.5.3 résumé de l'algorithme génétique de base

**[Début] :** générer aléatoirement une population de  $n$  chromosomes.

**[Evaluation] :** évalué l'adaptation (fitness)  $f(i)$  de chaque chromosome dans la population.

**[Nouvelle génération] :** créer une nouvelle population en répétant les étapes suivantes jusqu'à ce que la nouvelle population est complétée.

1. **[Sélection] :** Sélectionner des chromosomes parents de la population relativement à leurs fitness (meilleur fitness, une grande chance d'être sélectionné,...)

**2. [Croisement]:** Avec la probabilité du croisement fixée, croiser les parents pour former les nouveaux descendants. Si aucun croisement n'a eu lieu les descendants sont des copies exactes des parents.

**3. [Mutation] :** Avec la probabilité de mutation fixée, muter le nouveau descendant à chaque locus (position dans le chromosome).

**4. [Acceptation] :** Placer les nouveaux descendants dans la nouvelle population

**[Remplacer] :** Utiliser la nouvelle population générée pour l'exécution de l'algorithme de nouveau.

**[Test] :** Si la condition fin est satisfaite stop et retourner la meilleure solution dans la population courante.

**[Boucler] :** Aller à l'étape 2.

### Le but des l'algorithme génétique

Le but des l'algorithme génétique est de déterminer les extrêmes d'une fonction, où  $f: X \mapsto \mathbb{R}$ , ou  $X$  est un ensemble quelconque appelé espace de recherche et  $f$  est appelée fonction d'adaptation ou fonction **d'évaluation** ou encore fonction **fitness**. La fonction agit comme une «boite noire » pour l'algorithme génétique Aussi des problèmes très complexes peuvent être approchés par programmation génétique sans avoir de compréhension particulière du problème.

### Résultats pratiquement de l'algorithme génétique

Si nombre de villes ( $n=10$ ).

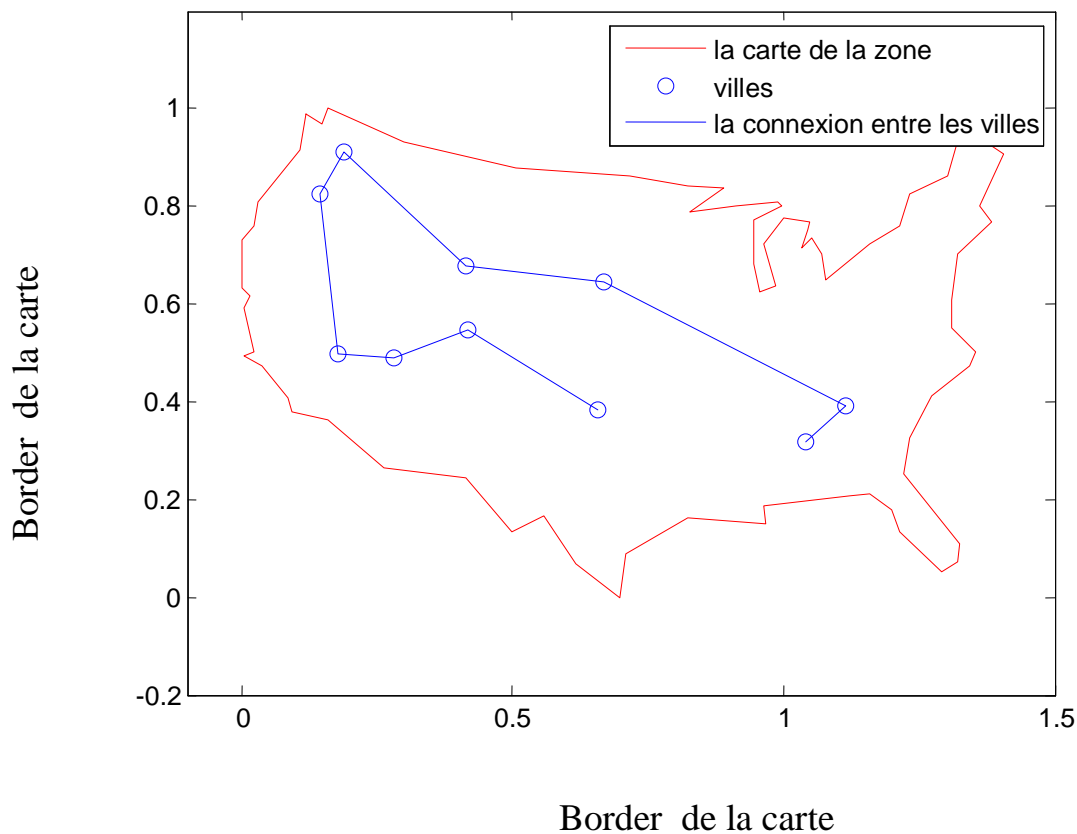


Fig. IV.5 résultats l'algorithme génétique si nombre de villes ( $n=10$ ).

Si nombre de villes (n =40)

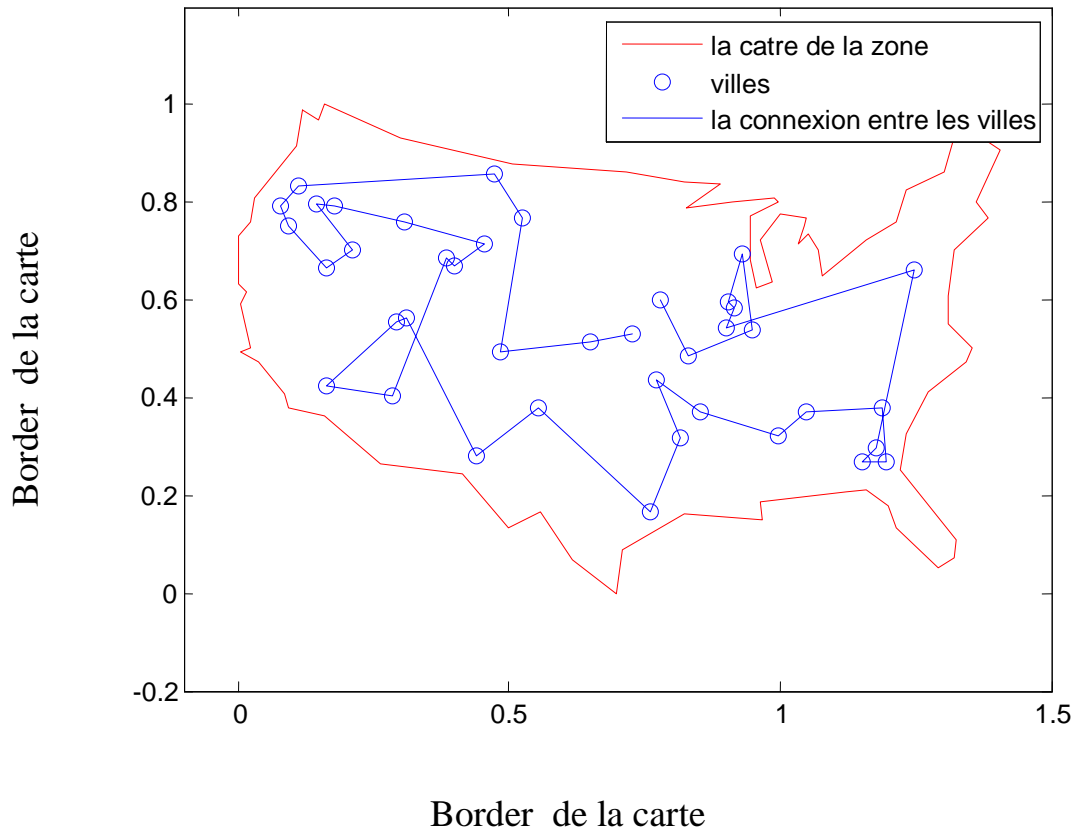
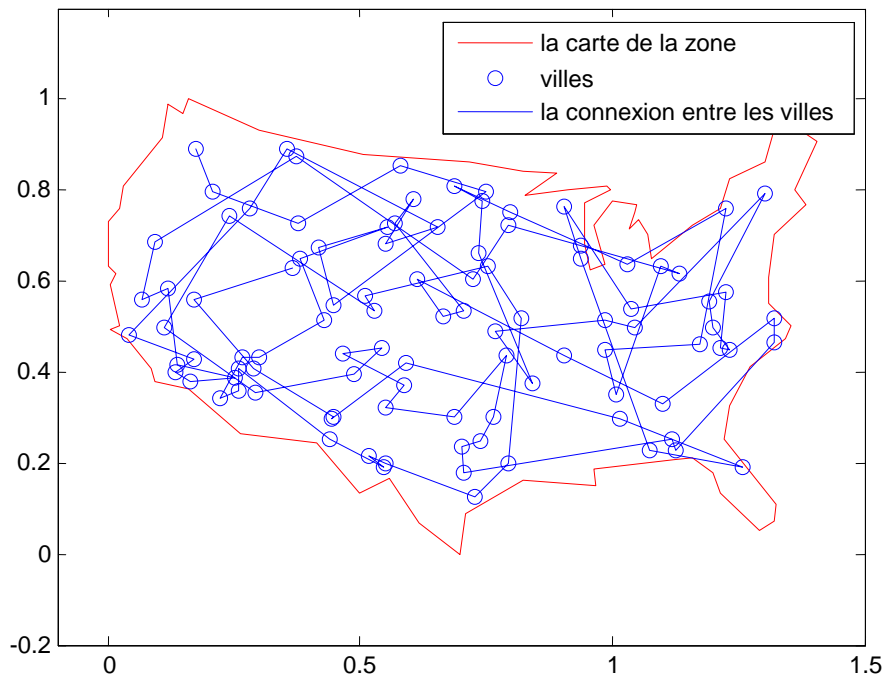


Fig. IV.6 résultats l’algorithme génétique si nombre de villes (n=40).



Si nombre de villes ( $n = 100$ )



**Fig. IV.7** résultats l'algorithme génétique si nombre de villes ( $n=100$ ).

### Discussion sur résultats l'algorithme génétique ( $n=10$ , $n=40$ , $n=100$ ).

Au cours de mon étude théorique et l'application de l'algorithme génétique on a est élevé obtenu le résultat suivant : (Fig. IV.5, Fig. IV.6, Fig. IV.7) on remarque et on conclut que plus le nombre de villes le problème de voyageur de commerce est plus difficile d' résoudre de sorte qu'on ne pas accéder au résultat à obtenir on note aussi que le temps d'exécution de génétique est environ 7 secondes. D'on constate également que problème du voyageur de commerce peut être difficile et complexe et que l'algorithme génétique et ne permet pas retour de voyageur de commerce

## IV.6 Utilisation des GRASP

### IV.6.1 Définition

**GRASP** : c'est un processus multi-Start ou itérative, chaque itération consiste de 2 phases principales : une phase de construction ou une solution réalisable est produite, et une phase de recherche locale ou un optimum sera trouver dans le voisinage de l'élément construit.

