université de BORDEAUX

UNITÉ DE FORMATION Mathématiques & interactions

PROJET PROFESSIONNEL

Le problème du Batch-Picking

Equipe :
GRANDA SEBASTIAN

Enseignants :
CLAUTIAUX FRANÇOIS
SCHLICK CHRISTOPHE

M2 ROAD
Année scolaire 2023 - 2024

1 Résumé

Dans le contexte des opérations d'entrepôt, le problème du Batch-Picking consiste à regrouper des commandes et à déterminer la séquence de positions de stockage pour ramasser tous les articles d'un batch. Les articles sont partitionnés en un ensemble de commandes, de telle manière que tous les articles d'une commande doivent être ramassés dans le même trajet, par le même préparateur. L'objectif est de minimiser la distance totale parcourue par les préparateurs pour satisfaire toutes les commandes dans l'entrepôt. Ce projet implémente deux principales approches d'optimisation: l'approche séquentielle et l'approche jointe.

2 Introduction

Le problème du Batch-Picking, connu dans la littérature comme « Joint Order Batching and Picker Routing Problem » (JOBPRP), est largement appliqué dans le contexte de la logistique d'entrepôt. Comme son nom l'indique, ce problème combine deux décisions: le regroupement de commandes et le routage de préparateur. Le problème de regroupement de commandes consiste à regrouper des commandes pour ramasser leurs articles ensemble si cela conduit à une réduction de la distance totale parcourue. D'autre part, le problème de routage de préparateur consiste à déterminer la séquence de positions de stockage pour ramasser toutes les commandes dans un batch. Le Batch-Picking a été étudié sous les approches jointe et séquentielle.

Un problème de routage de véhicules groupés (Clustered Vehicle Routing Problem, CluVRP) a été proposé par [1] pour l'approche jointe. Dans ce problème, les clients sont regroupés en clusters, qui correspondent à la condition d'intégrité de la commande pour ramasser tous les articles d'une commande dans le même trajet. Leur modèle permet aux véhicules d'entrer et de sortir des clusters plusieurs fois en utilisant des contraintes de cluster doux. Notre cas peut être adapté à ce modèle en considérant les clients comme les articles à ramasser. La formulation CluVRP pour notre contexte est décrite dans la section Annexes.

Le Batch-Picking a également été abordé séquentiellement, où le regroupement de commandes est résolu en premier, puis les trajets sont obtenus. L'avantage de cette approche est la réduction évidente de la complexité, car le routage peut être résolu pour chaque batch indépendamment. L'inconvénient est le manque de coordination entre les deux problèmes, ce qui peut conduire à des solutions sous-optimales car les décisions de regroupement sont prises sans tenir compte du problème de routage. Des idées relatives peuvent être trouvées dans [10], où les auteurs discutent des approches « batch-first route-second » par rapport à l'approche jointe, et des avantages de résoudre les problèmes simultanément. Cependant, ces approches nécessitent généralement un calcul coûteux de la métrique de distance, tel que le plus court chemin entre chaque combinaison de commandes (c'est-à-dire la meilleure séquence pour ramasser tous les articles dans un batch). Par conséquent, nous considérons un défi de recherche pertinent de trouver une métrique qui approxime au mieux la séquence la plus courte des commandes à ramasser dans un batch, sans avoir besoin d'une recherche exhaustive.

Dans ce travail, nous étudions le Batch-Picking en tant que variante du problème de ramassage et de livraison (PDP), et nous proposons une heuristique de type « batch-first route-second » pour résoudre de grandes instances. L'heuristique proposée est basée sur la distance de Hausdorff, qui mesure la proximité de deux ensembles d'articles (c'est-à-dire deux commandes), et est utilisée pour déterminer la meilleure façon de regrouper les commandes en batches. La solution initiale est obtenue en résolvant le problème de p-médian, et la séquence d'articles à ramasser

dans un batch est déterminée en résolvant un ensemble de problèmes indépendants de voyageur de commerce (TSP). Une fois la solution initiale obtenue, un algorithme de recherche locale est appliqué pour améliorer la solution en appliquant un ensemble d'opérateurs de déplacement et en ré-optimisant les trajets.

Le reste de ce document est organisé comme suit. Dans la section suivante, nous décrivons le problème. Ensuite, nous présentons l'heuristique proposée, les détails de l'implémentation et les expériences numériques. Enfin, nous fournissons une discussion des résultats et les conclusions de ce travail.

3 Description du problème

Cette section vise à formaliser le Batch-Picking en tant que variante du problème de ramassage et de livraison (Pickup and Delivery Problem, PDP). Plus précisément, nous le modélisons comme le problème de routage de véhicules multi-dépôts avec ramassage et livraison mixtes (Multi-Depot Vehicle Routing Problem with Mixed Pick-up and Delivery, MDVRPMPD) qui, selon [3], appartient à la catégorie des problèmes de ramassage et de livraison multi-véhicules un-à-plusieurs-à-un avec des demandes simples et des solutions mixtes [1-M-1|P/D|m].

Dans ce problème, nous cherchons à minimiser la distance totale parcourue par les préparateurs pour satisfaire toutes les commandes dans l'entrepôt. Donc, nous modélisons les articles au lieu des positions de telle manière que les articles appartenant à différentes commandes peuvent partager la même position. Nous disposons d'un ensemble d'articles I regroupés par des commandes disjointes O, où $i(o)$ fait référence aux articles de la commande $o \in O$.

Soit $G = (V, A)$ le graphe orienté représentant la disposition de l'entrepôt, où V est l'ensemble des nœuds et A est l'ensemble des arcs. L'ensemble des nœuds V est composé des nœuds d'articles I, des nœuds artificiels J et des nœuds de dépôt D. Ces ensembles sont disjoints et satisfont $V = I \cup J \cup D$. Les nœuds de dépôt $D = \{0, n\}$, où 0 est l'origine et $n = |I|+1$ est la destination de chaque trajet. Les nœuds artificiels $J = \{|I|+2, \ldots, |I|+|O|+1\}$ sont les points de consolidation pour chaque commande pour assurer la condition d'intégrité de la commande. Chaque nœud $i \in V$ est associé à une demande et un volume, notés $p_i \geq 0$ et $v_i \geq 0$, respectivement. Cependant, seuls les nœuds artificiels ont une demande et un volume positifs, qui sont égaux à 1 et au volume total de la commande, respectivement. La demande et le volume des nœuds d'articles et de dépôt sont 0 $p_i = v_i = 0$ pour $i \in I \cup D$. De plus, la position physique dans laquelle se trouve chaque nœud est représentée par $pos(i)$, $\forall i \in V$. Il est possible que plusieurs nœuds partagent la même position.

