

\Provisoire

# Thèse

présentée pour l'obtention du titre de

**Docteur de l'Université du Havre**

Spécialité : Informatique

par **Gaëtan Lesauvage**

# Optimisation Dynamique en Environnement Incertain

Soutenue publiquement le xx xxxxxxxxxxxx 2012 devant la commission d'examen composée de

*Président :* Toto

*Rapporteur :* Titi

*Examinateur :* Tata

*Invité :* Tutu

*Directeurs :* Pr. F. Guinand

Dr. S. Balev

---

Université de Normandie

Faculté des Sciences et Techniques, Université du Havre

25, rue Philippe-Lebon, BP 540 - 76058 Le Havre cedex



## **Remerciements**

*Remerciements...*

*Dédicace*

## Résumé

Résumé

**Mots cls :** Mot1, Mot2

## Résumé

Abstract

**Keywords :** 1stKey, 2ndKey



# Table des matières

<b>Introduction</b>	<b>7</b>
<b>I Applications aux terminaux à conteneurs</b>	<b>9</b>
Introduction . . . . .	9
I.1 Structure et organisation des terminaux à conteneurs . . . . .	12
I.1.1 Structure . . . . .	13
I.1.2 Engins de manutention . . . . .	14
I.1.3 Réseau routier . . . . .	16
I.1.4 Activités des chariots cavaliers . . . . .	17
I.1.5 Le Terminal de Normandie . . . . .	20
I.2 Optimisation des terminaux à conteneurs . . . . .	21
I.2.1 Structure du terminal . . . . .	22
I.2.2 Allocation des berges et transbordement . . . . .	23
I.2.3 Allocation des grues de quai . . . . .	24
I.2.4 Positionnement des conteneurs . . . . .	25
I.2.5 Routage des véhicules . . . . .	26
I.2.6 Ordonnancement et affectation des missions . . . . .	29
Conclusions . . . . .	29
<b>II Contexte théorique</b>	<b>31</b>
Introduction . . . . .	31
II.1 Dynamique et incertitude en optimisation . . . . .	31
II.1.1 Définition . . . . .	31
II.1.2 Optimisation et problème décisionnel . . . . .	32
II.1.3 Identifier l'incertitude . . . . .	33
II.1.4 Gérer l'incertitude . . . . .	34

II.1.5	Gestion de la dynamique . . . . .	34
II.1.6	Méthodes de résolutions adaptées aux environnements dynamiques . . . . .	37
II.2	Problèmes de voyageurs de commerce . . . . .	47
II.2.1	Définition . . . . .	47
II.2.2	Historique . . . . .	48
II.2.3	Méthodes de résolution . . . . .	49
II.2.4	Généralisation à $m$ voyageurs : The Multiple Traveling Salesman Problem (M-TSP) . . . . .	56
II.3	Problèmes de tournées de véhicules . . . . .	65
II.3.1	Definition . . . . .	65
II.3.2	Les variantes du problème . . . . .	66
II.3.3	Problème décisionnel et formulation mathématique . . . . .	68
II.3.4	Fonction objectif . . . . .	70
II.3.5	Méthodes de résolution . . . . .	70
II.3.6	Problème dynamique . . . . .	81
II.4	Problèmes d'ateliers . . . . .	85
II.4.1	Définition et variantes du problème . . . . .	85
II.4.2	Méthodes de résolution . . . . .	89
II.4.3	Problème dynamique . . . . .	95
	Conclusions . . . . .	98
<b>III</b>	<b>Ordonnancement et affectation des missions des chariots cavaliers</b>	<b>101</b>
III.1	Dynamique et incertitude . . . . .	101
III.1.1	Gestion du temps . . . . .	101
III.1.2	Incertitude . . . . .	102
III.2	Modélisations du problème . . . . .	105
III.2.1	Problème de tournées de véhicules . . . . .	105
III.2.2	Problème d'atelier . . . . .	113
III.2.3	Conclusion . . . . .	117
III.3	Méthodes de résolution . . . . .	117
III.3.1	Méthode de résolution exacte . . . . .	118
III.3.2	Méthodes de résolution approchées . . . . .	124
III.3.3	Méthodes dynamiques . . . . .	126

Conclusions . . . . .	147
<b>IV Simulateur de terminal à conteneurs</b>	<b>149</b>
Introduction . . . . .	149
IV.1 Spécifications . . . . .	151
IV.1.1 Technologie . . . . .	151
IV.1.2 Objectifs . . . . .	151
IV.1.3 Flexibilité . . . . .	152
IV.2 Modélisation du système . . . . .	152
IV.2.1 Graphe routier . . . . .	152
IV.2.2 Réseau de stockage . . . . .	153
IV.2.3 Système de géolocalisation laser (LDTT) . . . . .	154
IV.2.4 Mobilité au sein du terminal . . . . .	155
IV.2.5 Temporalité . . . . .	156
IV.2.6 Événements . . . . .	158
IV.3 Collecte des données . . . . .	158
IV.4 Structuration des données . . . . .	160
IV.4.1 Réseau routier . . . . .	161
IV.4.2 Zones de stockage . . . . .	161
IV.4.3 Conteneurs . . . . .	164
IV.4.4 Missions . . . . .	165
IV.4.5 Bornes de positionnement laser . . . . .	166
IV.4.6 Description des véhicules . . . . .	167
IV.4.7 Description de l'ordonnanceur de missions . . . . .	168
IV.4.8 Événements . . . . .	170
IV.5 Architecture logicielle . . . . .	173
IV.5.1 Modularité . . . . .	173
IV.5.2 Distribution . . . . .	173
IV.5.3 Descripteurs du contenu du terminal . . . . .	175
IV.5.4 Base de données . . . . .	175
IV.6 Modules proposés . . . . .	178
IV.6.1 Module de mobilité des chariots cavaliers . . . . .	178
IV.6.2 Module de routage des véhicules . . . . .	178

IV.6.3	Module d'ordonnancement des missions . . . . .	179
IV.6.4	Module de représentation 2D du terminal . . . . .	180
IV.6.5	Module de transformation XML ( <i>parser</i> ) . . . . .	182
IV.6.6	Modules de génération de données . . . . .	182
IV.7	Perspectives . . . . .	184
	Conclusions . . . . .	185
<b>V</b>	<b>Expérimentations et résultats</b>	<b>187</b>
	Introduction . . . . .	187
V.1	Protocole de test . . . . .	188
V.1.1	Scénarios de test . . . . .	189
V.2	Résultats . . . . .	191
V.3	Analyse et discussion . . . . .	191
	Conclusions . . . . .	191
<b>Conclusions</b>		<b>193</b>
<b>Bibliographie</b>		<b>195</b>

# **Introduction**

Introduction générale



# Chapitre I

## Applications aux terminaux à conteneurs

### Introduction

Avec le développement des échanges commerciaux qui n'ont cessé d'augmenter (de moins de 6 millions de EVP<sup>1</sup> en 1970 à 500 millions de EVP en 2008)<sup>2</sup>, le conteneur est devenu la forme de conditionnement la plus répandue. Ces boîtes métalliques ont été inventées par Malcolm McLean dans les années 1930, puis normalisées d'abord par l'*American National Search Institute* (anciennement l'ASA), puis par l'*International Organization for Standardization* (ISO). Les dimensions principalement rencontrées sont les conteneurs de 20 et 40 pieds :

- longueur : 20 pieds (6,096m) ou 40 pieds (12,192m)
- largeur : 8 pieds (2,438m)
- hauteur : 8,5 pieds (2,591m)

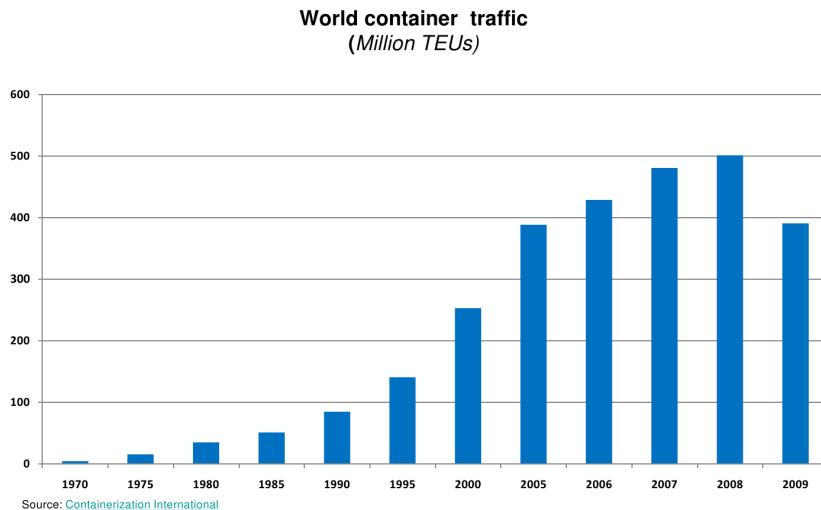
C'est cette normalisation qui est la cause du succès du conteneur vis-à-vis des autres modes de conditionnement. Il devient aisément de les transporter, de les manipuler et de les stocker.

### Identification des conteneurs

Concernant l'identification des conteneurs, un numéro permet de les distinguer. Il s'agit d'un code décrit par la norme ISO 6346. Ce numéro est appelé numéro BIC<sup>3</sup> et permet ainsi

---

1. Équivalent Vingt Pieds, en anglais *TEU* : *Twenty feet Equivalent Unit*. Un conteneur de 40 pieds vaut par exemple 2 EVP.
2. Source : Containerization International (<http://www.ci-online.co.uk>)
3. Bureau International des Conteneurs et du Transport Intermodal : [www.bic-codes.org](http://www.bic-codes.org)



**Figure I.1:** Trafic mondial de conteneurs (en millions d' EVP). Source : International Transport Forum, OECD, p1.

d'identifier de façon unique 90% des conteneurs du trafic mondial.

### Structure du code BIC

Il est composé de quatre lettres puis de six chiffres et enfin d'un dernier chiffre de contrôle permettant de vérifier qu'un numéro de série a été transmis correctement lors d'une transmission à distance.



Conteneur RWTU 961916 2

Les trois premières lettres correspondent au code du propriétaire du conteneur. La quatrième lettre identifie la catégorie du conteneur :

- U : conteneur de fret ;
- J : équipement détachable relatif à un conteneur de fret ;
- Z : grues et châssis ;
- R : conteneurs réfrigérés.

Les six chiffres suivants correspondent au numéro de série du conteneur. Le dernier chiffre sert de contrôle de validité du code et est calculé en 4 étapes :

**Étape 1 :** un numéro est affecté à chaque lettre de l'alphabet en commençant à 10 pour la lettre A et en omettant les multiples de 11 et les chiffres de 0 à 9 conservent leur valeur :

A	B	C	D	E	F	G	H	I	J	K	L	M
10	12	13	14	15	16	17	18	19	20	21	23	24

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
25	26	27	28	29	30	31	32	34	35	36	37	38

**Étape 2 :** chacun de ces nombres est multiplié par  $2^{\text{indice}}$  où l'indice est la position du nombre dans le numéro BIC de gauche à droite en partant de 0.

**Étape 3 :**

- a) additionner tous les nombres obtenus après l'étape 2.
- b) diviser ce résultat par 11.
- c) supprimer la partie décimale du résultat
- d) multiplier la partie entière ainsi obtenue par 11.
- e) soustraire le résultat de d) du résultat de a)

**Exemple :**

**Étape 1 :** le conteneur RWTU 961916 correspond aux valeurs suivantes :

R	W	T	U	9	6	1	9	1	6
29	35	31	32	9	6	1	9	1	6

**Étape 2 :**

$29 * 2^0$	$35 * 2^1$	$31 * 2^2$	$32 * 2^3$	$9 * 2^4$	$6 * 2^5$	$1 * 2^6$	$9 * 2^7$	$1 * 2^8$	$6 * 2^9$
$29 * 1$	$35 * 2$	$31 * 4$	$32 * 8$	$9 * 16$	$6 * 32$	$1 * 64$	$9 * 128$	$1 * 256$	$6 * 512$
29	70	124	256	144	192	64	1152	256	3072

**Étape 3 :**

- a)  $29 + 70 + 124 + 256 + 144 + 192 + 64 + 1152 + 256 + 3072 = 5359$
- b)  $5349 / 11 \simeq 487.1818182$
- c) la partie entière du résultat vaut 487
- d)  $487 * 11 = 5357$ .
- e)  $5359 - 5357 = 2$

⇒ Le chiffre de contrôle de ce conteneur est donc 2.

Les conteneurs sont ainsi chargés sur des navires géants appelés porte-conteneurs permettant de transporter par voie maritime des milliers de tonnes de biens aux quatre coins du monde. L'avantage majeur des conteneurs réside dans leur facilité de transbordement et c'est pourquoi



**Figure I.2:** Le succès de la conteneurisation a conduit à détourner l'utilisation principale du conteneur. Ici, la nouvelle cité universitaire du Havre.

un très grand nombre de plate-formes multimodales ont été créées afin de faciliter le transfert des conteneurs entre les différentes voies de transport que sont la mer, le rail et la route. Afin de réduire les durées ainsi que les coûts induits par ces transbordements, les ports doivent sans cesse innover en développant de nouvelles technologies ou en instaurant de nouveaux procédés.

Dans ce chapitre nous détaillerons la structure et l'organisation d'un terminal portuaire à conteneurs et nous préciserons le type d'équipement présents dans ces ports. D'autre part, nous étudierons les différentes parties du terminal où une optimisation permet d'améliorer la performance de la plate-forme. Nous apporterons une attention particulière aux déplacements des conteneurs à l'intérieur du port entre les différentes zones d'échanges ainsi qu'à l'ordonnancement et à l'affectation aux véhicules des conteneurs à déplacer.

## I.1 Structure et organisation des terminaux à conteneurs

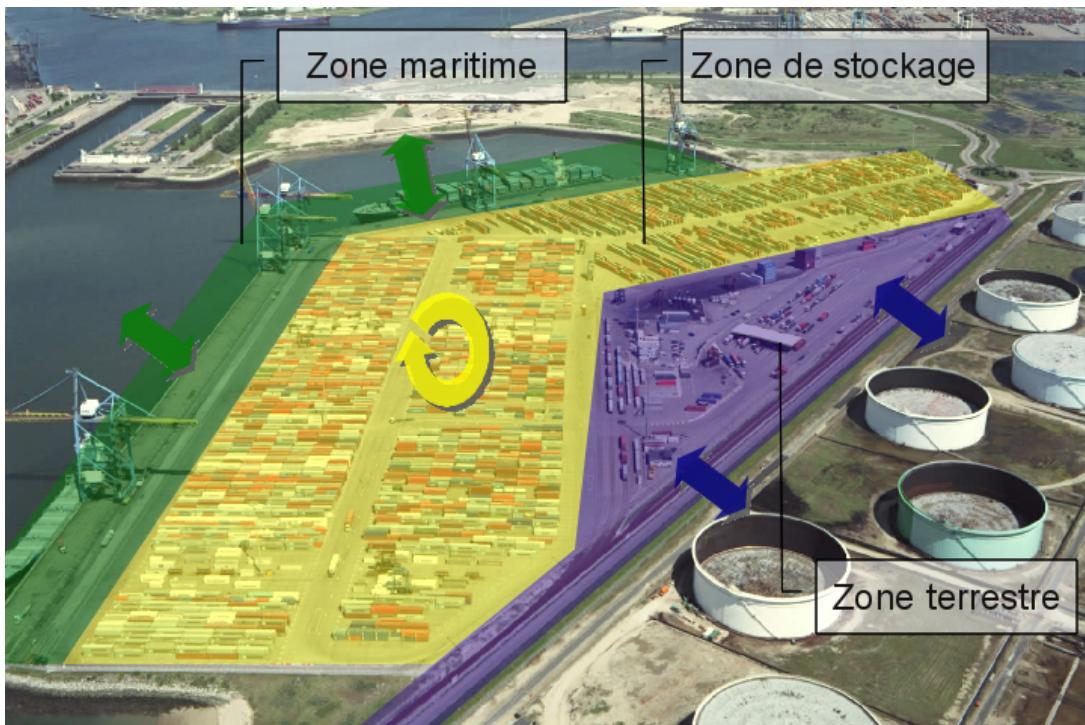
Dans la pratique, lorsqu'un navire arrive au sein du terminal, un espace d'accostage lui est attribué. Une fois à quai, les conteneurs sont déchargés et chargés à bord grâce à des grues. Les conteneurs ainsi déchargés sont placés dans des espaces de stockage dans l'attente d'être ré-acheminés via d'autres navires, trains, ou camions. Les conteneurs chargés à bord du navire proviennent également de l'espace de stockage du terminal suite au déchargement de navires, de trains ou de camions. Les clients sont donc des transporteurs maritimes ou terrestres et le

terminal constitue la plate-forme d'échange entre ces différentes voies de transport. Ce sont les caractéristiques de ces échanges qui déterminent la structure et l'organisation d'une telle plate-forme.

### I.1.1 Structure

Schématiquement, un terminal portuaire à conteneurs est divisé en cinq parties :

- La zone de stockage (*yard*) : composée de travées de conteneurs empilés sur plusieurs niveaux ;
- Le dépôt : où les véhicules de manutention sont garés ;
- La zone des camions : où les camions apportent ou viennent récupérer les conteneurs ;
- La zone des trains : où les wagons sont acheminés pour être déchargés et/ou chargés ;
- Les quais : où les porte-conteneurs sont arrimés.



**Figure I.3:** Les trois principales zones d'échange du Terminal de Normandie

La zone de stockage est divisée en blocs. Chaque bloc comporte une série de travées où sont empilés les conteneurs. La hauteur maximale d'empilement varie en fonction des terminaux. Il y a trois étages de conteneurs dans la zone de stockage des terminaux du port du Havre. La zone des trains comporte plusieurs rangées de rails souvent entrecoupées par des routes afin de permettre aux engins de manutention de passer sans devoir faire le tour du train entier.

La zone des camions est constituée à l'entrée du terminal par des guichets où les conducteurs annoncent leur arrivée aux portes du terminal et leur ordre de chargement/livraison. Puis à l'intérieur du terminal des zones camions sont réparties afin d'accueillir les véhicules en attente de (dé)chargement. La zone des quais comporte des portiques, le plus souvent mobiles permettant de décharger les conteneurs des navires sur les quais ou inversement de charger les conteneurs des quais sur les navires. Il est possible également que des camions stationnent sous un portique afin de recevoir directement un conteneur du navire déchargé, ou de décharger son conteneur directement dans le navire.

### I.1.2 Engins de manutention

Dans [Steenken et al., 2004], les auteurs distinguent deux catégories de véhicules pouvant être rencontrés au sein d'un terminal à conteneurs : d'une part les véhicules des clients (trains, camions et navires) et d'autre part les engins de manutention du terminal. Le transport des conteneurs au sein du terminal peut être soit vertical, soit horizontal. Le transport vertical concerne les opérations de levage et de dépose des conteneurs. Les engins concernés sont les portiques de quai ou de stockage. Ces engins sont la plupart du temps mobiles (soit sur rail, soit sur pneus) ce qui permet de les déplacer le long du quai ou le long des travées de stockage (voir fig. I.4).



