Projet MAOA
Autour du problème de production et distribution intégré

Ariana Carnielli et Milo Roucairol

Table des matières

1 Introduction

Ce projet s'intéresse au *problème de production et distribution intégré* (PDI) mono-produit avec véhicules identiques, dont la description détaillée est donnée dans [2]. L'objectif de ce problème est de minimiser les coûts dans une chaîne de production d'un seul type de produit, dans laquelle il y a un fournisseur et n revendeurs, le temps est supposé discrétisé et à un horizon fini, et les choix portent sur combien produire à chaque pas de temps et quand et comment livrer la production aux clients.

Il s'agit de la jonction de deux problèmes classiques, le *problème de lot-sizing* (LSP) et le *problème de tournée de véhicules* (VRP). Le LSP peut être vu comme une version du PDI dans lequel on s'intéresse uniquement aux décisions de production, ignorant le transport entre l'usine de production et les entrepôts des n revendeurs. Le choix porte donc uniquement sur combien produire et combien destiner à chaque entrepôt, à chaque pas de temps. Le VRP s'intéresse plutôt à la distribution en soi : étant données les quantités à distribuer à chaque entrepôt à un instant de temps donné, la quantité de véhicules identiques dont on disposé et leur capacité, l'objectif est de minimiser le coût de transport, avec la contrainte que chaque revendeur doit être livré par un seul véhicule. Dans ce rapport, on utilise comme synonymes les mots *fournisseur* et *usine*, ainsi que les mots *revendeurs*, *clients* et *entrepôts*.

Comme indiqué dans [2], l'optimisation en un seul temps des choix de production et distribution peut conduire à des solutions nettement meilleurs que l'optimisation séparée de ces deux types de choix, ce qui motive l'étude du PDI. Cependant, il s'agit d'un problème NP-difficile qui peut être modélisé par un PLNE et dont la résolution efficace est un défi important en recherche opérationnelle.

2 Description des problèmes traités

Dans cette section, pour fixer les notations utilisées dans la suite du projet et expliciter quelques choix qui ont été faits, on reprend les formulations de LSP, VRP et PDI par des PLNE données à l'énoncé du projet et dans [2]. Dans ce rapport, n désigne le nombre de revendeurs, ℓ représente la quantité de pas de temps considérée, le nombre de véhicules disponibles est noté par m et leur capacité vaut Q. L'ensemble des revendeurs et du fournisseur sont identifiés à l'ensemble des sommets d'un graphe à $n+1$ sommets : le sommet indexé par 0 représente le fournisseur et les sommets indexés par 1 à n représentent les revendeurs. On remarque que certaines quantités décrites dans la suite peuvent être définies pour tous les sommets, alors que d'autres ne sont définies que pour les revendeurs, et on précisera à chaque fois alors si l'on travaille dans l'ensemble $\{0,\dots,n\}$ ou $\{1,\dots,n\}$. On suppose aussi que les quantités de produits peuvent être des nombres réels positifs quelconques (c'est-à-dire qu'ils ne sont pas forcément des quantités entières).

2.1 Problème de lot-sizing

On utilise dans ce projet une variante du problème LSP qui essaie de prendre en compte les prix de visite des clients. Cela est motivé par le fait que cette variante est utilisée dans les solutions approchées au PDI mises en œuvre sur ce projet. Basée sur la formulation (1)–(5) de l'énoncé du projet, la formulation utilisée prend aussi en compte une heuristique du coût de transport et s'écrit comme un PLNE de la façon suivante :

$$\min \sum_{t=0}^{\ell-1} \left[up_t + fy_t + \sum_{i=0}^{n} h_i I_{it} + \sum_{i=1}^{n} SC_{it} z_{it} \right] \tag{1a}$$

$$\text{s.c. } I_{0,t-1} + p_t = \sum_{i=1}^{n} q_{it} + I_{0,t} \qquad t \in \{0,\dots,\ell-1\} \tag{1b}$$

$$I_{i,t-1} + q_{it} = d_{it} + I_{i,t} \qquad i \in \{1,\dots,n\},\ t \in \{0,\dots,\ell-1\} \tag{1c}$$

$$p_t \le M_t y_t \qquad t \in \{0,\dots,\ell-1\} \tag{1d}$$

$$I_{0,t} \le L_0 \qquad t \in \{0,\dots,\ell-1\} \tag{1e}$$

$$I_{i,t-1} + q_{it} \le L_i \qquad i \in \{1,\dots,n\},\ t \in \{0,\dots,\ell-1\} \tag{1f}$$

$$q_{it} \le \widetilde{M}_{it} z_{it} \qquad i \in \{1,\dots,n\},\ t \in \{0,\dots,\ell-1\} \tag{1g}$$

$$\sum_{i=1}^{n} q_{it} \le Qm \qquad t \in \{0,\dots,\ell-1\} \tag{1h}$$

$$I_{it} \in \mathbb{R}_+ \qquad i \in \{0,\dots,n\},\ t \in \{0,\dots,\ell-1\} \tag{1i}$$
$$q_{it} \in \mathbb{R}_+,\ z_{it} \in \{0,1\} \qquad i \in \{1,\dots,n\},\ t \in \{0,\dots,\ell-1\} \tag{1j}$$
$$p_t \in \mathbb{R}_+,\ y_t \in \{0,1\} \qquad t \in \{0,\dots,\ell-1\} \tag{1k}$$

Les variables de décision et les données de ce problème sont décrits dans les Tables 1 et 2. Les valeurs de M_t et $\widetilde{M}_{it}$ sont définies par

$$M_t = \min\left(C, \sum_{j=t}^{\ell-1}\sum_{i=1}^{n} d_{ij}\right) \qquad t \in \{0,\dots,\ell-1\} \tag{2a}$$
$$\widetilde{M}_{it} = \min\left(L_i, Q, \sum_{j=t}^{\ell-1} d_{ij}\right) \qquad i \in \{1,\dots,n\},\ t \in \{0,\dots,\ell-1\} \tag{2b}$$

Variable	Domaine	Indices	Description
p_t	$\mathbb{R}_+$	$0 \le t \le \ell-1$	Quantité produite par l'usine à l'instant t
y_t	$\{0,1\}$	$0 \le t \le \ell-1$	1 si production lancée à l'instant t, 0 sinon
I_{it}	$\mathbb{R}_+$	$0 \le i \le n$ $0 \le t \le \ell-1$	Stock en i à la fin de la période t
q_{it}	$\mathbb{R}_+$	$1 \le i \le n$ $0 \le t \le \ell-1$	Quantité à être livrée au revendeur i à la période t
z_{it}	$\{0,1\}$	$1 \le i \le n$ $0 \le t \le \ell-1$	1 si une livraison doit être faite à i à l'instant t, 0 sinon

Table 1 – Variables de décision du problème LSP (1)

Donnée	Domaine	Indices	Description
u	$\mathbb{R}_+$		Coût unitaire de production
f	$\mathbb{R}_+$		Coût fixe de production par période où la production est lancée
h_i	$\mathbb{R}_+$	$0 \le i \le n$	Coût unitaire de stockage
SC_{it}	$\mathbb{R}_+$	$1 \le i \le n$ $0 \le t \le \ell-1$	Coût fixe estimé pour acheminer la production à i à la période t
$I_{i,-1}$	$\mathbb{R}_+$	$0 \le i \le n$	Quantité initiale en stock en i
$d_{i,t}$	$\mathbb{R}_+$	$1 \le i \le n$ $0 \le t \le \ell-1$	Demande du client i à la période t
L_i	$\mathbb{R}_+$	$0 \le i \le n$	Capacité de stockage en i
C	$\mathbb{R}_+$		Capacité de production
Q	$\mathbb{R}_+$		Capacité maximale d'un véhicule
m	$\mathbb{N}$		Nombre de véhicules disponibles

Table 2 – Donnés du problème LSP (1)

La valeur M_t est une borne sur ce que l'on produit à l'instant t, p_t, qui est le minimum entre la capacité de production de l'usine et la somme des demandes de tous les clients entre l'instant t et la fin de l'horizon de temps considéré. De façon similaire, $\widetilde{M}_{it}$ est une borne de q_{it}, la quantité de

produit envoyée à i à l'instant t, qui est égale au minimum entre la capacité Q d'un véhicule, la capacité de stockage L_i de i, et la somme des demandes de i entre t et la fin de l'horizon de temps.

Par rapport à la formulation donnée dans l'énoncé du projet, la formulation (1) ci-dessus présente quelques modifications. La première est le décalage de l'indice t : plutôt que de considérer que t va de 1 à ℓ comme dans l'énoncé, on suppose ici que t va de 0 à $\ell - 1$, pour avoir une formulation plus proche de celle qui a été implémentée.

On a ajouté le terme $\sum_{i=1}^{n} SC_{it} z_{it}$ dans la fonction objectif. Chaque valeur SC_{it} représente une estimée du coût pour livrer le client i à l'instant t, qui est considérée comme une donnée dans ce problème mais qui, en pratique, est issue d'une heuristique, comme détaillé dans la Section 3.2. Les nouvelles variables de décision z_{it} sont binaires et déterminent si le client i sera visité à l'instant t ou pas (donc $z_{it} = 1$ si $q_{it} > 0$). Ainsi, la somme $\sum_{i=1}^{n} SC_{it} z_{it}$ représente une estimée du coût total d'aller de l'entrepôt aux clients auxquels on doit livrer à l'instant t.

Suite à l'ajout des variables de décision z_{it}, on a aussi ajouté la contrainte (1g), qui est la contrainte liante entre z_{it} et q_{it} implémentant le fait que $z_{it} = 1$ si $q_{it} > 0$. Elle est identique à la contrainte liante entre ces mêmes variables dans le PDI de la formulation (1)–(16) de [2].

Toujours dans la fonction objectif, une différence par rapport à l'énoncé est le terme $\sum_{i=0}^{n} h_i I_{it}$: à l'énoncé, cette somme commence à $i = 1$, ce qui revient à considérer qu'il n'y a pas de coût de stockage chez le fournisseur. Or, les formulations du problème PDI de [2] considèrent toutes qu'il y a bien un coût de stockage chez le fournisseur, et les instances fournies ont aussi des coûts de stockage non-nuls chez le fournisseur, ce pourquoi dans notre formulation cette somme commence à $i = 0$.

Notre formulation de la contrainte (1e) est aussi différente de celle de l'énoncé, qui contient un terme $I_{0,t-1}$ à la place de $I_{0,t}$ dans le membre de gauche de l'inégalité. Ainsi, la formulation de l'énoncé ne contraint pas la valeur finale du stock chez le fournisseur $I_{0,\ell-1}$ à être plus petite que la capacité de stockage, ce qui veut dire qu'une solution avec un stock final chez le fournisseur plus grand que L_0 serait admissible (bien que certainement pas optimale). D'autre part, la formulation de l'énoncé impose que $I_{0,-1} \leq L_0$, qui est une inégalité ne portant que sur des données du problème et pas sur des variables de décision. Le fait de remplacer $I_{0,t-1}$ par $I_{0,t}$ permet d'imposer une contrainte sur l'état du stock de l'usine pendant tout l'horizon de temps, y compris l'instant final, et fait toujours intervenir une variable de décision. Remarquons aussi que la formulation (1e) ci-dessus coïncide avec la contrainte correspondante pour le problème PDI donnée dans [2, équation (5)].

Notre formulation contient aussi la contrainte (1h), qui indique que, à chaque instant t, l'on ne peut envoyer aux clients qu'au plus Qm, qui correspond à la somme des capacités de tous les véhicules. Cette contrainte, absente de la formulation de l'énoncé, est importante dans notre formulation car, si elle est violée à un instant t, alors il est impossible de distribuer les produits aux clients avec les véhicules dont on dispose. Cela pose un problème pour la résolution par nos heuristiques approchées (décrites plus loin dans la Section 3.2). On a observé que, sans ajouter cette contrainte, elle peut être violée lors de la résolution approchée de quelques instances, comme nous avons constaté pour l'instance `B_050_instance2.prp`.

Finalement, on remarque que, dans les contraintes (1b), (1c) et (1f), le cas $t = 0$ a une différence par rapport aux autres : lorsque $t = 0$, les termes $I_{0,t-1}$ et $I_{i,t-1}$ deviennent $I_{0,-1}$ et $I_{i,-1}$, qui sont des données du problème, alors que, pour $t > 0$, $I_{0,t-1}$ et $I_{i,t-1}$ sont des variables de décision. Cette remarque est très importante pour l'implémentation pratique du problème (1).

2.2 Problème de tournée de véhicules

On présente maintenant la formulation du problème de tournée de véhicules utilisée dans ce projet, qui, comme le LSP de la section précédente, est utilisée dans les solutions approchées au PDI mises en œuvre sur ce projet. Dans le problème VRP, on souhaite livrer des produits à une certaine quantité de clients, qui peuvent ne pas être tous les clients $\{1, \ldots, n\}$. Pour avoir une notation plus claire, on dénote ici par $\mathcal{V}_C$ l'ensemble des clients à être livrés et $\mathcal{V} = \mathcal{V}_C \cup \{0\}$

l'ensemble contenant ces mêmes clients et le fournisseur. Le problème VSP peut alors se formuler, comme à l'énoncé du projet, par le PLNE suivant :

$$\min \sum_{i \in \mathcal{V}} \sum_{\substack{j \in \mathcal{V} \\ j \neq i}} c_{ij} x_{ij} \tag{3a}$$

$$\text{s.c.} \sum_{j \in \mathcal{V}_C} x_{0j} \leq m \tag{3b}$$

$$\sum_{i \in \mathcal{V}_C} x_{i0} \leq m \tag{3c}$$

$$\sum_{\substack{j \in \mathcal{V} \\ j \neq i}} x_{ij} = 1 \qquad i \in \mathcal{V}_C \tag{3d}$$

$$\sum_{\substack{i \in \mathcal{V} \\ i \neq j}} x_{ij} = 1 \qquad j \in \mathcal{V}_C \tag{3e}$$

$$w_i - w_j \geq b_i - (Q + b_i)(1 - x_{ij}) \qquad i, j \in \mathcal{V}_C, i \neq j \tag{3f}$$

$$x_{ij} \in \{0, 1\} \qquad i, j \in \mathcal{V}, i \neq j \tag{3g}$$

$$w_i \in [0, Q] \qquad i \in \mathcal{V}_C \tag{3h}$$

Les variables de décision et les données de ce problème sont décrits dans les Tables 3 et 4.

Variable	Domaine	Indices	Description
x_{ij}	$\{0,1\}$	$i, j \in \mathcal{V}$ $i \neq j$	1 si un véhicule va de i à j, 0 sinon
w_i	$[0, Q]$	$i \in \mathcal{V}_C$	Variables des contraintes MTZ pour briser les sous-cycles

Table 3 – Variables de décision du problème VRP (3)

Donnée	Domaine	Indices	Description
m	$\mathbb{N}^*$		Quantité de véhicules disponibles
Q	$\mathbb{R}_+$		Capacité maximale d'un véhicule
b_i	$\mathbb{R}_+$	$i \in \mathcal{V}_C$	Quantité de produit à livrer à i
c_{ij}	$\mathbb{R}_+$	$i, j \in \mathcal{V}$ $i \neq j$	Coût pour aller de i à j

Table 4 – Donnés du problème VRP (3)

La formulation (3) ci-dessus est identique à celle donnée à l'énoncé du projet, au détail près que, pour éviter la confusion avec les demandes d_{it} des clients dans les problèmes LSP et PDI, la quantité de produit à livrer au client i est notée ici par b_i.

