Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider Biskra -Algérie
Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie
Département d'Informatique
Laboratoire LESIA

N° d'ordre :…….

Série :……

Thèse présentée pour l'obtention du diplôme de

Doctorat troisième cycle (LMD)

Option

Techniques d'image et d'intelligence artificielle

Bee life Parallèle sur GPU pour résoudre le problème dynamique de tournées de véhicules avec une contrainte de capacité (DCVRP)

Réalisée par : Maroua GRID

Encadreur : Pr. NourEddine DJEDI

Co-encadreur : Dr. Salim BITAM

Devant le Jury:

Président:	Pr. Med Chaouki BABAHENINI,	Professeur,	Université Med Khider , Biskra.
Rapporteur:	Pr. NourEddine DJEDI,	Professeur,	Université Med Khider , Biskra.
Co-Rapporteur:	Dr. Salim BITAM,	MCA,	Université Med Khider, Biskra.
Examinateurs:	Pr. Mouloud KOUDIL,	Professeur,	Esi Alger.
	Pr. Karima BENATCHBA,	Professeur,	Esi Alger.
	Dr. Leila DJEROU,	MCA,	Université Med Khider, Biskra

Année Universitaire 2017/2018

ملخــص

في الوقت الحاضر، لا تزال هناك فجوة كبيرة بين متطلبات وإمكانيات النظم المساعدة على اتخاذ القرار، بالنسبة للكثير من المشاكل مثل مشكلة التخطيط لمسار المركبات، والذي يتمثل في إنشاء مجموعة من المسالك المثلى لأسطول من المركبات، التي تهدف إلى خدمة عدد معين من العملاء. ومع ذلك، يمكن تقديم طلبات عملاء جديدة في حين أن المخطط السابق في التقدم. ولذلك، ينبغي إعادة حساب المسالك بطريقة ديناميكية. في هذه الأطروحة، نقترح طريقة جديدة متوازية للتحسين الاندماجي تعتمد على وحدة معالجة الرسومات (GPU) تسمى الخوارزمية المتوازية لحياة النحل (P-BLA)، لحل المشكلة الديناميكية للتخطيط لمسار المركبات (DCVRP) من حيث وقت التنفيذ، وللحد من التعقيد الحسابي الذي غالبا ما يعتبر العائق الرئيسي لأساليب التحسين التقليدية. تم تطوير P-BLA باستخدام برنامج CUDA و بالاعتماد على GPU، والقائم على نموذج الجزيرة. بعد مجموعة من المقارنات مع الأساليب التقليدية وهي؛ الخوارزمية الجينية، نظام النمل، البحث تابو و BLA المتسلسل، P-BLA قدم نتائج فعالة والمتحصل عليها انطلاقا من معالم DCVRP الأكثر اختبارا.

الكلمات المفتاحية: DCVRP، k-means ، P-BLA، التحسين المتوازي، GPGPU.

Abstract

Nowadays, there is still a large gap between the requirements and the performance of decision support systems for many problems such as the vehicle routing problem, consists in conceiving a set of optimal routes for a fleet of vehicles, aiming at serving a given number of customers. Nevertheless, new customer orders could be introduced while a prior plan is in progress. Therefore, routes should be recalculated in a dynamic way. In this thesis, we propose a new parallel combinatorial optimization method based on Graphic Processing Unit (GPU) called Parallel Bees Life Algorithm (P-BLA) to solve effectively the Dynamic Capacitated vehicle routing problem (DCVRP) in terms of execution time, and to reduce computational complexity often considered as the major drawback of conventional optimization methods. P-BLA is developed using CUDA framework performed on island-based GPU. After a set of comparisons against conventional methods namely; genetic algorithm, ant system, Tabu search and sequential BLA, P-BLA has provided efficient results reached from the most tested DCVRP benchmarks.

Keywords: DCVRP; k-means; P-BLA; Parallel optimization; GPGPU.

Résumé

De nos jours, il existe encore un écart important entre les exigences et la performance des systèmes d'aide à la décision pour de nombreux problèmes tels que le problème de planification de tournées de véhicules. Ce problème consiste à concevoir un ensemble de routes optimales pour une flotte de véhicules visant à servir un nombre donné de clients. Néanmoins, de nouvelles demandes (clients) pourraient être introduites pendant qu'un plan préalable est en cours de réalisation. Par conséquent, les routes doivent être recalculées de manière dynamique.

Dans cette thèse, nous proposons une nouvelle méthode d'optimisation combinatoire parallèle appelée Parallel Bees Life Algorithm (P-BLA), basée sur l'unité de traitement graphique (GPU) pour résoudre efficacement le problème dynamique de tournées des véhicules (DCVRP), en termes de temps d'exécution. La réduction de la complexité de calcul a été souvent considérée comme l'inconvénient majeur des méthodes d'optimisation classiques. L'algorithme P-BLA a été développé en utilisant le logiciel CUDA, et a été implémenté sur GPU en se basant sur le modèle d'îlot. Nous avons réalisé, en outre, un ensemble de comparaisons entre P-BLA avec des méthodes conventionnelles comme l'algorithme génétique, système de fourmi, recherche Tabou et avec BLA séquentiel. P-BLA a fourni des résultats efficaces obtenus à partir des benchmarks de DCVRP les plus testés.

Mots-clés : DCVRP; k-means; P-BLA; Optimisation parallèle; GPGPU.

Publications et Communications

Travaux scientifiques relatifs à cette thèse :

Publications

1. Maroua Grid, Noureddine Djedi, Salim Bitam. (2017). 'GPU-based distributed bee swarm optimization for dynamic vehicle routing problem', Int. J. Ad Hoc and Ubiquitous Computing, en cours de publication, accepté le 27/11/2017.

Communications

1. Maroua Grid, Noureddine Djedi, Salim Bitam. (2014). A parallel bee life algorithm for DCVRP on GPUs. The 5th International Conference on Metaheuristics and Nature Inspired Computing, META2014, Marrakech (Morocco), pages 3-5.

Workshops

1. Maroua Grid, Noureddine Djedi, Salim Bitam. (2014). A Parallel Bee Life Algorithm for DCVRP on GPUs. Le Deuxième workshop Images, Graphiques et Vie Artificielle, Biskra (Algérie). 16,17-18 Juin, 2014.

AVANT PROPOS

Notre travail de recherche effectué dans le cadre de doctorat troisième cycle (LMD), sous la direction du professeur **NourEddine DJEDI**, et la codirection du docteur **Salim BITAM**, Laboratoire de Systèmes experts, Imagerie et leurs applications dans l'Ingénierie (**LESIA)**, Université Mohamed Khider, Biskra.

Je tiens premièrement à prosterner remerciant Allah le tout puissant de m'avoir donné le courage et la patience pour terminer ce travail. Ensuite, je tiens à exprimer mes sincères remerciements :

Au professeur **NourEddine DJEDI** d'avoir accepté de diriger les travaux de recherche de cette thèse et pour son dynamisme, et qui a été pour moi un directeur de thèse attentif et disponible malgré ses nombreuses charges. Sa compétence, sa rigueur scientifique et sa clairvoyance m'ont beaucoup appris. Ils ont été et resteront des moteurs de mon travail de chercheur.

Au docteur **Salim BITAM**, co-directeur de cette thèse pour son encadrement, son suivi rigoureux, ses recommandations précieuses, sa patience et ses grandes qualités humaines et également pour sa disponibilité tout au long de la réalisation de ce travail. Qu'il trouve ici le témoignage de mon estime et ma profonde connaissance pour ses conseils pertinents, et ses compétences scientifiques qu'il a mis à ma disposition pour mener à bien ces travaux de recherches.

Au professeur **Med Chaouki BABAHENINI**, Chef de département d'informatique de l'Université Mohamed Khider, Biskra, qui nous a fait l'honneur de présider le jury de cette thèse. Qu'il trouve ici le témoignage de mon estime et ma profonde reconnaissance pour ses grandes qualités scientifiques, de son dynamisme et pour les efforts qu'il déploie pour favoriser le développement de l'enseignement et de la recherche scientifique dans notre département.

Au professeur **Mouloud KOUDIL**, Professeur à l'ESI (Ecole Nationale Supérieure d'Informatique), Alger, pour l'intérêt qu'il a manifesté en participant en qualité de membre invité à ce jury. Je le remercie pour le temps consacré à la lecture de ce travail ainsi que pour les commentaires m'ayant permis de l'améliorer.

Au professeur **Karima BENATCHBA**, ESI (Ecole Nationale Supérieure d'Informatique), Alger, pour l'honneur qu'elle m'a fait pour sa participation à mon jury

de thèse en qualité d'examinatrice de mon travail, pour le temps consacré à la lecture de cette thèse, et pour les suggestions et les remarques judicieuses qu'il m'a indiquées.

Au docteur **Leila DJEROU**, Université Mohamed Khider, Biskra, pour le temps précieux qu'il a consacré pour examiner ce travail, qui m'a fait l'honneur de faire partie des membres de ce jury. Qu'il trouve ici le témoignage de mon estime et ma profonde reconnaissance.

Je remercie toutes les personnes formidables que j'ai rencontrées par le biais de l'ESIA. Merci pour votre support et vos encouragements. Je remercie particulièrement toutes les équipes de ce laboratoire.

Enfin, les mots les plus simples étant les plus forts, j'adresse toute mon affection à ma famille, et en particulier à ma maman qui m'a fait comprendre que la vie n'est pas faite que de problèmes qu'on pourrait résoudre grâce à des formules mathématiques et des algorithmes. Son intelligence, sa confiance, sa tendresse, son amour me portent et me guident tous les jours.

Merci à tous ceux que je n'ai pas pu citer mais qui ont toutes mes amitiés et ma reconnaissance.

Table des matières

Liste des figures

Liste des tableaux

Introduction générale

Le problème de tournées de véhicules avec une contrainte de capacité (Capacitated Vehicle Routing Problem, CVRP) représente l'une des variantes du problème de tournées de véhicules (Vehicle Routing Problem, VRP) impliquant différentes contraintes. Ce problème est l'un des problèmes les plus étudiés dans le domaine de l'optimisation combinatoire, en raison de ses applications importantes dans le domaine de la logistique et du transport. Initialement, VRP a été présenté comme une généralisation du problème de voyageur de commerce (Traveling Salesman Problem, TSP). Dans CVRP, une flotte de véhicules homogènes sert un ensemble de clients à partir d'un seul dépôt et elle retourne à ce dernier. Cette flotte est destinée à fournir à toutes les demandes connues de clients avec un cout minimum, qui est exprimées par la distance totale parcourue. CVRP considère que tous les véhicules possèdent une capacité de livraison identique et limitée attribuée à un certain nombre de clients.

La version dynamique du problème de tournées de véhicules avec une contrainte de capacité (Dynamic CVRP, DCVRP) est considérée comme une nouvelle classe de problème de VRP. Ce problème apparait grâce à des progrès récents dans la technologie de l'information et de la communication qui permet de nouveaux événements tels que de nouvelles demandes, se produisent même lorsque les clients sont servis. Ainsi, DCVRP nécessite une mise à jour dynamique et en temps réel de sa solution antérieure pendant son exécution. DCVRP est considérée comme un problème NP-difficile, c.-à-d. la recherche de la solution optimale pour ce dernier nécessite un temps de calcul très long en raison de sa très grande complexité. Par conséquent, des méthodes d'optimisation avancées (appelées aussi métaheuristiques) ont été appliquées comme approches efficaces pour résoudre ce type de problèmes avec une très faible complexité. Cependant, avec des instances DCVRP très élevées (c'est-à-dire le nombre de clients ou les nouveaux événements ou demandes), la qualité de la solution trouvée utilisant les métaheuristiques classiques peut être améliorée par des approches de calcul avancées, tel que le traitement parallèle.

Le développement des algorithmes parallèles est un défi qui pourrait réduire le temps nécessaire à l'optimisation et fournir des outils réactifs efficaces qui contribuent à prendre la bonne décision. Les dernières années ont vu l'émergence d'un grand nombre d'architectures informatiques parallèles. Presque tous les ordinateurs de bureau ou

portables, et même les téléphones mobiles, sont équipés de plusieurs cœurs de processeurs intégrés. Les GPUs ont également été découverts comme source de puissance de calcul massive sans coût supplémentaire. L'utilisation du GPU dans des applications non-graphiques a eu un intérêt croissant dans le monde de l'optimisation combinatoire depuis les années 2009. Le succès du GPU revient à son coût faible, car c'est une architecture grand public produite en série et facilement accessible. L'enjeu financier rend cette technologie très adéquate à l'implémentation des métaheuristiques parallèles pour la résolution des problèmes combinatoires. Grâce à des langages tels que CUDA (Compute Unified Device Architecture) et OpenCL (Open Computing Language), les informaticiens sont capables d'exploiter le potentiel du GPU pour le calcul généraliste (General Purpose GPU, GPGPU), c.-à-d. implémenter des algorithmes parallèles en exploitant la structure et la puissance de calcul du GPU.

Bees Life Algorithm BLA est considéré comme une métaheuristique bio-inspiré, et qui est une stratégie assez efficace pour trouver des solutions approximatives à des problèmes d'optimisation combinatoire, les temps de calcul associés à l'exploration de l'espace de solution peuvent être très importants. Contrairement aux autres métaheuristiques, l'efficacité de BLA est due à ses deux opérations importantes telles que la reproduction et la recherche de nourriture. En particulier, la reproduction est réalisée par les opérateurs de croisement et de mutation, qui garantit la diversification dans le processus de recherche de solution, néanmoins la recherche de nourriture aide à assurer l'intensification où la meilleure solution pourrait être découverte dans un espace de recherche local.

Dans cette thèse, nous présentons notre contribution qui se compose de deux composants principaux. Le premier composant est appelé le gestionnaire d'événements, qui effectue deux tâches principales : gérer les demandes des clients, et créer des instances de CVRP statiques. Le deuxième composant correspond à la méthode de résolution qui résout les occurrences de CVRP successives créées par le gestionnaire d'événements. La méthode de résolution a été divisée en deux phases, la construction de groupes où les clients sont affectés aux routes des véhicules, en utilisant la méthode k-means améliorée (*enhanced k-means*) et l'optimisation des routes effectuée par l'algorithme de la vie des abeilles (*Bees Life Algoriyhm, BLA*).

Une autre proposition qui est l'implémentation parallèle basée sur l'unité de traitement graphique (GPU) de BLA appelée Parallel Bees Life Algorithm (P-BLA) pour résoudre le DCVRP. Cette proposition tire parti de la puissance des GPGPU, en particulier en utilisant l'architecture CUDA de NVIDIA afin de surmonter la complexité de calcul des méthodes d'optimisation classiques, ainsi que de réduire le temps d'exécution pour découvrir les routes de manière dynamique.

Problématique

La plupart des problèmes dans la vie réelle sont caractérisés par leur nature dynamique où ils peuvent être résolus en passant par une modélisation en tant que systèmes dynamiques combinatoires. Dans cette catégorie, les systèmes de transport dynamiques ont connu un très grand intérêt durant ces dernières années dans le domaine de recherche en optimisation et ainsi que celui des systèmes de transport intelligents (Intelligent Transport System, ITS).

La planification de tournées en temps réel implique la construction et l'exécution de plans tout en évoluant dans un environnement dynamique. Un aspect important de la planification en continu réside dans la génération de plans, une tâche consistant à gérer des événements en temps réel au cours de la planification et durant l'exécution de plans. Plusieurs solutions proposées à ce problème, utilisant des métaheuristiques, sont purement prescriptives ne garantissant que la convergence à une solution réalisable au détriment de l'optimalité. De plus, ces approches ont un handicap majeur, qui nuit à leur intégration dans des systèmes d'aide à la décision pour des applications réelles, connu sous le nom du problème de calibrage des paramètres (des métaheuristiques).

Objectifs de la thèse.

Les besoins économiques et stratégiques, appuyés par les nouveaux moyens technologiques, nourrissent la recherche visant le développement d'outils d'aide à la décision qui sont en mesure de fournir de bonnes solutions, conformes à la réalité. Notre thèse s'inscrit dans le cadre de cette nouvelle direction de recherche et puise sa motivation plus particulièrement dans la volonté de répondre aux problématiques citées dans la section précédente. Plus précisément, les objectifs de cette thèse sont définis comme suit :

- Concevoir et implémenter une nouvelle approche de recherche pour supporter la planification de tournées dans un environnement dynamique, en respectant la contrainte de capacité de véhicules.
- Exploiter les performances de GPU, afin de produire des solutions robustes et de meilleure qualité au temps courant.

Organisation de la thèse

La structure de cette thèse est décrite ci-dessous :

Le chapitre1 explique le problème de tournée de véhicules statique (VRP), et le problème de tournée de véhicules dynamique (DVRP), en mettant l'accent sur leur formulation mathématique. Ainsi que, la différence entre ces deux problèmes. Puis, nous précisons les mesures de dynamisme, la classification de DVRP, et ces exigences techniques. Enfin, nous citons quelques exemples du monde réel, et quelques variantes de DVRP.

Le chapitre 2 présente un aperçu général et une synthèse sur les différentes méthodes utilisées pour la résolution du problème de DVRP, qui sont divisées en deux classes des méthodes séquentielles et des méthodes parallèles.

Le chapitre 3 aborde le principe de notre approche **BLA-DCVRP** inspirée de la vie des abeilles (Bees Life) que nous avons choisie pour résoudre le DCVRP. Nous indiquons les différents composants de notre contribution, en débutant par le gestionnaire d'évènements. Puis, en passant à la méthode de résolution qui est composée de deux phases : la construction des groupes effectuée par *k-means amélioré* (*Enhanced k-means*), et l'optimisation des routes en utilisant les colonies d'abeilles qui fait l'objet de notre étude. Dans ce contexte, nous allons discuter sur les abeilles dans la nature, en donnant la structure d'une colonie d'abeille, ensuite une description de la communication chez les abeilles, et leur comportement. Enfin, nous allons discuter sur le principe de l'algorithme de la vie des abeilles (Bees Life Algorithm, BLA), ainsi que les adaptations que nous avons faites sur cet algorithme séquentiel, pour qu'il puisse traiter le DCVRP.

Le chapitre 4 présente un aperçu général sur les métaheuristiques parallèles, et sur l'exploitation du potentiel de GPU pour le calcul généraliste (General Purpose GPU, GPGPU), pour implémenter des métaheuristiques parallèles.

Le chapitre 5 propose une nouvelle méthode d'optimisation combinatoire parallèle basée sur l'unité de traitement graphique (GPU) appelée Parallel Bees Life Algorithm (P-BLA) pour résoudre le DCVRP. Cette proposition tire parti de la puissance des GPGPU, en particulier en utilisant l'architecture CUDA de NVIDIA afin de surmonter la complexité de calcul des méthodes d'optimisation classiques, ainsi que de réduire le temps d'exécution pour découvrir les tournées de manière dynamique. Ensuite, nous expliquons l'implémentation parallèle de P-BLA sur CUDA, en spécifiant ces différent kernels.

Le chapitre 6 présente quelques expérimentations sur des instances de problème de tournées de véhicules dynamiques de service de collecte. Le but de cette étude expérimentale est d'évaluer la performance de nos algorithmes proposés BLA et P-BLA, par rapport à d'autres approches qui existent dans la littérature. L'évaluation de performance concerne les meilleures solutions trouvées, et la précision de chaque approche. En plus, nous avons étudié l'effet d'accélération des GPU sur le P-BLA par rapport à BLA.

Enfin, on conclut ce travail par une conclusion et certaines orientations de recherche futures.

Chapitre 1. Problème de tournées de véhicules dynamique (DVRP)

Sommaire

1.1. Introduction

Le problème de tournées de véhicules (Vehicle Routing Problem, VRP en anglais) est un problème d'optimisation combinatoire et de recherche opérationnelle, qui a été largement étudié durant plus de cinquante ans, suite à plusieurs raisons comme la difficulté de sa résolution, et ses nombreuses applications pratiques en logistique[1] pour des gains économiques liées à la minimisation des coûts des systèmes de transport, et l'optimisation des réseaux de distribution. Le VRP implique la planification de routes de livraison ou de collecte à moindre coût, par une flotte de véhicules dotés d'une capacité limitée, partants de et revenants à un ou plusieurs dépôts afin de servir un ensemble de clients dispersés géographiquement, en respectant les contraintes de capacité des véhicules. Dans ce cas-là, on se trouve face à un problème de tournées de véhicules avec une contrainte de capacité (Capacited Vehicle Routing Problem, CVRP) notre cas d'étude.

[1] Est définit par **Breusard** comme étant " l'ensemble des méthodes et moyens permettent d'apporter un bon produit (ou service) à un client en temps voulu, en quantité voulue et au lieu voulu à moindre coût. " **[1]**

Dans la version statique du VRP, on suppose que tous les clients sont connus à l'avance lors du processus de planification. Dans d'autres cas, les gestionnaires des tournées de véhicules sont souvent confrontés à des situations d'urgence comme l'apparition d'un nouveau client à servir, un accident sur la route, etc., qui nécessite une replanification. Dans le cas d'un VRP statique, ceci peut être considéré comme un manque d'information lors de l'élaboration des tournées et que des clients peuvent apparaître en cours de journée quand la planification ait été déjà effectuée et que les véhicules sont en train de servir des clients déjà planifiés. Les nouveaux clients doivent alors être insérés dans les routes confectionnées de manière à satisfaire leurs besoins et ceux du gestionnaire. Mais grâce aux progrès récents dans les technologies de l'information et de la communication, les flottes de véhicules peuvent être maintenant contrôlés en temps réel via des dispositifs spéciaux comme les systèmes d'information géographique (Geographic Information Systems, GISs), les systèmes de localisation globale (Global Positioning Systems, GPSs), et les téléphones mobiles qui peuvent fournir des données en temps réel, telles que les emplacements actuels des véhicules, les nouvelles demandes des clients, et l'évaluation périodique de temps de déplacement des routes. Lorsque tout est traité convenablement, cette grande quantité de données peut être utilisée pour réduire le coût et améliorer la qualité de service d'une entreprise moderne. C'est pour cela de nouvelles routes doivent être mises à jour dès que des nouveaux évènements se produisent. Le problème du VRP statique devient alors un problème de VRP dynamique (DVRP). Il en découle que les variantes du VRP statique peuvent devenir des variantes dynamiques comme le cas du CVRP et sa version dynamique DCVRP.

Le VRP est un problème de type NP-difficile c.-à-d. qui ne dispose pas un algorithme général permettant de le résoudre en un temps polynomial. Par conséquent, le DVRP appartient également à cette classe, mais il est plus compliqué que le VRP statique puisque ce dernier devrait être résolu à chaque fois qu'une nouvelle demande immédiate soit reçue. Pour cette raison, et pour résoudre le problème de DVRP, on a recouru à des procédures robustes et en ligne d'optimisation qui fonctionnent en temps réel puisque les demandes immédiates doivent être servies le plus rapidement possible.

Dans ce chapitre nous commençons par présenter le problème de tournée de véhicules statique (VRP) en mettant l'accent sur ses différents paramètres ainsi que sur sa modélisation mathématique. Ensuite, nous voyons le problème de tournée de

véhicules dynamique (DVRP), sa formulation mathématique générale, et nous discutons sur la différence entre le VRP statique et le DVRP. Puis, nous précisons les mesures de dynamisme, la classification de DVRP, et ces exigences techniques. Enfin, nous citons quelques exemples du monde réel, et quelques variantes de DVRP.

1.2. Le problème de tournées de véhicules VRP

Le problème de tournées de véhicules VRP représente à la fois des caractéristiques d'allocation de ressources (répartition des charges dans les camions : bin packing), et de construction de séquences (problème de voyageur de commerce, connu également sous le nom de Travelling Salesman Problem (**TSP**)) **[2]**.

Le VRP a été introduit pour la première fois par Dantzig et Ramser en 1959 **[3]**, sous le nom de « **Truck Dispatching Problem** » qui est une extension du problème du voyageur de commerce (TSP). Depuis lors, il est devenu l'objet de nombreux travaux qui ont donné de nombreuses variantes et différentes méthodes de résolution. Nous allons donc dans un premier temps définir le problème du voyageur de commerce, puis nous verrons son extension : le problème de tournées de véhicules.

1.2.1. Le problème du voyageur de commerce (Traveling Salesman Problem, TSP)

Le TSP est le problème le plus célèbre et le plus étudié en optimisation combinatoire. Dans ce problème, un voyageur de commerce doit visiter n villes (ou clients) puis il doit retourner à la ville de départ, en passant une et une seule fois par chacune d'entre elles, et en minimisant la distance totale parcourue **[4]**. Ce problème représente souvent un banc d'essai pour de nouveaux idées ou paradigmes avant de passer aux modèles plus avancés.

Plus formellement, un TSP est modélisé sous forme d'un graphe complet[2] G = (V, A) (Figure 1.1), où les sommets V = $\{v_1, v_2, \dots, v_n\}$ représentent les villes à visiter, et les arêtes $a_{ij} \in$A sont les liaisons entre ces villes **[5]**. La pondération d_i ou le poids associé à chaque arête représente le coût de la liaison entre les deux villes et correspond généralement à la distance qui les sépare.

[2] Un graphe complet à n sommets possède $\frac{n(n-1)}{2}$ arêtes.

L’objectif de ce problème est de trouver un cycle ou circuit de coût minimum, visitant l’ensemble des n sommets du graphe. Autrement dit, il s’agit de trouver un cycle ou circuit Hamiltonien, c-à-d un cycle passant une et une seule fois par tous les sommets du graphe, et de longueur minimale.

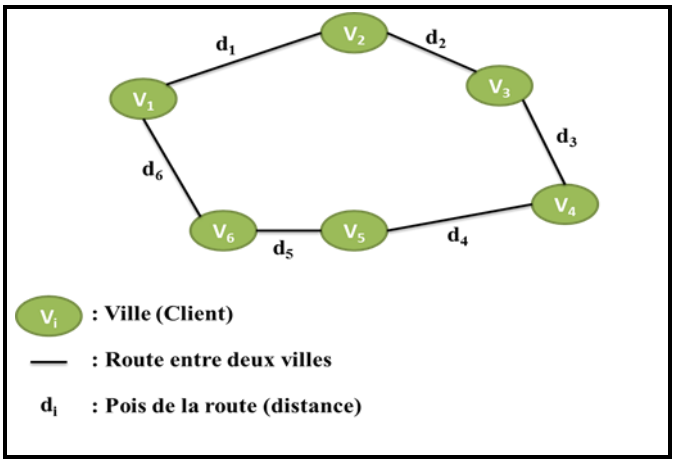

Figure 1.1 Une solution typique d'un exemple de TSP.

Malgré sa formulation simple, le TSP est un problème NP-difficile **[6]** parmi les problèmes les plus complexes en recherche opérationnelle, et en optimisation combinatoire. Dans sa version symétrique, c-à-d dans le cas où le graphe associé n'est pas orienté, le nombre total de solutions possibles est $\frac{(n-1)!}{2}$ où n est le nombre de villes. Avec une telle complexité factorielle, une résolution efficace du TSP nécessite donc le recours à des heuristiques spécialisées voire même à des métaheuristiques. En effet, les méthodes exactes restent limitées aux problèmes de petite taille.

Il existe de nombreuses variantes de ce problème dans la littérature **[7]**. En particulier, le problème de voyageurs de commerce multiples, également connu sous le nom de Multiple Traveling Salesman Problem (MTSP), a été étudié, par exemple par Bektas **[8]**. Le MTSP est un problème semblable au TSP qui cherche une tournée optimale des n villes par m vendeurs, et chaque ville doit être visitée par exactement un vendeur avec l’absence des sous-cycles. Par conséquence, il faut diviser les n villes en m tournées, où chaque tournée ayant pour résultat un TSP pour un vendeur. Le MTSP est plus difficile que le TSP parce qu'il exige déterminer quelles villes à assigner à chaque vendeur, aussi bien que la commande des villes soit optimale dans la tournée de chaque vendeur.

1.2.2. Définition de VRP

Le problème de construction de tournées de véhicules (VRP, Vehicle Routing Problem en anglais) est une extension du problème du voyageur de commerce classique. Chaque problème VRP peut être considéré comme un ensemble de plusieurs problèmes TSP dépendants (MTSP).

Pendant les années passées, un certain nombre d'auteurs ont renommé ce problème par le problème de tournée de véhicule (VRP) sous une contrainte de capacité (ou CVRP). Ce problème implique la conception des tournées pour une flotte de K véhicules placés dans un dépôt, de capacités limitées, et qui doit servir plusieurs clients (entrepôts, entreprises, usines, villes, etc.) ayant demandé chacun une certaine quantité de marchandises. L'ensemble des clients visités par un véhicule désigne la tournée de celui-ci. Chaque client doit être servi une et une seule fois et chaque tournée commence et se termine au dépôt (Figure 1.2).

L'objectif du VRP est de minimiser le coût total (la somme des distances ou des temps de parcours) des tournées, tout en respectant un certain nombre de contraintes. Ce problème relève du domaine de l'Optimisation Combinatoire ou discrète, qui consiste à trouver une meilleure solution, comme étant définie par une fonction objective, parmi un ensemble de solutions réalisables. Ainsi, un problème d'optimisation combinatoire peut être formulé **[9]** comme suit :

Soit S, l'ensemble des solutions possibles d'un problème. Soit $X \subseteq S$, l'ensemble des solutions admissibles vérifiant un ensemble de contraintes C. Soit une fonction $f : X \longmapsto R$ appelée fonction objectif. Un problème d'optimisation combinatoire consiste à déterminer : $\{\min f(x) : x \in X\}$

Un problème NP-difficile est un problème dont l'existence d'un algorithme déterministe exacte qui puisse le résoudre dans un temps polynomial est peu probable. Le VRP est évidemment NP-difficile **[10]** puisqu'il généralise le TSP qui est déjà NP-difficile.

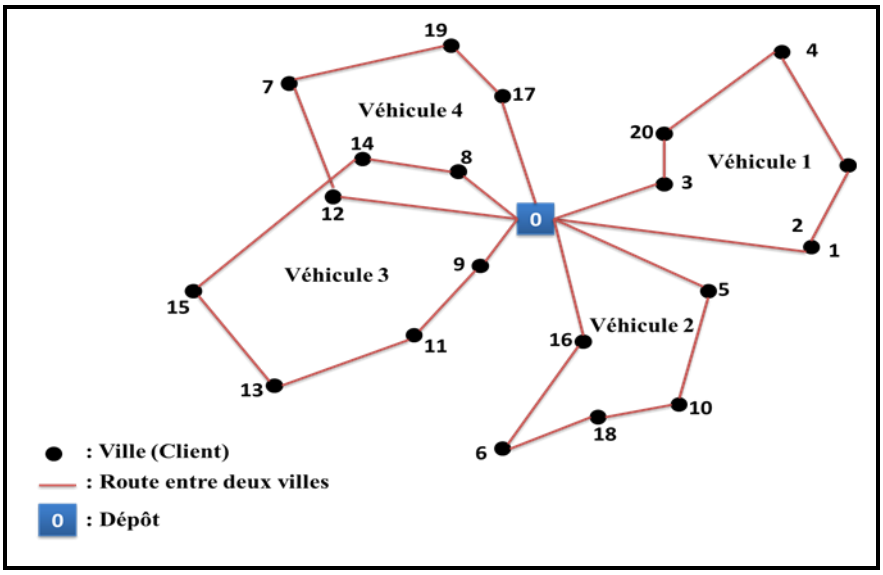

Figure 1.2 Un exemple de problème de VRP avec 20 clients et 4 véhicules.

1.2.3. Paramètres de VRP

Le VRP est caractérisé par un ensemble de paramètres : le réseau de transport, les demandes, les véhicules, les contraintes, et la fonction objective à optimiser.

a. Le réseau :

Le réseau routier peut être symétrique ou asymétrique ; par conséquence, le graphe associé G = (V, A) sera orienté ou non et les liaisons entre les sommets seront des arcs ou des arêtes pondérés par un coût, ce dernier désigne une distance ou un temps de parcours. Les tournées peuvent partir d'un seul dépôt ou de plusieurs dépôts.

b. Les demandes :

Chaque client exprime une quantité bien définie d'une ou plusieurs marchandises. Ces dernières peuvent être livrées aux clients et peuvent aussi remises aux véhicules. On parle alors de tournées de livraison/collecte ou de service mixte. Les temps de livraison et de collecte peuvent être non négligeables et sont ainsi pris en compte dans le calcul des durées de tournées.

c. Les véhicules :

Les véhicules constituent les ressources principales et indispensables pour l'organisation du transport. Le nombre de véhicules disponibles peut être fixe ou non, et ils ont des capacités limitées en termes de marchandises transportées (volume, poids etc.) à cause de contraintes techniques ou des réglementations en vigueur.

Les véhicules peuvent être homogène, c.-à-d. tous les véhicules ont la même capacité ou hétérogène. Dans le cas d'une flotte hétérogène, cette capacité peut différer

selon le type de véhicules. Chaque véhicule termine sa tournée doit revenir au même endroit (dépôt) de départ.

d. Les contraintes :

Les tournées de véhicules sont assujetties à un ensemble de contraintes, qui peuvent être partitionnées en trois types :

- **Les contraintes de capacité** : Dans cette contrainte la somme des quantités demandées par les clients dans une tournée ne doit pas dépasser la capacité du véhicule.
- **Les contraintes temporelles** : La livraison ou le collecte des marchandises peut être contrainte à s'effectuer dans une période de temps spécifiée par le client, appelée une fenêtre de temps (time Windows). Cette contrainte peut être dure ou souple. Dans le cas de contrainte dure, une arrivée avant la fenêtre de temps impose une attente, et les retards sont interdits ; alors que les contraintes souples peuvent être violées et induisent ainsi des pénalités.
- **Les contraintes de précédence** : Lorsqu‘on a un problème de collecte et livraison (Pick Up & Delivery Problem, PDP en anglais), cette contrainte doit être vérifiée pour que les marchandises puissent être transportées d’un lieu vers un autre.

e. La fonction objective :

La fonction-objectif représente une fonction de coût à minimiser ou une fonction de profit à maximiser. En ce qui concerne le VRP, on recherche généralement les objectifs suivants **[11]** :

- ✓ La minimisation de la durée totale des tournées ;
- ✓ La minimisation des distances parcourues au cours des tournées ;
- ✓ La minimisation des pénalités liées aux violations des contraintes, notamment dans le cas de fenêtres de temps,
- ✓ La minimisation du coût total des tournées (en prenant en compte les coûts des véhicules, des chauffeurs etc.).

1.2.4. Formulation de VRP

Le VRP est souvent remplacé par le CVRP (Capacitated VRP ou VRP avec contraintes de capacité). Dans celui-ci, une flotte de k véhicules (identiques de capacité Q, stationnés au dépôt) doit effectuer des tournées pour servir un ensemble de clients (service de livraison ou de collecte). Chaque client est servi avec une quantité finie, et il

est visité une seule fois par un véhicule de la flotte qui retourne au dépôt à la fin de sa tournée.

D'après Rego et Roucairol dans **[12]**, le problème de CVRP peut être modélisé par un graphe complet G = (V, A), c-à-d que tous les sommets sont reliés entre eux. Cela signifie qu'une ville peut être visitée à partir de toute autre ville. Les autres constantes du problème sont les suivantes :

- n nombre de clients (ou sommets),
- m nombre de véhicules,
- Q capacité des véhicules,
- q_i demande du client i,
- c_{ij} le coût de l'arête entre les sommets i et j (distance ou temps de parcours),

La variable de décision utilisée est ainsi définie :

$$x_{ij}^k = \begin{cases} 1 \text{ si (i, j) est parcouru par le véhicule k;} \\ 0 \qquad \text{sinon} \end{cases} \qquad (1.1)$$

Ainsi, en tant que problème d'optimisation, le CVRP s'écrit :

$$\text{Min } \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \sum_{k=1}^{m} x_{ij}^k \qquad (1.2)$$

Sujet aux contraintes suivantes :

$$\sum_{i=1}^{n} \sum_{k=1}^{m} x_{ij}^k = 1 \qquad \forall\, 1 \le j \le n \qquad (1.3)$$

$$\sum_{j=1}^{n} \sum_{k=1}^{m} x_{ij}^k = 1 \qquad \forall\, 1 \le i \le n \qquad (1.4)$$

$$\sum_{i=1}^{n} \sum_{l=1}^{n} x_{il}^k - \sum_{l=1}^{n} \sum_{j=1}^{n} x_{lj}^k = 0 \qquad (1.5)$$

$$\sum_{j=1}^{n} x_{0j}^k = 1 \qquad \forall\, 1 \le k \le m \qquad (1.6)$$

$$\sum_{i=1}^{n} x_{i0}^k = 1 \qquad \forall\, 1 \le k \le m \qquad (1.7)$$

$$\sum_{i=1}^{n} q_i \left(\sum_{j=1}^{n} x_{ij}^k \right) \le Q \qquad \forall\, 1 \le k \le m \qquad (1.8)$$

$$x_{ijk} \in \{0,1\} \qquad \forall 0 \le i, j \le n; 1 \le k \le m \qquad (1.9)$$

La sous formulation (1.2) signifie que l'objectif du problème d'optimisation est de minimiser la somme des coûts de toutes les tournées. Les contraintes (1.3) et (1.4) imposent que chaque client soit servi une et une seule fois, et la contrainte (1.5) assure la continuité d'une tournée par un véhicule : le sommet visité doit être quitté impérativement. Les contraintes (1.6) et (1.7) assurent que chaque tournée commence et

se termine au dépôt. Finalement, la contrainte (1.8) est une contrainte de capacité et la contrainte (1.9) est une contrainte de binarité sur les variables de décision x_{ij}^{k}.

1.3. Le problème de tournées de véhicules dynamique DVRP

Depuis le début des années 1970s, le problème de tournées de véhicules dynamique (DVRP, Dynamic Vehicle Routing Problem en anglais) a été étudié comme étant un problème du transport à la demande (DARP, Dial-A-Ride Problem en anglais) par Wilson et al. **[13]**, ils ont étudié ce problème avec un seul véhicule, dans lequel des demandes des clients apparaissent dynamiquement concernant des voyages d'une origine à une destination. Psaraftis **[14]** a introduit le concept de la demande immédiate, où un client demandant un service veut toujours être servi le plus tôt possible, ce qui nécessite la replanification immédiate de la tournée actuelle du véhicule. Après, plusieurs études et aperçus sont élaborés par (Psaraftis **[15,16]**, Gendreau et Potvin **[17]**).

1.3.1. Définition de DVRP

Psaraftis [15] avait défini le problème de VRP statique comme : « si la sortie d'une certaine formulation est un ensemble de routes préplanifiées qui ne sont pas ré-optimisées et qui sont calculées à partir des entrées qui n'évoluent pas en temps réel ». Tandis qu'il se réfère à un problème comme dynamique si « la sortie n'est pas un ensemble de routes, mais plutôt une stratégie qui prescrit comment les routes devraient évoluer comme une fonction de ces entrées qui évoluent en temps réel ».

D'après Allan Larsen [18], la différence entre un VRP statique et un VRP dynamique (DVRP) est présentée dans le Tableau 1.1, qui concerne la connaissance d'informations par le planificateur avant que le processus de planification commence, et le changement de ces dernières après la construction des routes initiales. Cette information inclut tous les attributs des clients tels que la position géographique, le temps de service passé sur place, et la demande de chaque client (c.-à-d. quantité à collecter ou à livrer). En plus, les informations concernant les temps de trajets nécessaires pour relier les clients à servir doivent être connues ou calculables par le planificateur.

	VRP statique	DVRP
-Toute information (données du problème) concernant la planification des routes est supposée connue par le planificateur avant que le processus de planification commence.	OUI	NON
-L'information peut changer après la construction des routes initiales.	NON	OUI

Tableau 1.1 Différence entre un VRP statique et un VRP dynamique (DVRP).

D'après les définitions ci-dessus, nous pouvons comprendre que dans un problème dynamique, il n'est pas possible de déterminer à l'avance l'ensemble de routes optimisées parce qu'il existe quelques entrées du problème sont indiquées pendant l'exécution de l'algorithme, au fur et à mesure que la solution de tel problème est évoluée.

Il est évident que le DVRP est plus complexe que le VRP classique (statique). Si la classe des problèmes du VRP est dénotée P(VRP) et que la classe des problèmes du DVRP est dénotée P(DVRP), alors P(VRP)⊂P(DVRP). Etant donné que Le problème de VRP statique appartient à la classe des problèmes NP-difficiles, c'est pourquoi il n'est pas toujours possible de trouver une solution optimale pour des problèmes de taille réaliste dans un temps de calcul raisonnable. Par conséquent, le DVRP appartient également à cette classe, mais il est plus compliqué que le VRP statique puisque ce dernier devrait être résolu à chaque fois qu'une nouvelle demande immédiate soit reçue.

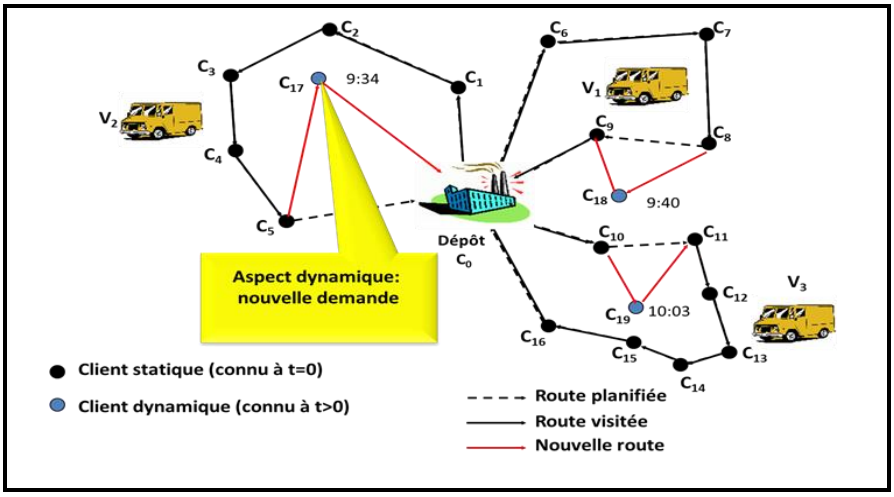

Figure 1.3 Scénario de DVRP avec 3 véhicules, 16 demandes connues à l'avance (demandes statiques), et 3 demandes immédiates (demandes dynamiques).