L'ensemble des arcs $A = \{(i, j) \in V \times V : i \neq j\}$ sont tous les trajets réalisables entre les nœuds, chacun avec une distance $d_{ij} \geq 0$ obtenue à partir des positions des nœuds $pos(i)$ et $pos(j)$ dans l'entrepôt. La distance vers ou depuis les nœuds artificiels est 0 $d_{ij} = d_{ji} = 0$, $\forall i \in V, j \in J$.

Les nœuds sont visités par un ensemble de préparateurs identiques $k \in K$, chacun avec une capacité unitaire et volumétrique. La contrainte de capacité unitaire fait référence au nombre maximal de commandes qu'un préparateur de commandes peut transporter dans un trajet, tandis que la contrainte de capacité volumétrique fait référence au volume maximal. Les bornes supérieures sont notées C_{unit} et C_{volume}, respectivement. Nous supposons qu'il y a suffisamment de préparateurs pour couvrir toutes les commandes dans l'entrepôt, ce qui est donné par $\underline{B} > 0$, le nombre minimal de batches nécessaires pour avoir toutes les commandes attribuées à un batch avec un $\alpha\%$ de marge 1.

$$\underline{B} = \max\left(\frac{\sum_{i\in I} v_i}{C_{volume}}, \frac{|O|}{C_{unit}}\right) \times (1+\alpha) \tag{1}$$

Soit $d(i) \in J$ le nœud artificiel, ou point de livraison, du nœud d'article i. Les nœuds artificiels agissent comme des points de consolidation pour chaque commande, en garantissant que tous les articles d'une commande sont ramassés dans le même trajet. Cela est réalisé en forçant les préparateurs à visiter les nœuds d'articles et à les livrer aux nœuds artificiels.

Les hypothèses sont résumées comme suit:

- Nous pouvons mélanger les positions de stockage de différentes commandes dans le même batch. En d'autres termes, les articles de différentes commandes peuvent être ramassés dans n'importe quel ordre.
- Tous les ramassages doivent être effectués pendant le même batch. Donc, plusieurs préparateurs pour la même commande ne sont pas autorisés.
- La quantité de préparateurs est suffisante pour couvrir tous les batches formés. C'est-à-dire qu'au moins les $\underline{B}$ véhicules sont disponibles. Il n'est pas intéressant de minimiser le nombre total de batches, mais la distance totale de picking à la place.
- Les préparateurs ont les mêmes caractéristiques (flotte homogène). Tous leurs trajets commencent et terminent le picking aux dépôts.
- Toutes les contraintes de mobilité sont prises en compte dans la matrice de distance. Ainsi, la distance entre deux positions est le plus court chemin entre elles, en tenant compte de la disposition de l'entrepôt et des contraintes de mobilité des véhicules.

Des formulations de flux de marchandises à trois indices pour le MDVRPMPD peuvent être trouvées dans [6] et [2]. Ces formulations impliquent deux types de variables: sélection et flux. Soit $x_{ijk} \in \{0,1\}$ indique si l'arc $(i,j) \in A$ est sélectionné par le préparateur $k \in K$. De plus, les variables de flux $u_{ik} \geq 0$ indiquent la charge cumulative du préparateur $k \in K$ après avoir visité le nœud $i \in V$. Cette charge représente le nombre de commandes ramassées (c'est-à-dire la quantité de nœuds artificiels).

Ce problème est NP-Difficile car il est un cas spécial du Problème de Routage de Véhicules Capacités (CVRP) et du Problème de Ramassage et de Livraison (PDP) [3]. Par conséquent, dans la section suivante, une méthode heuristique est proposée pour résoudre des grandes instances.

4 Méthode heuristique

La méthode heuristique proposée est composée de deux étapes: une heuristique de construction basée sur une approche séquentielle, et une méta-heuristique de recherche locale pour améliorer la solution initiale. L'approche séquentielle est une approximation de l'approche jointe, puisque les optima locaux du problème de picking sont nécessaires pour évaluer la qualité de la solution de batching. Cependant, cette perte de qualité est compensée par la réduction du temps de calcul de l'approche séquentielle.

4.1 Heuristique de construction

En tant que technique de décomposition, l'heuristique de construction résout le problème en fixant l'ensemble des batches et en résolvant le problème de routage indépendamment comme un TSP pour chaque batch. La principale motivation de cette approche est d'employer une métrique de distance qui ne nécessite pas d'énumérer tous les trajets possibles pour mesurer la convenance de regrouper les commandes en batches: la distance de Hausdorff.

4.1.1 Distance de Hausdorff

La proximité entre deux commandes est définie comme la distance de Hausdorff entre leurs ensembles respectifs de positions de ramassage [14]. La distance de Hausdorff dirigée $H(o,k)$ entre deux commandes $o,k \in O$ est la distance maximale entre un article de la commande o et son article le plus proche de la commande k (2).

$$H(o,k) = \max_{i \in i(o)} \min_{j \in i(k)} d_{ij} \tag{2}$$

Étant donné que cette distance est asymétrique, c'est-à-dire $H(o,k) \neq H(k,o)$, $\forall o,k \in O$, la proximité c_{ij} entre les commandes $i,j \in O$ est définie comme le maximum entre les deux distances dirigées (3).

$$c_{ij} = \max\{H(i,j), H(j,i)\} \tag{3}$$

Par exemple, supposons une seule allée avec 7 positions de stockage. La commande o a des articles dans les positions $i(o) = \{1,6\}$ et la commande k a des articles dans les positions $i(k) = \{2,3\}$ (7).

$$H(i,j) = \max\{\min\{d_{12}, d_{13}\}, \min\{d_{62}, d_{63}\}\} = \max\{1,3\} = 3 \tag{4}$$
$$H(j,i) = \max\{\min\{d_{21}, d_{26}\}, \min\{d_{31}, d_{36}\}\} = \max\{1,2\} = 2 \tag{5}$$
$$c_{ij} = \max\{H(i,j), H(j,i)\} = \max\{3,3\} = 3 \tag{6}$$
$$\tag{7}$$

L'avantage de cette métrique est qu'elle est beaucoup moins coûteuse en calcul que le plus court chemin entre chaque paire de commandes.

4.1.2 Problème de regroupement

La distance de Hausdorff mesure la proximité géographique entre les articles de deux commandes distinctes (c'est-à-dire que la proximité intra-commande est 0). Cela conduit à des inconvénients lors de la modélisation comme un problème de partitionnement, car il créerait toujours des batches de commandes uniques en raison du manque d'incitation à regrouper les commandes ensemble. Ni les algorithmes de regroupement classiques, tels que K-means ou DBSCAN, ne peuvent être facilement adaptés pour contrôler la capacité des batches. De plus, la plupart d'entre eux reposent sur la distance euclidienne et notre métrique désirée ne peut pas être appliquée. Donc, un problème de "localisation-allocation" est proposé pour exploiter la distance

de Hausdorff comme mesure de la convenance pour regrouper les commandes. Spécifiquement, le problème de p-médian détermine le sous-ensemble de commandes qui sont plus proches des autres commandes en fonction des contraintes de capacité unitaire et volumétrique.