## IV.6.2 principe de GRASP

### IV.6.1.1 Fonctionnement

GRASP est un algorithme metaheuristique basant dans son fonctionnement sur deux techniques d'optimisation largement connues, il s'agit de l'algorithme glouton (Greedy) et la recherche locale.

Pour un nombre donné d'itérations, on effectue la construction d'une solution suivant une heuristique semi-glouton (Randomized Greedy), puis on applique une recherche locale sur le voisinage de cette solution afin d'obtenir un optimum local, la solution finale sera la meilleur

Solution obtenue parmi toutes les solutions de chaque itération. On remarque l'indépendance des itérations dans GRASP. [18], [19].

La figure suivante montre l'organigramme général du déroulement de GRASP.

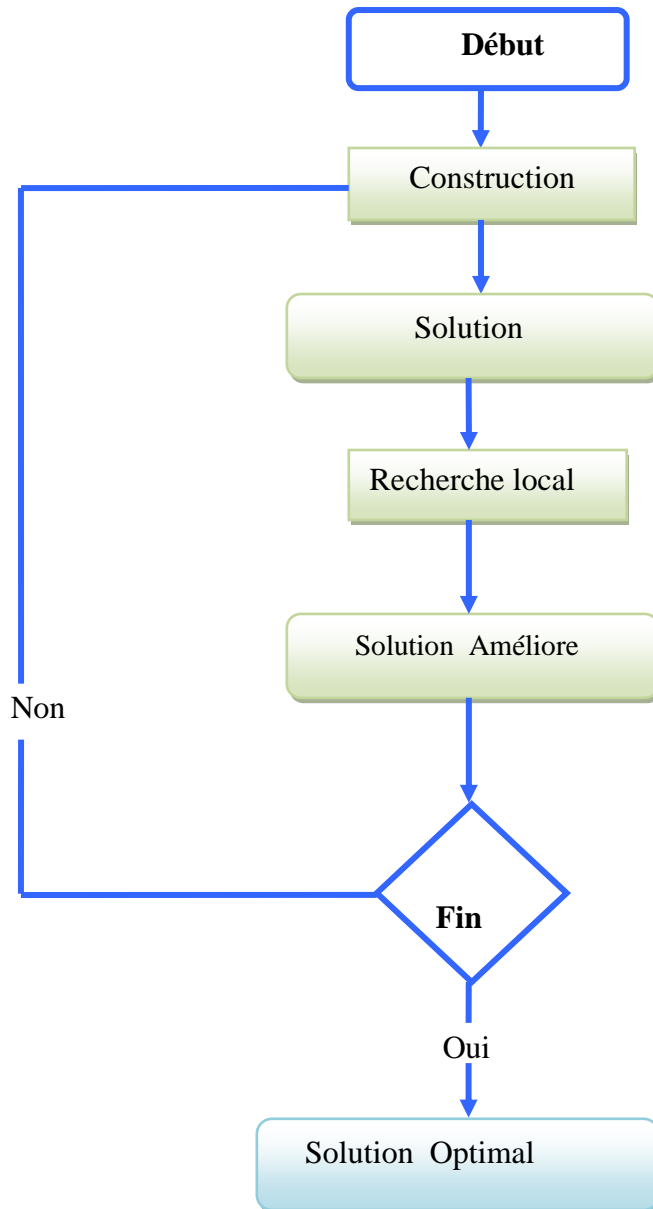


Figure VI.8 : organigramme générale de Greedy Randomized Adaptive Search Procédure

#### IV.6 .1.2 Phase de Construction

Dans la phase de construction une solution réalisable itérativement construite, un élément après l'autre. A chaque itération de construction, le choix de l'élément suivant qui va être ajouté est déterminé par l'ordre de tous les éléments candidats (élément candidat==élément qui peut être ajouté à la solution) dans la liste des candidats « C » «Candidate List »respectons la fonction avide  $g : C \rightarrow \mathbb{R}$

. Cette fonction mesure le bénéfice de sélectionner chaque élément, cette méthode est adaptative car les bénéfices associés à chaque élément sont  $m-a-j$  avec chaque itération de la phase de construction pour bien refléter le changement fait par l'ajout de l'élément précédent.

GRASP est caractérisé par la sélection aléatoire d'un élément parmi les meilleurs candidats dans la liste, mais pas nécessairement le TOP-candidat. La liste des Best-candidats appelée « Restricted candidate list » « RCL ».

Cette technique de sélection permet d'obtenir différentes solutions à chaque GRASP itération mais pas nécessairement compromettre l'efficacité de la nature avide adaptative de l'algorithme (méthode).

```

Procédure Construction (g (.), α , x)
 X= \emptyset ;
 Initialisation candidat liste C ;
 While C $\neq \emptyset$ Do
 sn= min { g(t)|t \in C } ;
 sx= max { g(t)|t \in C } ;
 RCL= {s \in C | g(s) \leq sn+ α (sx-sn)};
 Selecte "s" aléatoirement de la RCL;
 x=x \cup {s} ;
 Mise à jour C
 END-WHILE ;
End Construction

```

**Fig.VI.9:** Procédure «Construction»

Le paramètre  $\alpha$  contrôle deux caractéristiques majeures de cette méthode l'avidité et la recherche aléatoire

- Si  $\alpha=0$  correspond à une procédure de construction avide 100%.
- Si  $\alpha=1$  correspond à une procédure de construction aléatoire 100%.

Comme la plupart des méthodes déterministes, les solutions générées par phase de construction GRASP n'en pas localement optimale, la ou la recherche locale est introduit pour résoudre ce problème.

## IV.6 .1.3 Organigramme Phase de Construction

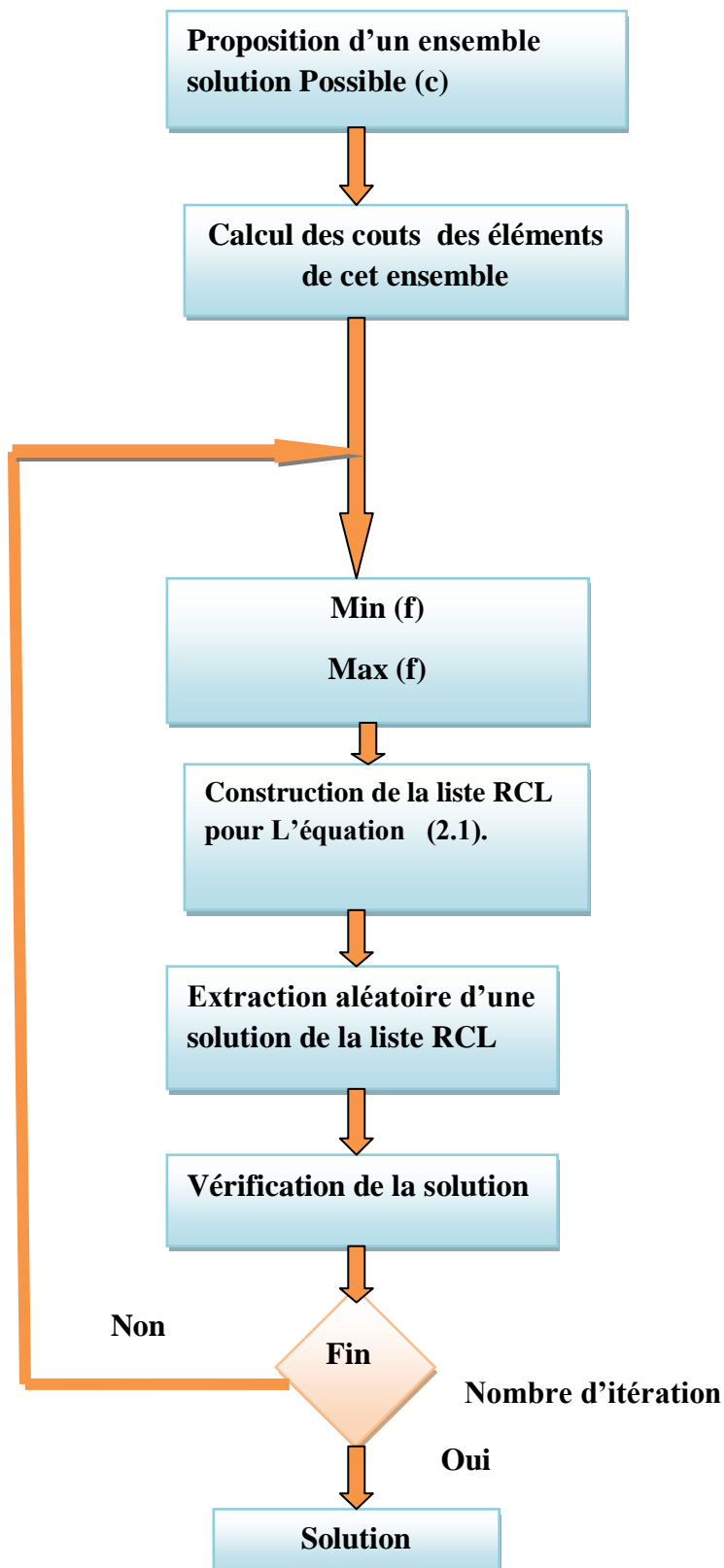


Fig.VI .10: Organigramme Phase de Construction

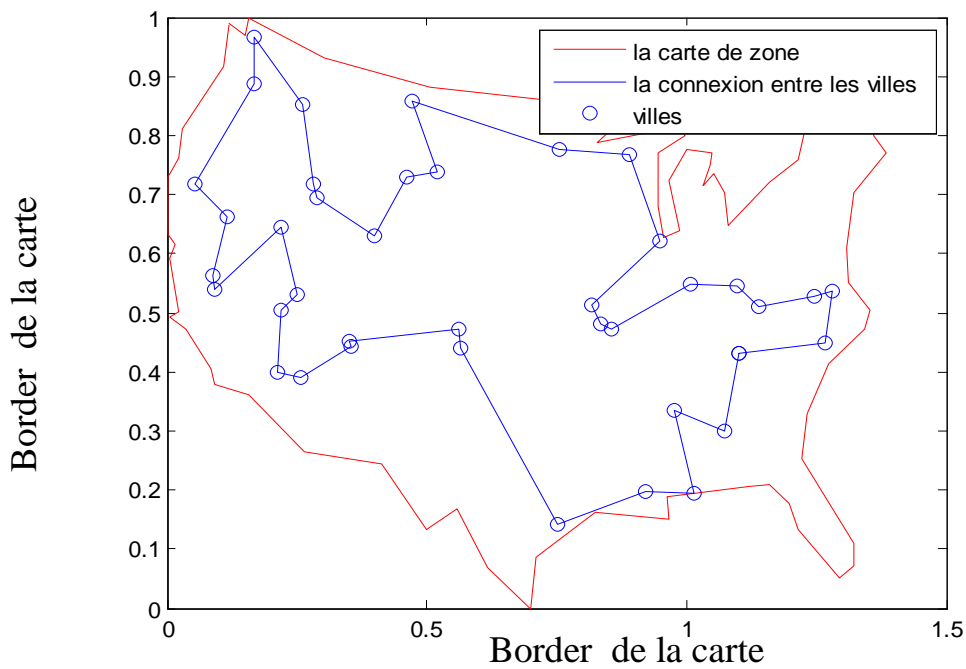
**IV.7 Résultats et discussions Phase de Construction**

