Université de Montréal

Scatter Search pour le problème du voyageur de commerce

par

Cristian Oliva San Martin

Département d'informatique et recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures

en vue de l'obtention du grade de

Maître ès Sciences (M.Sc.) en informatique

Mars, 1999

Université de Montréal

Faculté des études supérieures

Ce mémoire intitulé :

Scatter Search pour le problème du voyageur de commerce

Présenté par :

Cristian Oliva San Martin

a été évalué par un jury composé des personnes suivantes :

Président :	Jean-Yves Potvin
Directeur :	Philippe Michelon
Co-Directeur :	Jacques Ferland
Membre :	Michel Gendreau

Mémoire accepté le : 99.04.20

À la memoire de mon

Père.

Sommaire

Le problème du voyageur de commerce est très étudié en optimisation combinatoire. Le contraste entre la facilité de le présenter et la complexité de le résoudre le rend très attrayant pour les chercheurs. D'ailleurs, très souvent, lorsqu'une nouvelle méthode est introduite pour traiter des problèmes combinatoires son efficacité est rapidement verifiée sur le problème du voyageur de commerce [PVC].

C'est ce qui a motivé cette étude dont l'objectif est d'abord de proposer une approche "Scatter Search" pour le [PVC]. Nous croyons que c'est la première méthode "Scatter Search" à être proposée pour ce problème. Également, nous présentons une nouvelle variante d'algorithme génétique obtenu en exploitant un opérateur de diversification emprunté de l'approche "Scatter Search" et qui engendre un processus de réinitialisation. Ceci permet une comparaison plus équitable des méthodes génétiques et des méthodes "Scatter Search". Ce sont là les contributions les plus importantes de ce mémoire.

Nous notons que les résultats numériques indiquent que l'efficacité supérieure de la méthode "Scatter Search" à celle de la variante d'algorithme génétique et à celle d'autres algorithmes heuristiques.

Table des matières

Table des figures

Liste des tableaux

Introduction

Un voyageur de commerce désire visiter exactement une fois et à coût minimum un ensemble de villes. Ce problème est connu sous le nom du problème du voyageur de commerce [PVC].

En adoptant la notation de la théorie des graphes, dénotons $G = (N, A, C)$ un réseau où N est l'ensemble de nœuds (villes), A l'ensemble des arêtes et C la matrice de coûts associée à A. Le problème du voyageur de commerce consiste à determiner un cycle hamiltonien de coût minimum. Rappelons qu'un cycle hamiltonien en est un qui contient chaque nœud du graphe exactement une fois.

Selon les particularités de C, nous pouvons distinguer deux cas du [PVC].

[Cas 1] : Si les coûts sont symétriques (i.e., $c_{ij} = c_{ji}$ pour toute arête $(i, j) \in A$) alors il s'agit d'un [PVC] symétrique.

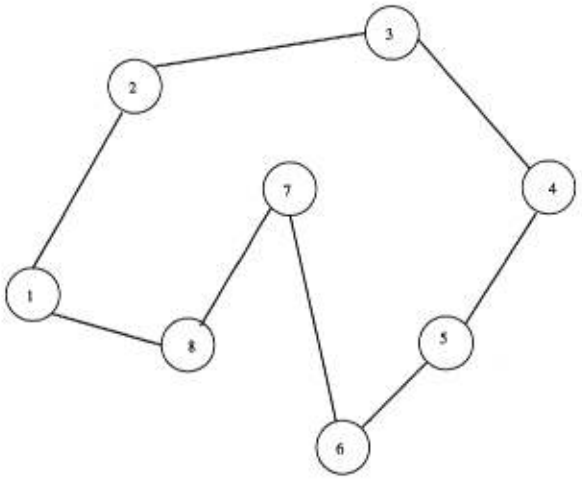

FIG. 1: Une solution réalisable pour un [PVC] symétrique

Une solution réalisable pour un problème de ce type avec 8 villes est illustrée à la FIG . 1. Dans ce cas, toute solution réalisable possède deux arêtes incidentes à chaque nœud.

[Cas 2] : Si les coûts sont asymétriques, alors il s'agit d'un [PVC] asymétrique. Notons que dans ce cas, A est un ensemble d'arcs orientés, et les coûts sont asymétriques puisque c_{ij} n'est pas nécessairement égal à c_{ji}.

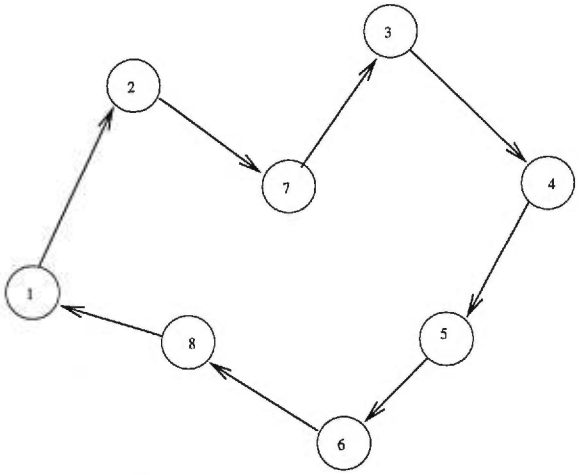

FIG. 2: Une solution réalisable pour un [PVC] asymétrique

En général, on fait l'hypothèse que la matrice C satisfait l'inégalité du triangle, c'est-à-dire que $c_{ij}+c_{jk} \geq c_{ik}\ \forall i, j, k \in N$. Une solution réalisable pour un tel problème est illustrée à la FIG . 2.

Pourquoi étudier le [PVC] ?

Bien que le [PVC] ait fait l'objet d'études depuis de nombreuses années, plusieurs raisons expliquent qu'il soit encore étudié aujourd'hui. Entre autres :

- Le [PVC] appartient à la classe des problèmes NP-complets ; il est ainsi peu probable qu'il existe un algorithme pour le résoudre en temps polynomial.
- Bien que théoriquement il soit possible d'énumérer toutes les solutions pour en déterminer la solution optimale, cette approche n'est pas réaliste en général. La nature du problème est, en effet, combinatoire.
- L'importance du [PVC] vient non seulement de la richesse de ses applications, mais aussi du fait qu'il appartienne à la gamme des problèmes d'optimisation combinatoire.
- Le [PVC] est très facile à décrire, mais il partage une inhérente complexité de résolution avec d'autres problèmes polynomialement équivalents (dans le sens où un algorithme polynomial pour un de ceux-là pourrait être adapté pour résoudre les autres).

Toutes ces raisons font du [PVC] un problème important en optimisation combinatoire.

C'est ce qui a motivé cette étude dont l'objectif est d'abord de proposer une approche "Scatter Search" pour le [PVC]. Nous croyons que c'est la première méthode "Scatter Search" à être proposée pour ce problème. Également, nous présentons une nouvelle variante d'algorithme génétique obtenu en exploitant un opérateur de diversification emprunté de l'approche "Scatter Search" et qui engendre un processus de réinitialisation. Ceci permet une comparaison plus équitable des méthodes génétiques et des méthodes "Scatter Search". Ce sont là les contributions les plus importantes de ce mémoire.

Nous notons que les résultats numériques indiquent que l'efficacité supérieure de la méthode "Scatter Search" à celle de la variante d'algorithme génétique et à celle

d'autres algorithmes heuristiques.

Plus précisément, trois chapitres composent ce mémoire. Le premier chapitre fait un rappel des formulations ainsi que des méthodes approximatives. Aux chapitres deux et trois, nous présentons une approche génétique pour le [PVC] et une approche "SCATTER SEARCH" ainsi que des comparaisons numériques des différentes méthodes analysées, respectivement. Des conclusions ainsi que des pistes pour des recherches futures sur le [PVC] complètent ce mémoire.

Chapitre 1

Formulations et méthodes de solution pour le [PVC]

1.1 Introduction

Ce chapitre fait une brève revue des méthodes heuristiques pour le [PVC], mais d'abord nous examinons quelques formulations du [PVC] comme un problème de programmation en nombres entiers [PLE].

1.2 Notation et terminologie

Voici la terminologie et la notation utilisées dans cette section. Dénotons par

Z : La valeur de la fonction économique ;

$N = \{1, 2, \cdots, n\}$ l'ensemble des villes à visiter ;

c_{ij} : Le coût de voyager de la ville i à la ville j.

S : L'ensemble des contraintes interdisant la formation de sous-tours.

$$x_{ij} = \begin{cases} 1 & \text{si le voyageur va de la ville } i \text{ à la ville } j. \\ 0 & \text{autrement.} \end{cases}$$

1.3 Formulations

Le problème du voyageur de commerce peut s'écrire comme un modèle de programmation linéaire en nombres entiers. Nous utilisons la formulation suivante que nous retrouvons dans Bodin[7].

$$min \qquad Z = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{1.1}$$

s.a

$$\sum_{i=1}^{n} x_{ij} = 1 \quad j \in N \tag{1.2}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad i \in N \tag{1.3}$$

$$\{x_{ij}\} \in S \tag{1.4}$$

$$x_{ij} \in \{0 \text{ ou } 1\} \quad i, j \in N \tag{1.5}$$

La fonction économique (1.1) décrit le coût du tour optimal. L'ensemble des contraintes (1.2) spécifie qu'il faut entrer dans chaque ville exactement une fois. L'ensemble (1.3) impose qu'il faut sortir de chaque ville exactement une fois. Les contraintes (1.4) interdisent la formation de sous-tours. Enfin, les restrictions (1.5) imposent que les variables soient binaires; plus précisément, $x_{ij} = 1$ si la ville j est visitée immédiatement après avoir visité la ville i, sinon $x_{ij} = 0$.

Plusieurs formulations de S existent dans la littérature. En voici quelques-unes qui nous paraissent plus intéressantes :

La formulation de Dantzig-Fulkerson-Johnson [16]

Soit $R \subset N$ un sous-ensemble de villes et $\bar{R} = N - R$. Si R est déconnecté de $\bar{R}$, alors $x_{ij} = 0$ pour $i \in R$ et $j \in \bar{R}$. Ainsi, il existe un sous-tour passant par les villes de R seulement si $\sum_{i \in R} \sum_{j \in \bar{R}} x_{ij} = 0$. Donc, un tel sous-tour est éliminé si :

$$\sum_{i \in R} \sum_{j \in \bar{R}} x_{ij} \geq 1.$$

Ainsi, un ensemble de contraintes d'élimination de sous-tours est comme suit :

$$S = \left\{ x_{ij} / \sum_{i \in R} \sum_{j \in \bar{R}} x_{ij} \geq 1 \qquad \forall R \subseteq N \right\} \tag{1.6}$$

Une formulation équivalente est :

$$S = \left\{ x_{ij} / \sum_{i \in R} \sum_{j \in R} x_{ij} \leq \mid R \mid -1 \qquad \forall R \subset N - \{1\} \right\} \tag{1.7}$$

Soulignons qu'il y a $O(2^n)$ sous-tour possibles et par conséquent autant de contraintes dans (1.6) et (1.7).

La formulation de Miller-Tucker-Zemlin [58]

La formulation de Miller-Tucker-Zemlin [58] modifie légèrement la définition du [PVC]. En effet, la ville de départ est artificiellement dédoublée en deux villes, la ville 0 et la ville n. Par conséquent, le tour débute en 0 et se termine en n. Ainsi, le tour qui était auparavant fermé est maintenant ouvert. Ainsi la spécification suivante :

$$S = \{ x_{ij} / Y_i - Y_j + n x_{ij} \leq n - 1, 2 \leq i \neq j \leq n \} \tag{1.8}$$

suppose qu'il existe des réels Y_i $(0 \leq Y_i \leq n)$ associés aux villes tels que les sous-tours soient éliminés. Le nombre de contraintes est de l'ordre $O(n^2)$.

La formulation de Gavish et Graves[7]

Une formulation différente est due à Gavish et Graves[7]. Ces derniers introduisent des variables de flot y_{ij}. Ils supposent d'une part que $(n-1)$ unités d'un produit sont disponibles à l'origine (nœud 1) et que, d'autre part, tous les nœuds, sauf l'origine, demandent une unité.

Formellement, le problème est constitué de la fonction économique (1.1), des contraintes (1.2), (1.3) et (1.5) auxquelles s'ajoutent les contraintes suivantes :

$$\sum_{j=1}^{n} y_{ij} - \sum_{j=1}^{n} y_{ji} = -1 \quad \text{i=2,..,n} \tag{1.9}$$

$$y_{ij} \leq U x_{ij} \quad \text{i,j=1,..,n} \tag{1.10}$$

$$y_{ij} \geq 0 \quad \text{i,j=1,..,n} \tag{1.11}$$

où $U \geq n-1$.

Un cas spécial très important est le problème du voyageur de commerce [PVC] symétrique, qui peut se formuler[51] comme suit :

$$Min\ Z = \sum_{i \in N} \sum_{j>i} c_{ij} x_{ij} \tag{1.12}$$

s.a.

$$\sum_{j<i} x_{ij} + \sum_{j>i} x_{ij} = 2, \quad i \in N \tag{1.13}$$

$$\sum_{i \in R} \sum_{j_j > i \in N-R} x_{ij} + \sum_{i \in N-R} \sum_{j_j > i \in R} x_{ij} \geq 2, \quad \forall R \subset N, R \neq \emptyset \tag{1.14}$$

$$x_{ij} \in \{0 \text{ ou } 1\} \quad i,j \in N, j > i \tag{1.15}$$

où les contraintes (1.14) représentent les contraintes d'élimination de sous-tours.

1.4 Méthodes heuristiques

Nous entreprenons la présentation de méthodes heuristiques. Plus particulièrement, quatre types de méthodes nous intéressent : celles de construction de solutions, d'amélioration de solutions, les méthodes mixtes et celles de métaheuristiques.

Brièvement, les procédures de construction génèrent un tour initial à partir de la matrice des distances. Les procédures d'amélioration de solution débutent avec un tour initial et trouvent, à travers une séquence d'échanges, un meilleur tour. Les procédures mixtes utilisent des méthodes de construction de solutions au départ et

poursuivent avec des méthodes d'amélioration. Enfin, les métaheuristiques débutent avec un ensemble de tours initial et trouvent un meilleur tour en utilisant des opérateurs spécifiques pour chaque problème.

1.4.1 Méthodes de construction de solutions

Les procédures de construction de tour s'appuient sur les trois composantes suivantes :

1. Le choix d'un sous-tour initial (point de départ).
2. Un critère de sélection de la prochaine ville à insérer.
3. Un critère d'insertion.

En ce qui concerne le choix d'un sous-tour de départ, la majorité des procédures de construction de tours choisit au hasard une ville comme point de départ. Évidemment, d'autres façons de choisir un sous-tour initial existent ; l'enveloppe convexe pour le [PVC] euclidien est un exemple [51].

Dans ce qui suit, nous décrivons quelques heuristiques.

Le plus proche voisin

Dans cette procédure, un tour est construit en considérant, à chaque itération, une décision qui est immédiatement profitable.

Étape 1 . Débuter avec n'importe quelle ville comme la ville initiale du tour.

Étape 2 . Trouver la ville la plus proche de la dernière insérée.

Étape 3 . Répéter l'étape 2 jusqu'à ce que toutes les villes aient été insérées.

Étape 4 . Faire le lien entre la première et la dernière ville insérées. Stop.

La faiblesse de cette procédure vient du fait que les premières arêtes insérées sont courtes alors que les dernières peuvent être très longues.

La complexité de cette procédure est $O(n^2)$. Toutefois, nous pouvons l'éxécuter n fois, en prenant successivement chaque ville comme la ville initiale, et la meilleure solution sera celle retenue. Cette stratégie est $O(n^3)$.

L'insertion arbitraire [68]

Étape 1 . Considérer n'importe quelle ville i comme la ville initiale du tour.