Le problème de p-médian consiste à sélectionner un sous-ensemble d'installations, parmi un ensemble de candidats, pour être utilisé pour desservir un ensemble de points de demande [11]. L'objectif est de minimiser la distance totale de déplacement entre les points de demande et les installations. Dans notre contexte, le concept de "batch" peut être interprété comme un hub installé qui "dessert" un sous-ensemble de commandes.

Soit I l'ensemble de commandes potentielles à partir desquelles sélectionner $\underline{B}$ batches, et J l'ensemble de commandes à desservir. De manière similaire aux sections précédentes, C_{unit} et C_{volume} sont le nombre maximal de commandes et le volume maximal qu'un batch peut desservir, respectivement; v_i est le volume de la commande $i \in I$; et $\underline{B}$ est calculé en utilisant l'équation (1).

Soit $x_{ij} \in \{0, 1\}$ la variable d'allocation, où $x_{ij} = 1$ si la commande j est attribuée au batch i, et $x_{ij} = 0$ sinon. Le livre [11] explique que, lorsque $I = J$ et $c_{ii} = 0$, $\forall i \in I$, les variables de localisation traditionnelles y_i peuvent être remplacées par les variables d'allocation x_{ii}, $\forall i \in I$. L'objectif est de maximiser la proximité totale entre les commandes dans le même batch, et peut être formulé comme suit:

$$\max \sum_{i \in I} \sum_{j \in J: j \neq i} c_{ij} x_{ij} \tag{8}$$

$$\text{s.t.} \sum_{i \in I} x_{ij} = 1 \quad \forall j \in J \tag{9}$$

$$\sum_{j \in J: j \neq i} x_{ij} \leq (|J| - \underline{B}) x_{ii} \quad \forall i \in I \tag{10}$$

$$\sum_{i \in I} x_{ii} \leq \underline{B} \tag{11}$$

$$\sum_{j \in J} x_{ij} \leq C_{unit} \quad \forall i \in I \tag{12}$$

$$\sum_{j \in J} v_j x_{ij} \leq C_{volume} \quad \forall i \in I \tag{13}$$

$$x_{ii} \in \{0, 1\} \quad \forall i \in I \tag{14}$$

$$x_{ij} \in \{0, 1\} \quad \forall i \in I, j \in J \tag{15}$$

$$\tag{16}$$

L'objectif (8) maximise la proximité totale entre les commandes dans le même batch. Les contraintes (9) garantissent que chaque commande est attribuée à exactement un batch. Les contraintes (10) interdisent l'attribution de commandes à des batches non sélectionnés. Les contraintes (11) garantissent que exactement $\underline{B}$ batches sont sélectionnés. Les contraintes (12) et (13) imposent les contraintes de capacité. Les contraintes (14) et (15) définissent le domaine des variables d'allocation.

Le problème de p-médian est NP-difficile. Cependant, puisque la quantité de commandes est relativement petite par rapport au nombre d'articles, nous pouvons utiliser des méthodes exactes pour le résoudre.

4.1.3 Problème de routage

Étant donné une solution au regroupement, le routage est décomposé pour chaque batch et résolu en parallèle en utilisant un solveur TSP, comme le montre [13]. La formulation des flux de marchandises proposée par [5] est utilisée, dans laquelle les sous-tours sont éliminés en utilisant les contraintes de conservation du flux (c'est-à-dire que le préparateur commence le trajet avec tous les articles et "livre" une unité à chaque position visitée).

La représentation du graphe G_k est similaire à celle présentée dans la formulation jointe, mais l'ensemble de nœuds est défini comme tous les articles du batch donné $\forall k \in K : V_k = \{i(o) : o \in O\}$, et l'ensemble d'arêtes, comme $E_k = \{(i,j) \in V_k \times V_k : i \neq j\}$. On définit la variable $x_{ij} \in \{0,1\}$ comme 1 si l'arc $(i,j) \in E_k$ est sélectionné, 0 sinon. La variable z^l_{ij} représente le nombre d'unités de l'article $l \in V_k \setminus \{0,n\}$ qui passent par l'arc $(i,j) \in E_k$. Le TSP associé au batch $k \in K$ peut être adapté de [5] comme suit.

$$\min \sum_{(i,j)\in E_k} d_{ij}x_{ij} \tag{17}$$

$$\text{s.t.} \quad \sum_{j\in V_k} x_{ij} = 1 \quad \forall i \in V_k \tag{18}$$

$$\sum_{i\in V_k} x_{ij} = 1 \quad \forall j \in V_k \tag{19}$$

$$\sum_{j\in V_k\setminus\{0,n\}} z^l_{0j} - \sum_{j\in V_k\setminus\{0,n\}} z^l_{j0} = -1 \quad \forall l \in V_k \setminus \{0,n\} \tag{20}$$

$$\sum_{j\in V_k} z^i_{ij} - \sum_{j\in V_k} z^i_{ji} = 1 \quad \forall i \in V_k \setminus \{0,n\} \tag{21}$$

$$\sum_{j\in V_k} z^l_{ij} - \sum_{j\in V_k} z^l_{ji} = 0 \quad \forall i \in V_k \setminus \{0,n\}, \forall l \in V_k \setminus \{0,n\} : l \neq i \tag{22}$$

$$z^l_{ij} \leq x_{ij} \quad \forall (i,j) \in E_k, \forall l \in V_k \setminus \{0,n\} \tag{23}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in E_k \tag{24}$$

$$z^l_{ij} \geq 0 \quad \forall (i,j) \in E_k, \forall l \in V_k \setminus \{0,n\} \tag{25}$$

$$\tag{26}$$

La fonction objectif (17) minimise la distance totale parcourue par le préparateur de commandes dans le batch $k \in K$. Les contraintes (18) et (19) garantissent que chaque article est visité exactement une fois. Les contraintes (20) garantissent que tous les articles sont collectés à 0 et livrés à un sommet de $V_k \setminus \{0,n\}$. Les contraintes (21) garantissent que chaque sommet de $V_k \setminus \{0,n\}$ reçoit exactement une unité d'article. Les contraintes (22) indiquent que la conservation du flux est satisfaite à chaque sommet de $V_k \setminus \{0,n\}$. La contrainte (23) évite le flux d'articles à travers les arcs qui ne sont pas sélectionnés.

Comme le montre [12], cette formulation nécessite $O(n^3)$ variables et contraintes, et elle fournit une relaxation LP plus forte que celle obtenue par la formulation MTZ (Miller-Tucker-Zemlin), qui est la plus courante pour le TSP.