Bien que sa formulation soit plus simple que le problème LSP, avec moins de variables de décision et de contraintes, il s'agit d'un problème beaucoup plus difficile à résoudre en pratique. Il s'agit en effet d'un problème très similaire au problème du voyageur de commerce (TSP), mais, plutôt que de visiter les sommets, éléments de $\mathcal{V}_C$, en une seule tournée, dans le problème VRP on divise les visites en au plus m tournées afin de respecter la contrainte de ne charger qu'au plus une quantité Q de produit dans chaque véhicule. La contrainte (3f) est essentiellement la même

contrainte MTZ que pour le problème TSP qui vise à briser des sous-cycles ne passant pas par l'usine. Il s'agit d'une contrainte compacte, qui conduit à une formulation compacte, mais pas efficace en termes de temps de calcul.

Afin d'améliorer cette formulation, on peut utiliser d'autres contraintes qui viennent soit renforcer la formulation (3), soit remplacer la contrainte MTZ (3f). On utilisera dans ce projet la contrainte de coupes

$$\sum_{i\in S}\sum_{j\in\mathcal{V}\setminus S} x_{ij} \geq \left\lceil \frac{1}{Q}\sum_{i\in S} b_i \right\rceil \qquad S\subset\mathcal{V}_C, |S|\geq 1 \tag{4}$$

Cette contrainte indique que, pour tout sous-ensemble de clients $S \subset \mathcal{V}_C$ (avec au moins un client), la quantité d'arcs sortant de S doit être supérieure ou égale à la quantité minimale de véhicules pour livrer les produits à tous les clients de S, qui est le membre de droite de (4). L'ajout de cette contrainte rend la formulation (3) non-compacte.

Concernant les coûts c_{ij}, on utilisera toujours dans ce rapport l'hypothèse qu'ils sont symétriques, c'est-à-dire que $c_{ij} = c_{ji}$ pour tous i, j. Cependant, on ne supposera pas qu'ils satisfont l'inégalité triangulaire, on peut donc avoir $c_{ij} + c_{jk} < c_{ik}$ pour certaines valeurs de i, j, k. Cela est bien observé dans quelques instances de type A fournies avec le projet : par exemple, pour l'instance `A_014_ABS1_15_1.prp`, on a $c_{0,13} = 300$, $c_{13,8} = 61$, et $c_{0,8} = 362$. Cela veut dire que, pour visiter le client 8, il est moins cher de faire une escale intermédiaire chez le client 13, même si l'on n'a rien à lui livrer. Il s'agit d'une conséquence de l'arrondi de la distance euclidienne réalisé dans le calcul des distances pour les instances de type A.

2.3 Problème de production et distribution intégré

Le problème de production et distribution intégré prend en compte, dans un seul problème d'optimisation, les étapes d'optimiser la production et d'optimiser la distribution. On présente ici les deux formulations utilisées dans ce projet, issues de [2].

2.3.1 Première formulation

La première formulation correspond à (1)–(16) de [2] et représente le PDI par le PLNE compact suivant :

$$\min \sum_{t=0}^{\ell-1}\left\lceil up_t + fy_t + \sum_{i=0}^{n} h_i I_{it} + \sum_{i=0}^{n}\sum_{\substack{j=0\\ j\neq i}}^{n} c_{ij}x_{ijt}\right\rceil \tag{5a}$$

$$\text{s.c. } I_{0,t-1} + p_t = \sum_{i=1}^{n} q_{it} + I_{0,t} \qquad t\in\{0,\dots,\ell-1\} \tag{5b}$$

$$I_{i,t-1} + q_{it} = d_{it} + I_{i,t} \qquad i\in\{1,\dots,n\},\ t\in\{0,\dots,\ell-1\} \tag{5c}$$

$$p_t \leq M_t y_t \qquad t\in\{0,\dots,\ell-1\} \tag{5d}$$

$$I_{0,t} \leq L_0 \qquad t\in\{0,\dots,\ell-1\} \tag{5e}$$

$$I_{i,t-1} + q_{it} \leq L_i \qquad i\in\{1,\dots,n\},\ t\in\{0,\dots,\ell-1\} \tag{5f}$$

$$q_{it} \leq \widetilde{M}_{it} z_{it} \qquad i\in\{1,\dots,n\},\ t\in\{0,\dots,\ell-1\} \tag{5g}$$

$$\sum_{\substack{j=0\\ j\neq i}}^{n} x_{ijt} = z_{it} \qquad i\in\{1,\dots,n\},\ t\in\{0,\dots,\ell-1\} \tag{5h}$$

$$\sum_{\substack{j=0\\ j\neq i}}^{n} x_{jit} + \sum_{\substack{j=0\\ j\neq i}}^{n} x_{ijt} = 2z_{it} \qquad i\in\{0,\dots,n\},\ t\in\{0,\dots,\ell-1\} \tag{5i}$$

$$w_{it} - w_{jt} \geq q_{it} - (Q + \widetilde{M}_{it})(1 - x_{ijt}) \qquad i, j \in \{1, \ldots, n\}, i \neq j, t \in \{0, \ldots, \ell - 1\}, \tag{5j}$$

$$w_{it} \leq Q z_{it} \qquad i \in \{1, \ldots, n\}, t \in \{0, \ldots, \ell - 1\} \tag{5k}$$

$$x_{ijt} \in \{0, 1\} \qquad i, j \in \{0, \ldots, n\}, i \neq j, t \in \{0, \ldots, \ell - 1\}, \tag{5l}$$

$$I_{it} \in \mathbb{R}_+ \qquad i \in \{0, \ldots, n\}, t \in \{0, \ldots, \ell - 1\} \tag{5m}$$

$$q_{it} \in \mathbb{R}_+, w_{it} \in \mathbb{R}_+, z_{it} \in \{0, 1\} \qquad i \in \{1, \ldots, n\}, t \in \{0, \ldots, \ell - 1\} \tag{5n}$$

$$p_t \in \mathbb{R}_+, y_t \in \{0, 1\}, z_{0t} \in \{0, \ldots, m\} \qquad t \in \{0, \ldots, \ell - 1\} \tag{5o}$$

Les variables de décision et les données de ce problème sont décrits dans les Tables 5 et 6 et les valeurs de M_t et $\widetilde{M}_{it}$, comme dans le problème LSP, sont données par (2).

Variable	Domaine	Indices	Description
p_t	$\mathbb{R}_+$	$0 \leq t \leq \ell - 1$	Quantité produite par l'usine à l'instant t
y_t	$\{0, 1\}$	$0 \leq t \leq \ell - 1$	1 si production lancée à l'instant t, 0 sinon
I_{it}	$\mathbb{R}_+$	$0 \leq i \leq n$ $0 \leq t \leq \ell - 1$	Stock en i à la fin de la période t
q_{it}	$\mathbb{R}_+$	$1 \leq i \leq n$ $0 \leq t \leq \ell - 1$	Quantité à être livrée au revendeur i à la période t
z_{0t}	$\{0, \ldots, m\}$	$0 \leq t \leq \ell - 1$	Quantité de véhicules utilisés à l'instant t
z_{it}	$\{0, 1\}$	$1 \leq i \leq n$ $0 \leq t \leq \ell - 1$	1 si une livraison doit être faite à i à l'instant t, 0 sinon
x_{ijt}	$\{0, 1\}$	$0 \leq i \leq n$ $0 \leq j \leq n$ $i \neq j$ $0 \leq t \leq \ell - 1$	1 si un véhicule va de i à j à la période t, 0 sinon
w_{it}	$\mathbb{R}_+$	$1 \leq i \leq n$ $0 \leq t \leq \ell - 1$	Variables des contraintes MTZ pour briser les sous-cycles

Table 5 – Variables de décision du problème PDI (5)

La formulation (5) correspond à un mélange de la formulation du LSP avec un VRP par pas de temps, dans lequel on prend en compte le vrai coût de transport dans la fonction objectif, plutôt que les estimées SC_{ij} de (1a) dans le LSP, et dans lequel les demandes b_i du problème VSP sont en effet les valeurs q_{it} à être envoyées à chaque client à chaque pas de temps. Il faut aussi remarquer que la contrainte MTZ (3f) du problème VRP doit être adaptée, car, comme b_i est remplacé par q_{it}, qui est une variable de décision et non plus une donnée du problème, (3f) serait devenue non-linéaire si on avait tout simplement remplacé b_i par q_{it}.

Par rapport à la formulation dans [2], quelques légères modifications ont été faites pour rapprocher la formulation (5) de l'implémentation réalisée. Ainsi, la contrainte (10) de [2] est prise en compte dans le domaine de la variable z_{0t} dans (5o), et la borne inférieure sur w_{it} dans la contrainte (12) de [2] est prise en compte dans le domaine de w_{it} dans (5n).

Il y a cependant une modification importante qui a été faite par rapport à la formulation (1)–(16) de [2], concernant sa contrainte MTZ (11), qui est (5j) ici. La contrainte (11) de [2] y est formulée comme

$$w_{it} - w_{jt} \geq q_{it} - \tilde{M}_{it}(1 - x_{ijt}) \qquad i, j \in \{1, \ldots, n\}, i \neq j, t \in \{0, \ldots, \ell - 1\}$$

Cette contrainte sert à éviter des cycles qui ne passent pas par le nœud correspondant au fournisseur, qui est une situation impossible, ce qui se fait en imposant que, lorsqu'un arc entre deux clients i et j est pris dans une tournée à la période t (c'est-à-dire $x_{ijt} = 1$), alors la quantité w doit

Donnée	Domaine	Indices	Description
u	$\mathbb{R}_+$		Coût unitaire de production
f	$\mathbb{R}_+$		Coût fixe de production par période où la production est lancée
h_i	$\mathbb{R}_+$	$0 \le i \le n$	Coût unitaire de stockage
c_{ij}	$\mathbb{R}_+$	$0 \le i \le n$ $0 \le j \le n$ $i \ne j$	Coût fixe pour transporter des produits de i à j
$I_{i,-1}$	$\mathbb{R}_+$	$0 \le i \le n$	Quantité initiale en stock en i
$d_{i,t}$	$\mathbb{R}_+$	$1 \le i \le n$ $0 \le t \le \ell - 1$	Demande du client i à la période t
L_i	$\mathbb{R}_+$	$0 \le i \le n$	Capacité de stockage en i
m	$\mathbb{N}^*$		Quantité de véhicules disponibles
C	$\mathbb{R}_+$		Capacité de production
Q	$\mathbb{R}_+$		Capacité maximale d'un véhicule

Table 6 – Donnés du problème PDI (5)

être plus grande en i qu'en j d'au moins la quantité q_{it} à être livrée à i. Lorsque l'arc de i à j n'est pas pris (c'est-à-dire $x_{ijt} = 0$), alors cette contrainte ne doit pas nuire à la résolution et doit être automatiquement satisfaite à travers les autres contraintes de la formulation du problème.

Or, telle qu'elle est écrite, [2, équation (11)] n'est pas automatiquement satisfaite lorsque $x_{ijt} = 0$ et cela conduit à des problèmes dans l'implémentation de cette formulation. En effet, avant de modifier cette contrainte, nous avons utilisé l'algorithme approché pour trouver une solution approchée au problème dans l'instance du fichier `A_014_ABS1_15_1.prp`, avec une seule itération de la boucle LSP–VRP. Dans cette solution approchée (qui est bien réalisable), on vérifie que, à la deuxième période ($t = 1$), un seul camion fera une tournée, qui est $0 \to 1 \to 4 \to 8 \to 0$. En passant cette solution approchée à l'algorithme exacte (ce qui peut se faire en imposant des bornes inférieures et supérieures pour chaque variable de décision égales à la valeur trouvée dans la solution approchée), CPLEX indique des violations des contraintes correspondant à $(i = 1, j = 4, t = 1)$ et $(i = 4, j = 1, t = 1)$. En affichant les contraintes correspondantes, on trouve qu'elles sont :

$$-20 \le w_{1,1} - w_{4,1} - q_{1,1} - 20x_{1,4,1},$$

$$-14 \le w_{4,1} - w_{1,1} - q_{4,1} - 14x_{4,1,1}.$$

Dans la solution approchée, on a $x_{1,4,1} = 1$ et $x_{4,1,1} = 0$, de sorte que les contraintes se simplifient à

$$w_{1,1} - w_{4,1} \ge q_{1,1}, \tag{6}$$

$$w_{4,1} - w_{1,1} \ge q_{4,1} - 14.$$

La première contrainte impose bien que $w_{1,1}$ est supérieur à $w_{4,1}$ d'au moins la quantité $q_{1,1}$ livrée au client 1 à la période 1. Cependant, la solution approchée contient $q_{4,1} = 14$, et donc la deuxième contrainte devient $w_{4,1} \ge w_{1,1}$. Or, cela n'a pas de sens : comme l'arc $(4,1)$ n'a pas été pris, on ne devrait pas imposer une inégalité entre $w_{4,1}$ et $w_{1,1}$, qui en plus contredit l'inégalité (6) que l'on veut avoir.

Pour résoudre à ce problème, en s'inspirant de la contrainte MTZ (3f) pour le problème VRP, on a remplacé la contrainte (11) de [2] par (5j). Cette contrainte coïncide avec celle de [2] dans le cas $x_{ijt} = 1$. Dans le cas $x_{ijt} = 0$, elle devient

$$w_{it} - w_{jt} \ge -Q - (\tilde{M}_{it} - q_{it}). \tag{7}$$

Or, d'après les contraintes (5k) et (5n), on a $0 \leq w_{it} \leq Q$, donc la valeur minimale du membre de gauche de (7) est $-Q$. D'après la contrainte (5g), on a $q_{it} \leq \tilde{M}_{it}$, et donc la valeur maximale du membre de droite de (7) est $-Q$. Ainsi, les contraintes (5g), (5k) et (5n) impliquent que (7) est toujours satisfaite, comme voulu.

Il est à noter que la formulation de (11) de [2] n'est pas la même que celle des articles de Bard et Nananukul : la contrainte correspondante est (1e) dans [5] et (1j) dans [6], et, par rapport à (11) de [2], les deux utilisent M_t à la place de $\widetilde{M}_{it}$ (par ailleurs, la définition de $\widetilde{M}_{it}$ de [5] est aussi différente de celle de [2]). Malgré le fait que ces formulations augmentent la valeur de la constante devant le terme $(1 - x_{ijt})$, il n'est pas tout à fait clair qu'elles conduisent toujours à une contrainte automatiquement satisfaite lorsque $x_{ijt} = 0$, ce pourquoi nous avons décidé de garder la contrainte (5j), dont la preuve de satisfiabilité automatique dans le cas $x_{ijt} = 0$, donnée ci-dessus, est assez simple.

Puisque la formulation (5) contient une formulation de type VRP similaire à (3) à chaque pas de temps, elle présente les mêmes difficultés de résolution en pratique que la formulation (3), et il est ainsi intéressant de renforcer la formulation (5) par l'ajout d'autres contraintes. L'article [2] en propose trois, ses contraintes (17)–(19). La contrainte (17), appelée contrainte de capacité fractionnaire, s'écrit comme

$$\sum_{i \in S} \sum_{j \in \{0,\dots,n\} \setminus S} x_{ijt} \geq \frac{1}{Q} \sum_{i \in S} q_{it} \qquad S \subset \{1,\dots,n\}, |S| \geq 1, t \in \{0,\dots,\ell-1\} \tag{8}$$

Elle peut en effet remplacer les contraintes MTZ (5j) et (5k) de (5) ou être ajoutée à (5) comme renforcement de la formulation. C'est la deuxième approche qui est utilisée dans notre projet. Son ajout à (5) rend cette formulation non-compacte.