Étape 2 . Trouver la ville k telle que c_{ik} soit minimum et former le sous-tour (i, k).

Étape 3 . Sélectionner aléatoirement une ville k qui n'est pas dans le sous-tour.

Étape 4 . Déterminer l'arête $\{i, j\}$ du sous-tour qui minimise $c_{ik} + c_{kj} - c_{ij}$. Insérer ensuite la ville k entre les villes i et j.

Étape 5 . Aller à l'étape 3 jusqu'à ce qu'un tour soit formé.

L'insertion de l'enveloppe convexe [51]

Étape 1 . Former l'enveloppe convexe de l'ensemble des villes. Cette enveloppe convexe constitue le sous-tour initial.

Étape 2 . Pour chaque ville k qui n'est pas dans le sous-tour, déterminer la paire de villes $\{i, j\}$ du sous-tour telle que $c_{ik} + c_{kj} - c_{ij}$ soit minimum.

Étape 3 . De tous les triplets (i, k, j) trouvés à l'étape 2, déterminer le (i^*, k^*, j^*) tel que $(c_{i^*k^*} + c_{k^*j*})/c_{i^*j^*}$ soit minimum.

Étape 4 . Insérer la ville k^* entre les villes i^* et j^*.

Étape 5 . Aller à l'étape 2 jusqu'à ce qu'un tour soit formé.

Cette procédure a été développée pour résoudre le [PVC] euclidien.

L'insertion le plus loin

Cette procédure se résume ainsi :

Étape 1 . Considérer n'importe quelle ville i comme la ville initiale du tour.

Étape 2 . Sélectionner la ville k qui n'est pas dans le sous-tour et qui *maximise* $min\{c_{jk} : j$ appartient au sous-tour$\}$.

Étape 3 . Insérer la ville k de telle sorte que le coût d'insertion soit le plus petit possible ; c'est-à-dire entre i et j du sous-tour tel que $c_{ik}+c_{kj}-c_{ij}$ soit minimum.

Quoique cet algorithme choisisse la "pire" ville à insérer, il existe des raisons de penser que cette idée n'est pas mauvaise. Si nous insérons d'abord les villes les plus éloignées, nous établissons une ébauche pour la forme du tour. Cette forme est raffinée au fur et à mesure que les villes sont insérées dans le sous-tour. Pratiquement, les dernières villes insérées n'ont pas d'effet sur la longueur du tour.

Selon le critère utilisé dans les procédures d'insertion, leur complexité varie entre $O(n^2)$ et $O(n \log n)$.

L'algorithme des économies de Clark et Wright [14]

Dans cette procédure, l'approche est différente de celle des méthodes précédentes d'insertion. En effet, au départ nous supposons que chaque ville est visitée à partir d'un dépôt à l'aide d'un aller et retour. Par la suite, le tour est construit graduellement en reliant entre elles des paires de villes pour réduire la longueur totale en évitant des allers et retours. La procédure se résume ainsi :

Étape 1 . Considérer n'importe quelle ville i comme le dêpot central. Dénoter le dêpot central comme le nœud 1.

Étape 2 . Pour chaque paire de villes différentes $ij(i, j = 2, 3, \cdots, n)$, évaluer l' économie $s_{ij} = c_{1i} + c_{1j} - c_{ij}$

Étape 3 . Ordonner les économies en ordre décroissante.

Étape 4 . En parcourant la liste de haut en bas, former un tour en reliant directement entre elles les villes des paires sélectionnées.

1.4.2 Méthodes d'amélioration de solutions

Ces méthodes sont utilisées pour améliorer un tour obtenu par n'importe quelle procédure de construction de tour. Nous les classons en trois catégories.

Algorithmes r-opt [53]

Dans le cas général, r arêtes d'un tour réalisable sont remplacées par r arêtes qui ne sont pas incluses dans le tour, afin de générer un tour réalisable de coût moindre. Ainsi, le nombre de solutions candidates à chaque itération de l'algorithme est $O(n^r)$ puisqu'il existe $\binom{n}{r}$ manières de choisir r arêtes parmi n. En général, $r = 2$ ou $r = 3$.

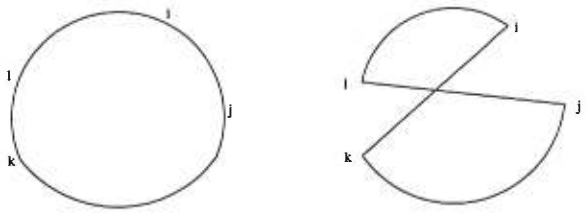

FIG. 1.1: Exemple de 2-opt. (a) Le tour initial. (b) Le tour après l'échange

La FIG. 1.1 illustre la procédure d'échange pour $r = 2$. Ici, les deux arêtes $\{i, j\}$ et $\{k, l\}$ sont remplacées par les arêtes $\{i, k\}$ et $\{j, l\}$.

Une méthode améliorée est presentée en [54]. Celle-ci modifie dynamiquement la valeur de r. Comparée à la méthode originale, cette méthode est considérablement plus compliquée à programmer. Toutefois, elle produit généralement des solutions plus proches de l'optimum.

L'algorithme Or-opt [62]

Or[62] propose une variante particulière du $3 - opt$ où seulement une partie des échanges sont considérés. Plus précisément, la procédure considère les échanges qui peuvent résulter de l'insertion de 1, 2 ou 3 villes actuellement adjacentes entre deux autres villes.

L'algorithme de Lin-Kernighan [47]

Ici aussi, il s'agit d'une variante du $3-opt.$ en résumé, un tour voisin est obtenu en remplaçant trois arêtes du tour courant à l'aide d'une série de modifications $2-opt$ complétées à l'aide d'une méthode heuristique gloutonne.

L'algorithme de Lin-Kernighan itéré [47]

Cet algorithme peut se résumer comme suit :

Étape 1 . Générer un tour T de façon aléatoire.

Étape 2 . Repéter les opérations suivantes pour un nombre spécifique M d'itérations :

2.1 . Générer un tour T' à l'aide d'une modification $4-opt$ choisie de façon aléatoire.

2.2 . Générer un tour T'' en appliquant l'algorithme de Lin-Kernighan sur T'.

2.3 . Si la longueur de T'' est plus petite ou égale que celle de T, alors $T = T''$.

Étape 3 . Retourner T.

1.4.3 Méthodes mixtes de construction et amélioration

Ces algorithmes construisent dans un premier temps un tour initial à l'aide d'une procédure de construction. Dans un deuxième, ils font appel à une ou plusieurs procédures d'amélioration pour déterminer une solution.

Dans la dernière décennie, plusieurs algorithmes mixtes furent développés. Mentionnons l'heuristique [CCAO] de Golden et Stewart [38] développée spécifiquement pour le [PVC] symétrique et qui exploite au mieux les propriétés de ceux-ci. Une autre [GENIUS] fut développée par Gendreau, Hertz et Laporte [27].

[CCAO][38]

Étape 1 . Définir un sous-tour initial comme l'enveloppe convexe des villes.

Étape 2 . Pour chaque ville k qui n'est pas dans le sous-tour, identifier les villes adjacentes i_k et j_k du sous-tour telles que $c_{i_k,k} + c_{k,j_k} - c_{i_k,j_k}$ soit minimum.

Étape 3 . Sélectionner la ville k^* qui maximise l'angle entre les arêtes (i_k, k) et (k, j_k) dans le sous-tour et l'insérer entre i_{k^*} et j_{k^*}.

Étape 4 . Aller à l'étape 2 jusqu'à ce qu'un tour soit construit.

Étape 5 . Appliquer une variante de la procédure 3-opt (due à Or[62]) au tour obtenu et Stop.

[GENIUS][27]

[GENIUS] comporte deux étapes : une étape d'insertion [GENI] et une étape de post-optimisation [US] qui déplace successivement les villes du tour pour les réinsérer à l'aide d'une règle d'insertion.

La principale caractéristique de [GENI] vient du fait qu'une réoptimisation locale est complétée à chaque fois qu'une nouvelle insertion est faite.

L'algorithme fut testé sur des problèmes symétriques et euclidiens de la littérature ainsi que des problèmes générés aléatoirement. Les tests indiquent que GENIUS est plus efficace que CCAO.

1.4.4 Métaheuristiques

Recuit simulé [49]

L'élaboration de la méthode du recuit simulé fut inspirée par un processus de refroidissement en métallurgie. En vue d'amener une barre métallique à un état solide d'énergie minimale, il faut la chauffer jusqu'à ce que son état soit liquide et ses particules soient uniformément distribuées. Pour éviter un minimum local, la température est graduellement réduite. À température élevée, tous les états possibles peuvent être atteints. En baissant la température le nombre d'états diminue. Finalement, le processus converge vers un état stable.

En optimisation combinatoire, l'objectif consiste à déterminer une solution de coût minimal, à partir d'une solution initiale et en appliquant des changements graduels aux solutions intermédiaires.

Dénotons par T, la température. Au départ, la valeur de T est élevée pour permettre d'accepter un plus grand nombre de mouvements. Celui-ci décroît avec T jusqu'à ce qu'il n'y ait plus de mouvement possible. Un minimum local est alors atteint.

Dans le cas du [PVC], pour une valeur de T, l'algorithme s'apparente à la procédure du $r-opt$ dans le sens où les solutions dans un voisinage d'une solution x sont analysées. Cependant, une solution y avec un coût plus grand que x peut être retenue avec une probabilité qui dépend de la température. Ceci réduit la possibilité de rester pris dans un minimum local.

Pour le problème en général $\min_{x \in S} f(x)$, l'algorithme se résume ainsi où **random(0,1)** est une fonction qui génère un nombre aléatoire entre 0 et 1 :

Étape 1 . Sélectionner une solution initiale $x \in S$ et fixer la température initiale à $T > 0$.

Étape 2 . Fixer le compteur de répétition $n = 0$.

Étape 3 . Générer une solution y voisine de x.

Étape 4 . Calculer $\delta = f(y) - f(x)$.

Étape 5 . Si $\delta < 0$ alors $x := y$ sinon si **random(0,1)** $< e^{\delta/T}$ alors $x := y$.

Étape 6 . Aller à l'étape 3 jusqu'à ce que $n = n(T)$.

Étape 7 . Réduire la température T.

Étape 8 . Aller à l'étape 2 jusqu'à ce qu'un critère soit satisfait.

Le recuit simulé fut appliqué à divers problèmes d'optimisation combinatoire, en particulier, au [PVC] par Bonomi et Lutton[8], Golden et Skiscim[37] et Nahar, Sahni et Shragowitz[59].

Recherche Tabou[28],[29],[30], [31],[33]

Dans les deux méthodes précédentes, les solutions voisines d'une solution x sont examinées. Dans la méthode du recuit simulé, la fonction économique peut se détériorer d'une itération à l'autre pour éviter de rester à un minimum local.

Le même principe s'applique dans la recherche tabou, mais afin d'éviter qu'une solution soit visitée plus qu'une fois, la recherche tabou insère les solutions dans une liste appelée "liste tabou". Notre présentation de la méthode pour le problème général $\min_{x \in S} f(x)$ s'inspire de celle dans [50].

Étape 1 . Considérer une solution initiale x de coût $f(x)$. Soit la liste tabou $LT := \emptyset$.

Étape 2 . Soit $N(x)$ un voisinage de x. Si $N(x) \setminus LT = \emptyset$, Stop. Autrement, identifier une solution de moindre coût y en $N(x) \setminus LT$ et fixer $x := y$. Mettre à jour LT et la meilleure solution connue.

Étape 3 . Si le nombre maximum d'itérations est atteint, alors arrêter, sinon aller à l'étape 2.

Plusieurs auteurs ont appliqué avec succès la recherche tabou au [PVC] , entre autres Malek[56] et Fiechter[21].

Réseaux de neurones [63]

Dans cette section nous nous inspirons fortement de la présentation de Potvin dans [63]. Nous résumons très brièvement trois types de modèles de réseaux de neuronnes pour le [PVC] : réseau de Hopfield-Tank, "elastic net" et réseau "self-organizing".

Réseau de Hopfield-Tank :

Ce modèle repose sur une fonction d'énergie qui diminue vers un minimun local. Pour l'appliquer au [PVC], il faut d'abord établir une correspondance entre la représentation graphique du [PVC] et celle d'un réseau de neuronnes. Ensuite il faut spécifier adéquadement la fonction d'énergie en utilisant des poids relatifs appropiés pour pénaliser les solutions associées à des tours non réalisables

et pour favoriser les solutions associées aux tours réalisables qui sont les plus courts.

Les résultats rapportés dans la littérature indiquent que la qualité des solutions obtenues avec ce genre de modèles n'est pas très bonne (voir [76], [43], [77], [63]). Dans son article [63], Potvin indique certaines modifications pour améliorer ces méthodes.

"Elastic net" :

L'algorithme *elastic net* est une procédure itérative. Au départ, un anneau de m points($m > n$) se trouve placé au centre des villes. Graduellement l'anneau est étiré jusqu'à ce qu'il soit suffisantement près de chaque ville. Deux forces gouvernent ce processus : une des deux forces minimise la longeur de l'anneau (élastique) alors que l'autre minimise la distance entre les villes et les points de l'anneau. Ces forces sont modifiées au fur et à mesure que la procédure évolue.

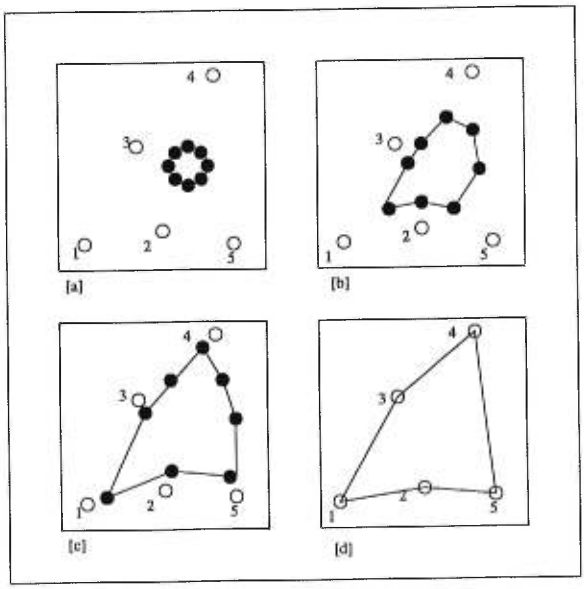

FIG. 1.2: L'évolution de l'élastique [a],[b], [c] et le tour final [d]

La FIG. 1.2 illustre comment l'élastique évolue dans le temps. Les points numérotés représentent les villes tandis que les autres points appartiennent à l'anneau. La FIG. 1.2[d] illustre la solution finale obtenue avec la méthode.

Réseau "self-organizing" :

Dans cette méthode, l'objectif est d'établir une correspondance entre la représentation des villes dans l'espace euclidien à deux dimensions et la représentation de la position des villes dans le tour dans l'espace à une dimension. Dans ce sens, il y a une relation entre cette approche et celle du "elastic net". Ces méthodes diffèrent dans la façon de mettre à jour les éléments de la représentation résultante. Des variantes de cette méthode sont proposées par Goldstein [39] et Angeniol [2].

Algorithmes génétiques [66]

Dans un contexte d'optimisation, un algorithme génétique est une procédure itérative visant à améliorer la valeur de la fonction économique du problème. Pour y arriver, une population de solutions candidates est maintenue. Pour modifier la population, de nouvelles solutions sont générées en sélectionnant des paires de solutions de la population courante et en leur applicant un opérateur de croisement permettant de retrouver certaines caractéristiques des solutions de la population. Nous revenons sur ces méthodes au chapitre 2 dans le contexte spécifique du [PVC].