**IV.7.1 Résultats Phase de Construction**

Si nombre de villes (n =10)

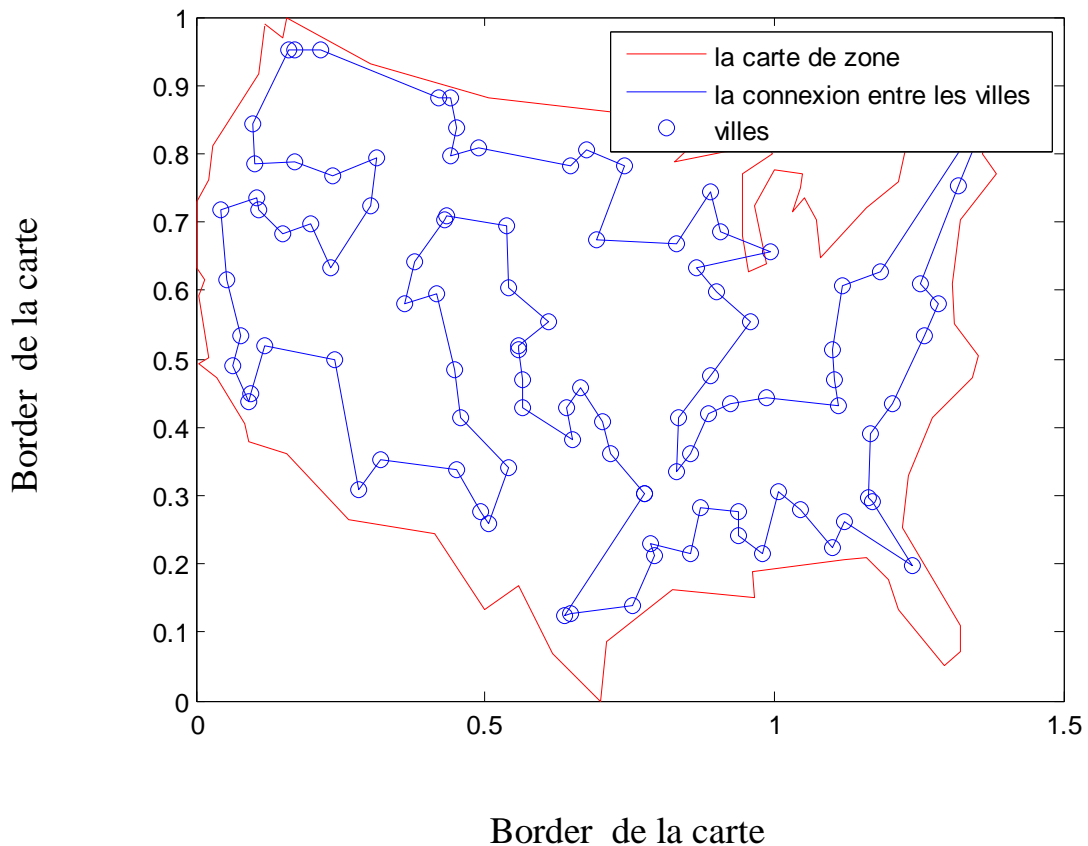
**Fig. IV.11** : résultats phase de construction si nombre de villes (n =10)

Si nombre de villes (n =40)



**Fig. IV.12** : résultats phase de construction si nombre de villes (n =40)

Si nombre de villes ( $n = 100$ )



**Fig. IV.13** : résultats phase de construction si nombre de villes ( $n = 100$ )

Discussion sur résultats phase de construction si nombre de villes ( $n=10, n=40, n=100$ ).

Au cours de mon étude théorique et l'application dans le cas de la phase de construction, a obtenu le résultat suivant que Je constate que quand vous obtenez le résultat dans la phase de construction obtenu rapidement et n'a pas pris de temps Il a été conclusion que chaque fois qu'on augmente le nombre de villes cela me prend pas beaucoup de temps d'exécution

### IV.7.3. Comparaison avec phase de construction et les algorithmes génétiques

Notez par mes études de l'algorithme génétique et la phase de construction, il ya une grande différence et claire, en particulier dans le cas d'un certain nombre 40 et 100 villes. C'est-à-dire que la phase de construction est un moyen facile de clair et simple et a un point de départ et le point final par rapport à l'algorithme génétique où

- Manque de connaissances théoriques.



- Puisque les résultats sont générés de manière stochastique, il est dit que leur comportement peut être imprévisible ou difficiles à vérifier.

- **IV.8 Phase de recherche locale**

- **IV.8.1 Définition**

C'est toujours bon d'appliquer une recherche locale dans une tentative d'améliorer chaque solution déjà construite. Un algorithme de recherche locale est itérative, il remplace successivement la solution existante par une meilleur solution dans le voisinage de la solution courante. L'algorithme s'achève ou aucune solution meilleure n'est trouvée dans ces voisins. La structure de voisinage « N » pour un problème 'P' la solution 's'est reliée à un nombre de solutions N(s). une solution 's'est dite « localement optimale » si n'existe aucune meilleure solution dans N(s). Le choix approprié de la structure de voisinage, une efficace technique de recherche de voisinage et la solution initiale sont des facteurs majeurs pour une efficace phase de recherche locale. Quand un tel de procédure d'optimisation locale peut exiger un temps exponentiel d'un point de départ arbitraire (aléatoire). Par expérience l'efficacité de la « recherche locale » sera meilleure quand la solution initiale est appropriée, avec des données personnalisés et modifier et une implémentation prudente une efficace phase de construction peut génère des bons solutions initiaux pour une phase de recherche locale efficace. C'est difficile d'analyser la qualité de la solution générée par le GRASP mais on a vu que le GRASP est répétitive, chaque itération produit une solution prototype depuis une distribution inconnue de tous les résultats obtenues.

La figure suivante illustre le pseudocode de la recherche locale.

```

Procédure Recherche Local(Solution)
 Tant-Que (Solution non optimal) faire
 Rechercher S' tq $f(S') \leq f(\text{solution})$
 Solution $\leftarrow S'$
 Fin Tant-Que
 Retourne Solution
Fin Recherche Local