La contrainte (8) n'est en effet rien d'autre qu'une adaptation de la contrainte (4) de VRP au problème PDI, mais, puisque la donnée b_i de (4) est remplacée par la variable de décision q_{it} dans (8), l'arrondi vers le haut de (4) n'est plus possible, car il mènerait à une contrainte non-linéaire.

On remarque aussi que la formulation (8) donnée ici est légèrement différente que [2, équation (17)], qui fait les sommes sur $i \in \{0,\dots,n\} \setminus S$ et sur $j \in S$. Ces différences ne sont pas importantes : le membre de gauche de (8) compte le nombre d'arcs sortant de S à l'instant t, alors que celui de [2, équation (17)] compte le nombre d'arcs rentrant dans S à l'instant t, mais ces deux quantités sont en effet égales pour toute solution admissible grâce aux contraintes (5h) et (5i).

La contrainte (18) de [2] est équivalente à (17), et n'a donc pas été prise en compte dans ce projet. La contrainte (19) correspond à un léger renforcement de (17) et (18) et s'écrit comme

$$Q \sum_{i \in S} \sum_{\substack{j \in S \\ j \neq i}} x_{ijt} \leq \sum_{i \in S} (Q z_{it} - q_{it}) \qquad S \subset \{1,\dots,n\}, |S| \geq 2, t \in \{0,\dots,\ell-1\} \tag{9}$$

2.3.2 Deuxième formulation

La deuxième formulation pour le PDI de [2] correspond à ses équations (20)–(33), qui donnent le PLNE non-compact suivant :

$$\min \sum_{t=0}^{\ell-1} \left[u p_t + f y_t + \sum_{i=0}^{n} h_i I_{it} + \sum_{i=0}^{n} \sum_{\substack{j=0 \\ j \neq i}}^{n} c_{ij} \sum_{k=0}^{K-1} x_{ijkt} \right] \tag{10a}$$

$$\text{s.c. } I_{0,t-1} + p_t = \sum_{i=1}^{n} \sum_{k=0}^{K-1} q_{ikt} + I_{0,t} \qquad t \in \{0,\dots,\ell-1\} \tag{10b}$$

$$I_{i,t-1} + \sum_{k=0}^{K-1} q_{ikt} = d_{it} + I_{i,t} \qquad i \in \{1,\dots,n\}, t \in \{0,\dots,\ell-1\} \tag{10c}$$

$$p_t \leq M_t y_t \qquad t \in \{0,\dots,\ell-1\} \tag{10d}$$

$$I_{0,t} \leq L_0 \qquad t \in \{0,\dots,\ell-1\} \tag{10e}$$

$$I_{i,t-1} + \sum_{k=0}^{K-1} q_{ikt} \leq L_i \qquad i \in \{1,\dots,n\},\ t \in \{0,\dots,\ell-1\} \tag{10f}$$

$$q_{ikt} \leq \widetilde{M}_{it} z_{ikt} \qquad k \in \{0,\dots,K-1\},\ i \in \{1,\dots,n\},\ t \in \{0,\dots,\ell-1\} \tag{10g}$$

$$\sum_{k=0}^{K-1} z_{ikt} \leq 1 \qquad i \in \{1,\dots,n\},\ t \in \{0,\dots,\ell-1\} \tag{10h}$$

$$\sum_{\substack{j=0 \\ j\neq i}}^{n} x_{ijkt} = z_{ikt} \qquad k \in \{0,\dots,K-1\},\ i \in \{0,\dots,n\},\ t \in \{0,\dots,\ell-1\} \tag{10i}$$

$$\sum_{\substack{i=0 \\ i\neq j}}^{n} x_{ijkt} = z_{jkt} \qquad k \in \{0,\dots,K-1\},\ j \in \{0,\dots,n\},\ t \in \{0,\dots,\ell-1\} \tag{10j}$$

$$\sum_{i\in S} \sum_{\substack{j\in S \\ j\neq i}} x_{ijkt} \leq \sum_{i\in S} z_{ikt} - z_{ekt} \qquad S \subset \{1,\dots,n\},\ |S| \geq 2,\ k \in \{0,\dots,K-1\},\ t \in \{0,\dots,\ell-1\},\ e \in S \tag{10k}$$

$$\sum_{i=1}^{n} q_{ikt} \leq Q z_{0kt} \qquad k \in \{0,\dots,K-1\},\ t \in \{0,\dots,\ell-1\} \tag{10l}$$

$$z_{0kt} \geq z_{0,k+1,t} \qquad k \in \{0,\dots,K-2\},\ t \in \{0,\dots,\ell-1\} \tag{10m}$$

$$\sum_{i=1}^{n} q_{ikt} \geq \sum_{i=1}^{n} q_{i,k+1,t} \qquad k \in \{0,\dots,K-2\},\ t \in \{0,\dots,\ell-1\} \tag{10n}$$

$$x_{ijkt} \in \{0,1\} \qquad k \in \{0,\dots,K-1\},\ i,j \in \{0,\dots,n\},\ i \neq j,\ t \in \{0,\dots,\ell-1\} \tag{10o}$$

$$I_{it} \in \mathbb{R}_+ \qquad i \in \{0,\dots,n\},\ t \in \{0,\dots,\ell-1\} \tag{10p}$$

$$q_{ikt} \in \mathbb{R}_+ \qquad k \in \{0,\dots,K-1\},\ i \in \{0,\dots,n\},\ t \in \{0,\dots,\ell-1\} \tag{10q}$$

$$z_{ikt} \in \{0,1\} \qquad k \in \{0,\dots,K-1\},\ i \in \{0,\dots,n\},\ t \in \{0,\dots,\ell-1\} \tag{10r}$$

$$p_t \in \mathbb{R}_+,\ y_t \in \{0,1\} \qquad t \in \{0,\dots,\ell-1\} \tag{10s}$$

Les variables de décision de ce problème sont décrites dans la Table 7. Ses données sont identiques à celles du problème (5), déjà présentées dans la Table 6. Les valeurs de M_t et $\widetilde{M}_{it}$ sont, comme précédemment, données par (2). La valeur de K a été choisie comme

$$K = \min(m, n) \tag{11}$$

Dans cette formulation, les tournées sont séparées par véhicule, dont l'indice k apparaît dans les variables de décision dépendant d'un véhicule. Le nombre maximal de véhicules disponibles est m mais, comme chaque client doit être livré par un seul véhicule (à cause de la contrainte (10h)), on n'aura jamais plus de n véhicules utilisés, ce qui explique le choix de K comme le minimum entre m et n. Cela permet aussi de traiter des instances fournies avec une valeur de m trop élevée mais n plus petit sans avoir à rajouter une grande quantité de variables de décision.

La contrainte (10k) utilisée ici correspond à la contrainte (33) de [2]. Cette référence donne d'abord une autre formulation avec une autre contrainte, similaire à (10k) mais avec le membre de droite remplacé par $|S|-1$, et explique ensuite que la contrainte (10k) est en effet plus efficace

Variable	Domaine	Indices	Description
p_t	$\mathbb{R}_+$	$0 \leq t \leq \ell - 1$	Quantité produite par l'usine à l'instant t
y_t	$\{0,1\}$	$0 \leq t \leq \ell - 1$	1 si production lancée à l'instant t, 0 sinon
I_{it}	$\mathbb{R}_+$	$0 \leq i \leq n$ $0 \leq t \leq \ell - 1$	Stock en i à la fin de la période t
q_{ikt}	$\mathbb{R}_+$	$0 \leq k \leq K - 1$ $1 \leq i \leq n$ $0 \leq t \leq \ell - 1$	Quantité à être livrée au revendeur i à la période t par le véhicule k
z_{ikt}	$\{0,1\}$	$0 \leq k \leq K - 1$ $0 \leq i \leq n$ $0 \leq t \leq \ell - 1$	1 si une livraison doit être faite à i (ou partir de l'usine si $i = 0$) à l'instant t par le véhicule k, 0 sinon
x_{ijkt}	$\{0,1\}$	$0 \leq k \leq K - 1$ $0 \leq i \leq n$ $0 \leq j \leq n$ $i \neq j$ $0 \leq t \leq \ell - 1$	1 si le véhicule k va de i à j à la période t, 0 sinon

Table 7 – Variables de décision du problème PDI (10)

pour la résolution du PDI par un algorithme de *branch and cut*, ce pourquoi nous avons décidé de l'utiliser.

La formulation (10) présente quelques différences par rapport à (20)–(33) de [2]. D'abord, au lieu de (10i) et (10j), [2] utilise une seule contrainte, (28), correspondant à la somme des deux contraintes utilisées ici. Or, dans la formulation de [2], les seules contraintes portant sur x_{ijkt} sont (28) et (33) (en plus de la contrainte de domaine), qui ne suffisent pas pour exclure quelques solutions qui doivent être non-réalisables. Par exemple, considérons une situation avec $n = 2$, $m = 1$ et un instant de temps t dans lequel les deux clients 1 et 2 doivent être visités par l'unique véhicule $k = 0$. La situation de la Figure 1 illustre un choix de variable de décision qui satisfait les contraintes de [2], qui correspond à prendre $z_{0kt} = z_{1kt} = z_{2kt} = 1$, $x_{01kt} = x_{12kt} = x_{02kt} = 1$, et toutes les autres variables x_{ijkt} égales à 0. La contrainte (28) dit ainsi que le degré de chaque sommet doit valoir 2, ce qui est bien satisfait sur la Figure 1, et la contrainte (33) ne peut s'appliquer qu'à $S = \{1,2\}$ et dit qu'il ne peut avoir qu'au plus un arc interne à cet ensemble, ce qui est aussi bien le cas. Cet exemple que nous présentons ici est en effet issu de nos tests de notre première implémentation de la formulation (20)–(33) de [2], dans lesquels nous avons pu observer que ce type de solution était retourné par CPLEX.

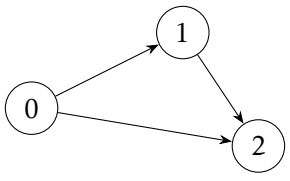

Figure 1 – Exemple de solution non-réalisable satisfaisant les contraintes (28) et (33) de [2]

Le problème avec la contrainte (28) est qu'elle ne sépare pas le degré sortant et le degré rentrant dans chaque sommet. Remarquons que, dans la formulation (1)–(16) de [2], cette séparation est faite avec les contraintes (8) et (9) ((5h) et (5i) dans la formulation (5) ci-dessus), qui portent sur le degré sortant et le degré total. Afin de corriger ce problème de (20)–(33), nous avons choisi d'utiliser les contraintes (10i) et (10j) portant sur les degrés sortant et rentrant de chaque sommet.

Une autre différence par rapport à la formulation (20)–(33) de [2] est l'ajout des contraintes (10m) et (10n), que nous avons repris de [1] (où elles sont nommées *symmetry breaking constraints*.

Sans ces contraintes, un des principaux problèmes de la formulation (10) est un grand nombre de symétries. En effet, étant donnée une solution, si l'on permute les véhicules utilisés pour faire chaque tournée, on obtient une autre solution qui représente la même situation (avec juste l'indexation des véhicules modifiée) et donc avec la même valeur de la fonction objectif. Cette symétrie peut contribuer à une augmentation importante du temps de résolution. Afin d'éviter ces situations symétriques, la contrainte (10m) impose qu'un véhicule d'indice $k+1$ ne peut être utilisé que si le véhicule d'indice k a été utilisé, ce qui implique que les véhicules utilisés seront forcément ceux de plus petits indices. Cela empêche plusieurs situations symétriques mais permet encore qu'il y ait des permutations parmi les véhicules utilisés, un phénomène empêché par la contrainte (10n), qui ordonne les véhicules par capacité utilisée décroissante.

3 Implémentation

Cette section décrit les implémentations réalisées dans le cadre de ce projet afin de résoudre le PDI de façon approché et exacte. Toutes les implémentations ont été réalisées en `C++`, en utilisant `CPLEX` pour la résolution de programmes linéaires.

Le projet est décomposé en huit classes principales. Deux classes, `PRP` (fournie avec l'énoncé) et `Solution`, permettent respectivement de représenter des instances du PDI et de stocker leurs solutions, et sont décrites plus en détail dans la Section 3.1. Trois classes, `SolApprocheeBase`, `SolApprocheeCoupe` et `SolApprocheeHeuristique`, portent sur la résolution approchée du PDI en utilisant les problèmes LSP et VRP, et sont présentées dans la Section 3.2. Les trois dernières classes, `SolExacteBase`, `SolExacteCoupe` et `SolExacteV2`, s'intéressent à des méthodes exactes de résolution du PDI basées sur les formulations (5) et (10) et sont décrites plus en détail dans la Section 3.3.

3.1 Classes de base

La classe `PRP` a été fournie avec l'énoncé et sert à représenter une instance du problème PDI ; elle stocke ainsi les données de la Table 6 ou des valeurs nécessaires pour calculer ces données (comme les positions de l'usine et des clients, utilisées pour calculer les coûts c_{ij}), et contient également une méthode de lecture à partir d'un fichier `.prp` et des méthodes d'écriture d'une instance à l'écran ou dans un fichier. Par rapport au fichier fourni avec l'énoncé, la seule modification apportée à cette classe a été l'implémentation d'une méthode `cost` qui prend en argument deux indices i et j et calcule leur distance avec la bonne formule, suivant qu'il s'agit d'une instance de type A ou B, comme décrit à l'énoncé.

La classe `Solution` permet de stocker une solution du PDI, sous la forme de vecteurs contenant les variables de décision p_t, y_t, I_{it}, q_{it} et z_{it}. Cette classe stocke également les informations des variables x_{ijt} mais sous une forme différente. En effet, les variables x_{ijt} peuvent être vues comme la donnée, pour chaque période t, d'un graphe sur les sommets $\{0,\dots,n\}$ codé comme une matrice d'adjacence. Afin de limiter la taille de mémoire nécessaire pour stocker ce graphe, nous avons décidé de l'implémenter plutôt comme une liste de successeurs représentant le graphe à chaque période t. Ainsi, la variable `x` stockée dans la classe `Solution` est un vecteur indexé par la période `t` et telle que, pour tout $i \in \{0,\dots,n\}$, `x[t][i]` est un vecteur (potentiellement vide) contenant les successeurs de i dans le graphe à l'instant t, c'est-à-dire les valeurs de j pour lesquelles $x_{ijt} = 1$. D'après les contraintes (5h) et (5n), cette liste contient au plus un élément, sauf pour $i = 0$, auquel cas, par les contraintes (5h) et (5o), elle contient au plus m éléments.

Bien que la structure de la classe `Solution` ait été codée ayant pour base la formulation (5), elle sert aussi à représenter des solutions de la formulation (10). Nous faisons l'hypothèse dans ce projet que les véhicules utilisés sont tous identiques, et donc les données stockées dans une instance de `Solution`, qui ne prennent pas en compte les indices de véhicule k, sont suffisantes pour reconstruire toutes les variables de décision de (10).