La Figure 1.3 illustre le cas d'un problème de tournées de véhicules dynamique dans lequel il existe trois véhicules doivent servir un ensemble de clients. Les clients statiques connus avant la planification des routes sont représentés par des nœuds noirs, alors que les clients dynamiques (immédiats) qui sont apparus après l'élaboration des routes sont représentés par des nœuds bleus. Les routes déjà planifiées pour les clients statiques sont représentées par des flèches en pointillés. Le trajet du véhicule avant l'apparition des clients dynamiques est représenté par des flèches pleines alors que les déviations proposées pour insérer les clients dynamiques afin de les servir sont représentés par des flèches rouges.

Pour mieux comprendre l'aspect dynamique dans le DVRP, la ci-dessus illustre le parcours exécuté par le véhicule numéro 1. Avant que le véhicule quitte le dépôt (à t=0), il existe une route initiale déjà planifiée pour servir les demandes des clients actuellement connues (C_6, C_7, C_8, C_9). Lorsque le véhicule est en cours de la route, et qu'une nouvelle demande d'un client (C_{18}) apparaisse à l'instant t=9:40, ceci nécessite une replanification de la manière suivante (C_6, C_7, C_8, C_{18}, C_9).

Dans la pratique, l'insertion de nouveaux clients est habituellement une tâche beaucoup plus compliquée et qu'il nécessite une replanification de la partie non-visitée dans un temps raisonnable.

1.3.2. Formulation de DVRP

Le problème dynamique de tournées de véhicules (DVRP) est fortement lié au VRP statique, il peut être décrit comme un problème de routage où tous les composants du problème peuvent changer pendant le processus d'optimisation. Soit la fonction objective, les variables de décision, ou les contraintes. Ceci implique que la solution optimale pourrait changer à tout moment en raison des changements de l'environnement comme par exemple l'arrivée de nouvelles demandes des clients, l'annulation des demandes, la modification de l'ordre de clients, la mise à jour de temps de déplacement pour certaines routes qui pourrait être augmenté en raison de mauvaises conditions météorologiques, lorsque la journée de service a déjà commencé, etc.

Le DVRP peut être par conséquent modélisée comme une séquence d'instances de VRP-statiques (voir section 1.2). En particulier, chaque VRP statique contiendra tous les clients connus à ce moment-là, mais pas encore servis. Le DVRP peut être formulé

de plusieurs façons, en fonction du problème de la vie réelle correspondant. Notre recherche a utilisé un modèle de DVRP proposé par Montemanni et al. **[19]**.

Dans la section 1.4 nous avons présenté la formulation mathématique du problème VRP. Pour adapter cette dernière au problème dynamique DVRP nous ajoutons la contrainte suivante concernant la réception d'une nouvelle demande :

$$0 < tr_i \leq T \; \forall \; i \in N' \qquad (1.10)$$

Où tr_i est l'instant de réception de la demande urgente, T est l'instant de fermeture du dépôt qui correspond à la fin du jour de service. Cette contrainte permet de respecter le fait que les demandes dynamiques doivent parvenir au planificateur après la sortie des véhicules et avant la fin de l'horizon de planification.

Le problème ne peut pas être résolu à l'avance parce que le décideur ne connait pas à priori le problème entier. Par conséquent, le but du processus d'optimisation ce n'est pas de trouver une seule solution optimale, mais plutôt, le suivi du déplacement des optimums au cours du temps puisque la solution optimale pour un exemple peut être une solution pauvre ou probablement même infaisable pour l'environnement prochain.

Psaraftis **[14]** prend une solution x_t du problème actuel p_t comme une solution expérimental. La solution expérimentale correspond seulement à l'ensemble actuel d'entrées. S'il n'y a aucune nouvelle demande de service reçue pendant l'exécution de cette solution, alors la route expérimentale deviendra optimale.

Ainsi, un problème dynamique de temps discret peut être exprimé comme une série de P instances : $P = \{(P_i, t_i, \Delta_t)/\ i=0,1, \ldots, i_{max}\}$. Chaque instance est un problème statique qui commence au temps t et doit être résolu dans un délai précis Δ_t.

Avec cette information la durée de l'instance i est $\Delta_i = t_{i+1} - t_i$. Le nombre maximum d'instances i_{max} peut être infini, si le problème est ouvert. Une nouvelle instance P_{i+1} est produite par l'action de changement de l'environnement δ_{i+1} sur l'instance P_i. Ceci est exprimé par $P_{i+1} = P_i \oplus \delta_{i+1}$. Alors, les solutions obtenues dans le temps t_i n'ont pas pu être les solutions faisables pour le temps t_{i+1}. Nous récapitulons cela comme indiqué dans la Figure 1.4.

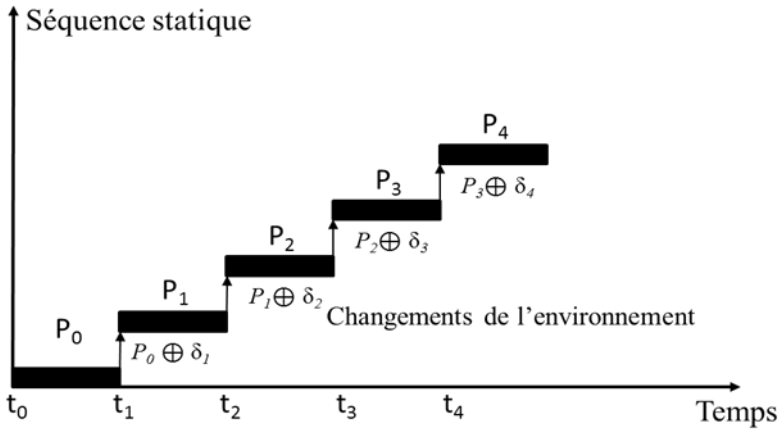

Figure 1.4 Décomposition d'un problème dynamique $P = (P_1, P_2, P_3, P_4)$ en une séquence d'instances statiques. [20]

Le changement de l'environnement correspond à l'arrivée du nouveau problème d'optimisation qui doit être résolu. Le temps consacré pour résoudre chaque instance dépend de la fréquence des changements **[21]**.

1.3.3. Différence entre le VRP statique et le DVRP

Les problèmes dynamiques de tournées de véhicules diffèrent de leurs homologues statiques, et les principales différences sont énumérés par Psaraftis [22] comme il est détaillé ci-dessous :

a. La dimension du temps est primordiale

Dans un problème de tournées de véhicules statique, la dimension du temps peut ou peut ne pas être importante. Mais dans le problème dynamique opposé, le temps est toujours essentiel parce que planificateur doit avoir un minimum de connaissances sur la position de tous les véhicules, la liste des clients servis et la liste des clients non encore servis à chaque instant, et particulièrement lorsqu'une nouvelle demande de service est reçue.

b. Un temps de calcul plus rapides est nécessaire

Dans un VRP statique, le planificateur peut attendre quelques heures afin d'obtenir une solution de haute qualité, parfois même optimale. Par contre, dans un DVRP ce n'est pas possible parce que le planificateur souhaite connaître la solution du problème actuel le plus tôt que possible (de préférence en quelques minutes ou secondes). La contrainte sur la durée des opérations implique que les réordonnancements et les ré-assignements sont souvent effectués en utilisant une

heuristique d'amélioration locale telle que l'insertion et le k-échange pour ne pas être contraint d'arrêter un ou plusieurs véhicules.

c. Des mécanismes de mise à jour de l'information sont essentiels

Vu que dans le cas d'un DVRP les données peuvent évoluer d'un instant à un autre à cause des changements de l'environnement, il est donc nécessaire que des mécanismes de mise à jour des informations soient intégrés dans la méthode de résolution.

d. Le réordonnancement et les décisions de ré-assignement sont nécessaires

Dans un DVRP l'apparition d'une ou de plusieurs nouvelles demandes implique que les décisions prises par le planificateur ne deviennent pas optimales voire mauvaises. Ceci oblige le planificateur à réordonnancer les clients et les véhicules et il peut même être contraint à utiliser en renfort un ou plusieurs véhicules.

e. Les événements à court terme sont plus importants

En raison de l'uniformité de la qualité de l'information et du manque des mises à jour des entrées, tous les événements portent le même poids dans un VRP statique. Mais, dans un cadre dynamique il serait imprudent immédiatement d'envisager une planification à long terme. Quand un problème dynamique est traité, le planificateur doit faire une planification qui devra mettre l'accent sur des événements à court terme.

f. Les données futures peuvent être imprécises ou inconnues

Dans les problèmes statiques, toutes les données sont connues avant la planification et elles sont de même qualité. Dans les problèmes dynamiques, la connaissance des événements futurs avec certitude n'est pas possible, c'est pourquoi, des données avec des probabilités peuvent être employées.

g. Les nouvelles demandes peuvent être retardées ou annulées

Pour une raison justifiée comme l'éloignement géographique important où l'impossibilité de satisfaire à temps une nouvelle demande, le planificateur peut la retarder indéfiniment ou l'annuler.

h. Une file d'attente pour la gestion des arrivées des tâches est importante

Dans le cas d'un problème dynamique, si le taux de demandes des clients dépasse un certain seuil, le système deviendra saturé ce qui mène les algorithmes à produire des résultats sans sens. Donc le système de planification doit posséder une file d'attente pour garder les demandes survenues lorsque ce dernier est occupé par le ré-ordonnancement.

i. Le problème peut être ouvert

Dans un VRP statique, Le processus est souvent borné temporellement où les tournées commencent et terminent au dépôt, et qui sont pré-planifiées avant la journée de service. Par contre, dans un DVRP le processus peut être non borné, avec par exemple des demandes dynamiques non servies au cours de cette journée et qui sont reportées à la journée suivante comme des demandes statiques.

1.3.4. Mesures de dynamisme et classification des DVRPs

Dans un VRP statique, la performance de la méthode de résolution utilisée dépend essentiellement du nombre de clients et de leur distribution spatiale Alors que dans un DVRP, le nombre de clients et leur distribution sont nécessaires, mais le nombre d'événements dynamiques, et leurs instants d'apparition jouent aussi un grand rôle. Par conséquent, une mesure pour décrire le dynamisme du système a une grande utilité quand nous voulons évaluer la performance d'une méthode de résolution dans des conditions variables. Dans la littérature, il existe plusieurs mesures de dynamisme parmi lesquelles on peut citer le degré de dynamisme sans fenêtres de temps (*degree of dynamism without time windows, dod*), le degré de dynamisme effectif (*effective degree of dynamism, edod*), et le degré de dynamisme effectif avec fenêtres de temps (*effective degree of dynamism with time windows, edod-tw*),

a. Degré de dynamisme sans fenêtres de temps

La notion de degré de dynamisme (*degree of dynamism without time windows, dod*) a été définie par Lund et al. [23] d'un problème dynamique de tournées de véhicules comme le rapport entre le nombre de demandes dynamiques et le nombre total de demandes (statiques et dynamiques), donc le *dod* ϵ [0,1].

$$dod = \frac{le\ nombre\ de\ demandes\ dynamiques}{le\ nombre\ total\ de\ demandes} \qquad (1.11)$$

Toutefois, le *dod* ne tient pas compte les temps d'arrivée des demandes dynamiques. Ceci signifie que plusieurs problèmes différents peuvent avoir la même valeur *dod* tout en ayant des distributions de l'apparition des demandes dynamiques différentes. La Figure 1.5 illustre deux scénarios de DVRP dans lesquels les instants de réception des demandes dynamiques diffèrent considérablement. Nous remarquons que dans le scénario A, chacune des six demandes dynamiques est reçue relativement tôt pendant l'horizon de planification. Par contre, dans le scénario B, les demandes sont uniformément distribuées sur tout l'horizon de planification (0-T). Les demandes statiques sont supposées être reçues avant le début de l'horizon de planification. Le temps d'apparition de la i^{ème} demande dynamique est dénoté t_i : $0 < t_i < T$.

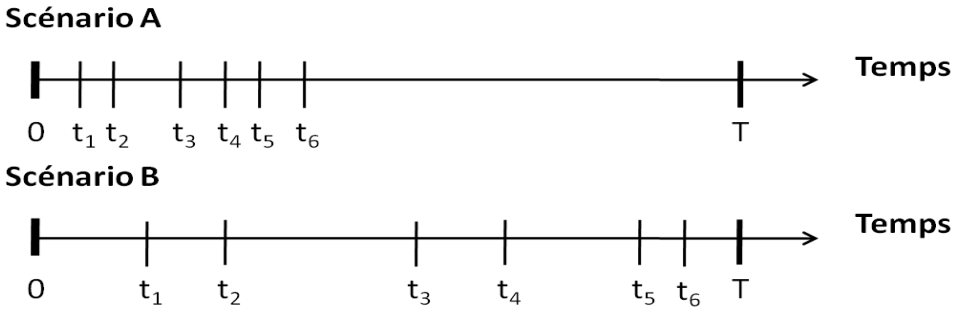

Figure 1.5 Deux scénarios de DVRP ayant des "dod" équivalents.

b. Degré de dynamisme effectif

Après pour corriger ce défaut Larsen [18] a défini une autre mesure de dynamisme représenté par le degré de dynamisme effectif (*effective degree of dynamism, edod*) qui prend en considération le temps d'occurrence des demandes. Cette mesure représente alors un pourcentage moyen des temps d'arrivée des demandes dynamiques par rapport à l'horizon temporel T. On suppose que l'horizon de planification est un intervalle de temps [0, T] et qu'il est divisé en un nombre fini de morceaux de temps, soit *ns* et *nd* le nombre des demandes statiques et dynamiques, respectivement. Et $t_i \in [0, T]$ est le moment où la demande *i* apparaît, les demandes statiques sont telles qui ont $t_i = 0$, tandis que les demandes dynamique ont $t_i \in [0, T]$.

$$edod = \frac{\sum_{i=1}^{nd}(t_i/T)}{ns+nd} \qquad (1.12)$$

Il a observé qu'un système dans lequel les demandes dynamiques sont reçues tard dans l'horizon de planification [0, T] est plus dynamique que d'autres, dans

lesquels les demandes dynamiques se produisent au début du jour de service. L'*edod* ϵ [0,1], quand l'*edod* = 1 le problème est complètement dynamique et lorsque l'*edod* = 0, le problème est purement statique.

c. Degré de dynamisme effectif avec fenêtres de temps

Dans plusieurs applications de tournées de véhicule le service des clients doit être effectué durant un intervalle de temps, qui est habituellement appelé fenêtre de temps (time window, tw). Larsen [18] a prolongé la définition de l'*edod*, où il a pris en considération les fenêtres de temps possible pour servir les demandes. Soit [e_i, l_i] l'intervalle du temps de client *i* qui représente la fenêtre de temps de celui-ci, avec e_i et l_i qui correspondant au premier et au dernier temps possible pour commencer le service, respectivement.

$$edod - tw = \frac{\sum_{i=1}^{ns+nd} (T-(l_i-t_i)/T)}{ns+nd} \qquad (1.13)$$

Le degré de dynamisme effectif avec fenêtres de temps permet de donner une notion et une valeur d'urgence à une demande en fonction de ($li - t_{i)}$ qui représente l'horizon de décision pour traiter la demande i. Plus cet horizon de décision est petit, plus la demande est urgente, il est aussi clair que l'*edod-tw* ϵ [0, 1]. Si aucune fenêtre de temps n'est imposée (c.-à-d., $e_i = t_i$ et l_i = T), alors *edod-tw* = *edod*.

d. Classification du DVRP

Nous utilisons ici la classification du DVRP présentée par Larsen **[18]**, qui a subdivisé les problèmes dynamiques en trois classes : les problèmes faiblement dynamiques, les problèmes modérément dynamiques et les problèmes fortement dynamiques comme il est montré dans la Figure 1.6.

Cette classification devrait être utile pour choisir les modèles et les algorithmes appropriés selon les caractéristiques dynamiques du système à examiner.

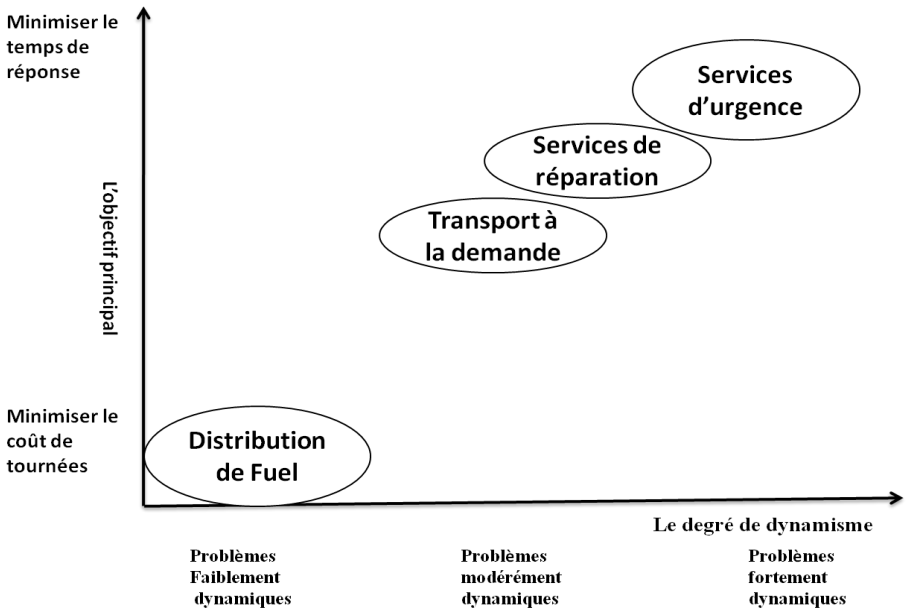

Figure 1.6 Classification du DVRP. [18]

- **Les problèmes faiblement dynamiques**

Dans les problèmes de DVRP avec un degré faible de dynamisme plus de 80% des clients sont statiques et connus avant la planification. Parmi ces problèmes, nous pouvons citer la distribution de Fuel, les services de réparation à domicile, le transport adapté (transport de personnes âgées ou handicapées), etc. En effet, l'objectif principal est de minimiser un coût total des tournées comme par exemple la distance totale parcourue par les véhicules.

- **Les problèmes modérément dynamiques**

Dans ce type de problèmes le nombre de clients statiques et dynamiques sont proches. Ces problèmes incluent le transport à la demande, la distribution de courrier, la distribution de nourriture aux revendeurs, les services de maintenance ou de dépannage à domicile pour le câble ou le téléphone, etc. L'objectif principal de ce type de problèmes est de minimiser le compromis entre le coût total des tournées et le temps de réponse aux demandes.

- **Les problèmes fortement dynamiques**

Ce type de problèmes concerne essentiellement les services de secours, tels que la police, les pompiers et les ambulances. L'objectif principal de ce type de problèmes est de minimiser le temps de réponse.

1.3.5. Exigences techniques

Allan Larsen [18] avait présenté les technologies les plus essentielles qui sont utilisées dans des applications réelles des problèmes de tournées de véhicule dans un environnement dynamique, comme il est indiqué dans cette section.

a. Equipement de positionnement et de communication :

La communication entre les conducteurs des véhicules et le centre d'expédition est essentielle afin d'introduire plus d'informations à jour dans le système de planification de tournées de véhicules. Ces équipements seront présentés ci-dessous.

- L'équipement de positionnement comme le système de positionnement global (Global positioning system, GPS) est indispensable pour un système dynamique de tournées de véhicules, qui représente une constellation de 24 satellites autour de la terre et qui envoient constamment des signaux en donnant leurs positions et le temps. Les signaux provenant de trois ou quatre satellites différents à un moment donné peuvent fournir aux récepteurs sur la terre des informations suffisantes pour calculer leur emplacement précis dans quelques mètres selon la version de GPS utilisé.

- L'équipement de communication entre le véhicule et le centre d'expédition est nécessaire pour la structure d'un système de planification. Les systèmes de communication des téléphones mobiles sont un exemple d'une technologie capable de fournir ces informations. Il existe aussi une autre technologie qui est les systèmes de communications radio. La différence principale dans ces deux technologies est les coûts initiaux et les coûts d'exploitation, le système de communications de téléphone mobile est relativement coûteux pour fonctionner, mais il a des coûts initiaux bas parce que la technique de base est fournie par les entreprises de téléphone et le système de GSM (Global System for Mobile Communications, en anglais) offre aujourd'hui presque une couverture totale dans la plupart des pays occidentaux industriels. D'autre part, les coûts initiaux dans l'implémentation d'un système de communication radio sont très élevés parce que les pylônes de transmission doivent être mis en place et relativement l'équipement radio installé dans chaque véhicule est trop cher, alors que les coûts d'exploitation sont presque négligeables. En outre, un système de communication radio n'offre pas la même flexibilité par rapport au système de communications de téléphone mobile.

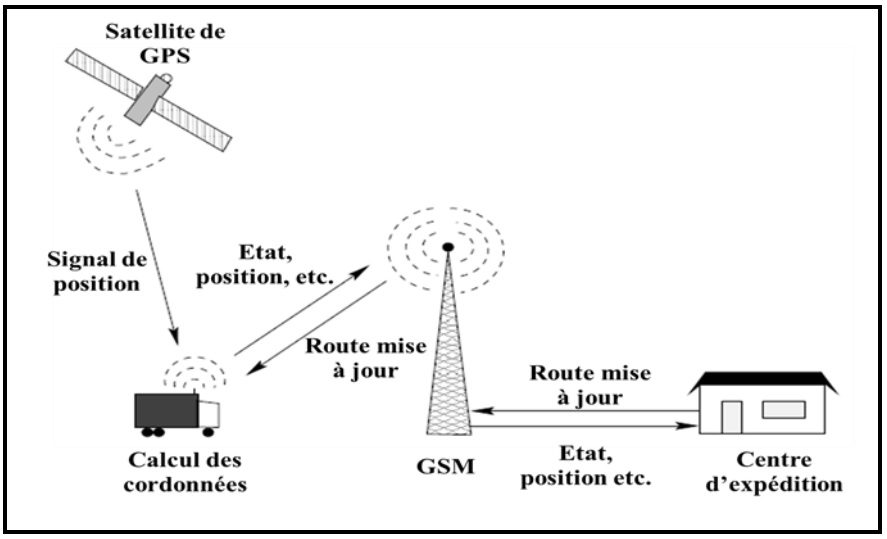

Figure 1.7 la circulation de l'information de données de base entre le véhicule et le centre d'expédition [18].

Idéalement, le centre d'expédition connaît l'état de véhicule et du conducteur à n'importe quel moment donné. Mais, comme il est indiqué ci-dessus ceci peut s'avérer infaisable pour quelques applications à cause des coûts d'exploitation de cette méthode. Cependant, dans un arrangement réaliste l'information de positionnement est transmise dans des intervalles précis. Par conséquent, un programme d'interpolation est utilisé afin d'estimer les positions des véhicules.

Alternativement, le conducteur envoie chaque fois qu'il finit le service d'un client un message concernant son état actuel et sa position au centre d'expédition. Mais cette approche n'offre pas le même niveau d'information pour que l'expéditeur prend sa décision, par exemple quel est le véhicule qui sera expédié au client suivant pour le servir.

b. Les systèmes d'information géographique :

Les progrès dans les cartes routières numériques et des systèmes d'information géographiques (SIG) ont été considérables dans les années 90. La plupart des pays occidentaux industrialisés ont détaillé maintenant -presque entièrement- des bases de données de réseau routier. Au Danemark, DAV (Dansk adress og vejdatabase -en danois, des bases de données d'adresse) offre une base de données numérique des routes qui est connectée aux informations détaillées sur chaque adresse dans le pays. La base de données DAV contient dans des codes zip des informations concernant par exemple les noms officiels de rues, les numéros de rues, la classification des routes, etc.

1.3.6. Exemples du monde réel de DVRP

Il existe plusieurs problèmes importants qui doivent être résolus en temps réel. Dans ce qui suit, on passe en revue les applications principales [24] qui motivent la recherche dans le domaine de planification de tournées de véhicules dynamiques.

- Gestion dynamique de flotte : Plusieurs opérations de camionnage nécessite l'expédition des véhicules en temps réel afin de rassembler ou de livrer des marchandises.
- Systèmes de distribution : Dans ces systèmes, les sociétés de distribution évaluent le niveau d'inventaire de client pour le reconstituer avant l'épuisement des stocks. Par conséquent, les demandes en principe sont connues à l'avance et tous les clients sont statiques. Mais, comme les demandes sont incertaines, quelques clients (habituellement un petit pourcentage) peuvent s'épuiser et ils doivent être servis urgent.
- Courriers : Les courriers à long distance ont besoin de rassembler localement les colis sortants avant l'envoi à un terminal distant pour fusionner les charges. En outre, les charges provenant des terminaux distants doivent être distribués localement. La plupart les demandes de collecte sont dynamiques et qui doivent être servies le même jour si possible.
- Sociétés de secours et de service de réparation : il existe plusieurs sociétés fournissent le secours ou des services de réparation (secours des voitures en panne, appareil de réparation, etc.). Cela répond aux demandes de client pour la maintenance ou pour la réparation de ses équipements.
- Systèmes de transport à la demande : Systèmes à la demande fournissent les services de transport aux gens entre une paire donnée d'origine-destination. Les clients peuvent réserver un voyage un jour à l'avance (clients statiques), ou faire une demande à court terme (clients dynamiques).
- Services d'urgence : Les services d'urgence comprennent la police, la lutte contre l'incendie et les services d'ambulance. Par définition, tous les clients sont dynamiques. En plus, le taux de demandes est habituellement bas c'est pour ça les véhicules deviennent oisifs de temps en temps.
- Services de taxi : Dans les services de taxi, presque chaque client est dynamique. Comme aux services d'urgence, le replacement temporaire des véhicules oisifs est un problème.

1.3.7. Variantes de DVRP

Selon Psaraftis [25] les applications du monde réel incluent souvent deux dimensions importantes :

- L'évolution d'information : où dans quelques problèmes les informations disponibles au planificateur peuvent changer pendant l'exécution de tournées, par exemple l'arrivée de nouvelles demandes, la modification de l'ordre de clients, ou le changement de temps de déplacement pour certaines routes qui peut être augmenté en raison de mauvaises conditions météorologiques.
- La qualité d'information : reflète une incertitude possible sur les données disponibles, par exemple la demande d'un certain client, les temps de déplacement entre certaines paires de clients. Ces données peuvent ne pas être connues avec certitude, mais seulement comme des prévisions. D'après ces deux dimensions, quatre catégories de VRP peuvent être identifiées, comme il est récapitulé dans le Tableau 1.2.

		Qualité d'information	
		Connaissance déterministe des données d'entrée	Connaissance stochastique des données d'entrée
Evolution d'information	Entrées connues à priori	Statique et déterministe	Statique et stochastique
	Changement d'entrées au cours du temps	Dynamique et déterministe	Dynamique et stochastique

Tableau 1.2 Taxonomie des problèmes de tournées de véhicules par l'évolution et la qualité des informations disponibles. [26]

A partir de la définition précédente de Psaraftis, nous proposons de classifier les DVRPs selon le degré de connaissance que nous avons sur les données d'entrée du problème et de la qualité des informations disponibles. Un problème dynamique peut être soit déterministe ou stochastique (voir Figure 1.8).

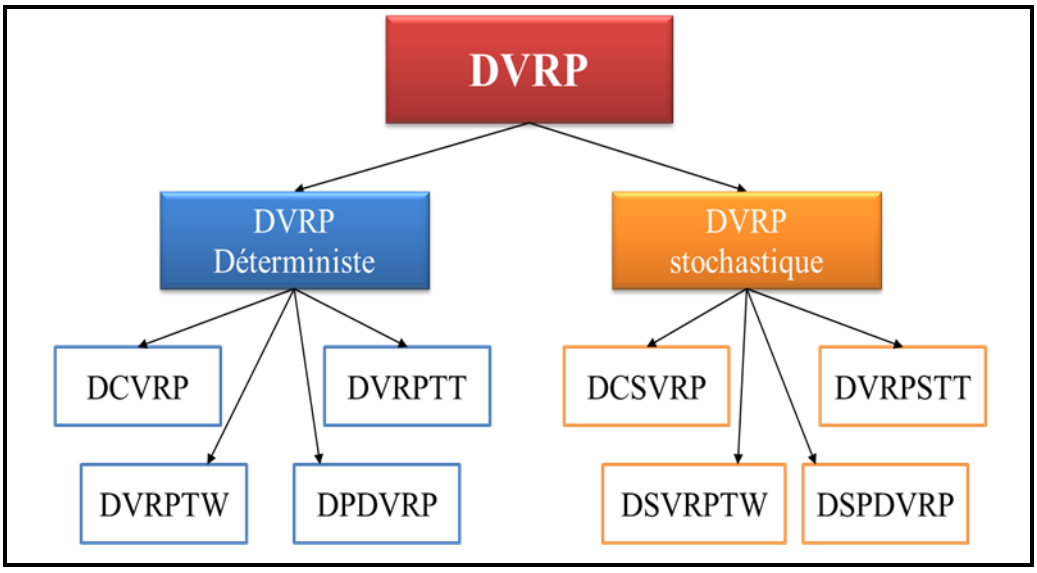

Figure 1.8 Variantes de DVRPs selon la qualité des informations disponibles.

DVRP est déterministe si toutes les données liées aux clients (leurs emplacements) sont connues lorsque les demandes des clients arrivent, sinon il est stochastique. Ces deux classes peuvent être soumises à plusieurs facteurs tels que la fenêtre de temps de service, l'embouteillage, la maintenance des routes, les changements climatiques, la panne des véhicules et ainsi de suite. Ces facteurs changent souvent la vitesse des véhicules le temps de déplacement pour arriver de destination au dépôt. Plusieurs variantes de ce problème ont été définies correspondant à plusieurs contraintes. Elles forment un champ de recherche vaste et très étudié en Recherche Opérationnelle (Figure 1.8).

a. DVRPs Déterministes :

Dans ce cas, toutes les données relatives aux entrées sont connues. Par exemple, lors de l'apparition d'une nouvelle demande d'un client, l'emplacement de ce client et la quantité de sa demande sont connus. Il existe plusieurs types de DVRP déterministe dans la littérature tels que :

- **Dynamic Capacitated Vehicle Routing Problem (DCVRP)**

Comme défini précédemment, un problème de tournées de véhicules problème dynamique de tournées de véhicules avec une contrainte de capacité (CVRP) consiste à affecter chaque client à une tournée effectuée par un seul véhicule de capacité finie. Ce véhicule commence et termine sa tournée au dépôt

Un nombre important de travaux existent sur le DCVRP **[27, 19]** qui représentent la définition conventionnelle du problème, où il existe tous les clients et

leurs localisations sont déterministes, mais leurs demandes peuvent arriver à tout moment. L'objectif est de trouver un ensemble de routes ayant une distance parcourue minimale, en respectant la capacité limitée de véhicule.

- **Dynamic vehicle Routing Problem with Time Windows (DVRPTW)**

Le problème dynamique de tournées de véhicules avec fenêtres de temps représente l'une des variantes les plus étudiées dans **[28, 29, 30, 31]**. Le DVRPTW a pour but de déterminer des tournées de livraisons dans des intervalles horaires prédéfinis. Outre les contraintes du DCVRP, les fenêtres horaires liées aux livraisons constituent la particularité du problème. Dans lequel, chaque client *i* dispose d'une fenêtre temporelle $[a_i, b_i]$ durant laquelle il peut être livré. En poursuivant les notations de la formulation mathématique de DVRP, d'autres variables de décision sont définies : pour $0 \leq i \leq n$, et $1 \leq k \leq m$, s_{ik} désigne l'instant où le véhicule *k* commence à servir le client *i*. Il en résulte une contrainte supplémentaire :

$$a_i \leq s_{ik} \leq b_i \qquad (1.14)$$

La contrainte (1.14) correspond tout simplement à la définition des fenêtres temporelles des clients. Ces dernières peuvent être définies de manière dure ou souple. La souplesse vient du fait d'admettre que les bornes peuvent varier dans une certaine mesure contrairement au cas où elles sont dures.

- **Dynamic Vehicle Routing Problem with time-dependant Travel Times (DVRPTT)**

Le problème dynamique de tournées de véhicules avec des temps de déplacement dépendant du temps est décrit dans **[32, 33]**. Dans ce type de problème les temps de déplacement de client *i* au client *j* sont variables dans le temps. Cette variation pourrait se produire en raison de type de route, les conditions météorologiques et les conditions du trafic qui peuvent fortement influencer la vitesse des véhicules et par conséquent les temps de déplacement.

- **Dynamic Vehicle Routing Problem With Pick-Up And Delivery (DVRPPD)**

Comme définit dans **[34]**, où les véhicules effectuent un double service : la livraison (Delivery) des clients ainsi que le collecte (Pick-up) de marchandises de ces derniers. Lors du service chez chaque client, la livraison et le collecte sont effectuées simultanément, ce qui constitue la principale caractéristique de ce problème. Chaque client *i* est associé à deux quantités de d_i et p_i, $1 \leq i \leq$ n représentant la demande des

marchandises homogènes à livrer et prises au client *i*, respectivement. Parfois, seulement un $d_i = d_i - p_i$ de quantité de demande est employé pour chaque client *i*, indiquant la différence nette entre les demandes de livraison et de collecte (de ce fait elle peut être négatif).

Le collecte induit une difficulté supplémentaire en rapport avec la contrainte de capacité des véhicules. Par convention, lorsqu'un véhicule dessert un client il commence par la livraison puis il effectue la collecte. Donc, la charge actuelle d'un véhicule avant l'arrivée à un emplacement donné est définie par la charge initiale sans toutes les demandes déjà fournie plus toutes les demandes déjà prises. Le dynamisme apparait à ce problème lorsque toutes les demandes ne sont pas connues à l'avance.

b. DVRPs stochastiques :

DSVRP (Dynamic and Stochastic Vehicle Routing Problem) est considéré comme une nouvelle classe de DVRPs, visant à atteindre le traitement approprié des événements dynamiques, en intégrant des données aléatoires où incertaines (emplacements des clients, les demandes, les temps de déplacement des véhicules) sur les événements futurs possibles. Ces données sont représentées par des processus stochastiques.

Flatberg et al. **[35]** et Pillac et al. **[36]** fournissaient un aperçu sur DSVRPs mais ils ont mis l'accent sur les DVRPs purs. Tandis que Ritzinger et Puchinger dans **[37]** passaient en revue les DSVRPs mais avec un accent exclusif sur les diverses méthodes hybrides appliquées dans ce domaine. Lors des dernières années, le domaine de l'optimisation anticipatoire **[38]** (concernant le traitement futur des paramètres pertinents) a été relié également à la prise de décision dynamique. Ritzinger et al. dans **[49]** présentèrent un aperçu complet et une classification détaillée sur la recherche en tenant compte les DSVRPs.

- **Dynamic and Stochastic Capacitated Vehicle Routing Problem (DSCVRP):**

Cette variante considère les demandes des clients comme des inconnues et qui sont indiquées au cours du temps **[40, 41, 42]**. En plus, les emplacements des clients et les temps de service sont des variables aléatoires et ils sont réalisés dynamiquement pendant l'exécution du plan.

- **Dynamic and Stochastic Vehicle Routing Problem with Time Windows (DSVRPTW):**

Dans ce problème **[43]**, chaque demande de service est générée selon un processus stochastique. Lorsqu'une demande de service apparait, elle reste active pour une certaine période du temps déterministe, puis elle expire. L'objectif est de minimiser le nombre de combinaisons de véhicules et d'assurer que chaque demande est visitée avant son expiration.

- **Dynamic Vehicle Routing Problem with Stochastic Travel Times (DVRPSTT):**

On suppose dans le DVRPSTT que le problème est soumis à un temps de parcours stochastique qui représente une variable aléatoire dans un intervalle. Les temps de déplacement changent d'une période à l'autre **[44, 45, 46]**.

- **Dynamic and Stochastic Pickup and Delivery Vehicle Routing Problem (DSPDVRP):**

Dans cette version du problème, nous avons une imprécision pour certaines informations (ex : la quantité de demande que le véhicule doit la ramasser ou livrer à chaque client) sur les demandes futures, qui sont connues comme une distribution de probabilités. Par exemple, les demandes entrantes peuvent suivre une distribution spécifique de Poisson (voir, Swihart et Papastavrou **[47]**) ou un processus de décision markovien (Markov Decision Process, MDP, voir Powell et al. **[48]**). En raison de la complexité du problème, la distribution de probabilité exacte des événements futurs n'est pas toujours connue mais peut être approchée par l'utilisation des données historiques Hvattum et al. **[49]**.

1.4. Conclusion

Dans ce chapitre, on a présenté le principe général du problème dynamique de tournées de véhicules (Dynamic Vehicle Routing Problem, DVRP), ainsi que ces variantes selon un ensemble de contraintes, en se basant sur l'une de ces dernières qui est le problème dynamique de tournées de véhicule avec une contrainte de capacité (Dynamic Capacitated Vehicle Routing Problem, DCVRP), sa formulation mathématique.

Dans le chapitre suivant, nous allons présenter un aperçu général et une synthèse sur les différentes méthodes utilisées pour la résolution du problème de DVRP.

Chapitre 2. Les méthodes de résolution de DVRP (Etat de l'art)

Sommaire

2.1. Introduction

Malgré le progrès des algorithmes et la croissance constante de la puissance de calcul, les approches exactes ne sont capables de résoudre des problèmes de tournées de véhicules qu'avec une centaine de sommets, ce qui est très inférieure à la taille typique des problèmes rencontrés dans l'industrie. De plus, les approches exactes peuvent prendre plusieurs heures pour produire une solution. En conséquence, un certain nombre de méthodes (plus rapides) approximatives ont été développées pour résoudre le DVRP et ses variantes. Dans ce qui suit, nous pouvons diviser ces méthodes en deux grandes classes : les méthodes séquentielles et les méthodes parallèles comme il est montré dans la Figure 2.1.

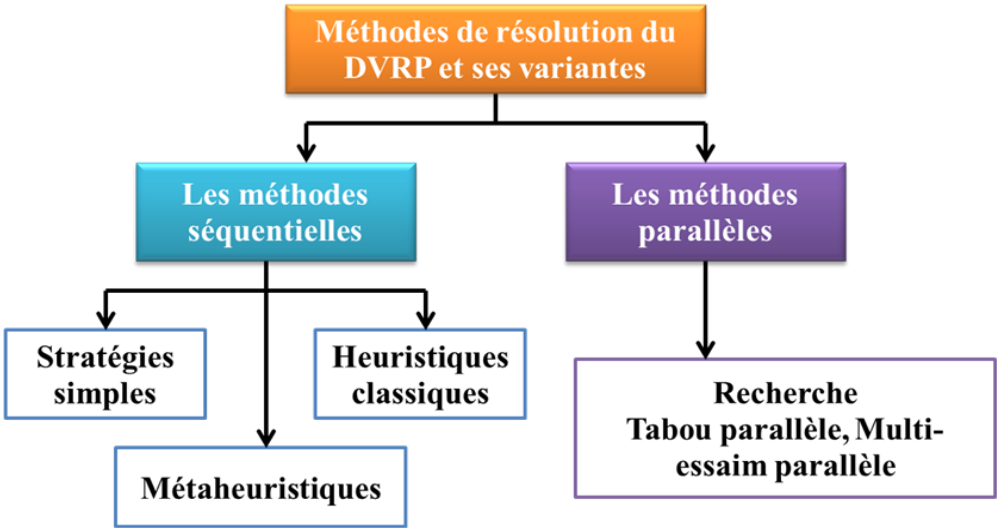

Figure 2.1 Méthodes de résolution du DVRP et ses variantes.

2.2. Les méthodes séquentielles

Les méthodes séquentielles approximatives utilisées pour résoudre le DVRP peuvent être divisés en trois catégories principales : les stratégies simples, les heuristiques classiques, les métaheuristiques.

2.2.1. Stratégies simples

En général, les stratégies de résolution sont représentées sous forme de politiques simples, qui spécifient les actions que le système doit apporter pour gérer l'état actuel et les propriétés du problème dynamique (par exemple, l'emplacement des clients, la fréquence des événements, le nombre de demandes connues à l'avance, la durée du jour de service, la durée de l'horizon de planification, etc.).