4.2 Recherche locale

En raison de la structure immuable des batches dans le routage, deux opérateurs de déplacement sont appliqués à la solution courante: les opérateurs d'échange et de relocalisation. À chaque itération, jusqu'à ce que le critère d'arrêt soit atteint, un opérateur de déplacement est sélectionné aléatoirement et la première solution améliorante est obtenue. Pour éviter de répéter les mêmes solutions, une mémoire de Tabu Search adaptative [8] est utilisée pour stocker les propriétés des solutions qui sont interdites d'être sélectionnées à nouveau. Cette mémoire est ajustée pendant le processus de recherche pour forcer l'algorithme à explorer différentes régions de l'espace de recherche (diversification). De plus, les solutions non améliorantes peuvent être acceptées pour échapper aux optima locaux en utilisant le critère de Metropolis [4]. La meilleure solution trouvée est retournée comme solution finale après un nombre maximal d'itérations. L'orchestration de cet algorithme s'inspire principalement de l'heuristique de Recherche de Voisinage Variable (Variable Neighborhood Search, VNS) [9].

4.3 Opérateurs de déplacement

Les opérateurs de déplacement proposés cherchent à évaluer de nouveaux trajets entre les commandes qui sont dans les différents batches résultant du problème de p-médian. Le premier opérateur logique échange deux commandes entre deux batches différents, tandis que l'opérateur de relocalisation déplace une commande d'un batch à un autre.

L'opérateur d'échange est défini comme suit: étant donné deux batches b_1 et b_2, et deux commandes $o_1 \in b_1$ et $o_2 \in b_2$, l'opérateur échange les commandes entre les batches. Le batch d'origine b_1 est choisi en priorisant les batches les moins remplis (c'est-à-dire les batches avec le plus grand résidu de capacité), et le batch de destination b_2 est choisi aléatoirement parmi les batches qui peuvent accueillir la commande. Pour évaluer la solution, le solveur TSP est utilisé pour recalculer les trajets des batches affectés, et l'amélioration est la différence de distance entre les nouveaux et les anciens trajets. Cependant, cet opérateur n'est pas suffisant car il maintient le même nombre de batches dans la solution.

L'opérateur de relocalisation est donné par deux batches b_1 et b_2, et une commande $o_1 \in b_1$, l'opérateur déplace la commande vers le batch b_2. Les mêmes critères de sélection sont utilisés pour choisir les batches d'origine et de destination. Seul le batch de destination est re-routé, tandis que le batch d'origine est simplement mis à jour sans les articles de la commande déplacée. La combinaison de ces deux opérateurs permet à l'algorithme d'explorer toutes les solutions possibles dans l'espace de recherche, car nous pouvons obtenir des solutions avec un nombre différent de batches et des commandes différentes dans chaque batch. Ainsi, nous confirmons la connexité du voisinage, qui est une propriété souhaitable pour les algorithmes de recherche locale.

4.3.1 Mémoire de Tabu Search

La mémoire de Tabu Search est utilisée pour stocker les propriétés des solutions qui sont interdites d'être sélectionnées à nouveau. Une solution est représentée comme une correspondance du batch à l'ensemble des commandes dans le batch. Aucune solution n'est autorisée à être sélectionnée à nouveau si tous les batches ont le même ensemble de commandes. Cette mémoire est ajustée pendant le processus de recherche pour forcer l'algorithme à explorer différentes régions de l'espace de recherche (diversification). À la fin des itérations, la recherche est diversifiée

pour échapper aux optima locaux et la mémoire est réduite de moitié.

4.3.2 Recuit simulé

La stratégie de recuit simulé est utilisée pour accepter des solutions non améliorantes pour échapper aux optima locaux en fonction d'une probabilité. La probabilité p_{accept} d'accepter une solution non améliorante est donnée par le critère de Metropolis, qui est défini comme l'exponentielle de la différence négative entre la nouvelle et l'ancienne solution divisée par la température 27.

$$p_{accept} = \exp\left(-\frac{d_{new} - d_{current}}{T}\right) \tag{27}$$

La température est réduite à chaque itération par un taux de refroidissement, qui est une fonction décroissante du nombre d'itérations. Ce processus évite de prendre des solutions pires à la fin du processus de recherche (intensification).

5 Implémentation

L'implémentation complète de ce projet peut être trouvée dans ce dépôt. Il se concentre sur la conception intuitive et modulaire, plutôt que sur la performance, car il s'agit d'une preuve de concept. Par conséquent, il est adéquat pour les applications hors ligne, sans aucune contrainte de temps d'exécution.

5.1 Architecture du projet

Ce projet se compose de trois composants principaux: le domaine, l'application et les services. Le domaine (`src/domain/`) contient la logique métier de l'application, y compris les modèles et les procédures d'optimisation. L'application (`src/app/`) implémente trois cas d'utilisation pour interagir avec le domaine: `optimize`, `experiment` et `describe`. Les services (`src/services/`) contiennent les procédures d'entrée/sortie, y compris les classes de lecture et d'écriture, les calculateurs de distance et d'autres utilitaires externes au domaine. Les instructions d'installation et d'utilisation peuvent être trouvées dans le fichier `README.md` du dépôt.

5.1.1 Application

Le fichier `src/main.py` est le point d'entrée de l'application. Il initialise l'application et envoie le cas d'utilisation à la fonction correspondante. Le cas d'utilisation `optimize` est responsable de résoudre une seule instance du problème en utilisant une méthode spécifique; le cas d'utilisation `experiment`, pour exécuter un ensemble d'instances pour comparer différentes méthodes; et le cas d'utilisation `describe`, pour fournir une analyse des résultats. Ce projet s'attend à ce que les instances soient situées dans `data/`, et les résultats seront enregistrés `results/`.

5.1.2 Domaine

En ce qui concerne le domaine, le fichier `src/domain/BatchPicking.py` contient la classe principale, `BatchPicking`, qui orchestre le processus d'optimisation. Cette classe est responsable

de lire les instances, de résoudre le problème et de sauvegarder la meilleure solution trouvée dans un nombre maximal d'itérations.

Il existe deux approches d'optimisation principales implémentées dans ce projet: l'approche séquentielle et l'approche jointe. Le fichier `src/domain/joint.py` contient l'implémentation de l'approche jointe, qui résout le problème en considérant simultanément les problèmes de regroupement et de routage. Il existe deux versions de l'implémentation de l'approche jointe: la première utilise la bibliothèque OR-Tools et la seconde, un solveur commercial.