Algorithmes de fourmis [18]

Les algorithmes de fourmis pour traiter des problèmes d'optimisation combinatoire comme le [PVC] s'inspirent du comportement d'une colonie de fourmis à la recherche de sources de nourriture. Lorsqu'une fourmi découvre une source de nourriture, elle indique à la colonie en laissant une trace de substance chimique, nommée *phéromone*, lors de son retour au nid. Les traces de cette substance guident les membres de la colonie vers des sources de nourriture. Pour le [PVC], l'analogie s'établit en considérant une fourmi (un agent) se déplacant d'une ville à l'autre dans le graphe associé au [PVC]. La fourmi détermine la ville où se déplacer à l'aide d'une fonction de probabilité qui dépend de la trace de phéromone sur les arêtes et de la longueur des arêtes. Au départ, m fourmis sont localisées aléatoirement aux villes. Guidées par la fonction

de probabilité, elles se déplacent de villes en villes en modifiant la trace de phéromone sur les arêtes utilisées (modification locale). Quand toutes les fourmis ont complété leur tour, la fourmi qui a identifié le tour le plus court modifie la trace de phéromone sur les arêtes du tour (modification globale) en ajoutant une quantité de substance inversement proportionnelle à la longueur du tour. Dorigo et Gambardella [18] ont appliqué ces méthodes avec succès.

Chapitre 2

UN ALGORITHME GÉNÉTIQUE POUR LE [PVC]

L'objectif principal de ce chapitre est de présenter un algorithme génétique pour le [PVC]. Pour cela, nous proposons une nouvelle idée sur l'ajustement de la mesure de performance pour la sélection des individus. Aussi, nous exploitons l'idée de diversification empruntée de l'approche "Scatter Search" pour notre algorithme génétique. Finalement, nous présentons des résultats numériques.

D'abord, nous faisons une brève revue de l'état de l'art des algorithmes génétiques pour le [PVC].

2.1 Algorithmes génétiques

Dans notre contexte, les algorithmes génétiques sont des métaheuristiques, lesquels maintiennent une population de solutions candidates : $P(t) =< x_1(t), x_2(t), \cdots, x_p(t) >$[1]

À chaque itération (appelée *génération*), la population actuelle est evaluée et sur la base de cette évaluation une nouvelle population de solutions candidates est construite. Un schéma général de la procédure est illustrée dans le Tableau 2.1.

[1] $P(t)$:Population à l'itération t ; $x_i(t)$: Solution candidate i à l'itération t.

Début

t :=0

Initialiser la population initiale $P(0)$

Évaluer les individus de $P(0)$

Répéter

t :=t+1

Sélectionner p individus de $P(t-1)$

Croiser les individus sélectionnés

Muter les individus générés

Évaluer les individus de la nouvelle population

Jusqu'à ce qu'un critère d'arrêt soit satisfait.

Arrêt

TAB. 2.1: Un algorithme génétique.

La population $P(0)$ est choisie, généralement, aléatoirement. Alternativement, la population initiale peut être générée en utilisant une heuristique. En tout cas, la population initiale contient une grande variété d'individus. Chaque individu de $P(0)$ est évalué. Par exemple, si nous cherchons à minimiser une fonction $f(x)$, il faut évaluer $f(x_1), f(x_2), \cdots, f(x_p)$

Les individus de la population $P(t)$ sont générés à partir de la population (précédente) $P(t-1)$. D'abord p individus de $P(t-1)$ sont sélectionnés selon un processus probabiliste favorisant qu'un individu soit sélectionné un nombre de fois proportionnel à sa qualité relative dans la population. Ainsi, les meilleurs individus (i.e., ceux avec la plus petite valeur de f) ont plus de chance d'être sélectionnés à plusieurs reprises.

Pour trouver d'autres points dans l'espace des solutions, quelques nouveaux individus (solutions) sont générés dans la nouvelle population à l'aide d'opérateurs de recombinaison génétiques. Le premier opérateur de recombinaison est appelé *croisement.* Deux individus parmi ceux sélectionnés s'échangent des portions de leurs

gènes.

Il existe un autre opérateur de recombinaison appelé *mutation* qui est appliqué aux individus générés par le croisement. L'idée de la mutation consiste à modifier légérement un individu en vue de diversifier la population.

2.2 Algorithmes génétiques pour le [PVC]

Cette section s'inspire étroitement de la présentation de Potvin dans [64]. Notons d'abord qu'il y a différentes façons de représenter une solution sous la forme d'un chromosome ce qui a amené différents auteurs à proposer divers types de croisement adaptés à ces représentations variées. En fait, il y a quatre types de représentations :

- *Représentation binaire* : Pour un problème asymétrique dans la représentation binaire, chaque arc $(i, j) \in A$ représente un gène. Un gène (i, j) prend la valeur 1 si et seulement si l'arc (i, j) existe dans le tour. Autrement le gène prend la valeur 0. L'ensemble des arcs représente un chromosome. Par exemple, le tour $T = (1, 2, 3, 4)$ est représenté sous le schéma du tableau 2.2.

arêtes	(1,2)	(1,3)	(1,4)	(2,1)	(2,3)	(2,4)	(3,1)	(3,2)	(3,4)	(4,1)	(4,2)	(4,3)
chromosome	1	0	0	0	1	0	0	0	1	1	0	0

TAB. 2.2: chromosome représentant le tour $T = (1, 2, 3, 4)$ dans la représentation binaire pour un [PVC] asymétrique.

Pour un problème symétrique, chaque arête $(i, j) \in A$ représente un gène. Le tour $T = (1, 2, 3, 4)$ est représenté sous le schéma du tableau 2.3.

Notons que la longueur du chromosome pour le [PVC] symétrique est plus petite que celle pour le [PVC] asymétrique.

Dans Forrest[24], un ingénieux opérateur de croisement est présenté pour cette représentation où les tours-fils sont tous réalisables.

arêtes	(1,2)	(1,3)	(1,4)	(2,3)	(2,4)	(3,4)
chromosome	1	0	1	1	0	1

TAB. 2.3: chromosome représentant le tour $T = (1, 2, 3, 4)$ dans la représentation binaire pour un [PVC] symétrique.

- *Représentation ordinale*[40] : Dans cette représentation, les chromosomes sont définis en référence à un tour de référence. Par exemple si $n = 6$, un tour de référence est 123456. Alors, un tour est codifié en considérant successivement chaque ville du tour. Pour une ville donnée, sa position dans le tour de référence courant est stockée à la position correspondante dans le chromosome résultant (représentation ordinale). Ensuite le tour de référence est mis à jour en éliminant la ville donnée.

  L'approche est illustrée à la figure 2.1 pour codifier $T = (1, 4, 5, 3, 6, 2)$.

Tour	Tour de reference courant	Representation ordinale
145362	123456	1
145362	23456	13
145362	2356	133
145362	236	1332
145362	26	13322
145362	2	133221

FIG. 2.1: Codification ordinale du tour $T = (145362)$

La première ville à être codifiée est la ville 1. La ville 1 se trouve à la position 1 dans le tour de référence. Ainsi, le premier gène du chromosome de la représentation ordinale est 1. Alors la ville 1 est éliminée du tour de référence. Ensuite, la ville 4 est codifiée. Elle se trouve à la position 3 du tour de référence courant. Ainsi, le gène suivant du chromosome est 3. Alors, la ville 4 est éliminée du tour de référence courant. La procédure continue ainsi jusqu'à ce que toutes les villes soient codifiées. Finalement, la représentation ordinale pour le tour $T = (1, 4, 5, 3, 6, 2)$ est 133221.

Cette ingénieuse représentation est dûe à Grefenstette[40]. Un opérateur de croisement classique peut être utilisé pour engendrer des tours-fils réalisables. Malheureusement, en général le tour-fils n'hérite que très peu de la structure de ses parents. Ainsi la recherche résultante se rapproche d'une recherche aléatoire.

- *Représentation du tour* : C'est la représentation la plus naturelle où le tour est représenté tel quel ; c'est-à-dire que le gène comporte la ville qui occupe la position correspondante dans le tour. Les opérateurs de croisement génèrent des tours-fils qui héritent de l'ordre relatif ou de la position absolue des villes de ses parents.
- *Représentation d'adjacence* : Cette représentation a été développée pour faciliter la manipulation des arêtes. Nous pouvons la décrire comme suit : la ville j occupe le gène i du chromosome s'il existe une arête de la ville i à la ville j dans le tour. Par exemple, le tour $T = (1, 4, 5, 3, 6, 2)$ est codifié comme suit 416532. La ville 4 se trouve à la position 1 dans le chromosome parce que l'arête $(1, 4)$ est dans le tour. La ville 5 se trouve à la position 4 dans le chromosome parce que l'arête $(4, 5)$ est dans le tour, etc.

Les opérateurs de croisement qui utilisent cette représentation génèrent des tours-fils qui héritent des arêtes de leurs parents.

Nous reviendrons plus loin pour discuter de divers croisements qui ont déjà été proposés. Pour le moment, nous reprenons les éléments de l'algorithme génétique de base pour les préciser dans le contexte du [PVC].

2.2.1 Génération de la population initiale

Pour générer chaque individu de la population initiale $P(0)$, nous utilisons l'heuristique d'*insertion le plus loin*. Tel qu'indiqué à la section 1.4.1, cette méthode est la procédure de construction la plus efficace (sans considérer l'enveloppe convexe). De plus, diverses solutions sont générées selon la ville retenue comme point de départ.

2.2.2 Mesure de la performance

Étant donné que l'objectif du problème est de minimiser la fonction économique $f(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$ mesurant le coût d'un tour, il s'ensuit que la performance d'une solution (un tour) est inversement proportionnelle à son coût $f(x)$. Ainsi, dans ce qui suit, la performance $g(x)$ d'un tour x est définie comme suit :

$$g(x) = \frac{1}{f(x)}$$

2.2.3 Sélection des p individus de $P(t-1)$

Rappelons que cette sélection a une influence directe sur la population $P(t)$ qui est générée à l'itération t à partir des individus de la population $P(t-1)$.

Nous développons une méthode de transformation linéaire pour ajuster la mesure de performance.

Soient :

- K le poids relatif de la probabilité de reproduction de l'individu le plus performant par rapport à l'individu le moins performant.
- g_b la mesure de performance de l'individu le plus performant.
- g_m la mesure de performance de l'individu le moins performant.
- g_i la mesure de performance de l'individu i.

Alors, l'ajustement que nous utilisons est le suivant :

$$g_i' = 1 + \frac{g_i - g_m}{g_b - g_m} \times (K - 1) \tag{2.1}$$

Ainsi, soient x_b et x_m les individus le plus performant et le moins performant, respectivement. Alors, $E[x_b] = K \times E[x_m]$. Le résultat de l'ajustement est illustré à la figure 2.2.

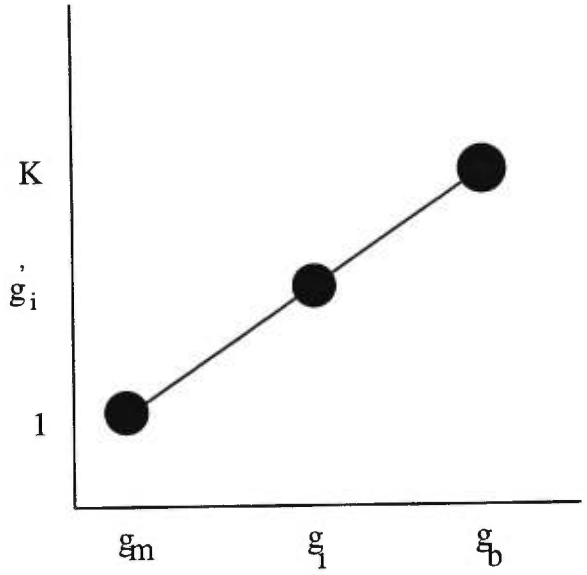

FIG. 2.2: Ajustement de la mesure de performance

Avec cette méthode nous fixons l'écart entre la performance de l'individu le plus performant et celle du moins performant à $K-1$, mais nous maintenons la performance relative entre ces deux individus à K.

De plus, pour réduire la variance de la reproduction, (voir Potvin[65]), nous ne choisissons qu'un nombre aléatoire entre 0 et 1, et nous y ajoutons 1 à $p-1$ reprises de façon à obtenir p nombres compris entre 0 et p. Donc pour sélectionner les individus, nous allons normaliser les mesures de performances $g_i^{'}$ pour les transformer en des valeurs $g_i^{''}$

$$g_i^{''} = \frac{p \times g_i^{'}}{\sum_{k=1}^{p} g_i^{'}} \tag{2.2}$$

de telle sorte que $\sum_{k=1}^{p} g_i^{''} = p$. Également nous associons à chaque individu i un intervalle de longeur $g_i^{''}$. Ainsi, à l'individu 1 est associé l'intervalle $[0,g_1^{''}]$, à l'individu 2 $(g_1^{''}, g_1^{''} + g_2^{''}]$, et ainsi de suite. Donc à l'individu i est associé l'intervalle $(\sum_{k=1}^{i-1} g_k^{''}, \sum_{k=1}^{i} g_k^{''}]$

Notons que l'approche précédente a les avantages tant de prévenir, au début de l'algorithme, la domination extraordinaire des individus très performants, que la

compétence (plus tard) parmi eux.

2.2.4 Opérateur de croisement

Tel que mentionné précédement, divers types de croisement ont été suggérés selon les types de représentation utilisés. Nous allons nous limiter à présenter des opérateurs de croisement pour la représentation du tour et la représentation d'adjacence. Notre présentation de ces opérateurs suit étroitement celle que nous retrouvons dans [64].

Représentation du tour

(i) CX("Cycle Crossover" dans Oliver et al.[61]) Cet opérateur considère les sous-ensembles de villes qui occupent le même sous-ensemble de positions dans les deux parents. Pour engendrer le premier tour-fils, les villes du premier parent dans ces positions sont reproduites. Les autres positions sont comblées par les villes qui les occupent dans le deuxième parent. De cette façon, chacune des positions du tour-fils est héritée d'un des parents. Pour le deuxième tour-fils, les rôles des parents sont inversés.

Dans la figure 2.3, le sous-ensemble de villes $\{1, 3, 7\}$ occupe le sous-ensemble de positions $\{1, 3, 7\}$ dans les deux parents. Ainsi, le premier tour-fils est crée en comblant les positions 1,3, et 7 avec les villes que nous retrouvons dans le Parent 1. Ensuite, les autres positions sont comblées avec les villes du Parent 2.

Parent 1	1	5	3	4	6	2	7	8
Parent 2	3	4	7	8	5	6	1	2
———	–	–	–	–	–	–	–	–
Fils 1	1	–	3	–	–	–	7	–
Fils 1	1	4	3	8	5	6	7	2

FIG. 2.3: L'opérateur CX

(ii) PMX("Partially-mapped crossover" dans Goldberg et Lingle [35]) Au départ, deux points sont sélectionnés aléatoirement. Afin de créer le premier tour-

fils, le segment entre les deux points de croisement dans le premier parent remplace le segment correspondant dans le deuxième parent. Pour éliminer les duplications, les remplacements inverses sont utilisés en dehors des points de croisement. Pour le deuxième tour-fils, les rôles des parents sont inversés.

Dans la figure 2.4, le premier tour-fils est crée en remplaçant le segment 2,8 et 5 dans le deuxième parent par le segment 3,4 et 6. Ici, les villes 6,4 et 3 sont duplicatées dans le tour-fils. Pour les éliminer, la ville 2 remplace la ville 3, la ville 8, la 4 et la ville 5, la 6.