Ces stratégies tiennent compte de la répartition spatiale et temporelle des demandes dynamiques et permettent au système de planifier le service chaque fois que le problème change. Elles sont appliquées à plusieurs reprises pour envoyer les demandes aux véhicules et pour construire des routes. Parmi les stratégies les plus connues, nous pouvons citer celles présentées dans les travaux de Larsen **[18]** et Bianchi **[50]** et qui traitent le problème DTRP (Dynamic Traveling Repairman Problem), qui est comme le DVRP dans sa version simple on a un " voyageur de réparation, Traveling Repairman en anglais" ou un véhicule doit servir un ensemble de clients apparaissent dynamiquement au cours du temps. Dans sa version complexe, il y a *m* véhicules avec une capacité limitée *q* (*q*= un nombre maximum de clients à servir avant de retourner au dépôt).

a. Premier arrivé premier servi (First Come First Served, FCFS) : Les demandes des clients sont traitées dans l'ordre dans lesquelles elles sont parvenues au planificateur.

b. Médiane stochastique de la file d'attente (Stochastic queue median, FCFS-SQM) : La politique FCFS-SQM est une modification de la politique FCFS. Selon cette politique, le véhicule se positionne dans la médiane de la région de service, puis il quitte directement cette médiane pour servir les clients selon la stratégie FCFS. Une fois le service terminé, le véhicule retourne à la médiane et attend la prochaine demande.

c. Voisin le plus proche (Nearest Neighbor, NN) : Après avoir terminé le service d'un client, le véhicule passe au client voisin (le plus proche) non encore servi.

d. Stratégie de problème de voyageur de commerce (Travelling Salesman Problem strategy, TSP) : Lorsque les demandes sont reçues, des ensembles composés de n clients sont formés et mis dans une file d'attente. Après, chaque ensemble est traité séparément comme étant un TSP.

e. Stratégie TSP modifiée (Modified TSP strategy, mod TSP) : L'espace de service est subdivisé en plusieurs zones, Pour un nombre entier définie k. La stratégie TSP est appliquée pour chaque zone. Dans ce cas, une file d'attente est utilisée pour déterminer l'ordre selon lequel les zones seront traitées comme il est montré dans la Figure 2.2.

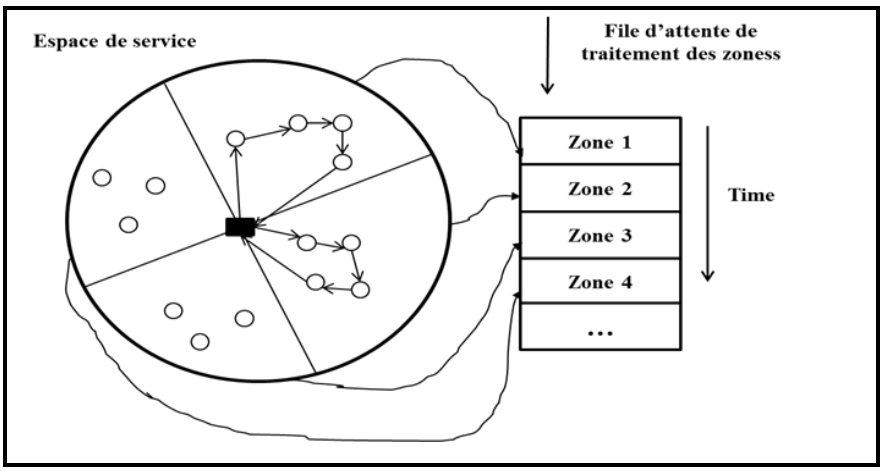

Figure 2.2 La stratégie mod TSP [50].

f. Partitionnement (Partitioning, PART) : La zone de service est divisée en sous-régions. Un véhicule visite les sous-régions dans un ordre donné, en passant d'une sous-région à une sous-région adjacente. Les demandes locales dans chaque sous-région sont traitées dans un ordre FCFS (voir Figure 2.3).

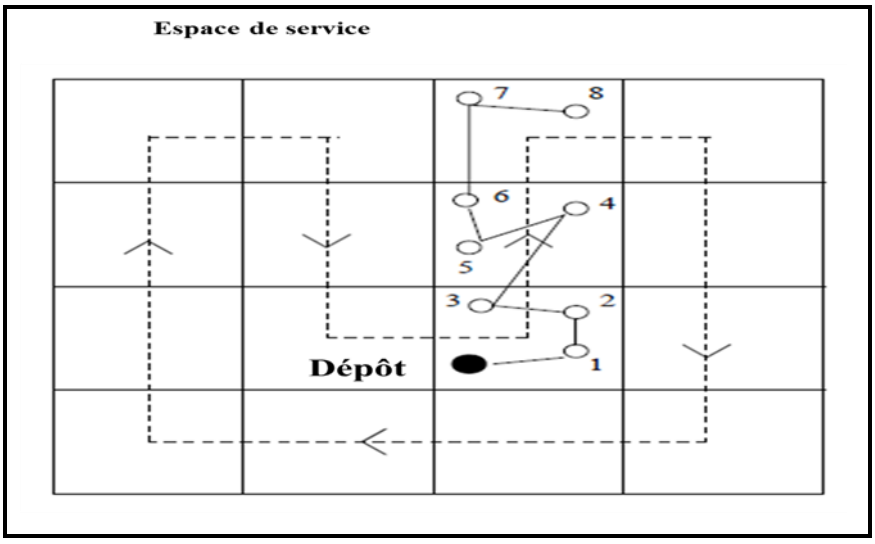

Figure 2.3 La stratégie PART [50].

g. Stratégie de génération (Generation strategy, GEN) : Cette stratégie a été utilisée pour la résolution du DTRP avec un seul véhicule, elle combine entre les deux stratégies SQM et TSP. Son principe est le suivant (voir Figure 2.4) :

-Etape 1 : Le véhicule est positionné dans la médiane de son espace de service.

-Etape 2 : Lors de l'arrivé d'une nouvelle demande, le véhicule se déplace pour le servir (cette première demande forme la première génération).

-Etape 3 : Une fois tous les clients d'une génération servis, on vérifie s'il n'y a pas d'autres demandes reçues dans la file d'attente, en revenant à l'étape1. Sinon, la stratégie TSP est appliquée sur cet ensemble de demandes qui formeraient une nouvelle génération à servir.

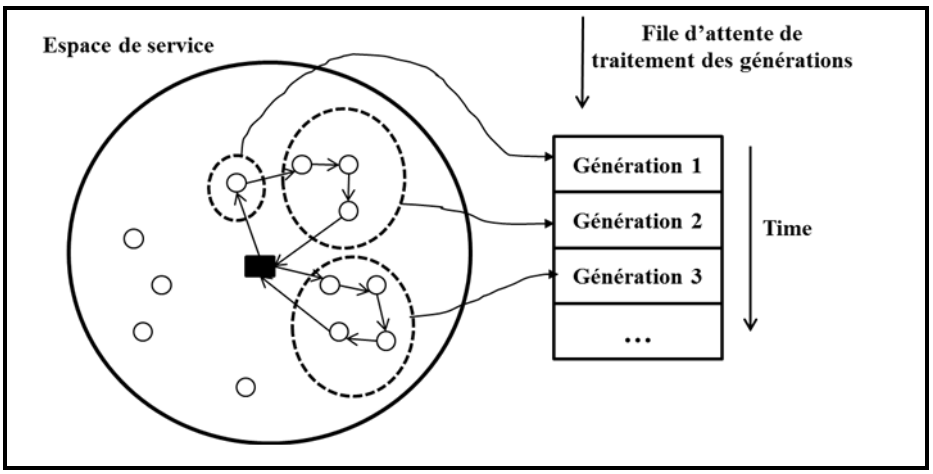

Figure 2.4 La stratégie GEN [50].

h. Courbe de remplissage de l'espace (Space Filling Curve, SFC) : Cette stratégie a été utilisée pour la résolution du DTRP avec un seul véhicule. Dans laquelle, les demandes sont servies comme elles sont rencontrées pendant des balayages répétées dans le sens horaire d'un cercle C, qui couvre l'espace de service.

2.2.2. Les heuristiques classiques

Les stratégies simples sont efficaces avec des problèmes spécifiques comme le problème du DTRP, mais la résolution du DVRP nécessite l'utilisation des heuristiques plus performantes. Les heuristiques classiques sont divisées en deux catégories : les heuristiques constructives, et les heuristiques d'amélioration.

a. Les heuristiques constructives : Une des premières heuristiques constructives pour le VRP était l'algorithme de gain de Clarke et Wright **[51]** qui commence par une solution initiale composée de tournées aller-retour c.-à-d. créer une tournée pour chaque client. Puis itérativement fusionne les tournées jusqu'à ce que la distance totale ne puisse plus être réduite. Comme avec cet exemple illustré dans la Figure 2.5, où on fusionne une tournée qui se termine par l'emplacement i avec une autre tournée qui commence par l'emplacement j, en maximisant le gain S_{ij}, tel que $S_{ij} = c_{i0} + c_{0j} - c_{ij}$ et c_{ij} est le coût de déplacement de l'emplacement *i* vers l'emplacement *j*, où l'emplacement 0 est le dépôt. Ce processus s'arrête lorsque ne reste aucune tournée faisable puisse être fusionné.

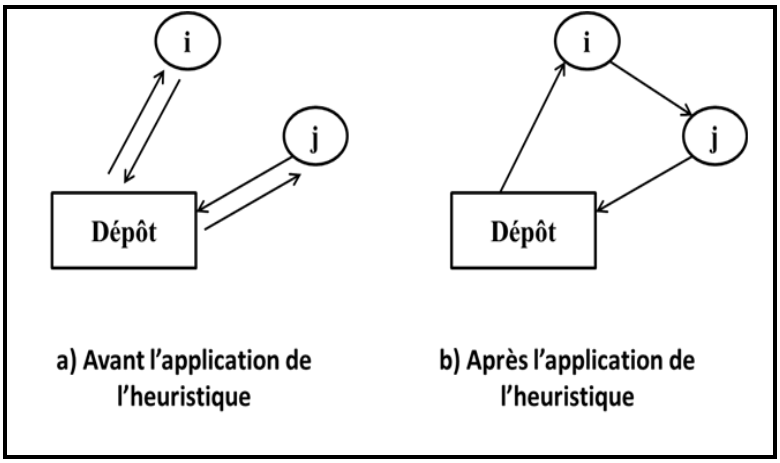

Figure 2.5 Heuristique de gain de Clarke et Wright [51].

Les heuristiques en deux phases incluent les heuristiques **cluster-first, route-second (CR)** et **route-first, cluster-second (RC)**. L'heuristique **CR** a été introduite par Fisher et Jaikumar **[52]**, et qui consiste à regrouper les clients en groupes sans violer la capacité du véhicule, puis à résoudre un TSP pour chaque groupe. L'heuristique **RC**

présentée par Prins et al. dans **[53]** commence par la conception d'une tournée géante en visitant tous les clients, et qui est ensuite divisée en un ensemble de routes faisables.

Les heuristiques d'insertion sont largement utilisées pour la résolution de problème VRP dynamique, et qui sont des algorithmes purement constructifs. Elle sert à ré-optimiser simplement les tournées de véhicules quand les nouvelles données deviennent disponibles. En cherchant à insérer le nouveau client dans la meilleure position des tournées courantes **[54].**

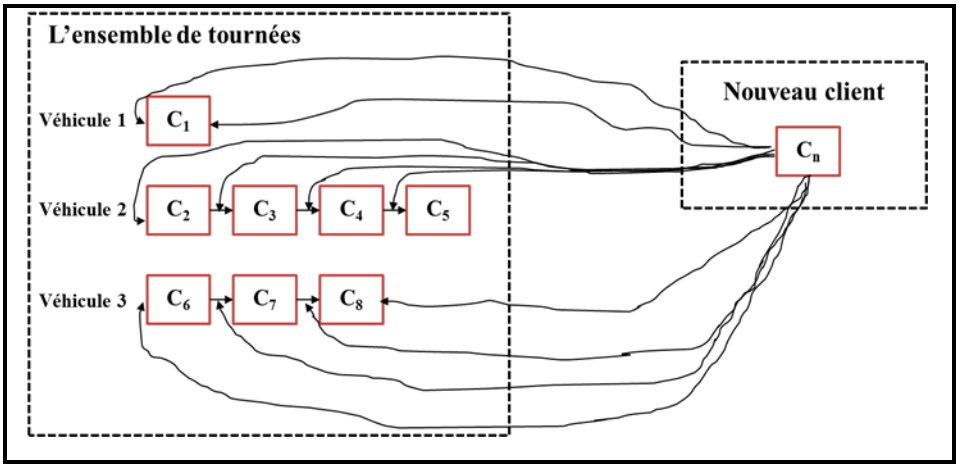

Figure 2.6 Heuristique d'Insertion.

La Figure 2.6 illustre le principe d'une heuristique d'insertion classique, dans cette figure nous remarquons que l'heuristique cherche à insérer le nouveau client dans la meilleure position des tournées courantes, en testant toutes les possibilités d'insertion du nouveau client C_n. Pour cela elle parcourt toutes les positions d'insertion possibles dans la matrice des tournées. Cette matrice ne contient que les clients non servis, les clients déjà visités n'y sont pas représentés.

L'heuristique du plus proche voisin construit une tournée à la fois ; la tournée courante débute au dépôt, visite à chaque fois le client disponible le plus proche et ne retourne au dépôt que lorsqu'aucun client supplémentaire ne peut y être ajouté, puis une nouvelle tournée débute. Cette heuristique est adaptée aux instances dont les clients sont regroupés par paquets (ou groupes). Elle est beaucoup moins efficace sur des instances dont les clients sont éparpillés tout autour du dépôt.

b. Les heuristiques d'amélioration : Ces heuristiques tentent d'améliorer une solution produite par une heuristique constructive. Parmi ces heuristiques, on cite la recherche locale qui a pour but de trouver les meilleures solutions dans le voisinage d'une solution donnée, en effectuant des transformations ou des mouvements qui réduisent le coût total et elle s'arrête dès qu'il n'existe plus de mouvement améliorant.

Chaque type de mouvement représente un voisinage de la solution considérée. Il existe deux types d'opérateurs de mouvement, les mouvements intra-route et les mouvements inter-routes. Les mouvements intra-route consistent à améliorer chaque tournée séparément, comme le mouvement k-opt **[55]**, et Or-opt **[56]**.

Un mouvement k-opt ou k-échange consiste à enlever *k* arêtes non consécutives d'une tournée et les remplacer par *k* autres arêtes de manière à constituer une nouvelle solution réalisable pour le problème traité. En général $k = 2$ ou $k = 3$, car l'exploration du voisinage *k-opt* (recherche du meilleur mouvement et application de ce mouvement) s'effectue en $O(n^k)$.

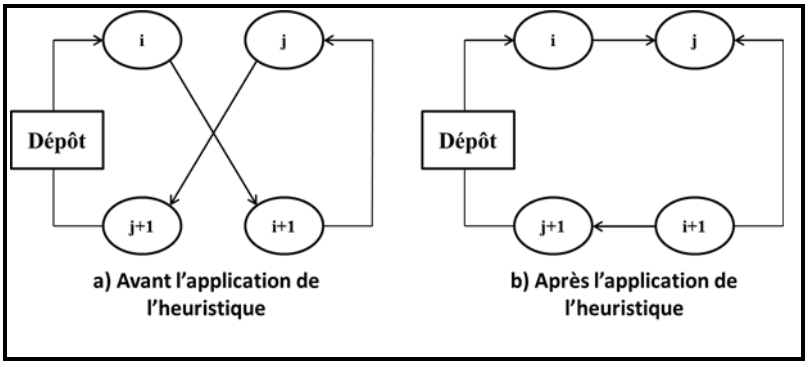

Figure 2.7 Heuristique 2-opt [55].

Un mouvement Or-opt-1 considère chaque demande à son tour et essaie d'améliorer la solution en réinsérant cette demande à un autre emplacement, comme il est illustré dans la Figure 2.8. L'heuristique Or-opt étend cela en considérant des séquences de 1, 2 et 3 emplacements adjacents dans une solution.

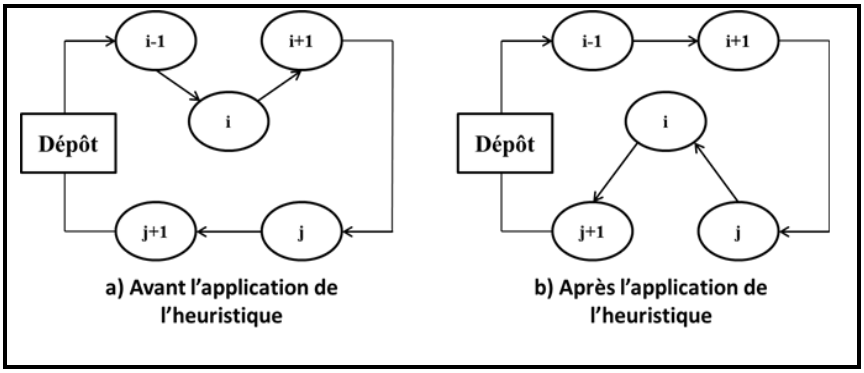

Figure 2.8 Heuristique Or-opt [56].

Alors que les mouvements inter-routes agissent sur plusieurs tournées simultanément, comme les mouvements CROSS de Taillard et al. **[57]**, et GENI de Gendreau et al. **[58]**.

L'idée de base de l'opérateur de mouvement CROSS est de retirer d'abord deux arêtes (i-1, i) et (k, k + 1) de la première route, et deux autres arêtes (j-1, j) et (L, L + 1)

de la deuxième route. Alors les segments i-k et j-l, qui peuvent contenir un nombre arbitraire de clients, sont échangés en introduisant les nouvelles arêtes (i-1, j), (L, k + 1), (j-L, i) et (k, L + 1). Notons que l'orientation des deux routes est conservée, comme illustré dans la Figure 2.9.

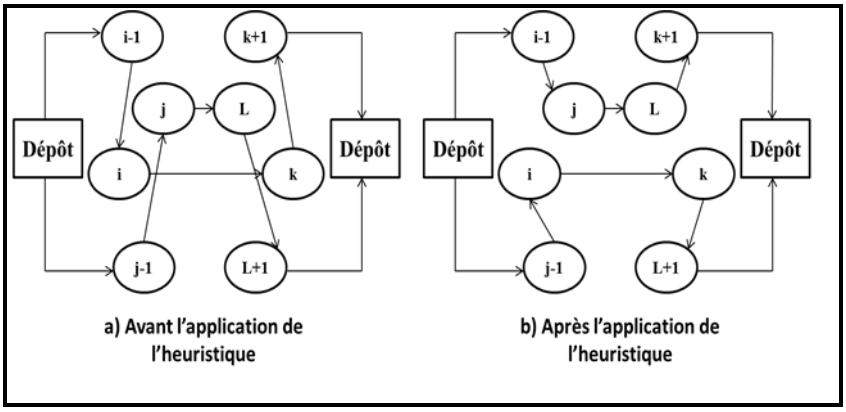

Figure 2.9 Heuristique CROSS.

Avec l'opérateur de mouvement GENI, un client d'une route peut être inséré entre les deux nœuds clients les plus proches sur la route de destination, même si ces nœuds clients ne sont pas consécutifs. Dans la Figure 2.10, le client i sur la route supérieure est inséré dans la route inférieure entre les clients j et k les plus proche en ajoutant les arêtes (j, i) et (i, k). Puisque j et k ne sont pas consécutifs, il faut réorganiser la route inférieure. Ici, la tournée faisable est obtenue en supprimant les arêtes (j, j + 1) et (k-1, k) et en relocalisant le chemin {j + 1, ..., k-1}.

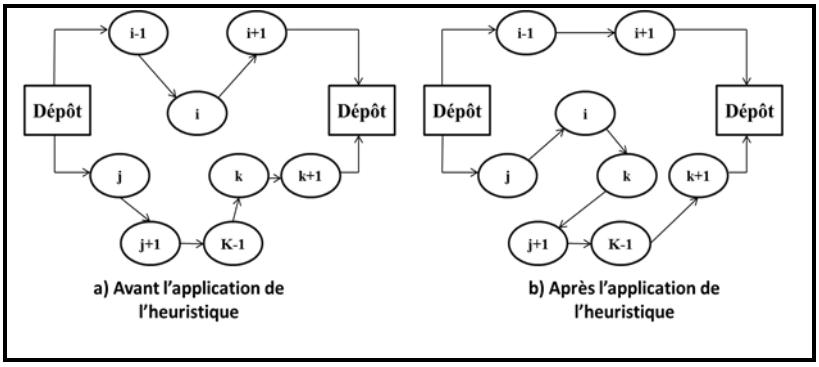

Figure 2.10 Heuristique GENI.

Kilby et al. dans **[27]** divisent le temps de simulation (le temps de la journée de service) en tranches (50 au total). Durant chaque tranche, le planificateur collecte les nouvelles demandes et les traite dans la prochaine tranche de temps. Autrement dit, le planificateur utilise la méthode d'insertion pour trouver les meilleures positions pour les nouvelles demandes reçues au cours de la tranche précédente. Dans le même temps, ce

planificateur essaie d'améliorer la solution obtenue en utilisant des heuristiques différentes comme 2-Opt, Or- Opt.

Mitrovic-Minic et al. **[59]** a introduit le concept de double horizon (court terme et moyen terme) pour la résolution du problème dynamique de collecte et de livraison de colis avec fenêtres de temps DPDPTW. Lorsqu'une nouvelle demande arrive, ils appliquent l'heuristique d'insertion la moins chère afin de déterminer les meilleures positions globales d'insertion pour cette demande avant son insertion. L'heuristique d'amélioration utilisée est basée sur la recherche tabou (Tabu Search, TS), elle est appliquée après l'heuristique de réinsertion et il s'exécute lorsque les nouvelles demandes sont reçues.

Branchini et al. **[60]** proposent une heuristique de recherche locale granulaire adaptative (adaptive granular local search, en anglais) pour résoudre le DVRP. Elle est basée sur une stratégie de liste de candidats qui a un voisinage réduit. Ce voisinage évite les segments de tournée longs qui représentent des arcs à coûts élevés en raison de la faible probabilité que ceux-ci font partie de solutions de bonne qualité et se concentre sur des segments de tournée courts. Cette heuristique suppose d'ajuster la taille de l'espace de recherche en fonction du temps court disponible pour la méthode d'optimisation en différentes périodes de la journée de service.

2.2.3. Les méthaheuristiques

Les métaheuristiques sont des paradigmes d'optimisation, dont l'objectif principal est de surmonter les limitations des heuristiques classiques, en particulier leur possibilité de tomber dans l'optimum local et leur manque de robustesse. Le mécanisme permettant à la métaheuristique de trouver l'optimum local de la solution courante est appelé intensification et se ramène souvent à une recherche locale. Par contre le mécanisme permettant de sortir de la région de ce minimum local[3] est appelé diversification et varie d'un algorithme à l'autre.

Dans le contexte des problèmes dynamiques, des informations critiques sont révélées au cours du temps, ce qui signifie que la définition complète d'une instance d'un problème donné n'est connue qu'à la fin de l'horizon de planification. Donc, une solution optimale à un instant *t* ne peut être trouvée que par la connaissance de toutes les entrées à cette même période, et cette solution pourrait être non-optimale à l'instant *t+1*, ce qui nécessite un mécanisme de mise à jour efficace et rapide, lors de l'arrivée de

[3] Une solution *s* est minimum local par apport à une structure de voisinage *N* si $\forall s' \in N(s)$, $f(s) \leq f(s')$.

nouvelles entrées. Par conséquent, la majeure partie de la recherche dans le domaine du routage dynamique des véhicules est basée sur des métaheuristiques au sens général, qui donnent une bonne solution dans un temps de calcul raisonnable. Ces métaheuristiques peuvent être classifiées en deux grandes familles : les métaheuristiques à solution unique, et les métaheuristiques à population de solutions.

a. Les métaheuristiques à solution unique qui tentent itérativement d'améliorer une solution sont appelées méthodes de recherche locale. Ils ont tendance à intensifier la recherche en exploitant une partie de l'espace de recherche. Parmi ce type de métaheuristiques on peut citer :

1) La recherche Tabou (Tabu Search , TS)

La recherche Tabou a été proposé par Glover **[61]**, elle devenu est rapidement l'une des meilleures méthodes de recherche locales et les plus répandues pour l'optimisation combinatoire. La méthode effectue une exploration de l'espace de solution en passant d'une solution x_t identifiée dans l'itération t à la meilleure solution x_{t+1} dans un sous-ensemble du voisinage $N(x_t)$ de x_t. Puisque x_{t+1} ne s'améliore pas nécessairement sur x_t, un mécanisme de tabou est mis en place pour empêcher le processus de faire un cycle sur une suite de solutions. Une façon incomparable d'empêcher les cycles est d'interdire le processus de revenir aux solutions précédemment rencontrées.

Mitrovic-Minic **[62]** dans sa thèse étudie le problème dynamique de collecte et de livraison à grande échelle avec des fenêtres de temps (DPDPTW), et il propose des approches de solutions heuristiques. Il a utilisé la recherche tabou couplée avec la méthode d'insertion à moindre coût. Ces travaux ont utilisé la notion de découpage de l'horizon en deux parties, le but étant d'optimiser plus finement la partie à court terme qui est la partie de collecte de colis, que la partie à long terme qui est également la partie livraison de colis. Les principaux résultats de sa recherche incluent: un modèle dynamique à deux buts pour un problème de routage dynamique, une heuristique à deux stratégies (un cadre dans lequel n'importe quelle heuristique ou métaheuristique peut être intégrée), une fonction d'utilité dynamique pour l'évaluation d'une insertion , reconnaissant l'importance de développer un planning dans un environnement dynamique (par opposition à un environnement statique), l'analyse de stratégies d'attente simples, la conception de stratégies d'attente complexes pour résoudre le problème d'ordonnancement et l'utilisation de graphes de précédence pour tester la faisabilité

d'insertion d'une demande dans une tournée dans le problème de ramassage et de livraison avec des fenêtres de temps. (Les graphiques de précédence peuvent également être utilisés pour trouver des limites sur le nombre de véhicules pour le problème du voyageur de commerce multiple avec des fenêtres de temps.)

Hanshar et al. **[63]** a implémenté une recherche tabou de base, dans laquelle deux opérateurs sont employés pour structurer le voisinage; l'opérateur d'inversion et l'opérateur d'échange λ de Osman **[64]**, chacun a été appliqué selon une certaine probabilité. Premièrement, il y avait 0.6 chance d'appliquer l'opérateur d'inversion, et 100 voisins ont été produits. La probabilité de 0.4 restante signifiait que nous appliquions l'échange λ de Osman [38], un standard opérateur de recherche locale qui a été appliqué dans divers travaux de VRP. L'opérateur d'échange λ a été utilisé pour les combinaisons suivantes :

(1, 0), (1, 1), (2, 0), (2, 1), (2, 2), (3, 0), (3, 1), (3, 2), (3 , 3).

Par exemple, l'opérateur (1, 0) sur les routes (Rp, Rq) génère un voisinage en permutant un client de la route p et en le plaçant dans la route q. Chaque position dans la route q est considérée. Ce processus est exécuté pour chaque client dans une route et chaque paire de routes dans une solution pour générer le voisinage entier de (1, 0). Les paramètres utilisés pour la recherche de tabous étaient les suivants : liste de tabous de 25, 800 itérations et 10 exécutions. Les résultats obtenus par la recherche taboue ont trouvé 8 des 21 nouvelles meilleures solutions.

2) La recherche à voisinage variable (ou Variable Neighborhood Search, VNS)

VNS a été développée originalement pour résoudre les problèmes combinatoires statiques par Hansen et Mladenovic **[65]**. L'idée principale de la recherche à voisinage variable (VNS) est d'améliorer itérativement une solution en considérant successivement différents voisinages. Un aperçu de l'approche est donné par l'Algorithme 2.1. L'algorithme commence par une solution initiale X, et il génère un voisin X^t à partir du voisinage actuel (ligne 6), ce qui est ensuite amélioré par une procédure de recherche locale (ligne 7). Si la nouvelle solution est acceptée (Ligne 8), elle remplace la solution actuelle et une nouvelle itération est effectuée avec le premier voisinage (ligne 10); Sinon, le voisinage suivant est choisi (ligne 12) et une nouvelle itération est effectuée avec la solution courante non modifiée. Comme pour les autres

heuristiques, les itérations sont effectuées jusqu'à ce qu'un critère d'arrêt soit atteint, généralement un temps maximum ou le nombre d'itérations. VNS a été appliquée dans un contexte dynamique par :

- Xu et al. **[66]** Afin de résoudre efficacement le problème de routage dynamique de véhicule avec des fenêtres de temps (DVRPTW), le modèle mathématique est établi et un algorithme de recherche de voisinage variable amélioré est proposé. Dans l'algorithme, les clients d'allocation et les itinéraires de planification pour la solution initiale sont complétés par la méthode de clustering. Les opérateurs hybrides d'insertion et d'échange sont utilisés pour réaliser le processus d'agitation, le processus d'optimisation ultérieur est présenté pour améliorer l'espace de solution, et la stratégie d'amélioration optimale est adoptée, ce qui permet un meilleur équilibre de la qualité de la solution et le temps d'exécution. L'idée de recuit simulé est introduite pour prendre le contrôle de l'acceptation de nouvelles solutions, et les influences de l'heure d'arrivée, de la distribution de la localisation géographique, et de la plage de temps sur la sélection du trajet sont analysées. Dans l'expérience, l'algorithme proposé est appliqué pour résoudre les problèmes de DVRP des différentes tailles. La comparaison avec d'autres algorithmes sur les résultats montre que l'algorithme est efficace et faisable.

- Le travail De Armas et Melian-Batista **[67]** pour résoudre le DVRPTW. Dans celui-ci le VNS conçu gère deux possibilités : rejeter les clients qui ne peuvent pas être insérés de manière réalisable en tenant compte des priorités, ou autoriser les fenêtres de temps dans les clients afin de fournir une solution à tous les clients. De plus, elles proposent des solutions avec des infaisabilités afin d'inclure tous les clients dans les solutions finales. Dans ce cas, logiquement, la distance totale augmente, mais leurs résultats sont encore très proches des meilleurs dans la littérature. Il est important de noter qu'elles considèrent le temps nécessaire pour insérer un nouveau client dynamique dans le plan, ce qui peut influencer les résultats finaux. Enfin, elles ont également analysé l'effet des différentes restrictions dans les solutions finales en utilisant des instances basées sur les vraies fourni par une entreprise. Dans ce cas, l'importance des priorités des clients sur le plan final est devenue claire.

Algorithme 2.1 La recherche à voisinage variable (ou Variable Neighborhood Search, VNS) [26]

Entrée : X une solution valide, z une fonction d'évaluation, et $N = \{N_1, .., N_K\}$ Un ensemble de structures de voisinage.
Sortie: X^* la meilleure solution trouvée
1: **fonction** VNS(X)

```
2:    X*←X;
3:    répéter
4:        k ← 1; . Select first neighborhood
5:        répéter
6:            X^t ← shake (N,X);              // Générer un voisin de l'ensemble N
7:            X^t ← ls(X^t);                  //  Recheche Locale pour améliorer X^t
8:              si accept(X^t,X) alors        // X^t est acceptée comme un solution
actuelle
9:                X ← X^t;                    //Mettre à jour la solution actuelle
10:               k ← 1;                      //Redémarrer du premier voisinage
11:           sinon
12:               k ← k + 1;                  //Aller vers le voisinage suivant
13:           fin si
14:           si z(X^t) < z(X*) alors         //Une Amélioration a été trouvée
15:               X* ← X^t;                   // Mettre à jour la meilleure solution
16:           fin si
17:       jusqu'à k = K;
18:   jusqu'à Critère d'arrêt ();
19:   retourner X* ;
20: fin fonction
```

3) La recherche à voisinage large (ou Large Neighborhood Search, LNS)

Large Neighborhood Search (LNS) a été introduit pour la première fois par Shaw **[68]** et elle considérée comme un cas particulier de VNS **[69]**. L'algorithme 2.2 présente la structure générale de LNS. L'exploration se fait en détruisant d'abord la solution courante, puis en la réparant (ligne 6). Le voisin résultant X^t est alors accepté ou rejeté en fonction d'un critère spécifique (ligne 7). Dans l'implémentation originale de Shaw **[68]**, la méthode de destruction enlève un sous-ensemble de clients des tournées, tandis que la fonction de réparation les réinsère en utilisant des heuristiques et une propagation avec contrainte. D'autre part, elle utilise un critère simple d'acceptation, acceptant uniquement les solutions améliorées. L'idée de base de LNS est d'explorer efficacement un voisinage exponentiel. En fait, avec n clients, il y a 2^n sous-ensembles candidats à supprimer et à réinsérer. Dans l'implémentation de Shaw **[68]**, ceci est réalisé par la fonction de destruction qui considère des sous-ensembles de cardinalité augmentée itérativement, en augmentant la taille de voisinage chaque fois qu'un nombre donné de voisins consécutifs non améliorés sont trouvés. La recherche à voisinage large a été adaptée au routage dynamique dans l'étude de Hong **[70]** pour résoudre le DVRPTW, en décomposant ce problème en une série de problèmes statique de VRPTW, puis LNS est proposé pour résoudre ces derniers. En utilisant le processus de suppression-réinsertion du LNS, les nœuds clients les plus récents sont considérés

comme une partie des nœuds supprimés ; Ces nœuds peuvent être insérés dans la solution originale ou actuelle en temps utile dans le processus de réinsertion.

Algorithme 2.2 la recherche à voisinage large (the Large Neighbourhood Search generic algorithm, LNS) [26]

Entrée : X une solution valide, z une fonction d'évaluation
Sortie: X* La meilleure solution trouvée
1: **fonction** LNS(X)
2: $X^* \leftarrow X$;
3: $i \leftarrow 0$; // Le nombre d'itérations
4: $k \leftarrow 0$; // Le nombre d'itérations non améliorées
5: **répéter**
6: $X^t \leftarrow$ repair(destroy(X, i, k)); //Générer un voisin
7: **si** accept(X^t,X) **alors** // X^t est acceptée comme une solution courante
8: $X \leftarrow X^t$; // Mettre à jour la solution courante
9: **finsi**
10: **si** $z(X^t) < z(X^*)$ **alors** //Une amélioration est trouvée
11: $X^* \leftarrow X^t$; // Mettre à jour la meilleure solution
12: $k \leftarrow 0$; // Réinitialiser le nombre d'itérations non améliorées
13: **sinon**
14: $k \leftarrow k + 1$; // Mettre à jour le nombre d'itérations non améliorées
15: **finsi**
16: $i \leftarrow i + 1$;
17: **jusqu'à** stopCriterion(i, k);
18: retourner X^*;
19: **fin fonction**

4) La recherche adaptative à voisinage large (The Adaptive Large Neighborhood Search, ALNS)

L'algorithme ALNS, initialement proposé par Pisinger et Ropke **[71]**, est une extension de l'algorithme LNS (Large Neighborhood Search) **[68]**. LNS travaille en détruisant successivement (en supprimant les clients) et en réparant (en réinsérant les clients) une solution actuelle, en utilisant des opérateurs de destruction et de réparation. ALNS ajoute une couche adaptative qui sélectionne de manière aléatoire les opérateurs en fonction de leurs performances passées, en ajustant automatiquement l'algorithme à l'instance en question.

Algorithm 2.3 la recherche adaptative à voisinage large (the Adaptive Large Neighbourhood Search, ALNS) [31]

Entrée : X_0 solution initiale, z fonction d'évaluation, Θ^-/ Θ^+ ensemble d'opérateurs de destruction / réparation, I nombre d'itérations
Sortie: X^* La meilleure solution trouvée
1: $X^* \leftarrow X_0$; // Initialiser la meilleure solution
2 : $X \leftarrow X_0$; // Initialiser la solution courante

3: **Pour** *I* itérations **faire**
4: $d \leftarrow select\ (\Theta^{-})\ ;\ r \leftarrow select\ (\Theta^{+})$; // Choisir détruire / réparer
5: $X' \leftarrow r(d\ (X))\ ;$ // Générer un voisin
6: **si** accept(X',X) **alors** // X' est acceptée comme une solution courante
7: $X \leftarrow X'$; // Mettre à jour la solution courante
8: **fin si**
9: **si** $z(X') < z(X^*)$ **alors** //Une amélioration est trouvée
10: $X^* \leftarrow X'$; // Mettre à jour la meilleure solution
11: **fin si**
12: updatescore(d,r, X') ; // Mettre à jour les scores
13 : **Fin pour**
14: retourner X^*;
15: **fin fonction**

L'algorithme 2.3 présente les grandes lignes de l'approche ALNS. ALNS commence avec une solution initiale $X_{0.}$ Ensuite, pour les *I* itérations, l'algorithme choisit les opérateurs de destruction et de réparation (ligne 4) avec la roue de roulette qui reflète leur performance passée. Les opérateurs de destruction suppriment un sous-ensemble de clients de la solution actuelle, tandis que les opérateurs de réparation les réinsèrent à l'aide d'heuristiques appropriées au problème en question (ligne 5). La nouvelle solution obtenue est conditionnellement acceptée comme solution actuelle selon un critère de recuit simulé (ligne 6). À la fin de chaque itération, les scores des opérateurs de destruction et de réparation sont mis à jour en fonction de la solution qu'ils ont générée (ligne 12).

b. Les métaheuristiques à population de solutions

Ces métaheuristiques améliorent une population de solution à chaque itération, donc elles ont une tendance à diversifier la recherche en explorant différentes parties de l'espace de recherche. Elles commencent à partir d'une population initiale de solutions. Ensuite, elles appliquent itérativement la génération d'une nouvelle population et le déplacement de la population actuelle. Dans la phase de génération, une nouvelle population de solutions est créée. Dans la phase de remplacement, une sélection est effectuée à partir des populations actuelles et nouvelles. Ce processus se déroule jusqu'à un critère d'arrêt donné. Les phases de génération et de remplacement peuvent être sans mémoire. Dans ce cas, les deux procédures sont basées uniquement sur la population actuelle. Sinon, un certain historique de la recherche stocké dans une mémoire peut être utilisé dans la génération de la nouvelle population et le remplacement de l'ancienne population.

La plupart des métaheuristiques à population de solutions sont des algorithmes inspirés de la nature, qui comprennent : les algorithmes génétiques (AG), les algorithmes de colonies de fourmis (Ant Colony Optimization, ACO), et l'algorithme d'essaim de particules (Particle Swarm Optimization, PSO)

1) Les Algorithmes génétiques (AG)

Les algorithmes génétiques dans des contextes dynamiques sont très semblables à ceux conçus pour les problèmes statiques : une population d'individus représentant les plans de routage est maintenue et soumise à des croisements, mutations et sélection. La principale différence est que les solutions sont en constante adaptation aux changements apportés à l'entrée, lors d'exécution de l'AG tout au long l'horizon de planification.

Dans un premier travail, Benyahia et Potvin **[73]** ont étudié le problème dynamique de collecte et de livraison (Dynamic Pickup and Delivery Problem, D-PDP), et ils ont proposé un algorithme génétique modélisant le processus décisionnel d'un distributeur humain. Dans ce travail, chaque véhicule est associé à un vecteur de valeurs d'attribut qui décrit sa situation actuelle par rapport aux nouvelles demandes de service entrantes. En utilisant cette description d'attribut, une fonction d'utilité visant à approximer le processus de décision d'un distributeur professionnel est construite par programmation génétique. Les résultats computationnels sont rapportés sur les demandes collectées auprès d'une société de services de courrier, et une comparaison est fournie avec un modèle de réseau de neurone et une politique de distribution simple.

Plus récemment, les algorithmes génétiques ont été utilisés pour le même problème par (Cheung et al. **[74]**, Haghani et Jung **[75]**), et pour le DVRP par Van Hemert et Poutré **[76]**. Dans ce dernier travail, chaque individu représente un plan pour tous les véhicules de système. C'est une liste de routes, où chaque route correspond exactement à un véhicule. Cette représentation est ensuite décodée en une solution valide, c'est-à-dire une solution dans laquelle aucune des contraintes n'est violée. Les assignations qui violent une ou plusieurs contraintes de temps sont ignorées par le décodeur. Lorsqu'un véhicule atteint sa capacité ou lorsque l'ajout de plusieurs affectations viole une contrainte de temps, le décodeur force une visite au dépôt. Concernant les opérateurs de variation, un seule est considérée qui est la mutation. Deux véhicules, éventuellement le même, sont choisis de manière aléatoire uniforme. Dans les deux véhicules, deux nœuds sont sélectionnés de façon aléatoire uniforme. Si un seul

véhicule est choisi, ces nœuds sont choisis pour être distincts. Ensuite, les nœuds sont échangés.

Housroum avait présenté dans sa thèse **[77]** une métaheuristique basée sur les algorithmes génétiques pour résoudre le DVRPTW. Cette méthode consiste à résoudre le problème directement « en ligne » (On-line), c'est-à-dire qu'elle traite toute nouvelle demande au moment où elle apparaît.

2) Les colonies de fourmis (Ant Colony Optimization, ACO)

Montemanni et al. **[19]** ont développé une métaheuristique basée sur les colonies de fourmis pour résoudre le DVRP. Leur approche utilise les tranches de temps, comme il est introduit par Kilby et al. **[27]**, qui divisent l'horizon de planification *T* en *nts* périodes de durée égale à *T/nts*. Le traitement d'une demande arrivant pendant une tranche de temps est reporté jusqu'à la fin de celui-ci, de sorte que le problème résolu pendant une tranche de temps considère uniquement les demandes connues dès son début. Par conséquent, l'optimisation est exécutée indépendamment pendant chaque tranche de temps, résolvant un problème statique. L'avantage principal de cette partition de temps est que l'effort de calcul autorisé soit similaire pour chaque tranche de temps, alors que si l'optimisation est effectuée à l'arrivée de chaque nouvelle demande, il n'y aura aucune garantie sur le temps de calcul disponible. Cette discrétisation est également possible par la nature des demandes, qui ne sont jamais urgentes, et peuvent être reportées. Une caractéristique intéressante de leur approche est l'utilisation de la trace de phéromone pour transférer les caractéristiques d'une bonne solution à la tranche suivante.