En effet, les variables de décision p_t, y_t et I_{it} sont les mêmes dans les formulations (5) et (10), les

seules différences se trouvent dans les variables q, z et x, qui ont l'indice k du véhicule en plus. La lecture de la variable `x` d'une instance de `Solution` permet de retrouver la quantité de véhicules utilisés dans la solution à la période t : il s'agit de la taille du vecteur `x[t][0]`, qui contient les successeurs de l'usine dans le graphe et dont la taille est donc égale au nombre de tournées. Si l'on note par κ_t cette taille, on a donc immédiatement que $q_{ikt} = z_{ikt} = x_{ijkt} = 0$ pour tout $k \geq \kappa_t$. Pour $t \in \{0,\dots,\ell-1\}$ et $k \in \{0,\dots,\kappa_t - 1\}$, les valeurs de x_{ijkt} représentent le graphe de la k-ième tournée à l'instant t, qui est donc un cycle partant de 0. Pour retrouver q_{ikt} et z_{ikt} pour $i \in \{1,\dots,n\}$, $t \in \{0,\dots,\ell-1\}$ et $k \in \{0,\dots,\kappa_t\}$, il suffit de faire $q_{ikt} =$ `q[i][t]` si i appartient à la k-ième tournée à l'instant t, et 0 sinon. De même, $z_{ikt} =$ `z[i][t]` si i appartient à la k-ième tournée à l'instant t, et 0 sinon. Finalement, $z_{0kt} = 1$ pour tout $t \in \{0,\dots,\ell-1\}$ et toute tournée faite, c'est-à-dire tout $k \in \{0,\dots,\kappa_t - 1\}$.

La procédure décrite dans le paragraphe précédent permet de retrouver les variables de décision de (10) à partir des valeurs stockées dans une instance de `Solution`. Le problème réciproque, c'est-à-dire retrouver les valeurs de l'instance de `Solution` à partir des valeurs des variables de décision de (10), est beaucoup plus simple, puisqu'il suffit de sommer sur k.

Outre le stockage de solutions du PDI, la classe `Solution` contient aussi des méthodes d'affichage en mode texte et de calcul de la valeur de la fonction objectif (5a) (ou, de façon équivalente, (10a)).

3.2 Méthodes approchées

Les classes `SolApprocheeBase`, `SolApprocheeCoupe` et `SolApprocheeHeuristique` implémentent des méthodes approchées de résolution du PDI. Elles suivent toutes le même principe, basée sur les pas suivants :

1. *Initialisation :* On estime d'abord les coûts de visite des clients par $SC_{it} = c_{0i} + c_{i0}$ pour tout $i \in \{1,\dots,n\}$ et $t \in \{0,\dots,\ell-1\}$.
2. *Résolution du LSP :* On résout le problème LSP avec les coûts estimés SC_{it}, en utilisant `CPLEX` pour résoudre la formulation par PLNE (1). Les valeurs des variables de décision p_t, y_t, I_{it}, q_{it} et z_{it} à l'optimum sont stockées dans une instance de la classe `Solution`.
3. *Résolution de ℓ problèmes VRP :* Pour chaque $t \in \{0,\dots,\ell-1\}$, on résout le problème VRP avec $\mathcal{V}_C = \{i \in \{1,\dots,n\} \mid z_{it} = 1\}$ (c'est-à-dire, en ne prenant en compte que les clients visités à la période t) et $b_i = q_{it}$ pour tout $i \in \mathcal{V}_C$. Les valeurs des variables de décision x_{ij} sont stockées dans une instance de la classe `Solution`, en faisant une conversion entre la représentation par matrice d'adjacence et celle par liste de successeurs, comme décrit à la Section 3.1.
4. *Mise à jour des coûts estimés et boucle :* On calcule d'abord la valeur de la fonction objectif du PDI (5a) dans la solution trouvée dans les étapes 2 et 3 et on sauvegarde la meilleure solution déjà trouvée. On utilise ensuite les valeurs dans l'instance `Solution` pour mettre à jour les estimées de SC_{it} de la façon suivante : si i est visité à l'instant t, $SC_{it} = c_{p_i i} + c_{i s_i} - c_{p_i s_i}$, où p_i et s_i dénotent le prédécesseur et le successeur de i dans la tournée. Si i n'est pas visité à l'instant t, alors SC_{it} est le plus petit coût d'insertion de i dans une tournée : on calcule SC_{it} comme le minimum entre le coût de création d'une nouvelle tournée pour i, $c_{0i} + c_{i0}$, et le coût d'insertion de i dans une tournée, qui vaut $c_{ui} + c_{iv} - c_{uv}$ pour tout arc (u,v) de chaque tournée. Après cette mise à jour des SC_{it}, on revient à l'étape 2 tant qu'un critère d'arrêt choisi n'est pas satisfait.

Dans nos implémentations, le critère d'arrêt a été choisi comme un nombre fixe d'itérations. Les trois implémentations réalisées ne diffèrent que par la méthode de résolution des problèmes VRP à l'étape 3 et sont décrites dans la suite.

3.2.1 Première heuristique approchée

Dans la première heuristique approchée, implémentée dans la classe `SolApprocheeBase`, on résout chaque problème VRP de l'étape 3 par une implémentation sur `CPLEX` du PLNE (3).

Cette implémentation naïve marche très bien pour les instances de taille petite (les instances de type A de taille $n = 14$ et quelques instances de type A de taille $n = 50$), mais elle est trop lente pour traiter des instances plus grandes, la plupart du temps de calcul étant passée à résoudre des problèmes VRP. Cela motive la recherche d'autres méthodes permettant de résoudre le problème VRP plus rapidement.

3.2.2 Renforcement de la formulation du VRP

La deuxième méthode approchée, implémentée dans la classe `SolApprocheeCoupe`, utilise la contrainte de coupe (4) pour renforcer la formulation (3). La quantité de ces contraintes étant exponentielles, il n'est pas réaliste de les ajouter toutes à `CPLEX`. Nous avons alors crée une classe héritant de la classe `UserCutCallbackI` de `CPLEX` afin d'implémenter la contrainte (4). Après avoir ajouté cette classe à la formulation de notre PLNE, `CPLEX` appelle sa fonction `main` à chaque fois qu'une solution réalisable (de la relaxation continue, donc pas forcément entière) est trouvée. La fonction `main` de notre classe doit donc implémenter un algorithme de séparation pour la contrainte (4) afin de déterminer si elle est satisfaite pour tout ensemble $S \subset \mathcal{V}_C$ ou, dans le cas contraire, d'ajouter au moins une contrainte violée.

Il y a un compromis à trouver pour l'implémentation de l'algorithme de séparation : si son exécution prend trop de temps, comme cet algorithme est appelé un grand nombre de fois pas `CPLEX`, il peut ralentir beaucoup la résolution du PLNE. Or, dans notre cas, le problème de séparation de (4) est NP-difficile, et donc il est préférable d'implémenter un algorithme approché de séparation avec une exécution plus rapide, cherchant de façon heuristique un ensemble S pour lequel la contrainte (4) est violée, mais sans garantie d'être capable d'en trouver un.

Nous avons implémenté l'heuristique gloutonne décrite en [3, 4] pour la résolution du problème de séparation de (4). Cette heuristique part d'un ensemble $S \subset \mathcal{V}_C$ choisi de façon aléatoire. Tant que $S \neq \mathcal{V}_C$, on cherche parmi tous les éléments $v \in \mathcal{V}_C \setminus S$ celui qui maximise $\sum_{i \in S} x_{iv}$ et on le rajoute à S. À chaque itération, on calcule la différence des membres de gauche et de droite de (4) pour voir si la contrainte est violée. À la fin de l'algorithme, on rajoute à la formulation la contrainte (4) la plus violée, si il y en a une. Un paramètre passé en argument permet de répéter cette procédure plusieurs fois en partant d'ensembles S générés de façon aléatoire, mais au maximum une seule contrainte est ajoutée à la fin. Cependant, dans quelques tests, nous avons trouvé que tirer plusieurs ensembles S aléatoires n'améliorait pas significativement la qualité de la résolution, et la version finale de notre code ne fait qu'un seul tirage aléatoire de S par appel à la fonction.

L'ajout de la contrainte de renforcement (4) permet le calcul de solutions approchées d'instances plus grandes que celles traitées par la méthode présentée en Section 3.2.1. Néanmoins, le problème reste trop long à résoudre pour des instances grandes, ce qui motive la recherche d'autres heuristiques pour trouver de bonnes solutions réalisables au problème VRP, et donc au problème PDI.

Un point faible de notre approche est que, bien que l'on cherche une solution approchée au PDI, on essaie de résoudre le problème VRP de façon exacte, ce qui constitue une étape trop longue de l'heuristique approchée. Or, d'après nos essais, une bonne solution réalisable au VRP, même si elle n'est pas optimale, peut permettre déjà de trouver une solution réalisable au PDI de bonne qualité. Ainsi, notre classe `SolApprocheeCoupe` peut arrêter la résolution du problème VRP par le PLNE (3) après un certain temps passé en argument, en gardant la meilleure solution entière trouvée jusqu'alors. Puisque typiquement la résolution du problème VRP par `CPLEX` trouve une solution entière assez rapidement, et que pendant les premières itérations `CPLEX` arrive typiquement à bien améliorer la solution, donner en argument un temps de l'ordre de quelques dizaines de secondes permet en général d'arrêter le problème VRP avec une bonne solution réalisable entière et d'obtenir ainsi des solutions réalisables de bonne qualité pour le PDI, tout en gardant un temps de calcul plus raisonnable et permettant ainsi de traiter des instances de taille plus importante.

3.2.3 Heuristique itérative pour le VRP

L'implémentation de la contrainte de coupes (4) en renforcement de la formulation (3) et l'utilisation d'un temps de calcul maximal pour le problème VRP ont permis d'obtenir dans la Section 3.2.2 un algorithme qui s'exécute de façon plus rapide que celui de la Section 3.2.1 mais qui donne des résultats tout aussi bons. Cependant, son temps de calcul n'est pas suffisamment rapide pour traiter des instances de très grande taille, comme les instances de type B avec $n = 200$.

Afin d'être capable de traiter ce type d'instance, nous avons abandonné la formulation (3) par PLNE du problème VRP et cherché à la place une heuristique pour trouver des solutions réalisables de bonne qualité pour le VRP. Cette heuristique, utilisée dans la classe `SolApprochee Heuristique`, se base sur l'heuristique proposée à l'énoncé du projet et consiste dans les étapes suivantes, réalisées pour chaque pas de temps t :

1. *Construction heuristique des tournées :* On construit l'ensemble $\mathcal{V}_C$ des clients à visiter. On commence avec une tournée partant de l'usine, on cherche le client le plus proche encore à visiter, on le retire de $\mathcal{V}_C$ et on le rajoute dans la tournée. Ensuite, itérativement, on rajoute à la tournée le client le plus proche du dernier client ajoutée à la tournée dont la demande q_{it} rentre encore dans le véhicule, en le retirant aussi de $\mathcal{V}_C$. Lorsqu'il qu'il n'y a plus de clients à rajouter dans cette étape (soit car $\mathcal{V}_C = \emptyset$, soit car il n'y a plus d'espace pour aucun client), on ferme la tournée en retournant le véhicule à l'usine. Tant que $\mathcal{V}_C$ n'est pas vide, cette procédure est itérée en construisant de nouvelles tournées.

   1.1. *Prise en compte du nombre de véhicules :* La construction précédente ne prend pas en compte le nombre de véhicules disponibles et pourrait construire plus de tournées que le nombre m de véhicules disponibles. Si cela est le cas, on construit une nouvelle solution de la façon suivante. On résout d'abord le problème de repartir les quantités à envoyer aux clients q_{it} en au plus m tournées, chacune avec une charge totale d'au plus Q. Cela est fait en résolvant avec `CPLEX` le PLNE suivant :

$$\min \sum_{k=0}^{K-1} z_k \tag{12a}$$

$$\text{s.c.} \sum_{k=0}^{K-1} x_{ik} = 1 \qquad i \in \mathcal{V}_C \tag{12b}$$

$$\sum_{i \in \mathcal{V}_C} q_{it} x_{ik} \leq Q z_k \qquad k \in \{0, \dots, K-1\} \tag{12c}$$

$$z_k \in \{0,1\} \qquad k \in \{0, \dots, K-1\} \tag{12d}$$

$$x_{ik} \in \{0,1\} \qquad i \in \mathcal{V}_C, k \in \{0, \dots, K-1\} \tag{12e}$$

   Dans cette formulation, comme dans (11), on a $K = \min(m, n)$. En plus, $z_k = 1$ si et seulement si le véhicule k réalise une tournée et $x_{ik} = 1$ si et seulement si le client i est desservi par le véhicule k. Une fois la répartition en tournées réalisée, on choisit de parcourir les clients des tournées dans un ordre arbitraire.

   1.2. *Traitement du cas infaisable :* Si le PLNE (12) est infaisable, cela veut dire que le LSP a déterminé un envoi de quantité de produits impossible à cet instant de temps. Dans ce cas, on arrête la résolution heuristique du problème VRP et on résout la version modifiée suivante du problème LRP, qui prend en compte les capacités de chacun des véhicules :

$$\min \sum_{t=0}^{\ell-1} \left[u p_t + f y_t + \sum_{i=0}^{n} h_i I_{it} + \sum_{i=1}^{n} SC_{it} \sum_{k=0}^{K-1} z_{ikt} \right] \tag{13a}$$

$$\text{s.c. } I_{0,t-1} + p_t = \sum_{i=1}^{n} \sum_{k=0}^{K-1} q_{ikt} + I_{0,t} \qquad t \in \{0, \dots, \ell-1\} \tag{13b}$$

$$I_{i,t-1} + \sum_{k=0}^{K-1} q_{ikt} = d_{it} + I_{i,t} \qquad i \in \{1,\dots,n\},\ t \in \{0,\dots,\ell-1\} \qquad (13c)$$

$$p_t \leq M_t y_t \qquad t \in \{0,\dots,\ell-1\} \qquad (13d)$$

$$I_{0,t} \leq L_0 \qquad t \in \{0,\dots,\ell-1\} \qquad (13e)$$

$$I_{i,t-1} + \sum_{k=0}^{K-1} q_{ikt} \leq L_i \qquad i \in \{1,\dots,n\},\ t \in \{0,\dots,\ell-1\} \qquad (13f)$$

$$q_{ikt} \leq \widetilde{M}_{it} z_{ikt} \qquad i \in \{1,\dots,n\},\ t \in \{0,\dots,\ell-1\},\ k \in \{0,\dots,K-1\} \qquad (13g)$$

$$\sum_{k=0}^{K-1} z_{ikt} \leq 1 \qquad i \in \{1,\dots,n\},\ t \in \{0,\dots,\ell-1\} \qquad (13h)$$

$$\sum_{i=1}^{n} q_{ikt} \leq Q \qquad t \in \{0,\dots,\ell-1\},\ k \in \{0,\dots,K-1\} \qquad (13i)$$

$$I_{it} \in \mathbb{R}_+ \qquad i \in \{0,\dots,n\},\ t \in \{0,\dots,\ell-1\} \qquad (13j)$$

$$q_{ikt} \in \mathbb{R}_+,\ z_{ikt} \in \{0,1\} \qquad i \in \{1,\dots,n\},\ t \in \{0,\dots,\ell-1\},\ k \in \{0,\dots,K-1\} \qquad (13k)$$

$$p_t \in \mathbb{R}_+,\ y_t \in \{0,1\} \qquad t \in \{0,\dots,\ell-1\} \qquad (13l)$$

Dans ce problème aussi, K est donné par (11). Une fois ce problème résolu, on recommence l'heuristique itérative pour le VRP de l'étape 1 à $t = 0$. Remarquons que le problème (13) n'est infaisable que si le PDI en soi est infaisable.