D'autre part, le fichier `src/domain/sequential.py` contient l'implémentation de l'approche séquentielle, qui résout le problème en le décomposant en deux sous-problèmes: le problème de regroupement et le problème de routage. En plus de la méthode de regroupement `PMedian`, le problème de partitionnement d'ensemble et les algorithmes de regroupement ont également été implémentés comme preuve de concept, dans les classes `GraphPartition` et `Clustering`, respectivement. D'un point de vue expérimental, le solveur TSP peut être un simple TSP (`TSPBase`), un TSP multi-flux de marchandises (`TSPMultiCommodityFlow`), ou une implémentation simplifiée de OR-Tools pour le problème VRP. La dernière option est la méthode de routage par défaut car elle montre la meilleure performance en termes de qualité de solution et de temps de calcul.

Comme on peut le voir, le projet est organisé de manière modulaire et implémente plusieurs modèles de conception (héritage, injection de dépendances, et autres) pour faciliter l'extension et la maintenance du code. Généralement, les méthodes `solve` et `optimize` sont les interfaces pour les méthodes d'optimisation génériques, tandis que les méthodes plus spécifiques sont implémentées sous les sous-modules correspondants, tels que `route`.

5.1.3 Services

Enfin, le fichier `src/services/benchmark.py` contient les procédures de benchmarking, qui sont responsables d'exécuter les expériences et d'analyser les résultats. La validation des résultats est effectuée en comparant les solutions avec le « S-shaped » chemin de traitement des commandes individuellement, et elle est extraite du dépôt UE. Parce que ces scripts sont une logique extérieure à l'application, nous conservons leur contenu tel qu'ils ont été reçus.

5.2 Améliorations potentielles

Des améliorations supplémentaires peuvent être apportées en utilisant un solveur plus spécialisé, tel que le solveur VRP développé à INRIA. Cependant, actuellement l'implémentation en Python ne couvre pas la fonction de ramassage et de livraison. Une solution de contournement consiste à implémenter le solveur en Julia et à l'intégrer avec python en utilisant PyJulia.

5.3 Langage de programmation et dépendances

Ce projet est implémenté en Python 3.10. Cette décision a été principalement motivée par les exigences du projet. Bien que Python soit un bon choix pour le prototypage en raison de sa simplicité et de sa polyvalence, il n'est pas le meilleur choix pour les applications critiques en termes de performances. Ce problème est connu sous le nom de Problème des Deux Langages, et Julia est une bonne alternative pour remplacer Python dans ce contexte. Julia offre un bon équilibre entre performances et productivité car c'est un langage compilé avec une syn-

taxe similaire à Python, et particulièrement adapté pour le calcul scientifique et les problèmes d'optimisation.

Toutes les dépendances de ce projet sont répertoriées dans le fichier `requirements.txt`. Une liste des principales est fournie ci-dessous:

- Pyomo: Un langage de modélisation d'optimisation open-source basé sur Python.
- Gurobi: Un solveur commercial pour les problèmes de programmation mathématique.
- OR-Tools: Un ensemble de bibliothèques pour les problèmes d'optimisation combinatoire.

6 Expériences numériques

Les expériences numériques sont effectuées sur toutes les instances disponibles dans `/data` en utilisant un processeur Intel Core i9 8 cœurs à 2,3 GHz. Les solutions sont comparées avec la solution de référence, qui est le S-shaped chemin fourni pour chaque commande.

6.1 Caractérisation des instances

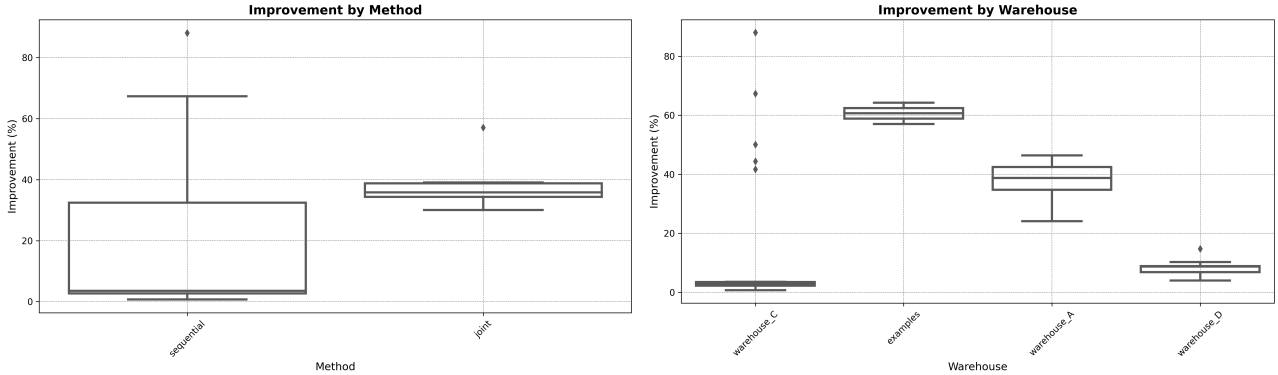
Ces instances sont classées par entrepôt: `A`, `B`, `C`, `D` et `toy`. Elles peuvent être caractérisées par la taille moyenne des commandes (c'est-à-dire le nombre d'articles par commande). Nous observons que les instances `A` et `D` ont la plus petite taille moyenne, tandis que les instances `B` et `C` ont la plus grande taille moyenne. Cette caractérisation est pertinente pour notre approche de solution car les commandes avec un grand nombre d'articles nécessitent plus d'efforts d'optimisation pour résoudre le problème de routage. Au contraire, si l'instance contient de nombreuses commandes avec peu d'articles chacune, il y a plus d'opportunités pour regrouper les commandes ensemble, et la qualité de la solution est attendue meilleure.

6.2 Résultats

L'instance `toy`, en plus de la solution du chemin S-shaped, contient une solution supplémentaire qui s'améliore de 44% par rapport à la solution de référence. Cependant, nous observons que notre solution, dans les approches séquentielle et jointe, surpasse cette amélioration de 20% et 13%, respectivement, comme le montre le tableau ci-dessous.

Instance	Provided	Sequential	Joint
toy	44%	65%	57%

Les résultats présentés concernent `50` exécutions uniques de combinaisons d'instances et de méthodes. Toutes les solutions ont été vérifiées comme réalisables à l'aide de l'outil de validation fourni dans le dépôt UE. La distribution de l'amélioration relative par rapport à la solution de référence est présentée dans les box-plots suivants. Chaque solution d'instance correspond à un point de données dans les box-plots. Ils sont regroupés soit par méthode ou par entrepôt.



Les premiers box-plots 6.2 comparent la distribution de l'amélioration (%) des méthodes séquentielle et jointe. Bien que l'amélioration médiane de la méthode jointe soit plus élevée que la méthode séquentielle, les plus grandes améliorations sont obtenues par la méthode séquentielle. Cela s'explique par le fait que la méthode jointe n'a pas pu trouver de solutions dans de nombreux cas. Comme avantage, la méthode jointe montre une plage interquartile plus étroite, suggérant une performance plus cohérente à travers les instances.