3) Les essaim de particules (Particle Swarm Optimization, PSO)

L'optimisation par des essaims de particules est une métaheuristique stochastique basée sur la population, inspirée de l'intelligence de l'essaim. Elle imite le comportement social des organismes naturels tels que le vol d'oiseaux et le banc de poissons. En effet, dans ces essaims, des comportements coordonnés utilisant des mouvements locaux émergent sans contrôle central. Originalement, PSO a été conçu avec succès pour des problèmes d'optimisation continue. Sa première application aux problèmes d'optimisation a été proposée par Kennedy et Eberhart **[78]**, par la suite Khouadjia avait utilisé cette méthode dans sa thèse **[20]** pour résoudre le DVRP. Dans le modèle de base, un essaim se compose de *N* particules volant dans un espace de recherche de *D* dimension. Chaque particule *i* est une solution candidate au problème et

qui est représentée par le vecteur x_i dans l'espace de décision. Une particule a sa propre position et sa vitesse, ce qui signifie la direction de vol et l'étape de la particule. L'optimisation profite de la coopération entre les particules. Le succès de certaines particules influencera le comportement de leurs pairs

2.3. Les méthodes parallèles

L'utilisation des méthodes parallèles pour résoudre un problème en temps réel est nécessaire afin de réduire les temps de calcul. Le parallélisme peut être mis en œuvre de manière différente selon l'architecture du matériel utilisé pour le résoudre. Parmi les méthodes parallèles qui existent dans la littérature pour la résolution des problèmes de transport dynamiques, nous pouvons citer la recherche adaptative à voisinage large parallèle, la recherche tabou parallèle, et le multi-essaim parallèle.

a. Les métaheuristiques parallèle à solution unique

1) La recherche adaptative à voisinage large parallèle (Parallel Adaptive Large Neighborhood Search, PALNS)

Pillac et al. proposèrent PALNS dans **[72]** comme une extension de l'algorithme ALNS (Adaptive Large Neighbourhood Search) qui inclut un nouveau schéma de parallélisation qui répartit efficacement l'effort de calcul entre des processeurs indépendants. L'algorithme 2.4 présente le principe de PALNS, qui maintient un ensemble P de N solutions qui sont optimisées dans les K sous-processus (notez que $N \geq K$). Pour chaque itération principale de I^m, un sous-ensemble de K solutions est choisi au hasard (ligne 2) et distribué entre des sous-processus indépendants. Chaque sous-processus effectue I^p ALNS itérations (lignes 3-14) en détruisant et en réparant la solution actuelle X^p comme dans l'algorithme ALNS original. La solution actuelle finale de chaque sous-processus est ajoutée à l'ensemble de solutions (ligne 13), et une procédure de filtrage assure que l'ensemble contient au maximum N solutions, y compris la meilleure solution trouvée à ce moment (ligne 15). L'algorithme s'arrête après les I^m itérations principales, ce qui correspond à $I = I^m \times I^p$ ALNS itérations. Notez que l'implémentation de PALNS assure qu'aucune synchronisation n'est requise entre les sous-processus pour éviter les blocages.

Algorithme 2.4 Parallel Adaptive Large Neighborhood Search (pALNS) algorithm [72]

Input : P solutions initiales, Z fonction d'évaluation, Θ^-/ Θ^+ ensemble d'opérateurs de destruction / réparation, N taille maximale de l'ensemble de solutions, K nombre de sous-processeurs, I^m nombre d'itérations principales, I^p nombre d'itérations effectuées en parallèle.
Output : X^* La meilleure solution trouvée
1: **Pour** I^m itérations **faire**
2: $P' \leftarrow$ selectSubset (P, K); //Choisir un sous-ensemble de k solutions
3: **parallèle pour tous** X **dans** P' **faire**
4: $X^p \leftarrow X$; // La solution courante pour ce sous-processus
5: **Pour** I^p itérations **faire**
6: d $\leftarrow$ select (Θ^-) ; r $\leftarrow$ select (Θ^+) ; // Choisir détruire / réparer
7: $X' \leftarrow r(d(X))$; // Détruire et réparer la solution courante
8: **Si** *accept* (X' ; X^p) **alors**
9: $X^p \leftarrow X'$; // X' est acceptée comme une solution courante
10: **Fin si**
11: updatescore(d,r, X') ; // Mettre à jour les scores d et r
12: **Fin pour**
13: $P \leftarrow P \cup \{ X^p \}$; // Ajouter X^p à l'ensemble P
14: **Fin pour tous**
15: $P \leftarrow$ *retain* (P, N) ; // Retenir au maximum N solutions dans l'ensemble P
16: **Fin pour**
17: retourner $X^* = argmin_{X \in P} \{Z(X)\}$
18: **fin fonction**

PALNS a été utilisé couplé à une politique de seuil modélisant un expert répartiteur pour s'attaquer aux instances de DVRPTW. Les résultats obtenus indiquent qu'il existe un compromis évident entre la minimisation de la distance parcourue et le maintien de la cohérence des itinéraires. De plus, il semble que pour les problèmes avec un faible degré de dynamisme, il peut être utile de sacrifier la rentabilité pour maintenir la cohérence. En revanche, dans les problèmes très dynamiques, la priorité doit être donnée à la minimisation des coûts, car cela ne conduit pas à des incohérences excessives dans le routage.

2) La recherche tabou adaptative parallèle

Gendreau et al. dans les deux articles **[17]** et **[79]**, ont présenté un algorithme de recherche tabou adaptative dont le pseudo code est décrit par l'algorithme 3, afin de résoudre les deux problèmes dynamiques (DPDPTW) et (DVRPTW) respectivement. Leur approche consistait à adapter le cadre introduit par Taillard et al. **[80]** à un contexte dynamique. L'idée générale est de maintenir un ensemble de bons plans de routage - la mémoire adaptative - qui est utilisé pour générer des solutions initiales pour une

recherche tabou parallèle. La parallélisation de la recherche se fait en divisant les routes de la solution actuelle, chaque sous-ensemble étant optimisé par un thread indépendant. Lorsqu'une nouvelle demande de client arrive, elle est vérifiée contre toutes les solutions de la mémoire adaptative avec une heuristique d'insertion rapide pour décider si elle doit être acceptée ou rejetée. Une fois acceptée, la recherche tabou est reprise, y compris la nouvelle demande. Il convient de noter que car les routes actuelles sont sujet à des changements à tout moment, les véhicules ne connaissent pas leur prochaine destination jusqu'à ce qu'ils terminent le service d'une demande.

Cet algorithme a été testé sur une machine multiprocesseurs où chaque processeur lance un algorithme de recherche tabou. En comparant les résultats donnés par cette méthode avec ceux donnés par d'autres méthodes conçues pour la résolution des problèmes de transport dynamiques, les auteurs ont montré que l'utilisation d'une méthode parallèle avec une recherche Tabou permet d'avoir de bien meilleurs résultats. Enfin, nous trouvons aussi dans la littérature des approches de Ichoua et al. **[33]** et **[81]** qui traitent des problèmes VRPs dynamiques mais la dynamique considérée ici est la variation du temps de trajet entre deux clients par rapport à la période de la journée.

Algorithme 2.5 La recherche Tabou adaptative parallèle [79].

Début Tant que (aucun évènement)
Améliorer la solution courante en utilisant la recherche tabou ;
Si (un évènement se produit) **alors**
Arrêter tous les processus de recherche (l'heuristique tabou) et ajouter les meilleures solutions à la mémoire adaptative qui représente un ensemble de bonnes solutions.
Si (l'évènement est l'arrivée d'une nouvelle demande) **alors**
Mettre à jour la mémoire adaptative par l'insertion de la nouvelle demande dans chaque solution ;
Si l'insertion de la nouvelle demande est impossible **alors**
Rejeter la demande ;
FinSi
Sinon (l'évènement est une fin de service d'un client) **alors**
-Identifier la prochaine destination du conducteur, en utilisant la meilleure solution stockée dans la mémoire adaptative ;
-Mettre à jour les autres solutions en conséquence ;
-Relancer les processus de recherche de tabou avec de nouvelles solutions obtenues à partir de la mémoire adaptative.
FinSi.
FinSi.
FinTantQue.
Fin.

b. Les métaheuristiques parallèle à population de solutions

1) Multi-essaim parallèle (Parallel Multi-Swarm Optimization)

Le modèle algorithmique parallèle utilisé par Khouadjia **[20]** pour l'optimisation des essaims de particules c'est le modèle d'îlot. Dans ce modèle, chaque nœud est responsable de l'évolution d'un sous-essaim, dont il exécute toutes les étapes de l'algorithme. Chaque îlot peut utiliser des valeurs différentes de paramètres et des stratégies différentes pour tout composant de recherche tel que la mise à jour de la vitesse, la mise à jour des particules et les encodages. L'objectif du modèle est de retarder la convergence globale et d'encourager la diversité (ce paradigme est illustré à la Figure 2.11).

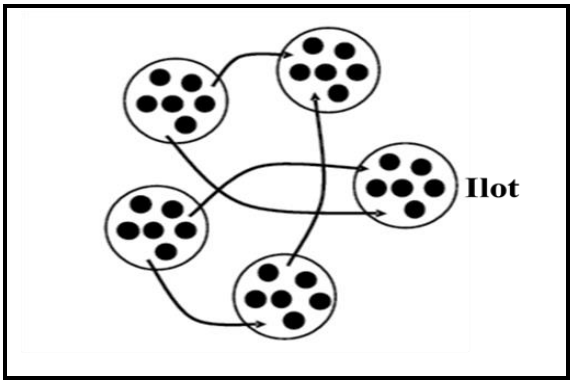

Figure 2.11 Modèle d'îlot parallèle pour multi-essaim [20].

Pour le DVRP, à chaque intervalle du temps, une instance de type VRP qui correspond à l'ensemble des clients arrivés à la fin de ces intervalles est donné à l'algorithme en tant que données d'entrée pour l'intervalle du temps suivant. Une population initiale est construite selon une heuristique de voisinage gourmande. Alors, l'algorithme se déroule en mettant à jour la vitesse des particules ainsi que leurs positions. Les particules se déplacent dans l'espace de recherche en recherchant une meilleure position. L'attracteur global qui correspond à la particule avec la meilleure position (en termes de fitness) est mis à jour. Lorsque le critère de migration est atteint après un certain nombre d'itérations, une sélection est effectuée sur la population et un ensemble de meilleures particules (c'est-à-dire celles ayant les meilleures positions) sont choisies pour la migration vers une population voisine définie par la topologie du multi-essaim. Dans l'autre partie, les particules immigrantes provenant d'une autre population sont reçues et intégrées dans la population locale en remplaçant les mauvaises particules (c'est-à-dire celles ayant les moins bonnes positions en termes de fitness).

Metaheuristiques			Auteur	problème	Principe
Séquentielles	Les métaheuristiques à solution unique	La recherche Tabou (Tabu Search , TS)	Mitrovic-Minic [62]	DPDPTW	TS couplée avec la méthode d'insertion à moindre coût.
			Hanshar et al. [63]	DCVRP	TS de base, deux opérateurs de voisinage : l'inversion et l'échange λ de Osman [64].
		La recherche à voisinage variable (ou Variable Neighborhood Search, VNS)	Armas et Melian-Batista [67]	DVRPTW	-Rejeter les clients qui ne peuvent pas être insérés de manière réalisable en tenant compte des priorités. - Autoriser les fenêtres de temps dans les clients afin de fournir une solution à tous les clients.
			Xu et al. [66]		-Les itinéraires de planification pour la solution initiale sont complétés par la méthode de regroupement. - Les opérateurs hybrides d'insertion et d'échange sont utilisés pour réaliser le processus d'agitation. - Le processus d'optimisation est utilisés pour améliorer l'espace de solution.
		La recherche à voisinage large (ou Large Neighborhood Search, LNS)	Hong [70]	DVRPTW	-Décomposer le problème en une série de problèmes statique de VRPTW. - Utiliser le processus de suppression-réinsertion du LNS successivement.
		La recherche adaptative à voisinage large (The Adaptive Large Neighborhood Search, ALNS)	Pillac et al. [31]	DCVRP	-Ajoute une couche adaptative qui sélectionne de manière aléatoire les opérateurs (suppression-réinsertion du LNS) en fonction de leurs performances passées.
	Les métaheuristiques à population de solutions	Les Algorithmes génétiques (AG)	Benyahia et Potvin [73]	D-PDP	-Chaque véhicule est associé à un vecteur de valeurs d'attribut. - processus de décision d'un distributeur professionnel est construite par programmation génétique.
			Van Hemert et Poutré [76]	DCVRP	- Chaque individu représente un plan pour tous les véhicules de système. - Un seul opérateur de variation : mutation (échange).

			Housroum [77]	DVRPTW	Croisement PMX 3 mutations (Or-Opt, 1-Opt, swap)
		Les colonies de fourmis (Ant Colony Optimization, ACO)	Montemanni et al. [19]	DCVRP	-Stratégie de résolution basée sur la partition de la journée de travail en tranches de temps. - ACO a été utilisé pour résoudre ces problèmes. - Les propriétés de ACO ont été exploitées pour transférer des informations sur les bonnes solutions d'une tranche de temps à la suivante.
		L'essaim de particules (Particle Swarm Optimization, PSO)	Khouadjia[20]		-Un essaim se compose de N particules volant dans un espace de recherche de D dimension. -Chaque particule i est une solution candidate au problème et qui est représentée par le vecteur x_i dans l'espace de décision. - Une particule a sa propre position et sa vitesse, ce qui signifie la direction de vol et l'étape de la particule.
Parallèles	Les métaheuristiques à solution unique	La recherche adaptative à voisinage large parallèle (Parallel Adaptive Large Neighborhood Search, PALNS)	Pillac et al.[72]	DCVRP	-Distribue l'optimisation de solutions prometteuses sur plusieurs processeurs selon le modèle maître-esclave. - La présence d'une mémoire partagée contient l'ensemble de solutions prometteuses avec gestion de la diversité, qui évite les blocages entre les threads d'optimisation.
		La recherche tabou adaptative parallèle	Gendreau et al.[79]	DVRPTW	- La division des routes de la solution, chaque sous-ensemble étant optimisé par un thread indépendant. - Une machine multiprocesseurs où chaque processeur lance un algorithme de recherche tabou.
	Les métaheuristiques à population de solutions	Multi-essaim parallèle (Parallel Multi-Swarm Optimization)	Khouadjia [20]	DCVRP	- Il utilise le modèle algorithmique parallèle d'îlot. - Chaque îlot est responsable de l'évolution d'un sous-essaim.

Table 2.1 Metaheuristiques pour résoudre le DVRP et ses variantes.

2.4. Conclusion

Nous avons vu dans ce chapitre que le problème du VRP dynamique est devenu un domaine recherche important avec de nombreuses applications dans la vie pratique. Par ailleurs, ces travaux ont utilisé plusieurs méthodes de résolution classifiées en deux grandes classes séquentielles et parallèles. Mais malgré sa nature évolutive, nous n'avons pas trouvé dans la littérature, une utilisation des algorithmes de la vie des abeilles pour la résolution de ce type de problèmes.

Dans le chapitre suivant, nous allons présenter notre premier travail de thèse sur l'utilisation de l'algorithme séquentiel de la vie des abeilles (Bees Life Algorithm, BLA) pour la résolution du problème de DCVRP.

Chapitre 3. Approche BLA-DCVRP (notre contribution)

Sommaire

3.1. Introduction

Dans cette thèse, nous traitons le problème de planification de tournées de véhicules dynamique avec une contrainte de capacité (DCVRP : Dynamic Capacitated Vehicle Routing Problem) qui est considérée comme une nouvelle classe de problèmes de VRP. Dans laquelle, une partie ou la totalité de l'entrée est inconnue et révélée au fil du temps, ce qui signifie que la définition d'une instance d'un problème donné est complètement connue à la fin de l'horizon de planification. Donc, le DCVRP nécessite une mise à jour dynamique et en temps réel de sa solution précédente pendant son temps d'exécution à chaque fois qu'une nouvelle demande immédiate soit reçue. Le DCVRP est considéré comme un problème NP-difficile **[18]**, c.-à-d. il ne dispose pas un algorithme général permettant de le résoudre dans un temps polynomial. Par conséquent, la plupart de ces approches dynamiques de résolution dépendent des heuristiques qui calculent rapidement une solution pour l'état actuel du problème.

La méthode d'optimisation par colonie d'abeille est l'une des récentes méthodes d'optimisation. Elle est représentée par un algorithme qui peut être appliqué à de nombreux problèmes d'optimisation dans le management, l'ingénierie, et le contrôle. Elle est basée sur le concept de coopération qui rend les abeilles plus efficaces et ainsi arrivées à leurs buts rapidement. Cette méthode a la capacité, grâce à l'échange d'informations et le processus de recrutement d'intensifier la recherche dans les régions prometteuses de l'espace de solutions.

Dans ce chapitre, nous présentons le principe de notre approche **BLA-DCVRP** inspirée de la vie des abeilles (Bees Life) que nous avons choisie pour résoudre le DCVRP. Nous indiquons les différents composants de notre contribution, en débutant par le gestionnaire d'évènements. Puis, en passant à la méthode de résolution qui est composée de deux phases : la construction des groupes effectuée par *k-means amélioré* (*Enhanced k-means*), et l'optimisation des routes en utilisant les colonies d'abeilles qui fait l'objet de notre étude. Dans ce contexte, nous allons discuter sur les abeilles dans la nature, en donnant la structure d'une colonie d'abeille, ensuite une description de la communication chez les abeilles, et leur comportement. Enfin, nous allons discuter sur le principe de l'algorithme de la vie des abeilles (Bees Life Algorithm, BLA), ainsi que les adaptations que nous avons faites sur cet algorithme séquentiel, pour qu'il puisse traiter le DCVRP.

3.2. Les composants de résolution de DCVRP

Pour résoudre le DCVRP, nous proposons l'architecture suivante, comme il est indiqué dans la Figure 3.1 qui se compose de deux composants principaux. Le premier composant est appelé le gestionnaire d'événements, qui effectue deux tâches principales : gérer les demandes des clients, et créer des instances de CVRP statiques. Le deuxième composant correspond à la méthode de résolution qui résout les occurrences de CVRP successives créées par le gestionnaire d'événements. Dans cette section, nous décrivons ces deux composants en détail.

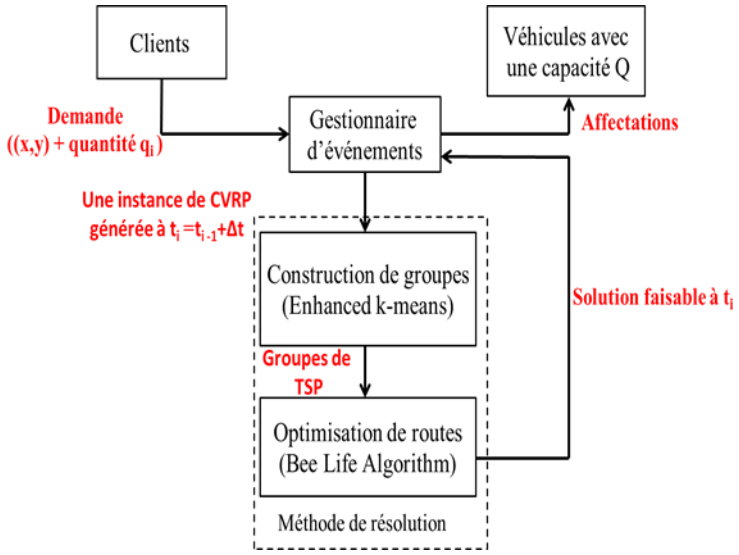

Figure 3.1 Les composants de résolution de DCVRP.

3.2.1. Le gestionnaire d'événement

Le gestionnaire d'événement joue le rôle d'une interface entre l'arrivée de nouvelles demandes et la méthode de résolution. Il reçoit les nouvelles demandes et fait le suivi des demandes déjà servies, concernant la position et la capacité résiduelle de chaque véhicule. Cette information est utilisée pour construire la séquence d'instances de CVRP statiques. La journée de travail est divisée en *nts* tranches de temps, chacune avec une même longueur $\Delta t = T / nts$, où T est la durée de la journée de travail. Cette idée a été d'abord proposée par Kilby et al. **[27]**, et adopté dans l'approche de Montemanni et al. **[19]** qui est basée sur les colonies de fourmis. Chacune de ces tranches de temps représente un CVRP statique partiel, qui considère toutes les demandes qui ont déjà été reçu mais pas encore exécuté. Les nouvelles demandes reçues pendant une tranche de temps sont reportées jusqu'à sa fin. Ce concept d'une tranche de temps discrète limite le temps associé à chaque problème statique, et il fournit un moyen ordonné de répondre aux nouvelles demandes.

Un temps de coupure T_{co} (cut-off time, en anglais) est également considéré comme il est présenté par Montemanni et al. **[19]**. T_{co} est défini par l'utilisateur, et il signifie que les demandes sont autorisées jusqu'à un certain temps T_{co}, et toutes les demandes reçues après ce temps sont reportées au jour suivant.

Un temps d'engagement avancé (advanced commitment time, T_{ac}) est également utilisé. T_{ac} permet à un conducteur de réagir aux nouvelles demandes avant leur traitement.

Le premier problème statique créé pour la première tranche de temps consiste en toutes les demandes restantes de la journée de travail précédente, c.-à-d. les demandes reçues après le T_{co} (les clients non servis). Cela signifie que l'optimisation commence par les clients qui ont été requises hier et n'ont pas servi le même jour en raison du décalage horaire. Le prochain problème statique consiste en des demandes reçues pendant la tranche de temps précédente. Dans ce problème, le véhicule démarre à partir de tout client qu'il a été déjà visité, avec des changements de capacité. Après chaque tranche de temps, la meilleure solution est choisie et les demandes avec un temps de traitement commençant dans les (T / *nts* + T_{ac}) secondes suivantes doivent être affectées à leurs véhicules appropriés. Lorsque tout véhicule a utilisé toute sa capacité, il est renvoyé au dépôt.

3.2.2. La méthode de résolution

En raison de la nature du CVRP (problème NP-difficile), la méthode de résolution a été divisée en deux phases distinctes dans chaque tranche de temps : la construction de groupes où les clients sont affectés aux routes des véhicules, en utilisant la méthode k-means améliorée (*enhanced k-means*) et l'optimisation des routes effectuée par l'algorithme de la vie des abeilles (*Bees Life Algoriyhm, BLA*).

Algorithme 3.1 La méthode de résolution pour une instance de CVRP

Entrée : Une instance de CVRP connue dans une tranche de temps (time slice, T_s), qui correspond à un ensemble de n clients $c_i(x_i, y_i)$ avec une quantité q_i de demande, et un nombre maximum m_{max} de véhicules stationnés dans un dépôt D.

Sortie : Un ensemble de routes, qui minimise la distance totale parcourue en respectant la contrainte de capacité des véhicules

1 : Appeler algorithme 3.2 (*enhanced k-means)* pour la construction de m groupes $\{cl_1, cl_2, ..., cl_m\}$ en respectant la contrainte de capacité des véhicules ; // $m \leq m_{max}$

2: **Pour** chaque groupe cl_j , ($1 \leq j \leq m$) **faire**

3: Appeler Bee Life Algorithm (cl_j) pour l'optimisation des routes;

7: **Fin pour**.

La méthode de résolution est donnée dans l'algorithme 3.1. Cette méthode reçoit comme paramètres d'entrée l'ensemble de *n* clients $c_i(x_i, y_i)$ et un nombre maximal de véhicules m_{max} stationné au dépôt *D*. La sortie de l'algorithme correspond à un ensemble contenant *m* routes réalisables ($m \leq m_{max}$) avec une distance minimale (respectant la contrainte de capacité des véhicules), chacun commence et se termine au dépôt *D*. La phase de construction des groupes (Algorithme 3.2) est appelée pour créer *m* groupes distincts (ligne 1). Ensuite, *BLA* est exécuté (ligne 3) séparément pour optimiser chaque groupe (route de véhicule).

a. La construction des groupes

La construction des groupes représente la première phase de notre méthode de résolution pour les instances de CVRP créées par le gestionnaire d'événements, afin de concevoir un outil efficace pour traiter des données d'optimisation complexes. Cette phase est proposée pour diviser les *n* clients dans des ensembles disjoints ou des groupes selon *m* véhicules. Ensuite, chaque véhicule visitera tous les clients de groupe qui lui sont affectés. La Figure 3.2 montre un exemple composé de 3 véhicules homogènes (avec la même capacité), stationnés dans un dépôt, et un ensemble contenant 17 points (c'est-à-dire les clients) divisé en trois groupes disjoints C1; C2; C3. Chaque groupe doit vérifier la contrainte de capacité des véhicules, dans laquelle les demandes totales des clients dans chaque groupe ne doivent pas dépasser la capacité C_{max} de véhicule.

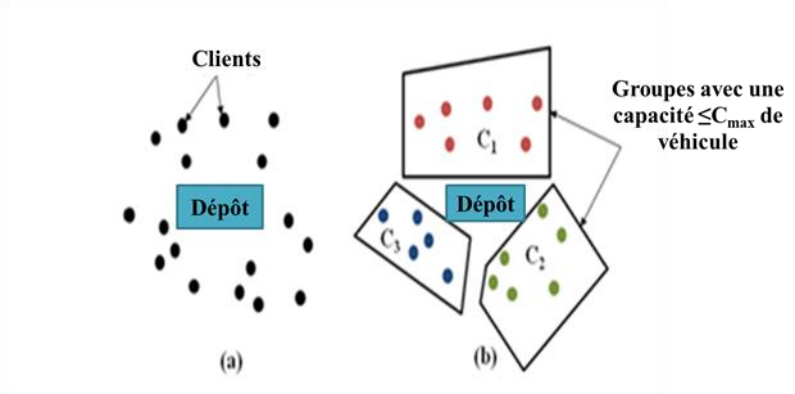

Figure 3.2 Décomposition d'une instance de CVRP (a) en 3 groupes formant des instances de TSP (b) par l'algorithme enhanced k-means.

Il existe plusieurs méthodes de regroupement, qui sont basées sur la dispersion (distribution) des points (*n* clients) autour d'un point central appelé centre. Dans notre cas, nous choisissons la méthode *k-means* **[82]** bien connue pour créer des groupes (ensemble de routes initiales). Le *k-means* est amélioré pour être *enhanced k-means*

[83], visant à sélectionner *m* groupes, chaque groupe a une capacité totale inférieure ou égale à la capacité C_{max} du véhicule. Algorithme 3.2 explique comment fonctionne *enhanced k-means*. Il reçoit l'ensemble des *n* clients comme paramètre d'entrée. En tant que sortie, il crée des groupes disjoints respectant la contrainte de capacité du véhicule.

Algorithm 3.2 *k-means amélioré* (*Enhanced k-means*).

Entrée : Une instance de CVRP, correspond à un ensemble de *n* clients $c_i(x_i,y_i)$ avec des demandes q_i, définie dans une tranche de temps T_s.
Un ensemble de véhicules V_k pour $k = 1,..., m_{max}$, avec une capacité *Cmax*

Sortie : Un ensembles de *m* groupes disjoints ($m \leq m_{max}$) chacun de la capacité totale ≤ *Cmax* .

1: Déplacer ← vraie;
2: Choisir au hasard le 1er client c_i de N. Laissez cette point être les centre de 1ier groupe avec une capacité $C \leftarrow Cmax$, $m \leftarrow 1$;
3: **Tantque** (déplacer) **faire**
4 : Calculer la matrice des distances entre les clients est les centres des groupes ;
5: **Pour** (chaque client $c_i \in$ N, $1 \leq i \leq n$) **faire**
6: **Si** (il existe un centre de groupe cl_j plus proche avec une capacité $C - q_i$ demande≥ 0, $1 \leq j \leq m$) **then**
7 : Affecter (client c_i au centre cl_j);
8 : **Sinon**
9: Ajouter un nouveau centre d'un nouveau groupe avec les cordonnées (x_{cli}, y_{cli}) de client c_i et avec une capacité $C \leftarrow Cmax$, m←m+1;
10 : Affecter (client c_i au centre cl_j);
11: Diminuer la capacité de groupe $C \leftarrow C - q_i$;
12: **Fin si**
13: Recalculer les cordonnées (x_{clj}, y_{clj}) de chaque centre;
14 : **Fin Pour**
15: **Si** aucun point (client) n'a déplacé d'un groupe vers un autre **alors**
16: Déplacer ← faux;
17: **Fin si**
18: **Fin tantque**

Le *k-means amélioré* (*Enhanced k-means*) commence en choisissant le 1er client c_i aléatoirement de N clients (ligne 2), ce qui correspond à un centre (appartenant au groupe Cl_i) avec une capacité maximale de groupe C_{max}. Ensuite (dans la ligne 4), la matrice de distance euclidienne entre chaque client c_i et les centres des groupes Cl_j est calculée selon la formule suivante :

$$d_{c_i cl_j} = \sqrt{(x_{c_i} - x_{cl_j})^2 + (y_{c_i} - y_{cl_j})^2}\,, \quad 1 \leq i \leq n, 1 \leq j \leq m. \qquad (3.1)$$

Après cela (dans la ligne 5), chaque client $c_i \in N$ non affecté à aucun groupe il sera affecté au centre de groupe Cl_j le plus proche au sens de la distance euclidienne minimale, et ce centre de groupe doit avoir une capacité suffisante concernant la demande q_i du client c_i (capacité C - $q_i \geq 0$). S'il n'y a pas de groupe respectant ces conditions (ligne 9), un nouveau centre de groupe Cl_j est ajouté avec les coordonnées (x_{cli}, y_{cli}) du client c_i et avec une capacité maximale. Dans la ligne 7, les coordonnées des m centres sont recalculées. Cela se fait en assignant à chaque groupe Cl_j, ($1 \leq j \leq m$) le centre des points appartenant à ce groupe. Plus précisément, $x_{cl_j} = \frac{1}{|cl_j|} \sum x_{c_k}$ et $y_{cl_j} = \frac{1}{|cl_j|} \sum y_{c_k}$ pour tous les points $c_k \in Cl_j$. Dans la ligne 15, s'il n'y a aucun point déplace d'un groupe à un autre, un regroupement stable est obtenu et la variable *déplacer* prend la valeur false (ligne 16) afin d'arrêter le recalcule des centres des groupes. Sinon, (le cas d'au moins un point a été déplacé), les instructions de la ligne 4 à la ligne 16 sont répétées jusqu'à l'obtention d'une configuration stable, en renvoyant les m groupes en fonction de la contrainte de capacité. Le schéma suivant illustre le principe d'exécution de *k-means amélioré* (*Enhanced k-means*).

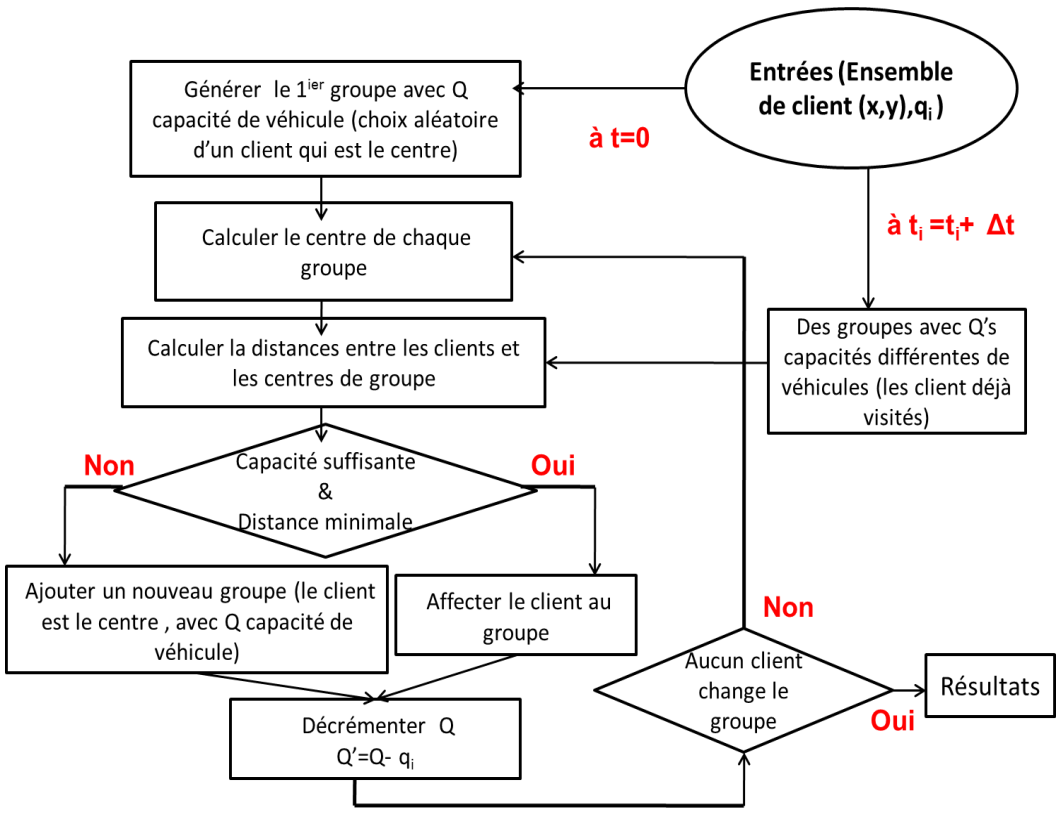

Figure 3.3 Organigramme d'exécution de *k-means amélioré* (*Enhanced k-means*).

b. L'optimisation des routes

L'optimisation des routes représente la deuxième phase de notre approche, qui est exécutée après la construction de *m* groupes en utilisant le k-means amélioré (algorithme 3.2). L'objectif de cette phase est de trouver le chemin le plus court en termes de distance euclidienne minimale dans chaque groupe en utilisant l'algorithme de la vie des abeilles (Bees Life Algoriyhm, BLA). Nous rappelons qu'un chemin commence au dépôt D, en visitant chaque point (client) exactement une fois, et en revenant ensuite au même dépôt D. En outre, un véhicule est associé à chaque groupe afin de servir l'ensemble de clients appartenant à ce dernier.

Dans cette sous-section, une brève définition de BLA est introduite. Ensuite, dans les sections suivantes en va présenter une description détaillée des deux versions de BLA (séquentielle et parallèle).

3.3. Optimisation par colonie d'abeilles (Etat de l'art)

L'optimisation par colonie d'abeilles est une technique stochastique de recherche aléatoire à base de population, qui a été motivé par l'analogie trouvée entre le comportement naturel des abeilles (lors de la reproduction et lors de la recherche de nourriture) et le comportement des algorithmes d'optimisation recherchant un optimum dans les problèmes d'optimisation combinatoire. Les abeilles artificielles explorent l'espace de recherche pour trouver les solutions réalisables. La méta-heuristique de colonie d'abeilles a été récemment utilisée comme un outil pour traiter des problèmes réels et complexes. Il a été démontré que cette méta-heuristique possède une capacité à trouver des solutions de haute qualité à des problèmes combinatoires difficiles dans un délai raisonnable **[84]**.

Cette approche de résolution fait l'objet de notre étude. Dans ce contexte, nous allons discuter sur les abeilles dans la nature, en donnant la structure d'une colonie d'abeille, ensuite une description de la communication chez les abeilles, et leur comportement.

3.3.1. Les abeilles dans la nature

L'abeille (Apis mellifera, qui signifie l'abeille à miel) est un insecte social et domestique originaire d'Europe et d'Afrique. Il y a entre 60 000 et 80 000 éléments vivants dans la ruche. Les abeilles se nourrissent de nectar en tant que source d'énergie dans leur vie et elle utilise le pollen comme une source de protéines dans l'élevage de larves. La colonie d'abeilles contient en général, une seule femelle reproductrice appelée Reine, quelques milliers de mâles connus sous le nom de Faux-bourdons, plusieurs milliers de femelles stériles appelées Ouvrières, et de nombreuses larves de jeunes abeilles s'appellent couvées **[85]**.

a. Communication entre les abeilles

Karl von Frisch a fait une découverte révolutionnaire sur le paradigme de la communication qui permet de partager des informations sur les régions de fleurs autour de la ruche. Il a vérifié expérimentalement que les abeilles peuvent informer les partenaires de la ruche de la direction et de la distance d'une source de nourriture par une suite de danse. La communication des abeilles est partagée quand ils cherchent des sources de nourriture, en commençant par des abeilles appelées « éclaireuses ou scoutes » qui naviguent et explorent la région dans le but de trouver une source de nourriture **[86]**.

L'abeille éclaireuse qui est une abeille ouvrière, lors qu'elle a trouvé une source de nourriture, elle régurgite une partie de sa récolte de nectar. Puis, aussitôt, elle exécute une série de mouvements très stéréotypés, pour informer les autres abeilles concernant la distance, la direction, la quantité et la qualité de nourriture fondée. Avec leur perception visuelle, tactile et olfactive, les autres abeilles perçoivent les informations transmises. Il existe deux types de danses ; la danse en rond quand la nourriture est très proche (source de nectar inférieur à 25 mètres). Cette danse indique seulement la direction.

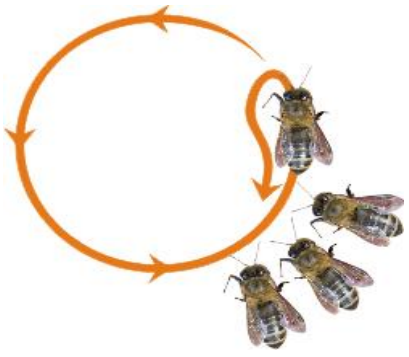

Figure 3.4 La danse en rond.

Le deuxième type est la danse frétillante. C'est une danse qui forme un schéma de huit. Elle indique la distance (source de nectar supérieur à 100 mètres) et la direction de la source de nourriture. La distance entre la source de nourriture et la ruche est transmise en fonction de la vitesse de la danse. Si la danse est plus rapide, la distance est plus petite. Le segment de ligne droite dans la Figure 3.5 indique la direction de la source de nourriture par rapport au soleil : dans l'obscurité de la ruche, l'angle de ce segment avec la verticale (noté α) correspond à celui de l'axe ruche-soleil avec la direction à prendre avec une précision de ± 3 °.

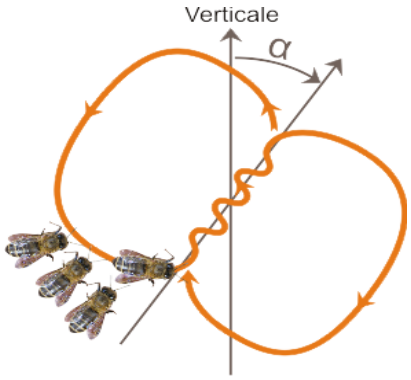

Figure 3.5 La danse frétillante.

La nature de nourriture est indiquée par l'odeur (appelée phéromone) émise par l'abeille éclaireuse quand elle s'est frottée. Cette odeur s'agit de substance messagère qui circule d'une abeille à une autre par la bouche et les antennes, dans lesquelles elles puisent toutes les informations. D'autre coté la quantité de nourriture trouvée dépend du frétillement de l'abeille. Plusieurs frétillements indiquent l'existence d'une grande quantité de nourriture

b. Comportement des abeilles

- **Le comportement de butinage**

➢ **Recherche de de l'emplacement de la ruche :** les colonies les plus prospères se reproduisent par essaimage. Au début du printemps, certaines cellules de la reine sont produites pour générer une nouvelle reine. Avant sa naissance, La reine ancienne quitte la colonie avec la moitié des composants de la colonie pour former une nouvelle colonie. Elles recherchent un nouvel emplacement de la ruche. Les abeilles scoutes recherchent environ douze emplacements pour la ruche. Elles indiquent ces

divers emplacements par des danses frétillantes. La qualité de la danse est liée à la qualité de lieu trouvé. Ainsi, et au fil du temps, les emplacements sélectionnés diminuent jusqu'à ce qu'un seul emplacement soit trouvé.

➢ **Recherche de sources de nourriture :** d'abord, quelques abeilles "éclaireuses" naviguent et explorent la région dans le but de trouver une source de nourriture. Dans le cas positif, elles viennent à la ruche dans un lieu appelé "le piste de danse" pour transmettre et partager cette découverte avec les autres abeilles par le langage de danse (danse en rond ou danse frétillante relatif à la distance de la source découverte). Certaines abeilles sont recrutées et ensuite, deviennent des butineuses. Leur nombre est proportionnel par rapport aux informations données par les scoutes sur la quantité de nourriture. Nous appelons cette étape une phase d'exploration qui est suivie par l'étape d'exploitation. L'abeille récolte la nourriture et calcule sa quantité pour prendre une nouvelle décision. Soit elle continue cette récolte par la mémorisation de ce meilleur emplacement, soit il quitte la source et retourne à la ruche comme une abeille simple.

- **Le comportement du mariage**

Le phénomène de reproduction dans la colonie d'abeilles est garanti par la reine. Après sa naissance, la jeune reine participera à des vols nuptiaux. Elle rejoindra un point de rassemblement avec certains drones. La reine s'approchera avec plusieurs males en plein vol, jusqu'à ce que sa spermathèque soit pleine. Après trois jours, il pond les œufs. L'œuf non fertilisé donnera naissance à un faux-bourdon, tandis que l'œuf fertilisé donnera naissance à une ouvrière ou à une reine selon la qualité de la nourriture donnée aux larves. **[85]**

3.3.2. Algorithme de la vie des abeilles (Bees Life Algorithm, BLA)

L'algorithme de la vie des abeilles (Bees Life Algorithm, BLA) est une méthode bio-inspirée, qui est proposée pour trouver une solution quasi optimale pour les problèmes de recherche. Dans cet algorithme, l'emplacement de la source de nourriture représente la solution possible au problème, et la quantité du nectar de cette source correspond à une valeur objective dite fitness. Les abeilles sont attribuées aux différentes sources de nourriture de façon à maximiser l'apport total de nectar. La

colonie doit optimiser l'efficacité globale de la collecte. La répartition des abeilles est donc en fonction de nombreux facteurs tels que la quantité du nectar et la distance entre la source de nourriture et la ruche. Le nombre des abeilles dans une colonie (population) représente le nombre de solution dans cette population.