**Figure I.4:** Portique du port du Havre, mobile sur rail, effectuant le déchargement d'une barge  
(source : <http://www.t-n.fr>)

Au contraire, le transport horizontal concerne les véhicules capables de déplacer un conteneur d'un endroit à un autre à l'intérieur du terminal. Les conteneurs doivent ainsi être déplacés entre la zone des camions et le yard, entre la zone des trains et le yard ainsi qu'entre les quais et

le yard (et réciproquement du yard vers les zones des camions, trains et navires). Il existe à l'heure actuelle deux procédés de gestion de ces déplacements, en fonction du type de véhicules utilisés : passifs ou actifs. Les véhicules passifs correspondent au procédé le plus moderne qui consiste à utiliser des véhicules automatiques (*Automated Guided Vehicles* ou *AGV*) capables de transporter des conteneurs entre les différentes zones de manutention. Un système de câbles permet de délimiter les couloirs de circulation de ces engins qui sont ensuite capables de s'orienter automatiquement. Ces véhicules sont dis passifs car ils ne sont pas autonomes dans le sens où ils ne sont pas capables de charger ou décharger un conteneur par eux-même. Ils doivent attendre d'être chargés ou déchargés par une grue. Ainsi, ces véhicules requièrent la présence de portiques dans toutes les zones du terminal. Dans les ports utilisant cette technologie, des portiques sont utilisés pour empiler et dépiler les conteneurs des travées. Les AGV se garent sous le portique afin d'être chargés ou déchargés.



**Figure I.5:** Automated Guided Vehicle (AGV) de marque MES (source : mescranes.com)

Les véhicules actifs d'autre part, sont pilotés par des opérateurs et capables à la fois de transporter et de charger ou décharger un conteneur sans l'aide d'un portique. Il existe trois principaux engins de ce type :

- *Forklift Truck* (chariots élévateurs) : ces chariots élévateurs permettent uniquement de manipuler des conteneurs vides. Ils servent donc à déplacer ces conteneurs dans une partie particulière du terminal où ils sont stockés ;
- *Reach Stacker* (chariots empileurs) : ces chariots sont utilisés pour charger et décharger les trains. La prise se fait par le côté, ce qui reprend l'avantage du chariot élévateur mais en permettant de soulever une charge beaucoup plus lourde ;
- *Straddle Carrier* (chariots cavaliers) : ces chariots sont les plus puissants. La prise se fait par le dessus. Ils sont capable d'enjamber une pile de conteneurs de trois étages et permettent ainsi de se déplacer dans les travées.

Ce sont ces véhicules qui sont utilisés pour la manutention des conteneurs au sein des terminaux du port du Havre. Certains chariots sont capables d'adapter dynamiquement la longueur de leur pince afin de s'adapter aux différents types de conteneurs. D'autres chariots nécessitent de rentrer au dépôt afin qu'un mécanicien modifie l'écartement des pinces.



**Figure I.6:** Engins de manutention utilisés par les Terminaux de Normandie (de gauche à droite) : chariot cavalier, chariot élévateur et chariot empileur (source : <http://www.t-n.fr>)



**Figure I.7:** Chariots cavaliers

Le contexte applicatif de ce travail est défini par les caractéristiques du port du Havre dont les opérateurs de manutention n'utilisent pas de véhicules automatiques, c'est pourquoi il ne sera question que de l'optimisation des déplacement des chariots et non pas des véhicules automatiques. Il est à noter qu'il ne sera fait référence qu'aux chariots cavaliers dans la suite de ce manuscrit tout en gardant à l'esprit que les mêmes procédés s'appliquent également aux autres chariots (*forklift trucks* et *reach stackers*).

### I.1.3 Réseau routier

Les différents blocs du terminal sont reliés grâce à un réseau constitué de routes et de carrefours. Ce réseau peut être modélisé par un graphe orienté  $G = (V, A)$  où  $V$  est l'ensemble des carrefours et où  $A$  est l'ensemble des routes. Les travées de conteneurs constituent des routes praticables par les chariots cavaliers. Cependant, le fait d'enjamber les conteneurs rend

impossible à deux chariots de se croiser ou de se doubler dans une même travée. L'espace-ment entre deux travées peut de même empêcher deux chariots de se croiser dans deux travées connexes. Ces routes spécifiques peuvent être modélisées par des arcs First-In-First-Out (voir [Orda and Rom, 1990]). Ces arcs garantissent que le premier véhicule utilisant l'arc sera le pre-mier à en ressortir. Cette spécificité des arcs FIFO a pour conséquence de provoquer des temps d'attente potentiels des chariots en entrée de travée. Ceci peut avoir un impact lourd sur les per-formances du routage des chariots et doit être pris en compte avec attention. En effet, certaines zones du réseau routier du terminal sont faiblement connexes et peuvent être rapidement sa-turées. Un routage efficace consistera principalement à ne plus considérer la distance parcourue comme unique critère d'optimisation mais également le temps de parcours.

#### I.1.4 Activités des chariots cavaliers

Les chariots cavaliers sont utilisés pour déplacer les conteneurs à l'intérieur du terminal. Chaque déplacement est appelé "mission". Lorsqu'un chariot n'effectue aucune mission, on dit qu'il est "libre". Il stationne alors dans une zone spécifique du terminal appelée "dépôt". Chaque mission comporte quatre phases. D'abord une phase de déplacement vers le point de collecte, puis la phase de chargement du conteneur sur le chariot, ensuite une nouvelle phase de déplacement mais cette fois vers l'emplacement de livraison, et enfin, la phase de déchargement du conteneur du chariot.

##### I.1.4.1 Les types de missions

Une mission peut être de quatre types :

- Entrée : un conteneur doit être stocké dans le yard
- Sortie : un conteneur doit être amené du yard vers une zone d'échange (camion, train ou quai)
- Transbordement : un conteneur doit être déplacé d'un quai à un autre
- Réorganisation : un conteneur doit être déplacé à l'intérieur du yard

La première catégorie de mission concerne le déchargement des camions, trains ou porte-conteneurs. Le chariot se rend sur l'emplacement de collecte du conteneur (au dessus de la remorque du camion, du wagon, ou sous le portique de quai), prend le conteneur concerné par la mission, puis se dirige vers l'emplacement de livraison à l'intérieur du yard et y dépose le conteneur. La seconde catégorie de mission concerne l'opération inverse, c'est-à-dire le charge-ment des camions, trains ou porte-conteneurs. Le chariot se rend sur l'emplacement de collecte

du conteneur dans le yard, prend le conteneur concerné, puis se dirige vers l'emplacement de livraison (au dessus de la remorque du camion, du wagon, ou sous le portique de quai) et dépose le conteneur sur le sol dans le cas d'un quai, ou sur la remorque du camion, ou encore sur le wagon du train. La troisième opération concerne le chargement des conteneurs d'un navire sur un autre porte-conteneurs. Le chariot doit alors récupérer le conteneur de la mission sous le portique de quai, puis se diriger vers le portique de chargement du navire à charger et y déposer le conteneur qui sera acheminé à bord par la grue. Enfin, l'opération de réorganisation consiste à déplacer les conteneurs à l'intérieur de la zone de stockage afin de préparer l'arrivée (la sortie) de conteneurs du terminal. Cette opération permet d'éviter d'avoir au dernier moment à dépiler un certains nombre de conteneurs pour accéder à un conteneur en bas d'une pile par exemple.

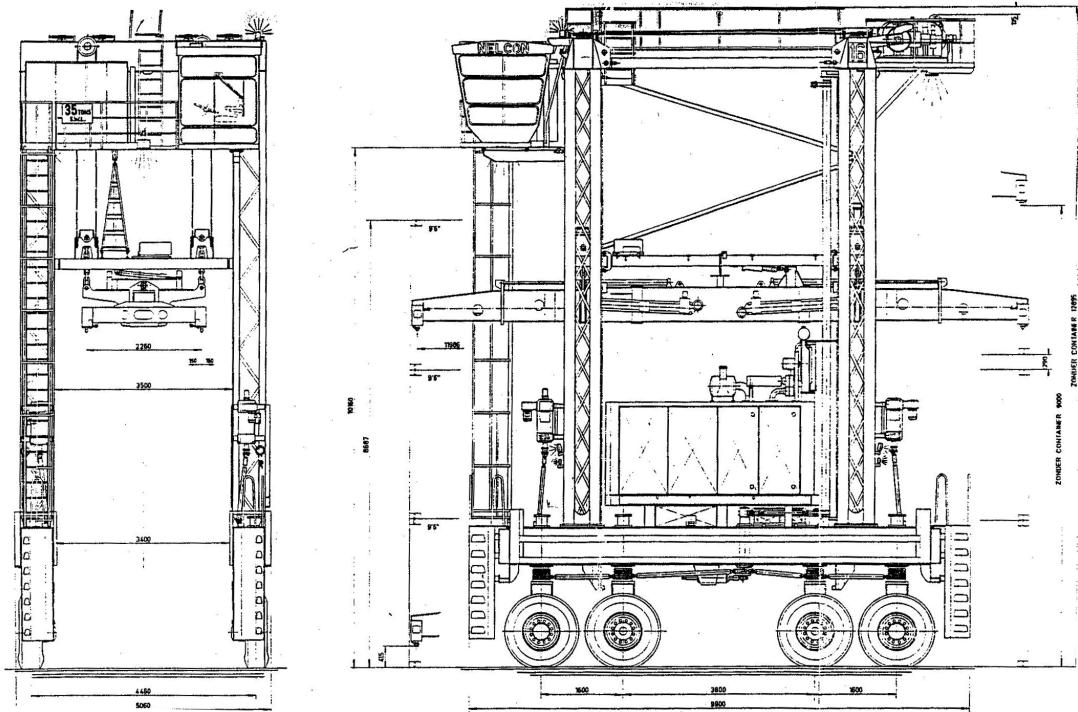
#### I.1.4.2 Fenêtres de temps

Deux fenêtres de temps sont attribuées à chaque mission. Ces fenêtres indiquent les périodes de temps à respecter d'une part pour collecter le conteneur, et d'autre part pour le livrer. Si un chariot cavalier arrive avant le début de la fenêtre de temps, il devra attendre. S'il arrive après la fin de la fenêtre de temps, c'est le véhicule client (camion, train, porte-conteneur) qui devra attendre. Cette attente des clients peut provoquer des frais supplémentaires pour le terminal qui devra payer des pénalités de retard. Les fenêtres de temps peuvent être ainsi strictes (*hard*) ou lâches (*soft*). Ainsi, les fenêtres impliquant un client seront strictes afin de réduire les coûts, alors que les fenêtres de temps n'impliquant que des véhicules du terminal seront lâches. En effet, lors d'une mission de chargement de camion par exemple, la collecte du conteneur à livrer se situe dans le yard. Si le chariot manque sa fenêtre de temps pour la collecte, cela n'entraînera pas de pénalité à payer pour la société qui gère le terminal. La fenêtre de collecte est donc lâche. En revanche la livraison du conteneur concerne un camion et le non respect de la fenêtre de temps entraînera des frais de pénalité à payer pour la société. La fenêtre de temps de livraison sera donc stricte.

#### I.1.4.3 Objectifs des chariots cavaliers

Un chariot cavalier mesure environ 15 mètres de haut pour 10 mètres de long et 5 mètres de large et pèse près de 60 tonnes à vide. Capable de déplacer une charge de 50 tonnes à la vitesse de 20 km/h, un chariot peut se déplacer à vide jusqu'à 40 km/h. Sa consommation de carburant dépasse les 20 litres/heure.

Ces caractéristiques techniques font qu'un chariot est extrêmement coûteux à la fois à l'achat



**Figure I.8:** Plan d'un chariot cavalier utilisé par le port du Havre dans les années 1990

(plus de 800000 €) et à l'utilisation (voir [Huang and Chu, 2003]). De plus, le pilotage de ces engins requiert une main d'œuvre de haute qualité dotée d'une grande précision afin de réaliser des manœuvres complexes impliquant de très lourdes charges en toute sécurité. Afin de conserver une concentration et une précision accrue, ces pilotes ne peuvent conduire un chariot cavalier que pendant une période donnée dépendant du terminal concerné. Puis une phase de repos doit être respectée avant que le conducteur puisse reprendre son travail. Pour le terminal, cette main d'œuvre est un coût conséquent et doit être gérée de façon optimale.

Du point de vue de la société de manutention, l'objectif est comme pour toute entreprise de services de dégager des profits. Il est donc important de chercher à réduire les coûts et ceci peut être réalisé en partie en minimisant le nombre de chariots à utiliser et en cherchant à réduire au minimum les distances parcourues par la flotte de chariots cavaliers.

Au niveau d'un chariot cavalier, la qualité de service offerte aux clients dépendra principalement de deux facteurs. D'une part, le soin pris par le conducteur du chariot pour charger/décharger un conteneur, et d'autre part le respect des fenêtres de temps de la mission. Les caractéristiques temporelles évoquées dans le paragraphe précédent doivent être prises en compte lors du calcul de l'ordonnancement des missions.

Pour toutes ces raisons et à la fois au niveau du manutentionnaire et au niveau des chariots

cavaliers, les objectifs sont double. Pour le manutentionnaire, il est nécessaire de minimiser la taille de la flotte d'engins de manutention, de mains d'œuvre... tout en maximisant la qualité de service. À l'échelle des chariots cavaliers, les objectifs sont de minimiser la distance parcourue tout en minimisant le dépassement des fenêtres de temps.

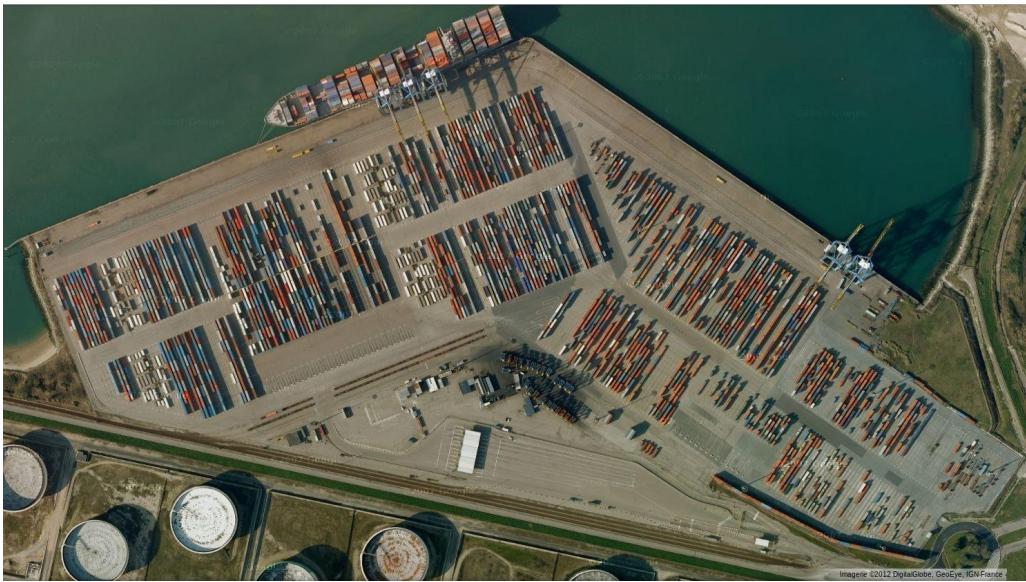
### I.1.5 Le Terminal de Normandie

Le Terminal de Normandie est l'un des terminaux du port du Havre. Il est situé dans l'estuaire de la Seine dans un bassin à marée et est délimité au Nord par deux quais. Le quai d'Asie (au Nord-Ouest) et le quai d'Osaka (au Nord-Est) offrant ainsi à tous les deux 1075 mètres d'accostage et une profondeur de 14 mètres. La superficie totale du terminal est de 40 Hectares et sa capacité annuelle est de 450000 EVP.

Le graphe représentant le réseau routier interne du terminal comporte 1170 sommets, 170 routes et 531 travées pour un total de 3500 emplacements de stockage de conteneurs. La zone de stockage est composée de douze blocs permettant aisément de disposer les conteneurs à proximité de leur future zone de transit. La zone des trains comporte trois voies ferrées différentes entrecoupées sur certaines parties afin de permettre le passage des engins de manutention. Enfin, il existe trois différentes zones de collecte et de livraison pour les camions.

Ce terminal a été créé en 1990 afin de contenir le flux d'échanges de conteneurs qui n'a cessé d'augmenter de façon de plus en plus importante. À l'époque de sa construction, ce terminal était capable d'accueillir les plus grands navires porte-conteneurs. En revanche, de nouveaux navires géants aux dimensions impressionnantes ont été construits depuis et sont gérés dans un autre terminal du port du Havre construit plus récemment.

Cinq portiques de quais et une vingtaine de chariots cavaliers assurent la manutention des conteneurs à l'intérieur du terminal où aucun portique n'est utilisé dans la zone de stockage. Les chariots cavaliers sont les seuls engins autorisés à circuler et à manipuler des conteneurs dans les blocs de travées. La régulation du trafic à l'intérieur du terminal est donc un point critique concernant la performance du terminal c'est pourquoi une bonne gestion des engins de manutention est primordiale afin d'assurer une qualité de service suffisante aux clients du terminal.



**Figure I.9:** Image satellite du Terminal de Normandie, Port Autonome du Havre (source : Google Maps)

## I.2 Optimisation des terminaux à conteneurs

### Introduction

Le Terminal de Normandie est un exemple de terminal portuaire à conteneurs. De façon générale, un tel terminal est une plate-forme logistique où les échanges doivent être réalisés dans le respect de certains critères (emplacements de stockage, fenêtres de temps, ressources humaines, etc.). Un certain nombre d'éléments peuvent ainsi être optimisés afin de répondre au mieux aux besoins des clients tout en maximisant les profits pour le terminal.

Les terminaux sont en concurrence les uns avec les autres. Les navires choisissent les ports leur permettant de décharger leur cargaison et de charger de nouveaux conteneurs le plus rapidement possible et pour le coût le plus faible possible. Les ports de Rotterdam, de Hambourg, d'Anvers, de Brême et du Havre sont ainsi en concurrence pour desservir le Nord de l'Europe. Il est donc primordial pour ces ports de réduire les coûts d'exploitation tout en assurant une productivité élevée.

Parmi les nombreuses sources d'optimisation possibles dans les terminaux à conteneurs citons :

- la structure du terminal ;
- l'allocation des berges aux navires ;
- l'allocation des grues de quai ;

- les plans de chargement des navires ;
- le transbordement ;
- la gestion des conteneurs vides ;
- la gestion des effectifs ;
- le routage des véhicules ;
- le stockage (allocation des travées et des blocs) ;
- les transferts (quai - stockage, stockage - stockage, stockage - terre) ;
- l'allocation des engins de manutention.

On retrouve ces problèmes dans la littérature en tant que *Berth Allocation Problem*, *Quay Cranes Scheduling Problem*, *Ship Loading Plan Problem*, *Shortest Path Problem*, *Block and Bay Allocation Problem*, *Vehicle Routing Problem* et *Job Scheduling Problem*.

### I.2.1 Structure du terminal

Comme indiqué au paragraphe II.1.1 le terminal est formé de cinq zones dont l'organisation et la disposition jouent un rôle clef pour la performance, en particulier pour les engins de manutention. En effet, une zone de stockage de grande dimension peut entraîner un éloignement important d'une partie des blocs par rapport à certaines zones de transfert (quais, rails, zones des camions). Un quai éloigné d'une zone de stockage destinée aux transferts sur les camions engendrera des durées de déplacement plus importants mais permettra par exemple de raccourcir la distance vis-à-vis de la zone de stockage destinée aux trains.