2. *Amélioration heuristique de la solution :* À partir de la solution trouvée à l'initialisation des tournées, on l'améliore de façon heuristique. On fixe un nombre d'itérations et, à chaque itération, on choisit de façon aléatoire de réaliser l'une des deux heuristiques suivantes :

   2.1. *Amélioration d'une tournée :* On choisit une tournée aléatoire et on essaie d'améliorer l'ordre de parcours des clients en faisant une recherche locale avec des voisinages de type 2-TSP. Plus précisément, on parcourt toutes les paires d'arcs non-adjacents de la tournée et, pour chaque paire, on détermine s'il est possible d'améliorer le coût total de parcours de la tournée en modifiant ces arcs. On choisit de modifier la paire d'arcs permettant d'améliorer le plus le coût de parcours, si une telle paire existe.

   2.2. *Modification de la tournée d'un client :* On choisit un client aléatoire et on parcourt toutes les autres tournées. Pour chaque autre tournée et chaque position possible dans cette tournée, on calcule quelle serait la différence de coût si on retire le client de sa tournée actuelle et on le met dans cette position. Lorsque le nombre de tournées est strictement plus petit que K, on évalue aussi le coût de créer une nouvelle tournée uniquement pour ce client. On choisit alors de faire le changement qui permet de minimiser les coûts, s'il y en a un. Si le fait de retirer un client d'une tournée la rend vide, on la supprime.

On a observé en pratique que l'étape 1 donne très souvent des solutions avec au plus K tournées, et donc la résolution du PLNE (12) dans l'étape 1.1 n'est exécutée que dans de très rares cas. D'ailleurs, même dans le cas où elle est nécessaire, le PLNE (12) est un problème beaucoup plus simple que le PLNE (3) pour le VRP, et sa résolution est typiquement beaucoup plus rapide, ce que nous avons pu constater en pratique.

Traiter le cas où (12) est infaisable est important car il peut y arriver que la résolution du problème LSP (1) donne un envoi de produit impossible. La contrainte (1h) que l'on a rajouté à (1) essaie d'empêcher ce type de problème en imposant la quantité maximale de produits à envoyer, mais elle n'est pas suffisante pour garantir que des situations impossibles ne se produisent. Par

exemple, il est tout à fait possible que, dans une instance avec $n = 6$, $m = 5$ et $Q = 10$ la solution du LSP donne à un certain instant de temps t les valeurs $q_{1t} = q_{2t} = q_{3t} = q_{4t} = q_{5t} = 9$ et $q_{6t} = 5$. Cela satisfait bien la contrainte (1h) mais il est impossible de faire cet envoi avec uniquement 5 véhicules à cause de la contrainte de ne passer qu'au maximum une seule fois par chaque client à chaque pas de temps.

Le problème (13) qui est résolu dans le cas où (12) est infaisable permet d'éviter ce problème. Inspiré par la formulation (10), les variables z_{it} et q_{it} de (1) sont décomposées selon les véhicules, en leur ajoutant un indice k. Elles s'interprètent donc comme les variables correspondantes dans la Table 7. La contrainte (13h) impose que chaque client est desservi par au plus un véhicule, alors que (13i) indique que chaque véhicule a une charge maximale de Q. Cette formulation ayant plus de variables que (1), sa résolution prend typiquement considérablement plus de temps. Cela explique notre choix de, dans notre algorithme, résoudre d'abord le problème LSP (1), ensuite faire l'étape heuristique, et ne résoudre (13) que si nécessaire, ce qui est très rare.

Strictement parlant, ce garde-fou de résoudre (13) lorsque (12) est infaisable devrait aussi être présent dans les algorithmes approchés des Sections 3.2.1 et 3.2.2, dans les cas où le problème VRP (3) est infaisable. Cependant, ce type de problème n'a été retrouvé qu'en traitant des instances de grande taille de type B avec peu de véhicules. À cause de la difficulté du problème VRP sur ces instances, le temps de résolution par les algorithmes `SolApprocheeBase` et `SolApprocheeCoupe` des Sections 3.2.1 et 3.2.2 est trop long pour que l'on puisse tester ces heuristiques. Par manque de temps, ce garde-fou n'a donc pas été implémenté sur `SolApprocheeBase` et `SolApprocheeCoupe`.

Globalement, cette méthode heuristique s'exécute sensiblement plus vite que les méthodes des Sections 3.2.1 et 3.2.2, même avec un grand nombre d'itérations dans l'étape 2. Elle retourne assez souvent le même résultat que ces deux autres méthodes, même s'il lui arrive de donner des résultats moins bons dans une proportion des cas. Les résultats quantitatifs sont décrits de façon détaillée dans la Section 5.

3.3 Méthodes exactes

Les trois classes `SolExacteBase`, `SolExacteCoupe` et `SolExacteV2` implémentent des méthodes exactes de résolution du PDI. Les deux premières, décrites respectivement dans les Sections 3.3.1 et 3.3.2, se basent sur la formulation (5) alors que la troisième, décrite dans la Section 3.3.3, se base sur la formulation (10).

3.3.1 Première résolution exacte

Nous avons d'abord implémenté une résolution à travers `CPLEX` de la formulation compacte (5) du PDI dans la classe `SolExacteBase`. Notre implémentation permet de prendre en argument un objet de type `Solution`, qui est passé à `CPLEX` pour être utilisé comme une première solution entière. Cela permet ainsi d'initialiser l'algorithme de résolution du PLNE déjà avec une solution entière, qui peut être calculée au préalable par une méthode approchée.

Comme attendu, cette formulation compacte du PLNE s'avère être très lente, même pour résoudre les instances les plus simples de type A et taille $n = 14$. Un des problèmes est la grande quantité de branches à explorer par `CPLEX`, ce qui résulte dans un arbre à stocker de très grande taille, qui peut vite atteindre plusieurs gigaoctets et rend l'utilisation de cet algorithme pour des instances de taille modérée infaisable.

À fin de tester cette implémentation, ainsi que celles décrites dans la suite, nous avons crée dans le fichier `Instance_0.prp` une instance simple, de type A, taille $n = 4$ et avec un horizon de temps $\ell = 3$, sur laquelle l'implémentation s'exécute en quelques secondes et pour laquelle il est possible de vérifier assez facilement manuellement que la solution retournée est bien réalisable et semble optimale. Cela nous a permis de vérifier la validité de notre algorithme.

Nous avons aussi permis à l'algorithme de prendre en argument la tolérance à laquelle il peut s'arrêter, qui correspond à l'écart relatif entre la meilleure solution entière et la meilleure borne inférieure trouvées par l'algorithme à chaque itération. La valeur par défaut de ce paramètre en

CPLEX est de 10^{-4}, mais en passant des valeurs plus grandes, comme 0,01 ou 0,02, l'algorithme se termine en moyenne assez rapidement pour des instances de type A et taille $n = 14$ et on peut bien vérifier que la solution retournée est réalisable.

3.3.2 Renforcement de la formulation

Afin d'améliorer la résolution exacte de la Section 3.3.1, nous avons crée une nouvelle classe, `SolExacteCoupe`, dans laquelle on a ajouté à (5) des contraintes renforçant la formulation, à l'image de ce qui a été décrit dans la Section 3.2.2 pour le problème VRP. Nous avons choisi d'utiliser les contraintes 8 et 9, qui sont en nombre exponentiel et doivent donc être insérées dans CPLEX à l'aide d'un algorithme de séparation implémenté dans une classe héritant de `UserCutCallbackI`, à l'image de la Section 3.2.2. Comme dans la Section 3.2.2, nous avons utilisé des heuristiques pour choisir quelles contraintes ajouter, plutôt qu'une méthode exacte dont la résolution serait trop lente. Quatre renforcements ont été implémentés :

1. *Heuristique gloutonne pour* (8) *:* Dans ce renforcement, on applique à la contrainte (8) la même heuristique gloutonne de [3, 4] utilisée dans la Section 3.2.2 pour la contrainte (4) du problème VRP.
2. *Heuristique gloutonne pour* (9) *:* Ce renforcement utilise la même heuristique gloutonne que le précédent, mais appliquée cette fois-ci à la contrainte (9).
3. *Première recherche tabou pour* (9) *:* L'heuristique utilisée dans ce renforcement pour la contrainte (9) est la recherche tabou telle que décrite dans [3, 4]. Cette heuristique part d'un ensemble de clients S aléatoire et le modifie en alternant des phases d'expansion et d'échange. Les phases d'expansion ne font qu'ajouter des sommets à S à l'aide d'une heuristique, et la phase d'échange détermine s'il y a des ajouts ou des suppressions d'éléments intéressantes à faire. Dans la phase d'échange, les éléments ajoutés ou supprimés rentrent dans une liste tabou et l'opération inverse ne peut être réalisée qu'après un certain nombre d'itérations. À la fin de la phase d'échange, on revient à la phase d'expansion, et on itère un nombre de fois égal à la quantité totale de véhicules disponibles. À chaque fois que l'ensemble S est modifié, on calcule de combien la contrainte (9) est violée, et à la fin on ajoute la contrainte la plus violée, si il y en a une.
4. *Deuxième recherche tabou pour* (9) *:* Nous avons aussi implémentée une autre recherche tabou pour (9), plus simple que celle de [3, 4], afin d'essayer de réduire le temps d'exécution de la recherche. Notre objectif est de chercher S minimisant la quantité

$$\sum_{i \in S} (Qz_{it} - q_{it}) - Q \sum_{i \in S} \sum_{\substack{j \in S \\ j \neq i}} x_{ijt} \tag{14}$$

   qui est la différence entre les membres de droite et de gauche de (9). Dans notre recherche tabou, on commence aussi par un ensemble S aléatoire de taille aléatoire. À chaque itération, on parcourt tous les sommets. Pour les sommets v qui sont dans S, on calcule la valeur de (14) si v était retiré de S et, pour les sommets v qui ne sont pas dans S, on calcule la valeur de (14) si v était ajouté à S. À la fin, on choisit le sommet dont l'ajout ou la suppression minimiserait (14) et on l'ajoute ou le supprime de S selon le cas. Le mouvement inverse est marqué comme tabou. On remarque que, suivant le principe d'une recherche tabou, on fait le mouvement même si cela ne conduit pas à une amélioration instantanée de (14). L'algorithme s'arrête lorsque l'on fait une certaine quantité de mouvements consécutifs sans amélioration de (14), et on ajoute alors à la formulation la contrainte (9) avec l'ensemble S qui a donné la plus petite valeur de (14), si cette valeur est négative.

Les quatre renforcements décrits ci-dessous permettent de faire une amélioration très importante par rapport à la classe `SolExacteBase` : même si le temps de calcul reste assez élevé, la taille en mémoire occupée par l'arbre diminue beaucoup, étant assez souvent inférieure à une dizaine

de mégaoctets, et le plus souvent inférieure à 1 Mo. Ces contraintes de renforcement permettent ainsi de couper un grand nombre de branches. La solution exacte peut donc être calculée par ces méthodes dans quelques instances.

Dans nos tests, nous avons trouvé que les recherches tabou prennent assez longtemps à s'exécuter à chaque fois qu'une solution réalisable fractionnaire est trouvée par `CPLEX` et que les contraintes ajoutées ne semblent pas être sensiblement meilleures que celles obtenues par les heuristiques gloutonnes, plus rapides à s'exécuter. Le temps de calcul avec les recherches tabou est en général plus long qu'avec les heuristiques gloutonnes, puisque, même si la recherche tabou permet d'éviter d'explorer un peu plus de l'arbre, le temps supplémentaire passé en chaque nœud augmente le temps total de l'algorithme de résolution.

3.3.3 Deuxième résolution exacte

La dernière résolution exacte, correspondant à la classe `SolExacteV2`, implémente le PLNE non-compact (10). À cause du nombre exponentiel des contraintes (10k), cette contrainte doit être traitée de façon spéciale. Contrairement aux contraintes (8) et (9) pour (5), qui viennent renforcer la formulation, la contrainte (10k) fait partie de la formulation. Elle ne peut donc pas être implémentée seulement comme une `UserCutCallbackI` de `CPLEX`, qui ne sert que pour les contraintes de renforcement, mais doit être implémentée comme une `LazyConstraintCallbackI`. Les « lazy constraints » de `CPLEX` sont appelées à chaque fois qu'une solution entière candidate est trouvée, afin de vérifier que cette solution satisfait bien toutes les contraintes du problème, y compris les contraintes en nombre exponentiel. Il faut donc que l'algorithme de séparation correspondant soit exacte et qu'il ne retourne rien que dans le cas où il n'y a aucune contrainte de ce type violée.

Puisque les « lazy constraints » ne sont appelées par `CPLEX` que sur des solutions entières, ce fait peut être utilisé pour simplifier l'algorithme de séparation. L'algorithme que nous avons utilisé pour (10k) est alors le suivant : pour chaque t et chaque k, on considère le graphe formé par les x_{ijkt}. On regarde les variables z_{ikt} pour déterminer quels sont les sommets i qui doivent être visités, qui sont stockés dans un ensemble S. On part alors de l'usine et on parcourt l'unique tournée du véhicule k, en supprimant de S les sommets visités dans cette tournée. Si S est non-vide à la fin, alors S contient des sous-tournées qui ne passent pas par l'usine, et donc on rajoute à la formulation la contrainte (10k) correspondante à S. Cet algorithme s'exécute en temps polynomial et sépare les contraintes de façon exacte sur les solutions entières.

Avec cette implémentation, on observe que le temps d'exécution de `SolExacteV2` est comparable à celui de `SolExacteBase`, et donc assez long et gourmand en mémoire pour des instances de taille modérée à grande. Un des problèmes est que, comme des contraintes de type (10k) ne sont ajoutées que lorsqu'une solution entière ne vérifiant au moins une de ces contraintes est trouvée, on n'en ajoute pas beaucoup, et l'algorithme de résolution peut tourner pendant longtemps sur des solutions fractionnaires de (10) qui ne vérifient pas des contraintes de type (10k). Afin de résoudre ce problème, notre stratégie a été d'implémenter (10k) aussi comme des contraintes de renforcement à travers une `UserCutCallbackI`.

Pour cela, on réécrit d'abord (10k) sous la forme équivalente suivante :

$$\sum_{i\in S}\sum_{j\notin S} x_{ijkt} \geq z_{ekt} \qquad S \subset \{1,\dots,n\}, |S| \geq 2, k \in \{0,\dots,K-1\}, t \in \{0,\dots,\ell-1\}, e \in S \tag{15}$$

Cette forme peut en effet être obtenue à partir de (10k) en ajoutant le terme $\sum_{i\in S}\sum_{j\notin S} x_{ijkt}$ des deux côtés et en utilisant (10i) et, réciproquement, en partant de cette forme, on peut rajouter le terme $\sum_{i\in S}\sum_{j\in S,\, j\neq i} x_{ijkt}$ des deux côtés et utiliser (10i) pour retrouver (10k).