Cette méthode s'inspire du comportement des abeilles dans la nature, et elle nécessite plusieurs paramètres définis par l'utilisateur comme suit **[87]** :

- **La taille de population :** Elle représente le nombre N d'abeilles (individus) dans la colonie qui comprend 1 reine, D faux –bourdons et W ouvrières.
- **Le critère d'arrêt** des cycles d'évolution de BLA.

L'algorithme commence par une initialisation de la population dans laquelle N abeilles sont placées au hasard dans l'espace de recherche. Ensuite, la fitness de chaque individu dans la population est évaluée pour qu'elle soit Rangée dans un ordre décroissant selon cette évaluation. En conséquence, une population d'abeilles (contient 1 reine, D faux-bourdons et W ouvrières) est générée, dans laquelle l'abeille qui a la meilleure fitness représente la reine, les D meilleures abeilles suivantes forment les faux-bourdons, et les dernières W abeilles forment les ouvrières. Chaque cycle de la vie d'une population d'abeilles comprend deux comportements d'abeilles : la reproduction et la recherche de nourriture, respectivement. Comme il est illustré dans la Figure 3.6.

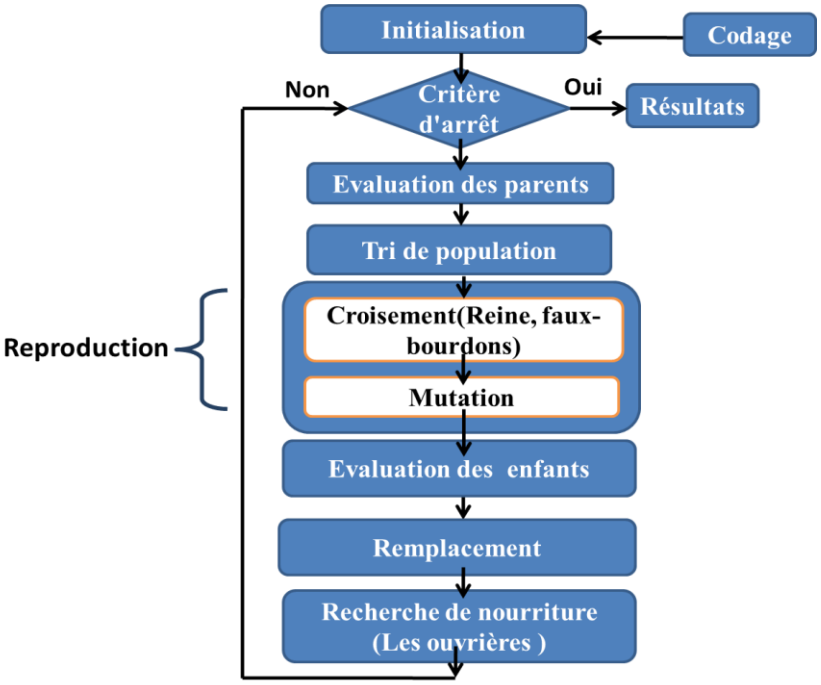

Figure 3.6 Organigramme de l'algorithme de vie des abeilles (BLA).

La reproduction commence dans l'espace par le vol d'accouplement entre la reine et les faux-bourdons, qui représentent les parents à l'aide des opérateurs de croisement et de mutation, afin d'élever N couvées (enfants). Par conséquent, la fitness de chaque couvée sera évaluée pour qu'elle soit rangée dans un ordre décroissant selon cette évaluation, comme il est indiqué dans la section précédente concernant les parents. Si la couvée possède une fitness plus grande que la fitness de la reine, elle sera considérée comme une nouvelle reine pour la population prochaine. En outre, les D faux-bourdons de la prochaine population sont formés par une sélection parmi les meilleures couvées suivantes et les faux-bourdons de la population actuelle. Ensuite, les W ouvrières de la prochaine population sont choisis parmi les derniers couvées et les ouvrières de la population actuelle.

Dans la recherche de nourriture, les W ouvrières cherche une source de nourriture dans les W régions de fleurs. Lorsque chaque ouvrière représente une région et qu'il existe d'autres abeilles pour chaque région, qui sont recrutés et employés pour rechercher la meilleure source de nourriture parmi les différentes sources de nourriture de la région. Les abeilles recrutées représentent des solutions voisines dans l'espace de recherche, et qui sont utilisées pour assurer la recherche de voisinage. BLA utilise plus d'abeilles recrutées pour les meilleures B régions parmi les W régions. B est un paramètre défini par l'utilisateur. Pour chaque région, une seule abeille avec la plus grande fitness sera choisie pour former la prochaine population d'abeilles. L'évaluation de fitness des abeilles de la nouvelle population est exécutée. Si le critère d'arrêt n'est pas satisfait, un nouveau cycle de vie de l'abeille est effectué, puis les deux étapes précédentes (reproduction, recherche de nourriture) sont exécutées.

3.4. BLA séquentielle pour le DCVRP

Après la phase de construction de groupes, on obtient *m* groupes disjoints concernant les *m* véhicules. Dans la deuxième phase d'optimisation des routes, le BLA est utilisé pour optimiser cet ensemble de routes afin de découvrir le chemin le plus court (en termes de distance euclidienne minimale) pour chaque groupe. Dans cette section, nous détaillons les étapes de notre BLA séquentielle pour résoudre le DCVRP. D'abord, on donne une représentation des individus (abeilles) de la population. Puis, on explique comment on va générer la population initiale de chaque groupe, qui contient l'ensemble initial de routes en tant que sous-solution du problème globale (DCVRP). Ensuite, on définit la fonction de fitness utilisée pour évaluer les individus d'une

population. Après, on précise la méthode utilisée pour trier les individus d'une population. En outre, on présente les trois étapes : la reproduction, le remplacement, et la recherche de nourriture, en expliquant les différents opérateurs de BLA (le croisement, et la mutation) utilisés dans ces dernières. Enfin, on montre le critère d'arrêt de notre BLA.

3.4.1. Codage de la solution (représentation des individus)

Dans un algorithme évolutionnaire, les solutions sont représentées par leur génotype (encore appelé chromosome ou individu), qui s'exprime sous la forme d'un phénotype. Le génotype se compose de plusieurs gènes, où chaque gène a une valeur possible (allèle) et une position spécifique dans son chromosome (locus). Le génotype représente le codage tandis que le phénotype représente la solution elle-même. Ainsi, dans le cas d'un codage direct, le génotype et le phénotype sont similaires. Le génotype contient toutes les informations nécessaires à la définition complète du phénotype. Par contre, ils présentent des différences plus importantes dans le cas d'un codage indirect.

Pour résoudre le DCVRP, nous proposons le codage de permutation qui représente dans notre cas un codage directe et discrète de la solution, où on n'a pas besoin d'une phase de décodage pour extraire la solution à partir des individus. La solution d'un DCVRP est représentée par un ensemble de m routes pour les m véhicules afin de servir les n clients S = $\{R_1, R_2, ..., R_m\}$. Où R_k est une séquence des nœuds clients servis par le véhicule V_k. Chaque client doit être visité une seule fois par un seul véhicule, et chaque route doit démarrer et terminer par le nœud 0 qui désigne le dépôt (par exemple R_k = (0,4,9,12,20,6,0)). Un individu représente une route de véhicule, qui est codée directement en utilisant un vecteur d'entiers. Dans la Figure 3.7, le nœud 0 représente le dépôt, et les 5 nœuds sont des clients assignés à un véhicule pour les servir. Ces nœuds sont liés par des arcs orientés, et chaque arc est pondéré par la distance euclidienne $d_{i,\, i+1}$ entre deux nœuds adjacents. Nous exprimons la dimension d'un individu en tant que le nombre de clients dans une route.

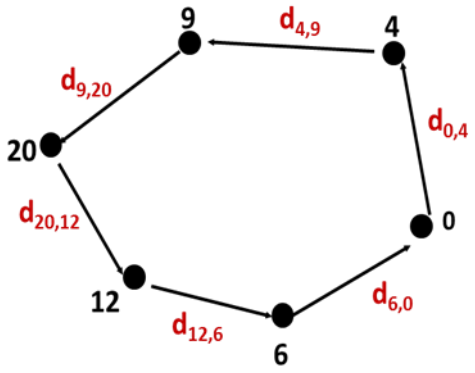

Figure 3.7 Le codage d'un individu (la route R_k de véhicule V_k).

En mentionnant les informations suivantes :

• dem_i : La demande de client c_i (la quantité de marchandises demandées par le client c_i).

• s_i: Une variable booléenne indique si le client c_i est déjà servi ou non.

• ST_I : Le temps de service de client c_i (le temps dans lequel la visite de client c_i devient disponible).

• ca_k : La capacité restante du véhicule v_k.

3.4.2. Initialisation

Dans la première étape, BLA génère une population initiale distribuée de N solutions (abeilles), en appliquant N combinaisons aléatoires sur chaque groupe de clients, où N désigne la taille de la population, et qui est égale au nombre de routes de véhicule dans la population. $N = Q + W + D$, où $Q = 1$ représente la reine, D est le nombre de faux-bourdons, et W est le nombre d'ouvrières (W et D sont des paramètres définis par l'utilisateur). Ce partitionnement de la population est expliqué dans la section 2.4.4. Dans notre proposition, le DCVRP est initialisé par m populations selon les m groupes obtenus par l'algorithme 2 (*Enhanced k-means*).

3.4.3. Evaluation des individus

Chaque fois qu'un nouvel individu a été créé, il faut lui associer une valeur (appelée fitness, force, adaptation ou encore fonction d'évaluation) qui mesure la qualité de la solution. Cette valeur sera utilisée par les processus de tri, remplacement, et recherche de nourriture, pour favoriser les individus les mieux adaptés, autrement dit les meilleures solutions au problème. L'évaluation représente donc la performance de l'individu vis-à-vis du problème à résoudre. De ce fait, cette fonction est en rapport avec

la fonction objectif de ce dernier. Dans le cas mono-objectif, la fonction d'évaluation est généralement une fonction de cet objectif.

Dans notre phase d'évaluation et parmi les différents individus $R_j \in [1, k]$ de chaque groupe (cl_i), nous commençons à chercher le meilleur individu en appliquant l'équation (2.1).

$$\begin{cases} F_{cl_i} = \min\limits_{j=1..k} Cost(R_j) & (2.1) \\ Cost(R_j) = \sum\limits_{i=0}^{n} d_{c_i,c_{i+1}} & (2.2) \\ F_{DC\,VRP} = \sum F_{cl_i} & (2.3) \end{cases}$$

La valeur de fitness de chaque individu dans le groupe est calculée en utilisant l'équation (2.2). En particulier, nous calculons le coût de chaque route R_j, en faisant la somme des distances euclidiennes ($d_{i,\,i+1}$) entre un client visité (c_i) et son voisin (c_{i+1}) qui est le prochain client visité sur cette route. Notez qu'un individu R_j est définit par $R_j = (c_0, c_1, ..., c_n, c_0)$, où c_0 représente le dépôt, n est le nombre de clients visités dans la route R_j.

Enfin, le total de toutes les meilleures valeurs de fitness trouvées dans les différents groupes constitue la solution optimale trouvée pour le DCVRP comme donnée par l'équation (2.3).

3.4.4. Tri de population

Après la phase d'initialisation et après avoir terminé les phases d'évolution (une itération) de BLA, la nouvelle population créée doit être triée pour se préparer à la prochaine itération. Dans cette étape, la population de taille $N = Q + D + W$ $(Q = 1)$ est triée par ordre décroissant de la séquence de fitness des individus. L'individu Q du haut représente la reine qui a la plus petite valeur de fitness correspondant au plus petit coût des routes ($F(cl_i)$), les D individus suivants représentent les faux-bourdons, et les W derniers individus sont les ouvrières (voir la Figure 3.8).

Il existe plusieurs méthodes de tri des éléments de vecteur. Dans notre contribution, nous avons choisi la fonction de tri « Sort » incluse dans l'en-tête <algorithm> de la bibliothèque standard de C ++ (Standard Template Library, STL) **[88]**.

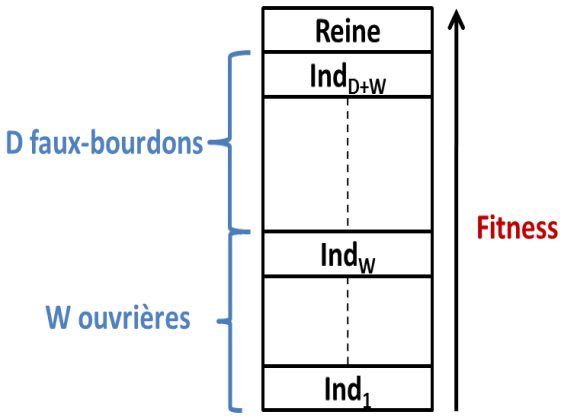

Figure 3.8 Le tri de la population.

La fonction de tri « Sort » porte trois arguments qui sont : le premier individu (nommé *premier*), le dernier (nommé *dernier*), et la fonction de comparaison entre deux individus (abrégé *comp*). Ici, le premier et le dernier sont des itérateurs à accès aléatoire (random access iterators), qui doivent définir une séquence de valeurs, c'est-à-dire que l'individu suivant doit être accessible par une application répétée de l'opérateur d'incrémentation sur le premier individu. Le troisième argument *comp* dénote un prédicat de comparaison. Ce prédicat de comparaison doit définir un ordre strict sur les éléments de la séquence à trier, dans notre proposition nous avons utilisé le prédicat de comparaison entre les fitness des individus. Nous notons que le troisième argument est facultatif ; sinon, l'opérateur "inférieur à, < " est utilisé, C'est ce qu'on appelle la surcharge des opérateurs en C ++ **[88]**. La complexité de la fonction de tri «Sort» est O(N log N) où N est le nombre d'éléments à trier.

3.4.5. Reproduction

Pour que l'algorithme puisse trouver des solutions meilleures que celles représentées dans une population, il est indispensable qu'elles soient transformées par l'application d'opérateurs de variation sur des copies des individus sélectionnés pour en engendrer de nouveaux. En général, on peut trouver deux classes principales :

• Les opérateurs croisement qui produisent un ou plusieurs enfants à partir de deux (ou plusieurs) parents,

• Les opérateurs de mutation qui produisent un nouvel individu à partir d'un seul individu, qui est l'individu lui-même.

Dans la phase de reproduction de BLA, nous avons appliqué ces deux types d'opérateurs cités précédemment, et qui sont expliqués comme suit :

a. Croisement (Reine, faux-bourdons)

L'opérateur de croisement est un opérateur génétique binaire, qui est effectué entre la reine et un faux-bourdon choisi au hasard, et par conséquent, deux nouveaux individus (abeilles) sont générés en remplaçant les mauvais individus en fonction de la valeur de fitness. Ce processus est répété jusqu'à atteindre N individus avec une probabilité de croisement p_c = [0.45, 0.95].

Nous proposons l'opérateur de croisement à deux points (crossover 2X) **[89]** pour notre codage de permutation, dans lequel deux points de coupure sont générés de manière aléatoire, à l'exception du premier et du dernier élément de chaque parent représentant le dépôt. Ces points sélectionnés coupent chaque parent en trois parties. Dans la construction du premier enfant (voir la Figure 3.9), les éléments (0, 2, 5 et 9, 0) des frontières du premier parent (P_1, Reine) sont hérités par l'enfant (E_1), et les éléments de la partie centrale (1, 4, 7, 8, 6, 3) sont réarrangés dans le même ordre que dans l'autre parent (P_2, faux-bourdon). Pour le deuxième enfant (E_2), les rôles des parents sont inversés. A travers cette reproduction les individus enfants héritent des parties de patrimoine génétique de leurs parents.

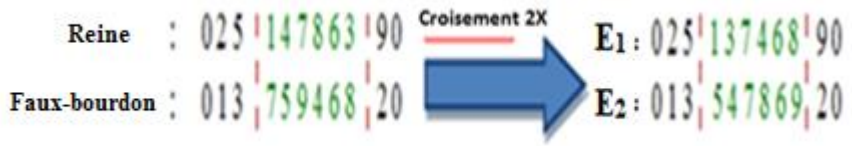

Figure 3.9 Exemple de l'opérateur de croisement 2X.

b. Mutation

Après le croisement, un opérateur de mutation est appliqué au sein de la population des enfants avec une probabilité très faible, nommée taux de mutation ou P_m. Cet opérateur joue le rôle d'un (élément perturbateur). Il introduit du (bruit). Il permet ainsi de maintenir la diversité de la population des enfants et d'explorer l'espace de recherche en évitant à l'algorithme de converger trop rapidement vers un optimum local. Les petits taux de mutation sont recommandés car un grand taux peut occasionner une destruction de l'information utile contenue dans les solutions et être considéré probablement comme une recherche aléatoire

En général, la mutation ne permet pas l'obtention de meilleures solutions, mais elle permet de garder une diversité dans l'évolution des individus et d'éviter les optimums locaux, et se protège contre une perte irrécouvrable dans les caractéristiques des individus. L'opérateur de mutation est appliqué sur chaque enfant de la population avec une probabilité prédéterminée p_m = [0.001, 0.01]. Nous proposons l'opérateur de mutation par insertion **[90]**. Pour ce faire, nous choisissons au hasard un numéro de client dans l'individu, puis nous le retirons de son ancien emplacement, et nous l'insérons dans un nouvel emplacement choisi au hasard dans le même individu, en exceptant l'endroit du dépôt. Par exemple, nous considérons le premier enfant (E_1) dans l'exemple de la Figure 3.10. Et nous supposons que l'opérateur de mutation par insertion sélectionne le client numéro 3, qui sera retiré et inséré au hasard après le client numéro 6. Par conséquent, l'enfant résultant (E_2) est obtenue, comme il est illustré dans la Figure 3.10.

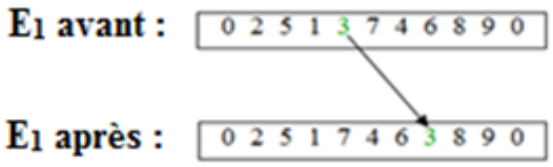

Figure 3.10 Exemple d'opérateur de mutation par insertion.

3.4.6. Remplacement

Après l'application des opérateurs de reproduction, chaque individu dans la population des enfants est évalué. Ensuite, le remplacement détermine quels individus devront disparaitre de la population à chaque génération et quels individus forts survivent dans la population de la prochaine génération.

La stratégie de remplacement définit comment intégrer les descendants (enfants) dans la population actuelle pour former la nouvelle population. On trouve essentiellement deux stratégies de remplacement différentes, y compris le remplacement stationnaire et le remplacement élitiste. Dans l'implémentation de notre BLA, nous avons choisi la stratégie de remplacement élitiste, où on garde au moins l'individu possédant les meilleures performances d'une génération à la suivante. En général, on peut partir du principe qu'un nouvel individu (enfant) prend place au sein de la population que s'il remplit le critère d'être plus performant que le moins performant des individus de la population précédente. Donc les enfants d'une génération ne

remplaceront pas nécessairement leurs parents comme dans le remplacement stationnaire, où les enfants remplacent automatiquement les parents sans tenir compte de leurs performances respectives, et le nombre d'individus de la population ne varie pas tout au long du cycle d'évolution simulé, ce qui implique donc d'initialiser la population initiale avec un nombre suffisant d'individus.

3.4.7. Recherche de nourriture

La recherche de voisinage dans la partie de recherche de nourriture de BLA est utilisée pour atteindre l'individu voisin de l'individu original (abeille ouvrière). Nous proposons un voisinage basé sur l'opérateur d'échange **[91]** qui consiste à permuter l'emplacement de deux éléments choisis aléatoirement (clients) c_i et c_j de la permutation (Figure 3.11). Pour une permutation de taille *n*, la taille de ce voisinage est *n (n-1) /2*.

Les abeilles recrutées représentent des solutions voisines dans l'espace de recherche, et qui sont utilisées pour assurer la recherche de voisinage. BLA utilise plus d'abeilles recrutées pour les meilleures B régions parmi les W régions. B est un paramètre défini par l'utilisateur. Pour chaque région, une seule abeille avec la plus grande fitness sera choisie pour former la prochaine population d'abeilles. L'évaluation de fitness des abeilles de la nouvelle population est exécutée.

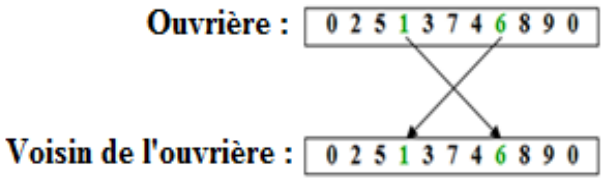

Figure 3.11 Exemple de l'opérateur d'échange.

3.4.8. Critère d'arrêt

Généralement, le cycle d'évolution est répété jusqu'à ce qu'un critère d'arrêt soit satisfait. Les critères d'arrêt varient en fonction du problème à résoudre, qui peut être notamment un nombre fixe d'itérations (générations), un temps maximal de calcul, ou/et lorsqu'une solution satisfaisante soit atteinte. Dans l'étude de notre BLA pour résoudre le DCVRP, nous choisissons d'appliquer un critère d'arrêt statique, dans lequel l'algorithme s'arrête après 1000 itérations. Parce qu'il est destiné à présenter que la

version GPU prend beaucoup moins de temps lorsque les conditions similaires se produisent par rapport à la version CPU.

3.5. Conclusion

Ce chapitre présente une approche d'optimisation bio-inspirée appelé Bees Life Algorithm (BLA) pour résoudre le problème de planification de tournées de véhicules dynamique avec une contrainte de capacité (DCVRP). Notre première proposition séquentielle est appliquée pour résoudre les CVRP statiques résultants de la partition de la journée de travail en tranches de temps. En outre, cette approche est divisée en deux phases distinctes dans chaque tranche de temps : la construction des groupes où les clients sont affectés premièrement aux routes initiales des véhicules (en utilisant la méthode k-means améliorée), et ensuite l'optimisation des routes qui représente la phase centrale dans laquelle les chemins les plus courts dans les groupes formés (routes de véhicules) sont découverts.

Dans le chapitre suivant, nous donnant un aperçu général sur les métaheuristiques parallèles, et sur l’exploitation du potentiel de GPU pour le calcul généraliste (General Purpose GPU, GPGPU), pour implémenter des métaheuristiques parallèles.

Chapitre 4. Métaheuristiques parallèles (Etat de l'art)

Sommaire

4.1. Introduction

Bien que les métaheuristiques fournissent des stratégies assez efficaces pour trouver des solutions approchées aux problèmes d'optimisation combinatoire, le temps de calcul associés à l'exploration de l'espace de solution peuvent être très importantes. Avec la prolifération des ordinateurs parallèles, des stations de travail puissantes et des réseaux de communication rapides, les implémentations parallèles de métaheuristiques apparaissent tout naturellement comme une alternative à l'accélération de la recherche de solutions approximatives, elles permettent également de trouver des solutions améliorées par rapport à leurs homologues séquentielles, en raison de la partition de l'espace de recherche et de plus de possibilités d'intensification et de diversification de recherche. Par conséquent, le parallélisme est un moyen non seulement de réduire le temps de fonctionnement des métaheuristiques, mais aussi d'améliorer leur efficacité et leur robustesse. Ces derniers sont susceptibles d'être la contribution la plus importante du parallélisme aux métaheuristiques.

Les dernières années ont vu l'émergence d'un grand nombre d'architectures informatiques parallèles. Presque tous les ordinateurs de bureau ou portables, et même

les téléphones mobiles, sont équipés de plusieurs cœurs de processeurs intégrés. Les GPU ont également été découverts comme source de puissance de calcul massive sans coût supplémentaire. L'utilisation du GPU dans des applications non-graphiques a eu un intérêt croissant dans le monde de l'optimisation combinatoire depuis les années 2009. Le succès du GPU revient à son coût faible, car c'est une architecture grand public produite en série et facilement accessible. L'enjeu financier rend cette technologie très adéquate à l'implémentation des métaheuristiques parallèles pour la résolution des problèmes combinatoires.

4.2. Les raisons de parallélisation des métaheuristiques

Le calcul parallèle et distribué peut être utilisé dans la conception et l'implémentation des métaheuristiques pour les raisons suivantes **[92]** :

- **Accélérer la recherche :** L'un des objectifs principaux de la parallélisation d'une métaheuristique est de réduire le temps de recherche. Cela permet de concevoir des méthodes d'optimisation interactives et en temps réel. Ceci est un aspect très important pour une classe de problèmes où il y a des exigences strictes sur le temps de recherche, comme dans les problèmes d'optimisation dynamiques et les problèmes de contrôle de temps critique tels que la planification "en temps réel".

- **Améliorer la qualité des solutions obtenues :** Certains modèles parallèles de métaheuristiques permettent d'améliorer la qualité de la solution trouvée. En effet, l'échange d'informations entre les métaheuristiques coopératives va modifier leur comportement en termes de recherche dans l'espace associé au problème. L'objectif principal d'une coopération parallèle entre les métaheuristiques est d'améliorer la qualité des solutions. Une meilleure convergence et un temps de recherche réduit peuvent se produire. Notons qu'un modèle parallèle pour les métaheuristiques peut être plus efficace qu'une métaheuristique séquentielle même sur un seul processeur.

- **Améliorer la robustesse :** Une métaheuristique parallèle peut être plus robuste en termes de résoudre de manière efficace différents problèmes d'optimisation et différentes instances d'un problème donné. La robustesse peut également être mesurée en termes de sensibilité de la métaheuristique à ses paramètres.

- **Résoudre des problèmes à grande échelle :** Les métaheuristiques parallèles permettent de résoudre les instances à grande échelle de problèmes d'optimisation

complexes. Un défi ici est de résoudre très grandes instances qui ne peuvent pas être résolues par une machine séquentielle. Un autre défi similaire consiste à résoudre des modèles mathématiques plus précis associés à différents problèmes d'optimisation. L'amélioration de la précision des modèles mathématiques augmente, en général, la taille des problèmes associés à résoudre. De plus, Certains problèmes d'optimisation nécessitent la manipulation des bases de données énormes telles que des problèmes d'exploration de données (data mining).

4.3. La modélisation parallèle des métaheuristiques

La raison qui pousse les chercheurs à utiliser les modèles parallèles dans les métaheuristiques est l'importante demande en puissance de calcul pour les problèmes à résoudre. En termes de conception de métaheuristiques parallèles, trois grands modèles parallèles sont identifiés **[92]** :

- **Le modèle parallèle au niveau de l'algorithme :** ce modèle parallèle consiste à paralléliser des algorithmes (parallélisation inter-algorithme). Il ne dépend pas du problème traité. Si les algorithmes sont indépendants les uns des autres, leur parallélisation n'apporte qu'une accélération de leur exécution, Il n'y a pas d'amélioration de la qualité des solutions trouvées par rapport au modèle séquentiel. En revanche, si les algorithmes sont coopératifs, leur parallélisation peut non seulement réduire leur temps d'exécution, mais aussi améliorer les solutions trouvées par rapport au modèle séquentiel.

- **Le modèle parallèle au niveau de l'itération :** Dans ce modèle, chaque itération d'une métaheuristique est parallélisée (parallélisation intra-algorithme), indépendamment du problème traité. Le comportement de la métaheuristique n'est pas modifié. L'objectif principal est d'améliorer l'algorithme en réduisant le temps de recherche, et non pas les solutions trouvées par rapport au modèle séquentiel. En effet, le cycle d'itération des métaheuristiques sur les grands voisinages pour les S-métaheuristiques ou les grandes populations pour les P-métaheuristiques nécessite une grande quantité de ressources de calcul, en particulier pour les problèmes du monde réel.

- **Le modèle parallèle au niveau de la solution :** dans ce modèle, le processus de parallélisation gère une seule solution de l'espace de recherche (parallélisation intra-

algorithme) dépendant du problème. En général, l'évaluation de la ou des fonction (s) objective (s) ou des contraintes pour une solution générée est souvent l'opération la plus coûteuse en métaheuristiques. Dans ce modèle, le comportement de la métaheuristique n'est pas modifié. L'objectif est principalement l'accélération de la recherche.

4.4. Les architectures parallèles

Un modèle d'architecture parallèle est une description abstraite d'un ordinateur parallèle dont le but est de représenter les particularités les plus importantes de la machine **[93]**.

Plusieurs classifications des ordinateurs parallèles sont proposées dans la littérature. La classification la plus connue est celle de Flynn **[94]**. Elle est fondée sur l'organisation des flots d'instructions (séquences d'instructions transmises à partir d'une unité de contrôle vers un ou plusieurs processeurs) et des flots de données (séquences de données provenant de la mémoire et se dirigeant vers un processeur ou provenant d'un processeur et se dirigeant vers la mémoire). Selon la classification de Flynn, quatre architectures sont proposées pour gérer les modèles de parallélisation Flynn développe quatre types d'organisation pour une machine de donnée. On distingue alors les architectures suivantes :

- **SISD (Single Instruction Single Data) :** Ce type correspond au mode de fonctionnement des architectures séquentielles conventionnelles dans lesquelles un module du microprocesseur est actif à un instant donné. Il utilise un unique flot d'instructions et opèrent sur un unique flot de données.

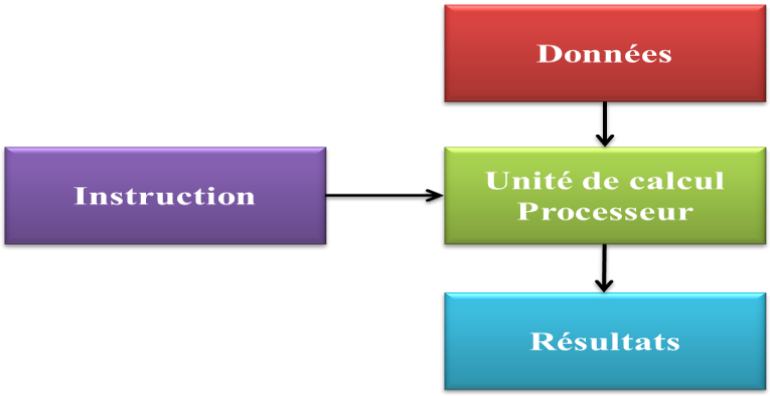

Figure 4.1 Architecture SISD.

- **SIMD (Single Instruction Multiple Data) :** Le module de traitement est dupliqué, et la mémoire de données est partagée en blocs disjoints, chaque bloc étant associé à un module de traitement. Tous les processeurs reçoivent la même instruction,

et qui sont supervisés par la même unité de contrôle, mais ils opèrent sur des données distinctes qui proviennent de différents flots de données. Ils opèrent de façon synchrone, chaque processeur exécutant la même instruction à chaque unité de temps.

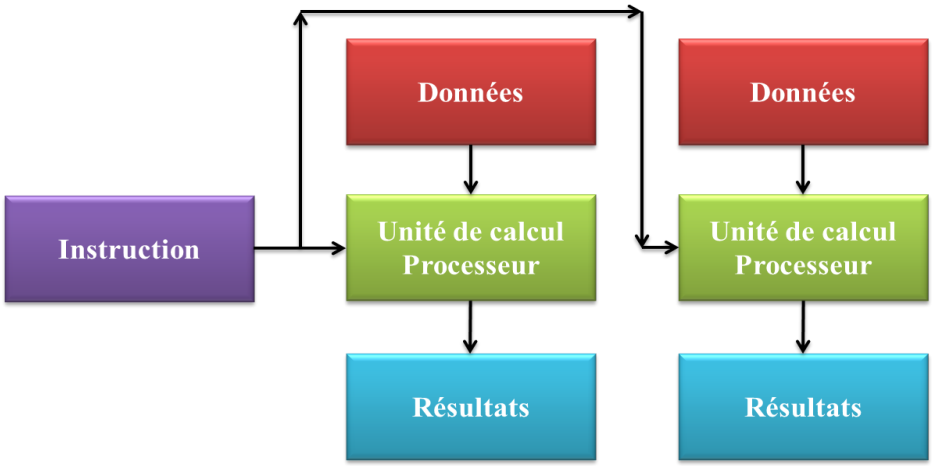

Figure 4.2 Architecture SIMD.

- **MISD (Multiple Instruction Single Data) :** Cette classe fait apparaître une duplication des unités de séquencement et des unités de traitement associées. Celles-ci exécutent simultanément, plusieurs flots d'instructions sur un unique flot de données.

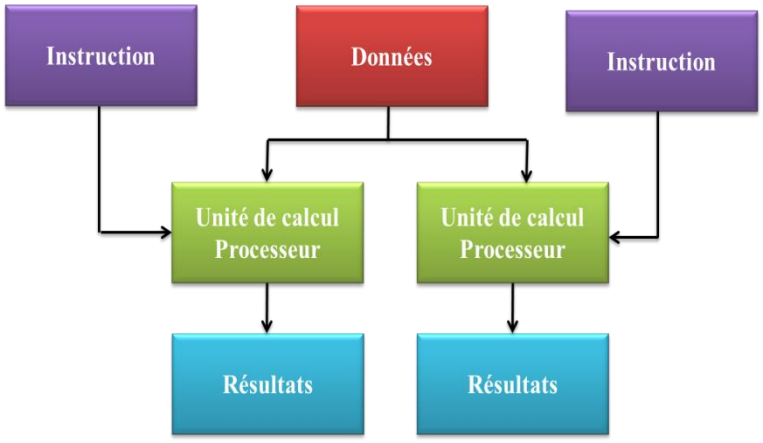

Figure 4.3 Architecture MISD.

- **MIMD (Multiple Instruction Multiple Data) :** La plupart des systèmes récents peuvent entrer dans cette catégorie. On constate que cette architecture ressemble à une réplique d'ordre n de la structure S.I.S.D. Dans une machine fonctionnant en mode MIMD, les processeurs sont indépendants les uns des autres et fonctionnent de façon asynchrone, chacun possédant sa propre unité de contrôle. Un schéma d'architecture MIMD est illustré à la Figure 4.4 Comme en SIMD, chacun utilise ses propres données mais y applique, en plus, ses propres flots d'instructions.

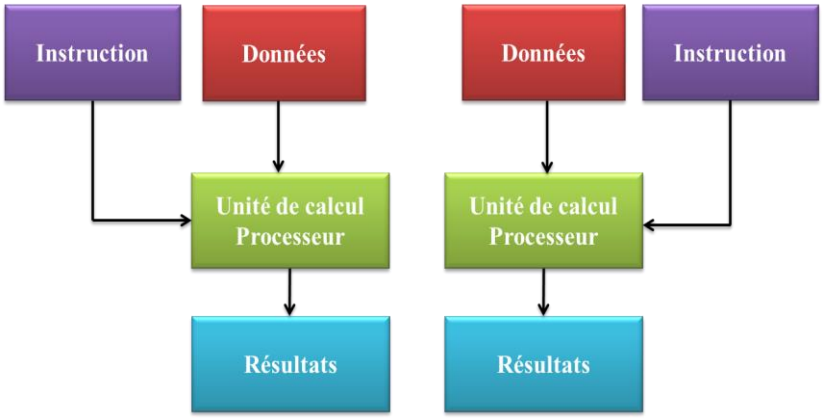

Figure 4.4 Architecture MIMD.

On peut préciser que cette catégorie MIMD peut être découpée en deux sous-catégories :

- **SPMD (Single Program Multiple Data) :** Consiste à exécuter un seul programme sur plusieurs données à la fois.
- **MPMD (Multiple Program Multiple Data) :** Consiste à exécuter des programmes en parallèle sur des données différentes.

4.5. Les schémas d'exécution des algorithmes évolutionnaire

Un des nombreux avantages des algorithmes évolutionnaires est qu'ils sont faciles à paralléliser. Le processus d'évolution artificielle peut être implémenté de différentes manières sur des machines parallèles. Il est possible de paralléliser des opérations spécifiques ou de paralléliser le processus évolutif lui-même. Cette dernière approche a conduit à une variété d'algorithmes de recherche **[95]** :

4.5.1. Modèle maître-esclave (Master-Slave)

Il existe plusieurs façons d'utiliser des machines parallèles. Un moyen simple d'utiliser la parallélisation consiste à exécuter des opérations sur des processeurs séparés. Cela peut concerner les opérateurs de variation comme la mutation et la recombinaison ainsi que les évaluations de fonctions. En fait, cela a plus de sens pour les évaluations de fonctions car ces opérations peuvent être effectuées indépendamment et elles sont souvent parmi les opérations les plus coûteuses. Dans ce modèle, une machine représente le maître et distribue la charge de travail pour l'exécution des opérations sur plusieurs autres machines appelées esclaves. Il est bien adapté à la

création des descendants des populations car il est possible de créer et d'évaluer ces derniers, une fois que les parents appropriés ont été sélectionnés.

Le système est généralement synchronisé, où le maître attend que tous les esclaves aient terminé leurs opérations avant de poursuivre. Cependant, il est possible d'utiliser des systèmes asynchrones où le maître n'attend pas les esclaves trop longs. Le comportement des modèles maître-esclave synchronisés n'est pas différent de leurs homologues séquentiels. L'implémentation est différente, mais l'algorithme est le même.

4.5.2. Modèle d'exécutions indépendantes (Independent Runs)

Des machines parallèles peuvent également être utilisées pour simuler différentes exécutions indépendantes du même algorithme en parallèle. Un tel système est très facile à mettre en place car aucune communication pendant le temps d'exécution n'est nécessaire. Une fois que toutes les analyses ont été arrêtées, les résultats doivent être collectés et la meilleure solution (ou une sélection de différentes solutions de haute qualité) est produite. Alternativement, toutes les machines peuvent communiquer périodiquement leurs meilleures solutions actuelles afin que le système puisse être arrêté dès qu'une solution satisfaisante a été trouvée. En ce qui concerne les modèles maître-esclave, cela nous évite d'attendre que la plus longue exécution se termine. Malgré sa simplicité, les exécutions indépendantes peuvent être assez efficaces. Considérons un paramètre où une seule exécution d'un algorithme a une probabilité de succès particulière, c.-à-d., une probabilité d'obtenir une solution satisfaisante dans un délai donné. Que cette probabilité soit notée p. En utilisant plusieurs exécutions indépendantes, cette probabilité de succès peut être augmentée de façon significative. Cette approche est communément appelée amplification de probabilité.

La probabilité que dans λ exécutions indépendants, aucune exécution est réussie est $(1-p)^{\lambda}$. La probabilité qu'il y ait au moins une exécution réussie parmi ceux-ci est donc $1-(1-p)^{\lambda}$. Nous pouvons voir que pour un petit nombre de processeurs, la probabilité de succès augmente presque linéairement. Si le nombre de processeurs est important, un effet de saturation se produit. Le bénéfice d'utiliser toujours plus de processeurs diminue avec le nombre de processeurs utilisés. Le point où la saturation se produit dépend crucialement de p: pour de plus petites probabilités de succès, la saturation ne se produit qu'avec un nombre assez important de processeurs.

En outre, des exécutions indépendantes peuvent être mises en place avec différentes conditions initiales ou différents paramètres. Ceci est utile pour explorer efficacement l'espace de recherche et pour trouver de bons paramètres dans un temps plus court.

4.5.3. Modèle d'îlot (island model)

Dans les modèles d'îlot, également appelés Algorithmes évolutionnaires distribués (distributed EAs,), modèle à grains grossiers (coarse-grained model) ou modèle multi-dèmes (multi-deme model), la population de chaque exécution est considérée comme un îlot. On parle souvent d'îlots en tant que sous-populations qui forment ensemble la population globale du modèle d'îlot. Les îlots évoluent indépendamment comme dans le modèle des exécutions indépendantes, la plupart du temps. Mais des solutions sont périodiquement échangées entre les îlots dans un processus appelé migration.

L'idée est d'avoir une topologie de migration, un graphe orienté avec des îlots comme nœuds et des arcs orientés reliant deux îlots. À certains moments, des individus sélectionnés de chaque îlot sont envoyés dans les îlots voisines, i. e., les îlots qui peuvent être atteintes par un arc orienté dans la topologie. Ces individus sont appelés migrants et ils sont inclus dans l'îlot cible après un processus de sélection supplémentaire. De cette façon, les îlots peuvent communiquer et se faire concurrence. Un exemple de modèle d'îlot est donné à la Figure 4.5.

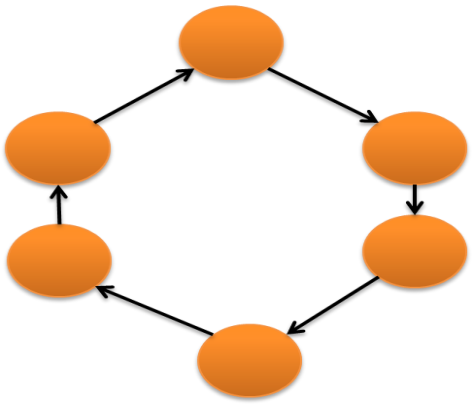

Figure 4.5 Exemple de topologie du modèle d'îlot.

Si tous les îlots utilisent le même algorithme dans des conditions identiques, on parle d'un modèle d'îlot homogène. Tandis que, les modèles d'îlots hétérogènes contiennent des îlots ayant des caractéristiques différentes, des représentations

différentes, des fonctions objectives ou des paramètres différents, et même des algorithmes différents peuvent être utilisés.