De même, les engins de manutention doivent la plupart du temps retourner au dépôt en cas d'inactivité. La position du dépôt est donc essentielle afin de minimiser la distance à parcourir pour se rendre du dépôt aux points de collecte des conteneurs, puis pour revenir du point de livraison vers le dépôt.

Tout est affaire de compromis entre le positionnement des composantes du terminal ainsi qu'entre le choix de ces éléments et de leur quantité. Ainsi, les plus grands terminaux disposent de quais immenses ce qui entraîne une distance importante entre l'extrémité d'une berge et le centre du terminal. La plupart de ces terminaux géants sont équipés en conséquence et utilisent des portiques dans les zones de stockage et des engins automatiques pour transporter les conteneurs entre les différentes zones.

Malgré leurs dimensions, ces terminaux peuvent rencontrer des difficultés au niveau des transferts comme l'illustre le terminal géant de Yangshan (voir Fig. I.10), situé à Shanghai en Chine qui est un exemple de ces paradoxes entre la dimension d'un terminal et sa performance.

Il est le plus grand terminal au monde à l'heure actuelle. Même si la construction n'est pas encore terminée, ses quais s'étendent déjà sur six kilomètres de long et comportent soixante portiques. Il a cependant été construit sur une île et a été relié à la métropole chinoise par le *Donghai Bridge*, pont de plus de trente cinq kilomètres de long, seul point le reliant au continent et limite importante pour le trafic. De plus, ce terminal rentre en compétition avec les terminaux déjà existant de Shanghai. Ainsi, les coûts d'utilisation seront très élevés ce qui rend le terminal peu attractif vis-à-vis des autres terminaux de la ville. L'intérêt majeur du terminal réside dans la profondeur de ses quais qui lui confère la possibilité d'accueillir les plus gros porte-conteneurs actuels. Grâce à cette caractéristique et après étude de la structure du terminal et plus particulièrement de son isolement par rapport à la métropole, et si aucune nouvelle infrastructure n'est construite, le terminal de Yangshan apparaît clairement destiné à des opérations de transbordement.



**Figure I.10:** Terminal de Yangshan à Shanghai en Chine (source : Google Earth)

La dimension et l'organisation interne d'un terminal à conteneurs doit donc être étudiée avec le plus grand soin car elle déterminera en grande partie la performance et l'attractivité du terminal.

### I.2.2 Allocation des berges et transbordement

Le problème d'affectation des berges (*Berth Allocation Problem* ou *Berth Scheduling Problem*) est l'un des problèmes d'optimisation logistique les plus connus. Il s'agit d'une variante du problème d'affectation quadratique (*Quadratic Assignment Problem*). Il consiste à déterminer l'affectation d'une partie d'un quai à un navire pendant une période donnée de façon à minimiser la durée de service associée au navire (et donc indirectement la distance des déplacements des conteneurs à charger sur le bateau, ou à décharger du bateau, ainsi que les coûts de manutention associés). Tout comme le problème d'affectation quadratique, le problème d'allocation

des berges est NP-Difficile.

Ce problème peut se définir dans le cas discret (les quais sont découpés en portions et les navires sont accostés sur un ou plusieurs segments contigus du quai) ou dans le cas continu (le quai n'est pas découpé en parties et les navires peuvent y accoster n'importe où), dans le cas statique (tous les navires sont présents au début du calcul) ou dans le cas dynamique (une partie des navires peuvent être présents au début des calculs, puis les autres arrivent en cours de planification).

Lorsqu'un navire désire accoster sur le terminal il faut lui attribuer une position d'accostage en fonction de la place disponible sur le quai au moment de son arrivée et jusqu'à son départ d'une part, et des contraintes de transfert des conteneurs à charger et décharger. On distingue trois types d'opérations.

Le transbordement désigne l'opération de manutention consistant à décharger un conteneur d'un navire puis à le charger sur un autre. Dans ce cas, il est nécessaire de déterminer conjointement la position d'accostage des deux navires sur le quai afin de minimiser le temps de transfert des conteneurs.

Lorsqu'un conteneur doit être chargé sur le navire, il faut prendre en compte la distance entre sa position de stockage dans le yard et la position où sera accosté le navire.

Lorsqu'un conteneur doit être déchargé du navire pour être stocké dans le yard, il faut déterminer la position de stockage de ce conteneur en fonction de sa destination future (navire, train ou camion) et chercher un emplacement à la fois proche de la destination et proche du quai où sera accosté le navire. Calculer la position d'accostage du navire revient donc à calculer les emplacements de stockage des conteneurs à décharger de façon à minimiser les déplacements.

### I.2.3 Allocation des grues de quai

Le problème d'affectation des portiques de quai (*Quay Crane Scheduling Problem*) est le problème rencontré une fois le problème de l'affectation du quai résolu, ou du moins en partie résolu. En effet, les portiques de quai (*Quay Crane, Gantry Crane, Container Crane* ou *Ship-to-Shore Crane*) sont des engins de manutention sur rails pouvant mesurer jusqu'à 60 mètres de haut. De tels équipements sont extrêmement onéreux et bien souvent les terminaux n'en comportent qu'un nombre restreint.

Allouer plusieurs portiques au déchargement d'un navire permet de raccourcir la durée de service du navire, mais le portique supplémentaire affecté ne pourra pas être utilisé pour les autres navires accosté au même instant et donc le temps de service des autres navires augmentera.

Les portiques étant posés sur des rails il est impossible de déplacer une grue d'un bout à l'autre du quai si un autre portique se trouve déjà sur le quai. L'affectation d'un portique devra tenir compte de cette caractéristique.

De même, l'affectation des grues de quai devra également être prise en compte lors de l'affectation de la berge au porte-conteneurs afin de permettre de déterminer son temps de service.

#### I.2.4 Positionnement des conteneurs

Une fois les navires accostés, les portiques affectés au porte-conteneurs se chargent de décharger les conteneurs sur le quai. Les conteneurs doivent dans un second temps être acheminés vers un emplacement de stockage en attendant d'être transférés vers un wagon ou un camion ou un autre navire. Le choix de cet emplacement doit tenir compte de deux facteurs. D'une part la provenance du conteneur, c'est-à-dire ne pas être situé trop loin du quai dans le cas d'un conteneur déchargé par un portique, ou de la zone du camion ou du train dans le cas d'un conteneur apporté par la route ou par le rail. D'autre part, l'emplacement doit être déterminé en tenant compte de sa destination. Si le conteneur doit être chargé sur un wagon alors il devra être stocké dans une travée d'un bloc proche de la zone où le train sera à quai. Si le conteneur doit être chargé sur un camion alors d'une part le camion sera dirigé vers une zone d'échange proche du point d'origine du conteneur, et le conteneur sera stocké dans une travée d'un bloc proche de cette zone. Si le conteneur est destiné à être chargé sur un navire, alors il devra être stocké dans la travée d'un bloc à proximité de la portion de quai affectée au navire sur lequel le conteneur devra être chargé.

Il existe également une position verticale à déterminer pour le conteneur. En effet, les conteneurs peuvent être empilés dans les travées sur plusieurs niveaux. Placer un conteneur devant rester dans le terminal longtemps sur le sommet d'une pile alors que les autres conteneurs de cette pile devront être déplacés rapidement engendrera des déplacements supplémentaires. Ces opérations de déplacement consistent un peu comme à la manière du jeu de Hanoï à déplacer temporairement les conteneurs d'une pile sur une autre pile afin de libérer le conteneur posé au sol, puis à reposer les conteneurs supérieurs sur la pile d'origine. Il est donc important de prendre en compte les fenêtres de temps des missions associées à un conteneur (mission d'entrée du conteneur sur le terminal et mission de sortie du conteneur du terminal) afin de déterminer son emplacement.

Les dates d'entrée et de sortie d'un conteneur sont prises en compte également afin de

favoriser l'accès aux emplacements proches des zones d'échanges (camions, trains, navires) aux conteneurs transitant peu de temps par le terminal. En revanche, les conteneurs devant rester sur le terminal plus longtemps seront placés plus loin sur les travées de ces zones, ou du moins seront positionnés sur le sol, à la base d'une pile.

Le choix d'un emplacement peut s'avérer difficile si le terminal est saturé. Il peut parfois être nécessaire de déplacer un conteneur à l'intérieur même du yard afin de libérer l'emplacement pour un autre conteneur. Un tel déplacement mobilise un engin de manutention pendant une durée parfois importante et le rend donc inutilisable pour d'autres missions.

Dans le cas statique, où toutes les entrées/sorties de conteneurs sont connues à l'avance, il est possible de déterminer le positionnement optimal des conteneurs afin de minimiser les distances ou les durées de service des clients. En revanche, dans le cas dynamique, les ordres de mission ne sont que partiellement connus au moment de la prise de décision et d'autres missions ne seront connues que plus tard, pendant le déroulement de la journée d'exploitation du terminal. Il est dans ce cas impossible de garantir le positionnement optimal des conteneurs. Tout l'enjeu de la résolution de ce problème consiste à déterminer la position optimale des conteneurs à un instant fixé en prenant en compte un facteur de risque lié à la méconnaissance des données.

### I.2.5 Routage des véhicules

Raccourcir les durées de déplacement entre deux points du terminal permet de réduire le temps de service d'un client. Il est possible d'agir sur trois facteurs :

- la vitesse de circulation des engins de manutention ;
- la limitation de vitesse sur le terminal ;
- le chemin suivi par le véhicule.

Le premier facteur dépend des caractéristiques techniques des véhicules, mais tout comme le second facteur, il a un impact sur la sécurité des personnes et des marchandises sur le terminal. L'augmentation de la vitesse d'un engin comme un chariot cavalier n'est pas envisageable. Elle demanderait aux conducteurs un effort intense pour maîtriser les soixante tonnes à vide de l'engin et risquerait de provoquer des accidents aux conséquence lourdes (le conducteur est positionné au sommet de l'engin, soit à plus de dix mètres du sol).

L'unique moyen de chercher à réduire les durées de déplacements au sein du terminal indépendamment de la position des conteneurs sur le yard est d'optimiser le guidage des véhicules. Il s'agit là de l'un des objectifs de ce travail de thèse.



**Figure I.11:** Accident d'un chariot cavalier sur le port d'Essex (Angleterre) le 23 octobre 2007

#### I.2.5.1 Modélisation et objectifs

Dans un problème de routage, un objectif courant est de minimiser la distance à parcourir entre une source et une destination. Comme décrit précédemment, le réseau routier d'un terminal constitue un graphe dont les nœuds sont les carrefours du terminal et dont les arcs sont les routes et les travées de conteneurs.

Les chariots cavaliers ont pour objectif d'atteindre des zones de collecte ou de livraisons de conteneurs pendant une fenêtre de temps définie à l'avance. Un routage efficace pour un chariot cavalier consiste donc à atteindre sa destination à l'heure prévue.

Le graphe routier du terminal comporte deux types d'arcs. Si les arcs représentant les routes peuvent facilement absorber un flot de véhicules, les arcs *FIFO* modélisant les travées de conteneurs constituent des points critiques sur le graphe. En effet, les chariots cavaliers enjambent les conteneurs empilés dans les travées et ne peuvent donc pas se croiser à l'intérieur de celles-ci. Si de tels blocages se produisent ils peuvent causer des retards importants pour la réalisation d'une mission.

Le routage des chariots cavaliers revient à calculer des itinéraires en prenant en compte les temps d'attente en entrée de travées afin de minimiser à la fois la durée totale du parcours (déplacement et attente), le coût de déplacement (consommation de carburant), et l'écart de temps entre la date d'arrivée du chariot et la fenêtre de temps.

#### I.2.5.2 Dynamique du graphe

Le graphe routier du terminal est donc un graphe partiellement *FIFO* mais ce n'est pas sa seule caractéristique. Ainsi, la durée de parcours d'un arc dépend à la fois de la distance à parcourir (longueur de l'arc) mais également du temps.

En effet, un chariot cavalier ne se déplace pas à la même vitesse lorsqu'il transporte un conteneur que lorsqu'il se déplace à vide. Sur le Terminal de Normandie, la vitesse d'un chariot cavalier est de 27km/h à vide contre 23 km/h en charge. Il est également à noter qu'un tel véhicule met environ 30 secondes afin d'atteindre les 20 km/h. D'autre part la vitesse de circulation d'un chariot cavalier est différente en fonction du type de voie empruntée. Ainsi les chariots se déplacent très lentement dans les travées et plus rapidement sur les routes du terminal.

La durée de parcours d'un arc dépend ainsi directement de la vitesse du chariot cavalier, du nombre de décélérations et d'accélérations réalisées, et également du trafic sur l'arc au moment de la traversée. Les durées d'accélérations et de décélérations se révèlent négligeables en pratique et il est donc possible de simplifier le modèle en ne les prenant pas en compte. La durée de parcours d'un arc  $(i, j)$  par un véhicule  $v$  au temps  $t$  sera représenté par la formule suivante :

$$(I.1) \quad \text{duree}(i, j, v, t) = \frac{\text{distance}(i, j)}{\text{vitesse}(i, j, v, t)} + \text{attente}(i, j, t)$$

Pour les arcs  $(i, j)$  non *FIFO* du graphe, le temps d'attente  $\text{attente}(i, j, t)$  sera toujours nul.

### I.2.5.3 Problème global

Dans un tel graphe, le problème du plus court chemin en temps peut être résolu en temps polynomial (voir [Orda and Rom, 1990]). En revanche, le problème de plus court chemin en coût est NP-difficile (voir [Ahuja et al., 2003]).

Toutefois ce problème devant être résolu pour chaque véhicule du terminal, les résultats du routage d'un véhicule auront un impact sur les caractéristiques du graphe et donc sur le calcul du routage des autres véhicules. La résolution globale du problème de routage, c'est-à-dire pour tous les véhicules, repose sur l'ensemble des routages de chaque véhicule (routages locaux). Ainsi, lorsqu'un itinéraire est affecté à un chariot cavalier, il est nécessaire de vérifier si une légère modification d'une route déjà établie pour un autre véhicule n'améliorerait pas de façon significative la solution globale. Pour trouver la solution globale optimale, il faudrait alors calculer toutes les permutations possibles des solutions locales. Ce problème global est donc NP-complet.

### I.2.6 Ordonnancement et affectation des missions

L'ordonnancement des missions consiste à déterminer une date d'exécution à chaque mission ainsi qu'à leur affecter une ressource, c'est-à-dire un chariot cavalier.

La date d'exécution de la mission dépendra de deux facteurs. D'une part la fenêtre de temps associé à la phase de collecte du conteneur, et d'autre part la position du chariot cavalier affecté à la mission. En effet, la durée du déplacement du véhicule de sa position initiale à l'emplacement de collecte du conteneur sera prise en compte afin de déterminer la date d'exécution de la mission.

D'autre part, le choix du véhicule affecté à la mission dépendra de 2 facteurs. Tout d'abord, la compatibilité du véhicule avec la mission. En effet, certains chariots cavaliers ne sont pas capable d'adapter la taille de leur *spreader* sans devoir être modifié au dépôt par un mécanicien. C'est pourquoi, en fonction du type de conteneur à déplacer il sera impossible pour certains véhicules de remplir la mission, ou la date d'exécution de la mission devra prendre en compte la durée de déplacement du chariot vers le dépôt ainsi que la durée de la modification par le mécanicien. Ensuite, le deuxième facteur de choix du véhicule pour une mission particulière correspond à son activité, c'est-à-dire à son plan de charge au moment de l'ordonnancement. C'est en effet son activité qui déterminera sa position au moment de l'exécution ainsi que sa future destination après avoir accompli la mission planifiée. L'objectif est de choisir le véhicule dont l'insertion de la mission dans le plan de charge fera le moins augmenter la distance totale parcourue par l'ensemble de la flotte de véhicules, tout en respectant les contraintes temporelles.

Il est clair que le calcul de la date d'exécution de la mission dépend du chariot qui lui a été affecté et que le choix du chariot est fonction de la date d'exécution de la mission. Le calcul des dates d'exécution et l'affectation des missions aux véhicules doivent donc faire partie du même processus de calcul.

Ce problème fait l'objet de cette thèse. Nous détaillerons ainsi dans le chapitre suivant, les différentes modélisations du problème ainsi que la méthode de résolution que nous avons élaboré permettant de proposer des solutions de façon dynamique.

## Conclusions

Un terminal multimodal à conteneurs est une plate-forme logistique d'échange de marchandises ouverte à la fois sur la terre et sur l'eau. Leur développement important montre l'importance croissante de ces terminaux dans le secteur logistique mondial.

tance des enjeux économiques de ces structures qui doivent être de plus en plus performantes en terme de rapidité de service et de coûts d'exploitation. L'optimisation du terminal est donc essentielle et concerne différents problèmes comme la définition de la structure du terminal, de l'allocation des berges, des portiques de quai, du positionnement des conteneurs, du routage des véhicules ainsi que de l'affectation des opérations de déplacement de conteneurs.

Ce chapitre a permis de définir les tenants et les aboutissants du contexte applicatif de cette thèse et à introduit le problème d'ordonnancement et d'affectation des missions aux chariots cavaliers qui sera étudié dans le chapitre suivant.

# **Chapitre II**

## **Contexte théorique**

### **Introduction**

Ce chapitre présente le contexte théorique de cette thèse. Il définit le cadre ainsi que les bases théoriques de l'étude et dresse un état de l'art des travaux en relation avec le problème d'affectation et de routage des chariots cavaliers sur un terminal à conteneurs.

Ce dernier est effectivement lié à plusieurs autres problèmes abondamment traités dans la littérature et qui seront présentés dans les quatre sections de ce chapitre. L'optimisation dynamique sous incertitude fait l'objet de la première section. La deuxième section sera consacrée au problème du voyageur de commerce et à sa généralisation à  $m$  voyageurs. La troisième section sera dédiée aux problèmes de tournées de véhicules. Enfin, la dernière section établira un tour d'horizon des problèmes d'ateliers et en particulier du problème de *Job-Shop*.

### **II.1 Dynamique et incertitude en optimisation**

#### **II.1.1 Définition**

L'optimisation peut être définie comme le processus visant à améliorer une situation vis-à-vis de son niveau de performance actuel. Mathématiquement, l'optimisation revient à minimiser ou maximiser une fonction objectif en agissant sur des paramètres tout en respectant un ensemble de contraintes. Par exemple, un commerçant cherche à définir le prix de vente de chacun de ses produits afin de maximiser son bénéfice global. Le but ici est de chercher le prix permettant de générer le plus de ventes. Un prix trop élevé repoussera les clients alors qu'un prix trop faible ne permettra pas de dégager suffisamment de marge. Stratégiquement, il est possible de définir un prix très faible pour un ensemble de produits afin d'attirer les clients dans la boutique et

ainsi de pouvoir fixer un prix plus élevé sur les autres produits. Mathématiquement cet exemple consiste à maximiser la fonction  $\sum_{i=1}^n ((P_i - p_i) \cdot q_i)$ , où  $P_i$  est le prix d'achat de l'article  $i$ ,  $p_i$  son prix de vente et  $q_i$  la quantité vendue.

On distingue deux types d'optimisation. Tout d'abord l'optimisation déterministe dans laquelle la réaction du système à la modification des paramètres est parfaitement calculable. D'autre part l'optimisation sous incertitude dans laquelle des composantes du système sont inconnues ou mal connues et peuvent rendre inadaptée voire inapplicable la solution calculée.