```

**Fig.VI.14 :** Procédure «Recherche locale»

## IV.8.2 Organigramme phase de recherche locale

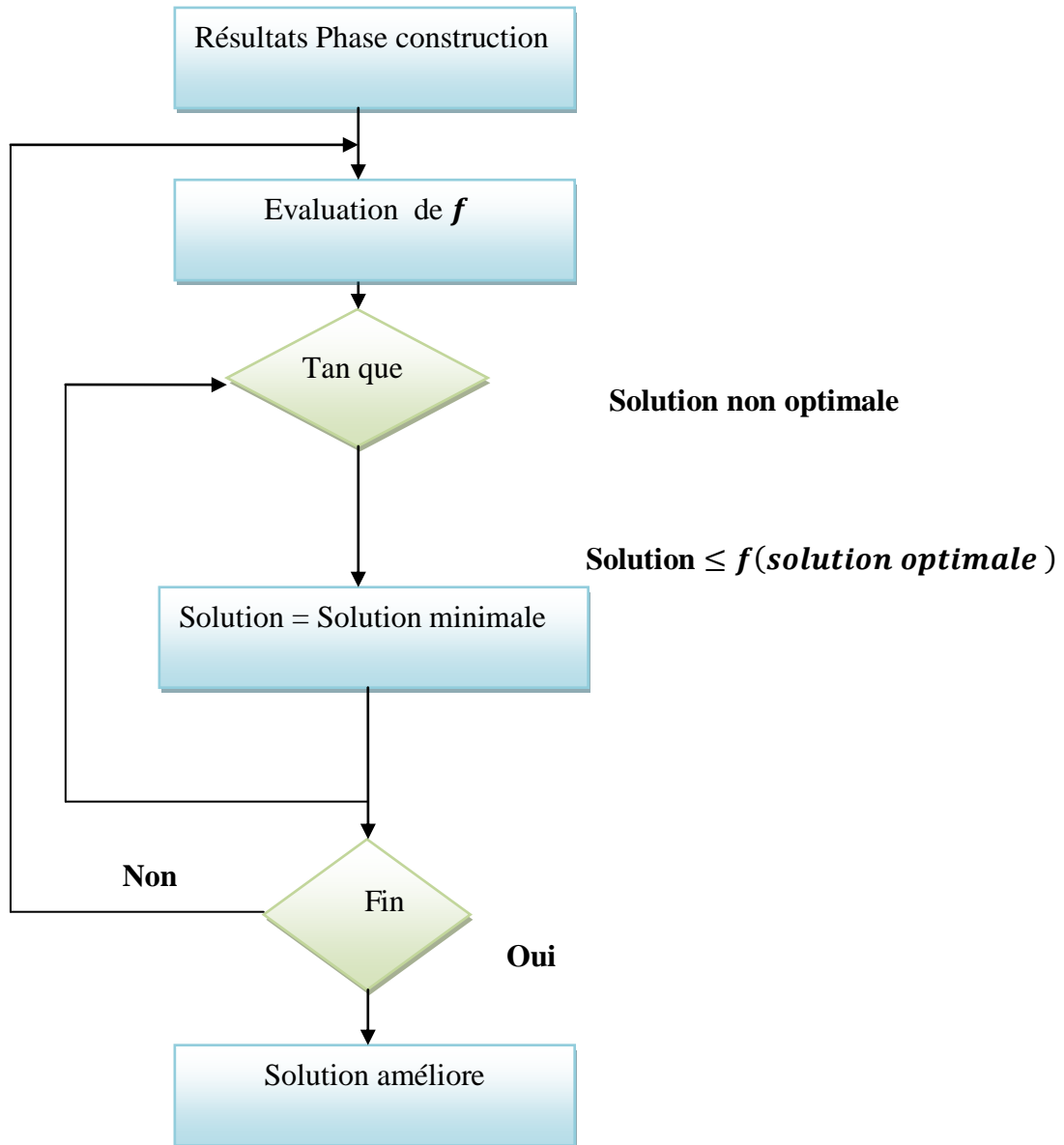
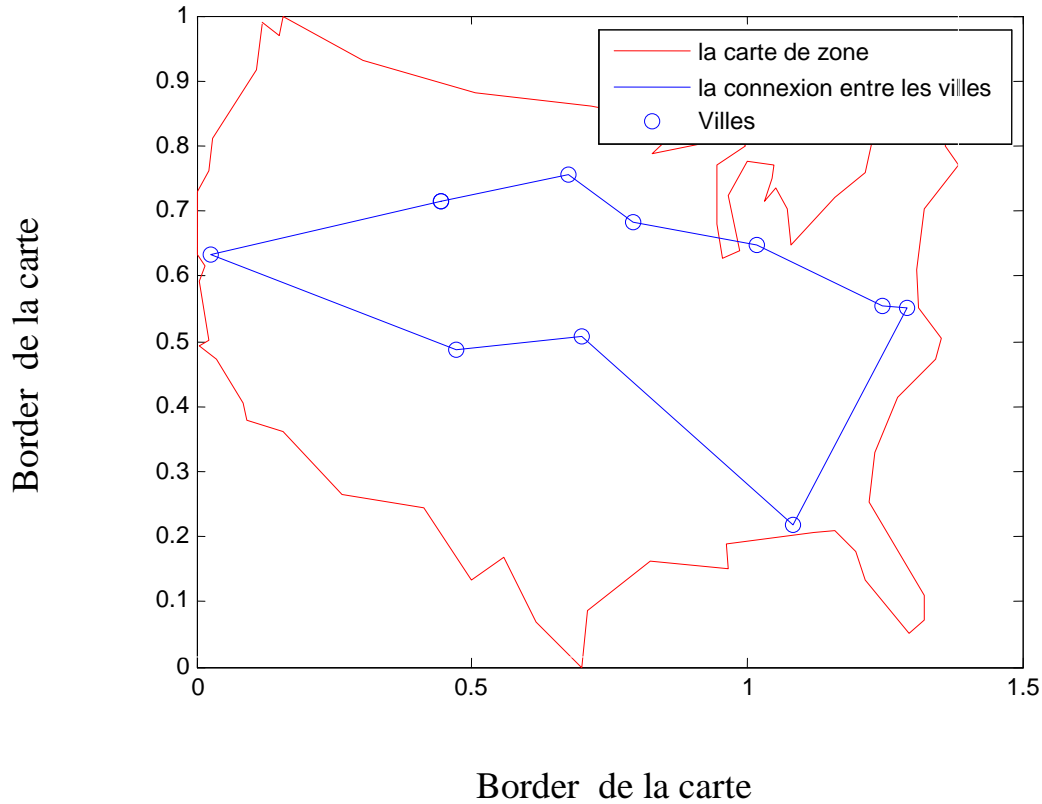


Fig. VI. 15 : Organigramme phase de recherche locale

**IV.7 Résultats et discussions** phase de recherche locale

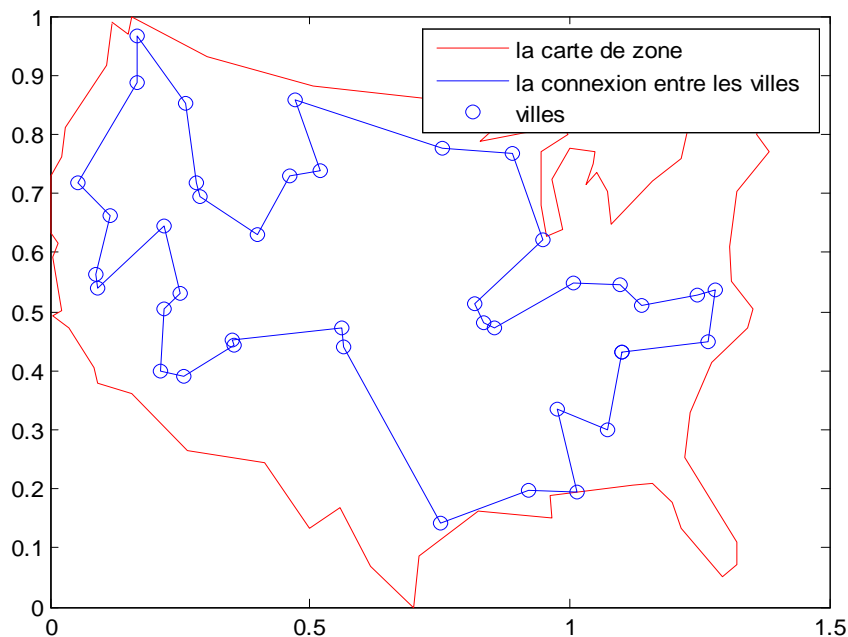
**IV.7.1 Résultats** phase de recherche locale

Si nombre de villes (n =10)



**Fig. IV.16** : résultats phase de recherche locale si nombre de villes (n =10)

Si nombre de villes (n =40)



**Fig. IV.17**: résultats phase de recherche locale si nombre de villes (n =40)

Si nombre de villes ( $n=100$ )

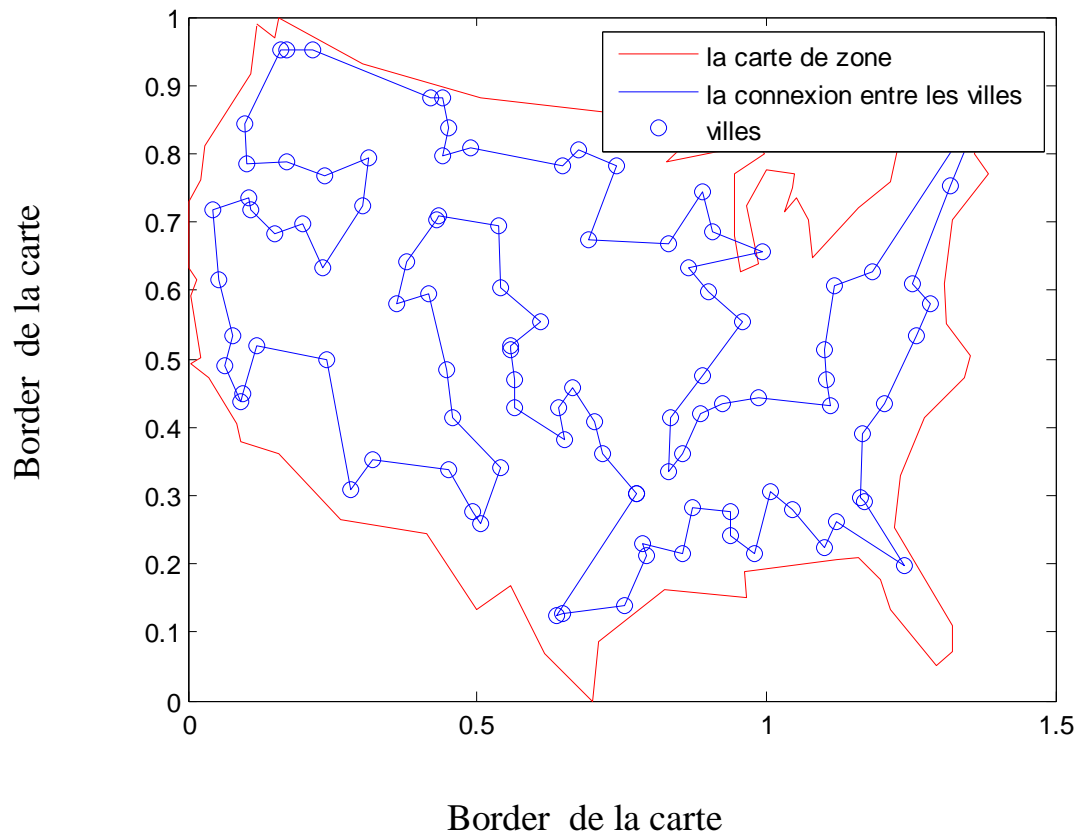


Fig. IV.18 : résultats phase de recherche locale si nombre de villes ( $n=100$ )

#### IV.8.4 Discussion sur résultats phase de recherche locale si nombre de villes ( $n=10$ , $n=40$ , $n=100$ )

On a remarqué Au cours de mon étude théorique et l'application dans lequel on a obtenu le résultat suivant. (Fig. IV.11 Fig. IV.12, Fig. IV.13) Je conclusion que le résultat de la phase de recherche locale est la même à la suite de la phase de construction. C'est-à-dire la phase de recherche locale et phase de construction du rôle amélioration.

#### IV.8.5 Comparaison avec phase de recherche locale et phase construction

Il n'y a aucune différence entre la phase de construction et la phase de la recherche locale et leur phase amélioration des rôles

### IV.8.6 Comparaison avec phase de recherche et les algorithmes génétique

Comparaison entre la phase de recherche locale et de l'algorithme génétique sont la même comparaison de la phase de construction avec l'algorithme génétique.

C'est-à-dire la phase de construction et la phase de recherche locale est de façon claire et simple qui nous donne le résultat audités et clair

### IV.8.7 Compression avec GRASP et les algorithmes génétique

#### Algorithmes génétique

##### ❖ Avantages

En dépit de ces insuffisances, les algorithmes génétiques peuvent exceller quand:

##### ❖ inconvénients

- Manque de connaissances théoriques.
- Puisque les résultats sont générés de manière stochastique, il est dit que leur comportement peut être imprévisible ou difficiles à vérifier.

#### Méthode GRASP

##### ❖ Avantages

- Implémentation simple et facile.
- Facile d'ajuster et modifier les paramètres.
- Possibilité d'introduire nouveaux concepts d'améliorât d'ajustements.