Parent 1	1	7	|3	4	6|	5	2	8
Parent 2	6	4	|2	8	5|	3	1	7
———	–	–	–	–	–	–	–	–
fils 1	6	4	3	4	6	3	1	7
fils 1	5	8	3	4	6	2	1	7

FIG. 2.4: L'opérateur PMX

(iii) OX**("Order Crossover" dans Oliver et al.[61] et Goldberg[34])** Au départ, deux points de croisement sont sélectionnés aléatoirement. Afin de construire le premier tour-fils, le segment entre les deux points de croisement dans le premier parent est reproduit. Les autres positions sont ensuite comblées en considérant la séquence des villes dans le deuxième parent (en évitant de créer des duplications) en commençant après le deuxième point de croisement. Pour le deuxième tour-fils, les rôles des parents sont inversés.

Dans la figure 2.5 le segment 346 du Parent 1 est reproduit alors que les autres positions sont comblées en considérant la séquence suivante : 31764285.

Parent 1	1	7	|3	4	6|	5	2	8
Parent 2	6	4	|2	8	5|	3	1	7
———	–	–	–	–	–	–	–	–
fils 1	–	–	3	4	6	–	–	–
fils 1	8	5	3	4	6	1	7	2

FIG. 2.5: L'opérateur OX

(iv) MX**("Modified Crossover" dans Davis [17])** Cet opérateur de croisement est une extension de l'opérateur de croisement classique pour les problèmes de permutation. Une point de coupe est choisie aléatoirement. Un tour-fils est créé en conservant les villes du premier parent en position avant le point et complétant selon l'ordre que nous retrouvons dans le deuxième parent en éliminant les duplications. L'exemple suivant illustre ce croisement.

Parent 1	1	2|	5	6	4	3	7	8
Parent 2	6	4	2	8	5	3	1	7
———	–	–	–	–	–	–	–	–
fils 1	1	2	6	4	8	5	3	7

FIG. 2.6: L'opérateur MX

(v) OBX**("Order-based Crossover" dans Potvin [63])** Un sous-ensemble de villes est sélectionné dans le premier parent. Dans le tour-fils, ces villes figurent dans le même ordre que dans le premier parent, mais aux positions où elles se retrouvent dans le deuxième parent. Alors, les positions restantes sont occupées par les villes tel qu'elles se trouvent dans le deuxième parent. Voici un exemple :

Parent 1	1	2	5	6	4	3	7	8
Parent 2	6	4	2	8	5	3	1	7
———	–	–	–	–	–	–	–	–
fils 1	6	5	2	8	4	3	1	7

FIG. 2.7: L'opérateur OBX

Dans la figure 2.7, les villes 5, 4 et 3 sont premièrement choisies dans le premier parent. Elles doivent apparaître dans le tour-fils. Elles occupent les positions 2, 5 et 6 tel que dans le deuxième parent. Les positions restantes sont occupées par les villes tel que dans le deuxième parent.

(vi) PBX**("Position-based Crossover" dans Potvin [63])** Un sous-ensemble de positions est sélectionné dans le premier parent. Alors, les villes correspondantes dans le premier parent sont positionnées aux mêmes endroits dans le tour-fils. Les autres positions sont comblées avec les villes restantes, selon l'ordre dans le deuxième parent. Voici un exemple.

Parent 1	1	2	5	6	4	3	7	8
Parent 2	6	4	2	8	5	3	1	7
———	–	–	–	–	–	–	–	–
fils 1	6	2	5	8	4	3	1	7

FIG. 2.8: L'opérateur PBX

Représentation d'adjacence

(i) AE**("Alternate Edges Crossover" dans Grefenstette [41])** Une arête (i, j) est d'abord sélectionnée aléatoirement dans un des deux parents. Alors, le tour est étendu en sélectionnant l'arête (j, k) dans l'autre parent, et ainsi de suite. Quand une arête introduit un cycle, alors une nouvelle arête est sélectionnée aléatoirement.

La figure 2.9 illustre un tour-fils généré à partir des deux parents qui représentent les tours 13564287 et 14236578, respectivement, selon la représentation d'adjacence. Si l'arête $(1,4)$ est d'abord sélectionnée dans le parent 2, alors la ville 4 en position 1 dans le parent 2 est reproduite à la même position dans le tour-fils. Alors, les arêtes $(4,2)$ du parent 1 ; $(2,3)$ du parent 2 ; $(3,5)$ du parent 1 et $(5,7)$ du parent 2 sont successivement sélectionnées et insérées dans le tour-fils. Mais comme l'arête $(7,1)$ dans le parent 1 introduit un cycle, dans le tour-fils, alors une nouvelle arête menant à une ville qui n'est pas encore visitée est sélectionnée aléatoirement. Supossons que l'arête $(7,6)$ est choisie. Mais comme l'arête $(6,5)$ dans le parent 2 introduit aussi un cycle, alors nous choissons l'arête $(6,8)$ qui est la seule à ne pas introduire un cycle. Finalement, le tour est completé en ajoutant l'arête $(8,1)$.

Parent 1	3	8	5	2	6	4	1	7
Parent 2	4	3	6	2	7	5	8	1
———	–	–	–	–	–	–	–	–
fils 1	4	3	5	2	7	8	6	1

FIG. 2.9: L'opérateur AE

L'opérateur AE a tendance à introduire trop d'arêtes aléatoirement dans le tour-fils, surtout vers la fin. Puisque le tour-fils doit hériter le plus possible des arêtes de ses parents, le nombre de ces arêtes aléatoire doit être réduit le plus possible. L'opérateur introduit dans le prochaine section a pour objet de réduire ce comportement myope de AE.

(ii) ER("Edge Recombination Crossover" dans Potvin [63]) Cet opérateur de croisement utilise une structure de données appelée *carte d'arêtes*. La carte d'arêtes maintient une liste d'arêtes qui sont incidentes à chaque ville des tours parents et qui ne sont pas encore introduite dans le tour-fils. Ainsi, ces arêtes dites actives sont encore disponibles pour étendre le tour. La stratégie est d'étendre le tour en sélectionnant

l'arête incidente à la ville ayant un nombre minimum d'arêtes actives. Dans le cas où plusieurs villes peuvent être sélectionnées selon ce critère, une de ces villes est sélectionnée aléatoirement. La figure 2.10 illustre la carte d'arêtes initiale pour les tours 13564287 et 14236578 (représentation d'adjacence)

Ville 1 a les arêtes menant à :	3 4 7 8
Ville 2 a les arêtes menant à :	3 4 8
Ville 3 a les arêtes menant à :	1 2 5 6
Ville 4 a les arêtes menant à :	1 2 6
Ville 5 a les arêtes menant à :	3 6 7
Ville 6 a les arêtes menant à :	3 4 5
Ville 7 a les arêtes menant à :	1 5 8
Ville 8 a les arêtes menant à :	1 2 7

FIG. 2.10: La carte d'arêtes

Maintenant, nous illustrons l'opérateur ER à l'aide du tableau 2.4. Dans chaque carte d'arêtes , les villes particulièrement intéressantes sont sous-lignées (celles-ci sont adjacentes à la ville sélectionnée dans les tour-parents et n'ont pas encore été visitées). Supposons que la ville 1 est sélectionnée comme point de départ. Toutes les arêtes incidentes à la ville 1 doivent être éliminées de la carte d'arêtes initiale. De la ville 1, nous pouvons aller à la ville 3, 4, 7 ou 8. La ville 3 a trois arêtes actives tandis que les villes 4, 7 et 8 en ont deux. Ainsi, une ville parmi ces dernières est choisie aléatoirement. Supposons que la ville 8 est sélectionnée. De la ville 8, nous pouvons aller à 2 et 7. Comme indique la carte d'arêtes , la ville 2 a deux arêtes actives et la ville 7 en a seulement une; ainsi cette dernière est choisie. La seule ville où nous pouvons aller à partir de la ville 7 est 5. Alors celle-ci est sélectionnée. Maintenant, la carte d'arêtes nous permet d'aller aux villes 3 ou 6 qui ont deux arêtes actives. Supposons que nous choisissons d'aller à la ville 6. De 6 nous pouvons aller à la ville 3 ou 4. Nous sélectionnons la ville 4 aléatoirement. De cette ville nous ne pouvons aller qu'à la ville 2. Finalement, nous choisissons la ville 3.

Ville 1 est sélectionnée

Ville 2 a les arêtes adjancentes menant : 3 4 8

Ville 3 a les arêtes adjancentes menant : 2 5 6

Ville 4 a les arêtes adjancentes menant : 2 6

Ville 5 a les arêtes adjancentes menant : 3 6 7 ⇒

Ville 6 a les arêtes adjancentes menant : 3 4 5

Ville 7 a les arêtes adjancentes menant : 5 8

Ville 8 a les arêtes adjancentes menant : 2 7

Ville 8 est sélectionnée

Ville 2 a les arêtes adjancentes menant : 3 4

Ville 3 a les arêtes adjancentes menant : 2 5 6

Ville 4 a les arêtes adjancentes menant : 2 6

Ville 5 a les arêtes adjancentes menant : 3 6 7

Ville 6 a les arêtes adjancentes menant : 3 4 5

Ville 7 a les arêtes adjancentes menant : 5

Ville 7 est sélectionnée

Ville 2 a les arêtes adjancentes menant : 3 4

Ville 3 a les arêtes adjancentes menant : 2 5 6 ⇒

Ville 4 a les arêtes adjancentes menant : 2 6

Ville 5 a les arêtes adjancentes menant : 3 6

Ville 6 a les arêtes adjancentes menant : 3 4 5

Ville 5 est sélectionnée

Ville 2 a les arêtes adjancentes menant : 3 4

Ville 3 a les arêtes adjancentes menant : 2 6

Ville 4 a les arêtes adjancentes menant : 2 6

Ville 6 a les arêtes adjancentes menant : 3 4

Ville 6 est sélectionnée

Ville 2 a les arêtes adjancentes menant : 3 4 ⇒

Ville 3 a les arêtes adjancentes menant : 2

Ville 4 a les arêtes adjancentes menant : 2

Ville 4 est sélectionnée

Ville 2 a les arêtes adjancentes menant : 3

Ville 3 a les arêtes adjancentes menant : 2 6

TAB. 2.4: L'évolution de la carte d'arêtes

(iii) HX**("Heuristic Crossover" dans Grefenstette [41])** Dans l'article de Grefenstette[41], un autre type de croisement, appelé HX (croisement heuristique) est décrit. Ce dernier exploite l'information reliées aux distances entre villes.

L'algorithme se résumé comme suit :

[**Étape 1**]. Choisir aléatoirement une ville comme la courante du tour-fils.

[**Étape 2**]. Considérer les arêtes incidentes à la ville courante dans les parents et sélectionner l'arête la plus courte.

[**Étape 3**]. Si l'arête la plus courte introduite un sous-tour, alors choisir une autre ville qui n'introduise pas de sous-tour.

[**Étape 4**]. Aller à 2 jusqu'à ce qu'un tour soit formé avec toutes les villes.

Une autre variante utilise le opérateur de croisement suivant :

[**Étape 1**]. Choisir aléatoirement une ville comme la courante du tour-fils.

[**Étape 2**]. Considérer les arêtes incidentes à la ville courante dans les parents. Définir une distribution de probabilité sur ces arêtes basée sur la longueur des arêtes, de telle façon que la probabilité associée avec une arête incidente à une ville déjà visitée est zéro.

[**Étape 3**]. Sélectionner une arête selon cette distribution (si aucune des arêtes ne sont pas incidentes à une ville non visitée, étendre le tour avce une arête choisie aléatoirement et qui ne crée pas de sous-tour).

[**Étape 4**]. Répéter jusqu'à toutes les villes soient visitées.

Dans le [Étape 2], la probabilité associée à chaque arête est définie comme suit :

$$p_i = \frac{c_i}{\sum_{j=1}^{4} c_j} \tag{2.3}$$

où $c_i = \frac{1}{\text{(la longueur de l'arête i)}}$, pour $i = 1, 2, 3, 4$. Cette distribution favorise les arêtes les plus courtes.

2.2.5 Opérateur de mutation

Notre opérateur de mutation repose sur la procédure d'amélioration $r-opt$. Ainsi, un algorithme $2-opt$ est appliqué à chaque solution générée par le croisement. Si cet algorithme ne réussit pas à améliorer cette solution, alors un algorithme $3-opt$ est appliqué.

2.2.6 Critère d'arrêt

L'algorithme s'arrête lorsque la meilleure solution identifiée n'est plus améliorée pendant 50 générations consécutives.

2.3 Algorithme génétique avec diversification

De sorte à faire une meilleure comparaison entre les algorithmes génétiques et celui du "Scatter Search" du chapitre 4, nous empruntons l'idée d'une diversification de cette dernière approche pour engendrer un processus de réinitialisation. Indiquons d'abord où ce nouvel opérateur est introduit dans l'algorithme de base avant de décrire l'opérateur comme tel.

Pour obtenir cette variante de l'algorithme génétique avec diversification, nous modifions l'algorithme de base décrit à la section 2.1. Ainsi, lorsque la meilleure solution identifiée n'est pas améliorée pendant 30 itérations consécutives, l'opérateur de diversification est appliqué pour réinitialiser l'algorithme génétique avec une nouvelle population. Notons que le compteur original d'itérations consécutives sans amélioration est maintenu pour conserver le critère d'arrêt lorsque pendant 50 itérations consécutives la meilleure solution identifiée n'est pas améliorée.

Cet opérateur de diversification a été proposé par Glover[32]. Il utilise la meilleure solution pour engendrer une nouvelle population. Sans perte de généralité dénotons $T = (1, 2, \cdots, n)$ le meilleur tour disponible. Définissons le sous-tour de référence

$T(h:s)$ comme suit :

$$T(h:s) = (s, s+h, s+2h, \cdots, s+rh)$$

où s est un entier positif entre 1 et h; r est le plus grand entier non-négatif tel que $s+rh \leq n$. Alors, nous définons un tour $T(h)$,$1 \leq h \leq n$, de la diversification comme suit :

$$T(h) = (T(h:h), T(h:h-1), \cdots, T(h:1))$$

Illustrons la procédure avec l'exemple suivant. Soit $T = (1,2,3,4,5,6,7,8,9,10,11,12)$. Si h=3, alors $T(3:3) = (3,6,9,12)$, $T(3:2) = (2,5,8,11)$ et $T(3:1) = (1,4,7,10)$. Par conséquent le tour $T(3)$ est de la forme :

$$T(3) = (3,6,9,12,2,5,8,11,1,4,7,10)$$

2.4 Résultats

Dans cette section, nous présentons les résultats obtenus avec les l'algorithmes génétiques décrits dans les sections précédentes. Nous faisons trois ensembles de tests. Le premier a pour but de comparer les opérateurs de croisement OX, PMX, CX et HX. Comme OX s'est avéré le croisement le plus performant, nous l'utilisons dans le reste des tests. Dans le deuxième ensemble, nous analysons l'effet de la valeur de K. Le troisième ensemble permet de comparer la performance de OX avec celle de sa variante avec diversification.

2.4.1 Comparaison des croisements OX, PMX, CX et HX

Nous avons décidé de retenir un ensemble de croisements afin de les comparer. Notre objectif est d'en identifier un bon avec lequel nous allons poursuivre les tests

pour analyser l'effet de la transformation linéaire décrit à la section 2.2.3 et celle d'ajouter la diversification.

Nous comparons donc les quatre croisements suivants : OX, PMX, CX et HX. Dans [64], Potvin indique les résultats d'une comparaison de six croisements différents à l'aide d'un problème de 30 villes. Le croisement ER a été le plus performant suivi de prés par OX. Par contre CX et PMX ont été les moins performants. Dans notre analyse, nous voulons entre autres vérifier les performances relatives de OX, CX, PMX et HX. Pour les comparer, nous générons 20 problèmes aléatoirement en utilisant la procédure décrite dans l'article de Arthur et Frendewey[3].