L'incertitude peut être décrite de façon stochastique ou bornée. Dans le premier cas, on cherche généralement à optimiser le système dans le cas moyen (minimiser l'espérance mathématique de la fonction d'évaluation). Dans la représentation par intervalle borné, c'est le pire des cas qui est la plupart du temps l'objet de l'optimisation. Ainsi, une plage de valeurs possibles est prise dans un intervalle donné, et le cas le plus défavorable est utilisé afin de garantir un niveau de performance peu importe la dégradation du système liée à l'incertitude. On parle ainsi d'optimisation robuste. Le critère du regret maximal est également très répandu en optimisation robuste. Il se définit comme l'écart minimum entre les valeurs obtenues sur différents scénarios pour chaque stratégie étudiée.

Parallèlement, on distingue l'optimisation statique de l'optimisation dynamique. Dans cette dernière, le système évolue dans le temps. Dans ce cas, il devient nécessaire d'adapter les paramètres au cours de cette évolution, on parle alors de contrôle optimal (ou commande optimale). Chaque action aura des conséquences sur l'état futur du système. Dans le cas d'un problème déterministe il sera possible de calculer l'évolution du système en fonction de chaque action (scénarios, trajectoires). En revanche, si de l'incertitude est introduite dans le système, il devient impossible de prévoir son comportement futur. Par exemple, un système de guidage missile qui tente de calculer la trajectoire permettant d'atteindre un objectif en dépensant le moins d'énergie possible, adaptera la trajectoire si l'objectif est mobile. Le déplacement de l'objectif peut-être totalement prévisible (train, autre missile...) ou incertain (véhicule piloté par un humain). Dans ce dernier cas, il faudra effectuer des corrections de trajectoire à chaque nouvel état du système.

### II.1.2 Optimisation et problème décisionnel

Tout problème d'optimisation peut être vu comme un problème décisionnel. En effet le problème peut être énoncé de la façon suivante : existe-t-il une valeur de  $V$  pour laquelle  $f(V) \leq k$ ? Avec  $V$  vecteur de paramètres et  $k$  une valeur donnée. Par exemple, le problème

bien connu du sac-à-dos (*Knapsack Problem*) consiste à choisir des objets ayant chacun un poids  $p_i$  et une valeur  $v_i$ , pour remplir un sac-à-dos en cherchant à maximiser la somme des valeurs sans dépasser le poids maximal autorisé  $p_{MAX}$ . Le problème peut être formulé de la façon suivante. Pour chaque  $x_i$ , variable binaire indiquant si l'objet  $i$  est mis dans le sac ( $x_i = 1$ ) ou non ( $x_i = 0$ ), existe-t-il une valeur pour laquelle  $\sum_{i=1}^n p_i x_i \leq p_{MAX}$  et  $\sum_{i=1}^n v_i x_i \geq v$  pour  $v$  donné ? Il suffit pour résoudre ce problème de procéder de façon itérative en diminuant la valeur de  $v$  et d'arrêter la recherche dès qu'aucune solution n'est trouvée pour un  $v$  donné.

Notons qu'un problème de décision peut-être NP-complet alors qu'un problème d'optimisation ne peut-être que NP-difficile. Ainsi, sous forme de problème d'optimisation, le problème du sac-à-dos est NP-difficile, alors que dans sa version décisionnelle, ce problème est NP-complet.

La plupart des problèmes d'optimisation proviennent de situations réelles dans lesquelles des décideurs doivent agir en prenant en compte leur environnement afin de prendre la décision qui s'adapte le mieux à la situation rencontrée. Mais dans certains cas, voire dans la plupart des cas réels, les décideurs n'ont qu'une connaissance partielle de leur environnement, parfois même une vision subjective, biaisée par leur vécu et leurs émotions. Comment est-il possible, dans ces conditions, de déterminer les actions à réaliser pour obtenir la meilleure réponse possible au problème d'optimisation ?

### II.1.3 Identifier l'incertitude

L'incertitude peut provenir de 3 sources différentes :

- le jeu de données
- le scénario
- l'interprétation

La première source concerne l'incertitude sur les données du problème. Dans un problème d'ordonnancement par exemple, il peut s'agir d'incertitude au niveau des durées d'exécution des tâches, ou de la date limite. Il peut également s'agir des coûts de traitement, des probabilités de panne du système, etc.

L'incertitude liée au scénario concerne le contexte de l'application de la solution calculée. Ainsi, le coût de la main d'œuvre, le prix du carburant ou de l'électricité pour les machines, par exemple, font partie du contexte de l'exécution.

Enfin, l'incertitude liée à la subjectivité du décideur influence également la qualité de la solution calculée. Les valeurs ainsi que la culture propre à chaque individu influencent son com-

portement et provoque des divergences dans ses réactions. Par exemple, une personne habituée à jouer au casino sera plus enclin à prendre des risques qu'un gestionnaire habitué à rationaliser chaque décision. Toute décision prise en fonction d'éléments subjectifs est une source d'incertitude.

Après avoir identifié les sources potentielles d'incertitude dans un problème, il est nécessaire de déterminer la façon dont elle va être gérée.

#### **II.1.4 Gérer l'incertitude**

Il existe des moyens de gérer l'incertitude. Les systèmes prenant en compte cette incertitude sont dits pro-actifs, ou anticipatifs. Ils ont pour objectif de calculer une solution robuste, c'est-à-dire valable peu importe les valeurs prises par les éléments sujets à l'incertitude.

Toutefois cette gestion est limitée aux cas où l'incertitude est quantifiable (obéit à une loi stochastique par exemple) ou bornée. Lorsque l'incertitude est trop importante, c'est-à-dire que les éléments sujets à l'incertitude sont trop nombreux ou que l'incertitude n'est pas quantifiable, il est impossible d'envisager une prise en compte efficace. Il est important dans ce cas d'être réactif face à la découverte de nouvelles informations concernant le problème. Les systèmes comportant cette spécificité sont appelés systèmes réactifs. Dans ces systèmes, il est possible de détecter une variation si l'application d'une solution calculée ne donne pas les mêmes résultats escomptés. Il est alors possible d'agir afin de prendre en compte cette variation dans le calcul d'une nouvelle solution.

Dans cette thèse nous ne voulons pas essayer de prévoir l'imprévisible, de faire des prédictions, mais plutôt, à contrario, de s'adapter à tout événement. C'est pourquoi la solution proposée dans la suite de ce manuscrit sera inscrite dans la classe des systèmes réactifs. De plus, nous nous baserons sur la robustesse non pas de la solution proposée, mais plutôt par la robustesse de l'algorithme permettant d'obtenir la solution.

#### **II.1.5 Gestion de la dynamique**

L'optimisation dynamique peut être définie comme le processus visant à améliorer une situation vis-à-vis de son niveau de performance actuel au fil du temps. La fonction mathématique à minimiser ou maximiser sera également dépendante du temps. Dans [Powell et al., 1995], Powell et al. ont défini :

- qu'un problème est dynamique si au moins l'un de ses paramètres est fonction du temps ;

- qu'un modèle est dynamique s'il incorpore de façon explicite les interactions entre les activités au fil du temps ;
- qu'une application est dynamique si le problème sous-jacent doit être résolu à chaque fois qu'une nouvelle information est reçue.

Du fait de l'évolution temporelle du système, une solution calculée au temps  $t$  peut ainsi devenir caduque au temps  $t + n$ . À l'inverse de l'optimisation statique, il n'existe aucune garantie sur la stabilité des composantes du système dans le temps. Par exemple, le commerçant qui cherche à définir ses prix de vente pour maximiser son bénéfice devra prendre en compte l'évolution des prix d'achat de ses marchandises ainsi que le pouvoir d'achat de sa clientèle. La solution calculée à l'ouverture de son commerce ne sera peut-être plus la solution optimale six mois plus tard. Dans cet exemple, le commerçant a besoin d'informations concernant ses ventes afin de pouvoir calculer ses prix (quantité vendue). Il est donc possible de s'appuyer sur les résultats précédents pour définir les prix futurs. En revanche, dans le cas où l'optimisation est basée sur la situation courante, à l'instant  $t$ , il est impossible d'utiliser la solution calculée précédemment. La dynamique joue alors un rôle hautement perturbateur dans le processus d'optimisation et le plus souvent rend impossible l'application de méthodes de résolution exacte qui demanderont un temps de calcul supérieur à la durée de validité de la solution fournie.

En optimisation, la dynamique peut être vue comme un vecteur d'incertitude. Les composantes du système apparaissent, disparaissent, et évoluent au fil du temps. La connaissance que l'on a du système, sa perception, est alors elle même sujet à la dynamique. La dynamique étant une source d'incertitude elle peut provenir du jeu de données (les valeurs des données changent au fil du temps), du scénario (des événements surviennent au cours de l'exécution), ou de l'interprétation (la perception du décideur peut évoluer).

Dans [Psaraftis, 1988, Psaraftis, 1995], Psaraftis a identifié les problèmes liés à la dynamique dans le cadre d'un problème de tournée de véhicules (II.3) :

- **évolution** : prise en compte du temps ;
- **qualité** : l'information est de meilleur qualité lorsqu'elle est connue en avance ;
- **incertitude** : l'état futur du système est incertain ;
- **importance des événements proches du terme** : plus un événement est proche de sa date de déclenchement, plus il est important pour le système ;

- **importance du mécanisme de mise à jour de l'information** : il faut être en mesure de communiquer efficacement les multiples variations d'information dans le système ;
- **réajustements** : les solutions calculées au préalables peuvent devenir sous-optimales avec le temps ;
- **réduction des temps de calcul** : besoin de pouvoir recalculer une solution entre deux événements ;
- **report infini** : possibilité de devoir retarder une tâche à l'infini si des nouvelles tâches plus urgentes arrivent au fil du temps ;
- **fonction objective adaptée à la dynamique** : les fonctions objectif classique peuvent ne plus s'appliquer au problème car bien souvent l'horizon de planification est inconnu ;
- **flexibilité des contraintes temporelles** : il est parfois plus avantageux, dans les problèmes dynamique, de dépasser une fenêtre de temps plutôt que de décaler une tâche dont la fenêtre de temps n'est pas atteinte ;
- **optimisation des ressources** : le fait de ne pas connaître l'évolution de la demande fait qu'il est difficile de réduire au minimum le nombre de ressources à utiliser (engorgement), d'autre part une partie des ressources risquent de ne pas être utilisées si la demande est insuffisante.

On parle de dynamicité d'un scénario lorsqu'on mesure la part de dynamique dans le problème. Dans [Larsen, 2000], Allan Larsen indique trois principales mesures du degré de dynamicité.

Tout d'abord, le degré de dynamicité (*Degree Of Dynamism* : **dod**) est défini par Lund et al. (voir [Lund et al., 1996]) comme le rapport entre le nombre de requêtes dynamiques et le nombre total de requêtes (voir eq. II.1). Une requête est dite statique si elle est connue avant le début de la planification. Une requête est dite dynamique (on trouve également le terme “immédiate” dans la littérature) si elle n'est connue qu'une fois que l'exécution a commencé.

$$(II.1) \quad dod = \frac{\eta_d}{\eta_s + \eta_d}$$

Si cette mesure permet de quantifier la part de dynamique dans un scénario, elle ne permet pas d'évaluer son influence sur le système. En effet, cette mesure ne prend pas en compte la date d'arrivée des requêtes. Si les requêtes dynamiques entrent dans le système en début de scénario, le système est considéré comme aussi dynamique que si elles entraient en toute fin.

Pourtant, plus ces requêtes sont connues tard et plus le système doit être réactif afin de calculer une solution dans les temps. Ces retards ont donc un impact négatif sur les performances et il est important de les prendre en compte dans la mesure de dynamicité. Pour cette raison, Larsen a défini le degré effectif de dynamicité (*Effective Degree Of Dynamism* : **edod**) selon l'équation II.2. Ici,  $\eta_s$  et  $\eta_d$  sont respectivement le nombre de requêtes statiques et dynamiques.  $t_i$  est la date d'arrivée d'une requête  $i$  (avec  $0 < t_i < T$ ) et  $T$  correspond à la date de fin de scénario. Cette mesure prend en compte la moyenne des dates d'arrivées des requêtes dans le système. Plus les requêtes dynamiques arrivent tard, et plus *edod* sera important. Si *edod* = 0, alors le système est totalement statique. Si *edod* = 1, le système est purement dynamique.

$$(II.2) \quad edod = \frac{\sum_{i=1}^{\eta_d} \left( \frac{t_i}{T} \right)}{\eta_s + \eta_d}$$

Un certain nombre de problèmes d'optimisation utilisent des fenêtres de temps comme pour le problème de voyageur de commerce (voir II.2), le problème de tournées de véhicules (voir II.3) ou encore les problèmes d'ateliers (voir II.4). Larsen a adapté *edod* afin de prendre en compte l'écart temporel entre la date d'arrivée d'une requête dans le système et sa date d'exécution au plus tard. En effet, si une requête est connue à la fin de sa fenêtre de temps, le respect de celle-ci est problématique. En revanche, une requête arrivant bien avant sa date limite pourra être traitée dans les temps. Le degré effectif de dynamicité avec fenêtre de temps (*Effective Degree Of Dynamism with Time Window* : **edod-tw**) se définit selon l'équation II.3 où  $r_i$  est le temps de réaction (*reaction time*) de la requête  $i$ , ce qui correspond à l'écart entre la date d'exécution au plus tard de la requête  $i$  et sa date d'arrivée dans le système ( $t_i$ ).

$$(II.3) \quad edod - tw = \frac{1}{\eta_s + \eta_d} \cdot \sum_{i=1}^{\eta_s + \eta_d} \left( 1 - \frac{r_i}{T} \right)$$

Ces mesures permettent de quantifier de façon efficace la difficulté introduite dans le système par la dynamique.

### II.1.6 Méthodes de résolutions adaptées aux environnements dynamiques

Les modifications provoquées par la dynamique imposent une grande réactivité de la méthode de résolution du problème. Pour pouvoir prendre en compte les solutions calculées pour un ins-

tant  $t$ , dans le calcul de solutions valides à  $t+n$ , il faut que la méthode d'optimisation utilise une forme de mémoire. C'est pour cette raison que les méthodes de recherche tabou, les réseaux de neurones artificiels, les algorithmes génétiques ainsi que les algorithmes fournis semblent adaptés et seront détaillés dans les paragraphes suivants.

### II.1.6.1 Recherche tabou

La recherche tabou (Tabu Search), a été proposée pour la première fois par Glover en 1986 (voir [Glover and McMillan, 1986][Glover, 1989][Glover, 1990]). Elle consiste à voyager dans l'espace des solutions à partir du voisinage de la solution courante en ne conservant que la position minimisant la fonction objectif. Afin d'empêcher l'algorithme de visiter une nouvelle fois des parties de l'espace de recherche déjà étudiées, une sorte de mémoire est ajoutée sous forme de liste tabou dans laquelle les solutions déjà épuisées sont insérées. Appliquée au problème du voyageur de commerce, cette méthode consiste à construire un tour grâce à une heuristique quelconque (plus proches voisins par exemple) puis d'étudier les solutions voisines en inversant deux à deux les positions des arêtes à l'intérieur du tour. La meilleure solution ainsi obtenue est alors étudiée à son tour et la solution de départ est ajoutée à la liste tabou. L'algorithme se poursuit tant que le nombre maximal d'itérations n'est pas atteint ou que toutes les solutions voisines sont déjà dans la liste tabou.

### II.1.6.2 Réseaux de neurones artificiels

Les réseaux de neurones artificiels (Artificial Neural Networks) ont été introduits par McCulloch et Pitts en 1943 ([McCulloch and Pitts, 1943]). Ils se sont basés sur les observations de W. James, père du concept de la mémoire associative qui a défini un principe de fonctionnement qui deviendra plus tard la loi de Hebb. Ainsi, en 1949, Hebb a expliqué le conditionnement chez l'animal (expérience du chien de Pavlov par exemple), par les propriétés des neurones. Selon lui, il est donc possible de modifier les réactions d'un animal vis-à-vis d'un stimulus grâce à un processus d'apprentissage. En 1948, Von Neumann définit l'automate cellulaire (voir [Neumann, 1966]). Cet automate est capable d'accomplir une tâche élémentaire et de se multiplier pour y parvenir. Les automates cellulaires sont à la base notamment du jeu de la vie de Conway (voir [Gardner, 1970]), où l'état des cellules est déterminé en fonction de règles de voisinage.

Un neurone artificiel est un automate pourvu d'une fonction de transfert (non linéaire

bornée). Cette fonction a pour entrée un vecteur  $x$  associé à un vecteur  $w$  de pondération (coefficients synaptiques) ainsi qu'à un éventuel biais ( $b$ ) (voir fig. II.1). La fonction d'activation peut-être continue ou à seuil (voir fig. II.2), et s'active ou non en fonction du signal reçu en entrée.

$$y = f\left(\sum_{i=1}^N w_i \cdot x_i + b\right)$$

**Figure II.1:** Potentiel d'activation d'un neurone

$$f(x) = \begin{cases} 1 & \text{si } x > 0 \\ -1 & \text{sinon} \end{cases}$$

**Figure II.2:** Exemple de fonction d'activation à seuil

Dans un réseau de neurones, les neurones sont reliés les uns aux autres par des relations de voisinage formant ainsi un graphe orienté. L'activation ou la non-activation des neurones prédecesseurs d'un neurone déterminera la sortie de ce neurone.

Les réseaux de neurones peuvent être bouclés ou non. Dans un réseau non bouclé, le graphe représentant le réseau est acyclique. Dans les réseaux bouclés (ou récurrents), le graphe contient au moins un cycle.

La pondération des neurones peut être déterminée par un processus d'apprentissage qui peut être supervisé ou non. L'apprentissage supervisé consiste à fournir des paramètres d'entrée associés à des valeurs de sorties désirées. Les poids sont donc modifiés afin d'assurer l'obtention de la sortie fournie. Dans le cas de l'apprentissage non supervisé, aucune sortie désirée n'est fournie et il s'agit alors de déterminer les poids permettant de grouper les composantes de  $X$  selon une fonction  $F_w$  permettant une bonne généralisation (voir [Kohonen, 1982]).

Concernant le domaine de l'optimisation, le premier champ d'application fut la reconnaissance de formes. Ainsi, en 1957, Rosenblatt a développé le modèle du Perceptron (voir [Rosenblatt, 1958]). En 1960, Widrow a introduit le modèle Adaline (*Adaptive Linear Element*, voir [Widrow, 1960]) basé sur le perceptron de Rosenblatt mais utilisant un processus

d'apprentissage différent, qui sera plus tard à l'origine de l'algorithme de rétropropagation de gradient utilisé couramment avec les Perceptrons multicouches.

En 1969, Minsky et Papert ont montré dans [Minsky and Papert, 1969] les limites du modèle du Perceptron, ce qui a eu pour effet de décourager la recherche sur les réseaux de neurones artificiels.

Suite à l'article de Hopfield (voir [Hopfield, 1982]), en 1982, utilisant une méthode d'apprentissage basée sur un résultat escompté, les recherches sur les réseaux de neurones artificiels reprennent. Ainsi, en 1983, la Machine de Boltzmann permet de lever les limitations du Perceptron décrites par Minsky. En 1985, apparaît l'algorithme d'apprentissage par rétropropagation de gradient, utilisé dans les réseaux multicouches.