De même, les deuxièmes box-plots 6.2 montre la distribution de l'amélioration (%) par entrepôt. L'entrepôt `C` montre les plus grandes améliorations (88% au maximum) suivi, hors instances "exemples", par l'entrepôt `A` (45%). Cependant, l'entrepôt `A` présente une amélioration médiane plus élevée par rapport aux autres entrepôts, avec presque 40% d'amélioration.

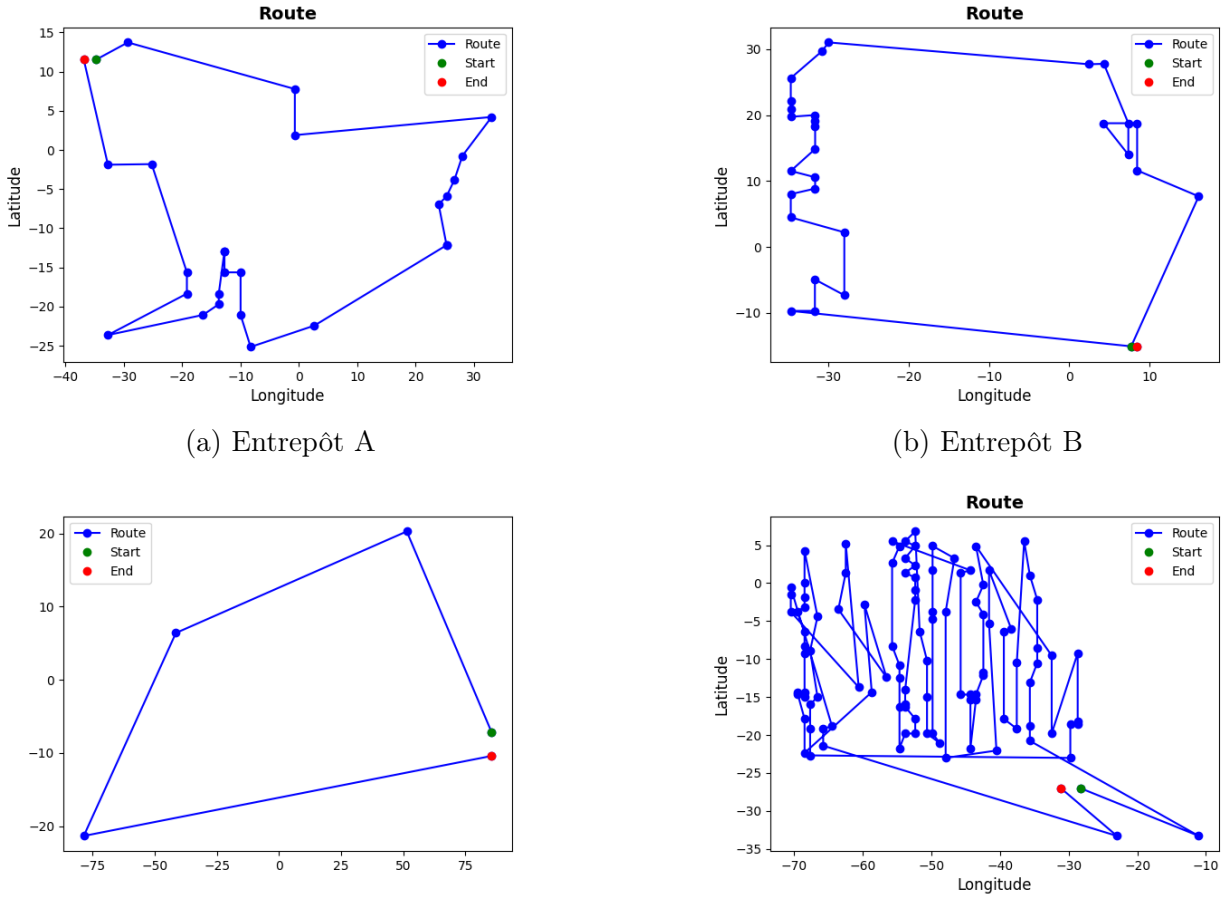
Les métriques globales obtenues à partir de toutes les méthodes et instances sont résumées dans le tableau suivant 1. L'amélioration globale moyenne est de 20% en moyenne, et le 90ème percentile d'amélioration est de 47%. En ce qui concerne le temps de calcul, l'approche séquentielle est considérablement plus rapide que l'approche jointe, comme le montre le table suivant. Le temps d'exécution moyen est de 305 secondes, et le 90ème percentile du temps d'exécution est de 1804 secondes, ce qui est un temps raisonnable pour les applications hors ligne. Ces résultats suggèrent que la méthode proposée fournit un bon compromis entre la qualité de la solution et le temps de calcul, et elle est adaptée pour les grandes instances du problème.

Métrique	Valeur
Amélioration moyenne (%)	19,98
Amélioration P90 (%)	46,74
Temps d'exécution moyen (s)	304,81
Temps d'exécution P90 (s)	1804

Table 1: Métriques globales

Plusieurs instances ont été résolues uniquement par l'approche séquentielle, car l'approche jointe était trop lente et a atteint la limite de temps. Des exemples de chemins pour chaque entrepôt sont affichés dans les images 1. Les transits entre allées n'étant pas modélisés, les itinéraires ne représentent pas la trajectoire réelle du préparateur. Cependant, ceux-ci servent de référence pour visualiser l'enchaînement des positions visitées dans l'entrepôt.

Il convient de noter que, bien que nous ayons testé toutes les instances fournies, nous ne rapportons que les instances avec des solutions améliorées par rapport au chemin S-shaped, car



(a) Entrepôt A

(b) Entrepôt B

(c) Entrepôt C

(d) Entrepôt D

Figure 1: Exemples d'itinéraires pour chaque entrepôt

c'est notre borne supérieure sur la distance totale parcourue. Il y avait plusieurs instances avec 0% d'amélioration, et aucune d'entre elles n'a rapporté une amélioration négative. Un tableau complet des résultats peut être trouvé dans la section Annexes.

7 Conclusions

Le Batch-Picking est un problème pertinent dans le contexte de la logistique d'entrepôt, et il a été abordé dans la littérature sous les approches jointe et séquentielle. Principalement, la littérature connexe se concentre sur un problème de partitionnement d'ensemble pour l'approche séquentielle, et le problème de routage de véhicules groupés pour l'approche jointe. Dans ce travail, nous avons proposé une méthode heuristique pour résoudre le problème, qui est basée sur le problème de p-médian et le problème du voyageur de commerce (TSP). Cet algorithme surpasse la solution de référence dans presque toutes les instances, et dans celles où ce n'est pas le cas, l'amélioration n'est pas négative. Une approche alternative basée sur le problème de ramassage et de livraison (PDP) a également été proposée pour modéliser le problème conjointement, et elle a été résolue en utilisant la méta-heuristique de recherche locale. La méthode proposée fournit un bon compromis entre la qualité de la solution et le temps de

calcul, et elle est adaptée pour les grandes instances du problème.