4.5.4. Modèle cellulaire des algorithmes évolutionnaires (Cellular EAs)

Le modèle cellulaire représente un cas particulier de modèle d'îlot avec une forme de parallélisation plus fine. Comme dans le modèle d'îlot, nous avons des îlots reliés par une topologie fixe. Les anneaux et les graphiques de tore bidimensionnels sont le choix le plus commun. La caractéristique la plus forte est que chaque îlot ne contient qu'un seul individu. Les îlots sont souvent appelés cellules dans ce contexte, ce qui explique le terme cellulaire. Chaque individu est seulement autorisé à s'accoupler avec ses voisins dans la topologie. Ce type d'interaction se produit à chaque génération. La Figure 4.6 montre un schéma d'un modèle cellulaire sur un graphe de grille 7×7.

Les modèles cellulaires donnent un système beaucoup plus fin; ils ont donc été appelés modèles à grain fin (fine-grained models), modèle de voisinage (neighborhood model) ou modèle de diffusion. La différence par rapport aux modèles d'îlot est qu'aucune évolution n'a lieu sur la cellule elle-même, i. e., il n'y a pas d'évolution intra-îlot. Les améliorations ne peuvent être obtenues que par une interaction entre les cellules. Il est cependant possible qu'un îlot puisse interagir avec elle-même.

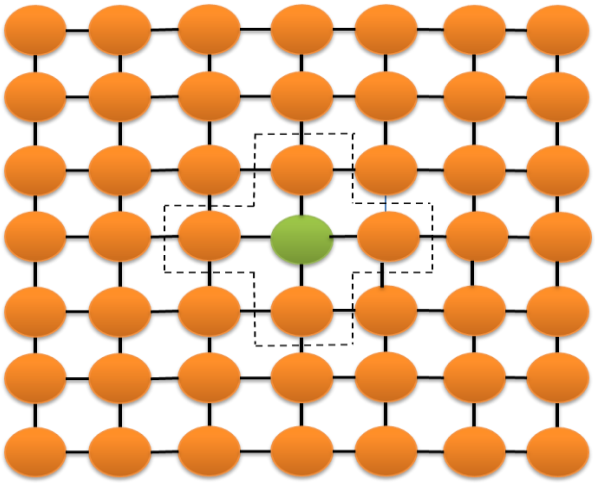

Figure 4.6 Modèle cellulaire sur un graphe de grille 7×7. La ligne pointillée indique le voisinage de la cellule verte.

4.5.5. Modèles hybrides

Il est également possible de combiner plusieurs modèles. Par exemple, on peut imaginer un modèle d'îlot où chaque îlot exécute un algorithme évolutionnaire cellulaire pour promouvoir la diversité. Ou on peut penser à des modèles d'îlot hiérarchiques où

les îlots sont elles-mêmes des modèles d'îlot. Les modèles d'îlots et les modèles cellulaires peuvent également être implémentés en tant que modèles maître-esclave pour obtenir une meilleure accélération.

4.6. Métaheuristiques parallèles sur GPU

La plupart des ordinateurs personnels, qu'ils soient fixes ou portables, comportent un GPU (Graphic Processing Unit). Ce processeur graphique a la particularité d'être un composant hautement parallèle avec une puissance de calcul très élevée. Pendant des années, l'utilisation de processeurs graphiques a été dédiée aux applications graphiques. Motivés par la demande de graphiques 3D haute définition sur les ordinateurs personnels, les GPU ont évolué vers un environnement hautement parallèle, multithread et multicœur. En effet, cette architecture fournit une puissance de calcul énorme et une bande passante mémoire très élevée par rapport aux processeurs traditionnels. Le potentiel énorme de ce composant est exploité dans les applications non graphiques, comme l'implémentation des métaheuristiques parallèles. Dans cette section, on met l'accent sur la description de calcul par les GPUs. Une compréhension claire des caractéristiques GPU est nécessaire pour fournir une implémentation efficace des métaheuristiques parallèles.

4.6.1. Architecture GPU

Le processeur graphique GPU est conçu à l'origine comme un coprocesseur appelé device, il est spécialisé et dédié à accélérer les calculs intervenant dans le rendu interactif d'images de synthèse. Il prend place au sein d'un système hôte qu'il est appelé host, constitué d'un ou plusieurs CPU, d'un espace mémoire partagé et d'autres périphériques. En tant que périphérique de calcul, le GPU utilise un environnement d'exécution et un pilote de périphérique. Comme le montre Figure 4.7, un programme utilisant le GPU sera divisé en deux parties : un programme principal exécuté par le processeur hôte, dont une des tâches consistera à configurer le GPU, et éventuellement un ou plusieurs noyaux de calcul exécutés par le GPU.

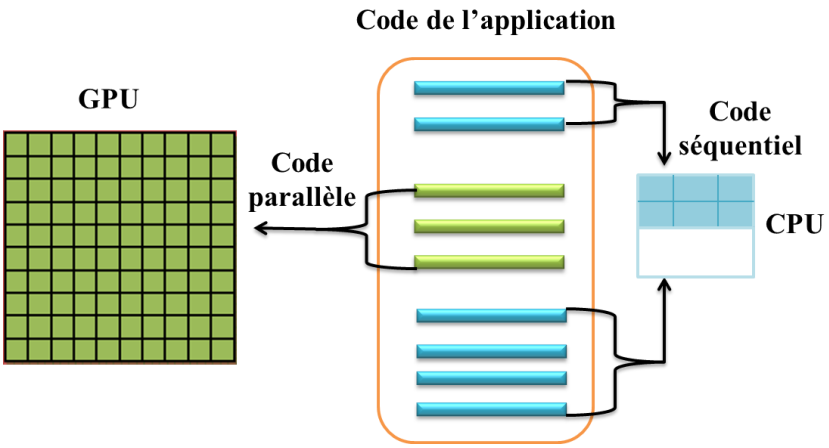

Figure 4.7 Exécution coopérative entre CPU et GPU.

Tout d'abord, le CPU exécute ses instructions séquentiellement dans le host. Le device et le host n'ont pas accès à leur mémoire respective. Donc, nous devons allouer la mémoire que le device doit utiliser pour l'exécution des instructions sur le GPU. Une fois la mémoire allouée, il faut transférer les données du host vers le device pour l'exécution d'une manière parallèle. Cette exécution se fait à l'aide d'une portion de code nommée kernel. Ce kernel est invoqué par le host et exécuté dans le device. Finalement, le résultat est transféré du GPU vers le CPU (voir la figure 4.8).

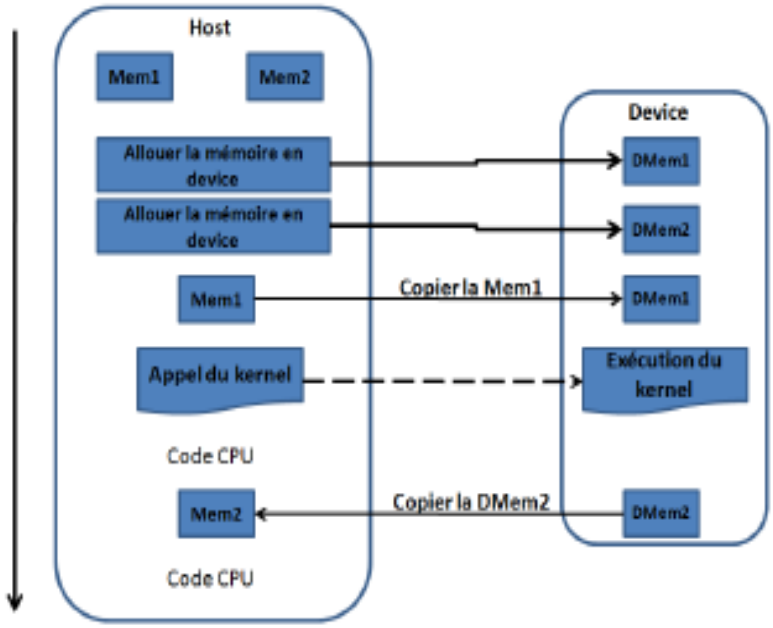

Figure 4.8 Modèle d'exécution sur GPU [96].

On peut voir dans la Figure 4.9 que le CPU n'a pas beaucoup d'Unités Arithmétiques et Logiques (UAL), ces unités sont très puissantes et elles utilisent une grande mémoire cache et une grande Unité de Contrôle (UC), et sa propre mémoire

globale DRAM. Cela permet au CPU de gérer des tâches multiples et différentes en parallèle qui nécessitent beaucoup de données. Ainsi, les données sont stockées dans un cache pour accélérer ses accès. L'unité de contrôle gère le flux d'instructions afin de maximiser l'occupation des UALs et d'optimiser la gestion du cache. Pour ces raisons, le CPU est très puissant et optimisé dans l'exécution séquentielle car il accède plus rapidement aux données. Par contre, le GPU a un grand nombre d'unités arithmétiques avec un cache limité et peu d'unités de contrôle, et sa propre mémoire globale DRAM. Ceci permet au GPU de calculer massivement et parallèlement le rendu de petits éléments indépendants, tout en ayant un grand flux de données traitées. Comme plus de transistors sont consacrés au traitement des données plutôt qu'à la mise en cache des données et au contrôle de flux, le GPU est spécialisé dans les calculs hautement parallèles et le calcul intensif.

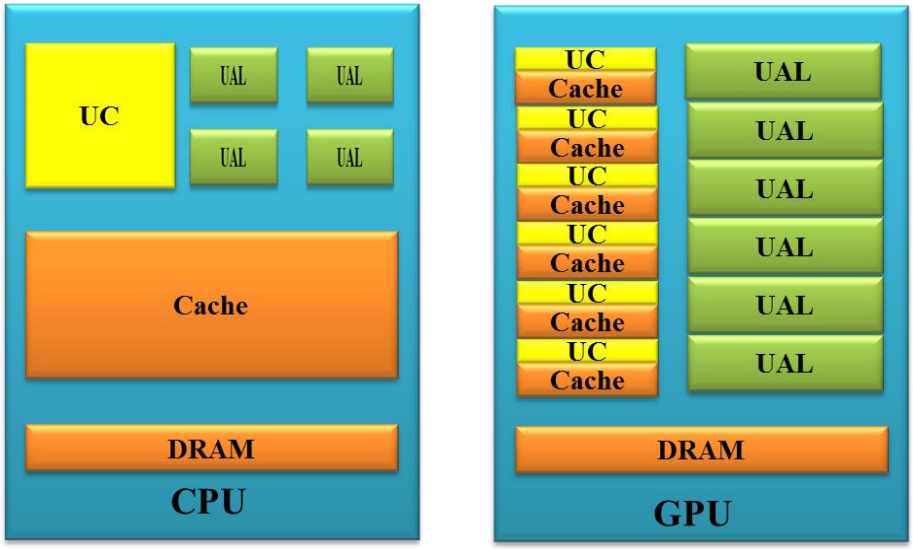

Figure 4.9 Différence entre CPU et GPU.

4.6.2. Défis des GPUs pour les métaheuristiques

L'optimisation combinatoire parallèle sur GPU n'est pas simple, et elle nécessite un énorme effort au niveau de conception ainsi qu'au niveau de l'implémentation. En effet, plusieurs défis scientifiques liés principalement à la gestion de la mémoire hiérarchique doivent être atteints. Les enjeux majeurs sont la distribution efficace du traitement des données entre CPU et GPU, la synchronisation des threads, l'optimisation du transfert de données entre les différentes mémoires, les contraintes de capacité de ces mémoires, etc. Ces problématiques doivent être prises en compte pour la reconception des modèles parallèles des métaheuristiques, afin de résoudre des problèmes d'optimisation à grande échelle sur les architectures GPU. Ainsi, il est nécessaire

d’identifier ces problèmes pour construire des métaheuristiques parallèles efficaces sur GPU.

- **Coopération entre le CPU et le GPU :** Une telle étape nécessite de définir la répartition des tâches dans les métaheuristiques. Dans ce but, l'optimisation du transfert de données entre ces deux composants est nécessaire pour obtenir les meilleures performances.

- **Contrôle du parallélisme :** Le calcul GPU est basé sur un multi-threading massivement parallèle, et l'ordre dans lequel les threads sont exécutés n'est pas connu. Par conséquent, d'une part, un contrôle de thread efficace doit être appliqué pour répondre aux contraintes de mémoire. D'autre part, un mappage efficace doit être défini entre chaque solution candidate et un thread désigné par un identifiant unique attribué lors de l'exécution du GPU.

- **Gestion de la mémoire :** L'optimisation des performances des applications GPU implique souvent l'optimisation des accès aux données, ce qui inclut l'utilisation appropriée des différents espaces mémoire GPU. Les différentes structures d'optimisation doivent être placées efficacement sur les différentes mémoires en tenant compte de leurs tailles et de leurs latences d'accès.

4.6.3. Calcul généraliste par GPU (GPGPU)

Grâce à des langages tels que CUDA (Compute Unified Device Architecture) **[97]** et OpenCL (Open Computing Language) **[98]**, les informaticiens sont capables d'exploiter le potentiel du GPU pour le calcul généraliste (General Purpose GPU, GPGPU), c.-à-d. implémenter des algorithmes parallèles en exploitant la structure et la puissance de calcul du GPU. Néanmoins, il faut une maîtrise de l’architecture GPU pour pouvoir implémenter des métaheuristiques qui exploitent au maximum les capacités du GPU. Pour implémenter notre BLA parallèle sur GPU, nous avons choisi le framework CUDA (Compute Unified Device Architecture). Cette boîte à outils promet une meilleure accélération sur GPU, et elle combine un modèle de programmation relativement accessible au travers d’un langage niveau C, et une architecture matérielle permettant les lectures et écritures mémoire arbitraires et offrant un accès à des mémoires locales et des synchronisations entre threads.

Selon l'architecture CUDA qui est illustrée dans la Figure 4.10, une carte graphique ***GPU*** appelé ***Device*** est utilisée par le processeur de la machine hôte (CPU)

comme coprocesseur de calcul. De cette façon, chaque GPU a sa propre mémoire et ses propres éléments de traitement qui sont séparés de l'hôte. Les données doivent être transférées entre l'espace mémoire de l'hôte vers la mémoire du GPU lors de l'exécution des programmes. Une carte graphique ***GPU*** peut être équipée de plusieurs GPU. Comme il est indiqué dans la figure, chaque GPU possède de nombreux multiprocesseurs (Streaming Multiprocessors, SMs), chacun avec son propre contrôle de flux et sa propre mémoire partagée. Chaque SM est constitué d'une série de cœurs ou d'éléments de traitement (Processing Elements, PEs).

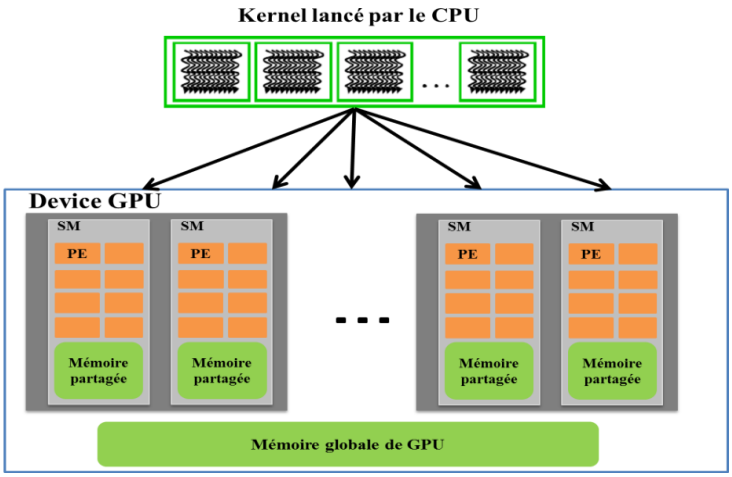

Figure 4.10 Les composants de GPU selon l'architecture CUDA. [97]

Le ***kernel*** (noyau) est une fonction appelable de l'hôte CPU et exécutée sur le GPU (*device)* par plusieurs SMs en parallèle. Ce kernel est dupliqué sur le GPU comme un ensemble de « ***threads »***. Cet ensemble de threads est organisé à l'intérieur de blocs de threads « ***threads blocks »***. Les blocs de threads peuvent être organisés en une grille appelée « ***Grid »*** qui est un tableau 1D, 2D ou 3D de « ***threads blocks »***. Au maximum 65536 blocks par dimension. Chaque bloc de threads est un tableau 1D, 2D ou 3D de « ***threads »*** chacun exécutant un clone (instance) du kernel. En général chaque block est composé 512 threads (maximum). Chaque bloc est identifié par un unique « ***blockId »*** et chaque thread est identifié par un unique « ***threadId »***. Lorsqu'un programme CUDA sur le CPU appelle une kernel, les blocs de threads de la grille sont énumérés et distribués aux SMs avec la capacité d'exécution disponible. La Figure 4.11 montre un exemple de Grille 2D de dimension 2 x 3 blocks. Chaque block est formé par 12 threads disposés dans une structure 2D de dimension (3x4).

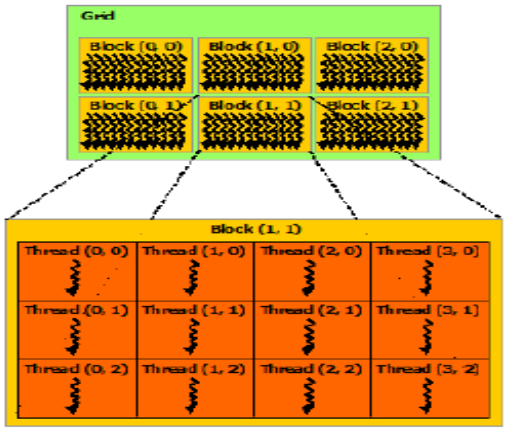

Figure 4.11 Le regroupement des threads en blocks dans une grille 2D selon l'architecture CUDA. [97]

Les threads d'un bloc partagent un logiciel de contrôle de cache (mémoire partagée) et un matériel de synchronisation rapide. La communication entre les blocs (la synchronisation et l'échange de données) nécessite un accès via une mémoire globale plus lente et déconnectée, Il est à noter que la synchronisation entre les threads d'un bloc est possible. De plus, la synchronisation entre blocs n'est également possible que sur les limites de kernel. L'ensemble de threads adjacents (appelés ***Warp***, actuellement 32 threads pour les GPU NVIDIA) dans un bloc doivent partager le même chemin de flux de contrôle pour s'exécuter en parallèle sur les PE. L'exécution sur GPU se fait en système SPMD (Singel Program Multiple Data). Ce système signifie que le même code ou programme, contenu dans le kernel, s'exécute sur différentes données.

Il existe plusieurs types de mémoire dans CUDA :

- Mémoire globale du GPU : la plus lente (400 à 600 cycles de latence, accessible en lecture et en écriture à toute la grille,
- Mémoire partagée : c'est un tampon partagé entre les threads dans le même bloc, elle est rapide mais limitée (16Ko par multiprocesseur), accessible en lecture et en écriture à tout un block.
- Registres (16384 par multiprocesseur) : rapide mais très limitée, accessible en lecture et en écriture à un thread.
- Mémoire locale : lente (200 à 300 cycles) et limitée, accessible en lecture et en écriture. Gérée automatiquement lors de la compilation quand les structures ou les tableaux sont trop grands pour être stockés dans les registres.

- Mémoire constante et texture : rapide, accessible en lecture uniquement depuis le GPU, et accessible en lecture et écriture depuis le CPU. C'est une mémoire constante de très petite taille (8 à 64 Ko).

4.7. Conclusion

Le calcul parallèle et distribué peut être utilisé dans la conception et l'implémentation des métaheuristiques pour plusieurs raisons. Cela permet de concevoir des méthodes d'optimisation interactives et en temps réel. Comme l'objectif de notre travail.

Dans ce chapitre, nous avons présenté un aperçu sur les systèmes parallèles et distribués. Nous avons procédé par une description des modèles parallèles des métaheuristiques, une classification des architectures parallèles suivie des concepts fondamentaux liés aux schémas d'exécution des algorithmes évolutionnaire. Enfin, nous parlons sur l'exploitation du potentiel de GPU pour le calcul généraliste (General Purpose GPU, GPGPU), c.-à-d. implémenter des algorithmes parallèles en exploitant la structure et la puissance de calcul de GPU.

Dans le chapitre suivant, nous passons à la version parallèle de notre contribution afin d'améliorer notre approche et pour accélérer le temps d'exécution par rapport à la version séquentielle.

Chapitre 5. BLA parallèle (P-BLA) pour résoudre le DCVRP (notre contribution)

Sommaire

5.1. Introduction

Le BLA représente une métaheuristiques inspirée de la vie des abeilles concernant les deux comportements, le mariage et la recherche de sources de nourriture. En utilisant des opérateurs comme la reproduction et la recherche local, respectivement. L'évolution artificielle dans le BLA explore plusieurs solutions de l'espace de recherche, afin de générer de bonnes solutions au problème traitée. Dans la version parallèle de BLA, les sous-populations (les colonies d'abeilles) évoluent sur différents processeurs. En plus, ce processus d'évolution artificielle peut être paralléliser de différentes manières sur des machines parallèles, au niveau de ces fonctions telles que l'initialisation, l'évaluation, la reproduction, le tri, et la recherche de nourritures.

Nous présentons dans ce chapitre notre approche P-BLA implémentée sur le GPU, pour résoudre le DCVRP. L'objectif est d'adapter notre nouvelle approche à l'unité de traitement graphique GPU afin de profiter de ses avantages. C'est pour cela, en utilise le calcul généraliste par GPU (GPGPU) via l'une des langages dédiés tel que le CUDA (Compute Unified Device Architecture). Pour l'implémentation parallèle de P-BLA, en spécifiant ces différent kernels.

5.2. BLA parallèle (P-BLA)

Après la construction de *m* groupes par le k-means amélioré (algorithme 2.2) pour une instance CVRP de DCVRP dans une tranche de temps t_i, on obtient *m* sous-problèmes, dans lesquels le plus court chemin (en termes de distance euclidienne minimum) sera calculé. Ces sous-problèmes peuvent être résolus indépendamment en parallèle à l'aide de l'algorithme Parallel Bees Life Algorithm (P-BLA) (parallélisme inter-P-BLA) **[99]**. D'un autre côté, il y a un processus parallèle intra-P-BLA parce qu'il n'y a pas de dépendance entre les abeilles d'une colonie pour le processus d'initialisation, l'évaluation, …, et la recherche de nourriture. Par conséquent, toute la colonie d'abeilles peut être exploitée en parallèle pour chaque génération. Dans cette section, nous décrivons la conception du P-BLA sur CUDA en détail, ce qui apporte une bonne efficacité. Pour modifier le BLA standard en BLA parallèle et pour l'exécuter sur CUDA, nous devons surmonter de nombreux problèmes d'implémentation, et les examiner en détail comme suit.

Dans notre proposition parallèle, nous avons utilisé le "Modèle parallèle au niveau de l'algorithme" (voir la section 4.3), où plusieurs instances de P-BLA sont simultanément lancées dans des blocs de threads séparés pour calculer les meilleures solutions. Outre la recherche d'accélération, une amélioration de la qualité de la solution devrait également être recherchée dans ce modèle parallèle. Pour implémenter le P-BLA sur GPU, nous avons utilisé le modèle d'îlot (voir la Figure 4.5) - sans échange entre les sous-populations de chaque îlot-. Dans lequel, nous avons mappé la résolution du problème global (multi-colonies) en *m* blocs de *N* threads correspondant à *m* colonies (îlots) de *N* abeilles. Chaque bloc de threads est responsable de l'évolution d'une sous-population (c'est-à-dire d'une colonie d'abeilles), qui est associée à un sous-problème obtenu et dérivé en appliquant le k-means amélioré (algorithme 2.2). Chaque bloc de threads exécute toutes les étapes de BLA à partir de l'initialisation à la recherche de nourriture, avec les mêmes valeurs de paramètres.

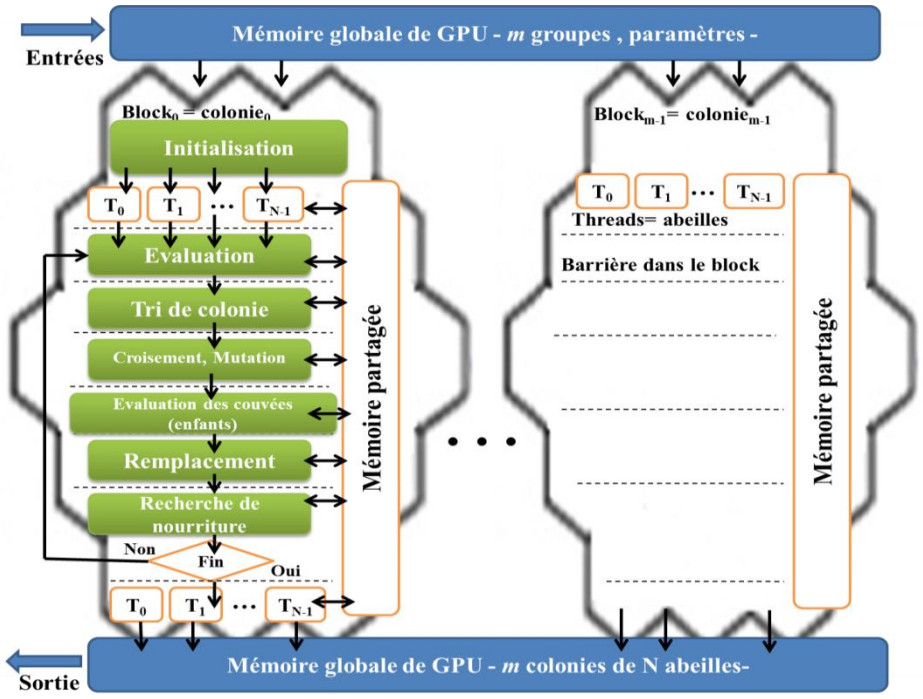

Figure 5.1 Mappage de BLA sur CUDA en utilisant le modèle d'îlot distribué.

La Figure 5.2 illustre que nous avons adopté l'implémentation synchrone, qui comprend les six kernels de notre P-BLA (Initialisation, Evaluation, Tri des colonies d'abeilles, Reproduction, Remplacement, et la recherche de nourriture). Chaque kernel est parallélisé pour qu'il soit exécuté par un bloc de threads pour chaque sous-problème. Afin d'effectuer une génération de P-BLA pour un sous-problème, les six kernels doivent être exécutés séquentiellement, où la synchronisation devrait avoir lieu à la fin de chaque kernel. Les blocs de threads utilisent les mêmes valeurs de paramètre d'évolution telles que la taille de la population, le taux de mutation et de croisement, dans le but de trouver la meilleure tournée dans chaque sous-problème.

Les threads dans les blocs peuvent être synchronisés en utilisant les barrières afin de maintenir la cohérence des données. Une synchronisation de barrière définit un point de synchronisation où chaque thread doit attendre que tous les autres threads aient également atteint ce point de synchronisation. Ainsi, aucun thread dans le bloc n'exécute aucune instruction après le point de synchronisation jusqu'à ce que tous les autres threads soient également arrivés à ce point. Une synchronisation de barrière a également pour effet de définir un état global de l'espace d'adressage partagé dans lequel toutes les opérations spécifiées avant le point de synchronisation ont été exécutées. Les instructions après le point de synchronisation peuvent être sûres dès lors que cet état global ait été établi.

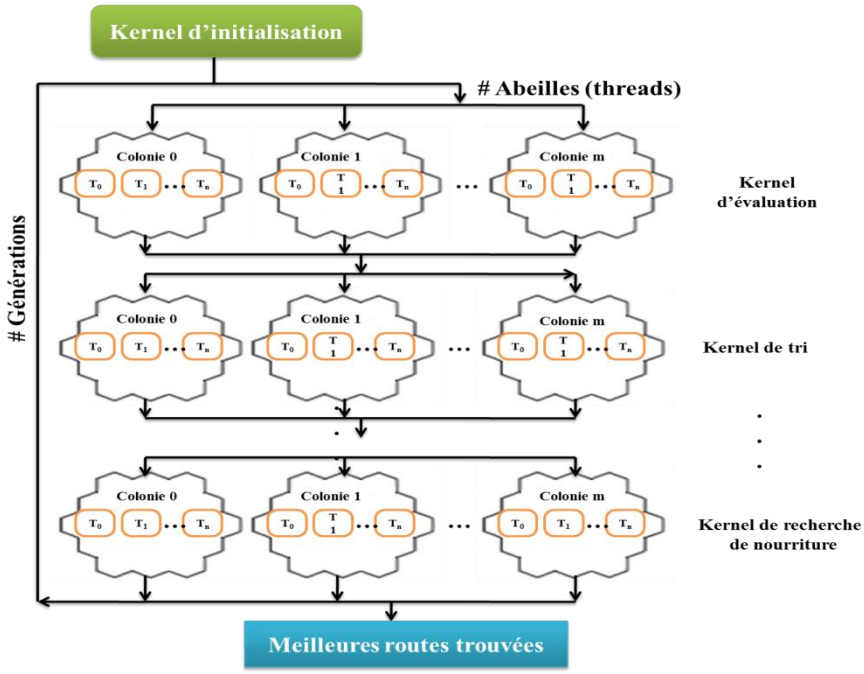

Figure 5.2 Implémentation synchrone de P-BLA.

Nous mentionnons qu'il n'y a pas de phase de migration entre les îlots car tous les îlots sont indépendants, donc il n'y a pas de communication entre les îlots. Lorsque plusieurs îlots atteignent un nombre fixe de générations de P-BLA (généralement 1000 générations), chaque bloc de threads copie ses abeilles évoluées dans la mémoire globale de GPU, d'où elles seront transférées vers la mémoire globale de CPU. Finalement, nous calculons la somme des fitness de la première meilleure abeille de chaque colonie ordonnée pour obtenir la longueur totale de routes des véhicules.

Pour implémenter notre idée, chaque thread est affecté à une abeille pour rechercher la meilleure solution dans sa colonie. Par conséquent, le nombre de threads distribués sur les blocs CUDA est égal au nombre d'abeilles dans la colonie ($N = Q + W + D$). Les blocs communiquent entre eux via la mémoire globale de GPU. A l'intérieur des blocs, les threads communiquent entre eux via la mémoire partagée qui est un peu plus rapide. Le code dédié au GPU est représenté sous forme de kernels qui sont des fonctions distinctes, et qu'ils sont entièrement exécutés dans le GPU. Les kernels ne peuvent utiliser que les données conservées dans la mémoire de GPU. Cette mémoire est allouée au moyen de la fonction cudaMalloc (). La fonction cudaMemcpy () est responsable du transfert de données entre la mémoire globale de CPU et la mémoire globale de GPU (et vice versa). L'ensemble des kernels utilisés dans notre contribution sera expliqués en détails.

5.2.1. Kernel d'initialisation

Dans l'approche de BLA séquentiel, l'initialisation des *N* abeilles (qui représente les tournées initiales dans chaque sous-problème) dans *m* colonies sera réalisée une par une. Dans le P-BLA, cela peut être fait en parallèle. Par conséquent, P-BLA les distribuera et les calculera via des threads parallèles dans GPU (voir Figure 5.3).

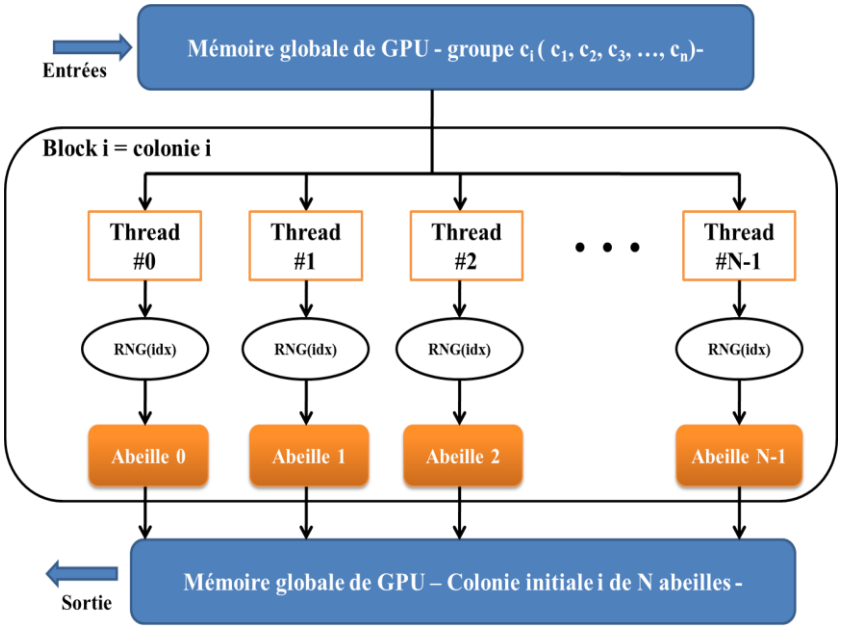

Figure 5.3 Initialisation parallèle d'une colonie d'abeille sur GPU.

Dans la première étape, nous devons définir le nombre de blocs dans la grille et le nombre de threads par bloc qui est égal à *m* colonies et *N* abeilles, respectivement. Chaque bloc possède un numéro d'identification de 0 à *m*-1, et chaque thread possède un numéro d'identification de 0 à *N*-1. En outre, nous devons premièrement allouer la mémoire globale de GPU, puis nous transférons les données requises de CPU vers le GPU pour les *m* groupes du problème globale (DCVRP). L'algorithme doit utiliser CuRAND () **[97]**, qui est une bibliothèque NVIDIA CUDA pour la génération de nombres aléatoires afin de générer des tournées initiales aléatoires sur le GPU. Il est très flexible et accessible via le fichier d'en-tête curand_kernel.h.

5.2.2. Kernel d'évaluation

Dans la version parallèle de BLA, nous maintenons la même fonction de fitness utilisée pour l'évaluation des abeilles dans le BLA séquentiel, pour atteindre une solution acceptable, dans laquelle la fitness de chaque abeille est calculée de manière parallèle en utilisant un thread par abeille (voir la Figure 5.4). L'utilisateur peut accéder à l'abeille comme un tableau 1D et écrire la valeur de fitness calculée dans ce dernier.

Le fragment du code C réservé au kernel d'évaluation sera exécuté par chaque thread de bloc, pour calculer la fitness de chaque abeille. Ceci est possible car CUDA est compatible avec C. Finalement, les fitness calculés sont copiés dans la mémoire partagée du GPU.

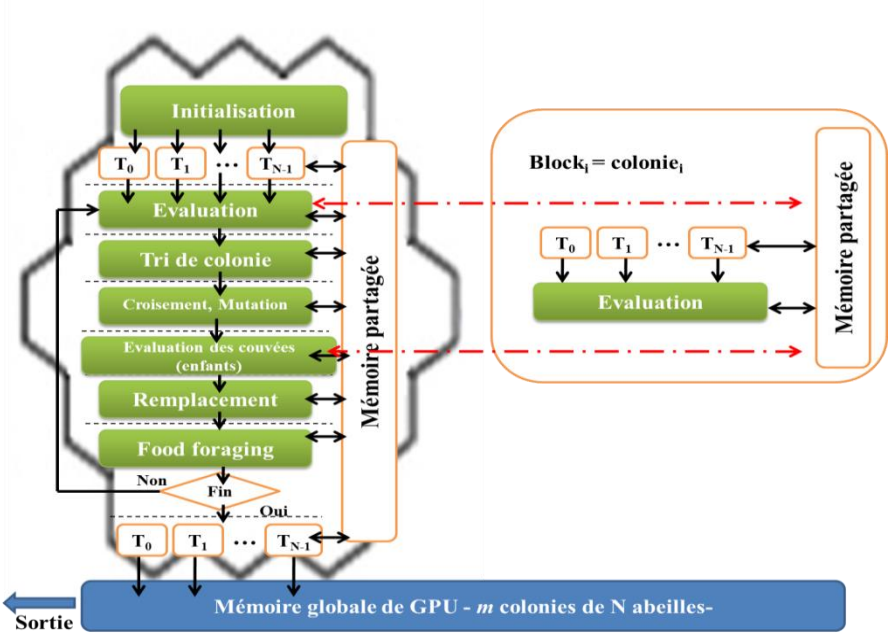

Figure 5.4 Évaluation parallèle des abeilles d'une colonie sur GPU.

5.2.3. Kernel de tri des colonies d'abeilles

Garder la population triée selon la valeur de fitness des abeilles est une étape difficile. Dans notre étude, nous avons choisi le tri parallèle par transposition paire-impaire (odd-even), c'est un algorithme de tri parallèle basé sur la technique de tri à bulles (Bubble Sort). Dans laquelle, les paires de threads adjacents dans un bloc sont échangés s'ils ne sont pas ordonnés (voir la Figure 5.6), jusqu'à ce que tous les threads dans ce bloc soient dans l'ordre.

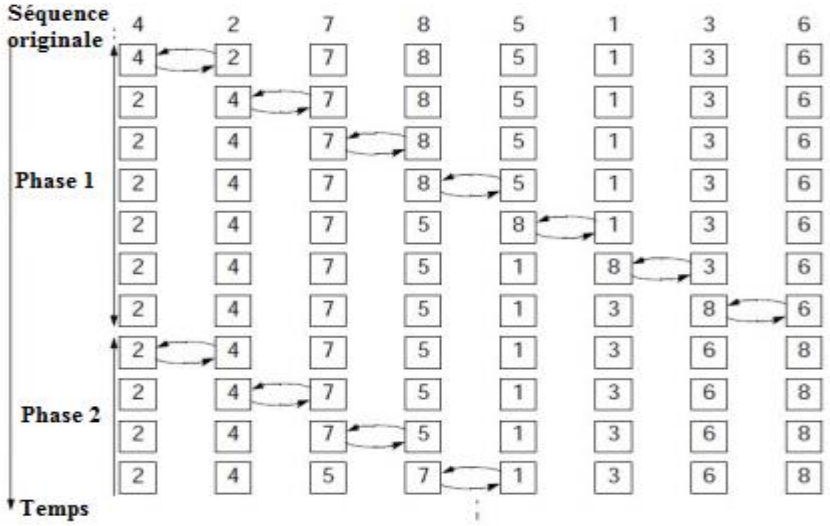

Figure 5.5 Le principe de tri à bulle.

La technique de tri parallèle par transposition paire-impaire (odd-even) distincte de tri à bulles (voir la Figure 5.5), où ce tri parallèle consiste à travailler sur des paires disjointes, c'est-à-dire en utilisant des paires alternées de threads pairs-impairs et impairs-pairs du bloc. La technique fonctionne en plusieurs passes sur une file Q de taille *N*. A chaque passe, les threads avec des identificateurs impairs dans le bloc effectuent un contrôle de comparaison basé sur le tri à bulles, après quoi les threads ayant des identifiants pairs dans le bloc font la même chose. Le nombre maximal d'itérations ou de passes pour le tri par transposition paire-impaire parallèle est N / 2. La durée totale de cette technique est O (log 2N) **[100]**.

L'algorithme fonctionne comme suit:

Algorithme 5.1 Le tri parallèle par transposition paire-impaire (odd-even)

Pour k = 1→N/2 **faire**
Faire en parallèle
Si $x_i > x_{i+1} \forall$ i%2 !=0 **alors**
échanger x_i, x_{i+1};
Fin si
Fin parallèle
Faire en parallèle
Si $x_i > x_{i+1} \forall$ i%2 ==0 **then**
échanger x_i, x_{i+1};
Fin si
Fin parallèle
Fin pour

Rappelons que dans le code CUDA, nous avons *m* blocs séparés de *N* threads qui est égal à *m* colonies et *N* abeilles respectivement, ces blocs ne peuvent pas se

synchroniser les uns avec les autres. De plus, nous utilisons _syncthreads () dans chaque bloc pour la plupart des opérations de comparaison et d'échange.

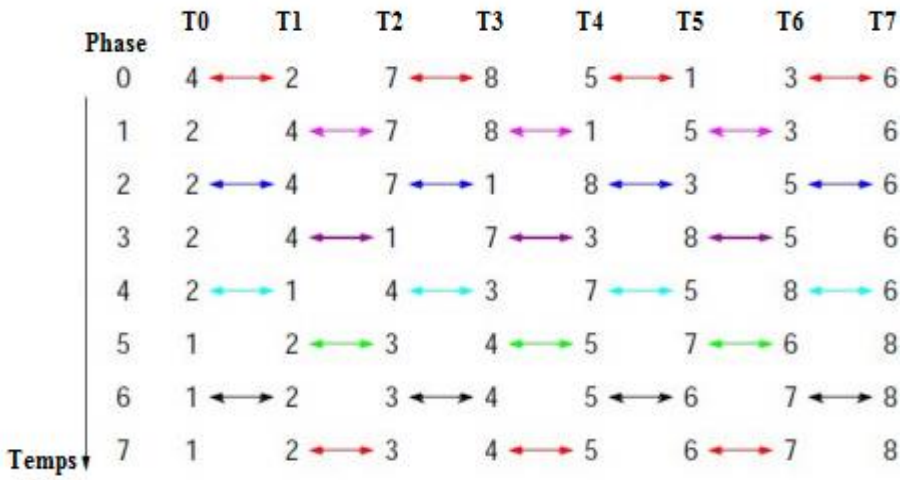

Figure 5.6 Le principe de de tri parallèle par transposition paire-impaire.