Plus récemment, des réseaux de neurones appelés cartes auto-organisatrices (ou auto-adaptatives) ont été développé par Kohonen (voir [Kohonen, 1982]). Elles peuvent être utilisées notamment pour la compression de données, d'images. Le principe est d'entraîner un réseau de neurones afin de reproduire une image d'entrée sans recopier chacun de ses pixels. Dans le cas d'une image en couleurs, chaque pixel peut être codé grâce à 3 entiers  $r$ ,  $v$  et  $b$  tels que  $0 \leq r, v, b < 2^8$ , décrivant une couleur par ses composantes : rouge ( $r$ ), vert ( $v$ ) et bleu ( $b$ ). Un pixel peut donc être stocké sur 24 bits (3 octets). Une image de résolution 1980x1080 comporte 2073600 pixels, et représente donc 49766400 bits (environ 5.93Mo). Le but de la compression est de diminuer le nombre de pixels à stocker sans pour autant dégrader l'image de façon trop importante. Pour cela chaque neurone sera chargé de décrire un groupe de pixels de l'image. Les neurones sont initialisés de façon aléatoire, puis un processus d'apprentissage non-supervisé permet de déterminer quels neurones décrivent le mieux chaque groupe de pixels. Le résultat de la compression regroupe la description de chaque neurone ainsi que les couples {groupe de pixels, neurone descripteur}.

#### II.1.6.3 Algorithmes génétiques

Les algorithmes génétiques (Genetic Algorithms) sont une métaheuristique reposant sur la théorie de l'évolution. Cette méthode générale a été introduite en 1954 par Barricelli (voir [Barricelli, 1957]) puis développée par Fraser à partir de 1957 (voir [Fraser, 1957]). Mais c'est à partir des travaux de Holland dans les années 1970 (voir [Holland, 1975]) que les algorithmes génétiques se sont démocratisés. Le principe est de représenter une solution (ici appelée individu) sous forme de chromosome (suite de gènes). Un nombre important d'individus (population, ou génome) est généré. Puis des opérateurs génétiques sont appliqués sur les gènes des chromo-

somes :

- **La sélection** : cet opérateur consiste à déterminer quels individus seront conservés dans la génération suivante
- **Le croisement** : cet opérateur consiste à simuler la reproduction de deux individus et ainsi à créer un nouvel individu conservant une partie des gènes de chacun de ses parents (héritage).
- **La mutation** : cet opérateur vise à altérer une partie des gènes d'un individu. Le but est de favoriser l'exploration de l'espace de recherche afin d'empêcher de piéger l'algorithme sur des optima locaux.

L'algorithme se déroule de la façon suivante. Tout d'abord une population initiale est générée (la plupart du temps de façon aléatoire). Les individus sont évalués. L'opérateur de sélection est ensuite appliqué et permet de sélectionner des couples d'individus. La génération suivante ne comportant pas les individus non sélectionnés, la population est complétée par les résultats de l'opérateur de croisement sur les individus sélectionnés. Enfin, l'opérateur de mutation est appliqué sur une très petite partie des individus (choisis de manière aléatoire) et consiste généralement à ne modifier qu'une infime portion de leur génome afin de ne pas annihiler les effets de la sélection et du croisement sur la convergence de l'algorithme. Le processus est répété un nombre fixé d'itération ou jusqu'à ce que la meilleure solution trouvée atteigne un score objectif préalablement fixé.

La plus grande difficulté dans l'implémentation de cet algorithme réside dans le codage des chromosomes. En effet, il faut que les chromosomes puissent être facilement construits tout en permettant d'appliquer les opérateurs sans provoquer d'inconsistance. Ainsi, dans certains problèmes il est impossible de croiser deux individus sans garantir la validité de l'individu ainsi créé. Or, c'est bien cet opérateur de croisement qui permet d'améliorer la convergence de l'algorithme et lui permet d'être plus performant qu'une recherche basée sur un recuit simulé ou une recherche tabou. D'autre part, la fonction d'évaluation d'un individu doit être rapide à calculer car c'est d'elle (ainsi que de la taille du problème) que dépendra le temps d'exécution de l'algorithme. Il existe bien-sûr différentes méthodes de sélection des individus. Les plus courantes sont :

- la sélection par rang : les n meilleurs individus sont choisis
- la sélection par tournoi : deux individus aléatoirement choisis sont mis en compétition, celui ayant le meilleur score est sélectionné
- la sélection uniforme : les individus ont tous la même probabilité d'être choisis

- la sélection par probabilité : les individus ont une probabilité  $p_i = \frac{score_i}{\sum_n^N score_n}$  d'être choisis.

#### II.1.6.4 Algorithmes fourmis

L'optimisation par colonie de fourmis (Ant Colony Optimization) est également une métaheuristique basée sur une observation du vivant.

Cette technique est issue de l'observation de Deneubourg en 1983 (voir [Deneubourg et al., 1983]) et 1989 (voir [Goss et al., 1989]) qui a montré que les fourmis étaient capable collectivement de trouver le plus court chemin entre leur nid et une source de nourriture. En effet, Deneubourg a observé que les fourmis commençaient à coloniser les chemins de façon aléatoire puis au fil du temps commençaient à suivre le même chemin. Il s'avère que ce chemin est le plus court. Ce phénomène est dû à la méthode de communication des fourmis. Elles utilisent un marqueur chimique : la phéromone afin d'indiquer, dans le cas d'une recherche de nourriture, l'attractivité d'un chemin. Ainsi, un chemin permettant de trouver de la nourriture contiendra de la phéromone et sera donc suivi par les autres fourmis de la colonie.

Le point fort de cette méthode de communication indirecte, appelée stigmergie (voir [Grassé, 1959]), est de permettre de trouver un chemin optimal sans pour autant qu'il n'ai été parcouru en totalité par une fourmi. En effet, la solution optimale est construite en cherchant le chemin comportant le plus de phéromone entre le nid et la source de nourriture. Les fourmis construisant leurs chemins en déposant de la phéromone, c'est bien la somme des dépôts de toutes les fourmis qui permettra de découvrir le chemin globalement optimal. Chaque individu agit positivement sur le choix des composantes du chemin (processus auto-catalytique).

D'autre part, la phéromone étant volatile, le chemin comportant le plus de phéromone sera celui sur lequel elle se sera le moins évaporée. Ce sera donc le chemin le plus court qui contiendra le plus de phéromone. Ce phénomène d'évaporation (rétroaction négative) permet également aux fourmis de ne pas réutiliser des chemins qui étaient les plus attractifs (les plus court) au moment de leur découverte.

L'algorithme de colonie de fourmis fut inventé par Dorigo lors de sa thèse en 1991 (voir [Dorigo, 1992] et [Dorigo et al., 1991]). Il a indiqué que la méta-heuristique pouvait s'appliquer à tout problème d'optimisation : recherche de plus court chemin, problèmes d'ordonnancement de tâches, problème de tournées de véhicules, problème d'affectations quadratique, etc. Les fourmis artificielles de Dorigo et al. sont légèrement différentes des fourmis réelles. Elles ont une mémoire leur permettant d'éviter de coloniser des morceaux de chemin qu'elles ont déjà

parcouru. D'autre part, leur vision leur permet de choisir une destination en fonction de la distance vis à vis de leur position courante. Ce sont ces légères différences qui vont permettre aux algorithmes fourmis de converger plus rapidement vers la solution optimale.

Dans [Dorigo, 1992], Dorigo et al. présentent l'algorithme *Ant-System* avec 3 variantes concernant le marquage des chemins : *Ant-density*, *Ant-quantity* et *Ant-cycle*.

*Ant-System* est défini comme suit :

- Soit  $\tau_{ij}(t)$  l'intensité de la trace de phéromone sur l'arc  $(i, j)$  au temps  $t$  ;
- Soit  $\Delta_{ij}(t, t + 1)$  la quantité de phéromone déposée sur l'arc  $(i, j)$  par les fourmis entre le temps  $t$  et  $t + 1$  ;
- Soit  $\rho$  un coefficient de persistance de la trace de phéromone tel que  $0 \leq \rho \leq 1$  ;
- À chaque fin de parcours les traces de phéromones sont ainsi mises à jour (eq. II.4) :

$$(II.4) \quad \tau_{ij}(t + 1) = \rho \cdot \tau_{ij}(t) + \Delta_{ij}(t, t + 1)$$

- Soit  $\eta_{ij}$  l'attractivité de l'arc  $(i, j)$  (représentant l'inverse de sa longueur ( $1/d_{ij}$  dans le cas d'une recherche de plus court chemin par exemple)).
- Une fourmi  $k$  choisit sa destination en fonction de la loi de transition proportionnelle suivante (eq. II.5) :

$$(II.5) \quad p_{ij}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in \text{visibles}} [\tau_{il}(t)]^\alpha \cdot [\eta_{il}]^\beta} & \text{si } j \text{ n'a pas déjà été visité} \\ 0 & \text{sinon} \end{cases}$$

où  $\alpha$  et  $\beta$  sont des paramètres permettant de contrôler l'importance de l'heuristique d'attractivité ( $\eta_{ij}$ ) par rapport à l'intensité de la trace de phéromone ( $\tau_{ij}(t)$ ).

Les variantes *Ant-density*, *Ant-quantity* et *Ant-cycle*, se définissent par rapport à la politique de dépôt de phéromone. Dans *Ant-density*, la fourmi dépose une quantité fixe de phéromone sur chaque arc  $(i, j)$  qu'elle emprunte (eq. II.6) :

$$(II.6) \quad \Delta_{i,j}^k(t, t + 1) = \begin{cases} Q & \text{si la fourmi } k \text{ va de } i \text{ à } j \text{ entre le temps } t \text{ et } t + 1 \\ 0 & \text{sinon} \end{cases}$$

Dans *Ant-quantity*, la fourmi dépose une quantité de phéromone fonction de la pondération  $w_{ij}$  de l'arc emprunté (eq. II.7) :

$$(II.7) \quad \Delta_{i,j}^k(t, t + 1) = \begin{cases} \frac{Q}{w_{ij}} & \text{si la fourmi } k \text{ va de } i \text{ à } j \text{ entre le temps } t \text{ et } t + 1 \\ 0 & \text{sinon} \end{cases}$$

Dans *Ant-cycle*, la fourmi dépose une quantité de phéromone fonction de la somme des pondérations du chemin suivi :  $L_k = \sum_{a_{ij}}^{\text{chemin}} w_{ij}$  (eq. II.8) :

$$(II.8) \quad \Delta_{i,j}^k(t, t+n) = \begin{cases} \frac{Q}{L_k} & \text{si la fourmi } k \text{ emprunte l'arc } (i, j) \text{ dans son chemin} \\ 0 & \text{sinon} \end{cases}$$

Une stratégie élitiste est également définie. Elle consiste à renforcer la trace de phéromone sur le meilleur chemin trouvé jusqu'à présent par une quantité de phéromone égale au nombre de fourmis élitistes multiplié par  $Q^*$  sur la somme des pondérations des arcs du meilleur chemin (eq. II.9) :

$$(II.9) \quad e \cdot Q^*/L^*$$

Dorigo et al. ont montré que l'utilisation de cette stratégie permettait de converger plus rapidement vers l'optimum global à partir d'une certaine valeur de  $e$ , mais qu'au delà de cette valeur l'algorithme peut être piégé dans des optimum locaux.

Le principal inconvénient de cette métaheuristique réside dans le nombre de ses paramètres. Il faut en effet définir une valeur pour  $\alpha$  et  $\beta$ , pour  $\rho$ , et dans le cas où la stratégie élitiste est utilisée il faut également déterminer les valeurs de  $Q^*$  et de  $e$ .

Suite au succès d'*Ant-System*, des extensions ont été développées et sont regroupées sous le terme d'ACO (Ant Colony Optimization, ie. Optimisation par Colonie de Fourmis) (voir [Geurts, 1998]). Les plus connues sont :

- **Elitist Ant System (AS<sub>elite</sub>)** : de la phéromone est déposée à chaque itération sur les arcs du meilleur chemin découvert ;
- **Min-Max Ant System (MMAS)** : développé en 2000 par Stützle et Hoos (voir [Stützle and Hoos, 2000]). Il repose sur l'ajout d'une borne inférieure et supérieure d'intensité de phéromone. Ainsi, à l'initialisation, une quantité  $q = \tau_{\max}$  est déposée sur tous les arcs, puis la version élitiste d'*Ant-System* se déroule normalement et lors du processus d'évaporation, les arcs contenant une quantité de phéromone inférieure à  $\tau_{\min}$  sont réinitialisés à la valeur  $\tau_{\min}$  ;
- **Ant Colony System (ACS)** : dans cette version d'*Ant-System* appliquée au problème du voyageur de commerce (voir [Dorigo and Gambardella, 1997]) :
  - le paramètre  $\alpha$  d'*Ant-System* a été rendu constant et vaut 1 ;

- $\alpha$  devient le paramètre d'évaporation globale modifiant ainsi la formule d'évolution de la phéromone d'*Ant-System* (eq. II.4 devient eq. II.10) :

$$(II.10) \quad \tau_{ij} = (1 - \alpha) \cdot \tau_{ij} + \alpha \cdot \Delta\tau_{ij}$$

$$\text{où } \Delta\tau_{ij} = \begin{cases} 1/L^* & \text{si } (i, j) \in \text{meilleur chemin global} \\ 0 & \text{sinon} \end{cases}$$

- lors de la construction de leur chemin, les fourmis déposent de la phéromone localement selon la règle suivante (eq. II.11) :

$$(II.11) \quad \tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \rho \cdot \Delta\tau_{ij}$$

où  $0 < \rho < 1$  est le paramètre d'évaporation locale.

Dorigo et al. ont défini plusieurs politiques de dépôt local de phéromone. On trouve ainsi les politiques :

- d'absence de dépôt local de phéromone où  $\Delta\tau_{ij} = 0$
- de marquage constant où  $\Delta\tau_{ij} = \tau_0$  ( $\tau_0$  est la valeur d'initialisation de la marque de phéromone)
- *Ant - Q* (voir [Gambardella et al., 1995]), basée sur l'algorithme *Q-learning*, où  $\Delta\tau_{ij} = \gamma \cdot \max_{l \in J_k(j)} \tau_{jl}$  ( $0 \leq \gamma \leq 1$  est un paramètre)
- une fois que toutes les fourmis ont terminé leur chemin, la règle globale de dépôt de phéromone est appliquée (eq. II.10)
- la règle de transition utilisée est appelée règle de transition pseudo aléatoire et suit la formule suivante (eq. II.12) :

$$(II.12) \quad v = \begin{cases} \arg \max_{j \in J_k(i)} \{[\tau_{i,j}] \cdot [\eta_{i,j}]^\beta\} & \text{si } q \leq q_0 \\ S & \text{sinon} \end{cases}$$

Ainsi, un nombre  $0 < q < 1$  est tiré au sort. S'il est inférieur à  $q_0$ , alors la fourmi choisit la ville ayant le meilleur score. Sinon, la fourmi choisit une ville selon la formule classique d'*Ant-System* (voir eq. II.5).

- **Rank-based Ant System (AS<sub>rank</sub>)** : toujours appliquée au problème de voyageur de commerce, cette variante d'*Ant-System* est basée sur le renforcement des pistes de phéromones des  $\sigma$  meilleures fourmis de la colonie [Bullnheimer et al., 1997c]. Les fourmis sont classées selon la longueur de leur solution et la formule de dépôt de phéromone

devient :

$$(II.13) \quad \tau_{ij}(t+1) = \rho \cdot \tau_{ij} + \Delta\tau_{ij} + \Delta\tau_{ij}^*$$

Avec  $\Delta\tau_{ij} = \sum_{\mu=1}^{\sigma-1} \Delta\tau_{ij}^\mu$

et  $\Delta\tau_{ij}^\mu = \begin{cases} (\sigma - \mu) \cdot \frac{Q}{L_\mu} & \text{si la } \mu^{\text{eme}} \text{ meilleure fourmi emprunte l'arc } (i, j) \\ 0 & \text{sinon} \end{cases}$

et  $\Delta\tau_{ij}^* = \begin{cases} \sigma \cdot \frac{Q}{L_*} & \text{si l'arc } (i, j) \text{ fait partie du meilleur chemin} \\ 0 & \text{sinon} \end{cases}$

Les méthodes de résolution de problèmes dynamique d'optimisation doivent donc, dans l'idéal utiliser les solutions calculées précédemment afin d'éviter de repartir de zéro à chaque nouvel événement. Les métaheuristiques, de par leur vitesse de calcul ainsi que la qualité de leurs solutions peuvent donc être utilisées à la condition de comporter un mécanisme de mémoire. La recherche tabou est une métaheuristique à trajectoire pouvant être utilisée en environnement dynamique car elle consiste à effectuer une recherche locale grâce à une heuristique, rapide en terme de calcul, et à mémoriser les trajectoires étudiées afin de ne pas les évaluer de nouveau avant un certain temps. Les réseaux de neurones artificiels consistent à relier des neurones modélisés par une fonction mathématique simple et à déterminer par un mécanisme d'apprentissage des pondérations entre eux. Ce sont ces poids qui représentent la part de mémoire de l'algorithme ou plutôt une réponse déterministe à un stimulus précis. Lorsque l'environnement évolue, une adaptation des poids pourra être nécessaire. En 1994, Rayward-Smith a indiqué dans [Rayward-Smith, 1994] que les algorithmes génétiques étaient robustes. Cette remarque est en effet liée à la possibilité intrinsèque de l'algorithme de s'adapter aux variations de l'environnement grâce notamment à la taille du génome ainsi qu'aux opérateurs permettant un brassage effectif de la population de gènes. Quant aux algorithmes fourmis, ils font partie de la famille des métaheuristiques à apprentissage et population. Ici, la mémoire est modélisée par le système de marquage des fourmis par dépôt de phéromones. Les traces de phéromones s'évaporent avec le temps ce qui permet en quelque sorte "d'oublier" certaines solutions passées.

## Conclusion

Dans cette partie nous avons introduit la notion d'optimisation dynamique sous incertitude. Nous nous placerons volontairement dans cette thèse dans l'idée de ne pas chercher à évaluer

les valeurs futures des données incertaines, ou à prévoir l'évolution du système. Nous proposeront une solution au problème d'optimisation étudié appartenant à la classe des systèmes réactifs. D'autre part, dans les problèmes classiques d'optimisation dynamique rencontrés dans la littérature, c'est la fonction objectif qui évolue au fil du temps. Dans le problème étudié dans cette thèse c'est l'environnement qui est sujet à la dynamique et qui influence le système d'optimisation. Nous avons donc développé, non pas une fonction dynamique, mais un algorithme suffisamment flexible pour s'adapter à l'optimisation dynamique.

## II.2 Problèmes de voyageurs de commerce

### II.2.1 Définition

Le problème du voyageur de commerce (PVC), ou en anglais *Traveling Salesman Problem* (TSP), est un problème classique d'optimisation combinatoire de recherche de cycle hamiltonien de poids minimal dans un graphe complet. Dans ce problème un voyageur de commerce doit rendre visite à une liste de clients situés dans des villes. Il s'agit donc de calculer le plus court chemin depuis la ville de départ du voyageur de commerce (appelée dépôt), passant une fois par chaque ville à visiter, puis revenant au point de départ du voyageur.

Soit  $G = (V, E, w)$  le graphe complet composé des  $n \in V$  villes à visiter et d'une fonction  $w$  de pondération des arêtes  $e_{i,j} \in E$  ( $\forall i, j \in V, i \neq j$ ).