##### ❖ inconvénients

- La solution générée peut être sous-optimale.

### IV.9 Conclusion

Au cours de mon étude théorique et l'application qu'on a effectué, l'utilise. de la méthode GRASP en problème de voyageur de commerce a donné sons résultats par rapport aux algorithmes génétique soit de coté temps d'exécution, soit de coté résultats (trajectoire optimal)



*Conclusion générale*  
*Et perspectives*

## *Conclusion générale et perspectives*

---

On peut constater que le problème de voyageur de commerce est un plan standard dont il est sujet à d'application de plusieurs techniques d'optimisation stochastiques ou déterministe. C'est pourquoi on a choisi de l'utiliser dans notre application. en L'utilise de la méthode GRASP en le problème de voyageur de commerce a donné sons résultats par rapport aux algorithmes génétique soit de coté temps d'exécution, soit de coté résultats (trajectoire optimal)

### **Perspectives**

Dans le futur, nous proposons ;

- ❖ Utiliser autre méthode métaheuristiques.
- ❖ Choisir la segmentation de visage.
- ❖ Utiliser une méthode plus évolué pour l'extraction des paramètres



## *Références bibliographiques*

---

- [1] Christelle Guéret, Christian Prins et Marc Sevaux : *Programmation Linéaire*, Eyrolles, 2000.
- [2] A.Sbihi : *Les méthodes hybrides en optimisation combinatoire : Algorithmes exactes et heuristiques*. Thèse de doctorat de l'université Paris-I Panthéon-Sorbonne. Année 2003.
- [3] Chandru V. and Rao M.R. *Linear Programming*, Chapter 31 in *Algorithms and Theory of Computation Handbook*, edited by M.J.Atallah, CRC Press 1999.
- [4] J.Dréo, A.Pétrowski, P.Siarry, E.Taillard : *Métaheuristiques pour l'optimisation difficile*. Edition Eyrolles, 2003.
- [5] : Paul Feautrier, « Recherche opérationnelle », 16 novembre 2005.
- [6] : C. Blum and Roli, « Metaheuristics in combinatorioptimization : overview and conceptual comparison », *ACComputing survey*, vol 35, N°3, september 2003.
- [7] : <http://www.metaheuristics.org>
- [8] C.H.Papadimitriou. *The complexity of combinatorial optimization problems*. PhD thesis, Princeton, 1976
- [9] S. Kirkpatrick, D.C.Gelatt, and M.P.Vechhil: *Optimization by simulated annealing* *Science*, 220:671–680, May1983.
- [10] Glover, F: *Tabu search—Part II*, *ORSA J. Comput.* 2:4-32. 1990.
- [11] T.Feo et M.Resende: *Greedy Randomised Adaptive Search procedure*. *Journal of Global Optimization*, tome 42, pages 860-878, 1995.
- [12] K. Price, R. Storn, *Differential evolution – a simple and efficient adaptive scheme foglobal optimization over continuous spaces*, *Technical Report*, InternationalComputer Science Institute, Berkley, 1995.
- [13] A.Coloni, M.Dorigo, V.Maniezzo : *Distributed Optimization by Ant Colonies*. *Proceedings of ECAL'91-First European Conference on Artificial Life*, édité par F.Varela, et al. pages 134\_142.Elsevier Publishing, Paris, France, 1992.
- [14] Johann Dréo, Patrick Siarry : *Métaheuristiques pour l'optimisation et auto- organisation dans les systèmes biologiques*. Université de Paris XII Val-de-Marne.
- [15] Kennedyj., Eberhartr. C., « Particle swarm optimization », *Proc. Ieee Int.Conf. On Neural Networks*, vol. IV, Piscataway, NJ: Ieee Service Center, 1995, p. 1942-1948.
- [16] Yann Cooren, Maurice Clerc et Patrick SIARRY: *Optimisation par essaim particulaire amélioré par hybridation avec un algorithme à estimation de Distribution*. *Laboratoire Images, Signaux et Systèmes Intelligents, Lieei, E.A. 3956 Université de Paris XII*.

## *Références bibliographiques*

---

- [17] R.C. Eberhart, Y. Shi, Comparing inertia weights and constriction factors in particle swarm optimization, Proceedings of the IEEE Congress on Evolutionary Computation 1 (2000) 84–88.
- [18] M. Resende and C. Ribeiro, "Greedy Randomized Adaptive Search Procedures," International Series in Operations Research and Management Science, pp. 219–250, 2003.
- [19] J. Skaf and S. Boyd, "Filter Design With Low Complexity Coefficients," IEEE Transactions on Signal Processing, vol. 56, pp. 3162–3169, 2008.
- [20] W. Sung and K. Kum, "Simulation-based word-length optimization method for fixed-point digital signal processing systems," IEEE Transactions on Signal Processing, vol. 43, no. 12, pp. 3087–3090, 1995.
- [21] Mauricio G.C. Resende, Celso C. Ribeiro - Greedy Random Adaptive Search Procedure 29/08/2002. PAOLA FESTA, MAURICIO G.C. RESENDE - Grasp: An Annotated.
- [22] Jason Brownlee - Clever Algorithms: Nature-Inspired Programming Recipes – 2011 – CC.
- [23] El-Ghazali Talbi - Metaheuristics From Design To Implementation-2000.
- [24] Thomas A. Feo, Mauricio G.C. Resende A Probabilistic 1995.  
Heuristic For A Computationally Difficult Set Covering Problem, Avril 1989. Implementation – 2009 – Wiley.
- [25] Baumgartner, Johannes J. Schneider - On the Neighborhood Structure of the Traveling Salesman Problem Generated by Local Search Moves – 2007.
- [26] Paola Festa, Mauricio G.C. Resende - Grasp: An Annotated Bibliography – 2001.
- [27] T. Christof and G. Reinelt, (1995) "Parallel cutting plane generation for the TSP in Parallel Programming and Applications" (P. Fritzon, L. Finmo, eds.) IOS Press, 163–169.
- [28] [http://www.lim.univ-mrs.fr/~vancan/mait/documents/cours4\\_8.pdf](http://www.lim.univ-mrs.fr/~vancan/mait/documents/cours4_8.pdf).
- [29] français, <http://www.crdp.ac-grenoble.fr/imel/index.htm>.
- [30] <http://labo.algo.free.fr/defi250/definitiondes250villes.html>.
- [31] [http://www.claire-language.com/files/whitepaper/thesis\\_fla.pdf](http://www.claire-language.com/files/whitepaper/thesis_fla.pdf).
- [32] <http://www.inra.fr/bia/T/degivry/Givry03c.pdf>.

## *Références bibliographiques*

---

## Famille Les algorithmes génétiques

### \*/ Les algorithmes génétiques

Les algorithmes génétiques sont des algorithmes d'optimisation développés dans les années 70 avec le travail de Holland puis approfondis par Goldberg qui a largement contribué à les vulgariser. Les algorithmes génétiques sont la branche des algorithmes évolutionnaires la plus connue et la plus utilisée. La particularité de ces algorithmes est qu'ils font évoluer des populations d'individus codés par des chaînes de longueur fixe (codage classique utilisé à l'origine : des chaînes de bits (0/1)) tout en utilisant des opérateurs d'évolution génétique de mutation et de croisement. Les individus codés de cette manière sont appelés chromosomes. Les opérateurs agissent sur un ou plusieurs individus sans connaissance sur ce qu'ils manipulent (ni sur le problème).

Un algorithme génétique recherche le ou les extrema d'une fonction définie sur un espace de données. Pour l'utiliser adéquatement, on doit disposer des six éléments suivants : (étapes préalables à la programmation).

A. 1- Un principe de codage de l'élément de population (individu). Cette étape associe à chacun des points de l'espace d'état une structure de donnée. 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 les premiers à être utilisés. Actuellement, on se sert plus souvent de codages directs (réels et entiers,...).

A. 2- Un mécanisme génération suivante. Le choix de la population initiale est important car il peut rendre plus au moins rapide la convergence vers l'optimum global. Dans le cas où l'on ne connaît rien du problème à résoudre, il est essentiel que la population initiale soit répartie

Sur tout l'espace de recherche. En pratique, on a souvent recours à la génération aléatoire de la population initiale.

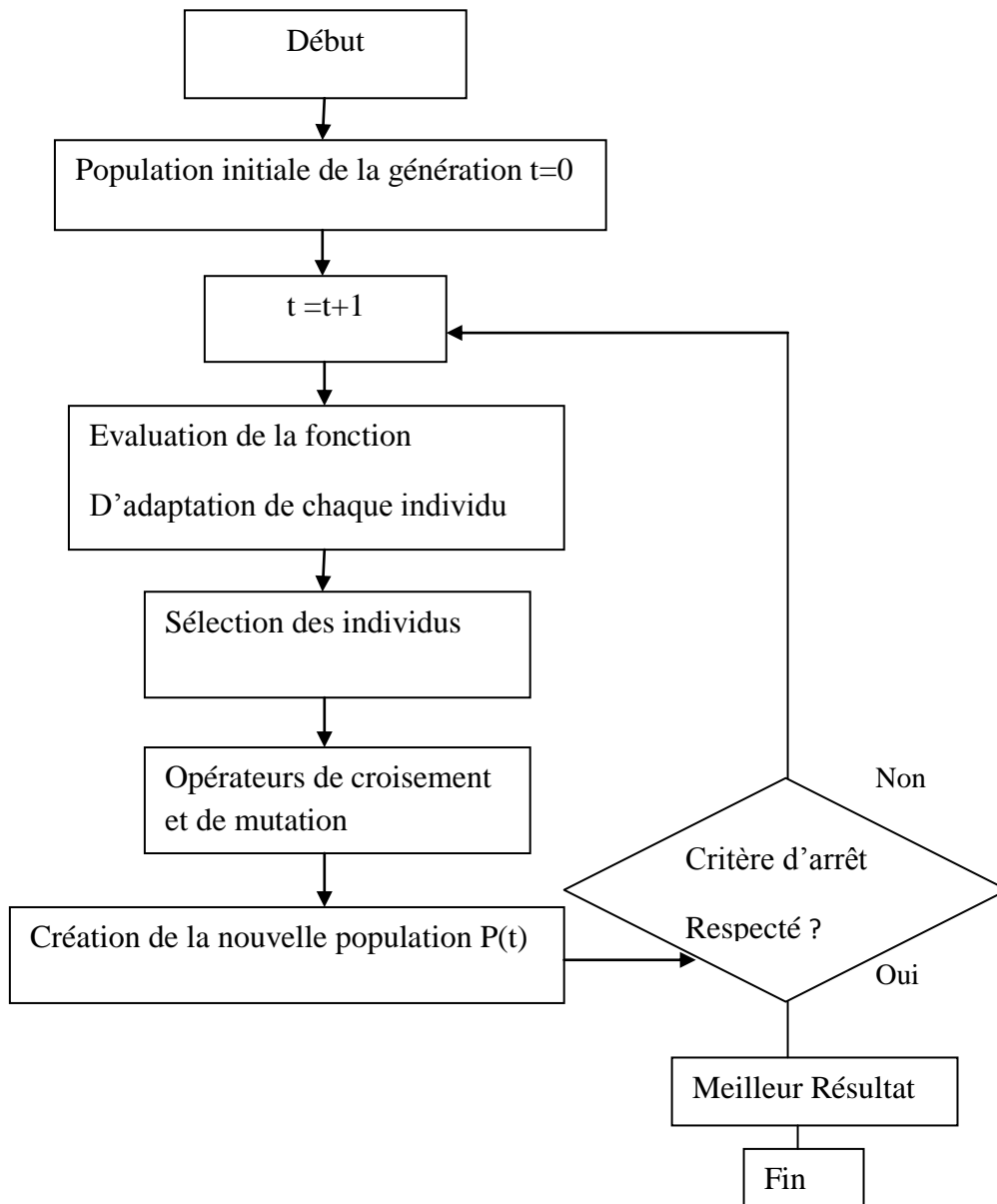
**A. 3-** Une fonction d'évaluation afin de mesurer les performances de chaque individu. Elle constitue le critère à base duquel l'individu sera ou non sélectionné pour être reproduit dans la génération suivante. Il faut savoir que la qualité de cette fonction conditionne, pour une grande part, l'efficacité de l'algorithme génétique.