Génération de problèmes [PVC] avec tour de coût optimal connu.

La méthode exploite le dual du [PA] associé au [PVC]. Au début , la méthode génère une permutation aléatoire des entiers ($[1, \cdots, n]$) dénotée comme l'ensemble ordonné $\{i_1, \cdots, i_n\}$. Alors, $x_{i_k i_{k+1}}$ est fixée à 1 et toutes les autres sont fixées à 0. En se référant aux conditions des écarts complementaires $(c_{ij} - u_i - v_j)x_{ij} = 0$ nous obtenons que les conditions suivantes doivent être satisfaites pour $k = 1, \cdots, n$: $u_{i_k} + v_{i_{k+1}} = c_{i_k i_{k+1}}$. Si le [PVC] est symétrique, alors il est aussi nécessaire que $u_{i_{k+1}} + v_{i_k} = c_{i_{k+1} i_k}$ pour $k = 1, \cdots, n$. Dû à la symétrie de la matrice de coûts, il s'ensuit que $u_{i_k} + v_{i_{k+1}} = u_{i_{k+1}} + v_{i_k}$ pour $k = 1, \cdots, n$. Par conséquent, nous fixons $v_i = u_i$, $i = 1, \cdots, n$. Ceci nous permet de calculer les coûts $c_{i_k i_{k+1}}$, $k = 1, \cdots, n$. Les autres coûts sont restreints a l'intervalle $[u_i + v_j, C*]$ où $C*$ est un paramètre.

Avec cette procédure, nous générons 10 problèmes de 40 villes et 10 autres de 80 villes. Nous fixons la taille de la population égale à la taille du probléme traité. Afin de bien comparer les opérateurs, nous éliminons l'opérateur de mutation. Le tableau 2.5 illustre le pourcentage d'erreur moyen par rapport à l'optimum pour les trois opérateurs. Le nombre d'itérations est fixé à 1000.

Les résultats du tableau 2.5 indique la supériorité de l'opérateur OX. En effet, les pourcentage d'erreur avec OX est de 2 à 3 fois moins grand que celui avec PMX

Opérateur	% d'erreur moyen (40-villes)	% d'erreur moyen (80-villes)
OX	8.54	8.97
PMX	15.90	25.70
HX	21.03	17.14
CX	33.80	47.92

TAB. 2.5: Pourcentage d'erreur moyen par rapport à l'optimum après 1000 générations.

et HX. De plus, les pourcentages d'erreur avec CX sont très élevés.

2.4.2 Valeur de K

Pour analyser l'effet de la transformation linéaire permettant d'amplifier l'importance relative des meilleures individus selon l'approche présenté à la section 2.2.3, nous considérons trois valeurs possibles pour K, soient 2,3 et 4. Nous utilisons 7 problèmes de la littérature[67] pour poursuivre l'analyse. La taille de la population utilisée est 40. L'heuristique génère un nombre de solutions de départ égal au nombre de villes. Parmi celles-ci nous retenons les 40 meilleures. Ce nombre de 40 a été choisi pour faciliter les comparaisons avec la méthode "Scatter Search" du prochain chapitre. Chaque problème est résolu à 10 reprises et l'algorithme s'arrête lorsqu'il n'existe pas d'amélioration pendant 50 itérations consécutives.

Dans le tableau 2.8, la première colonne indique le problème à résoudre, la deuxième colonne le nombre de fois que l'optimum est atteint pour les différentes valeurs de K, la troisième la longueur du pire tour trouvé et la dernière, le pourcentage d'erreur moyen par rapport à la longueur optimale.

Les résultats de tableau 2.8 semblent indiquer que la valeur $K = 2$ est la meilleure. Cette supériorité est davantage marquée pour les problèmes rat-99 et kroB-100 (avec respectivement 99 et 100 villes) où le nombre de fois où l'optimum est atteint et où le pourcentage d'erreur moyen sont nettement meilleurs avec $K = 2$.

2.4.3 Performance de OX et de OX avec diversification

Pour ces tests, nous utilisons 25 problèmes de la littérature[67]. La taille de la population est 40 comme dans la section précédente. La valeur de K est fixée à 2.

Dans la variante avec diversification, nous utilisons l'opérateur décrit dans la section 2.3 et nous générons les tours $T(1), T(2), \cdots, T(\varrho)$ où ϱ est égal au plus grand entier plus petit ou égal à $n/2$. Les 40 meilleurs de ces tours constituent la nouvelle population. Dans le tableau 2.9, nous comparons les performances de OX et de sa variante avec diversification relativement au nombre de fois où la valeur optimale est atteinte et relativement au pourcentage d'erreur de la longueur moyenne par rapport à la longueur optimale. La dernière colonne indique le temps moyen sur les dix expériences (TMS) en secondes requis par chaque variante pour atteindre la meilleure solution en utilisant une station de travail model 170 SUN/Ultra 1.

Les résultats du tableau 2.9 indiquent que la diversification permet d'améliorer les résultats tant au point de vue du nombre de fois que l'optimum est atteint qu'au point de vue du pourcentage d'erreur moyen. Évidement, ce gain de performance est en général obtenu aux dépens d'une augmentation du temps de résolution (TMS). Par contre, il y a trois exceptions (rat-195, d-198 et gil-262) où le temps de résolution diminue plutôt que d'augmenter, mais où l'amélioration est moins grande, si amélioration il y a. En somme, les résultats semblent indiquer qu'en général, le gain de performance justifie l'augmentation du temps de résolution.

2.4.4 Comparaison avec d'autres résultats déjà publiés

Il est très difficile de compléter une comparaison exacte avec d'autres résultats obtenus avec d'autres variantes d'algorithmes génétiques pour le problème du voyageur de commerce. En effet, les problèmes sont généralement différents de même que la qualité de la programmation de ces méthodes. Par contre nous allons tenter de situer la qualité de nos résultats obtenus avec le croisement OX par rapport à d'autres résultats déjà publiés et qui sont résumés dans [63].

Considérons le pourcentage d'erreur moyen comme critère de comparaison. Certaines résultats sont résumés dans le tableau 2.6.

Référence	nombre de villes	croisement	mutation	nombre de résolution	% moyen d'erreur
Suh and Gucht [71]	100	HX	2-opt	1	1.7
Suh and Gucht [71]	100	HX	—	1	25
Jog et al. [46]	100	HX	2-opt	10	2.6
Jog et al. [46]	100	HX	Or-opt	10	1.4
Ulder et al.[74]	48 à 666 (8 prob.)	OX	Lin-Kernighan	5/prob.	0.4
Braun [9]	431	OX	2-opt et Or-opt	?	0.5

TAB. 2.6: Pourcentage d'erreur moyen

Dans notre cas, les résultats dans le tableau 2.7 ont été obtenus avec le croisement OX et une mutation définie par une combinaison d'un 2-opt et d'un 3-opt. La deuxième variante (diver) est obtenu en ajoutant à cette variante l'opérateur de diversification présentée à la section 2.3. Les % de erreur moyen sont obtenus en faisant la moyenne de ceux associés aux 25 problèmes tests et qui se trouvent dans le tableau 2.9.

Variantes	nombre de villes	nombre de rsolutions	% d'erreur moyen
OX	99 à 264	10/prob.	1.04
diver	99 à 264	10/prob.	0.51

TAB. 2.7: Pourcentage d'erreur moyen

Ainsi, nos résultats obtenus avec OX semblent comparables et légérement meilleurs que ceux rapportés dans le tableau 2.6 lorsqu'une mutation de type $2-opt$ est utilisée. Ceci est consistant puisque notre opérateur de mutation est définie par une combinaison d'un $2-opt$ et d'un $3-opt$. Également, nos résultats obtenus avec

la variante incluant un opérateur de diversification semblent comparables avec ceux rapportés dans le tableau 2.6 lorsqu'un opérateur de mutation de type $Or - opt$ est utilisé. Ainsi, l'opérateur de diversification semble compenser, en quelque sorte, pour l'utilisation d'un opérateur de mutation définie par une combinaison d'un $2 - opt$ et d'un $3 - opt$ plutôt qu'un opérateur de mutation de type $Or - opt$.

TAB.2.8Résultats pour différentes valeurs de K

Problèmes	Nombre d'optimaux sur 10 exp.			Longeur du pire tour			% d'erreur moyen		
(tour opt.)	$K=2$	$K=3$	$K=4$	$K=2$	$K=3$	$K=4$	$K=2$	$K=3$	$K=4$
rat-99 (1211)	9	5	2	1213	1215	1218	0.16	0.74	1.32
KroA-100 (21282)	10	10	9	21282	21282	21305	0.00	0.00	0.11
KroB-100 (22141)	6	2	0	22199	22364	22234	1.04	3.00	2.94
KroC-100 (20749)	10	10	10	20749	20749	20749	0.00	0.00	0.00
KroD-100 (21294)	3	4	3	21389	21495	21389	1.55	1.56	1.51
KroE-100 (22068)	6	4	5	22121	22130	22106	0.80	1.14	0.86
rd-100 (7910)	9	9	6	7944	7944	7944	0.43	0.43	1.30

TAB.2.9 Comparaisons des deux variantes.

Problèmes	Nombre d'optimaux sur 10 exp.		% d'erreur moyen		Temps TMS sec.	
(tour opt.)	OX	diver	OX	diver	OX	diver
rat-99 (1211)	9	10	0.16	0.00	248.9	263.2
KroA-100 (21282)	10	10	0.00	0.00	276.8	276.8
KroB-100 (22141)	6	9	1.04	0.26	404.9	452.5
KroC-100 (20749)	10	10	0.00	0.00	99.9	99.9
KroD-100 (21294)	3	6	1.55	0.59	490.6	747.7
KroE-100 (22068)	6	7	0.80	0.37	280.4	428.9
rd-100 (7910)	9	10	0.43	0.00	386.2	412.4
eil-101 (629)	9	9	0.64	0.16	383.5	508.1
lin-105 (14379)	10	10	0.00	0.00	146.1	146.1
pr-107 (44303)	5	9	1.43	0.22	569.8	613.9
pr-124 (59030)	10	10	0.00	0.00	123.6	123.6
bier-127 (118282)	9	9	0.60	0.037	633.9	750.3
pr-136 (96772)	8	8	0.20	0.20	944.7	950.6
pr-144 (58537)	10	10	0.00	0.00	197.1	197.1
KroA-150 (26524)	5	7	0.17	0.01	781.9	1083.1
KroB-150 (26130)	4	4	0.11	0.11	739.7	741.0
pr-152 (73682)	4	6	1.11	0.74	601.4	2987.8
u-159 (42080)	1	9	6.76	0.75	821.6	2892.6
rat-195 (2323)	0	0	3.83	4.35	4109.1	3373.9
d-198 (15780)	1	2	0.22	0.20	5973.3	4994.9
KroA-200 (29368)	4	4	1.66	1.54	4029.6	4262.2
KroB-200 (29437)	0	4	0.65	0.23	4000.6	4678.3
pr-226 (80369)	5	7	0.25	0.15	4094.0	4250.2
gil-262 (2378)	2	2	2.31	1.68	9299.8	9109.2
pr-264 (49135)	3	5	2.10	1.06	13654.1	14202.3

Chapitre 3

UN ALGORITHME "SCATTER SEARCH" POUR LE [PVC]

Ce chapitre s'inspire de l'article de Glover[32]. L'objectif principal est de présenter une méthode Scatter Search pour le [PVC]. Pour cela, nous proposons quatre types de combinaison des solutions et nous les comparons à l'aide des mêmes problèmes [67] que dans le chapitre 2. Aussi, nous présentons diverses stratégies sur le nombre de solutions à combiner. Finalement, nous présentons des résultats numériques.

3.1 La méthode "SCATTER SEARCH"

La méthode "Scatter Search" est une méthode évolutive qui s'est récemment montré prometteuse pour résoudre des problèmes d'optimisation non-linéaire et d'optimisation combinatoire (Fleurent et al[22], Cung et al[15]). Commençons par introduire un schéma de la méthode que nous spécialisons par la suite à notre problème.

Le schéma comporte les procédures suivantes :

1. *Un générateur de diversification* : Cette procédure utilise une solution pour générer diverses autres solutions candidates.
2. *Une méthode d'amélioration* :Cette procédure permet d'améliorer la solution

actuelle.

3. *Une méthode pour la mise à jour de l'ensemble des solutions de référence* : Cette procédure permet de constituer et maintenir un ensemble de solutions de référence. Cet ensemble contient les b[1] meilleures solutions trouvées.
4. *Une méthode de génération de sous-ensembles* : Cette méthode génère des sous-ensembles des solutions de l'ensemble de référence. Ces sous-ensembles servent à créer des solutions combinées.
5. *Une méthode de combinaison de solutions* : Cette méthode a pour but de transformer un sous-ensemble de solutions donné en une ou plusieurs solutions combinées.

Il est intéressant de noter l'analogie avec certains éléments des algorithmes génétiques. Ainsi la méthode d'amélioration est liée à la mutation, l'ensemble de référence, à la population et la combinaison de solutions, au croisement dans l'algorithme génétique.

Voici maintenant un schéma qui est fréquemment utilisé par les chercheurs.

1. **Phase Initiale**
   (a) *Étape de génération de solutions de départ* : Créer une ou plusieurs solutions de départ. Ces solutions sont utilisées pour initialiser le reste de la méthode.
   (b) *Étape de génération par diversification* : Utiliser le générateur de diversification afin de générer diverses solutions à partir des solutions de départ.
   (c) *Étape d'amélioration* : Pour chaque solution produite à l'étape précédente, utiliser la méthode d'amélioration. Pendant cette étape, maintenir et mettre à jour l'ensemble de référence.
   (d) *Répéter* : Exécuter les étapes 1b et 1c jusqu'à obtenir l'ensemble de référence voulu.

[1] La valeur de b est typiquement petite, par exemple entre 20 et 40

2. **Phase Scatter Search**

   (a) *Génération de sous-ensembles* : Générer des sous-ensembles de l'ensemble de référence afin de créer des solutions combinées.

   (b) *Combinaison de solutions* : Pour chaque sous-ensemble X produit à l'étape précédente, utiliser une méthode de combinaison de solutions afin de produire un ensemble $C(X)$ comportant une ou plusieurs solutions combinées.

   (c) *Amélioration et mise à jour* : Pour chaque $C(X)$ produit à l'étape précédente, utiliser une méthode d'amélioration pour créer une ou plusieurs solutions améliorées. Maintenir et mettre à jour l'ensemble de référence.

   (d) *Répétition* : Éxécuter les étapes 2a à 2c jusqu'à ce qu'une condition d'arrêt soit satisfaite.

Un élément essentiel de la méthode Scatter Search est l'emphase qu'elle met sur l'utilisation de stratégies d'intensification et de diversification. Dans le schéma précédent, l'intensification est réalisée grâce à :

(i) l'utilisation répétée de la méthode d'amélioration ;

(ii) la mise à jour de l'ensemble de référence qui comporte les b meilleures solutions trouvées.

L'utilisation répétée de l'opérateur de diversification est responsable de la diversification.

Dans le schéma précédent, la diversification n'est présente que dans la phase initiale. Par contre, il est possible d'obtenir des variantes en introduisant la diversification au cours de la phase "Scatter Search".

3.2 Un algorithme Scatter Search pour le [PVC]

Dans cette section, nous présentons un algorithme Scatter Search pour le [PVC] qui est résumé dans le tableau 3.1

Début

Initialiser l'ensemble de solutions de référence P

Évaluer P

$t \leftarrow 0$

Répéter

•Générer des $sous - ensembles$ X de P

•Combiner les solutions des sous-ensembles X pour générer les sous-ensembles $C(X)$

•Appliquer la méthode d'amélioration à chaque solution générée et

mettre à jour l'ensemble de référence.