L'algorithme de séparation utilisé est alors le suivant, inspiré par la description dans [7] : pour tout $t \in \{0,\dots,\ell-1\}$ et $k \in \{0,\dots,K-1\}$, on parcourt les sommets $e \in \{1,\dots,n\}$. Pour chaque sommet e avec $z_{ekt} > 0$, on résout le problème de coupe minimum dans le graphe dont les sommets sont $\{0,\dots,n\}$, le sommet initial est e, le sommet final est 0, les seuls arcs existants sont les arcs (i,j) pour lesquels $x_{ijkt} > 0$, et la capacité d'un arc est x_{ijkt}. La valeur d'une coupe S n'est rien d'autre

que le membre de gauche de (15), et donc il suffit de vérifier si la valeur de la coupe minimum trouvée est plus petite que z_{ekt}. Si oui, la contrainte est violée par l'ensemble S correspondant à la coupe minimum ; si non, alors toutes les contraintes du type (15) sont satisfaites pour les valeurs fixées de t, k et e de cette itération.

La résolution du problème de coupe minimum a été implémentée sur `CPLEX` à travers sa formulation classique en programme linéaire (à variables réelles). Plus précisément, le programme linéaire résolu à t, k et e donnés est le suivant :

$$\min \sum_{(i,j)\in E} x_{ijkt} d_{ij} \tag{16a}$$

$$\text{s.c. } d_{ij} - w_i + w_j \geq 0 \qquad (i,j) \in E \tag{16b}$$

$$d_{ij} \in \mathbb{R}_+ \qquad (i,j) \in E \tag{16c}$$

$$w_i \in \mathbb{R} \qquad i \in \{1,\dots,n\} \setminus \{e\} \tag{16d}$$

$$w_e = 1,\ w_0 = 0 \tag{16e}$$

Dans ce problème, on dénote $E = \{(i,j) \in \{0,\dots,n\}^2 \mid x_{ijkt} > 0\}$, les variables de décision sont w_i pour $i \notin \{0,e\}$ et d_{ij} pour $(i,j) \in E$, les valeurs de w_0 et w_e sont fixées, et les valeurs de x_{ijkt} pour $(i,j) \in E$ sont considérées comme données. Par des résultats classiques sur le problème de coupe minimum, à l'optimum, les variables de décision sont toutes booléennes et s'interprètent en termes de la coupe optimale (S,T) de la façon suivante : on a $w_i = 1$ si et seulement si $i \in S$, et on a $d_{ij} = 1$ si et seulement si $i \in S$ et $j \in T$.

Cet algorithme de séparation s'exécute dans un temps raisonnable, puisque (16) est un PL à variables réelles, et permet ainsi d'obtenir des solutions par la formulation (10) dans un temps raisonnable, comme indiqué dans la Section 5.

4 Utilisation du logiciel

Notre projet comporte un fichier principal, écrit en `C++` et décrit dans la Section 4.1, qui permet la résolution des instances par les méthodes décrites précédemment, ainsi qu'un fichier auxiliaire, écrit en `Python` et décrit dans la Section 4.2, qui permet l'affichage par `graphviz` des instances résolues.

4.1 Résolution du PDI

Notre implémentation comporte un fichier exécutable, qui peut être généré en compilant le fichier `MAOA.cpp` et toutes ses dépendances, et qui permet de résoudre de PDI de façon exacte ou approchée sur une instance donnée à travers la ligne de commande. Pour ce faire, il faut appeler le logiciel par :

```
MAOA nom_du_fichier OPTIONS
```

Le nom du fichier ne doit pas contenir d'espaces et doit être le nom d'un fichier `.prp` valide contenant l'instance à être résolue. Les options facultatives permettent de choisir la méthode de résolution, chaque option est de la forme

```
--nom_de_l_option valeur
```

Les options disponibles et leurs valeurs possibles sont :

- *Option :* `--mode`
  Valeurs possibles : `AB`, `AC`, `AH`, `EB`, `EC`, `E2`
  Description : Choisit le mode de résolution du PDI :
  - `AB` : mode approché de base (classe `SolApprocheeBase`, Section 3.2.1)
  - `AC` : mode approché avec coupe (classe `SolApprocheeCoupe`, Section 3.2.2)
  - `AH` : mode approché heuristique (classe `SolApprocheeHeuristique`, Section 3.2.3)
  - `EB` : mode exact (première formulation) de base (classe `SolExacteBase`, Section 3.3.1)

- EC [par défaut] : mode exact (première formulation) avec coupes (classe SolExacteCoupe, Section 3.3.2)
- E2 : mode exact (deuxième formulation) (classe SolExacteV2, Section 3.3.3)

- *Option :* `--start`
  Valeurs possibles : AB, AC, AH, none
  Description : Mode de calcul de la solution approchée passée comme solution initiale à la méthode exacte. N'est pris en compte que si `--mode` vaut EB, EC ou E2.
  - AB : mode approché de base
  - AC : mode approché avec coupe
  - AH [par défaut] : mode approché heuristique
  - none : sans calcul de solution initiale
- *Option :* `--coupe`
  Valeurs possibles : G1, G2, T1, T2
  Description : Algorithme de coupe utilisé dans la classe SolExacteCoupe, selon les algorithmes décrits dans la Section 3.3.2. N'est pris en compte que si `--mode` vaut EC.
  - G1 [par défaut] : heuristique gloutonne pour (8)
  - G2 : heuristique gloutonne pour (9)
  - T1 : première recherche tabou pour (9)
  - T2 : deuxième recherche tabou pour (9)
- *Option :* `--verbose`
  Valeurs possibles : 0, 1, 2
  Description : Niveaux d'affichage de messages.
  - 0 : aucun affichage
  - 1 [par défaut] : affichage de principaux messages et de la solution trouvée
  - 2 : affichage de tous les détails de la résolution (y compris affichages de CPLEX)
- *Option :* `--repeat_approx`
  Valeurs possibles : Nombre entier strictement positif
  Description : Nombre de répétitions de la boucle principale des algorithmes approchés, comme décrit au début de la Section 3.2. N'est pris en compte que si `--mode` vaut AB, AC ou AH. Vaut 10 par défaut.
- *Option :* `--timeout`
  Valeurs possibles : Nombre réel strictement positif
  Description : Temps maximal de résolution des algorithmes exacts et du PLNE du VRP pour l'algorithme approché avec coupes. Ces algorithmes s'arrêtent après `timeout` secondes et la meilleure solution retrouvée jusqu'alors est retournée. N'est pris en compte que si `--mode` vaut AC, EB, EC ou E2, ou si `--start` vaut AC. Par défaut, les algorithmes s'exécutent jusqu'à la convergence sans temps maximal défini.
- *Option :* `--heuristic_steps`
  Valeurs possibles : Nombre entier strictement positif
  Description : Nombre de tours de boucle de l'étape 2 de l'algorithme approché heuristique de la Section 3.2.3. N'est pris en compte que si `--mode` vaut AH. Vaut 1000 par défaut.
- *Option :* `--tol_exact`
  Valeurs possibles : Nombre réel dans l'intervalle $(0, 1)$
  Description : Tolérance pour les algorithmes exactes. L'algorithme exacte s'arrête lorsque l'écart relatif entre la meilleure solution entière et la meilleure borne inférieure trouvées est inférieur ou égal à `tol_exact`. N'est pris en compte que si `--mode` vaut EB, EC ou E2. Vaut 1e-4 par défaut.
- *Option :* `--output`
  Valeurs possibles : Nom de fichier
  Description : Nom du fichier où la solution sera écrite. Par défaut, identique au nom du fichier de l'instance à être résolue mas avec l'extension `.prp` remplacée par `.txt`.

4.2 Affichage des résultats

L'affichage des résultats est faite par le fichier `solViz.py`. Il faut l'appeler par :
`python solViz.py nom_de_l_instance OPTIONS`
Le nom de l'instance ne doit pas contenir d'espaces ni d'extension de fichier. Il doit être le nom à la fois d'un fichier avec l'extension `.prp` contenant l'instance et d'un fichier avec l'extension `.txt` contenant sa résolution, calculée au préalable avec le logiciel `MAOA` de la Section 4.1. L'option facultative permet de régler un paramètre d'échelle :

- *Option :* `--scale`
  Valeurs possibles : Nombre entier strictement positif
  Description : Valeur d'échelle de la visualisation. Plus elle est grande, plus espacés seront les sommets dans la visualisation. Vaut 10 par défaut.

5 Résultats

Cette section décrit les résultats de plusieurs tests réalisés des méthodes décrites dans les sections précédentes. Sauf mention explicite du contraire, les tests ont été réalisés sur un ordinateur de bureau avec un processeur Intel i3-7100 3.90 GHz avec 8 Go de mémoire RAM, tournant sous Windows 10.

5.1 Temps d'exécution des algorithmes sur une instance simple

Nous avons d'abord testé nos algorithmes sur l'instance simple du fichier `Instance_1.prp`. Cette instance avec $n = 10$ et $\ell = 4$ correspond à une modification de celle du fichier `A_014_ABS1_15_1.prp` par suppression des 4 derniers clients (indices 11 à 14) et des 2 derniers instants de temps. Avec cette taille réduite, nous pouvons calculer des temps moyens d'exécution pour tous les algorithmes, alors que certains prennent déjà trop de temps et de mémoire sur l'instance `A_014_ABS1_15_1.prp`.

Algorithme	Temps de calcul (en ms)	
	Moyenne	Écart-type
`SolApprocheeBase`	974,595	258,871
`SolApprocheeCoupe`	192,010	21,348
`SolApprocheeHeuristique`	112,258	5,676
`SolExacteBase`	6533,490	449,900
`SolExacteCoupe` avec heuristique gloutonne pour (8)	414,912	588,431
`SolExacteCoupe` avec heuristique gloutonne pour (9)	628,588	378,190
`SolExacteCoupe` avec première recherche tabou	1707,150	836,062
`SolExacteCoupe` avec deuxième recherche tabou	7265,110	9375,070
`SolExacteV2`	22025,300	1045,750

Table 8 – Temps de calcul des algorithmes sur une instance de test avec $n = 10$ et $\ell = 4$ (moyennes sur 10 répétitions)

Les résultats pour les trois algorithmes approchés et les trois algorithmes exacts (avec les quatre renforcements implémentés dans `SolExacteCoupe`) sont donnés dans la Table 8, où des temps de calcul moyens et les écart-type correspondants ont été calculés sur 10 répétitions de chaque algorithme sur la même instance. Dans cette instance simple, tous les algorithmes (même les approchés) ont retrouvé la même solution, dont la valeur est 16996. Cette solution est représentée dans la Figure 2, où les couleurs d'arcs différentes représentent les différents instants de temps. Comme l'instance utilisée est très simple, les clients 6, 7, 9 et 10 n'ont pas été visités.

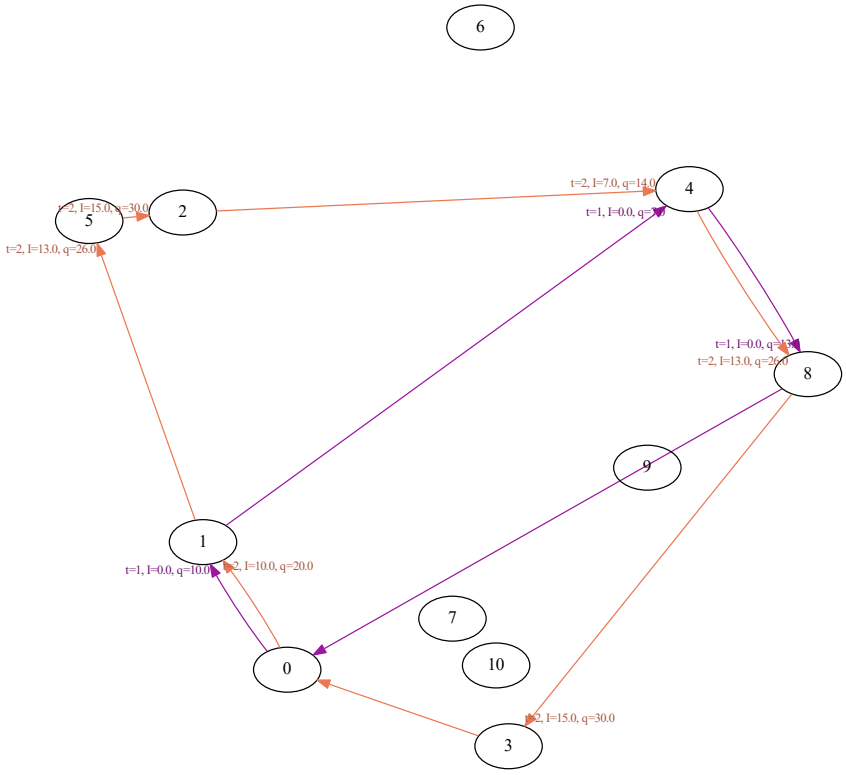

Figure 2 – Solution trouvée de l'instance `Instance_1.prp`

Les algorithmes approchés ont été exécutés avec 10 itérations de la boucle principale, sans arrêt anticipé du VRP pour `SolApprocheeCoupe`, et avec 1000 tours de boucle d'amélioration heuristique de la solution pour `SolApprocheeHeuristique`. Les algorithmes exacts ont été tournés sans passer des solutions initiales en argument, sans sortie anticipée, et avec une tolérance de 10^{-6}.

Concernant les algorithmes approchés, on observe que l'ajout des contraintes de renforcement pour le problème du VRP conduit à une amélioration importante de la vitesse. La résolution heuristique du VRP permet de réduire davantage le temps de calcul.

Pour les algorithmes exacts, les trois premières heuristiques de `SolExacteCoupe` permettent une réduction significative du temps de calcul par rapport à `SolExacteBase`, mais l'heuristique avec la deuxième recherche tabou ne permet pas d'améliorations significatives. Comme toutes les heuristiques se basent sur des choix initiaux d'ensembles aléatoires, elles présentent des grandes variations du temps de calcul entre les exécutions. Cela est particulièrement vrai pour la deuxième recherche tabou, qui s'exécute souvent de façon beaucoup plus vite que `SolExacteBase` mais dont certaines exécutions sont considérablement plus lentes que `SolExacteBase`, car notre recherche tabou, trop simpliste, a du mal à trouver de bonnes contraintes à rajouter, et son temps d'exécution s'ajoute au temps d'exécution de `SolExacteBase`.

Concernant `SolExacteV2`, sont temps de calcul est plus important car l'ajout de l'indice des véhicules augmente beaucoup la dimension de l'espace de décision du problème et l'algorithme de séparation de la contrainte (10k) prend un temps important à s'exécuter.

Les résultats donnés dans la Table 8 ne peuvent pas être extrapolés à des instances de taille plus grande. On aurait par exemple l'impression que l'algorithme `SolExacteBase` est plus performant que `SolExacteV2`, mais en effet, sur l'instance `A_014_ABS1_15_1.prp`, le deuxième se termine en environ 3 min, alors que le premier a dû être arrêté au bout d'une dizaine de minutes, avant de se terminer, car son arbre prenait trop d'espace en mémoire. Néanmoins, la Table 8 permet déjà d'illustrer l'intérêt des heuristiques à la fois pour les solutions exactes et pour le

problème VRP dans les solutions approchées.