5.2.4. Kernel de reproduction

a. Kernel de croisement

L'opérateur de croisement à deux points (crossover 2X) est implémenté différemment dans le P-BLA par rapport à son homologue séquentiel, où nous devons utiliser CuRAND () pour générer de manière aléatoire les deux valeurs de points de coupure. P-BLA exécute les opérations de croisement multiples sur les *m* blocs (voir la Figure 5.7), dans lequel un thread effectue un croisement entre la reine et un faux-bourdon choisis au hasard dans le même bloc, afin de générer deux couvées (abeilles descendantes). Ce processus est répété jusqu'à atteindre *N* abeilles avec une probabilité de croisement p_c == [0.45, 0.95]. Par conséquent, nous avons *N / 2* threads par bloc. Tous les threads effectuant le croisement doivent utiliser des points de croisement communs.

b. Kernel de mutation

Dans la version parallèle de BLA, pour chaque thread dans les *m* blocs et qui correspond à une couvée (abeille descendante), on décide de la muter ou pas en fonction de sa probabilité de mutation p_m = [0.001, 0.01] (voir la Figure 5.7). L'opérateur de mutation utilisé dans le P-BLA est similaire à ce qu'il est utilisé dans le BLA séquentielle, où on choisit l'opérateur de mutation par insertion. Cependant, dans le P-BLA on doit utiliser la fonction CuRAND () pour générer aléatoirement la valeur du

point de mutation, et tous les threads dans le même bloc connaissent la même valeur de point de mutation.

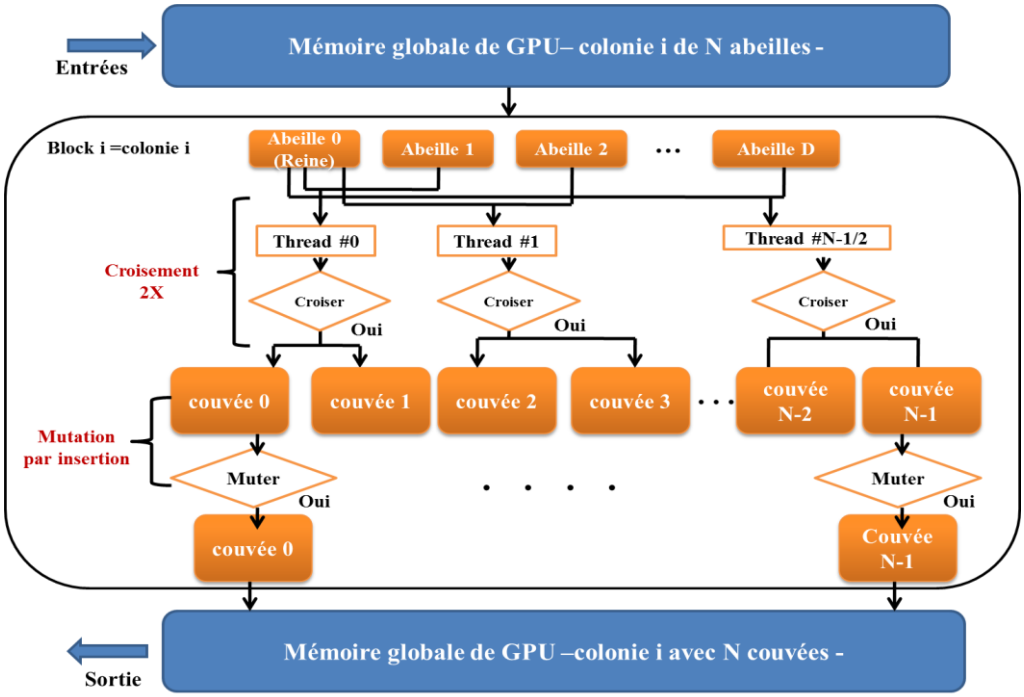

Figure 5.7 Reproduction parallèle d'une population d'abeilles en utilisant le croisement 2X et la mutation d'insertion sur GPU.

5.2.5. Kernel de remplacement

La stratégie de remplacement considérée dans P-BLA est la même que celle utilisée dans la BLA séquentielle, qui est le remplacement élitiste avec la même décomposition des blocs, à l'exception des dimensions des blocs, qui sont dérivées de la taille de la population des descendants.

Chaque thread dans le bloc correspond à une couvée générée après la phase de reproduction qui est comparée à ses parents. Si la fitness de cette couvée est plus élevée que l'un de ses parents, cette couvée remplacera ce dernier dans l'ancienne population (colonie) *i* pour générer une nouvelle population i + 1. Notons que les threads dans le même bloc sont synchronisés à l'aide de la fonction _syncthreads ().

5.2.6. Kernel de la recherche de nourriture

Nous associons chaque abeille à un thread dans le bloc pour chercher la meilleure solution dans sa colonie. Par conséquent, le nombre de threads répartis entre les blocs de threads CUDA est égal au nombre d'abeilles dans la colonie (N = Q + W +

D). La recherche de voisinage parallèle est appliquée sur les derniers *W* threads dans le bloc. Elle est basée sur l'opérateur *swap* qui consiste à échanger l'emplacement de deux éléments choisis aléatoirement dans un thread, qui est affecté à une abeille ouvrière en utilisant CuRAND (). Pour une abeille de taille *n*, la taille de ce voisinage est *n (n-1) / 2*. P-BLA utilise plus de threads recrutés pour les *B* meilleures régions parmi les *W* régions, où *B* est un paramètre défini par l'utilisateur.

5.3. Conclusion

Dans ce chapitre, nous expliquant notre nouvelle méthode d'optimisation combinatoire parallèle basée sur l'unité de traitement graphique (GPU), qui est nommée Parallel Bees Life Algorithm (P-BLA) pour résoudre le DCVRP. Cette proposition tire parti de la puissance des GPGPUs, en particulier P-BLA a été développé en utilisant l'architecture CUDA de NVIDIA. Afin de surmonter la complexité de calcul de la version séquentielle de cette méthode, ainsi que de réduire le temps d'exécution pour découvrir les routes de manière dynamique. Dans le chapitre suivant nous allons faire une étude expérimentale sur l'algorithme P-BLA, et sa version séquentiel. Cette étude consiste à établir une comparaison entre ces deux derniers, et entre d'autres méthodes se trouvent dans la littérature pour résoudre le DCVRP. Dans le but d'évaluer la performance et la robustesse de notre contribution.

Chapitre 6. Étude expérimentale et résultats

Sommaire

6.1. Introduction

Dans ce chapitre, Nous avons mené quelques expérimentations sur des instances de problème de tournées de véhicules dynamiques de service de collecte. Le but de cette étude expérimentale est d'évaluer la performance de nos algorithmes proposés BLA et P-BLA, par rapport à d'autres approches qui existent dans la littérature. L'évaluation de performance concerne les meilleures solutions trouvées, et la précision de chaque approche. En plus, nous avons étudié l'effet d'accélération des GPU sur le P-BLA par rapport à BLA.

Nous avons implémenté nos deux algorithmes, en utilisant C/C ++ sous l'environnement Visual Studio C ++ 10.0, et CUDA version 5.0. Tous les tests ont été effectués avec les paramètres suivants de la machine : processeur Intel Core i7 (2,2 GHZ), avec le périphérique GPU NVIDIA GeForce 710M.

6.2. Description des benchmarks

Nous rapportons les résultats produits par le BLA séquentiel et le P-BLA sur l'ensemble des données de benchmarks dynamiques, qui sont proposés par Montemanni et al. **[19]**. Ces dernières ont été organisées en deux classes, des instances de service de collecte et des instances de service de livraison. Il faut noter que nous avons testé nos algorithmes proposés avec les instances de collecte (les instances avec des demandes positives).

Les instances sont nommées comme suit : Taillard **[101]** (13 instances), Christophides et Beasley **[102]** (7 instances) et Fisher et al. **[103]** (2 instances). La résolution de chaque instance est limitée par une durée de la journée de travail T, qui correspond à la fenêtre de temps de dépôt. Les clients ou les demandes dans ces instances possèdent un temps d'apparence (c'est-à-dire lorsque la demande est connue par le système), et une durée qui représente le temps nécessaire pour servir cette demande. Afin d'obtenir des instances dynamiques, certaines caractéristiques ont été ajoutées aux instances (voir l'annexe A) :

- La longueur de la journée de travail [0, T] ;
- Le temps d'occurrence de chaque demande. Il contient, pour chaque demande, le moment du jour de travail, lorsque la commande est connue par le planificateur ;
- La durée de service de chaque demande. Il représente, pour chaque demande, le temps nécessaire pour servir le client correspondant ;
- Le Nombre de véhicules Il contient la dimension en nombre de véhicules de la flotte disponible pour servir les clients.

Le nombre de clients à servir dans ces instances varie de 50 à 199 clients, et ce nombre peut être déduit du nom de l'instance. Par exemple, c120 correspond à l'instance de Christophides avec 120 clients et un seul dépôt.

6.3. Réglage des Paramètres

Afin de comparer l'efficacité de notre P-BLA par rapport au BLA séquentiel et aux autres algorithmes qui résolvent le DCVRP, nous adoptons les valeurs suivantes, qui ont été choisies en fonction des paramètres de ce type de problème proposés par Montemanni et al. **[19]** comme il est indiqué dans la Table 6.1.

Pour résoudre le DVRP, Montemanni et al. avaient fixé le temps de coupure $T_{co} = 0{,}5 \times T$, où T est la longueur de journée de travail. Par conséquence, les demandes qui arrivent après T_{co} sont reportées le jour suivant et elles sont considérées comme des demandes statiques, alors que celles qui arrivent avant ce temps sont considérées comme des demandes dynamiques. Etant donné que, les clients arrivent selon une distribution uniforme au cours de la journée de travail. Avec $T_{co} = 0{,}5 \times T$, la moitié des clients sont considérés comme des clients dynamiques, tandis que les autres sont des clients statiques. Cela correspond à un *dod* = 0,5 (voir la section 1.3.4).

Le temps d'engagement avancé (advanced commitment time, T_{ac}), qui permet à un conducteur de réagir aux nouvelles demandes avant leur traitement, n'est pas utilisé. Pour cette raison, il est égal à 0.

Table 6.1 Les Paramètres de DCVRP.

Paramètres	Description	Valeur
n_{ts}	Nombre de tranche du temps	25
T_{co}	Temps de coupure	$T \times 0.5$
T_{ac}	Temps d'engagement avancé	0
T	la longueur de journée de travail	Depot time window

De plus, la sélection des paramètres spécifiques du BLA est une décision très importante qui a un impact direct sur la qualité de la solution trouvée, ainsi que sur le temps d'exécution. Dans nos expérimentations, ces paramètres sont définis comme il est indiqué dans la Table 6.2.

Table 6.2 Les Paramètres de BLA.

Paramètres	Valeurs
Taille de population : Nombre de faux-bourdons Nombre d'ouvrières	301 200 100
Nombre d'itérations (critère d'arrêt)	1000
Taux de mutation	0.05(5%)
Taux de croisement	0.75(75%)

6.4. Résultats et Discussion

La Figure 6.1 illustre les résultats obtenus de regroupement par le k-means amélioré de certaines instances de benchmark de DCVRP proposées par Montemanni et al. **[19]**. Ici, les points de même couleur décrivent les clients d'un même groupe avec une demande totale inférieure ou égale à la capacité maximale du véhicule.

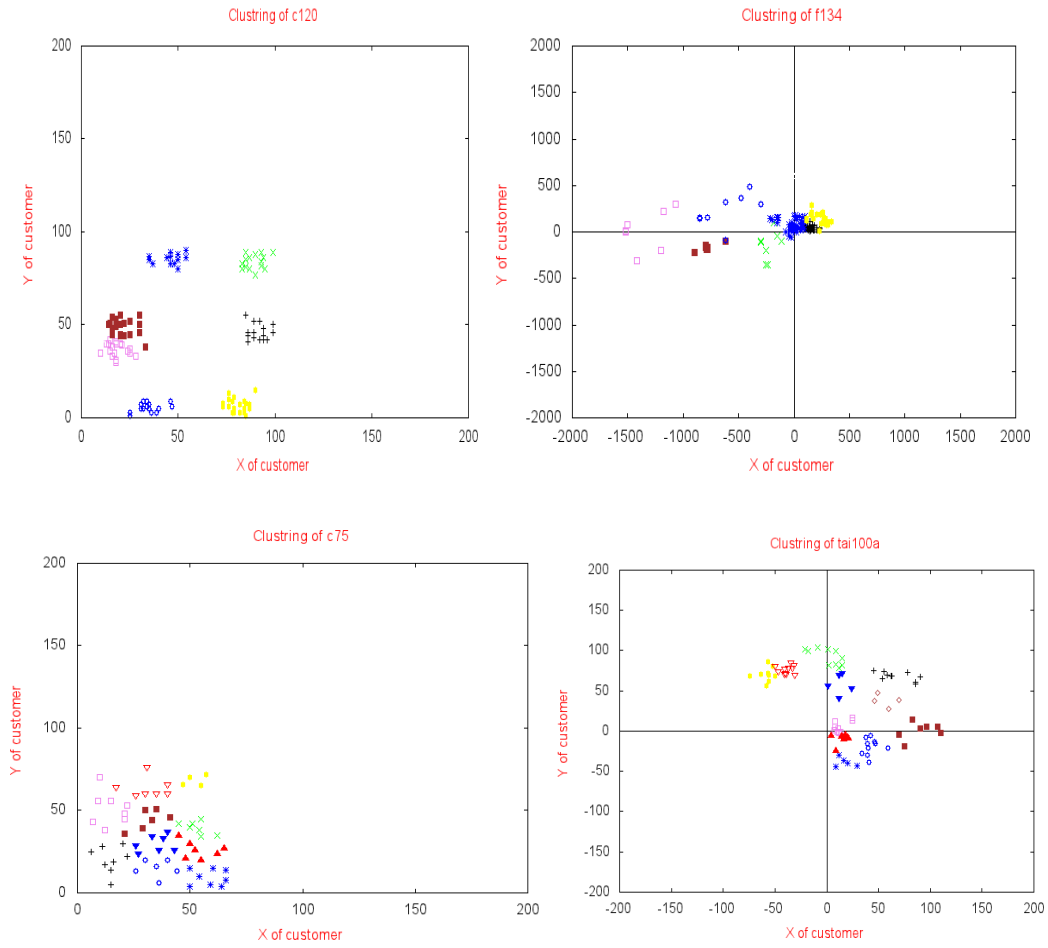

Figure 6.1 Résultats du regroupement avec k-means amélioré de certaines instances de DCVRP.

Les résultats obtenus par k-means amélioré ont prouvé l'utilité de l'étape de regroupement proposée, où le DCVRP à grande échelle est divisé en plusieurs sous-problèmes. L'objectif de partition par zones est de trouver des petits groupes de clients géographiquement proches. Les groupes obtenus à partir de cette première étape représentent l'ensemble de tournées de véhicules initiales, qui seront optimisées après par notre BLA proposée dans ces deux versions séquentielles ou parallèles. Dans la section suivante, nous allons comparer nos résultats présentés dans la Table 6.3, avec les résultats d'autres approches qui résout le DCVRP.

La Table 6.3 rapporte les meilleures solutions trouvées pour résoudre les instances de collecte de DCVRP de Montemanni et al. **[19]**, en utilisant l'algorithme génétique (Genetic Algorithm, GA) **[63]**, la recherche tabou (Tabu Search, TS) **[63]** et la colonie de fourmi (Ant System, AS) **[19]**. Dans ce cas, le conducteur du véhicule ne se préoccupe pas de ce qui est transporté, mais seulement de la quantité à prélever auprès du client. Nous avons mis les meilleures solutions trouvées dans les cellules

ombrées foncées et les moyennes des résultats dans les cellules claires, pour 10 exécutions des algorithmes sur chaque instance.

Les résultats présentés dans la Table 6.3 montrent que P-BLA produit des solutions de qualité par rapport aux autres algorithmes. L'algorithme P-BLA fournit également la distance totale parcourue la plus courte dans les 12 instances parmi les 21 instances de Kilby **[27]**. Par exemple, avec l'instance de Taillard « tai75a », la meilleure et la moyenne solution trouvée par P-BLA sont respectivement 1646.48 et 1749.02. Ce résultat est décrit dans le format suivant : {nom de l'instance}: {meilleur distance totale parcourue}: \\ {numéro de route} {tournée du véhicule}. Les tournées trouvées par P-BLA sont détaillées dans l'annexe B de la thèse, pour chaque instance utilisée.

Benchmark tai75a: best cost =1646.48
Route 1: 0 -24 -18 -27 -21 -19 -25 -15 -23 -0
Route 2: 0 -4 -6 -11 -2 -3 -10 -9 -7 -1 -8 -5 -0
Route 3: 0 -66 -51 -67 -65 -26 -53 -0
Route 4: 0 -56 -58 -63 -39 -50 -60 -54 -69 -0
Route 5: 0 -36 -44 -35 -43 -47 -38 -55 -46 -64 -57 -0
Route 6: 0 -74 -75 -0
Route 7: 0 -20 -13 -12 -16 -17 -14 -22 -28 -0
Route 8: 0 -71 -73 -72 -70 -52 -62 -0
Route 9: 0 -33 -59 -31 -68 -29 -42 -45 -30 -61 -0
Route 10: 0 -40 -41 -34 -32 -37 -48 -49 -0

Table 6.3 Comparaison des performances de nos approches (BLA, P-BLA) sur les instances de DCVRP de collecte.

Instances	Metaheuristiques									
	AS [19]		GA [63]		TS [63]		BLA		P-BLA	
	Meilleure solution	Moyenne	Meilleure Solution	Moyenne	Meilleure solution	Moyenne	Meilleure Solution	Moyenne	Meilleure Solution	Moyenne
c50	631.30	681.86	570.89	593.42	603.57	627.90	602.66	624.47	592.47	639.59
c75	1009.36	1042.39	981.57	1013.45	981.51	1013.82	906.78	939.94	858.23	923.22
c100	973.26	1066.16	961.10	987.59	997.15	1047.60	1025.26	1104.15	992.28	1113.32
c100b	944.23	1023.60	881.92	900.94	891.42	932.14	970.24	1027.19	938.41	1031.31
c120	1416.45	1525.15	1303.59	1390.58	1331.22	1468.12	1274.65	1305.74	1229.63	1298.10
c150	1345.73	1455.50	1348.88	1386.93	1318.22	1401.06	1368.90	1456.37	1372.61	1488.16
c199	1771.04	1844.82	1654.51	1758.51	1750.09	1783.43	1683.70	1734.75	1631.92	1689.78
f71	311.18	358.69	301.79	309.94	280.23	306.33	334.50	370.12	328.24	391.18
f134	15135.51	16083.56	15528.81	15986.84	15717.90	16582.04	15248.20	16325.18	14948.90	16308.47
tai75a	1843.08	1945.20	1782.91	1856.66	1778.52	1883.47	1729.88	1787.54	1646.48	1749.02
tai75b	1535.43	1704.06	1464.56	1527.77	1461.37	1587.72	1495.10	1580.17	1407.02	1489.94
tai75c	1574.98	1653.58	1440.54	1501.91	1406.27	1527.72	1428.31	1522.47	1415.15	1535.61
tai75d	1472.35	1529.00	1399.83	1422.27	1430.83	1453.56	1380.90	1472.11	1378.02	1405.63
tai100a	2375.92	2428.38	2232.71	2295.61	2208.85	2310.37	2177.85	2328.75	2206.20	2323.52
tai100b	2283.97	2347.90	2147.70	2215.93	2219.28	2330.52	2167.97	2292.42	2087.57	2214.78

tai100c	1562.30	1655.91	1541.28	1622.66	1515.10	1604.18	1567.53	1646.01	1574.59	1636.30
tai100d	2008.13	2060.72	1834.60	1912.43	1881.91	2026.76	1923.41	2012.23	1882.09	1995.85
tai150a	3644.78	3840.18	3328.85	3501.83	3488.02	3598.69	3288.04	3388.96	3202.86	3202.86
tai150b	3166.88	3327.47	2933.40	3115.39	3109.23	3215.32	3018.90	3103.66	2861.18	3044.30
tai150c	2811.48	3016.14	2612.68	2743.55	2666.28	2913.67	2544.15	2725.64	2513.92	2660.96
tai150d	3058.87	3203.75	2950.61	3045.16	2950.83	3111.43	2899.26	3040.17	2722.22	3043,23
Totale	50876.23	53794.02	49202.73	51089.37	49987.8	52725.85	49036.19	51788.03	47789.99	51264.66

6.4.1. Mesure de performance dynamique

Pour mesurer les performances dynamiques de P-BLA conçues pour le DCVRP, nous avons choisi la métrique de précision (accuracy) selon les indicateurs de Weicker **[104]**. La précision devrait mesurer la proximité de la meilleure solution connue (trouvée dans la littérature) à la meilleure solution trouvée par P-BLA. Formellement, la précision à l'instant *t* d'une fonction de fitness *F* correspondant à un algorithme d'optimisation *A* est définie par l'équation (4.1) **[104]** :

$$accuracy_{F,A}^{(t)} = \frac{F\left(best_A^{(t)}\right) - Min_F^{(t)}}{Max_F^{(t)} - Min_F^{(t)}} \qquad (6.1)$$

Où $best_A^{(t)}$is est la meilleure solution trouvée par l'algorithme (*A*) à l'instant t. Le fitness maximum et minimum les valeurs dans l'espace de recherche sont représentées par $Max_F^{(t)}$ et $Min_F^{(t)}$. La précision d'optimisation est comprise entre 0 et 1, où une précision égale à 1 est la valeur idéale. La précision a été calculée en utilisant les meilleures solutions connues des instances statiques, comme une borne inferieure pour calculer cette précision ($Min_F^{(t)}$ dans l'équation 6.1). Ces meilleures solutions connues considèrent tous les clients comme étant statiques, et qui ne sont donc pas des solutions réalisables pour le DVRP. Ils peuvent être considérés comme une limite pour nos algorithmes.

Dans cette étude, BLA séquentiel et P-BLA ont été exécutés sur les 21 instances **[19]** pour mesurer la précision des tournées découvertes en utilisant l'équation 6.1. Les résultats atteints sont représentés dans le tableau 4.4 et comparés à ceux obtenus par les autres métaheuristiques : le système de fourmis (Ant System, AS), l'algorithme génétique (Genetic Algorithm, GA), et la recherche Tabou (Tabu Search, TS). A partir de la Table 6.4, nous voyons que P-BLA atteint la plus grande précision pour trouver la meilleure solution avec une valeur moyenne égale à 0,89.

Table 6.4 Précision de BLA, P-BLA, et les autres métaheuristiques sur les instances de DCVRP.

Instances	Min^{T}_{F}	Metaheuristiques									
		AS [19]		GA [63]		TS [63]		BLA		P-BLA	
		Meilleure solution	Précision	Meilleure Solution	précision	Meilleure solution	précision	Meilleure Solution	précision	Meilleure Solution	précision
c50	524.61	631.3	0.83	570.89	0.92	603.57	0.87	602.66	0.87	592.47	0.89
c75	835.26	1009.36	0.83	981.57	0.85	981.51	0.85	906.78	0.92	858.23	0.97
c100	826.14	973.26	0.85	961.1	0.86	997.15	0.83	1025.26	0.81	992.28	0.83
c100b	819.56	944.23	0.87	881.92	0.93	891.42	0.92	970.24	0.84	938.41	0.87
c120	1042.11	1416.45	0.74	1303.59	0.80	1331.22	0.78	1274.65	0.82	1229.63	0.85
c150	1028.42	1345.73	0.76	1348.88	0.76	1318.22	0.78	1368.9	0.75	1372.61	0.75
c199	1291.45	1771.04	0.73	1654.51	0.78	1750.09	0.74	1683.7	0.77	1631.92	0.79
f71	241.97	311.18	0.78	301.79	0.80	280.23	0.86	334.5	0.72	328.24	0.74
f134	11620.96	15135.51	0.77	15528.81	0.75	15717.9	0.74	15248.2	0.76	14948.9	0.78
tai75a	1618.36	1843.08	0.88	1782.91	0.91	1778.52	0.91	1729.88	0.94	1646.48	0.98
tai75b	1344.64	1535.43	0.88	1464.56	0.92	1461.37	0.92	1495.1	0.90	1407.02	0.96
tai75c	1291.01	1574.98	0.82	1440.54	0.90	1406.27	0.92	1428.31	0.90	1415.15	0.91
tai75d	1365.42	1472.35	0.93	1399.83	0.98	1430.83	0.95	1380.9	0.99	1378.02	0.99
tai100a	2047.90	2375.92	0.86	2232.71	0.92	2208.85	0.93	2177.85	0.94	2206.2	0.93
tai100b	1940.61	2283.97	0.85	2147.7	0.90	2219.28	0.87	2167.97	0.90	2087.57	0.93
tai100c	1407.44	1562.3	0.90	1541.28	0.91	1515.1	0.93	1567.53	0.90	1574.59	0.89
tai100d	1581.25	2008.13	0.79	1834.6	0.86	1881.91	0.84	1923.41	0.82	1882.09	0.84
tai150a	3055,23	3644.78	0.84	3328.85	0.92	3488.02	0.88	3288.04	0.93	3202.86	0.95
tai150b	2727.99	3166.88	0.86	2933.4	0.93	3109.23	0.88	3018.9	0.90	2861.18	0.95
tai150c	2362.79	2811.48	0.84	2612.68	0.90	2666.28	0.89	2544.15	0.93	2513.92	0.94
tai150d	2655.67	3058.87	0.87	2950.61	0.90	2950.83	0.90	2899.26	0.92	2722.22	0.98
Moyenne	1982.32	2422.68	0.83	2342.99	0.88	2380.37	0.87	2335.06	0.87	2275.71	0.89

6.4.2. Accélération par GPU

Dans cette section, nous évaluons la performance de P-BLA, et la mesure la plus importante d'un algorithme parallèle est l'accélération. Cette métrique compare le temps d'exécution d'un algorithme séquentiel avec le temps d'exécution de son homologue parallèle pour résoudre un problème particulier. Par conséquent, le choix le plus prudent pour mesurer la performance d'un code parallèle est le temps d'horloge pris pour résoudre le problème concerné. Cela signifie le temps consommé entre le début et la fin de l'algorithme entier.

Nous avons mesuré le temps de calcul total de BLA sur CPU, et le temps total de calcul de P-BLA sur GPU. Ensuite, nous avons calculé l'accélération de dix temps d'exécution pour les deux algorithmes, qui est définie comme le rapport entre le temps pris par le code séquentiel et celui de l'implémentation parallèle, tel que donné par l'équation (6.2).

$$Speedup = \frac{sequential\_time\ (BLA)}{Parallel\_time(PBLA)} \qquad (6.2)$$

Pour estimer le temps d'exécution de nos algorithmes, les paramètres pour toutes les instances de benchmark sont donnés dans la Table 6.1 selon les paramètres originaux dans **[19]**. De plus, la Table 6.2 montre l'ensemble des paramètres de P-BLA en utilisant les meilleurs que nous avons trouvés. Les résultats présentés dans la Table 6.5 montrent que le P-BLA proposé est entre 3 et 9 fois plus rapide que le BLA séquentiel dans différentes instances de benchmark de DCVRP utilisé.

Table 6.5 Accélération du P-BLA pour les 21 instances de DCVRP.

Instances	Temps total de calcul sur CPU en sec	Temps total de calcul sur GPU en sec	Accélération
c50	88.71	13.26	6.69
c75	103.73	16.32	6.36
c100	112.31	18.63	6.03
c100b	118.19	18.62	6.35
c120	140.12	19.97	7.02
c150	176.09	27.17	6.48
c199	192.49	47.95	4.01
f71	106.37	16.50	6.44
f134	311.62	34.41	9.06
tai75a	104.63	27.35	3.83
tai75b	106.01	18.00	5.89
tai75c	99.20	17.70	5.61
tai75d	95.38	19.98	4.77
tai100a	104.95	21.58	4.86
tai100b	135.01	22.22	6.08
tai100c	127.44	20.18	6.31
tai100d	112.69	20.24	5.57
tai150a	146.87	40.66	3.61
tai150b	179.99	38.16	4.72
tai150c	157.89	41.39	3.81
tai150d	177.12	39.72	4.46
Totale	2896.80	540.01	117.96

De plus, les résultats présentés dans la Figure 6.2 sont plutôt satisfaisants. Il y a eu une augmentation significative de l'accélération du temps de calcul de P-BLA sur GPU par rapport au temps de calcul consommé par le BLA séquentiel sur CPU. Dans notre étude, nous avons pris en compte le temps nécessaire pour l'allocation mémoire de GPU et la copie de données entre le CPU et le GPU. Parce que cela affecte grandement la vitesse de l'algorithme.

On considère que l'exécution sur GPU peut masquer la latence de l'accès à la mémoire en exécutant plusieurs threads en parallèle, tandis que sur CPU l'exécution de BLA sera faite de manière séquentielle. De plus, il est à noter que dans notre implémentation parallèle, tous les processus de P-BLA sont exécutés sur GPU (Figure 5.1). Cela peut supprimer la fréquence de transfert de données entre l'hôte (CPU) et le GPU, ce qui est probablement le goulot d'étranglement[4] pour l'accélération par GPU.

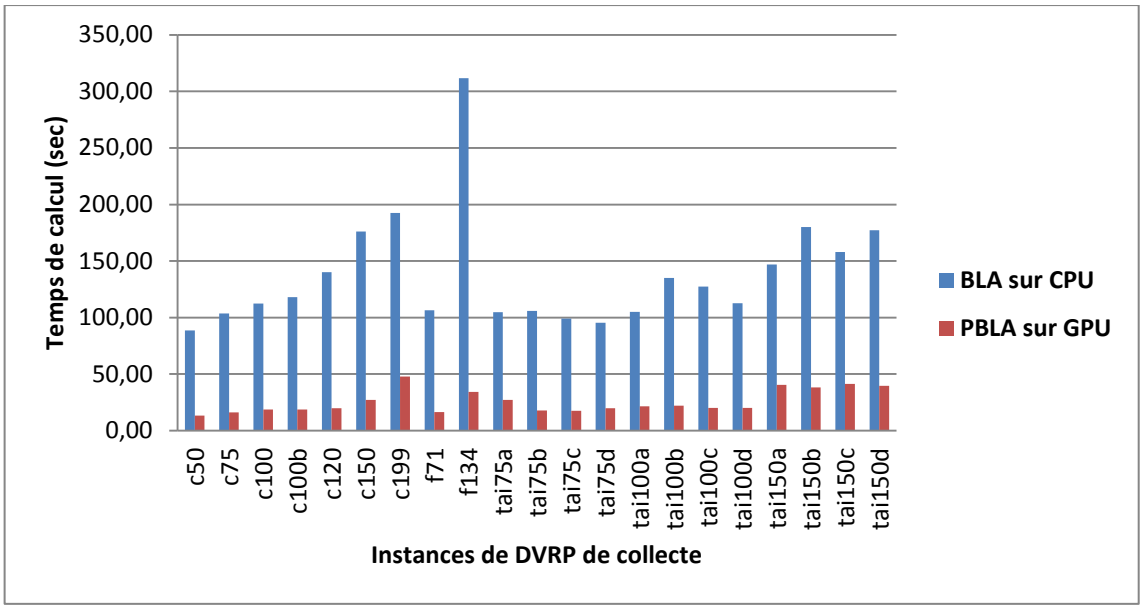

Figure 6.2 Mesures de temps de calcul de BLA et P-BLA.

6.5. Conclusion

Dans ce chapitre nous avons présenté les résultats de notre étude expérimentale sur nos algorithmes conçus, cette dernière est réalisée en utilisant les 21 instances de benchmark de Montemanni et al. **[19]** contenant de 50 à 199 clients. Les résultats obtenus confirment que notre algorithme parallèle proposé (P-BLA) est très efficace dans la plupart des cas, et qu'il est très compétitif avec d'autres métaheuristiques telles

[4] C'est un point d'un système limitant les performances globales, et pouvant avoir un effet sur les temps de traitement et de réponse.

que l'algorithme génétique (Genetic Algorithm, GA) **[63]**, la recherche Tabou (Tabu Search, TS) **[63]**, et le système de fourmis (Ant System, AS) **[19]**.

En termes de précision, notre approche parallèle fournit des résultats très compétitifs par rapport aux autres métaheuristiques qui résout le DCVRP, où elle a une précision égale à 0,89 ce qui dénote qu'elle est capable de produire de bonnes solutions sur les benchmarks dynamiques conventionnels.

En termes d'accélération, nous avons trouvé que P-BLA est 3 à 9 fois plus rapide que le BLA séquentiel dans différentes instances de benchmark de DCVRP utilisé, ce qui confirme la capacité de passage à l'échelle[5] de notre algorithme pour les grandes instances.

[5] C'est la capacité d'une solution à un problème de fonctionner même si la taille du problème augmente.

Conclusion générale et perspectives

Dans cette thèse, nous avons proposé une méthode d'optimisation bio-inspirée appelée Bees Life Algorithm (BLA), pour résoudre le problème dynamique de planification de tournées de véhicules avec une contrainte de capacité (DCVRP), intégrant des demandes de collecte. Cette proposition est appliquée pour résoudre les CVRPs statiques résultants de la partition d'une journée de travail en tranches de temps. En outre, cette approche a été divisée en deux phases distinctes dans chaque tranche de temps : la construction de groupes où les clients sont affectés aux tournées des véhicules, en utilisant la méthode k-means améliorée, et l'optimisation de routes qui représente la phase centrale dans laquelle on doit trouver les meilleures tournées dans chaque groupe qui est affecté à un véhicule.

Pour optimiser la découverte des tournées, nous avons conçu une nouvelle méthode d'optimisation combinatoire parallèle basée sur l'unité de traitement graphique (GPU). Cette méthode, nommée Parallel Bees Life Algorithm (P-BLA) permet de résoudre le problème de DCVRP. Cette proposition tire parti de la puissance des unités de calcul graphique dites GPUs et du concept de GPGPU, en particulier P-BLA a été développé en utilisant l'architecture CUDA de NVIDIA, et en se basant sur le modèle d'îlot. Afin de surmonter la complexité de calcul de la version séquentielle de cette méthode, ainsi que de réduire le temps d'exécution pour découvrir les routes de manière dynamique.

Nos algorithmes ont été testés avec les 21 instances de benchmark **[19]** contenant de 50 à 199 clients. Les résultats computationnels confirment que l'algorithme proposé est très efficace dans la plupart des cas, et qu'il est très compétitif avec d'autres métaheuristiques telles que l'algorithme génétique (Genetic Algorithm, GA) **[63]**, la recherche Tabou (Tabu Search, TS) **[63]**, et le système de fourmis (Ant System, AS) **[19]**.

En termes de précision, notre approche parallèle fournit des résultats très compétitifs par rapport aux autres métaheuristiques qui résolvent le DCVRP, où elle a une précision égale à 0,89 ce qui dénote qu'elle est capable de produire de bonnes solutions sur les benchmarks dynamiques conventionnels.

En termes d'accélération, nous avons trouvé que P-BLA est, selon le cas, 3 à 9 fois plus rapide que le BLA séquentiel pour différentes instances de benchmark de

DCVRP utilisé, ce qui confirme la capacité de passage à l'échelle de notre algorithme pour les grandes instances.

Comme perspectives à notre travail de recherche, nous envisageons de lancer d'autres travaux dans les thématiques suivantes :

L'amélioration de l'algorithme proposé (P-BLA), en implémentant et en mettant en œuvre d'autres algorithmes pour le tri parallèle tel que le tri fusion (merge sort), et le tri par échantillon (sample sort), ainsi qu'en utilisant d'autres structures de voisinage dans l'étape de recherche de nourriture, comme le voisinage d'inversion, et le voisinage d'insertion.

L'utilisation d'un nouveau mappage, dans lequel nous étendons l'implémentation multi-colonies sur un seul GPU à une implémentation multi-GPU, développant une stratégie qui distribue le calcul de la plus courte tournée pour chaque colonie, tout en profitant des accélérateurs GPU à chaque nœud. qui permet la résolution d'instances à grande échelle du DCVRP.

Nous prévoyons d'étendre l'algorithme pour résoudre d'autres problèmes liés au DVRP, comme le DVRP avec fenêtres de temps (DVRPTW**),** le DVRP avec des déplacements dépendant du temps (DVRPTT), et le DVRP avec double service de collecte et de livraison (DVRPPD). Il y'a d'autres aspects liés à l'optimisation dans les environnements dynamiques qui ont des intérêts croissants. En effet, les modèles du monde réel permettent d'ajouter une prédiction sur les événements à venir. Dans notre cas, nous pouvons prédire les demandes futures des clients en suivant des modèles probabilistes obtenus en analysant les demandes des clients (heure d'arrivée, lieu, quantité, etc.) sur une certaine période. Ainsi, notre problème convergera vers un problème dynamique de tournées de véhicules stochastique (SDVRP). Celui-ci concerne une large classe de problèmes d'optimisation combinatoire sous incertitude, où une partie de l'information sur les données problématiques est inconnue au stade de la planification, mais une connaissance sur sa distribution de probabilité est assumé. Ici, l'objectif est d'étendre et d'adapter nos algorithmes pour faire face à cette classe de problèmes.

Bibliographie

[1] J.P Breusard, D. Fromentin. Gestion pratique de la chaine logistique : une vision globale des outils de management et de progrés. les éditions démos, 2004.

[**2]** T. Vidal, T. Crainic, M. Gendreau, & C. Prins. Heuristiques unifiées pour les problèmes de tournées de véhicules multi-attributs. Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport (CIRRELT), 2011.

[3] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. Management Science, 6(1) :pp. 80-91, 1959.

[4] E. L. Lawler, J. K. Lenstra, A.H.G.R. Kan, and D.B. Shmoys. The traveling salesman problem : a guided tour of combinatorial optimization, volume 3. Wiley New York, 1985.

[5] C. Dhaenens, M.L. Espinouse, and Bernard Penz. Problèmes combinatoires classiques. In Recherche opérationnelle et réseaux : méthodes d'analyse spatiale. Hermès Science Publications, 2002.

[6] D.S. Johnson and C.H. Papadimitriou. Computational complexity. In A.H. G. Rinnooy Kan E. L Lawler, J. K. Lenstra and D. B. Shmoys (eds), editors, The Traveling Salesman Problem. John Wiley & Sons Chichester, 1985.

[7] D L Applegate, R E Bixby, V Chvátal et W J Cook : The Traveling Salesman Problem : a computational study. Princeton University Press, Princeton, NJ, 2007.

[8] T. Bektas : The multiple traveling salesman problem : an overview of formulations and solution procedures. Omega, 34(3):209-219, juin 2006.

[9] M. Sakarovitch. Optimisation combinatoire : Programmation discrète, Collection Enseignement des sciences, Hermann, 1984.

[10] N Christofides, A Mingozzi et P Toth, The vehicle routing problem. In Combinatorial Optimization, John Wiley, Vol. 11 pages: 315–338, 1979.

[11] P. Toth, D. Vigo. The vehicle routing problem. Society for Industrial and Applied Mathematics Philadelphia, Vol. 9, p. 367PA, USA ©2002.

[12] C. Rego et C. Roucairol. "Le problème de tournées de véhicules : Etude et Résolution Approchée". Technical Report, inria, Février 1994.

[13] N. Wilson, J. Sussman, H. Wong, T. Higonnet. Scheduling algorithms for a dial-a-ride system. Urban Systems Laboratory Report USL TR70-13, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1971.

[14] Psaraftis, H.. A dynamic-programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. Transportation Science, 14(2):130-154, 1980.

[15] Harilaos N. Psaraftis. Vehicle Routing: Methods and Studies, chapter Dynamic Vehicle Routing Problems, pages 223-248. Elsevier Science Publishers B.V. (North Holland), 1988.

[16] Psaraftis, H. N.. Dynamic vehicle routing: Status and prospects. Annals of Operations Research, 61(1):143-164, 1995.

[17] Gendreau, M. and Potvin, J.-Y.. Dynamic vehicle routing and dispatching. In Crainic, Teodor G. and Laporte, Gilbert, editors, Fleet management and logistics, pages 115-126. Kluwer, Boston, 1998.

[18] A. Larsen, The dynamic vehicle routing problem, Ph.D. Thesis, Institute of Mathematical Modelling, Technical University of Denmark, 2001.

[19] R. Montemanni, L. M. Gambardella, A. E. Rizzoli, and A. V. Donati. Ant colony system for a dynamic vehicle routing problem. Journal of Combinatorial Optimization, 10(4):327-343, 2005.

[20] Khouadjia, M. R. Solving Dynamic Vehicle Routing Problems: From Single-solution Based Metaheuristics to Parallel Population Based Metaheuristics Ph.D. Thesis, Lille-1 University, 2011.

[21] J. Branke. Evolutionary optimization in dynamic environments. Kluwer Academic Publishers, 2002.

[22] Harilaos N. Psaraftis. Vehicle Routing: Methods and Studies, chapter Dynamic Vehicle Routing Problems, pages 223-248. Elsevier Science Publishers B.V. (North Holland), 1988.

[23] Lund K., B. G. Madsen and J. M. Rygard. Vehicle Routing Problems with Varying Degrees of Dynamism. Technical report IMM-REP-1996-1, Institute of Mathematical Modeling, Technical University of Denmark, Lyngby, Denmark, 1996.

[24] Gianpaolo Ghiani, Francesca Guerriero, Gilbert Laporte, Roberto Musmanno. Real-time vehicle routing: Solution concepts, algorithms and parallel computing strategies.European Journal of Operational Research 151. Elsevier, 1–11, 2003.

[25] Harilaos N. Psaraftis. Dynamic vehicle routing: Status and prospects. Annals of Operations Research, pages 143-164, 1995.

[26] Victor PILLAC, Christelle GUERET, Andres MEDAGLIA. Dynamic Vehicle Routing Problems:State Of The Art and Prospects. Technical Report 10/4/AUTO.Ecole des Mines de Nantes, France July 2010.