•Si l'ensemble de référence comporte une solution meilleure que la meilleure

solution trouvée, cette solution devient la meilleure solution trouvée, et $t \leftarrow 0$

Sinon $t \leftarrow t + 1$

•Si la meilleure solution courante n'a pas été améliorée

au cours des η dernières itérations, alors appliquer **diversification**

Jusqu'à ce qu'un critère d'arrêt soit satisfait

Arrêt

TAB. 3.1: Un schéma de la méthode Scatter Search

3.2.1 Génération de l'ensemble de solutions de référence

Comme dans le cas de l'algorithme génétique, pour générer chaque individu de l'ensemble de référence P, nous utilisons l'heuristique *insertion le plus loin*. Diverses solutions sont générées suivant la ville choisie comme point de départ. Le nombre de solutions que nous générons ainsi est donc égal au nombre de villes du [PVC] à traiter. Ainsi, nous évitons la duplication des solutions.

3.2.2 La taille de l'ensemble de référence et des sous-ensembles

La taille de l'ensemble de référence P dépend en général de la stratégie de combinaison. Nous avons opté pour utiliser toutes les solutions dans le processus de combinaison. Dans ce cas, Glover[32] suggère d'utiliser une population de référence de taille 40. Nous fixons la taille à 40 puisque notre objectif est d'analyser différents opérateurs de combinaison et non l'effet de la taille de l'ensemble de référence.

Nous analysons l'effet de la taille des sous-ensembles X. Nous considérons trois tailles différentes, 2, 3, et 4 de même qu'une stratégie où la taille varie aléatoirement entre 2 et 4.

Chaque sous-ensemble X est généré aléatoirement. Or comme nous avons choisi d'utiliser toutes les solutions de la population de référence P dans le processus de combinaison, il faut donc que la taille P soit un multiple entier de la taille de X. Ainsi, si la taille X est égale à 2 ou 4, alors une taille de 40 pour P est acceptable. Par contre, lorsque la taille de X est égale à 3, nous utilisons une taille de 42 pour P. Finalement, pour la stratégie où la taille de X varie aléatoirement entre 2 et 4, la taille de P doit être ajustée en conséquence en utilisant un opérateur de diversification pour ajouter des solutions ou un opérateur d'élimination pour en enlever. Ainsi, en utilisant l'opérateur de diversification décrit à la section 2.3, nous générons $T(2)$ et $T(3)$ que nous ajoutons à la population de référence. L'opérateur d'élimination élimine les moins bonnes solutions en trop.

3.2.3 Évaluation des solutions

Nous utilisons la fonction économique $f(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$ pour chaque solution.

3.2.4 Génération des sous-ensembles à partir de l'ensemble de référence

Chaque sous-ensemble est généré aléatoirement et est de taille τ. Cette taille indique le nombre de solutions à combiner. Nous analysons les effets de tailles égales à 2,3 et 4, et d'une procédure qui modifie dynamiquement cette taille entre les valeurs 2 et 4.

3.2.5 Combinaisons des solutions des sous-ensembles

Les quatres opérateurs qui suivent sont inspirés de l'opérateur de croisement HX proposé par Grefenstette[41] pour l'algorithme génétique avec la représentation d'adjacence. Considérons un sous-ensemble X. Les opérateurs génèrent un nombre de solutions égal à la taille de X ; c'est-à-dire que la taille de $C(X)$ est égal à celle de X. À titre d'exemple, considérons que le sous-ensemble X comporte les 4 solutions illustrées dans le tableau 3.2.

Le processus pour amorcer la combinaison est le même pour les quatre opérateurs. Deux points de combinaison sont sélectionnés aléatoirement. Les gènes du tour-parent i sont utilisés pour amorcer la génération de tour-fils i. Dans l'exemple du tableau 3.2, les deux points sont les positions 3 et 5. Ainsi la génération du tour-fils 1 est amorcé avec (-,-,3,8,7,-,-,-), le tour-fils 2 est amorcé avec (-,-,7,4,5,-,-,-), le tour-fils 3, avec (-,-,3,8,1,-,-,-), et le tour-fils 4, avec (-,-,6,8,1,-,-,-). Le reste de la génération varie selon l'opérateur.

OXHX-BO-Uniforme . La génération de chaque tour-fils est un processus itératif. Dénotons la dernière ville introduite dans le tour-fils comme la ville courante. Identifions l'arête la plus fréquente dans les tours-parents qui comportent la ville courante et une autre ville qui n'est pas déjà intégrée dans le tour-fils. S'il existe plusieurs arêtes ayant la plus grande fréquence, nous en choisissons une aléatoirement selon une distribution uniforme. La ville à l'autre extrémité de

l'arête devient la prochaine ville introduite dans le tour-fils. Si toutes les villes à l'autre extrémité de la ville courante sont déjà dans le tour-fils, nous choisissons aléatoirement la prochaine ville introduite. La procédure se poursuit jusqu'à ce que toutes les villes soient introduites.

Solution 1	2	6	3	8	7	1	5	4		Solution 1	2	6	3	8	7	1	5	4
Solution 2	3	6	7	4	5	2	1	8		Solution 2	3	6	7	4	5	2	1	8
Solution 3	6	4	3	8	1	2	5	7	⇒	Solution 3	6	4	3	8	1	2	5	7
Solution 4	4	5	6	8	1	3	2	7		Solution 4	4	5	6	8	1	3	2	7
Fils			3	8	7					Fils			3	8	7	4		

Solution 1	2	6	3	8	7	1	5	4		Solution 1	2	6	3	8	7	1	5	4
Solution 2	3	6	7	4	5	2	1	8		Solution 2	3	6	7	4	5	2	1	8
Solution 3	6	4	3	8	1	2	5	7	⇒	Solution 3	6	4	3	8	1	2	5	7
Solution 4	4	5	6	8	1	3	2	7		Solution 4	4	5	6	8	1	3	2	7
Fils			3	8	7	4	5			Fils			3	8	7	4	5	2

Solution 1	2	6	3	8	7	1	5	4		Solution 1	2	6	3	8	7	1	5	4
Solution 2	3	6	7	4	5	2	1	8		Solution 2	3	6	7	4	5	2	1	8
Solution 3	6	4	3	8	1	2	5	7	⇒	Solution 3	6	4	3	8	1	2	5	7
Solution 4	4	5	6	8	1	3	2	7		Solution 4	4	5	6	8	1	3	2	7
Fils	1		3	8	7	4	5	2		Fils	1	6	3	8	7	4	5	2

TAB. 3.2: Une combinaison OX-BO-Uniforme

Considérons l'exemple illustré dans le tableau 3.2. La configuration du tour-fils est amorcé avec (-,-,3,8,7,-,-,-). La ville courante est alors 7. Ensuite, nous notons que les arêtes (7,4) et (7,6) apparaissent deux fois dans les tours-parent et (7,5) et (7,2), une fois. Nous choisissons aléatoirement entre (7,4) et (7,6). Supposons

que (7,4) est sélectionné. Alors 4 devient la ville courante et le processus est répété.

OXHX-BO-DA . Cet opérateur diffère du précédent uniquement dans la sélection de l'arête lorsque plusieurs ont la même fréquence la plus élevée. La distribution de probabilité de sélection dépend du coût des arêtes. Ainsi, si nous devons choisir entre $(i_1, j_1), (i_2, j_2), \cdots, (i_\varrho, j_\varrho)$, alors la probabilité associée à chaque arête (i_μ, j_μ) est comme suit :

$$\frac{\frac{1}{c_{i_\mu j_\mu}}}{\sum_{\tau=1}^{\varrho} \frac{1}{c_{i_\mu j_\mu}}}$$

afin de favoriser les arêtes les plus courtes.

OXHX-BO-PC . Cet opérateur diffère également de OXHX-OB-Uniforme uniquement dans la sélection de l'arête lorsque plusieurs ont la même fréquence la plus élevée. Alors, nous choisissons la plus courte parmi celle-ci (en choisissant aléatoirement parmi les plus courtes s'il y en a plusieurs).

OXHX-PC . Cet opérateur diffère de OXHX-OB-Uniforme par la sélection de l'arête identifiant la prochaine ville à introduire. Ainsi, parmi les arêtes dans les parents qui comportent la ville courante et une autre ville qui n'est pas déjà intégrée dans le tour-fils, nous choisissons la plus courte (choisissant aléatoirement parmi les plus courtes s'il y en a plusieurs).

3.2.6 La méthode d'amélioration

De la même façon que pour les algorithmes génétiques, nous utilisons une méthode d'amélioration qui repose sur la procédure $r-opt$. Plus spécifiquement, un algorithme $2-opt$ est appliqué à chaque solution générée. Si cet algorithme ne peut améliorer la solution, alors un algorithme $3-opt$ est appliqué.

3.2.7 Mise à jour de l'ensemble de référence

Pour faciliter la mise à jour de l'ensemble de référence, nous utilisons un monceau où la racine contient la pire solution.

3.2.8 Le critère d'arrêt

Suite à des test préliminaires, comme critère d'arrêt nous fixons à 50 le nombre d'itérations consécutives sans amélioration de la meilleure solution trouvée (c'est-à-dire que l'algorithme s'arrête lorsque t atteint la valeur 51).

3.2.9 Diversification

Nous utilisons un opérateur de diversification lorsqu'il n'existe pas d'amélioration depuis 30 itérations (c'est-à-dire lorsque t atteint la valeur 30). Dans notre algorithme, nous utilisons l'opérateur proposé par Glover[32] qui a été décrit à la section 2.3.

3.3 Résultats

Dans cette section, nous présentons les résultats obtenus avec les différentes variantes de l'approche "Scatter Search" décrits dans la section précédente. Les tableaux 3.4, 3.5, 3.6 et 3.7 présentent les résultats pour les valeurs de τ égale à 2,3,4 et entre 2 et 4, respectivement, pour tous les types de combinaison des solutions. La première ligne indique l'opérateur de combinaison utilisé. Les colonnes dénotées par **No-Op** indiquent le nombre de fois que la solution trouvée est optimal avec la stratégie de combinaison des solutions utilisée. Les colonnes dénotées par **% E-M** indiquent le pourcentage d'erreur moyen par rapport à l'optimum.

Chaque problème est résolu à 10 reprises et l'algorithme s'arrête lorsqu'il n'existe pas d'amélioration pendant 50 itérations. La diversification est complétée lorsqu'il n'y a pas d'amélioration pendant 30 itérations consecutives.

3.3.1 Comparaisons de ces méthodes et des algorithmes génétiques

Les résultats numériques dans les tableaux 3.4, 3.5, 3.6 et 3.7 comparent les résultats obtenus avec les différentes stratégies.

Les résultats numériques dans les tableaux 3.4, 3.5, 3.6 et 3.7 permettent de comparer la qualité des résultats obtenus avec divers stratégies du point de vue du nombre de fois où l'optimum est atteint (No-op) et du pourcentage d'erreur moyen ((τ) des sous-ensembles utilisés. Or, les résultats des tableaux 3.4, 3.5, 3.6 et 3.7, ne permettent pas d'identifier une stratégie clairement dominante.

Les temps d'exécution pour les diverses stratégies sont résumés dans les tableaux 3.8, 3.9, 3.10 et 3.11. Ces résultats permettent d'illustrer en quel que sort le nombre τ de parents à combiner, en général la strategie OXHX-OB-Uniforme prend moins de temps. Ceci est clairement verifié pour les 13 dernières problèmes dans les tableaux.

Des plus, ces résultats indiquent qu'en général le temps de résolution augmente avec le nombre τ de parents à combiner. En conséquence, il ne semble pas y avoir avantage à utiliser plus de 2 parents.

Si nous comparons ces résultats avec ceux obtenus avec OX-diversifié tel que résumés dans le tableau 2.9, nous voyons qu'en général les résultats obtenus avec les stratégies de *scatter search* sont d'une qualité un peu supérieure à celle de ceux obtenus avec OX-diversifié. En ce qui concerne les temps d'exécution, en considerant la meilleure stratégie identifiée au paragraph précédent, nous constatons que celle-ci requiert moins de temps que la méthode OX-diversifié. Ceci s'explique par le fait que le nombre d'itérations completés avec la méthode $OXHX - OB$-Uniforme est moins grand qu'avec la méthode OX-diversifié.

Finalement, il est intéressant de noter que lorsque $\tau = 2$, les diverses stratégies correspondent à des variantes de type génétique puisque deux parents sont utilisés pour faire les combinaison. Les résultats des tableaux 3.4, 3.5, 3.6 et 3.7 illustrent qu'en général les résultats sont semblables à ceux obtenus avec un plus grand nombre de parents.

3.3.2 Performance des algorithmes génétiques, du scatter search et d'autres méthodes heuristiques

Il est difficile de situer la performance des algorithmes génétiques et du scatter search par rapport aux autres heuristiques pour le problème du voyageur de commerce. En effet, les problèmes sont généralement différents de même que la qualité de la programmation de ces méthodes. Par contre, nous allons tenter de situer la qualité du pourcentage d'erreur obtenu avec le croisement OX avec diversification et avec scatter search avec $\tau = 2$ en utilisant la procédure $OXHX - OB - Uniforme$ par rapport à d'autres résultats déjà publiés. Plus précisement, nous utilisons les résultats présentés dans [51]. Ils sont résumés dans le tableau 3.3 où nous avons ajoutés nos résultats obtenus avec l'algorithme génétique avec diversification et avec la variante du scatter search mentionnée plus haut.

Dans le tableau 3.3, le premier, le deuxième et le troisième groupe de méthodes correspondent respectivement à des méthodes de construction de solutions, à des méthodes d'amélioration de solutions, CCAO et des métaheuristiques.

Pour ces problèmes l'algorithme scatter search présente les meilleurs résultats. Également notre algorithme génétique avec diversification donne des meilleurs résultats que les autres méthodes heuristiques présentées dans le tableau 3.3.

Méthode	Problème de 100 villes				
	Kro-A	Kro-B	Kro-C	Kro-D	Kro-E
(tour optimal)	21282	22141	20749	21294	22068
Le plus proche voisin	16.67	16.88	13.35	16.51	13.27
L'insertion arbitraire	4.46	3.50	3.28	2.90	5.03
Enveloppe convexe	3.64	2.52	2.54	2.35	3.45
L'insertion le plus loin	5.14	6.97	3.17	1.99	7.42
Clark et Wright	1.62	3.77	6.37	3.41	2.85
2-opt (La meilleure de 25 exécutions)	1.11	3.05	0.51	3.27	3.24
3-opt (une fois)	7.82	2.87	3.30	1.15	1.40
CCAO	0.00	0.97	0.50	0.97	2.54
Algorithme génétique avec diver	0.00	0.26	0.00	0.59	0.37
Scatter Search $\tau = 2$ OXHx-OB-Uniforme	0.00	0.00	0.00	0.00	0.00

TAB. 3.3: Comparaisons des pourcentages d'erreur moyen avec des algorithmes génétiques, avec la méthode du scatter search et avec d'autres méthodes heuristiques.