8 Annexes

8.1 Formulation jointe: problème de routage de véhicules groupés

Étant donné que les articles sont regroupés en commandes, le problème de picking peut être modélisé comme un problème de routage de véhicules groupés (clustered vehicle routing problem, CluVRP) comme le montre [1]. Dans le CluVRP, les articles appartenant à une commande doivent être visités par le même préparateur de commandes, ce qui permet au préparateur de visiter leurs positions dans n'importe quel ordre au sein du batch (c'est-à-dire que les positions de ramassage sont mélangées entre les commandes dans le même batch). Ce dernier est une considération importante car la plupart de la littérature CluVRP suppose des contraintes de cluster rigides, de sorte que le préparateur doit terminer tous les articles d'une commande avant de commencer la suivante, même s'il existe un article plus proche d'une autre commande dans le même batch.

On définit la variable d'affectation $y_{ik} \in \{0,1\}$ comme 1 si l'article $i \in V$ est sélectionné pour le batch $k \in K$, 0 sinon. On définit la variable de séquence $x_{ijk} \in \{0,1\}$ comme 1 si l'article $i \in V$ est visité avant l'article $j \in V$ dans le batch $k \in K$, 0 sinon. Le modèle présenté par [1] peut être adapté comme suit.

$$\min \sum_{k\in K} \sum_{(i,j)\in E} d_{ij} x_{ijk} \tag{28}$$

$$\text{s.t.} \quad \sum_{k\in K} y_{ik} = 1 \quad \forall i \in V \setminus \{0, n-1\} \tag{29}$$

$$\sum_{k\in K} y_{0k} = \sum_{j\in V\setminus\{0,n-1\}} \sum_{k\in K} x_{0jk} \leq \underline{K} \tag{30}$$

$$\sum_{j\in V} x_{ijk} = \sum_{j\in V} x_{jik} = y_{ik} \quad \forall i \in V, \forall k \in K \tag{31}$$

$$\sum_{j\in V\setminus\{0,n-1\}} v_j y_{jk} \leq C_{\text{vol}} \quad \forall k \in K \tag{32}$$

$$\sum_{j\in V\setminus\{0,n-1\}} q_j y_{jk} \leq C_{\text{qty}} \quad \forall k \in K \tag{33}$$

$$\sum_{i\in S} \sum_{j\in V\setminus S} x_{ijk} \geq y_{pk} \quad \forall S \subset V \setminus \{0, n-1\}, \forall p \in S, \forall k \in K \tag{34}$$

$$\sum_{(i,j)\in\delta^+(P(r))} \sum_{k\in K} x_{ijk} = \sum_{(i,j)\in\delta^-(P(r))} \sum_{k\in K} x_{ijk} \geq 1 \quad \forall r \in R \tag{35}$$

$$y_{ik} = y_{jk} \quad \forall i,j \in P(r), \forall r \in R, \forall k \in K \tag{36}$$

$$x_{ijk} \in \{0,1\} \quad \forall (i,j) \in E, \forall k \in K \tag{37}$$

$$y_{ik} \in \{0,1\} \quad \forall i \in V, \forall k \in K \tag{38}$$

La fonction objectif (28) minimise la distance totale parcourue par les préparateurs de commandes. Les contraintes (29) garantissent que chaque article est affecté à exactement un

batch. Les contraintes (30) assurent que tous les batches commencent au dépôt. Les contraintes (31) garantissent que la conservation du flux est satisfaite pour l'emplacement de stockage de chaque article. Les contraintes (32) et (33) sont les contraintes de capacité de volume et de quantité pour chaque batch, respectivement. Les contraintes (34) éliminent les sous-tours en forçant le flux à être nul pour tous les sous-ensembles d'articles $S \subset V$. Les contraintes (35) forcent les articles du même batch à être visités par le même préparateur de commandes, ce qui permet de visiter leurs emplacements dans n'importe quel ordre. Dernière contrainte (36) se réfère à la condition d'intégrité de la commande pour s'assurer que tous les articles d'une commande sont affectés au même batch.

8.2 Tableau de résultats

Le tableau suivant présente les résultats complets des expériences numériques.

Table 2: Benchmarking Results

Instance Name	Method	Improvement (%)	Execution Time (s)
warehouse_C/2023-09-20_18-00-00_RACK-64	sequential	88.02	3.47
warehouse_C/2023-09-15_15-00-00_RACK-6	sequential	67.29	12.09
examples/toy_instance	sequential	64.25	1.60
examples/toy_instance	joint	57.05	2.00
warehouse_C/2023-09-11_12-00-00_RACK-4	sequential	50.01	3.18
warehouse_A/data_2023-05-23	sequential	46.38	6.32
warehouse_A/data_2023-05-22	sequential	46.33	7.07
warehouse_C/2023-09-08_15-00-00_RACK-4	sequential	44.38	1.29
warehouse_A/data_2023-05-27	sequential	43.22	8.55
warehouse_A/data_2023-05-24	sequential	42.26	6.62
warehouse_C/2023-09-22_18-00-00_RACK-4	sequential	41.63	1.56
warehouse_A/data_2023-05-25	sequential	40.75	6.73
warehouse_A/data_2023-05-23	joint	39.08	1804.02
warehouse_A/data_2023-05-22	joint	38.43	1804.48
warehouse_A/data_2023-05-27	joint	35.83	1806.19
warehouse_A/data_2023-05-24	joint	35.19	1807.01
warehouse_A/data_2023-05-25	joint	33.42	1804.47
warehouse_A/data_2023-05-26	joint	30.07	1808.86
warehouse_A/data_2023-05-26	sequential	24.12	6.34
warehouse_D/data_2023-01-30_00	sequential	14.68	1.31
warehouse_D/data_2023-01-30_08	sequential	10.28	9.90
warehouse_D/data_2023-01-31_20	sequential	8.86	15.31
warehouse_D/data_2023-01-30_20	sequential	8.85	37.72
warehouse_D/data_2023-01-31_08	sequential	8.75	927.78
warehouse_D/data_2023-01-30_16	sequential	8.31	6.79
warehouse_D/data_2023-01-30_12	sequential	6.85	928.87
warehouse_D/data_2023-01-30_04	sequential	5.29	9.58