Le nombre de cycles possibles évolue de façon combinatoire en fonction du nombre de villes à visiter. Le graphe étant complet, il existe  $(|V| - 1)!$  chemins possibles car il commence et fini forcément par la ville du voyageur de commerce. La fonction de coût étant symétrique dans un graphe non orienté, il revient au même d'effectuer un chemin dans un sens que dans l'autre d'où la réduction du nombre de possibilités à  $\frac{(|V|-1)!}{2}$ .

Il existe plusieurs variantes de ce problème dans lesquelles :

- le graphe est asymétrique (Asymmetric TSP) :  $G = (V, A, w)$ . Le nombre de solutions est ici  $(|V| - 1)!$
- des contraintes supplémentaires sont ajoutées :
  - Fenêtres de temps (TSP-TW) : chaque ville doit être visitée dans une certaine période de temps. Dans ce cas, la fonction de pondération utilise le temps de parcours et non

- plus la distance entre deux villes. Il s'agit donc de trouver le cycle hamiltonien de poids minimal respectant les contraintes temporelles.
- Capacité : le voyageur doit repasser par son point de départ tous les  $n$  villes visitées.
  - Précédence : le voyageur doit visiter certaines villes avant de pouvoir en visiter d'autres.
  - la fonction objectif est multicritère : dans la version comportant des fenêtres de temps, le but du voyageur est de visiter tous ses clients en minimisant la distance totale parcourue et en minimisant le retard. Dans ce cas il existe 3 façons d'associer les différents critères :
    - Dominance Pareto
    - Priorisation (l'un puis l'autre)
    - Linéarisation (association pondérée des critères pour n'en former qu'un seul)

### II.2.2 Historique

Dans [Schrijver, 2005], Schrijver à dressé un historique du problème du voyageur de commerce.

Les premières traces de la problématique remontent à 1882 dans un manuel pour voyageur de commerce. La première formulation mathématique du problème a été donnée par Karl Menger en 1930. Il a également présenté l'algorithme de recherche exhaustive permettant de trouver la solution optimale en parcourant toutes les solutions et a observé la non-optimalité de l'heuristique du plus proche voisin. Entre 1931 et 1934, Hassler Whitney aurait introduit pour la première fois le nom de Traveling Salesman Problem.

La principale avancée dans la résolution exacte du problème est apparue en 1954. Dantzig, Fulkerson et Johnson présentèrent plusieurs nouvelles méthodes de résolution du problème mettant en avant l'utilisation de la méthode des "Cutting Planes" (plans de coupes) leur permettant de résoudre une instance du problème comportant 49 villes (voir [Dantzig et al., 1954]).

C'est en 1956 que Flood fit le rapprochement entre le problème du voyageur de commerce et les jeux de Hamilton consistant à rechercher des chemins hamiltoniens dans un graphe. En 1972, Karp a caractérisé le problème de recherche de cycle hamiltonien de poids minimal dans un graphe comme appartenant à la classe des problèmes NP-Complet, ce qui par conséquent classe le problème du voyageur de commerce comme NP-difficile.

Dans les années 1980 des instances comportant jusqu'à 2392 villes ont été résolues par des méthodes de Branch-and-Bound et de Cutting Planes.

En 2005, Cook a trouvé la solution optimale à une instance du problème comportant 33810

villes. Un an plus tard, en 2006, Applegate a résolu une instance comportant 85900 villes.

### II.2.3 Méthodes de résolution

Le problème du voyageur de commerce étant NP-difficile il n'existe pas d'algorithme permettant de le résoudre en temps polynomial. Toutefois des méthodes de résolution exacte ont été développées au cours des 80 dernières années et ont permis, conjointement au progrès technologique (puissance brute des micro processeurs, grilles de calculs, etc.), de résoudre des instances de taille intéressante (jusqu'à 85900 villes). D'autres chercheurs se sont intéressés à la résolution approchée du problème afin de fournir des solutions proches de la solution optimale et parfois même avec une garantie de performance.

Nous présenterons ici les méthodes de résolution exacte, puis approchée, en détaillant dans ce cas les approches heuristiques et les résolutions utilisant des méta-heuristiques.

#### II.2.3.1 Méthodes de résolution exacte

Il existe différentes approches de résolution exacte pour le problème du voyageur de commerce qui ont permis de résoudre des instances de taille de plus en plus importante au fil du temps.

Tout d'abord, la plus naturelle est la recherche exhaustive (Brute Force Search). Elle a été décrite par Menger dans [Menger, 1928]. Cette méthode consiste à construire toutes les permutations de villes et mesure la combinaison de coût minimum ( $O(n!)$ ).

Au début des années 1960, Bellman décrit dans [Bellman, 1962] une méthode de résolution par programmation dynamique (Dynamic Programming) du problème de voyageur de commerce. L'approche est basée sur le fait que le coût de la visite d'une ville dépend uniquement du chemin précédemment parcouru. Mais c'est l'algorithme de Held-Karp [Held and Karp, 1961], présenté quelques mois avant celui de Bellman, qui s'avère le plus performant et résout le problème en  $O(n^2 2^n)$ .

Dès 1963, Little et al. proposent dans [Little et al., 1963] un algorithme de séparation et évaluation (Branch and Bound) pour résoudre le problème de voyageur de commerce. Cette méthode consiste à calculer une borne inférieure sur le cycle de poids minimal trouvé jusqu'à présent. Les tours sont construits grâce à la méthode de recherche exhaustive tant que la borne n'est pas atteinte. Lorsqu'un tour complet est construit une nouvelle borne est définie et permet de resserrer l'espace de recherche aux solutions inférieures ou égales à celle-ci (voir

[Balas and Toth, 1985]). L'algorithme de Little et al. permet de résoudre des instances du problème jusqu'à 40 villes.

Ce sont finalement les méthodes de résolution par programmation linéaire qui s'avèrent les plus performantes pour résoudre le problème du voyageur de commerce. Elles utilisent une définition du problème sous forme de minimisation d'une fonction linéaire sur un polyèdre convexe. L'algorithme du Simplexe [Dantzig, 1963] permet ainsi de résoudre des instances jusqu'à 200 villes.

Enfin, des méthodes basées sur des procédures de séparation et évaluation, et sur l'optimisation linéaire ont été adaptées à des instances particulières du problème. Ces adaptations "sur-mesures" permettent ainsi de résoudre des instances de très grande taille. En 2006, Applegate a résolu un problème comportant 85900 villes (voir [Applegate, 2006]).

### **II.2.3.2 Méthodes de résolution approchée**

Dans la littérature on trouve deux grandes classes de méthodes de résolution approchée du problème du voyageur de commerce. D'une part les méthodes de construction, et d'autre part les méthodes d'amélioration de solution.

#### **II.2.3.2.1 Heuristiques constructivistes :**

La première heuristique utilisée pour résoudre le problème du voyageur de commerce fut la méthode des plus proches voisins (Nearest Neighbour). Dès le début des années 1930 Menger a montré la non optimalité de cet algorithme glouton qui consiste à construire un chemin de façon itérative en sélectionnant à chaque itération la ville la plus proche. Cette méthode permet d'obtenir très rapidement des cycles assez court (environ 25% de l'optimal dans les instances comportant des villes réparties aléatoirement dans l'espace). Cependant l'heuristique des plus proches voisins peut conduire à de très mauvais résultats dans certains cas particuliers.

En 2007, Ray et al. ont proposé dans [Ray et al., 2007] l'heuristique des plus proches fragments (Nearest Fragment). Il s'agit d'une variante de la méthode des plus proches voisins qui permet d'obtenir de meilleurs résultats en sélectionnant à chaque itération le groupe de villes le plus proche. Le fait de regrouper les villes en clusters permet de répartir les effets de la recherche locale sur une plage plus large de l'espace de recherche. Il s'agit d'une heuristique moins locale que celle des plus proches voisins.

D'autres heuristiques permettent de réduire localement les cycles en calculant les arbres couvrants de poids minimal grâce aux algorithmes de Kruskal (voir [Kruskal, 1956]) ou de Prim

(voir [Prim, 1957]). Ainsi, à chaque ville visitée la ville voisine étant la racine du sous arbre de poids minimal est choisie et insérée dans le tour. Là aussi les performances ne sont pas proches de l'optimal.

En 1976, Christofides a présenté dans un rapport technique (voir [Christofides, 1976]) une nouvelle heuristique permettant de résoudre les instances du problème vérifiant l'inégalité triangulaire précisant que la distance entre les villes  $i$  et  $k$  est inférieure ou égale à la distance entre les villes  $i$  et  $j$  plus la distance entre  $j$  et  $k$  :

$$(II.14) \quad w_{i,k} \leq w_{i,j} + w_{j,k} \text{ avec } i \neq j \neq k \quad \forall i, j, k$$

Son algorithme permet d'obtenir une solution approchée garantie à 1.5 fois la solution optimale.

En 1990 Bentley à définit une heuristique de construction d'un cycle hamiltonien sous forme de tour bitonique (voir [Bentley, 1990]). Le principe est de relier les villes en formant un polygone monotone en cherchant le polygone ayant le périmètre le plus court. Pour maintenir la propriété de monotonie du polygone il peut être nécessaire d'inverser des arêtes.

En 2004, Kahng et Reda ont proposé une heuristique basée sur l'enchaînement de deux couplages sur le graphe (voir [Kahng and Reda, 2004]). Le second couplage est réalisé sur le graphe privé des arêtes présentent dans le premier couplage. Le résultat de ces deux couplages successifs est un ensemble de cycles. La seconde phase de l'algorithme de Kahng et Reda consiste à assembler ces cycles pour former un tour comportant tous les sommets du graphe. Leur heuristique donne de meilleurs résultats que les autres heuristiques de construction de tour évoquée précédemment.

### **II.2.3.2.2 Heuristiques d'amélioration :**

Concernant les méthodes d'amélioration les principales heuristiques sont basées sur la suppression de certaines arêtes dans la solution préalablement calculée puis à l'ajout d'arêtes permettant de reconnecter le chemin de en minimisant la distance totale. L'heuristique  $V\text{-}opt$  consiste à retirer un nombre variable d'arêtes à la solution calculée puis de rajouter autant de nouvelles arêtes afin de reconnecter le cycle. Ainsi, en 1973, Lin et Kernighan proposèrent l'heuristique  $2\text{-}opt$  (voir [Lin and Kernighan, 1973]) qui est une heuristique dérivée de  $V\text{-}opt$  où le nombre d'arête à supprimé est fixé à 2. Ainsi, à chaque itération, 2 arêtes sont échangées dans la solution. D'autres approches utilisent un nombre fixe  $k$  d'arêtes à chaque itération ( $k\text{-}opt$ ). La méthode la plus utilisée est  $3\text{-}opt$ . En 1990, Johnson a étendu l'algorithme de Lin-Kernighan en construisant une solution grâce à l'heuristique  $3\text{-}opt$  puis en déplaçant au moins 4 arêtes dans

la solution afin d'empêcher l'algorithme de rester bloqué sur une solution localement optimale (voir [Johnson, 1990]).

En 1991, Martin et al. ont introduit un algorithme basé sur la procédure de Markov chain Monte Carlo (MCMC) (voir [Martin et al., 1991]). Ils ont ainsi résolu de façon optimale des instances du problème jusqu'à 783 villes. Pour des instances plus grandes, leur procédure améliore le score de l'heuristique 3-opt de 1.6% et l'algorithme de Lin-Kernighan de 1.3%.

#### II.2.3.2.3 Métaheuristiques :

Des méthodes plus généralistes (méta-heuristiques) sont également largement répandues dans la littérature afin de résoudre de façon approchée le problème du voyageur de commerce.

**Le recuit simulé** (*Simulated Annealing*) est une métaheuristique pouvant être utilisée pour résoudre des instances du problème de voyageur de commerce. Cette méthode a été inventé en 1983 par Kirkpatrick, Gelatt et Vecchi (voir [Kirkpatrick et al., 1983]) et fonctionne selon une analogie avec des phénomènes physiques et quantiques. En effet, en métallurgie, les matériaux sont chauffés pour être façonnés, puis refroidis lentement. Cette baisse de température peut fragiliser le matériau si elle est trop rapide. Une opération de recuit (réchauffement) est alors utilisée pour permettre aux atomes de se placer dans la configuration la plus stable et ainsi redonner toute sa solidité au matériau.

Du point de vue de l'optimisation, il s'agit de parcourir un espace de recherche très vaste de façon locale en parcourant le voisinage d'une solution (refroidissement). Des solutions performantes diminuent l'énergie du système et des solutions moins performantes fragilisent le système (augmentent son énergie).

Une solution dégradant l'énergie du système est acceptée selon une probabilité  $e^{-\frac{\Delta_E}{T}}$  (règle de Metropolis [Metropolis et al., 1953]) où  $T$  est un paramètre virtuel de température et  $\Delta_E$  est l'écart d'énergie entre la meilleure solution trouvée et la solution courante. Ce processus permet d'empêcher l'algorithme de rester piégé sur une solution localement optimale.

Au fil de la recherche le seuil de performance (la température) est diminué afin de resserrer la recherche autour de la solution courante. L'algorithme s'arrête quand l'énergie de la solution trouvée est inférieure à un niveau d'énergie défini par avance (la qualité de la solution est satisfaisante), ou que le nombre maximal d'itérations a été atteint.

Dès 1983 Kirkpatrick et al. ont résolu grâce à leur algorithme une instance du voyageur de commerce contenant 6000 villes alors que la plus grande instance résolue à cette époque ne

contenait que 318 villes.

**La recherche tabou** (*Tabu Search*) a été appliquée pour la première fois au PVC par Glover dans [Glover and McMillan, 1986]. L'algorithme utilise l'heuristique  $2 - opt$  afin de parcourir l'espace des solutions associée à une mémoire afin de ne pas explorer le voisinage d'une solution déjà étudiée précédemment. Dans sa thèse de doctorat, Troyon décrit un algorithme tabou appliqué au problème du voyageur de commerce et qui utilise également l'heuristique  $2 - opt$  (voir [Troyon, 1988]). Dans [Malek et al., 1989], Malek et al. ont développé un algorithme tabou appliqué à des instances du PVC comportant de 25 à 100 villes capable d'être exécuté en environnement parallèle. Ils ont montré que la méthode de recherche tabou était plus performante que la méthode du recuit simulé en terme de temps de calculs tout en obtenant des performances similaires en terme de qualité de solution.

**Les réseaux de neurones artificiels** (*Artificial Neural Networks*) ont été utilisés dans trois principales méthodes pour résoudre le problème du voyageur de commerce.

Tout d'abord, en 1985, Hopfield résout le problème du voyageur de commerce de taille  $n$  en utilisant un réseau de  $n^2$  neurones (voir [Hopfield and Tank, 1985]). Il s'agit d'un réseau bouclé sans apprentissage où toute sortie d'un neurone est reliée à une entrée de tous les neurones (même à lui-même). Le principe de l'algorithme est de chercher, par la méthode d'apprentissage, la configuration des poids des neurones permettant de minimiser la distance parcourue par le voyageur de commerce.

D'autre part, en 1987, Durbin et al. ont proposé et étudié une approche par "filet élastique" (*elastic net*) (voir [Durbin and Willshaw, 1987] et [Durbin et al., 1989]) vis-à-vis de la résolution du problème de voyageur de commerce. Le principe est d'associer l'espace en deux dimensions où sont positionnées les  $n$  villes et une chaîne de  $n$  neurones artificiels. En 1992, Ritter et al. ont amélioré la méthode en utilisant un nombre de neurones supérieur au nombre de villes et en initialisant les poids sur un cercle (appelé  $N$ -gon) positionné sur le centre de gravité des villes (voir [Ritter et al., 1992]). Le cercle est alors graduellement étendu jusqu'à se trouver suffisamment proche de chaque ville pour former une tournée. L'algorithme cherche à minimiser à la fois la taille du cercle, et la distance entre le cercle et les villes. Ce modèle donne de meilleur résultats que celui de Hopfield. En effet, sur un problème de taille 30, la méthode de Hopfield obtient un tour de longueur 5.07 après plusieurs essais alors que la méthode de Durbin et al. donne une longueur de 4.26 dès le premier résultat.

Enfin, en 1988, Angéniol et al. ([Angéniol et al., 1988]) ont également résolu le problème de voyageur de commerce grâce à une carte auto-organisatrice (*Self Organized Map*) et ont permis de démocratiser cette méthode de résolution. Ainsi, Favata et Walker [Favata and Walker, 1991], Budinich et Rosario [Budinich and Rosario, 1996], Aras et al. [Aras et al., 1999], ou plus récemment Leung et al. [Leung et al., 2004], ont utilisé une carte auto-organisatrice pour résoudre des problèmes de voyageurs de commerce comportant jusqu'à 2400 villes et ont montré que cette méthode obtenait des résultats comparables à ceux obtenus par la méthode du filet élastique.

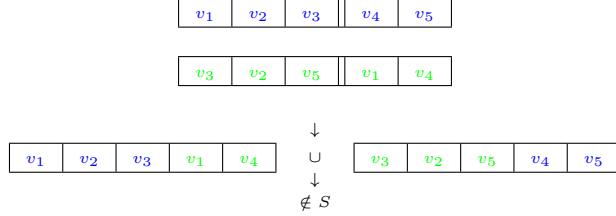
**Les algorithmes génétiques** (*Genetic Algorithms*) ont largement été utilisés pour résoudre le problème de voyageur de commerce. Un état de l'art des différents opérateurs et codages de chromosomes pour le problème du voyageur de commerce a été établi par Larrañaga et al. dans [Larrañaga et al., 1999]. La façon la plus classique de résoudre le problème consiste à coder les chromosomes sous forme de suite de villes. Un chromosome contient donc toutes les villes dans un certain ordre. C'est sur cet ordre que l'algorithme va agir afin de converger vers la combinaison de villes permettant de minimiser la distance à parcourir. Ainsi, le chromosome  $\{v_1, v_2, v_3, v_4, v_5\}$  signifie que le voyageur de commerce devra aller de son point de départ à la ville  $v_1$ , puis à  $v_2$ , ..., jusqu'à la ville  $v_5$ , puis enfin revenir à son point de départ. La fonction d'évaluation consiste à additionner les distances à parcourir ( $O(n)$ ). Concernant l'opérateur de croisement, il n'est pas possible d'associer les moitiés de deux chromosomes en garantissant la validité de la solution. Par exemple le croisement entre deux chromosomes tels que décrits dans la figure II.4a peut donner un chromosome non valide (qui n'appartient pas à l'ensemble  $S$  des solutions possibles). Ainsi, la figure II.4b montre un croisement consistant à recopier le début d'un des parents, puis de recopier les villes manquantes dans l'ordre dans lequel elles apparaissent dans le chromosome de l'autre parent.

L'opérateur de mutation consiste à inverser deux villes dans le chromosome afin de modifier ses gènes sans pour autant produire une solution invalide (voir figure II.3).