[27] P. Kilby, P. Prosser, and P. Shaw. Dynamic VRPs: A study of scenarios. University of Strathclyde Technical Report, pages 1-11, 1998.

[28] Gendreau, M., Guertin, F., Potvin, J.-Y., and Taillard, E.. Parallel tabu search for real-time vehicle routing and dispatching. Transportation Science, 33(4):381–390, 1999.

[29] Chen, Z. and Xu, H..Dynamic column generation for dynamic vehicle routing with time windows. Transportation Science, 40(1):74–88, 2006.

[30] Hong, L..An improved lns algorithm for real-time vehicle routing problem with time windows. Computers & Operations Research, 39(2):151 – 163, doi:10.1016/j.cor.2011.03.006, 2012.

[31] Pillac, V., Guéret, C., and Medaglia, A. L.. A fast re-optimization approach for dynamic vehicle routing. p. 1-22 Technical Report 12/X/AUTO, École des Mines de Nantes, France, 2012.

[32] Haghani, A. and Jung, S.. A dynamic vehicle routing problem with time-dependent travel times. Computers & Operations Research, 32(11):2959 – 2986, 2005.

[33] Ichoua, S., Gendreau, M., and Potvin, J.-Y.. Vehicle dispatching with time-dependent travel times. European Journal of Operational Research, 144(2):379 – 396, 2003.

[34] Yang, J., Jaillet, P., and Mahmassani, H.. Realtime multivehicle truckload pickup and delivery problems. Transportation Science, 38(2):135–148, 2004.

[35] Flatberg T, Hasle G, Kloster O, Nilssen EJ, Riise A Dynamic and Stochastic Aspects in Vehicle Routing - A Literature Survey. Technical Report STF90A05413, SINTEF ICT, 2005.

[36] Pillac V, Gendreau M, Guéret C, Medaglia A. A review of dynamic vehicle routing problems. European Journal of Operational Research 225(1):1–11, 2013.

[37] Ritzinger U, Puchinger J. Hybrid Metaheuristics for Dynamic and Stochastic Vehicle Routing. In: Talbi EG (ed) Hybrid Metaheuristics, vol 434, Springer, pp 77–95, 2013.

[38] Meisel S. Anticipatory Optimization for Dynamic Decision Making, Operations Research/Computer Science Interfaces, vol 51. Springer, New York, 2011.

[39] Ritzinger, Ulrike, Jakob Puchinger, Richard F Hartl. A survey on dynamic and stochastic vehicle routing problems. International Journal of Production Research 1–17, 2015.

[40] Zhang C, Dellaert N, Zhao L, Woensel TV, Sever D. Single Vehicle Routing with Stochastic Demands: Approximate Dynamic Programming. Tech. rep., Tsinghua University, Beijing, China, 2013.

[41] Meisel S, Suppa U, Mattfeld D. Serving Multiple Urban Areas with Stochastic Customer Requests. In: Kreowski HJ, Scholz-Reiter B, Thoben KD (eds) Dynamics in Logistics, Springer Verlag, pp 59-68, 2011.

[42] Bent RW, Van Hentenryck P. Scenario-Based Planning for Partially Dynamic Vehicle Routing with Stochastic Customers. Operations Research 52:977-987, 2004.

[43] Pavone M, Bisnik N, Frazzoli E, Isler V. A stochastic and dynamic vehicle routing problem with time windows and customer impatience. Mobile Networks and Applications 14(3):350–364, 2009.

[44] Taniguchi E, Shimamoto H. Intelligent transportation system based dynamic vehicle routing and scheduling with variable travel times. Transportation Research Part C: Emerging Technologies 12(3- 4):235-250, 2004.

[45] Potvin JY, Xu Y, Benyahia I. Vehicle routing and scheduling with dynamic travel times. Computers & Operations Research 33(4):1129-1137, 2006.

[46] Schilde M, Doerner K, Hartl R. Integrating stochastic time-dependent travel speed in solution methods for the dynamic dial-a-ride problem. European Journal of Operational Research 238(1):18-30, 2014.

[47] M. R. Swihart and J. D. Papastavrou. A stochastic and dynamic model for the single vehicle pick-up and delivery problem. European Journal of Operational Research, 114: 447–464, 1999.

[48] Powell, W. B., She, Y., Nickerson, K. S., Butterbaugh, K., and Atherton, S.. Maximizing profits for North American Van Lines' truckload division: A new framework for pricing and operation. Interfaces, 18(1):21-41, 1988.

[49] L. M. Hvattum, A. Løkketangen, and G. Laporte. Solving a dynamic and stochastic vehicle routing problem with a sample scenario hedging heuristic. Transportation Science, 40: 421–438, 2006.

[50] L. Bianchi. "Notes on Dynamic Vehicle Routing- The State of the Art". Technical Report . IDSIA-05-01, 20 December 2000.

[51] Clarke, G. and Wright, J. W.. Scheduling of vehicles from a central depot to a number of delivery points. Operations Research, 12(4):568–581, doi:10.1287/opre.12.4.568, 1964.

[52] Fisher, M. L. and Jaikumar, R.. A generalized assignment heuristic for vehicle routing. Networks, 11(2):109–124, 1981.

[53] Prins, C., Labadi, N., and Reghioui, M.. Tour splitting algorithms for vehicle routing problems. International Journal of Production Research, 47(2):507–535, 2009.

[54] S. Roy , J.-M. Rousseau , G. Lapalme , and J. A. Ferland . « Routing and scheduling of transportation services for disabled : summary report ». Technical Report, Centre de recherche sur les transports, Université de Montréal, June 1984.

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

[56] Or, I.. Traveling salesman-type combinatorial optimization problems and their relation to the logistics of regional blood banking. PhD thesis, Department of Industrial Engineering and Management Science, Northwestern University, 1976.

[57] E. Taillard , P. Badeau , M. Gendreau , F. Guertin , and J.-Y. Potvin . « A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows ». Transportation Science 31, pages 170–186, 1997.

[58] Gendreau, M., A. Hertz and G. Laporte. "A New Insertion and Postoptimization Procedures for the Traveling Salesman Problem", Operations Research 40, 1086−1093, 1992.

[59] S. Mitrovic-Minic , R. Krishnamurti , and G. Laporte . « Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows ». Transportation Research vol. 38, no. 8, pages 669–685, 2004.

[60] R.M. Branchini, A.V. Armentano and A. Lkketangen. Adaptive granular local search heuristic for a dynamic vehicle routing problem. Computers & Operations Research, vol. 36, no. 11, pages 2955-2968, 2009.

[61] F. Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. Computers & Operations Research, 13:533–549, 1986.

[62] S. Mitrovic-Minic . « The Dynamic Pickup and Delivery Problem with Time Windows ». PhD thesis, Simon Fraser University, 2001.

[63] F.T. Hanshar and B.M. Ombuki-Berman. Dynamic vehicle routing using genetic algorithms. Applied Intelligence, vol. 27, pages 89-99, 2007

[64] I.H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Annals of Operations Research, vol. 41, no. 4, pages 421-451, 1993.

[65] Hansen, P. and Mladenovic, N.. Variable neighborhood search: Principles and applications. European Journal of Operational Research, 130(3):449 – 467, 2001.

[66] Y. Xu, L. Wang, Y. Yang, Dynamic vehicle routing using an improved variable neighborhood search algorithm, Journal of Applied Mathematics, 2013.

[67] De Armas J., Melián-Batista B.. Variable neighborhood search for a dynamic rich vehicle routing problem with time windows. Comput Indus Eng 85:120–131, 2015.

[68] Shaw, P. . Using constraint programming and local search methods to solve vehicle routing problems. In Principles and Practice of Constraint Programming – CP98, volume 1520 of Lecture Notes in Computer Science, pages 417–431. Springer Berlin / Heidelberg, 1998.

[69] Bent, R. and Hentenryck, P. V.. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. Computers & Operations Research, 33(4):875 – 893, 2006.

[70] Hong L.. An improved LNS algorithm for real-time vehicle routing problem with time windows. Comput Oper Res 39(2): 151–163, 2012.

[71] Pisinger, D. and Ropke, S. . A general heuristic for vehicle routing problems. Computers & Operations Research, 34(8):2403–2435, 2007.

[72] Pillac, V., Guéret, C., & Medaglia, A.. On the dynamic technician routing and scheduling problem, 2012.

[73] Benyahia, I. and Potvin, J. . Decision support for vehicle dispatching using genetic programming. IEEE Transactions on Systems Man and Cybernetics Part A - Systems and Humans, 28(3):306– 314, 1998.

[74] Cheung, B. K. S., Choy, K. L., Li, C.-L., Shi, W., and Tang, J.. Dynamic routing model and solution methods for fleet management with mobile technologies. International Journal Of Production Economics, 113(2):694–705, 2008.

[75] Haghani, A. and Jung, S.. A dynamic vehicle routing problem with time-dependent travel times. Computers & Operations Research, 32(11):2959 – 2986, 2005.

[76] Van Hemert, J. I. and Poutré, J. L.. Dynamic routing problems with fruitful regions: Models and evolutionary computation. Parallel Problem Solving from Nature, volume 3242 of Lecture Notes in Computer Science, pages 692-701. Springer Berlin / Heidelberg, 2004.

[77] H. Housroum. "Une approche génétique pour la résolution du problème VRPTW dynamique". Thèse en génie informatique et automatique à l'université d'Artois. Soutenue le 3 Mai 2005.

[78] J. Kennedy and R. Eberhart. Particle swarm optimization. In IEEE International Conference on Neural Networks, 1995., volume 4, pages 1942-1948., 1995.

[79] M. Gendreau, F. Guertin, J-Y. Potvin and E. Taillard. Parallel Tabu Search for Real-Time Vehicle Routing and Dispatching. Transportation Science, vol. 33, no. 4, pages 381-390, 1999.

[80] Taillard, E., Badeau, P., Gendreau, M., Guertin, F., and Potvin, J.. A tabu search heuristic for the vehicle routing problem with soft time windows. Transportation Science, 31(2):170–186, 1997.

[81] Y.-B. Park and S.-H. Song . « Vehicle Scheduling Problems with Time-Varying Speed ». Computers & Industrial Engineering 33, pages 853–856, 1997.

[82] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, volume 1, pages 281-297. Oakland, CA, USA., 1967.

[83] G. Maroua, N. Djedi, and S. Bitam. A parallel bee life algorithm for dcvrp on gpus. The 5th International Conference on Metaheuristics and Nature Inspired Computing, META2014, Marrakech (Morocco),pages 3-5, 2014.

[84] D. Teodorović, M. Šelmić, and T. Davidović. Bee colony optimization-part ii: The application survey. Yugoslav Journal of Operations Research, 25(2): 185-219, 2015.

[85] S. Bitam, M. Batouche, E-G. Talbi. "A survey on bee colony algorithms," 24th IEEE International Parallel and Distributed Processing Symposium, NIDISC Workshop, Atlanta, Georgia, USA, pages 1-8, 2010.

[86] K. von Frisch, "The Dance Language and Orientation of Bees", Harvard University Press, Cambridge, 1967.

[87] S. Bitam, A. Mellouk. Bee life-based multi constraints multicast routing optimization for vehicular ad hoc networks. Journal of Network and Computer Applications, vol. 36, no 3, pages 981-991, 2013.

[88] Standard for Programming Language C++.

[89] K. DeJong. An analysis of the behavior of a class of genetic adaptive systems. Ph. D. Thesis, University of Michigan, 1975.

[90] D. B. Fogel. An evolutionary approach to the traveling salesman problem. Biological Cybernetics, 60(2):139- 144, 1988.

[91] W. Banzhaf. The molecular traveling salesman. Biological Cybernetics, 64(1):7-14, 1990.

[92] E.-G. Talbi. Metaheuristics: from design to implementation, volume 74. John Wiley & Sons, 2009.

[93] Leopold, C.. Parallel and Distributed Computing: A Survey of Models, Paradigms and Approaches, Wiley-Interscience, 2001.

[94] Flynn, M. J. .Very High Speed Computing Systems. Proceedings IEEE, 1966.

[95] Sudholt, D. .Parallel evolutionary algorithms. In Springer Handbook of Computational Intelligence. Springer Berlin Heidelberg. pp. 929-959, 2015.

[96] Luong, T. V.. Métaheuristique parallèles sur GPU. Ph.D. thesis Ecole doctorale sciences Pour l'ingenieur Universite Lille Nord-de-France, 2011.

[97] CUDA C PROGRAMMING GUIDE v5.0, 2012.

[98] Munshi, A., Gaster, B., Mattson, T. G., & Ginsburg, D. (2011). OpenCL programming guide. Pearson Education.

[99] G. Maroua, N. Djedi, and S. Bitam. (2017). 'GPU-based distributed bee swarm optimization for dynamic vehicle routing problem', Int. J. Ad Hoc and Ubiquitous Computing, en cours de publication, accepté le 27/11/2017.

[100] B. Jan, B. Montrucchio, C. Ragusa, F. G. Khan, and O. Khan. Fast parallel sorting algorithms on gpus. International Journal of Distributed and Parallel Systems, 3(6):107, 2012.

[101] É. Taillard. Parallel iterative search methods for vehicle routing problems. Networks, 23(8):661-673,1993.

[102] N. Christofides and J. E. Beasley. The period routing problem. Networks, 14(2):237-256, 1984.

[103] M. Fisher. Vehicle routing. Handbooks in operations research and management science, 8:1-33, 1995.

[104] K. Weicker. Performance measures for dynamic environments. In PPSN, pages 64-76. Springer, 2002.

Annexe A : Spécification de format pour les instances de DVRP

Ce document est ecrit en anglais, et qui spécifie un format de type TSPLIB pour le problème de tournées de véhicule et d'autres problèmes de routage avec contraintes latérales. Fondamentalement, il s'agit d'un nouveau format, avec certains mots-clés de TSPLIB utilisés pour notre propre usage.

La spécification est une série de clauses. De nombreuses clauses peuvent être omises de la spécification et une valeur par défaut raisonnable sera utilisée. Les éléments de base de la spécification sont les visites, les dépôts, les emplacements, les véhicules, les capacités et les poids.

Visits: (les visites)

Une livraison ou un ramassage à un endroit particulier. Une visite peut être soit un ramassage d'un ensemble de marchandises; elles ne sont différenciées que par le signe sur la quantité.

Depots: (Dépôts)

Où commencent et se terminent tous les véhicules. Les dépôts ont aussi un emplacement.

Location: (Emplacement)

Une localisation géographique Les heures / les distances, etc. sont données d'un endroit à l'autre.

Vehicles: (Véhicules)

Les véhicules qui effectuent les visites. Une flotte homogène (tous les véhicules identiques) peut être facilement définie, ou hétérogène avec quelques autres clauses. Dans une flotte hétérogène, les identifiants de véhicule sont 1 à NUM_VEHICLES.

Capacities: (Capacités)

Les capacités sur les véhicules qui doivent être observés. Ceux-ci peuvent être la charge totale, le volume total, le nombre de palettes, etc.

Weights: (Poids)

Les "poids" sont les coûts de déplacement entre les emplacements. Cela peut être la distance entre les visites, le temps pris pour voyager, le coût en dollars ou quoi que ce soit. L'unité de poids est l'unité utilisée pour tous les autres coûts. Une façon de spécifier des poids est de donner une coordonnée pour tous les emplacements, puis d'utiliser une fonction comme la distance euclidienne. Une autre façon consiste à spécifier explicitement la matrice de distance. la matrice peut être spécifiée comme une matrice complète, une demi-matrice (pour les problèmes symétriques) ou comme une matrice d'adjacence.

Le format est une représentation relationnelle utilisant comme clé un identifiant non négatif pour chaque Dépôt, Visite et Véhicule. (Les ID de dépôt et les ID de visite peuvent se chevaucher, mais devraient probablement être différents pour la lisibilité). Il y a deux sections, une section de spécification et une section de données. Les nombres entre crochets ([]) sont des valeurs par défaut.

VRPTEST 1

Identifiant de format Obligatoire et doit être le premier caractère non vide dans le fichier. "1" est le numéro de version du format.

NAME:*{string}*

Nom du problème Le reste de la ligne du premier non-blanc est le nom.

TYPE: *{enum}*

Type de problème Non utilisé par le lecteur - pour information seulement. Valeurs possibles :

VRP (Constrained Vehicle Routing Problem)

PDP (Pickup and Delivery Problem)

...

COMMENT: *{string}*

La description. Ignorer par programme. Peut être répété.

NUM_VISITS: *{int}*

Nombre de points de ramassage et de livraison. Champs obligatoires. N'inclut pas les dépôts.

NUM_DEPOTS: *{int}* [1]

Nombre de depots.

NUM_VEHICLES: *{int}*

Nombre maximum de véhicules disponibles. Champs obligatoires.

NUM_CAPACITIES: *{int}* [1]

Le nombre de contraintes de capacité. La valeur par défaut est 1 (charge).

NUM_LOCATIONS: *{int}* [NUM_VISITS + NUM_DEPOTS]

Le nombre d'emplacements distincts pour les dépôts et les visites. La valeur par défaut est une pour chaque visite plus une pour chaque dépôt.

CAPACITIES: *{real}* ... [0 ...]

Exactement NUM_CAPACITIES valeurs donnant des capacités pour chaque dimension.

Implique une flotte homogène. (Utilisez VEHICLE_CAPACITIES_SECTION pour une flotte hétérogène). Doit suivre NUM_CAPACITIES spec.

SPEED: *{real}* [1.0]

Une vitesse utilisée pour convertir les poids de bord en temps. Sera surchargé par les vitesses du véhicule si elles sont données.

MAX_TIME: *{real} [inf]*

Durée maximale (dans le temps) d'une tournée unique.

EDGE_WEIGHT_TYPE: *{enum}*

Comment le poids des arêtes est spécifié. Valeurs valides:

EUC_2D

Euclidien [par défaut]

MAN_2D

Manhattan métrique

MAX_2D

Métrique maximale

EXPLICIT
Matrice plus tard dans les données
EDGE_WEIGHT_FORMAT: *{enum}*
Pour EDGE_WEIGHT_TYPE EXPLICIT, spécifie la forme de la matrice:
FULL_MATRIX
Matrice complète [par défaut]
LOWER_TRIANG
Triangle inférieur de la matrice symétrique, diagonale non incluse.
ADJ
Liste d'adjacence
...
OBJECTIVE: *{enum}*
La fonction objectif à minimiser, avec des paramètres optionnels. Valeurs possibles:
VEH-WEIGHT
Minimiser le nombre de véhicules, puis la somme des poids des arêtes
WEIGHT
Minimiser la somme des poids des arêtes dans un nombre donné de véhicules [Par défaut]
MIN-MAX-LEN
Minimisez la longueur maximale de l'itinéraire. Équivalent à make-span dans un problème de job-shop
...
DATA_SECTION
Sépare la spécification de la section de données. Champs obligatoires. Aucun des noms de sections ci-dessus ne peut apparaître sous ce marqueur. Toutes les sections ci-dessus qui n'ont pas été spécifiées seront définies à leurs valeurs par défaut.
DEPOTS
{depot-id} ...
Identifiants de NUM_DEPOTS dépôts. Champs obligatoires. Cette section définit les identifiants de tous les dépôts.
DEMAND_SECTION
{visit-id} {quantity-1} [{quantity-2}...]
Demande à chaque visite, dans chacune des dimensions. Obligatoire, et doit être la première clause à mentionner les ID de visite. Cette section, en plus de spécifier le dmand, spécifie également les identifiants de toutes les visites. Il doit y avoir une ligne pour chaque visite. Une quantité négative implique une livraison, un positif est un ramassage.
visit-id:
(int)
quantity-n:
(real)

LOCATION_COORD_SECTION

{location-id} {x-coord} {y-coord}

Coordonnées des emplacements. Doit être une ligne pour chacun des emplacements NUM_LOCATION. Cela doit être la première section référençant les emplacements. L'identifiant de localisation provient de cette section. Si elle n'est pas spécifiée, tous les emplacements sont à (0,0) et l'identificateur de chaque emplacement est par défaut l'identifiant de la visite ou du dépôt correspondant.location-id:
(int)
x-coord:
(real)
y-coord:
(real)

VISIT_LOCATION_SECTION

{visit-id} {location_id}

Id de lieu pour la visite. Doit avoir une ligne pour chaque visite. Si elle n'est pas spécifiée et que EDGE_WEIGHT_TYPE est EXPLICIT, location-id aura la valeur par défaut visit-id.
visit-id:
(int)
location-id:
(int)

DEPOT_LOCATION_SECTION

{depot-id} {location-id}

Identique à VISIT_LOCATION_SECTION, mais pour les dépôts. Si elle n'est pas spécifiée et que EDGE_WEIGHT_TYPE est EXPLICIT, location-id sera défini par défaut sur depot-id.
depot-id:
(int)
location-id:
(int)

VEHICLE_CAPACITY_SECTION

{vehicle-id} {capacity-1} [{capacity-2}...]

Liste explicite des véhicules disponibles, avec une capacité pour chaque dimension de capacité. Doit être NUM_VEHICLES lignes.
vehicle-id:
(int)
capacity-n:
(real)

VEHICLE_SPEED_SECTION

{vehicle-id} {speed}

Vitesses pour chaque véhicule. Doit être NUM_VEHICLES lignes.
ID du véhicule:
vehicle-id:

(int)
speed:
(real)

VEHICLE_COST_SECTION

{vehicle-id} {use-cost} {distance-cost} {time-cost}

Définit la fonction objectif. Contributions du véhicule du tout (coût d'utilisation); un coût par unité de poids de bord traversé; coût par unité de temps. Si la clause n'est pas présente, le coût d'utilisation 0.0, le coût de la distance 1.0 et le coût de temps 0.0 sont utilisés

vehicle-id:
(int)
use-cost:
(real)
distance-cost:
(real)
time-cost:
(real)

EDGE_WEIGHT_SECTION

Poids de l'arête est obligatoire si EDGE_WEIGHT_TYPE == EXPLICIT.
Format dicté par EDGE_WEIGHT_FORMAT.
Pour FULL_MATRIX et LOWER_TRIANG {poids} ...
Pour ADJ {from-loc-index} {to-loc-index} {poids} ... -1f
rom-loc-index:
(int) basé sur 0 à partir de l'emplacement
to-loc-index:
(int) basé sur 0 de l'emplacement
weight:
(real)

TIME_WINDOW_SECTION

{visit-id} {early-1} {late-1} ... -1

Données de fenêtre de temps disjointes pour toutes les visites. Spécifie l'heure à laquelle le service doit commencer. Il doit y avoir une ligne pour chaque visite, suivie d'une liste de fois pour chaque visite. Si non spécifié, aucune fenêtre de temps n'est placée sur les visites.

visit-id:
(int)
early-n:
(real) Temps en format local (par exemple minutes).
late-n:
(real) Temps en format local (par exemple minutes).

DEPOT_TIME_WINDOW_SECTION

{depot-id} {early} {late}

Fenêtre de temps pour le dépôt, en spécifiant l'heure de départ la plus proche et la dernière heure de retour. Si cela n'est pas spécifié, aucune heure plus tôt ou plus tard n'est utilisée.

depot-id:

(int)

early:

(real) Temps en format local (par exemple minutes).

late:

(real) Temps in local format (e.g. minutes).

VEH_TIME_WINDOW_SECTION

{vehicle-id} {early-1} {late-1} ... -1

Horaires disponibles pour tous les véhicules.

vehicle-id:

int

early-n:

(real) Temps en format local (par exemple minutes).

late-n:

(real) Temps en format local (par exemple minutes).

DURATION_SECTION

{visit-id} {time}

Durée de la visite dans les unités de temps de localisation. Si non spécifié, 0 est utilisé.

id:

(int)

time:

(real) Temps en format local (par exemple minutes).

TIME_AVAIL_SECTION

{visit-id} {time}

Le temps (en unités de temps locales) une visite devient disponible, ou est connu dans le système - pour la simulation du routage dynamique. Si non spécifié, 0 est utilisé.id:

(int)

time:

(real) Temps en format local (par exemple minutes).

DURATION_BY_VEH_SECTION

{visit-id}
{veh-1-id}{duration-1}
{veh-2-id}{duration-2}
...
...

Permet la spécification de la durée par véhicule. Doit répertorier chaque ID de visite et dans chaque visite, chaque ID de véhicule doit apparaître.

visit-id:
(int)
veh-n-id:
(int)
duration-n:
(real)

VISIT_COMPAT_SECTION

{visit1-id}{visit2-id}{sense}
...
-1

Indique la compatibilité de 2 visites à être desservies par le même véhicule. Utilisez l'ID de visite du dépôt pour indiquer qu'une visite doit être desservie par un dépôt particulier. La liste est terminée par -1.

visit1-id:
(int)
visit2-id:
(int)
sense:
-1 doit être le même véhicule), 1 (doit être un véhicule différent)

DEPOT_COMPAT_SECTION

{visit-id} {depot-id} {sense} ... -1

Indique la compatibilité d'une visite avec un dépôt particulier. La liste est terminée par -1.

visit-id:
(int)
depot-id:
(int)
sense:
-1 (doit être servi à partir de ce dépôt), 1 (ne doit pas être servi à partir de ce dépôt)

VEH_COMPAT_SECTION

{visit-id}{veh-id}{sense}
...
-1

Indicates compatability of a visit to be served by a vehicle. List terminated by -1.

visit-id:
(int)
veh-id:
(int)
sense:
-1 (vehicle must serve visit), 1 (vehicle must not serve visit)

VEH_DEPOT_SECTION

{veh-id}{depot-id}{sense}
...
-1

Indique la compatibilité d'une visite à être desservie par un véhicule. Liste terminée par -1.
veh-id:
(int)
depot-id:
(int)
sense:
-1 (véhicule doit rendre visite), 1 (véhicule ne doit pas servir de visite)

ORDER_SECTION

{visit1-id}{relation}{visit2-id}{margin}
...
-1

Indique que visit1 doit être chronologiquement avant la visite2 dans la solution. En outre, la section de relation indique si la différence est supérieure, inférieure ou égale à la marge donnée.
visit1-id:
(int)
relation:
(enum) Un de GT, GE, LT, LE, EQ.
visit2-id:
(int)
margin:
(real) Marge au format local

OPTIONAL_VISIT_SECTION

{visit-id}{cost}
...
-1

Liste des visites qui sont facultatives, et le coût de ne pas faire le travail. Le coût est ajouté à l'objectif si la visite n'est pas visitée, et doit donc être dans des unités compatibles avec les poids des arêtes.
visit-id:
(int)
cost:
(real)

VISIT_AVAIL_SECTION

{visit-id} {time}

Le temps à laquelle chaque visite est connue du système. Pour une utilisation dans une simulation dynamique. Suivi par une ligne pour chaque visite.
visit-id:
(int)
time:
(real) Temps au format local

EOF Fin des données Optionnel.

Annexe B: Meilleures solutions trouvées

Dans cette annexe, nous présentons les meilleures routes construites par le P-BLA, avec les véhicules disponibles. Ces résultats sont décrits dans le format suivant: {nom de l'instance}: {meilleur coût}: \\ {numéro de route} {circuit du véhicule}.

Benchmark C75: best cost =858.23
Route 1: 0-5-47-36-69-71-60-70- 20- 37-0
Route 2: 0-3-44-49-24 -18 -25 -55 -50 -32-0
Route 3: 0 -67 -46 -27 -52 -45 -29 -15 -57 -13 -54 -34 -0
Route 4: 0 -73 -1 -43 -41 -22 -64 -42 -23 -56 -63 -0
Route 5: 0 -11 -66 -65 -38 -0
Route 6: 0 -26 -17 -51 -16 -33 -6 -68 -75 -0
Route 7: 0 -4 -30 -48 -21 -61 -28 -62 -74 -2 -0
Route 8:0 -12 -72 -58 -10 -31 -39 -9 -40 -0
Route 9:0 -8 -19 -59 -14 -53 -35 -7 -0

Benchmark C120: best cost =1229.63
Route 1: 0 -19 -22 -24 -25 -27 -34 -36 -29 -32 -35 -31 -30 -33 -28 -23 -21 -26 -20 -17 -16 -0
Route 2: 0 -39 -41 -44 -42 -38 -37 -46 -49 -51 -48 -47 -50 -45 -43 -40 -0
Route 3: 0 -52 -59 -65 -62 -61 -64 -66 -63 -58 -56 -60 -55 -54 -57 -53 -0
Route 4: 0 -67 -69 -70 -72 -75 -74 -76 -71 -77 -78 -79 -80 -68 -73 -0
Route 5: 0 -120 -87 -86 -111 -112 -117 -83 -113 -81 -119 -82 -88 -89 -84 -85 -92 -98 -0
Route 6: 0 -106 -101 -102 -105 -103 -107 -104 -99 -100 -96 -93 -18 -114 -118 -108 -109 -115 -116 -110 -97 -94 -91 -90 -95 -0
Route 7: 0 -2 -3 -1 -15 -11 -10 -6 -4 -5 -7 -9 -14 -13 -12 -8 -0

Benchmark C199: best cost =1631.92
Route 1: 0 -59 -99 -104 -173 -61 -85 -93 -98 -193 -91 -100 -37 -92 -151 -0
Route 2: 0 -197 -56 -186 -75 -72 -74 -133 -22 -171 -41 -145 -73 -21 -198 -0
Route 3: 0 -48 -124 -168 -47 -123 -19 -107 -143 -36 -49 -64 -175 -0
Route 4: 0 -196 -184 -29 -134 -24 -163 -68 -177 -169 -121 -79 -129 -3 -158 -150 -80 -116 -77 -0
Route 5: 0 -152 -58 -137 -117 -87 -13 -95 -97 -94 -96 -183 6 -112- 0
Route 6: 0 -110 -67 -23 -39 -187 -139 -55 -4 -155 -170 -25 -165 -130 -54 -0
Route 7: 0 -7 -182 -148 -62 -159 -11 -108 -10 -189 -63 -126 -31 -88 -0
Route 8: 0 -51 -120 -81 -103 -161 -71 -136 -65 -35 -34 -78 -164 -135 -9 -0
Route 9: 0 -113 -16 -141 -86 -38 -140 -119 -14 -192 -44 -191 -0
Route 10: 0 -105 -180 -149 -179 -195 -109 -12 -26 -28 -138 -154 -40 -53 -0
Route 11: 0 -122 -188 -20 -66 -32 -90 -181 -128 -131 -160 -30 -70 -0
Route 12: 0 -2 -144 -57 -15 -43 -142 -42 -172 -115 -178 -0
Route 13: 0 -167 -147 -166 -89 -146 -27 -127 -190 -153 -18 -106 -194 -52 -156 -0
Route 14: 0 -83 -114 -82 -46 -8 -174 -45 -125 -199 -5 -17 -84 -60 -118 -0
Route 15: 0 -69 -111 -50 -102 -157 -76 -185 -33 -101 -162 -176 -1 -132 -0

Benchmark f134: best cost =14948.9
Route 1: 0 -27 -28 -15 -88 -14 -16 -13 -87 -89 -90 -12 -10 -2 -5 -6 -7 -11 -8 -9 -4 -92 -

30 -0
Route 2: 0 -118 -46 -78 -64 -47 -134 -48 -61 52 -62 -49 -60 -24 -91 -21 -26 -59 -34 -32 -72 -31 -25 -23 -22 -1 -79 -63 -133 -67 -66 -71 -18 -17 -77 -76 -75 -73 -74 -0
Route 3: 0 -117 -116 -112 -124 -121 -129 -128 -127 -113 -123 -126 -111 -125 -122 -110 -81 -0
Route 4: 0 -19 -119 -130 -65 -84 -83 -85 -86 -20 -82 -0
Route 5: 0 -115 -114 -106 -108 -107 -109 -120 -131 -0
Route 6: 0 -68 -70 -69 -132 -80 -33 -0
Route 7: 0 -57 -58 -105 -56 -103 -99 -100 -41 -3 -40 -36 -97 -50 -51 -54 -53 -104 -35 -96 -55 -101 -98 -37 -94 -93 -45 -38 -39 -44 -42 -43 -95 -102 -29 -0

Benchmark tai75a: best cost =1646.48
Route 1: 0 -24 -18 -27 -21 -19 -25 -15 -23 -0
Route 2: 0 -4 -6 -11 -2 -3 -10 -9 -7 -1 -8 -5 -0
Route 3: 0 -66 -51 -67 -65 -26 -53 -0
Route 4: 0 -56 -58 -63 -39 -50 -60 -54 -69 -0
Route 5: 0 -36 -44 -35 -43 -47 -38 -55 -46 -64 -57 -0
Route 6: 0 -74 -75 -0
Route 7: 0 -20 -13 -12 -16 -17 -14 -22 -28 -0
Route 8: 0 -71 -73 -72 -70 -52 -62 -0
Route 9: 0 -33 -59 -31 -68 -29 -42 -45 -30 -61 -0
Route 10: 0 -40 -41 -34 -32 -37 -48 -49 -0

Benchmark tai75b: best cost =1407.02
Route 1: 0 -55 -56 -61 -58 -57 -60 -62 -63 -59 -0
Route 2: 0 -13 -14 -40 -71 -15 -9 -4 -2 -0
Route 3: 0 -19 -23 -16 -31 -17 -35 -28 -24 -34 -26 -0
Route 4: 0 -30 -21 -27 -18 -25 -29 -22 -33 -32 -0
Route 5: 0 -52 -39 -48 -70 -51 -38 -47 -45 -46 -41 -3 -0
Route 6: 0 -64 -67 -66 -65 -20 -0
Route 7: 0 -7 -68 -73 -69 -36 -6 -10 -12 -5 -72 -75 -74 -11 -1 -8 -0
Route 8: 0 -37 -49 -54 -42 -44 -43 -53 -50 -0

Benchmark tai75d: best cost =1378.02
Route 1: 0 -13 -14 -19 -20 -24 -12 -17 -70 -75 -74 -73 -72 -71 -15 -18 -22 -23 -21 -16 -0
Route 2: 0 -28 -31 -39 -29 -25 -27 -0
Route 3: 0 -38 -33 -36 -26 -32 -35 -30 -34 -37 -0
Route 4: 0 -55 -46 -53 -52 -49 -44 -56 -0
Route 5: 0 -41 -45 -51 -59 -43 -42 -54 -0
Route 6: 0 -66 -68 -67 -61 -62 -63 -65 -69 -64 -60 -0
Route 7: 0 -2 -5 -6 -7 -11 -0
Route 8: 0 -58 -57 -48 -50 -47 -40 -0
Route 9: 0 -3 -8 -1 -9 -10 -4 -0

Benchmark tai100b: best cost =2087.57
Route 1: 0 -66 -67 -51 -88 -85 -87 -83 -82 -86 -84 -0
Route 2: 0 -12 -24 -18 -27 -21 -19 -25 -23 -15 -0
Route 3: 0 -43 -35 -49 -37 -40 -29 -31 -68 -47 -38 -48 -32 -34 -41 -42 -59 -45 -30 -61 -0
Route 4: 0 -54 -63 -64 -44 -55 -57 -46 -36 -33 -50 -60 -69 -0
Route 5: 0 -74 -75 -76 -0

Route 6: 0 -98 -100 -91 -93 -0
Route 7: 0 -1 -7 -9 -10 -6 -11 -2 -3 -4 -5 -8 -0
Route 8: 0 -97 -92 -89 -95 -99 -96 -90 94 0
Route 9: 0 -20 -13 -16 -17 -14 -22 -28 -26 -65 -0
Route 10: 0 -73 -72 -70 -78 -79 -80 -77 -81 -0
Route 11: 0 -53 -62 -52 -56 -58 -39 -71 -0

Benchmark tai150a: best cost =3202.86
Route 1: 0 -1 -14 -12 -13 -9 -4 -11 -15 -2 -7 -3 -17 -6 -18 -10 -5 -16 -8 -0
Route 2:0 -21 -36 -27 -20 -34 -23 -22 -19 -0
Route 3:0 -62 -53 -64 -54 -55 -60 -57 -61 -56 -65 -63 -137 -52 -59 -58 -0
Route 4:0 -40 -47 -41 -39 -45 -44 -48 -50 -46 -43 -49 -51 -42 -0
Route 5:0 -70 -68 -73 -71 -66 -72 -75 -67 -0
Route 6:0 -143 -90 -93 -85 -79 -80 -91 -88 -78 -81 -84 -89 -87 -92 -82 -83 -77 -86 -76 -0
Route 7:0 -97 -96 -94 -95 -0
Route 8:0 -106 -113 -100 -109 -102 -104 -99 -101 -110 -107 -108 -0
Route 9:0 -122 -114 -123 -117 -129 -74 -69 -127 -0
Route 10:0 -133 -136 -135 -119 -134 -132 -131 -130 -0
Route 11: 0 -37 -25 -35 -26 -38 -24 -30 -33 -31 -32 -29 -28 -0
Route 12: 0 -105 -98 -111 -112 -103 -0
Route 13: 0 -146 -140 -145 -139 -144 -147 -149 -142 -141 -148 -138 -150 -0
Route 13: 0 -116 -126 -121 -115 -118 -125 -120 -128 -124 -0

Benchmark tai150b: best cost =2861.18
Route 1: 0 -53 -56 -52 -62 -58 -54 -69 -67 -65 -66 -51 -0
Route 2: 0 -14 -144 -28 -142 -26 -148 -150 -149 -147 -143 -140 -146 -141 -145 -0
Route 3: 0 -76 -74 -110 -119 -111 -113 -117 -109 -118 -115 -112 -116 -114 -0
Route 4: 0 -106 -98 -90 -94 -89 -95 -99 -96 -102 -0
Route 5: 0 -101 -91 -97 -108 -104 -107 -103 -92 -105 -0
Route 6: 0 -1 -9 -10 -7 -3 -2 -11 -6 -4 -5 -8 -0
Route 7: 0 -22 -13 -127 -131 -120 -136 -123 -135 -121 -137 -133 -128 -129 -125 -126 -16 -17 -0
Route 8: 0 -47 -33 -31 -30 -36 -35 -44 -45 -29 -42 -32 -34 -40 -41 -37 -49 -48 -43 -38 -46 -0
Route 9: 0 -71 -73 -72 -70 -78 -80 -79 -77 -81 -0
Route 10: 0 -20 -21 -138 -132 -139 -18 -19 -122 -24 -134 -12 -124 -130 -27 -25 -15 -23 -0
Route 11: 0 -100 -93 -75 -0
Route 12: 0 -84 -83 -88 -85 -87 -82 -86 -0
Route 13: 0 -61 -68 -59 -55 -57 -39 -64 -60 -50 -63 -0

Benchmark tai150c: best cost =2513.92
Route 1: 0 -98 -88 -85 -90 -53 -43 -96 -86 -94 -0
Route 2: 0 -7 -13 -6 -1 -10 -71 -12 -74 -73 -68 -70 -11 -8 -107 -0
Route 3: 0 -19 -16 -23 -26 -34 -24 -28 -35 -31 -17 -0
Route 4: 0 -33 -21 -27 -18 -25 -29 -20 -22 -30 -32 -0
Route 5: 0 -63 -59 -57 -60 -62 -61 -58 -56 -55 -0
Route 6: 0 -144 -150 -149 -145 -146 -148 -147 -64 -67 -66 -65 -0
Route 7: 0 -110 -99 -111 -115 -114 -112 -113 -104 -0

Route 8: 0 -41 -46 -38 -39 -40 -51 -54 -49 -37 -44 -42 -47 -45 -52 -0
Route 9: 0 -89 -48 -50 -93 -87 -97 -95 -92 -91 -36 -0
Route 10: 0 -14 -3 -100 -109 -106 -15 -9 -2 -4 -0
Route 11: 0 -116 -119 -122 -121 -124 -118 -120 -123 -125 -117 -0
Route 12: 0 -78 -82 -81 -84 -79 -80 -83 -0
Route 13: 0 -130 -137 -140 -139 -138 -141 -142 -136 -132 -134 -128 -135 -131 -133 -127 -129 -126 -143 -0
Route 14: 0 -5 -103 -108 -101 -102 -105 -76 -77 -75 -69 -72 -0

Benchmark tai150d: best cost =2722.22
Route 1: 0 -48 -50 -40 -33 -37 -43 -35 -46 -34 -42 -39 -44 -36 -0
Route 2: 0 -22 -14 -21 -16 -19 -15 -13 -23 -25 -20 -18 -17 -28 -12 -27 -24 -0
Route 3: 0 -31 -32 -30 -29 -101 -100 -102 -104 -108 -106 -105 -103 -107 -0
Route 4: 0 -59 -60 -63 -61 -64 -53 -62 -52 -3 -8 -9 -6 -7 -57 -51 -4 -55 -2 -58 -56 -65 -54 -1 -0
Route 5: 0 -79 -73 -78 -76 -72 -81 -74 -75 -11 -5 -10 -0
Route 6: 0 -136 -137 -134 -135 -0
Route 7: 0 -98 -97 -99 -94 -96 -95 -0
Route 8: 0 -91 -92 -89 -93 -119 -117 -112 -120 -121 -110 -109 -118 -115 -111 -116 -113 -26 -0
Route 9: 0 -85 -83 -131 -128 -124 -130 -133 -84 -123 -125 -132 -127 -86 -82 -0
Route 10: 0 -45 -38 -49 -126 -129 -122 -41 -47 -0
Route 11: 0 -68 -69 -70 -71 -67 -66 -90 -87 -88 -114 -0
Route 12: 0 -77 -80 -148 -149 -0
Route 13: 0 -140 -144 -142 -147 -138 -139 -143 -145 -146 -141 -150 -0