Continued on next page

Table 2: Benchmarking Results

Instance Name	Method	Improvement (%)	Execution Time (s)
warehouse_D/data_2023-01-31_04	sequential	4.01	32.32
warehouse_C/2023-09-17_19-30-00_RACK-64	sequential	3.56	1.71
warehouse_C/2023-09-19_18-00-00_RACK-10	sequential	3.52	17.77
warehouse_C/2023-09-14_08-30-00_RACK-30	sequential	3.42	150.95
warehouse_C/2023-09-10_19-30-00_RACK-10	sequential	3.22	30.82
warehouse_C/2023-09-17_19-30-00_RACK-30	sequential	3.13	18.80
warehouse_C/2023-09-11_15-00-00_RACK-4	sequential	3.10	2.55
warehouse_C/2023-09-14_15-00-00_RACK-30	sequential	3.10	54.81
warehouse_C/2023-09-15_18-00-00_RACK-30	sequential	3.01	197.76
warehouse_C/2023-09-12_18-00-00_RACK-10	sequential	2.99	16.79
warehouse_C/2023-09-21_08-30-00_RACK-30	sequential	2.96	32.06
warehouse_C/2023-09-08_15-00-00_RACK-30	sequential	2.71	22.54
warehouse_C/2023-09-11_08-00-00_RACK-30	sequential	2.67	2.79
warehouse_C/2023-09-11_15-00-00_RACK-12	sequential	2.43	461.33
warehouse_C/2023-09-08_15-00-00_RACK-10	sequential	2.35	2.03
warehouse_C/2023-09-20_18-00-00_RACK-30	sequential	2.27	119.06
warehouse_C/2023-09-13_15-00-00_RACK-10	sequential	2.22	13.03
warehouse_C/2023-09-24_19-30-00_RACK-6	sequential	2.14	4.57
warehouse_C/2023-09-15_12-00-00_RACK-12	sequential	1.82	931.05
warehouse_C/2023-09-20_08-00-00_RACK-30	sequential	1.68	249.37
warehouse_C/2023-09-14_08-30-00_RACK-64	sequential	1.35	29.28
warehouse_C/2023-09-15_12-00-00_RACK-6	sequential	1.06	2.52
warehouse_C/2023-09-21_08-30-00_RACK-64	sequential	0.73	7.24

8.3 Batching: problème de partitionnement

Pour formuler le problème de partitionnement, on définit la variable d'affectation $y_{ik} \in \{0, 1\}$ comme 1 si la commande $i \in R$ est affectée au batch $k \in K$, 0 sinon. La variable $x_{ij} \in \{0, 1\}$ est 1 si les commandes $i, j \in R$ sont affectées au même batch, 0 sinon. Le problème de batching est formulé comme suit [7].

$$\max \sum_{(i,j)\in R\times R} c_{ij}x_{ij} \tag{39}$$

$$\text{s.t.} \quad \sum_{k\in K} y_{ik} = 1 \quad \forall i \in R \tag{40}$$

$$y_{ik} + y_{jk} \leq 1 + x_{ij} \quad \forall (i,j) \in R \times R, \forall k \in K \tag{41}$$

$$\sum_{i\in R} v_i y_{ik} \leq C_{\text{vol}} \quad \forall k \in K \tag{42}$$

$$\sum_{i\in R} q_i y_{ik} \leq C_{\text{qty}} \quad \forall k \in K \tag{43}$$

$$x_{ij} = x_{ji} \quad \forall (i,j) \in R \times R \tag{44}$$

$$x_{ii} = 0 \quad \forall i \in R \tag{45}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in R \times R \tag{46}$$

$$y_{ik} \in \{0,1\} \quad \forall i \in R, \forall k \in K \tag{47}$$

$$\tag{48}$$

La fonction objectif (39) minimise la proximité totale entre les commandes dans le même batch. Les contraintes (40) garantissent que chaque commande est affectée à exactement un batch. Les contraintes (41) stipulent que si deux commandes sont affectées au même batch, alors l'arc entre elles est un arc intra-batch. Les contraintes (42) et (43) sont les contraintes de capacité de volume et de quantité pour chaque batch, respectivement. Les contraintes (44) et (45) suppriment les arcs symétriques et les boucles, respectivement. Ce problème de partitionnement est connu pour être NP-difficile lorsque le nombre de batches est supérieur à 2 [7]. Ce modèle n'a pas réussi à fournir des lots avec plus d'une commande en raison de la limitation présentée dans la description de l'algorithme proposé.

References

[1] Babiche Aerts, Trijntje Cornelissens, and Kenneth Sörensen. The joint order batching and picker routing problem: Modelled and solved as a clustered vehicle routing problem. *Computers & Operations Research*, 129:105168, 2021.

[2] Maria Battarra, Jean-François Cordeau, and Manuel Iori. Chapter 6: pickup-and-delivery problems for goods transportation. In *Vehicle Routing: Problems, Methods, and Applications, Second Edition*, pages 161–191. SIAM, 2014.

[3] Gerardo Berbeglia, Jean-François Cordeau, Irina Gribkovskaia, and Gilbert Laporte. Static pickup and delivery problems: a classification scheme and survey. *Top*, 15:1–31, 2007.

[4] Dimitris Bertsimas and John Tsitsiklis. Simulated annealing. *Statistical science*, 8(1):10–15, 1993.

[5] A Claus. A new formulation for the travelling salesman problem. *SIAM Journal on Algebraic Discrete Methods*, 5(1):21–25, 1984.

[6] Guy Desaulniers, Jacques Desrosiers, Andreas Erdmann, Marius M Solomon, and François Soumis. Vrp with pickup and delivery. *The vehicle routing problem*, 9:225–242, 2002.

[7] Carlos Eduardo Ferreira, Alexander Martin, C Carvalho de Souza, Robert Weismantel, and Laurence A Wolsey. The node capacitated graph partitioning problem: a computational study. *Mathematical programming*, 81:229–256, 1998.

[8] Fred Glover. Tabu search: A tutorial. *Interfaces*, 20(4):74–94, 1990.

[9] Pierre Hansen, Nenad Mladenović, Jack Brimberg, and José A Moreno Pérez. *Variable neighborhood search*. Springer, 2019.

[10] Sebastian Henn, Sören Koch, and Gerhard Wäscher. *Order batching in order picking warehouses: a survey of solution approaches*. Springer, 2012.

[11] Gilbert Laporte, Stefan Nickel, and Francisco Saldanha-da Gama. *Introduction to location science*. Springer, 2019.

[12] Manfred Padberg and Ting-Yi Sung. An analytical comparison of different formulations of the travelling salesman problem. *Mathematical Programming*, 52(1-3):315–357, 1991.

[13] André Scholz, Sebastian Henn, Meike Stuhlmann, and Gerhard Wäscher. A new mathematical programming formulation for the single-picker routing problem. *European Journal of Operational Research*, 253(1):68–84, 2016.

[14] Abdel Aziz Taha and Allan Hanbury. An efficient algorithm for calculating the exact hausdorff distance. *IEEE transactions on pattern analysis and machine intelligence*, 37(11):2153–2163, 2015.