TAB. 3.4 **Comparaisons des différentes stratégies de cmbinaisons des solutions avec $\tau = 2$**

Problèmes	OXHX-OB-Uniforme		OXHX-OB-DA		OXHX-OB-PC		OXHX-PC	
(tour opt.)	No-Op	% E-M	No-Op	% E-M	No-Op	% E-M	No-Op	% E-M
rat-99 (1211)	10	0.00	10	0.00	10	0.00	10	0.00
KroA-100 (21282)	10	0.00	10	0.00	10	0.00	10	0.00
KroB-100 (22141)	10	0.00	10	0.00	10	0.00	10	0.00
KroC-100 (20749)	10	0.00	10	0.00	10	0.00	10	0.00
KroD-100 (21294)	10	0.00	10	0.00	10	0.00	10	0.00
KroE-100 (22068)	8	0.03	8	0.03	8	0.03	8	0.03
rd-100 (7910)	10	0.00	10	0.00	10	0.00	10	0.00
eil-101 (629)	9	0.48	9	0.48	9	0.48	9	0.48
lin-105 (14379)	10	0.00	10	0.00	10	0.00	10	0.00
pr-107 (44303)	10	0.00	10	0.00	10	0.00	10	0.00
pr-124 (59030)	10	0.00	10	0.00	10	0.00	10	0.00
bier-127 (118282)	10	0.00	10	0.00	10	0.00	10	0.00
pr-136 (96772)	10	0.00	10	0.00	10	0.00	10	0.00
pr-144 (58537)	10	0.00	10	0.00	10	0.00	10	0.00
KroA-150 (26524)	8	0.0007	8	0.0007	8	0.0007	8	0.0007
KroB-150 (26130)	7	0.002	7	0.002	7	0.002	7	0.002
pr-152 (73682)	10	0.00	10	0.00	10	0.00	10	0.00
u-159 (42080)	10	0.00	10	0.00	10	0.00	10	0.00
rat-195 (2323)	2	0.22	2	0.22	2	0.22	2	0.22
d-198 (15780)	7	0.005	7	0.005	7	0.005	7	0.005
KroA-200 (29368)	10	0.00	10	0.00	10	0.00	10	0.00
KroB-200 (29437)	9	0.0004	9	0.0004	9	0.0004	9	0.0004
pr-226 (80369)	10	0.00	10	0.00	10	0.00	10	0.00
gil-262 (2378)	0	0.27	0	0.27	0	0.27	1	0.23
pr-264 (49135)	10	0.00	10	0.00	10	0.00	10	0.00

TAB. 3.5 **Comparaisons des différentes stratégies de combinaisons des solutions avec τ =3**

Problèmes	OXHX-OB-Uniforme		OXHX-OB-DA		OXHX-OB-PC		OXHX-PC	
(tour opt.)	No-Op	% E-M	No-Op	% E-M	No-Op	% E-M	No-Op	% E-M
rat-99 (1211)	10	0.00	10	0.00	10	0.00	10	0.00
KroA-100 (21282)	10	0.00	10	0.00	10	0.00	10	0.00
KroB-100 (22141)	10	0.00	10	0.00	10	0.00	10	0.00
KroC-100 (20749)	10	0.00	10	0.00	10	0.00	10	0.00
KroD-100 (21294)	10	0.00	10	0.00	10	0.00	10	0.00
KroE-100 (22068)	9	0.017	10	0.00	10	0.00	10	0.00
rd-100 (7910)	10	0.00	10	0.00	10	0.00	10	0.00
eil-101 (629)	10	0.00	10	0.00	10	0.00	10	0.00
lin-105 (14379)	10	0.00	10	0.00	10	0.00	10	0.00
pr-107 (44303)	10	0.00	10	0.00	10	0.00	10	0.00
pr-124 (59030)	10	0.00	10	0.00	10	0.00	10	0.00
bier-127 (118282)	10	0.00	10	0.00	10	0.00	10	0.00
pr-136 (96772)	10	0.00	10	0.00	10	0.00	10	0.00
pr-144 (58537)	10	0.00	10	0.00	10	0.00	10	0.00
KroA-150 (26524)	8	0.0007	7	0.002	8	0.0007	8	0.0007
KroB-150 (26130)	7	0.002	6	0.03	7	0.002	7	0.002
pr-152 (73682)	10	0.00	10	0.00	10	0.00	10	0.00
u-159 (42080)	10	0.00	10	0.00	10	0.00	10	0.00
rat-195 (2323)	1	1.89	2	0.22	2	0.22	2	0.22
d-198 (15780)	4	0.098	7	0.005	5	0.092	5	0.092
KroA-200 (29368)	10	0.00	10	0.00	10	0.00	10	0.00
KroB-200 (29437)	6	0.02	8	0.007	8	0.007	3	0.04
pr-226 (80369)	10	0.00	10	0.00	10	0.00	10	0.00
gil-262 (2378)	1	0.23	3	0.21	1	0.24	0	0.51
pr-264 (49135)	10	0.00	10	0.00	10	0.00	10	0.00

TAB 3.6 **Comparaisons des différentes stratégies de combinaisons des solutions avec τ =4**

Problèmes	OXHX-OB-Uniforme		OXHX-OB-DA		OXHX-OB-PC		OXHX-PC	
(tour opt.)	No-Op	% E-M	No-Op	% E-M	No-Op	% E-M	No-Op	% E-M
rat-99 (1211)	10	0.00	10	0.00	10	0.00	10	0.00
KroA-100 (21282)	10	0.00	10	0.00	10	0.00	10	0.00
KroB-100 (22141)	10	0.00	10	0.00	10	0.00	10	0.00
KroC-100 (20749)	10	0.00	10	0.00	10	0.00	10	0.00
KroD-100 (21294)	10	0.00	10	0.00	10	0.00	10	0.00
KroE-100 (22068)	9	0.017	9	0.017	10	0.00	10	0.00
rd-100 (7910)	10	0.00	10	0.00	10	0.00	10	0.00
eil-101 (629)	9	0.048	9	0.048	10	0.00	9	0.048
lin-105 (14379)	10	0.00	10	0.00	10	0.00	10	0.00
pr-107 (44303)	10	0.00	10	0.00	10	0.00	10	0.00
pr-124 (59030)	10	0.00	10	0.00	10	0.00	10	0.00
bier-127 (118282)	10	0.00	10	0.00	10	0.00	10	0.00
pr-136 (96772)	10	0.00	10	0.00	10	0.00	10	0.00
pr-144 (58537)	10	0.00	10	0.00	10	0.00	10	0.00
KroA-150 (26524)	8	0.0007	9	0.0003	9	0.0003	9	0.0003
KroB-150 (26130)	7	0.002	6	0.003	6	0.003	7	0.002
pr-152 (73682)	10	0.00	10	0.00	10	0.00	10	0.00
u-159 (42080)	10	0.00	10	0.00	10	0.00	10	0.00
rat-195 (2323)	1	0.23	2	0.22	2	0.22	1	0.23
d-198 (15780)	8	0.0025	8	0.0025	8	0.0025	8	0.0025
KroA-200 (29368)	9	0.16	10	0.00	10	0.00	10	0.00
KroB-200 (29437)	8	0.007	8	0.007	9	0.0003	5	0.002
pr-226 (80369)	10	0.00	10	0.00	10	0.00	10	0.00
gil-262 (2378)	1	0.23	3	0.21	1	0.24	1	0.24
pr-264 (49135)	10	0.00	10	0.00	10	0.00	10	0.00

TAB. 3.7 **Comparaisons des différentes stratégies de combinaisons des solutions avec τ =2,3 et 4**

Problèmes	OXHX-OB-Uniforme		OXHX-OB-DA		OXHX-OB-PC		OXHX-PC	
(tour opt.)	No-Op	% E-M	No-Op	% E-M	No-Op	% E-M	No-Op	% E-M
rat-99 (1211)	10	0.00	10	0.00	10	0.00	10	0.00
KroA-100 (21282)	10	0.00	10	0.00	10	0.00	10	0.00
KroB-100 (22141)	9	0.26	10	0.00	10	0.00	10	0.00
KroC-100 (20749)	10	0.00	10	0.00	10	0.00	10	0.00
KroD-100 (21294)	10	0.00	10	0.00	10	0.00	10	0.00
KroE-100 (22068)	7	0.52	8	0.34	9	0.17	7	0.52
rd-100 (7910)	10	0.00	10	0.00	10	0.00	10	0.00
eil-101 (629)	9	0.48	9	0.48	9	0.48	9	0.48
lin-105 (14379)	10	0.00	10	0.00	10	0.00	10	0.00
pr-107 (44303)	10	0.00	10	0.00	10	0.00	10	0.00
pr-124 (59030)	10	0.00	10	0.00	10	0.00	10	0.00
bier-127 (118282)	10	0.00	10	0.00	10	0.00	10	0.00
pr-136 (96772)	9	0.37	9	0.37	9	0.37	9	0.37
pr-144 (58537)	10	0.00	10	0.00	10	0.00	10	0.00
KroA-150 (26524)	9	0.003	10	0.00	9	0.003	10	0.00
KroB-150 (26130)	8	0.01	6	0.03	7	0.02	7	0.02
pr-152 (73682)	10	0.00	10	0.00	10	0.00	10	0.00
u-159 (42080)	10	0.00	10	0.00	10	0.00	10	0.00
rat-195 (2323)	2	1.85	1	1.89	4	1.42	2	1.85
d-198 (15780)	4	0.09	7	0.04	4	0.11	7	0.05
KroA-200 (29368)	9	0.16	10	0.00	10	0.00	10	0.00
KroB-200 (29437)	9	0.0004	8	0.007	8	0.007	3	0.04
pr-226 (80369)	10	0.00	10	0.00	10	0.00	10	0.00
gil-262 (2378)	1	0.23	3	0.21	1	0.24	1	0.24
pr-264 (49135)	10	0.00	10	0.00	10	0.00	10	0.00

TAB. 3.8 **Comparaisons des Temps d'exécution avec $\tau = 2$**

Problèmes	OXHX-OB-DA	OXHX-OB-Uniforme	OXHX-OB-PC	OXHX-PC
(tour opt.)	Temps (sec.)	Temps (sec.)	Temps (sec.)	Temps (sec.)
rat-99 (1211)	300.0	335.0	284.0	398.0
KroA-100 (21282)	284.0	383.0	334.0	266.0
KroB-100 (22141)	400.0	450.0	384.0	383.0
KroC-100 (20749)	315.0	249.0	384.0	267.0
KroD-100 (21294)	235.0	301.0	499.0	350.0
KroE-100 (22068)	335.0	416.0	450.0	401.0
rd-100 (7910)	351.0	367.0	283.0	334.0
eil-101 (629)	321.0	283.0	233.0	300.0
lin-105 (14379)	275.0	243.0	233.0	165.0
pr-107 (44303)	310.0	231.0	185.0	350.0
pr-124 (59030)	296.0	337.0	264.0	633.0
bier-127 (118282)	815.0	378.0	350.0	1499.0
pr-136 (96772)	751.0	50.0	649.0	767.0
pr-144 (58537)	267.0	16.0	333.0	251.0
KroA-150 (26524)	1550.0	83.0	317.0	3482.0
KroB-150 (26130)	1666.0	33.0	499.0	1935.0
pr-152 (73682)	435.0	50.0	399.0	432.0
u-159 (42080)	2316.0	168.0	566.0	648.0
rat-195 (2323)	1783.0	1550.0	6766.0	3304.0
d-198 (15780)	3200.0	83.0	1393.0	4246.0
KroA-200 (29368)	4521.0	1345.0	3473.0	4328.0
KroB-200 (29437)	3207.0	3612.0	2678.0	3129.0
pr-226 (80369)	431.0	324.0	756.0	687.0
gil-262 (2378)	5123.0	4351.0	5468.0	5893.0
pr-264 (49135)	5789.0	5813.0	6721.0	6452.0

TAB. 3.9 **Comparaisons des Temps d'exécution avec $\tau = 3$**

Problèmes	OXHX-OB-DA	OXHX-OB-Uniforme	OXHX-OB-PC	OXHX-PC
(tour opt.)	Temps (sec.)	Temps (sec.)	Temps (sec.)	Temps (sec.)
rat-99 (1211)	484.0	334.0	798.0	299.0
KroA-100 (21282)	269.0	385.0	633.0	383.0
KroB-100 (22141)	432.0	418.0	747.0	434.0
KroC-100 (20749)	267.0	316.0	283.0	384.0
KroD-100 (21294)	767.0	398.0	400.0	519.0
KroE-100 (22068)	651.0	330.0	283.0	402.0
rd-100 (7910)	418.0	315.0	301.0	251.0
eil-101 (629)	447.0	314.0	566.0	385.0
lin-105 (14379)	398.0	251.0	500.0	215.0
pr-107 (44303)	147.0	196.0	434.0	283.0
pr-124 (59030)	401.0	316.0	316.0	250.0
bier-127 (118282)	624.0	496.0	334.0	334.0
pr-136 (96772)	611.0	117.0	788.0	671.0
pr-144 (58537)	232.0	451.0	374.0	417.0
KroA-150 (26524)	1867.0	1652.0	1555.0	1240.0
KroB-150 (26130)	1376.0	1172.0	2222.0	2056.0
pr-152 (73682)	489.0	147.0	381.0	241.0
u-159 (42080)	2031.0	1320.0	2339.0	2126.0
rat-195 (2323)	2144.0	2902.0	3094.0	2368.0
d-198 (15780)	3934.0	3009.0	4243.0	4251.0
KroA-200 (29368)	4011.0	4347.0	4464.0	5795.0
KroB-200 (29437)	5156.0	5691.0	6988.0	5576.0
pr-226 (80369)	606.0	372.0	537.0	660.0
gil-262 (2378)	5348.0	5394.0	6583.0	6409.0
pr-264 (49135)	6352.0	6198.0	6159.0	6338.0

TAB. 3.10 **Comparaisons des Temps d'exécution avec $\tau = 4$**

Problèmes	OXHX-OB-DA	OXHX-OB-Uniforme	OXHX-OB-PC	OXHX-PC
(tour opt.)	Temps (sec.)	Temps (sec.)	Temps (sec.)	Temps (sec.)
rat-99 (1211)	298.0	350.0	433.	350.0
KroA-100 (21282)	283.0	333.0	416.0	252.0
KroB-100 (22141)	400.0	468.0	416.0	483.0
KroC-100 (20749)	269.0	300.0	300.0	201.0
KroD-100 (21294)	568.0	569.0	434.0	601.0
KroE-100 (22068)	632.0	683.0	384.0	316.0
rd-100 (7910)	316.0	300.0	318.0	366.0
eil-101 (629)	239.0	293.0	266.0	358.0
lin-105 (14379)	305.0	495.0	318.0	223.0
pr-107 (44303)	579.0	247.0	267.0	346.0
pr-124 (59030)	319.0	479.0	309.0	309.0
bier-127 (118282)	707.0	110.0	324.0	327.0
pr-136 (96772)	511.0	78.0	158.0	140.0
pr-144 (58537)	306.0	96.0	290.0	473.0
KroA-150 (26524)	1514.0	70.0	754.0	833.0
KroB-150 (26130)	1810.0	47.0	723.0	780.0
pr-152 (73682)	126.0	59.0	304.0	351.0
u-159 (42080)	2411.0	153.0	2346.0	2197.0
rat-195 (2323)	3482.0	1487.0	907.0	1437.0
d-198 (15780)	3053.0	100.0	2319.0	2710.0
KroA-200 (29368)	4781.0	3938.0	5236.0	4118.0
KroB-200 (29437)	4938.0	3846.0	3642.0	3617.0
pr-226 (80369)	742.0	110.0	534.0	512.0
gil-262 (2378)	5892.0	5631.0	6476.0	5020.0
pr-264 (49135)	6313.0	6353.0	6611.0	5596.0

TAB. 3.11 **Comparaisons des Temps d'exécution avec** $\tau = 2, 3et4$

Problèmes	OXHX-OB-DA	OXHX-OB-Uniforme	OXHX-OB-PC	OXHX-PC
(tour opt.)	Temps (sec.)	Temps (sec.)	Temps (sec.)	Temps (sec.)
rat-99 (1211)	383.0	366.0	317.0	400.2
KroA-100 (21282)	368.0	368.0	385.0	350.0
KroB-100 (22141)	500.0	534.0	500.0	385.0
KroC-100 (20749)	282.0	351.0	383.0	417.0
KroD-100 (21294)	483.0	566.0	535.0	483.0
KroE-100 (22068)	433.0	552.0	348.0	350.0
rd-100 (7910)	366.0	266.0	299.0	232.0
eil-101 (629)	384.0	201.0	302.0	268.0
lin-105 (14379)	282.0	267.0	168.0	316.0
pr-107 (44303)	234.0	217.0	2102.0	300.0
pr-124 (59030)	316.0	218.0	283.0	402.0
bier-127 (118282)	2100.0	852.0	3183.0	834.0
pr-136 (96772)	733.0	33.0	3114.0	2051.0
pr-144 (58537)	150.0	49.0	298.0	368.0
KroA-150 (26524)	333.0	34.0	1501.0	567.0
KroB-150 (26130)	1898.0	84.0	2451.0	533.0
pr-152 (73682)	232.0	50.0	350.0	619.0
u-159 (42080)	1382.0	798.0	1934.0	1248.0
rat-195 (2323)	4933.0	799.0	5467.0	4333.0
d-198 (15780)	4099.0	166.0	799.0	832.0
KroA-200 (29368)	6285.0	2817.0	1652.0	1816.0
KroB-200 (29437)	6882.0	6986.0	4783.0	13201.0
pr-226 (80369)	799.0	2365.0	2267.0	567.0
gil-262 (2378)	4916.0	5833.0	5767.0	5268.0
pr-264 (49135)	6682.0	6721.0	7342.0	8056.0

Conclusions

Notre objectif est de vérifier l'efficacité de la méthode "Scatter Search" pour résoudre le [PVC]. Nous la comparons avec une nouvelle approche génétique qui exploite aussi le principe de la diversification. Voici les conclusions que nous considérons être les plus importantes.