**A. 4-** Un mécanisme de sélection des individus candidats à l'évolution.

**5-** Des opérateurs permettant de diversifier la population au cours des générations et d'explorer l'espace de recherche. L'opérateur de croisement recompose les gènes d'individus existants dans la population, l'opérateur de mutation a pour but de garantir l'exploration de l'espace de recherche.

**A.6-** Des paramètres de dimensionnement : taille de la population, nombre total de génération ou critère d'arrêt, probabilités d'applications des opérateurs génétiques de croisement et de mutation.

Le principe général du fonctionnement d'un algorithme génétique standard est représenté sur la figure 2.2.



**Fig. A. 2.2 :** Principe général d'un algorithme génétique standard

L'opérateur de croisement est appliqué avec une probabilité  $P_c$  (généralement autour de 0.6) L'opérateur de mutation est appliqué avec la probabilité  $P_m$  qui est très inférieure à  $P_c$  (généralement entre 0.005 et 0.01).

## **Différents critères d'arrêt de l'algorithme peuvent être choisis :**

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

L'ensemble des points précédent peut être détaillé comme suit :

### **1. le codage**

Les algorithmes génétiques travaillent sur des codages et non sur des solutions réelles. Ces codes sont appelés chromosomes. Il faut définir et coder convenablement le problème afin de cerner l'espace des solutions possibles. Ce codage doit, de plus, être aussi compact que possible pour permettre une évolution rapide.

Historiquement le codage utilisé par les algorithmes génétiques était le codage binaire. Cependant, ce type de codage n'est pas toujours bon pour certains problèmes d'optimisation pour cela d'autres types de codage sont utilisés, le codage entier, le codage réel ...etc.

### **2. la genèse de la population**

Si la position de l'optimum dans l'espace de recherche est totalement inconnue, le choix de la population initiale se fait d'une manière aléatoire. Si par contre, des informations à priori sur le problèmes sont disponibles, il paraît bien évidemment naturel de générer les individus dans un sous-espace particulier afin d'accélérer la convergence. Dans l'hypothèse où la gestion des contraintes ne peut se faire directement, les contraintes sont généralement incluses dans le critère à optimiser sous forme de pénalités et lorsque c'est possible ne générer que des éléments de la population respectant les contraintes.

### **3. la sélection**

La sélection permet d'identifier statistiquement les meilleurs individus d'une population et d'éliminer les mauvais. Un individu ayant une forte valeur d'adaptation a alors plus de chances d'être sélectionné qu'un individu mal adapté à l'environnement (ceci va permettre de favoriser la reproduction des « bons » individus).

La sélection ne crée aucune nouveauté, elle se contente de choisir parmi la population mère celles qui seront ou ne seront pas en mesure de contribuer à la création de la population fille, suivant une stratégie particulière.

Cependant, la sélection est relativement délicate à manipuler. Un excès de sélection peut entraîner une perte de la diversité au sein de la population qui bloque ainsi la résolution du problème donné, en convergeant par exemple vers des optima locaux. Une sélection trop faible pose la difficulté inverse : la non convergence de l'algorithme. Pour cela on distingue deux techniques de sélection : la sélection stochastique et la sélection déterministe.

### **a) la sélection stochastique**

Cette technique permet de favoriser les meilleurs individus mais de manière stochastique, ce qui laisse la chance aux individus moins performants d'être eux aussi sélectionnés. Par contre, il peut arriver que le meilleur individu ne soit pas sélectionné.

On distingue trois types de stratégie : La sélection proportionnelle, La sélection proportionnelle avec reste stochastique et la sélection par tournois.

#### **1- La sélection proportionnelle**

Dans ce cas chacun a une chance d'être sélectionné en fonction de son efficacité. La méthode roulette Wheel sélection (RWS), ou loterie, est la plus classique de ce mode. La première étape consiste à attribuer à chaque individu  $i$  une probabilité  $P_i$  de sélection proportionnelle à son fitness  $f_i$  et à la somme des fitness de tous les individus de la population.

Si  $N$  est le nombre d'individus de la population alors :

$$p_i = \frac{f_i}{\sum_{i=1}^N f_i}$$

La deuxième étape détermine les  $N$  individus de la population fille tirés au hasard en fonction de ces probabilités. Le tirage s'effectue généralement avec remise offrant la possibilité à un excellent individu de se retrouver plusieurs fois dans la population finale. Cette sélection permet



Donc à chacun des individus de la population d'avoir une chance aussi minime soit-elle, d'être sélectionnée.

## 2. La sélection proportionnelle avec reste stochastique

Ce mode de sélection tire profit du précédent auquel est ajouté un aspect plus déterministe. L'ensemble des  $P_i$  défini précédemment, est conservé et intervient dans l'équation qui détermine le nombre d'occurrences de la chaîne  $i$  reproduite dans la population fille. Le reste stochastique  $R_i$  est alors défini en fonction de  $N_i$ .

$$N_i = E(N * P_i) \quad R_i = (N * P_i) - N_i$$

$$S = N - \sum_{i=1}^N N_i \quad S_i = \frac{R_i}{S}$$

Les  $N_i$  représentent l'ensemble des chromosomes reproduit de manière déterministe. Généralement la somme des  $N_i$  est inférieure à  $N$ . il reste donc un certain nombre de chaînes à sélectionner défini par  $S$ . A partir des restes  $R_i$  et du nombre de chaînes  $S$ , les probabilités  $S_i$  (pour chaque individu, d'être sélectionné par le reste stochastique) sont calculées. Pour obtenir les dernières chaînes sélectionnées, une sélection proportionnelle est effectuée  $S$  fois en fonction des probabilités  $S_i$ .

Cette sélection offre, comme la précédente, une chance à toutes les chaînes, mais impose tout de même la présence des meilleurs parmi la population fille. Elle évite donc une disparition prématurée des bonnes chaînes due à un tirage aléatoire défavorable, surtout lorsque les populations de chromosomes sont de faibles tailles.

## 3. la sélection par tournois

La sélection par tournois est une alternative aux techniques de sélection proportionnelle. Le tournoi le plus simple consiste à choisir aléatoirement un certain nombre d'individus dans la population et à sélectionner pour la reproduction celui qui a la plus grande adaptation. Au cours d'une génération, il y'a autant de tournois que d'individus à remplacer. Les individus qui participent à un tournoi restent dans la population et sont de nouveau disponibles pour les tournois ultérieurs. La variance de ce processus est également élevée. La pression de sélection est ajustée par le nombre de participants à un tournoi. Choisir de nombreux participants conduit à une forte pression de sélection car un individu moyen ou faible aura moins de chance d'être sélectionné.

Une méthode dérivée fait intervenir un tournoi après l'évaluation des chaînes déjà recombinaées. Chaque couple d'enfants entre alors dans un tournoi avec ses parents respectifs afin de conserver les deux meilleurs individus des quatre en combinaison.

### **b) la sélection déterministe**

On sélectionne les meilleurs individus ( au sens de la fonction d'adaptation), cela suppose un tri de l'ensemble de la population. Les individus les moins performants sont éliminés de la population, et le meilleur individu est toujours sélectionné, on parle alors d'élitisme.

### **La sélection élitiste**

La stratégie élitiste est utilisée en plus des méthodes précédentes et non pas séparément. Elle consiste à conserver dans la population, d'une génération à l'autre, au moins l'individu ayant la meilleure adaptation. Les individus reproduits remplacent les individus les moins bons de la génération courante pour obtenir la nouvelle génération, préservant ainsi les meilleurs.

Une des variantes de la stratégie élitiste, impose la présence des X% meilleurs individus de la population initiale dans la population finale. Ainsi, ces chaînes sont tout simplement protégées d'un quelconque dérapage du hasard, elles sont

Automatiquement sélectionnées. Ce type de variante provoque donc une amélioration constante des performances de la population puisque le ou les meilleurs individus sont toujours conservés d'une population à l'autre.

## **4. le croisement**

Traditionnellement, le croisement est vu comme l'opérateur de recherche essentiel d'un algorithme génétique. Après la sélection, un croisement peut éventuellement avoir lieu. Dans ce cas, deux individus de la nouvelle génération échange une ou plusieurs partie(s) de leur génotype pour former deux individus différents de ceux d'origine.

Cet opérateur est essentiel, car il permet d'obtenir de nouveaux individus distincts de ceux déjà existants et donc d'explorer tout l'espace de recherche. Il existe différentes méthodes pour croiser deux chromosomes :

- le croisement <sup>2</sup>à un point<sup>2</sup> :

Ce type de croisement est standard dans les algorithmes génétiques. Il consiste à choisir un emplacement aléatoirement sur une chaîne et d'intervertir tous les gènes d'un côté de ce point entre les deux chaînes.