5.2 Comparaison entre `SolExacteCoupe` et `SolExacteV2`

Afin de comparer les méthodes de résolution exacte `SolExacteCoupe` et `SolExacteV2`, nous avons testé ces deux algorithmes sur l'instance `A_014_ABS1_15_1.prp` avec 10 répétitions chacun. La solution approchée retournée par `SolApprocheeHeuristique` a été fournie comme condition initiale à chacun de ces algorithmes exacts. Par manque de temps de calcul disponible, on a décidé de tester `SolExacteCoupe` uniquement avec l'heuristique gloutonne pour (8), qui a été la plus rapide dans la Table 8 et qui a aussi donné des résultats assez rapides dans d'autres tests réalisés. Les deux algorithmes exacts ont été tournés avec une tolérance de 10^{-6} et sans sortie anticipée. Les temps de calcul sont donnés dans la Table 9. Toutes les exécutions de l'algorithme ont trouvé la même valeur optimale, de 40390, et la solution optimale trouvée est donnée dans la Figure 3.

Algorithme	Temps de calcul (en s)	
	Moyenne	Écart-type
`SolExacteCoupe` avec heuristique gloutonne pour (8)	68,887	48,337
`SolExacteV2`	566,929	26,824

Table 9 – Temps de calcul des algorithmes `SolExacteCoupe` et `SolExacteV2` sur l'instance `A_014_ABS1_15_1.prp` (moyennes sur 10 répétitions)

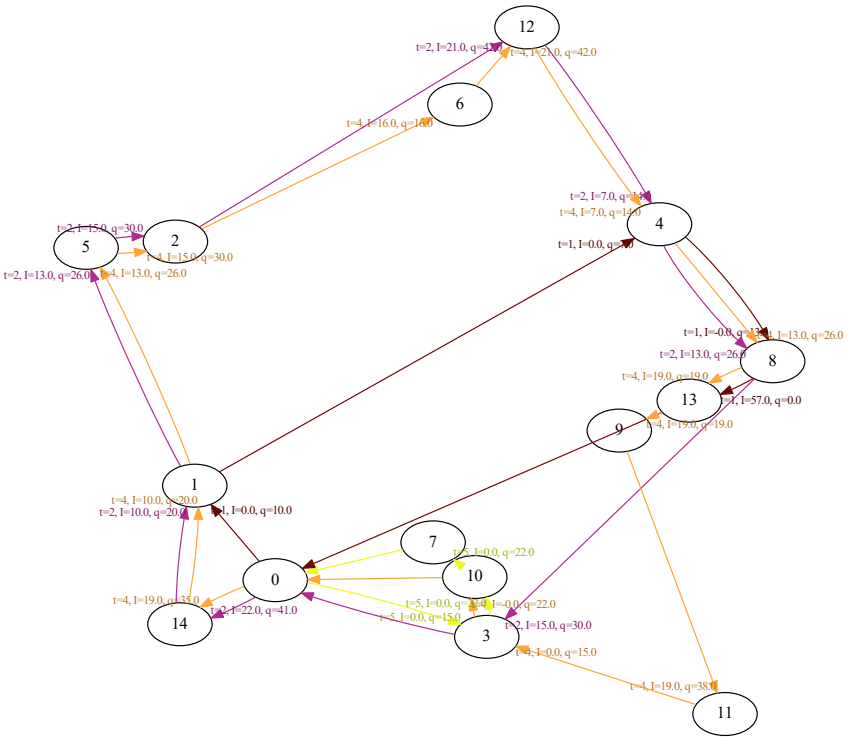

Figure 3 – Résolution exacte de l'instance `A_014_ABS1_15_1.prp`

On observe sur la Table 9 que le temps de calcul avec `SolExacteCoupe` est de l'ordre de 8 fois plus petit que celui avec `SolExacteV2`. Sur l'instance simple de la Table 8, cet rapport était de l'ordre de 50 : l'écart des temps entre les deux algorithmes s'est réduit avec l'augmentation de la difficulté de l'instance. Comme notre implémentation de l'algorithme de séparation pour la

contrainte (10k) est assez lente et utilise la résolution d'un PL, cet écart pourrait probablement être encore réduit avec une implémentation de l'algorithme de séparation plus efficace.

On a observé aussi, qualitativement, que, lors de la résolution par `SolExacteCoupe`, peu de contraintes de renforcement sont ajoutées (de l'ordre de 250), probablement car notre heuristique gloutonne a du mal à en trouver d'autres, et l'arbre de la résolution du PLNE est assez explorée (environ 3000 nœuds). Inversement, la résolution par `SolExacteV2` ajoute beaucoup de contraintes de renforcement (de l'ordre de 7500), puisque l'algorithme de séparation correspondant est exacte, et cela permet de parcourir moins l'arbre du PLNE avant de trouver la solution (moins de 100 nœuds).

Une autre constatation que l'on peut dégager de la Table 9 est que la variation du temps de calcul est beaucoup plus importante pour `SolExacteCoupe`. Son écart-type relatif (écart-type divisé par la moyenne) est de 70,2%, alors que celui de `SolExacteV2` est de 4,73%. Ainsi, le temps de résolution de `SolExacteCoupe` est beaucoup plus imprévisible : dans les données utilisées pour construire la Table 9, le temps minimal a été de 20,1 s, alors que le temps maximal a été de 3 minutes.

5.3 Résultats des algorithmes sur de petites instances de type A

Les algorithmes `SolApprocheeBase`, `SolApprocheeCoupe`, `SolApprocheeHeuristique` et `SolExacteCoupe` avec heuristique gloutonne pour (8) ont été tournés sur 98 instances de type A et taille $n = 14$ afin de comparer leurs temps d'exécution mais aussi l'écart entre les valeurs retournées par les algorithmes approchés et la solution exacte. Les algorithmes approchés ont été exécutés avec 10 itérations de la boucle principale, sans arrêt anticipé du VRP pour `SolApprocheeCoupe`, et avec 1000 tours de boucle d'amélioration heuristique de la solution pour `SolApprocheeHeuristique`. L'algorithme exact `SolExacteCoupe` a été initialisé à la meilleure solution approchée et a été tourné sans sortie anticipée avec une tolérance de 10^{-4}. Les histogrammes des temps de calcul pour les quatre algorithmes sont donnés dans la Figure 4 et les écarts relatifs des solutions approchées par rapport à la solution exacte sont donnés dans la Figure 5. Des statistiques de temps de calcul et d'écart sont présentées dans les Tables 10 et 11, alors que la Table 12 présente, pour chaque algorithme approché, le nombre d'instances dans lequel il a donné la meilleure solution approchée et le nombre d'instances dans lequel sa valeur a coïncidé avec celle de la solution exacte. Le temps total de calcul a été d'environ 6 jours.

Algorithme	Temps de calcul (en s)				
	Moyenne	Écart-type	Quartile 1	Médiane	Quartile 3
`SolApprocheeBase`	54,288	259,665	1,149	1,760	6,147
`SolApprocheeCoupe`	0,908	0,570	0,675	0,770	1,055
`SolApprocheeHeuristique`	0,460	0,194	0,321	0,405	0,526
`SolExacteCoupe`	5173,936	10179,242	236,150	721,510	4605,790

Table 10 – Statistiques de temps de calcul sur les instances de type A avec $n = 14$

Algorithme	Écart par rapport à la solution exacte (en %)				
	Moyenne	Écart-type	Quartile 1	Médiane	Quartile 3
`SolApprocheeBase`	2,043	3,908	0,070	0,274	1,831
`SolApprocheeCoupe`	2,048	3,967	0,171	0,434	1,556
`SolApprocheeHeuristique`	2,559	4,293	0,312	0,893	2,896

Table 11 – Statistiques d'écart par rapport à la solution exacte sur des instances de type A avec $n = 14$

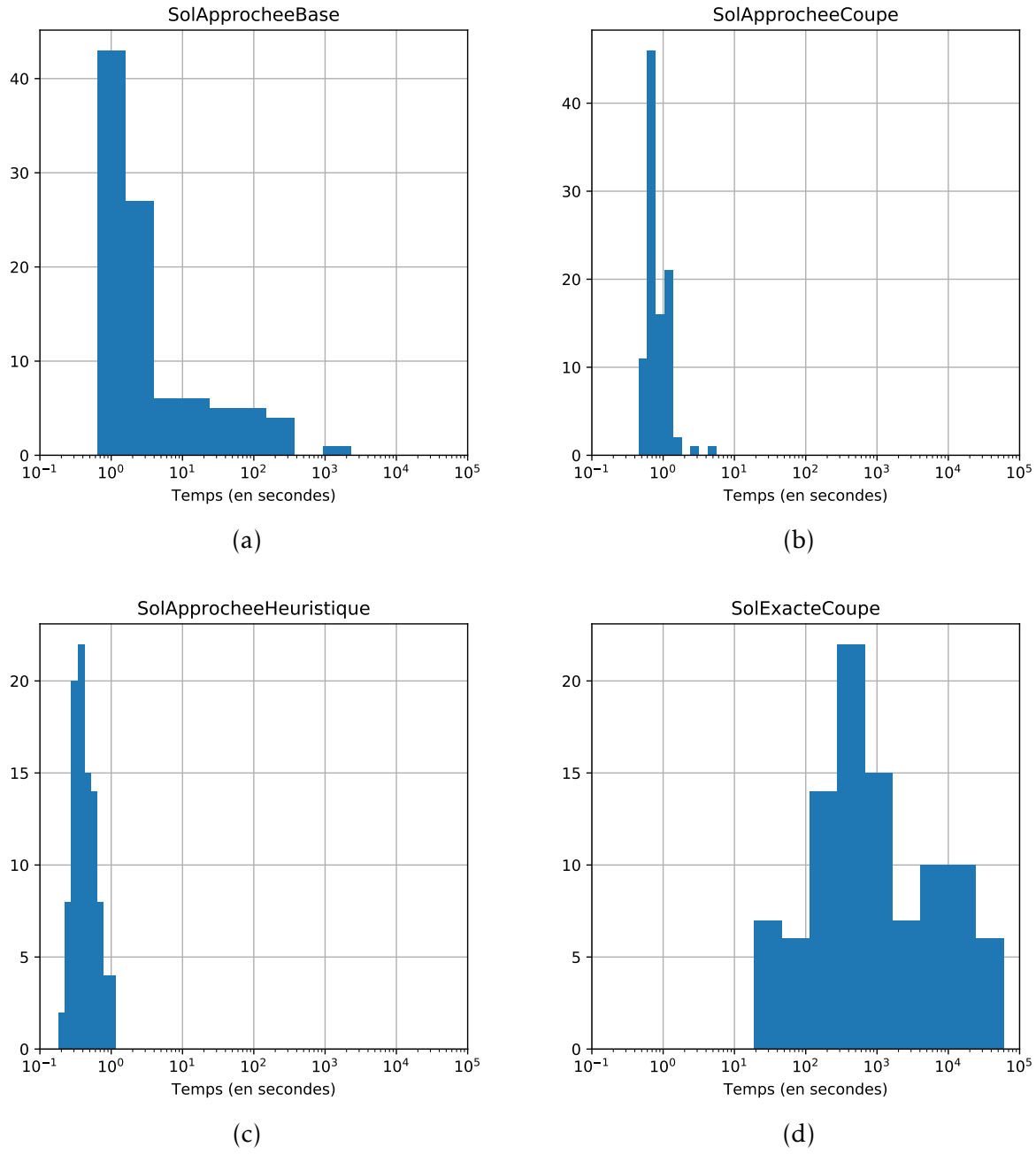

FIGURE 4 – Temps de calcul des algorithmes (a) SolApprocheeBase, (b) SolApprocheeCoupe, (c) SolApprocheeHeuristique et (d) SolExacteCoupe sur les instances de type A avec $n = 14$

Concernant le temps de calcul, on remarque que SolApprocheeBase est en général assez rapide sur ces instances, et que, pour environ de 75% des instances, son temps de résolution est inférieur à 6,2 s. Cependant, quelques instances ont un temps de résolution très long par cet algorithme, de l'ordre de 10^3 s, soit environ un quart d'heure, ce qui est beaucoup pour une résolution approchée. Cela contribue ainsi à augmenter le temps moyen de résolution de SolApprochee Base sur les instances testées, et donne un écart-type presque 5 fois supérieur à la moyenne. La contrainte de renforcement introduite dans la formulation du VRP dans SolApprocheeCoupe permet de réduire de façon considérable le temps de résolution : il est toujours inférieur à 10 s, et inférieur à 1 s dans environ 75% des instances. La résolution heuristique du VRP dans SolApprocheeHeuristique réduit davantage le temps de résolution : 1 seule instance a pris plus de 1 s à être résolue, et plus de la moitié d'entre elles a été résolue en moins de 0,5 s.

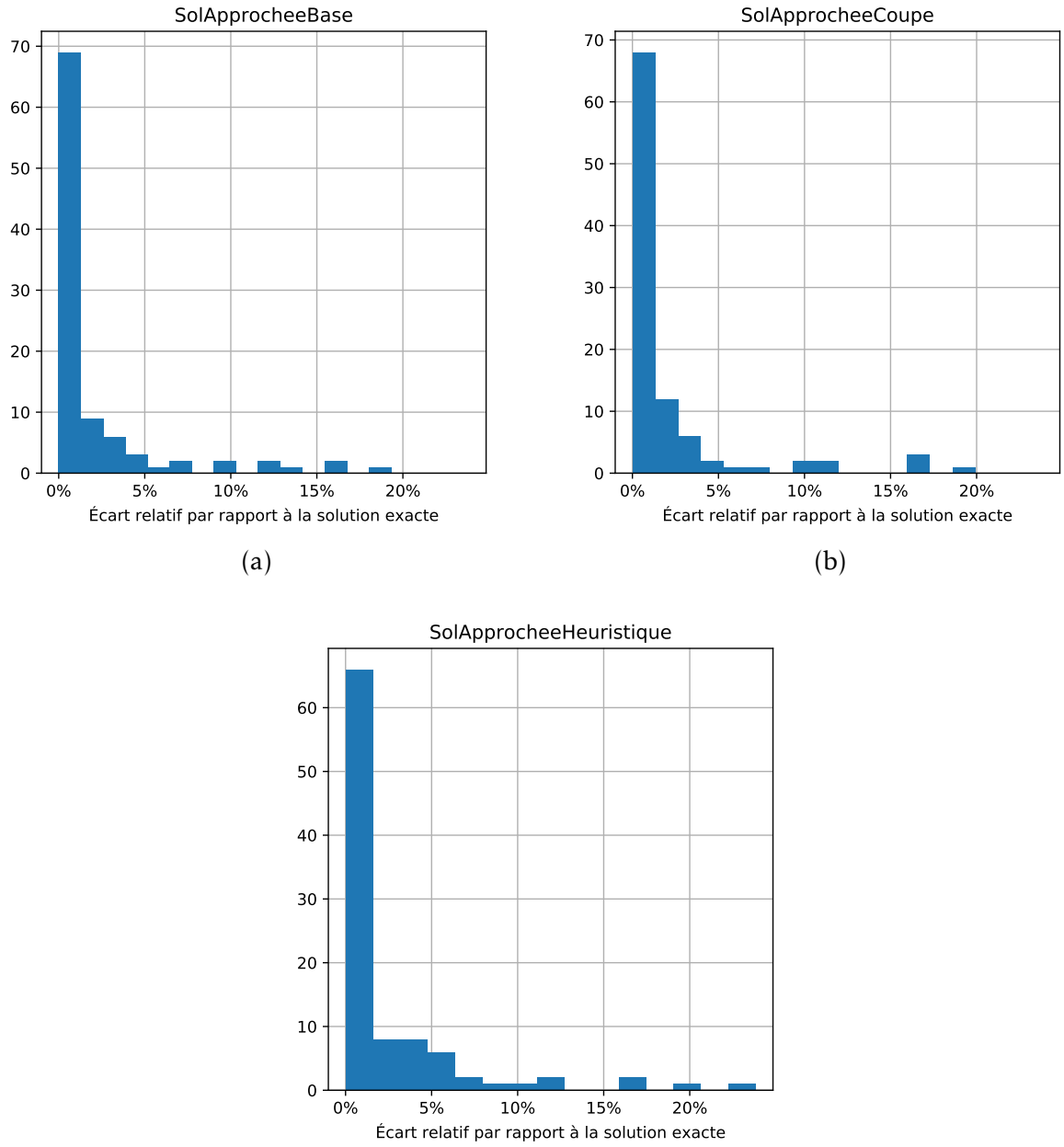

FIGURE 5 – Écart relatif par rapport à la solution exacte des algorithmes (a) SolApprocheeBase, (b) SolApprocheeCoupe et (c) SolApprocheeHeuristique sur des instances de type A avec $n = 14$

Comme attendu, la résolution exacte prend beaucoup plus de temps. Au total, 98,9% du temps d'exécution de nos tests a été passé sur la résolution exacte. Si quelques instances peuvent bien être résolues dans quelques dizaines de secondes, il y en a aussi qui ont pris à elles seules une demi-journée de calcul. La variation observée est très grande, due à la fois à la variation de la méthode SolExacteCoupe avec l'heuristique gloutonne pour (8), comme remarqué dans la Section 5.2, mais aussi aux différents degrés de difficulté des instances. On a ainsi un écart-type des temps de résolution exacte de l'ordre de 3 h, alors que le temps moyen de résolution s'établit à 1,5 h, mais 50% des instances sont résolues en environ 12 min ou moins.