1. À la lumière des résultats obtenus, nous notons que la diversification permet d'élargir le champ d'exploration du domaine réalisable afin de trouver de meilleures solutions que la variante sans diversification n'a pu identifier.
2. L'utilisation tant des meilleures solutions dans l'ensemble de référence comme de l'incorporation de l'information dans les opérateurs de combinaison a fait que l'approche "Scatter Search" est la meilleure parmi les heuristiques étudiées dans ce mémoire.

Voici maintenant quelques pistes intéressantes de futures recherches sur la résolution du [PVC] en utilisant l'approche "Scatter Search" :

1. Diversifier toute solution engendrée par la combinaison de solutions de la Phase Scatter Search.
2. Appliquer une méthode d'amélioration tant aux solutions engendrées par la combinaison qu'à celles engendrées pour la diversification.

Bibliographie

[1] Adrabinski A. and Syslo M.M. Computational experiments with some heuristic algorithms for the travelling salesman problem. Technical Report N-78, Institute of Computer Science, Wroclaw University, Wroclaw,Poland, 1980.

[2] Angeniol B. Vaubois G. and Texier J. Self-organizing feature maps and the TSP. *Neural Networks*, 3 :289–293, 1988.

[3] Arthur J.L. and Frendewey J.O. Generating travelling salesman problems with known optimal tours. *Operations Research Society*, 2 :153–159, 1988.

[4] Balas E. and Christofides N. A restricted lagrangean approach to the traveling salesman problem. *Mathematical Programming*, 21 :19–46, 1981.

[5] Bellmore M. and Malone J.C. Pathology of traveling-salesman subtour-elimination algorithms. *Operations Research*, 19 :278–307, 1971.

[6] Bland R.G. and Shallcross D.F. Large traveling salesman problems arising experiments in x-ray crystallography : A preliminary report on computation. *Operations Research Letters*, 8 :125–128, 1989.

[7] Bodin L. Golden B. Assad A. and Bail M. Routing and scheduling of vehicles and crews : The state of art. *Computers and Operational Research*, 2, 1983.

[8] Bonomi E. Lutton J.L. The n-city traveling salesman problem :statistical mechanics and the metropolis algorithm. *SIAM Review*, 26 :551–568, 1984.

[9] Braun H. On solving travelling salesman problems by genetic algorithms. *in H. Schwefel and R. Manner (eds.), Parallel Problem-Solving from Nature. Lecture Notes in Computer Science 496*, 129–133, 1991.

[10] Carpaneto G. Fischetti M. and Toth P. New lower bounds for the symmetric travelling salesman problem. *Mathematical Programming*, 45 :233–254, 1988.

[11] Carpaneto G. and Toth P. Some new branching and bounding criteria for the asymmetric traveling salesman problem. *Management Science*, 26 :736–743, 1980.

[12] Cepeda M. and Oliva C. L'implantation de modèles génétiques pour le problème du voyageur de commerce. *Rapport de travail de cours. Université de Montréal*, Automme 1997.

[13] Christofides N. and Eilon S. Algorithms for large-scale traveling salesman problem. *Operations Research Quarterly*, 23 :511–518, 1972.

[14] Clarke G. and Wright J. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12 :568–581, 1964.

[15] Cung V-D. Mautor T. Michelon P. and Tavares A. Scatter Search for the Quadratic Assignment Problem. Technical Report URA 1525, Laboratoire PRiSM-CNRS, 1996.

[16] Dantzig G.B. Fulkerson D.R. Johnson S.M. Solution of a large-scale traveling salesman problem. *Operations Research*, 2 :393–410, 1954.

[17] Davis L. Applying adaptive algorithms to epistactic domains. *Proceedings International Joint Conference on Artificial Intelligence.* Los Angeles, California, 162–164, 1985.

[18] Dorigo M. and Gambardella L. Ant colonies for the traveling salesman problem. *Biosystem*, 43 :73–81, 1997.

[19] Durbin R. and Willshaw D. An Analogue Approach to the Traveling Salesman Problem using Elastic Net Method. *Nature*, 326 :689–691, 1987.

[20] Eastman W.L. Linear Programming with Pattern Constraints. *Ph.D. thesis, Harvard University, Cambridge, MA*,1958.

[21] Fiechter C-N. A Parallel Tabu Search Algorithm for Large Traveling Salesman Problems. *Discrete Applied Mathematics*, 51 :243–267, 1994.

[22] Fleurent C. Glover F. Michelon P. and Valli Z. A Scatter Search Approach for Unconstrained Continuous Optimization. *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, 643–648, 1996.

[23] Flood M. The traveling-salesman problem. *Operations Research*, 4 :61–75, 1956.

[24] Forrest S. Genetic algorithms :principle of natural selection applied to computation. *Science*, 261 :872–878, 1993.

[25] Garfinkel R.S. Minimizing wallpaper waste part I :a class of traveling salesman problems. *Operations Research*, 25 :741–751, 1977.

[26] Gavish B. and Srikanth K.N. An optimal solution method for large-scale multiple traveling salesman problems. *Operations Research*, 34 :698–717, 1986.

[27] Gendreau M. Hertz A. and Laporte G. New insertion and post-optimization procedures for the traveling salesman problem. *Operations Research*, 40 :1086–1094, 1992.

[28] Glover F. Heuristic for integer programming using surrogate constraints. *Decision Sciences*, 8 :156–166, 1977.

[29] Glover F. Tabu search : Part I. *ORSA journal on Computing*, 1 :190–209, 1989.

[30] Glover F. Tabu search : Part II. *ORSA journal on Computing*, 2 :4–32, 1990.

[31] Glover F. et Laguna M. *Tabu Search.* Kluwer Academic Publishers, 1997.

[32] Glover F. A Template for Scatter Search and Path Relinking. School of Business, University of Colorado, Boulder, U.S.A, 1997.

[33] Glover F. and McMillan C. The general employee scheduling problem : An integration of ms and ai. *Computers and Operations Research*, 13 :563–573, 1986.

[34] Goldberg D.E. *Genetics Algorithms in Search, optimization and Machine Learning.* Addison-Wesley, 1989.

[35] Goldberg D.E. and Lingle R. Alleles, loci and the traveling salesman problem. In *Proceeding 1st. International Conference on Genetic Algorithms and Their Applications.* Carnegie-Mellon University, Pittsburg, 154–159, 1985.

[36] Golden B. Bodin L. Doyle T. and Stewart W.Jr. Approximate traveling salesman algorithms. *Operations Research*, 28 :694–711, 1980.

[37] Golden B.L. and Skiscim C.C. Using simulated annealing to solve routing and location problems. *Naval Research Logistics Quarterly*, 33 :261–280, 1986.

[38] Golden B.L. and Stewart W.R. Empirical analysis of heuristics *in The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization.* Wiley-Interscience Series in Discrete Mathematics and Optimization, 207–249, 1985.

[39] Goldstein M. Self-organizing feature maps for the multiple traveling salesman problem (MTSP). *Proceedings International Neural Network Conference.* Paris, Francia, 258-261, 1990.

[40] Grefenstette J. Gopal R. Rosmaita B.J. and Gucht D.V. Genetic algorithms for the traveling salesman problem. *Proceedings 1st. International Conference on Genetic Algorithms and Their Applications.* Carnegie-Mellon University, Pittsburg, 160–168, 1985.

[41] Grefenstette J. Incorporating problem specific knowledge into genetic algorithms *in Genetic Algorithms and Simulated Annealing.* Morgan Kaufmann, 42–60, 1987.

[42] Hansen K. and Krarup J. Improvements of the held-karp algorithm for the symmetric traveling salesman problem. *Mathematical Programming*, 7 :87–96, 1974.

[43] Hegde S. Sweet J. and Levy W. Determination of parameters in a Hopfield/Tank computational network. *Proceedings IEEE Second International Conference on Neural Networks.* San Diego, California, 259–266, 1988.

[44] Held M. and Karp R.M. The traveling salesman problem and minimun spanning trees : Part i. *Operations Research*, 18 :1138–1162, 1970.

[45] Held M. and Karp R.M. The traveling salesman problem and minimun spanning trees : Part ii. *Mathematical Programming*, 1 :6–26, 1971.

[46] Jog P. Suh J. and Gucht D. The effects of population size, heuristic crossover and local improvement on a genetic algorithm for the traveling salesman problem. *Proceedings 3rd. International Conference on genetic algorithms.* George Mason University, Fairfax, 110–115, 1989.

[47] Johnson D.S. Local Optimization and the traveling salesman problem. *in G. Goos and J.Hartmanis (eds.), Automata, Languages and Programming, Lectures Notes in Computer Science 443* , 446–461, 1990.

[48] Kanellakis P.C. and Papadimitriou C.H. Local search for the asymmetric traveling salesman problem. *Operations Research*, 28 :1086–1099, 1980.

[49] Kirkpatrick S. Gelatt Jr. and Vecchi M.P. Optimization by simulated annealing. *Science*, 220 :671–680, 1983.

[50] Laporte G. The traveling salesman problem : An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59 :231–247, 1992.

[51] Lenstra J.K. Lawler E.L. Rinnooy Kan A.H.G. and Shmoys D.B. *The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization.* Wiley-Interscience Series in Discrete Mathematics and Optimization, 1985.

[52] Lenstra J.K. and Rinnooy Kan A.H.G. Some simple applications of the traveling salesman problem. *Operational Research Quarterly*, 26 :717–733, 1975.

[53] Lin S. Computer solutions of the traveling salesman problem. *Bell System Computer Journal*, 44 :2245–2269, 1965.

[54] Lin S. and Kernighan B.W. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21 :498–516, 1973.

[55] Little J.D.C Murty K.G. Sweeney D.W. and Karel C. An algorithm for the traveling salesman problem. *Operations Research*, 11 :972–989, 1963.

[56] Malek M. Guruswamy M. Pandya M. and Owens H. Serial and Parallel Simulates Annealing and Tabu Search for the Traveling Salesman Problem. *Annals of Operations Research*, 21 :5-84, 1989.

[57] Miller D.L. and Pekny J.F. Exact solution of large asymmetric traveling salesman problems. *Science*, 251 :754–761, 1991.

[58] Miller C.E. Tucker A.W. and Zemlin R.A. Integer programming formulations and the traveling salesman problem. *Journal of the Association for Computing Machinery*, 7 :326–329, 1960.

[59] Nahar S. Sahni S. and Shragowitz. Simulated annealing and comibatorial optimization. *International Journal of Computer Aided VLSI Design*, 1 :1–23, 1989.

[60] Nemhauser and Wolsey. *Integer and Combinatorial Optimization.* John Wiley & Sons, 1988.

[61] Oliver I.M. Smith D.J. and Holland J.R.C. A study of permutation crossover operators on the traveling salesman problem. In *Proceedings 2nd. International Conference on Genetic Algorithms and Their Applications.* Massachusetts Institute of Technology, Cambridge, 224–230, 1987.

[62] Or I. Traveling Salesman-Type Combinatorial Problems and their Relation to the Logistics of Regional Blood Banking. *Ph.D. thesis,Northwestern University, Evanston, Illinois*,1976.

[63] Potvin J-Y. The traveling salesman problem : A neural network perspective. *ORSA Journal on Computing*, 5 :328–348, 1993.

[64] Potvin J-Y. Genetic algorithms for the traveling salesman problem. *Annals of Operations Research*, 63 :339–370, 1996.

[65] Potvin J-Y. Notes du cours Sujets spéciaux d'optimisation. Université de Montréal, Automme 1997.

[66] Reeves C. *Modern Heuristic Techniques for Combinatorial Problems.* John Wiley & Sons, 1993.

[67] Reinelt G. A traveling salesman problem library. *ORSA Journal on Computing*, 3 :376–384, 1991.

[68] Rosenkrantz D.J. Stearns R.E. and Lewis P.M. An analysis of several heuristics for the travelling salesman problem. *SIAM Journal on Computing*, 6 :563–581, 1977.

[69] Shapiro D.M. Algorithms for the solution of the Optimal Cost Traveling Salesman Problem. *Sc.D. thesis, Washington University, St.Louis,Missouri*,1966.

[70] Smith T.H.C. and Thompson G.L. A lifo implicit enumeration search algorithm for the symmetric traveling salesman problem using held and karp's 1-tree relaxation. *Annals of Discrete Mathematics*, 1 :479–493, 1977.

[71] Suh J. and Gucht D. Incorporating heuristic information into genetic search. *Proceedings 2nd. International Conference on genetic algorithms.* MIT, Cambridge, 100–107, 1987.

[72] Syslo M. Deo N. and Kowalik J. *Discrete Optimization Algorithms with Pascal Programs.* Prentice-Hall Inc., 1983.

[73] Szwarcfiter J.L. *Grafos e Algoritmos Computacionais.* COPPE, 1984.

[74] Ulder N. Aarts E. Bandelt H. van Laarhoven P. and Pesch E. Genetic local serach algorithms for the traveling salesman problem. *in H. Schwefel and R. Manner (eds.), Parallel Problem-Solving from Nature. Lecture Notes in Computer Science 496*, 109–116, 1991.

[75] Volgenant T. and Jonker R. A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation. *European Journal of Operational research*, 9 :83–89, 1982.

[76] Wilson G. and Pawley G. On the stability of the TSP algorithm of Hopfield and Tank. *Biological Cybernetic*, 58 :63–70, 1988.

[77] Yue T. and Fu L. Ineffectiveness in solving combinatorial optimization problems using a Hopfield network : A new perspective from aliasing effect. *Proceedings of the International Joint Conference on Neural Network*. San Diego, California, 787–792, 1972.

Remerciements

Je désire exprimer ma gratitude à mon co-directeur de recherche Jacques Ferland pour sa disponibilité, sa grande patience et ses conseils judicieux pour la rédaction de ce mémoire.

Je tiens à remercier à Gloria et Carlos pour son amitié.

Je voudrais aussi remercier mon amie Lorena Pradenas pour ses encouragements et son aide dans ce long projet.

Je remercie mes parents et mes sœurs pour leur soutien moral tout au long de ma vie.

Je réserve le mot de la fin à ma famille et spécialement à ma femme, Claudia, qui a su me témoigner sa confiance et son amour dans les moments difficiles.