- le croisement <sup>2</sup>à n points<sup>2</sup> :

Le croisement à n points est une généralisation du croisement à un point avec n points de coupure sur les chaînes. Il s'agit alors de déterminer n points sur ces chaînes, puis d'échanger sur les chaînes les blocs entre ces points afin d'obtenir les enfants.

### **A. 5. la mutation**

Elle constitue une exploration totalement au hasard de l'espace de recherche et permet également d'éviter la perte du matériel génétique potentiellement utile. Mais, par rapport au croisement, son rôle reste secondaire. En général, on lui attribue une probabilité faible, de l'ordre de 0.01.

Cet opérateur consiste à apporter de l'innovation dans la population en modifiant un sel gène aléatoirement. Si toutes les chaînes possèdent une valeur identique sur le même gène, alors ni la sélection, ni le croisement ne pourront créer un individu pourvu d'une différence au niveau de ce gène. La mutation permet des variations de cet ordre, rendant possible la sortie d'un optimum local.

Bien que sa part dans les algorithmes génétiques soit moindre par rapport au croisement, son emploi est cependant indispensable. Dans un codage binaire muter un bit revient, tout simplement, à basculer sa valeur de 0 à 1 et inversement.

## Annexe B

### Exemple de la méthode du "2-opt améliorée" avec 30 villes.

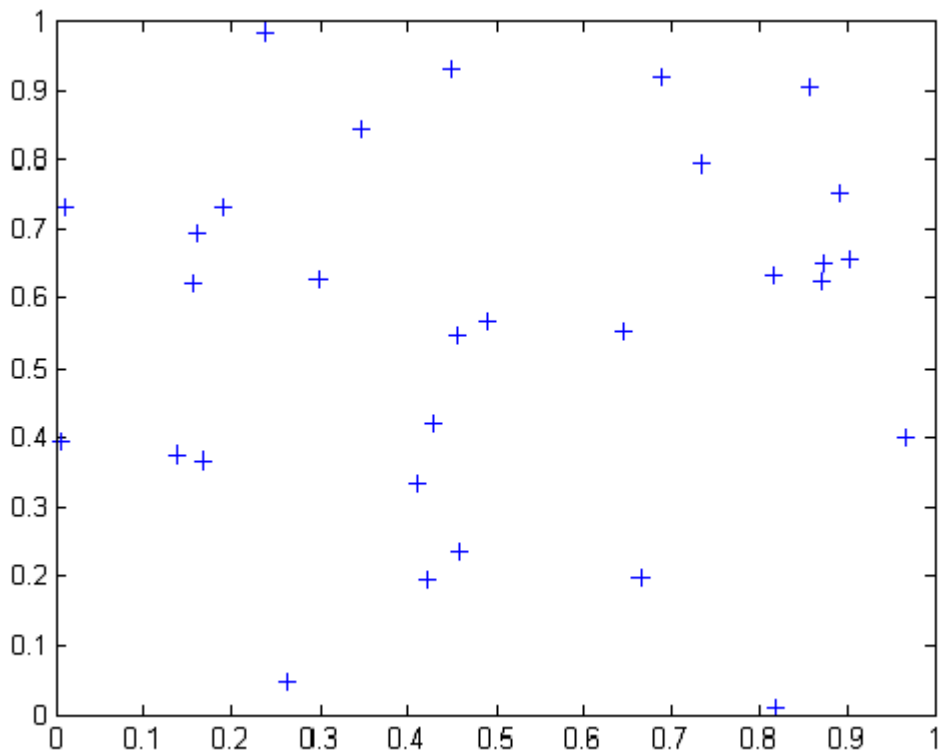
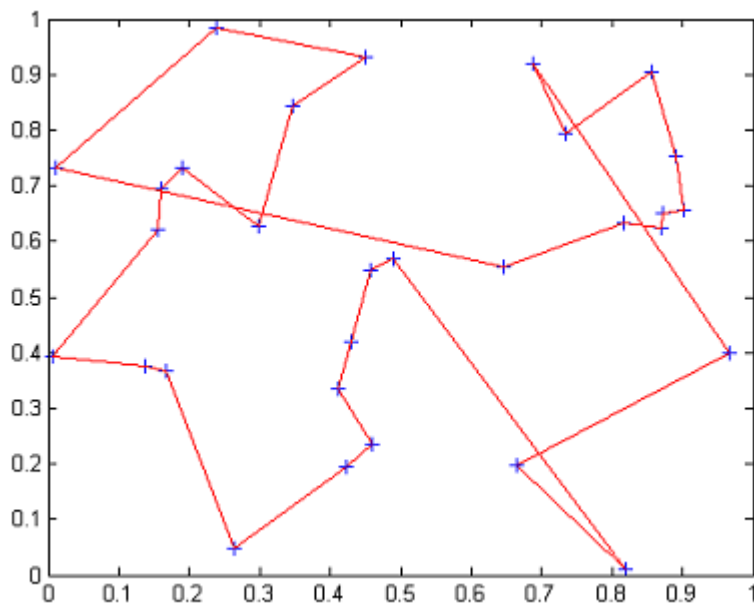
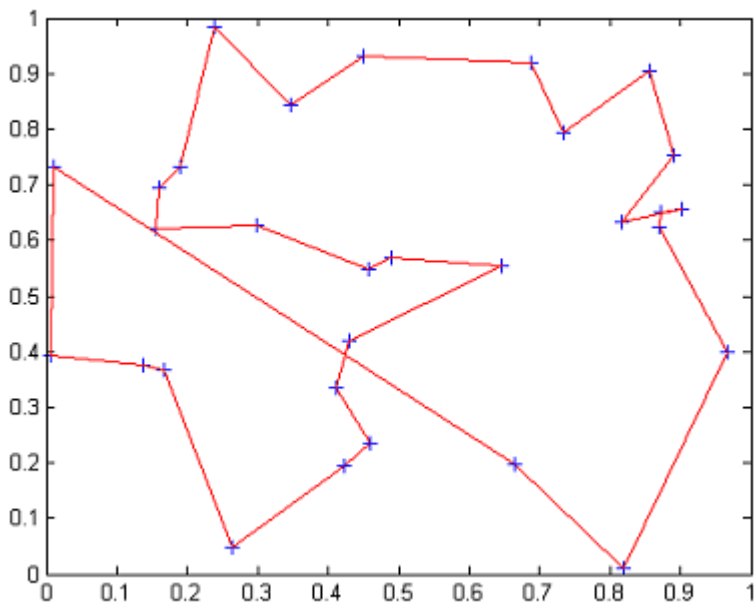


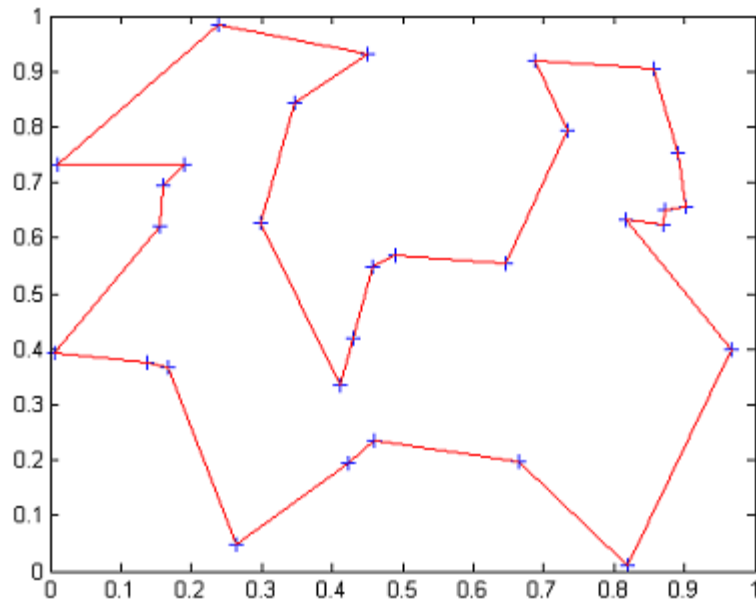
Fig. B.1 – Problème avec 30 villes.



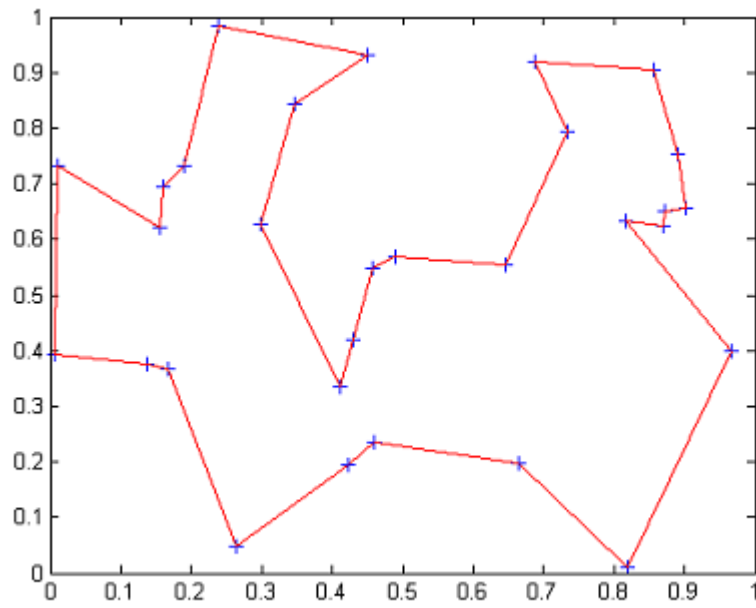
**Fig. B.2** – Chemin avec un algorithme glouton



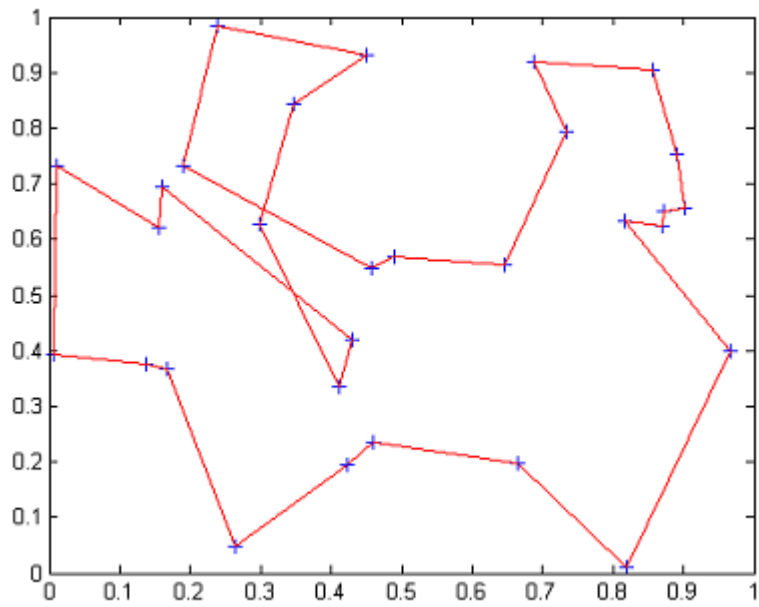
**Fig. B.3** – Chemin avec le meilleur algorithme glouton.