La Figure 5 et les Tables 11 et 12 montrent que, en général, les algorithmes approchés donnent des solutions approchées de bonne qualité. L'écart moyen est essentiellement le même pour SolApprocheeBase et SolApprocheeCoupe, de l'ordre de 2,1%, et légèrement supérieur pour SolAp-

	SolApprochee		
	Base	Coupe	Heuristique
Nombre d'instances sur lesquelles chaque algorithme approché a donné la meilleure solution approchée	85	61	44
Nombre d'instances sur lesquelles chaque algorithme approché a donné la même solution que la résolution exacte	21	8	4

Table 12 – Nombres d'instances sur lesquelles chaque algorithme approché a donné la meilleure solution ou la solution exacte

procheeHeuristique. Pour les trois algorithmes, l'écart moyen ne dépasse pas environ 3% pour plus de 75% des instances. Cependant, quelques instances exceptionnelles peuvent avoir des écarts plus importants, dépassant le 20% pour SolApprocheeHeuristique.

Le plus souvent, SolApprocheeBase donne la meilleure valeur parmi les solutions approchées, mais cela n'est pas systématique et les autres méthodes donnent aussi des solutions avec une bonne qualité assez souvent. C'est aussi SolApprocheeBase qui retrouve la solution exacte avec la plus grande fréquence, ce qui arrive sur 21 de nos instances. Les autres méthodes retrouvent très rarement la solution exacte.

Le facteur limitant qui nous empêche de faire la comparaison de cette section sur toutes les instances de taille 14 est le temps de calcul de SolExacteCoupe : avec une simple extrapolation linéaire, il aurait fallu environ 30 jours de calcul pour résoudre de façon exacte toutes ces instances. Nous avons cependant bien réussi à calculer des solutions approchées par les trois algorithmes approchés. Puisque les caractéristiques observées sont essentiellement les mêmes que celles de la Figure 4(a)–(c) et des Tables 10 et 12, nous ne détaillons pas ces résultats ici.

5.4 Solutions approchées pour toutes les instances de taille jusqu'à 100 et quelques instances de taille 200

Comme vu dans les sections précédentes, l'algorithme approché heuristique s'exécute assez rapidement pour des petites instances, ce qui permet d'envisager d'appliquer cet algorithme à des instances de tailles plus grandes. Afin d'illustrer que cela est bien possible, nous avons tourné l'algorithme SolApprocheeHeuristique sur toutes les 1500 instances fournies de taille au maximum 100, avec 10 répétitions de la boucle principale de l'algorithme approché et 1000 pas d'amélioration heuristique de la solution du VRP. Les résultats, séparés par type et taille d'instance, sont donnés dans la Figure 6.

Pour chaque type d'instance (A ou B) et pour chaque taille ($n \in \{14, 50, 100\}$ pour A, $n \in \{50, 100\}$ pour B), la Figure 6 représente la dispersion des temps de résolution obtenus. L'extrémité inférieure du rectangle en bleu clair représente le premier quartile, son extrémité supérieure représente le troisième quartile, la ligne rouge à l'intérieur du rectangle représente la médiane, et le cercle représente la moyenne. Les barres extérieures au rectangle donnent les valeurs minimales et maximales observées.

Comme déjà remarqué précédemment, les instances de type A et taille $n = 14$ se résolvent en moins d'une seconde dans la grande majorité des cas, avec un temps médian de l'ordre de 0,3 s. Comme attendu, l'augmentation de la taille n fait augmenter le temps de résolution, mais il reste assez limité : toutes les instances de type A restent avec un temps de résolution inférieur à 10 s, avec un temps médian d'environ 1,5 s pour $n = 50$ et de 3 s pour $n = 100$.

Le nombre n de clients n'est pas le seul paramètre qui détermine le temps de résolution, et cela est illustré sur les temps de résolution des instances de type B, largement supérieur. Les instances de type A se font avec un horizon de temps $\ell = 6$, alors que celles de type B ont un horizon de temps $\ell = 20$. En plus, les instances de type A ont en général un très grand nombre de véhicules, plus que le nécessaire pour résoudre le problème, alors que les instances de type B disposent en général de peu de véhicules. Ces caractéristiques différentes expliquent le temps de

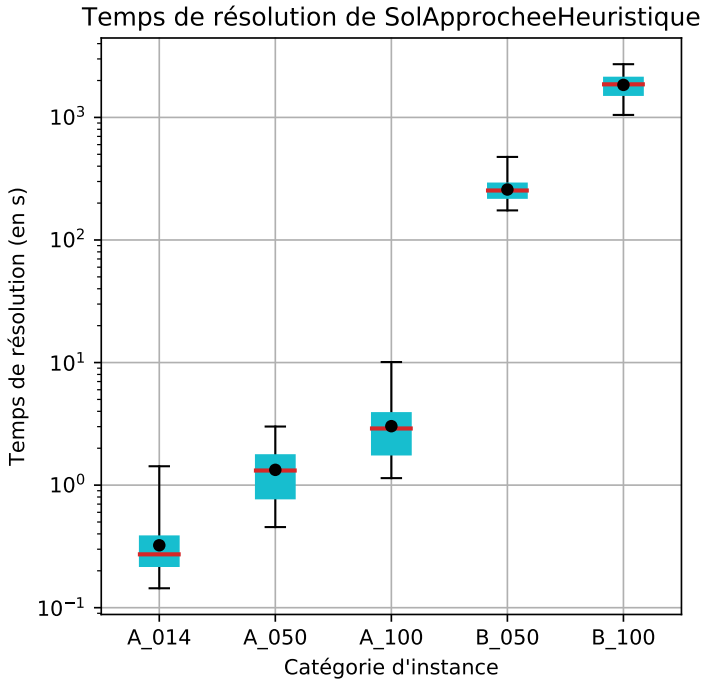

FIGURE 6 – Temps d'exécution de `SolApprocheeHeuristique` sur les 1500 instances de taille $n \leq 100$

résolution supérieurs des instances de type B, qui dépassent les 3 min dans la quasi-totalité des cas et peuvent arriver jusqu'à environ 30 min sur quelques instances de taille $n = 100$.

Le temps de résolution des instances de type B avec $n = 200$ est nettement supérieur, ce qui nous a empêché, par manque de temps, de les traiter toutes, mais nous avons réussi à résoudre quelques unes sur un ordinateur portable avec un processeur Intel i5-10310U de 2,21 GHz avec 32 Go de mémoire RAM. À titre d'illustration, la Figure 7 représente les solutions trouvées pour les quatre premières instances de type B et taille $n = 200$, dont les temps de calcul ont été, respectivement, de 1 h 20 min, 1 h 25 min, 2 h et 1 h 30 min.

On a constaté que le facteur limitant principal de la résolution de ces instances est la résolution exacte du problème LSP lors de la première itération de l'algorithme approché, qui peut prendre plus de 1 h. Améliorer la résolution de ce problème LSP consiste donc en la meilleure piste pour améliorer le temps de calcul des solutions. Nous avons fait le choix simplificateur dans notre implémentation de résoudre ce problème de façon exacte, ce qui marche très bien pour les instances de type A et pour les petites instances de type B. Cependant, puisque l'on est à la recherche dans `SolApprocheeHeuristique` d'une solution approchée de bonne qualité, il n'est pas forcément nécessaire de résoudre LSP de façon exacte, ou du moins pour le premier tour de boucle. On pourrait alors utiliser une heuristique pour la première résolution du LSP, ou alors arrêter la résolution du PLNE dès qu'un nombre minimal de solutions ait été trouvé ou qu'une tolérance relativement grande ait été franchie. Par manque de temps, ces idées n'ont pas été implémentées afin de privilégier l'implémentation de méthodes exactes de résolution.

Références

[1] Y. Adulyasak, J.-F. Cordeau, and R. Jans. Formulations and branch-and-cut algorithms for multivehicle production and inventory routing problems. *INFORMS Journal on Computing*, 26(1):103–120, feb 2014.

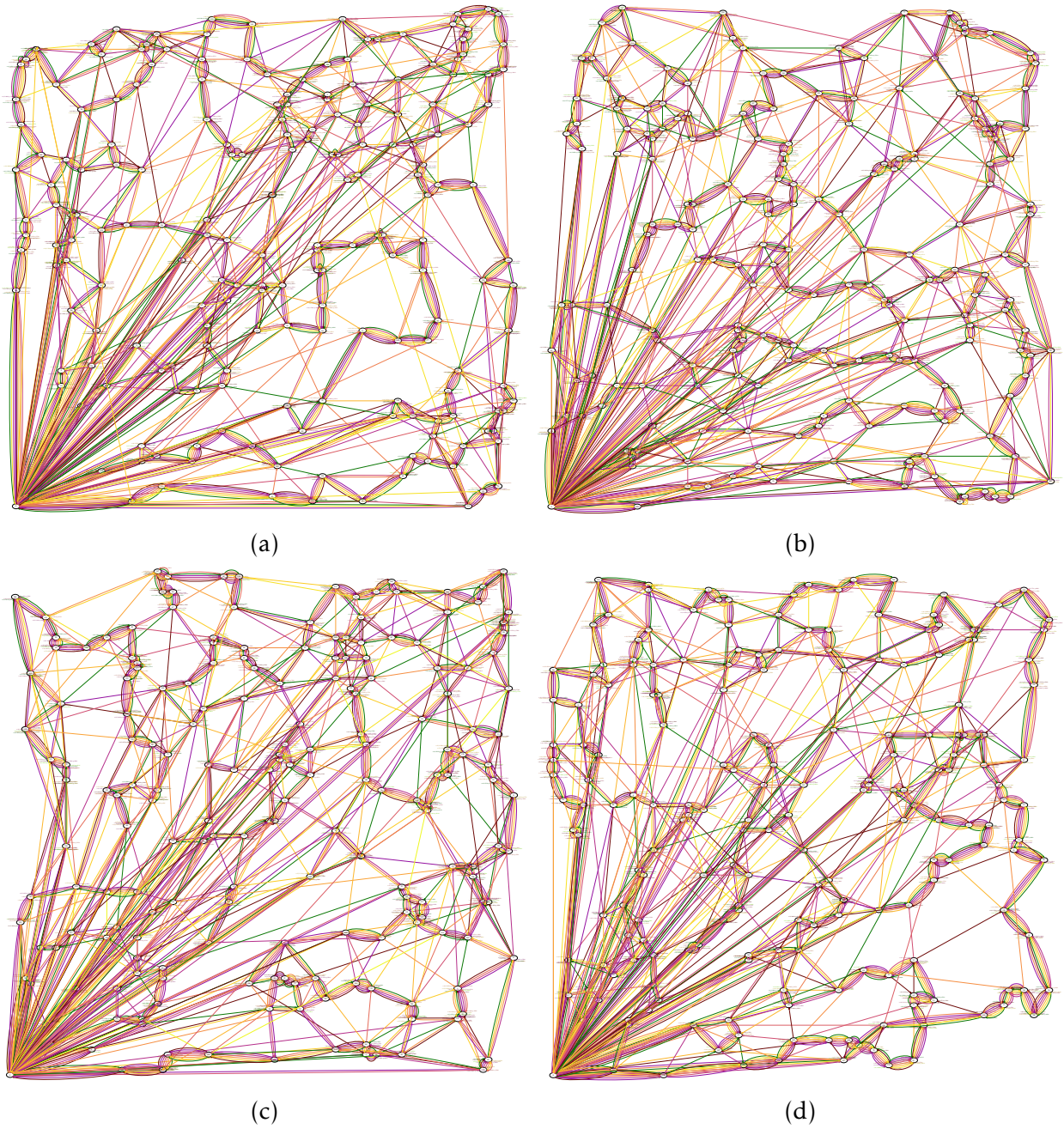

FIGURE 7 – Solutions trouvées avec `SolApprocheeHeuristique` pour les quatre premières instances (dans l'ordre (a), (b), (c) et (d)) de type B de taille $n = 200$

[2] Y. Adulyasak, J.-F. Cordeau, and R. Jans. The production routing problem: a review of formulations and solution algorithms. *Comput. Oper. Res.*, 55:141–152, 2015.

[3] P. Augerat, J. Belenguer, E. Benavent, A. Corberán, and D. Naddef. Separating capacity constraints in the CVRP using tabu search. *European Journal of Operational Research*, 106(2-3):546–557, apr 1998.

[4] P. Augerat, J. Belenguer, E. Benavent, A. Corberán, D. Naddef, and G. Rinaldi. Computational results with a branch and cut code for the capacitated vehicle routing problem. Technical Report INPG-RR-949-M, Laboratoire ARTEMIS, Institut IMAG, 1995.

[5] J. F. Bard and N. Nananukul. Heuristics for a multiperiod inventory routing problem with production decisions. *Computers & Industrial Engineering*, 57(3):713–723, oct 2009.

[6] J. F. Bard and N. Nananukul. A branch-and-price algorithm for an integrated production and inventory routing problem. *Comput. Oper. Res.*, 37(12):2202–2217, 2010.

[7] M. Ruokokoski, O. Solyali, J.-F. Cordeau, R. Jans, and H. Süral. Efficient formulations and a branch-and-cut algorithm for a production-routing problem. Technical Report G-2010-66, Group for Research in Decision Analysis (GERAD), HEC Montréal, Canada, 2010.