**Figure II.3:** Opérateur de mutation pour un chromosome modélisant une solution à un problème de voyageur de commerce

**L'optimisation par colonie de fourmis** (*Ant Colony Optimization*) est également une métahéuristique basée sur une observation du vivant utilisée pour résoudre le problème du



**Figure II.4:** Exemple d'une méthode non-valide (II.4a) et d'une autre méthode valide (II.4b) de croisement de deux chromosomes représentant une solution au problème de voyageur de commerce

voyageur de commerce. Dans [Dorigo et al., 1991], Dorigo et al. ont ainsi appliqué *Ant-System* pour résoudre une instance de 30 villes du problème du voyageur de commerce. Ils ont également établi que *Ant-cycle* donne de meilleurs résultats que les deux autres variantes. Ils ont également montré que :

- les meilleurs résultats sont obtenus en utilisant un nombre de fourmis proche du nombre de villes à visiter ;
- la valeur du paramètre  $Q$  peut être choisie arbitrairement et n'influence pas ni la capacité de l'algorithme à trouver une solution optimale, ni la rapidité de cette convergence ;
- la disposition initiale des fourmis sur le graphe n'influence que très peu la convergence de l'algorithme (avec des résultats légèrement meilleurs en utilisant une répartition aléatoire) ;
- l'utilisation d'une version élitiste permet d'améliorer la convergence.
- l'utilisation de bons paramètres permettait de trouver beaucoup plus rapidement des solutions proches de l'optimal.

Les auteurs ont comparé les résultats obtenus avec la stratégie *Ant-cycle* avec les performances des heuristiques *2 – opt* et *Lin – Kernighan* (voir II.2.3.2.2). Les résultats montrent que l'algorithme fourmis dépasse les performances de l'heuristique *2 – opt* et donne le même résultat que l'algorithme de *Lin – Kernighan*.

Concernant le paramétrage, Dorigo et al. ont indiqué que  $\{\alpha=1, \beta=2, \rho=0.5, Q^*=100, e=5\}$  est une bonne configuration permettant de résoudre le problème de voyageur de commerce de taille 30 en utilisant *Ant-cycle*.

Dans [Dorigo and Gambardella, 1997], Dorigo et Gambardella ont montré que l'algorithme *Ant Colony System* donne de meilleurs résultats avec le marquage constant de phéromone sur diverses instances du problème de taille 50 ainsi que sur des instances bien connues de 30 et 48 villes. Le score moyen ainsi obtenu sur 25 exécutions est meilleur que celui obtenu en utilisant  $Ant - Q$ , ou sans utiliser le marquage en fonction du meilleur chemin trouvé ( $\Delta\tau_{ij} = 0$ ), ou encore sans utiliser le marquage local.

Selon les auteurs le paramétrage de l'algorithme est relativement indépendant du problème. Ils ont défini l'ensemble de paramètre suivant :  $\{\beta=2, q_0=0.9, \alpha=\rho=0.1, \tau_0=(n \cdot L_{nn})^{-1}\}$  où  $L_{nn}$  correspond à la longueur du chemin construit grâce à l'heuristique du plus proche voisin (voir II.2.3.2.1) et  $n$  est le nombre de villes de l'instance du problème. Comparé aux autres métahéuristiques comme notamment le recuit simulé ou les algorithmes génétiques, Dorigo et Gambardella ont montré qu'*ACS* donne de meilleurs résultats sur des problèmes de taille 50, 75 et 100. *ACS* permet à chaque fois de trouver la solution optimale et ce plus rapidement que les autres méthodes.

Les algorithmes fournis sont, avec les algorithmes génétiques, la méthode la plus efficace pour trouver rapidement une solution proche de l'optimale au instances de taille diverses du problème de voyageur de commerce. Toutefois, l'inconvénient majeur de cette métahéuristique concerne inexistance, à l'heure actuelle, de preuve de convergence pour toutes les variantes d'algorithmes fournis (voir [Gutjahr, 2000, Dorigo and Blum, 2005]).

## II.2.4 Généralisation à $m$ voyageurs : The Multiple Traveling Salesman Problem (M-TSP)

### II.2.4.1 Définition

Le problème de voyageur de commerce évoqué précédemment (voir II.2.1) est défini pour 1 unique voyageur. Néanmoins, il existe une version généralisée à  $m$  voyageurs, où  $M$  est l'ensemble des voyageurs de commerce, qui caractérisée par la minimisation de la distance des tournées de plusieurs voyageurs sur un ensemble de  $n$  villes. Les villes doivent être ainsi visitées une et une seule fois par n'importe lequel des voyageurs. Deux voyageurs différents ne peuvent donc

pas visiter la même ville dans leur tournée respective. Comme dans le problème classique, la tournée de chaque voyageur doit commencer et se terminer au dépôt. Il est évident que lorsque  $m=1$ , il s'agit du problème classique du voyageur de commerce comme défini précédemment.

Cette généralisation du problème est encore plus difficile que le problème classique (où  $m = 1$ ) car il s'agit de déterminer l'affectation des villes aux voyageurs ainsi que d'optimiser l'ordre de passage des voyageurs dans les villes de leur tournée respective. Il n'existe donc pas non plus d'algorithme polynomial pour le résoudre et toutes les variantes du problème classique se retrouvent dans le problème des multi-voyageurs de commerce (PMVC).

Les variantes les plus courantes utilisent :

- un graphe asymétrique (Asymmetric M-TSP) :  $G = (V, A, w)$  ;
- des vitesses différentes en fonction des voyageurs :  $G = (V, E, w_k)$  si le graphe est symétrique, ou  $G = (V, A, w_k)$  dans le cas asymétrique,  $\forall k \in M$  ;
- une fonction objectif basée sur la minimisation :
  - de la somme des distances parcourues par les voyageurs (critère *MinSum*) ;
  - de la distance du plus long tour parmi les tours de tous les voyageurs (critère *MinMax*, voir [França et al., 1995]).
- une fonction objectif multicritère (répondre aux contraintes en minimisant  $m$  et la somme des longueurs des tours de chaque voyageur) ;
- plusieurs dépôts : il existe deux sous variantes :
  - en fin de parcours, les voyageurs doivent retourner au dépôt depuis lequel ils ont débuté leur tournée respective départ (voir [Laporte et al., 1988]) ;
  - en fin de parcours, les voyageurs doivent retourner dans un dépôt, peu importe lequel (voir [GuoXing, 1995]) :
    - il peut y avoir plusieurs voyageurs dans le même dépôt ;
    - il ne peut y avoir qu'un seul voyageur par dépôt.
- des contraintes supplémentaires :
  - Fenêtres de temps (M-TSP-TW) : chaque ville doit être visitée au cours d'une certaine période de temps ;
  - Capacité : un voyageur doit repasser par son point de départ dès qu'il a visité  $n$  villes ;
  - Précédence : un voyageur doit visiter certaines villes avant de pouvoir visiter d'autres ;

Les versions du problème comportant une contrainte de capacité sont généralement catégorisés

en tant que problèmes de tournées de véhicules (Vehicle Routing Problem) et seront évoqués dans la partie II.3 de ce manuscrit.

#### **II.2.4.2 Problèmes rencontrés**

Contrairement au PVC classique, la littérature ne contient que très peu de références au PMVC. En 2006, Bektas a dressé, dans [Bektas, 2006], une liste des problèmes réels pouvant être résolus grâce à une modélisation sous forme de *m-TSP* :

- Le problème de l'imprimeur de presse : il s'agit d'affecter des pages à imprimer à des paires de rouleaux d'impression (recto-verso) de forme diverses (4 pages, 6 pages, 8 pages) afin d'imprimer des versions différentes d'un journal périodique. Il faut déterminer quelle forme sera utilisée dans quelle série d'impression et à quel endroit dans la série.
- Le problème de l'annonceur de presse : il s'agit de déterminer les publicités à imprimer dans les encarts de journaux en fonction de leur ville ou région de distribution.
- Le problème de gestion de ressources : il s'agit de répartir un ensemble de ressources (employés, véhicules...) sur un ensemble de tâches à effectuer à des endroits différents. Par exemple : dans le problème des réparateurs de cabines téléphoniques, un ensemble de réparateurs doivent réparer un certain nombre cabines téléphoniques réparties géographiquement. Il faut ici minimiser la distance totale parcourue par les réparateurs, ou minimiser le nombre de réparateurs utilisés.
- Le problème des bus de ramassage scolaires : une flotte de bus scolaires doivent parcourir un ensemble d'arrêts. L'objectif est de minimiser le nombre de bus ainsi que la distance parcourue tout en assurant que la capacité des bus permettent de transporter tous les élèves et que le temps d'une tournée ne dépasse pas une durée maximale prédéfinie.
- La planification de missions : déterminer l'ordonnancement optimal afin que les ressources accomplissent les missions dans le temps le plus court possible.

Bektas indique également que le PMVC peut être utilisé pour proposer des solutions à des sous-problèmes d'autres problèmes d'optimisation comme l'ordonnancement et l'affectation des portiques de (dé)chargement de navires sur un terminal à conteneurs par exemple. Dans [Kim and Park, 2004], Kim et Park utilisent une modélisation sous forme de PMVC afin de définir des bornes inférieures de façon précise pour leur algorithme de Branch and Bound leur permettant de répartir les opérations de manutention en fonction de leur position géographique.

De part sa proximité avec le problème de tournées de véhicules, le PMVC peut être utilisé pour minimiser la distance parcourue par la flotte de véhicules ou minimiser le nombre de

véhicules utilisés pour servir tous les clients en respectant les contraintes du problème. Ainsi, dans [Mitrovic-Minic and Krishnamurti, 2006], les auteurs proposent de minimiser le nombre de voyageurs de commerce dans le cadre d'un PMVC avec fenêtres de temps en modélisant 2 graphes de précédence : l'un sur le minimum des fenêtres de temps, et l'autre sur le maximum. Grâce à cette modélisation il est possible de déterminer de façon polynomiale le nombre de voyageurs nécessaires pour visiter toutes les villes sans jamais dépasser une fenêtre de temps.

#### **II.2.4.3 Formulation du problème**

Bektas a identifié plusieurs formulations du problème dans la littérature.

Une formulation sous forme de programmation linéaire permet à la méthode du simplexe de Dantzig (voir [Dantzig et al., 1954]), également utilisée pour le PVC, de rester applicable au PMVC. Cependant, les contraintes dérivées permettant d'empêcher de créer des sous-tours ne contenant pas le dépôt augmentent exponentiellement avec la taille du problème. Miller et al. ont proposé dans [Miller et al., 1960] d'introduire d'autres variables continues permettant d'éviter ce problème en générant un nombre polynomial de contraintes d'élimination de sous-tours. Laporte et Norbert ont également proposés une formulation du problème induisant un nombre exponentiel de contraintes d'élimination de sous-tours. Leur formulation permet de minimiser à la fois la distance totale parcourue ainsi que le nombre de voyageurs de commerce.

Tout comme pour le PVC classique, Christofides et al. ont proposé une formulation basée sur l'utilisation d'un arbre caractérisé par un centre de degré  $k$  (le sommet représentant le dépôt possède  $k$  arcs adjacents) afin de résoudre le problème (voir [Christofides et al., 1981]). Laporte [Laporte and Nobert, 1980] a étendu utilisation de cette formulation afin de déterminer une borne inférieure pour un problème de tournée de véhicules (voir II.3).

Enfin, une formulation basée sur des flots a été dérivée d'un modèle de problème de tournées de véhicules proposé par Christofides ([Christofides et al., 1981]) est adaptée au PMVC en excluant les contraintes de capacité et de coût.

#### **II.2.4.4 Méthodes de résolution**

Dans [Bektas, 2006], Bektas a également indiqué les méthodes de résolution utilisées (exactes et approchées). Il existe selon lui 2 façons de résoudre le  $m$ -TSP :

- résoudre  $m$  fois le problème de voyageur de commerce ;
- modéliser le  $m$ -TSP sur 1 seul graphe et résoudre le problème de voyageur de commerce.

#### **II.2.4.4.1 Transformation en problème de voyageur de commerce classique**

Afin de pouvoir utiliser les méthodes de résolution du problème classique de voyageur de commerce, il est courant de chercher à transformer le PMVC en PVC classique. Le principe le plus répandu consiste à ajouter  $m-1$  villes fictives représentant le dépôt pour les  $m$  voyageurs de commerce. Les poids des arêtes entre ces sommets fictifs sont fixés à  $+\infty$  et les coûts des arêtes reliant ces sommets aux autres sommets sont fixés à 0.

Bellmore et Hong ont montré dans [Bellmore and Hong, 1974] que le PMVC asymétrique avec  $m$  voyageurs et  $n$  villes peut être converti en problème asymétrique classique de voyageur de commerce avec  $(m+n-1)$  nœuds. Un coût est associé à chaque voyageur et est pris en compte dès que le voyageur est utilisé dans la solution. Hong et Padberg ont également développés un graphe contenant  $(n + m + 4)$  sommets. Ainsi, Russell a proposé une heuristique de résolution basée sur la transformation du PMVC en PVC grâce à un graphe étendu (voir [Russell, 1977]). Leur algorithme est une extension de l'heuristique de Lin-Kernighan (voir II.2.3.2.2).

Jonker et Volgenant (voir [Jonker and Volgenant, 1988]) ont quant à eux utilisé le même nombre de sommets que pour le PVC classique mais en réduisant le nombre d'arcs. Ils ont indiqués que d'ajouter des copies du dépôt dégrade fortement le problème.

En ce qui concerne la résolution directe, c'est-à-dire sans transformation du problème multi voyageurs, plusieurs approches ont été définies au fil du temps.

#### **II.2.4.4.2 Résolution exacte**

Laporte et Norbert ont été les premiers à proposer une méthode de résolution exacte au PMVC (voir [Laporte and Nobert, 1980]). Leur algorithme est basé sur la méthode du *Branch and Price* et consiste à relaxer certaines contraintes du problème. Après chaque solution proposée, l'algorithme vérifie qu'aucune contrainte n'est violée. Si c'est le cas une nouvelle contrainte est introduite afin d'éliminer les sous-tours. Une autre version de leur algorithme consiste à évaluer les violations de contraintes avant d'obtenir une solution complète et montre de meilleurs résultats. Gavish et Srikanth ont proposé une méthode de résolution basée sur le principe du *Branch and Bound* (voir [Gavish and Srikanth, 1986]) pour résoudre des instances de taille plus importante du PMVC symétrique. Enfin, Gromicho et al. ont proposé un algorithme de *Branch and Bound* pour résoudre le PMVC où le nombre de voyageurs est fixé (voir [J et al., 1992]). Ils utilisent une procédure d'affectation sur le problème relaxé et utilisent une

borne inférieure déterminée grâce à différentes procédures de relaxation (r-arborescence et r-anti-arborescence). Ce procédé a permis de résoudre des instances de taille  $n=120$  et  $2 \leq m \leq 12$ .

#### II.2.4.4.3 Résolution approchée

Seules les métaheuristiques utilisées pour résoudre le problème classique peuvent être appliquées au problème multi-voyageurs sans requérir de transformation du problème. Ainsi, Ryan et al. ont utilisé une recherche tabou pour résoudre le PMVC avec fenêtres de temps (voir [Ryan et al., 1998]). Song et al. ont utilisé un algorithme de recuit simulé pour résoudre de façon approchée le PMVC avec des coûts fixes associés aux voyageurs. Ils ont réussi à résoudre des problèmes contenant 400 villes et 3 voyageurs (voir [Song et al., 2003]).

En 1989, Wacholder et al. ont proposé une méthode basée sur les réseaux de neurones d'Hopfield pour résoudre le PMVC (voir [Wacholder et al., 1989]). Toutefois leur modèle a été jugé trop complexe. Dans [Hsu et al., 1991], les auteurs ont développé un réseau de neurones permettant de résoudre successivement  $m$  problèmes classiques de voyageur de commerce. En 1994, Vakhutinsky et al. ont proposé un réseau de neurones artificiels basé sur la méthode du filet élastique afin de résoudre le PMVC (voir [Vakhutinsky and Golden, 1994]). Dans [Somhom et al., 1999] et [Modares et al., 1999], Somhom et al. ont introduit une méthode de résolution à base de réseau de neurones pour résoudre le problème des multi-voyageurs de commerce en prenant comme objectif de minimiser la distance de la route la plus longue parmi les routes des voyageurs. Cet objectif a été nommé *MinMax*. Les performances de cet algorithme sont meilleures que celles des méthodes basées sur le filet élastique.

Fogel a introduit une approche évolutionnaire considérant deux voyageurs de commerce et une fonction d'évaluation minimisant la différence de distance entre les deux tours (voir [Fogel, 1990]). L'auteur a montré que les solutions obtenues étaient très proches de la solution optimale. D'autres auteurs ont utilisé des algorithmes génétiques afin de résoudre le PMVC. Une approche récente de Tang et al. (voir [Tang et al., 2000]) consiste à transformer le PMVC en problème de voyageur de commerce classique et d'appliquer un algorithme génétique. Les auteurs utilisent ainsi un unique chromosome contenant les tournées des voyageurs, codées par des entiers (indices des villes), et séparées par un gène spécifique permettant d'identifier la fin d'un tour et le début de la tournée du voyageur suivant (voir fig. II.5). Un chromosome aura donc une taille égale à  $m + n - 1$ . Cette technique a été baptisée “one-chromosome technique”.

2	5	14	6	-1	1	11	8	13	-2	4	10	3	-3	12	15	9	7
Voyageur n° 1				Voyageur n° 2				Voyageur n° 3				Voyageur n° 4					

**Figure II.5:** Exemple de chromosome codé selon la méthode de Tang et al. pour  $n = 15$  et  $m = 4$ .

Villes :	1	2	5	12	4	14	15	6	11	9	7	8	10	3	13
Voyageurs :	2	1	1	4	3	1	4	1	2	4	4	2	3	3	2

**Figure II.6:** Exemple de chromosome codé selon la méthode à deux chromosomes pour  $n = 15$  et  $m = 4$ .

En 2001, Park a introduit un codage utilisant 2 chromosomes (voir [Park, 2001]), l'un pour les villes et l'autre pour les voyageurs (voir fig. II.6). L'association de ces deux chromosomes permet d'obtenir des tuples {ville, voyageur} représentant une solution au problème. Chaque voyageur devra parcourir les villes associées dans l'ordre où elles apparaissent dans le chromosome des villes. Ce codage est appelé “two-chromosomes technique”.

Notons qu'avec ces deux types de codage, avec un ou deux chromosomes, la population peut contenir plusieurs individus décrivant les mêmes solutions sans avoir le même code génétique. La population peut donc contenir plusieurs fois les mêmes solutions avec des individus différents. Ainsi, Carter et Ragsdale ont introduit une nouvelle forme de codage utilisant un seul chromosome mais contenant deux parties (voir [Carter and Ragsdale, 2006]). La première partie du chromosome contient  $n$  gènes représentant les villes à parcourir. La seconde partie du chromosome contient  $m$  gènes qui décrivent le nombre de villes associées à chaque voyageur (voir fig. II.7). Ce procédé permet de réduire l'espace de recherche en évitant la récurrence des solutions. En revanche les opérations de croisement et de mutations deviennent plus délicates car la somme des gènes de la seconde partie du chromosome doit être égale à  $n$ .

Les résultats de l'étude de Carter et Ragsdale (voir [Carter and Ragsdale, 2006]) montrent

Villes																Villes par voyageur			
2	5	14	6	1	11	8	13	4	10	3	12	15	9	7	4	4	3	4	
$v_1$				$v_2$				$v_3$			$v_4$				$v_1$	$v_2$	$v_3$	$v_4$	

**Figure II.7:** Exemple de chromosome à deux parties pour  $n = 15$  et  $m = 4$ .

Population initiale :

$m_1 =$	2	5	14	6
$m_2 =$	1	11	8	
$m_3 =$	4	10	3	13
$m_4 =$	12	15	9	7

Après croisement entre  $m_1$  et  $m_3$  :

$m_1 =$	2	5	10	3	13
$m_2 =$	1	11	8		
$m_3 =$	4	14	6		
$m_4 =$	12	15	9	7	

**Figure II.8:** Exemple du modèle génétique multi-chromosomes de Kiràly et Abonyi pour  $n = 15$  et  $m = 4$

que les codages utilisant un seul chromosome sont plus performant que celui à deux chromosomes. Le codage par chromosomes à deux parties se révèle également plus performant que le codage avec un seul chromosome à une seule partie et spécialement lorsque l'objectif est de répartir l'effort entre les différents voyageurs.