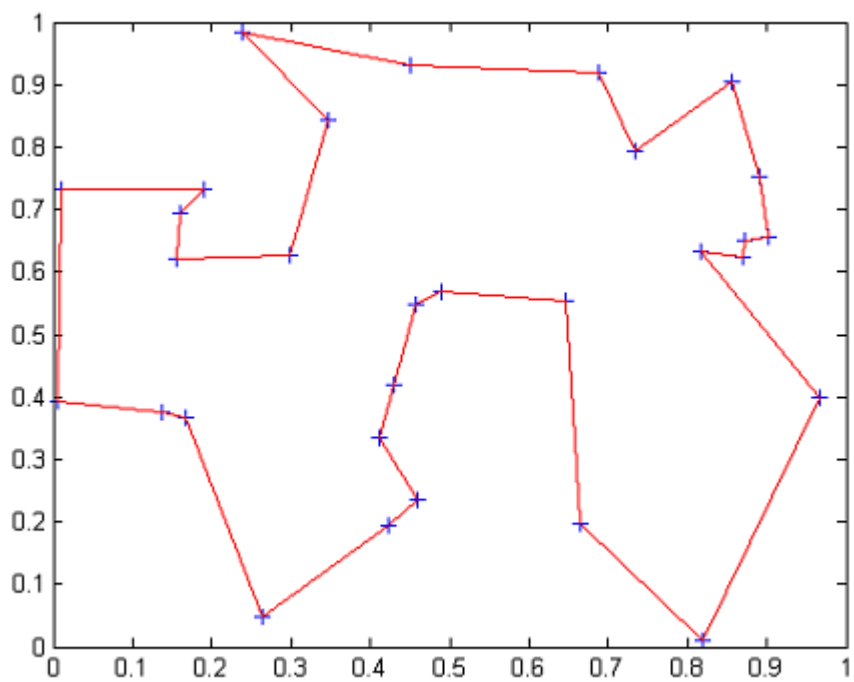
**Fig. B.4** – Chemin décroise



**Fig. B.5** – Chemin après insertion.



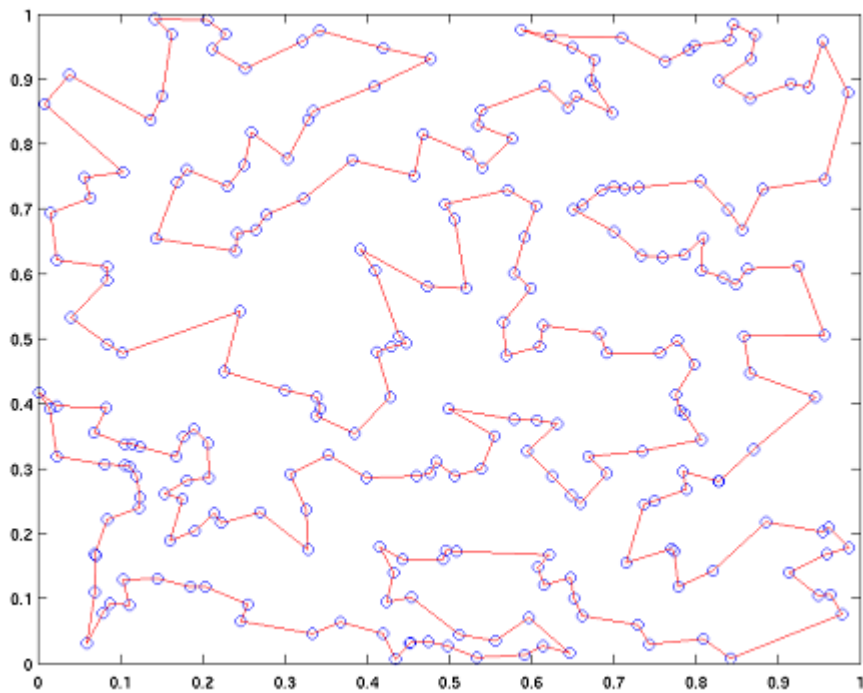
**Fig. B.6** – Exemple de flip



**Fig. B.7** – Meilleur chemin trouvé avec la méthode de "2-opt améliorée"

## Annexe C

### Le défi des 250 villes



**Fig. C.1** – Mon meilleur résultat au défi des 250 villes : 11,9926



Location: http://albo.algo.free.fr/defi250/classement.php?id=1

Labo Algo

## Le problème du voyageur de commerce : le défi des 250 villes

### Le défi des 250 villes : le classement

Cliquez sur le nom du participant pour avoir plus de détails sur ses résultats, son algorithme et son logiciel !

| Pos | Participant                                              | Meilleur trojet | Temps de calcul | Temps de calcul corrigé | Configuration         | Logiciel | Langage       | Algorithme                                           | Date d'entrée |
|-----|----------------------------------------------------------|-----------------|-----------------|-------------------------|-----------------------|----------|---------------|------------------------------------------------------|---------------|
| 1   | <a href="#">Xavier Clurixt</a>                           | 11.8093         | 1200            | 360                     | PC K6/2               | moon     | Procal        | Personnel                                            | 21/12/2002    |
| 2   | <a href="#">François P-Fressencourt</a>                  | 11.8093         | 480             | 480                     | PC P-III              | moon     | C             | Topologies statistiques adaptatives                  | 23/11/2001    |
| 3   | <a href="#">Nuel Le Boucher</a>                          | 11.8093         | 1420            | 640                     | PC PII 450MHz         | moon     | Fortran/MSDev | Algo génétique modifié                               | 07/01/2004    |
| 4   | <a href="#">Alexandre Augerit</a>                        | 11.8093         | 580             | 773.14                  | Athlon 1.3 GHz        | trypen   | C++ (gcc 3.2) | Algo génétique + optimisations                       | 30/09/2000    |
| 5   | <a href="#">Konstantin Artouchine</a>                    | 11.8093         | 3600            | 1620                    | BI.ppc                | moon     | CLAME         | Relaxation lagrangienne et Lin-kamijian (methode)    | 03/08/2001    |
| 6   | <a href="#">Léon-Joseph</a>                              | 11.8093         | 10800           | 1080000                 | PII-500MHz            | moon     | C++/gcc 3.3   | Algorithme génétique                                 | 11/04/2003    |
| 7   | <a href="#">Tredé</a>                                    | 11.8093         | 123456          | 12345600                | PC2                   | moon     | ??            | Perso                                                | 31/08/2003    |
| 8   | <a href="#">Anceine Simon-Chautemps</a>                  | 11.9            | 960             | 432                     | PC PII-450MHz         | moon     | Delphi        | Personnel (Type Monte Carlo)                         | 29/03/2003    |
| 9   | <a href="#">Raphaël Condamin</a>                         | 11.93           | 1234            | 123400                  | PC1                   | moon     | C             | Algo génétique                                       | 26/02/2002    |
| 10  | <a href="#">Martin Lerang</a>                            | 12.0363         | 4500            | 6300                    | Pentium II 450 MHz    | moon     | C/Board       | Algo génétique + methode perso                       | 19/08/2001    |
| 11  | <a href="#">Ahmed Badredine Naama</a>                    | 12.0365         | 1234            | 123400                  | MHz                   | moon     | Delphi 5      | Méthode secrète                                      | 30/09/2000    |
| 12  | <a href="#">Aurelien Bonas et Jean-William Couhin</a>    | 12.07           | 18900           | 18900                   | PC 1GHz               | moon     | Maple 7       | Proches voisins + mutations + optimisation           | 06/04/2003    |
| 13  | <a href="#">Abbebe</a>                                   | 12.1265         | 1200            | 360                     | PC AMD 300 mhz        | moon     | Java          | 20gc synchrétique (exploration partielle de voisinag | 20/03/2001    |
| 14  | <a href="#">Colas</a>                                    | 12.1723         | 3600            | 4140                    | PC AMD Athlon 1.15GHz | moon     | Maple 8       | Colone de fournis                                    | 05/06/2003    |
| 15  | <a href="#">Thomas Grunberg</a>                          | 12.29           | 1200            | 0                       | KE-2-350MHz           | moon     | Java          | Méthode perso                                        | 27/11/2001    |
| 16  | <a href="#">Vincent Bour, Bruno Pionnet, Oscar Derys</a> | 12.528          | 70              | 119                     | PC athlon xp 1700     | moon     | C/DevC++      | Plus proches voisins et échanges de portions de tr   | 19/01/2003    |
| 17  | <a href="#">Jean-Marc Varenne</a>                        | 12.539          | 65              | 65                      | PC Duron 1 GHz        | moon     | Java          | PVC/PPV + Remunions de points + permutation de s     | 24/11/2002    |

**Le meilleur trojet trouvé actuellement**

Appartenance

11.8093

Trojet (reconstruit par Xavier Clurix)

Longueur =

**Les autres défis**

**Double spirale 3 (200 villes)**

**La quadrature du cercle (200 villes)**

**Le défi des 256 villes**

**Spirale 4 (256 villes)**

**rest783 (783 villes)**

**pr1002 (1002 villes)**

Faites participer votre algorithme !

**Sujets connexes sur ce site**

Introduction au problème du voyageur de commerce (PVC)

Les méthodes de résolution du PVC : Algorithme Génétique, Algorithme de la colonie de fourmis, Plus proches voisins, Algorithme 3 opt, Algorithme génétique (meilleur matériel)

**trypen**, moon logiciel le plus absolu pour résoudre le PVC et participer au défi des 250 villes ->

**DapperTop**, l'applet Java d'affichage d'un parcours du PVC

Application du PVC à la Casse de France de Hétéroclite 2004

Sites intéressants sur le PVC

**Les forums de Labo Algo**

Donnez votre avis et discutez de cette page sur le forum de Labo Algo !

Fig.C.2. Le classement du défi des 250 villes.