Plus récemment, Kiràly et Abonyi ont proposé un algorithme génétique utilisant la technique des multi-chromosomes (voir [Kiràly and Abonyi, 2011]). Dans leur modèle, il existe  $m$  chromosomes et les opérateurs génétiques consistent à échanger des morceaux de code génétique entre ces chromosomes (voir Fig. II.8). Cette modélisation revient, sous une forme différente, à utiliser le codage en un seul chromosome à deux parties. Le principal avantage de ce codage réside donc dans sa clarté et sa facilité d'implémentation.

Junjie et Dingwei ont développé un algorithme fourmi pour résoudre directement le problème des multi-voyageurs de commerce (voir [Junjie and Dingwei, 2006]). Ils utilisent une contrainte supplémentaire afin de limiter le nombre de villes qu'un voyageur peut visiter. Ainsi, l'algorithme fonctionne de la façon suivante. Une fourmi débute son parcourt en modélisant le tour du voyageur  $i$ . Ce voyageur devra parcourir  $n_i$  villes (avec  $n_i$  nombre aléatoire et  $n_i \leq MAX$ ). La fourmi choisit sa destination selon la règle définie par Dorigo et al. (voir [Dorigo, 1992]) parmi la liste des villes non visitée. Lorsque la fourmi a parcourut ses  $n_i$  villes, elle continue en cherchant le tour du voyageur suivant ( $j$ ) en parcourant  $n_j$  villes (avec  $n_j$  nombre aléatoire tel que  $n_j \leq MAX$ ). Le parcourt d'une fourmi s'arrête lorsque toutes les villes ont été visitées. Le processus

marquage s'applique lorsqu'une solution a été trouvée (marquage local) et lorsqu'un nombre défini de solutions ont été trouvées (marquage global). La formule de marquage correspond à celle de Dorigo et al. utilisée par *Ant-System*. Leur algorithme présente de meilleurs résultats que l'algorithme génétique modifié sur des instances de grandes taille.

En 2008, Vallivaara a proposé un algorithme fourmi afin de résoudre la version *MinMax* du problème (voir [Vallivaara, 2008]). L'algorithme s'intitule *Team Ant Colony Optimization* (TACO) et consiste à remplacer chaque fourmi des algorithmes fourmis traditionnels, comme *Ant-System* par exemple, par des groupes de fourmis. Chaque groupe, ou équipe, de  $m$  fourmis (où  $m$  correspond au nombre de voyageurs de commerce) possède sa propre liste de villes visitées et les membres du groupe choisissent leur destination parmi les villes de cette liste. À chaque itération c'est la fourmi ayant la tournée la plus courte se déplace. Ce processus constructiviste permettant de distribuer les villes de façon uniforme pour chaque voyageur de commerce conduit souvent à des routes sous-optimales. C'est pourquoi Vallivaara a introduit une procédure de reconstruction des chemins qui consiste à vérifier, pour chaque ajout de ville dans le chemin d'une fourmi, si l'insertion de cette ville dans la tournée d'un autre membre du groupe ne permettrait pas d'améliorer la solution. Enfin, Vallivaara utilisent une procédure *2-opt*, puis *3-opt* afin d'améliorer la solution de l'algorithme TACO. Les résultats obtenus sont meilleurs que ceux des réseaux de neurones sur diverses instances du problème contenant de 51 à 417 villes pour  $m \in \{2, 3, 4\}$ .

## Conclusion

Dans cette section nous avons introduit le problème du voyageur de commerce ainsi que ses diverses variantes. Nous avons dressé un état de l'art des méthodes les plus couramment utilisées afin de résoudre le problème de voyageur de commerce de façon exacte ou approchée et en détaillant dans ce dernier cas les méthodes de construction, d'amélioration et les mét-heuristiques.

Les méthodes de résolution exactes permettent de résoudre des instances de petite taille grâce à des méthodes de programmation dynamique et de Branch and Bound. Des méthodes basée sur la programmation linéaire permettent de résoudre des instances de taille significative (jusqu'à 200 villes). Combinées, les méthodes de Branch and Bound et la programmation linéaire, apportent des solutions à des problèmes de taille importante (jusqu'à 85900 villes) mais le temps de calcul est encore très long (136 années CPU à l'échelle d'un AMD Opteron 250 de 2.4GHz).

Les méthodes approchées permettent, quant à elles, de fournir des solutions exactes ou

proches de l'optimal à des problèmes de taille importante dans un temps raisonnable. Elles concernent les méthodes de construction heuristique (plus proches voisins, plus proches fragments, recherche d'arbres couvrants de poids minimal, algorithme de Christofides, tour bitonique, couplages successifs), les méthodes d'amélioration de tour (V-opt et k-opt ainsi que ses dérivées Lin-Kernighan, 2-opt et 3opt, et d'autre part les méthodes à base de chênes de Markov (MCMC)). Mais ce sont les métaheuristiques qui permettent d'approcher les solutions optimales des problèmes les plus grands (recherche tabou, recuit simulé, réseaux de neurones artificiels, algorithmes génétiques, les algorithme de colonie de fourmis).

Pour plus d'informations sur les méthodes locales d'optimisation pour le problème de voyageur de commerce, le lecteur pourra se référer à [Johnson and Mcgeoch, 1997]. Les instances du problème évoquées dans cet état de l'art sont disponibles à cette adresse : <http://www.tsp.gatech.edu/>.

La version généralisée du problème à  $m$  voyageurs a également été étudiée. Il existe deux paradigmes de résolution du problème : d'une part résoudre  $m$  problèmes de voyageur de commerce, et d'autre part résoudre le problème classique issu de la transformation du problème multiple. Là aussi plusieurs méthodes de résolutions existent. Concernant les méthodes exactes, des méthodes de *Branch and Price* et de *Branch and Bound* permettent de résoudre des instances de 120 villes pour 2 à 12 voyageurs. Les méthodes approchées permettent de résoudre des instances plus importantes grâce à des méta-heuristiques (recherche tabou, recuit simulé, réseaux de neurones artificiels, algorithmes génétiques et algorithmes fourmis).

## II.3 Problèmes de tournées de véhicules

### II.3.1 Définition

Le problème de tournées de véhicules (*Vehicle Routing Problem* : VRP) est une reformulation du problème des multi-voyageurs de commerce (*Multiple Traveling Salesman Problem* : M-TSP) où chaque voyageur correspond à un véhicule. Dans le cas où il n'y a qu'un seul véhicule (*Single Vehicle Routing Problem* : SVRP), le problème correspond à un problème de voyageur de commerce classique (*Traveling Salesman Problem* : TSP). L'objectif reste le même que pour le TSP : il faut, en partant du dépôt, visiter chaque client (ville), puis rentrer au dépôt, tout en minimisant la distance totale parcourue. La première formulation du problème a été introduite en 1959 par Dantzig et Ramser [Dantzig and Ramser, 1959]. Les auteurs parlent ainsi pour la première fois de problème de tournées de véhicule et de problème de la feuille de trèfle. En effet, les tournées de chaque véhicule commencent et se terminent au dépôt, formant ainsi une figure

géométrique en forme de feuille de trèfle. On retrouve ce problème dans de nombreuses applications concrètes, notamment dans le domaine de la logistique. Ces applications diverses ont permis de favoriser le développement de la recherche sur les VRP et ont abouties à la modélisation de différentes variantes du problème.

### II.3.2 Les variantes du problème

La spécificité du VRP vis-à-vis du problème de voyageur de commerce, réside dans la présence de contraintes supplémentaires sur les ressources (véhicules pour le VRP, voyageurs pour le M-TSP). Ainsi, il existe différentes variantes du VRP :

- **Capacitated Vehicle Routing Problem (CVRP)** : les véhicules ont une capacité maximale de chargement. Il s'agit de la version la plus classique du problème. L'exemple le plus utilisé est celui de la compagnie de fuel qui doit livrer ses clients pour qu'ils puissent alimenter leur chaudière. La compagnie dispose de  $m$  véhicules pour livrer  $n$  clients. Les véhicules peuvent emporter  $Q$  litres de fuel et livrent  $q_i$  litres au client  $i$ . Lorsque le camion de livraison est vide, il doit retourner au dépôt pour être rempli afin de procéder à une autre tournée. Il existe des versions du problème dans lesquelles tous les véhicules ont la même capacité. On dit que la flotte de véhicule est homogène. Dans d'autres versions la flotte est hétérogène (voir [M. et al., 1999]).
- **Distance Constrained Vehicle Routing Problem (DCVRP)** : une contrainte sur la distance maximale par tournée est ajoutée. Une fois que les véhicules ont atteint cette distance, ils doivent revenir au dépôt.
- **Split Delivery Vehicle Routing Problem (SDVRP)** : un client peut être livré par plusieurs véhicules. Ainsi, un client demandant 500 litres de fuel peut être livré par un premier camion pour une quantité de 300 litres, puis de 200 litres par un second camion.
- **Vehicle Routing Problem with Pickup and Delivery (VRPPD)** : les tournées comportent à la fois des points de collecte et des points de livraison de marchandise. Il faut dans ce cas prendre en compte la collecte de marchandise afin d'être en mesure de livrer les clients. L'exemple le plus répandu de ce problème est celui de l'entreprise postale. Une entreprise emploie  $m$  postiers qui doivent collecter le courrier dans des boîtes spéciales et livrer ensuite le courrier collecté aux usagers. Il faut donc définir les tournées de chaque postier en fonction de l'emplacement des boîtes aux lettres de la compagnie et des boîtes

aux lettres des clients. Ce problème est également appelé Pickup and Delivery Problem (PDP). Il existe également une version de ce problème dans laquelle l'objet livré doit être le dernier collecté (structure Last-In First-Out) afin de modéliser les problèmes rencontrés dans le monde réel par certaines entreprises de livraison où les marchandises ne peuvent pas être déchargées sans vider les marchandises plus proches de l'ouverture du camion au préalable (livraison de gros électroménager par exemple, ou de meubles, etc.).

- **Vehicle Routing Problem with Time Windows** (VRPTW) : les clients doivent être livrés dans un intervalle de temps fixé. Si le véhicule arrive en avance, c'est-à-dire avant le début de la fenêtre de temps, il devra attendre. Les fenêtres de temps peuvent être dures (hard) ou molles (soft). Avec des fenêtres de temps dures, il est strictement interdit de dépasser l'intervalle de temps. Les fenêtres de temps font partie des contraintes du problème. Avec des fenêtres molles, il faut éviter de les dépasser. Le respect des fenêtres de temps fait partie de la fonction objectif du problème. Dans ce cas, la politique la plus courante est de pénaliser le dépassement des fenêtres de temps par un coût supplémentaire. L'ajout de contrainte temporelle peut également être rencontré dans les autres versions du problème comme le Capacitated Vehicle Routing Problem with Time Windows (CAVRP-TW) ou le Vehicle Routing Problem with Pickup and Delivery and Time Windows (VRPPD-TW).
- **Multiple Depot Vehicle Routing Problem** (MDVRP) : dans cette version il existe plusieurs dépôts. Il est possible de rencontrer les formes du problèmes où chaque véhicule doit débuter et terminer sa tournée au même dépôt : dans ce cas il s'agit de résoudre  $k$  VRP différents (où  $k$  est le nombre de dépôts), ou la version du problème où un véhicule peut terminer sa tournée dans n'importe quel dépôt. Par exemple, une grande compagnie de livraison à domicile de fuel possède  $k$  dépôts de carburant répartis sur un territoire. Lorsqu'un véhicule débute une tournée il part avec une certaine quantité de carburant à livrer. Une fois sa tournée terminée il se rend au dépôt le plus intéressant afin de refaire le plein de fuel et de pouvoir repartir pour une autre tournée sans avoir besoin obligatoirement de retourner au dépôt initial et donc de couvrir une distance supplémentaire.
- **Vehicle Routing Problem with Backhauls** (VRPB) : ce problème inclus un ensemble de clients à la fois à livrer ainsi qu'à reprendre des marchandises afin de les rapporter au dépôt. Par exemple, une société de vente en ligne de produits d'électroménager livre ses clients par camions. Lorsqu'un client constate une panne sur un appareil en garantie, la compagnie doit venir à ses frais chercher le produit afin de pouvoir le réparer puis le retourner au client. La différence avec le VRPPD est la provenance des produits à livrer/-

collecter. Dans le VRPPD les marchandises sont livrées et collectées chez le clients. Dans le VRPD, les véhicules partent du dépôt avec la marchandise à livrer, puis visitent les clients et collectent les marchandises à ramener au dépôt. La plupart du temps, les VRPD ont comme contrainte de réaliser toutes les livraisons avant de pouvoir effectuer une collecte.

- **Dial A Ride Problem (DARP)** : ce problème consiste à transporter des personnes à la demande. Par exemple, une compagnie de taxis collecte des demandes de transport par téléphone. Une fois les demandes connues les tournées des taxis sont calculées afin de minimiser la distance parcourue pour transporter les clients. Le *Dial A Ride Problem* est un sous-problème du *Pickup and Delivery Problem* avec une contrainte supplémentaire sur la durée maximale de transport pour chaque passager.

Les durées de manutention pour la livraison (ainsi que pour les collectes dans les versions du problème concernées) peuvent ne pas être négligeables et il est parfois indispensable de les prendre en compte dans le calcul de la solution optimale, notamment lors de l'utilisation de fenêtre de temps. Dans certains problèmes, le temps de livraison (et de collecte) est fixe, alors que dans d'autres versions la durée de manutention peut dépendre de la marchandise, du client ou de l'heure de livraison ou de collecte. Ainsi, en reprenant l'exemple de la compagnie postale, le temps de livraison du courrier est le même pour tous les clients car les boîtes aux lettres sont situées devant les habitations, dans la rue. En revanche, dans l'exemple des livreurs d'électroménager, la livraison prendra plus de temps si le clients habite au 6<sup>ème</sup> étage dans un immeuble sans ascenseurs, que s'il habite au rez de chaussé. De même, la livraison sera plus rapide si le client a commandé un article de taille et de poids permettant de le déposer dans la boîte à lettres plutôt qu'un réfrigérateur par exemple.

### II.3.3 Problème décisionnel et formulation mathématique

Les problèmes de tournées de véhicule étant une reformulation du M-TSP, ils sont également NP-difficiles. Le problème peut-être formulé en problème décisionnel où l'on pose la question suivante : doit-on insérer le client  $j$  dans la tournée du véhicule  $k$  après le client  $i$  ?

Golden et al. (voir [Golden et al., 1977]) ont proposé une formulation du problème utilisant des variables binaires à 3 indices. Ainsi, la variable  $x_{ij}^k$  correspond à la réponse à la question précédente, c'est-à-dire  $x_{ij}^k = 1$  si le véhicule  $k$  doit parcourir l'arc  $(i, j)$  dans sa tournée,  $x_{ij}^k = 0$  sinon. Le problème décisionnel est ainsi NP-Complet.

Le graphe représentant les clients est complet. Dans le problème symétrique le graphe est non orienté et est défini ainsi :  $G = (V, E)$ , alors que dans la version asymétrique le graphe est

orienté :  $G = (V, A)$ .  $V$  est l'ensemble des sommets du graphe qui correspondent aux clients à livrer ( $v_1, \dots, v_n$ ) et  $v_0$  représente le dépôt. La pondération des arcs correspond à la distance à parcourir pour relier les deux clients connectés par l'arc. Ainsi, le coût  $c_{ij}$  est la distance entre le client  $i$  et le client  $j$ .

Mathématiquement le problème s'écrit de la façon suivante :

$$(II.15) \quad \min \sum_{i=1}^n \sum_{j=1}^n \left( c_{ij} \sum_{k=1}^m x_{ij}^k \right)$$

La contrainte indiquant que les clients ne doivent être livrés qu'une seule fois s'écrit selon les équations II.16 et II.17 :

$$(II.16) \quad \sum_{i=1}^n \sum_{k=1}^m x_{ij}^k = 1, \forall 1 \leq j \leq n$$

$$(II.17) \quad \sum_{j=1}^n \sum_{k=1}^m x_{ij}^k = 1, \forall 1 \leq i \leq n$$

La contrainte de continuité de la tournée indiquant qu'un véhicule livrant un client repart de son point de livraison après avoir accompli sa tâche est la suivante :

$$(II.18) \quad \sum_{i=1}^n x_{ip}^k - \sum_{j=1}^n x_{pj}^k = 0, \forall 1 \leq k \leq m ; \forall 1 \leq p \leq n$$

Les véhicules débutent et terminent leur tournée au dépôt :

$$(II.19) \quad \sum_{i=1}^n x_{i0}^k = 1, \forall 1 \leq k \leq m$$

$$(II.20) \quad \sum_{j=1}^n x_{0j}^k = 1, \forall 1 \leq k \leq m$$

Les clients ne peuvent demander plus que le véhicule est capable de transporter au cours de sa tournée :

$$(II.21) \quad \sum_{i=0}^n \sum_{j=1, j \neq i}^n x_{ij}^k \cdot q_j \leq Q^k, \forall 1 \leq k \leq m$$

Enfin la variable  $x_{ij}^k$  est binaire :

$$(II.22) \quad x_{ij}^k \in \{0, 1\}, \forall 0 \leq i, j \leq n, \forall 1 \leq k \leq m$$

### **II.3.4 Fonction objectif**

Concernant la fonction objectif, les plus courantes sont la minimisation de la :

- distance totale parcourue ;
- durée des tournées ;
- taille de la flotte de véhicules ;
- du coûts des tournées.

Il existe également des approches multi-objectifs (voir [ghazali Talbi, 2001]) dans lesquelles il est possible d'utiliser une combinaison linéaire de plusieurs de ces critères afin de transformer le problème multi-objectif en problème mono-objectif, ou de chercher une ou plusieurs solutions Pareto-optimales. En effet, certains objectifs peuvent être antinomiques. La réduction de la taille de la flotte de véhicules peut pousser les véhicules restant à devoir repasser par le dépôt afin d'être en mesure de livrer une quantité suffisante de marchandise aux clients et donc vont parcourir plus de distance qu'en utilisant plus de véhicules.

### **II.3.5 Méthodes de résolution**

Les problèmes de tournées de véhicules ont été largement étudiés au cours des dernières décennies. Dérivé du problème de voyageur de commerce, ce problème d'optimisation combinatoire peut-être résolu par des méthodes exactes ou approchées. Toutefois, les approches exactes restent très peu applicable pour des problèmes réels à cause de l'explosion combinatoire du nombre de solutions possibles. Les approches approchées font appel à des heuristiques ou à des métahéuristiques. En pratique, seules les méthodes approchées sont utilisées en raison du temps requis pour l'obtention de résultats sur des instances réelles de taille souvent importante. Le livre de Toth et Vigo (voir [Toth and Vigo, 2001]) propose un état de l'art complet des méthodes de résolution exactes et approchées des problèmes de tournées de véhicules. Les auteurs sont des spécialistes du domaine et il n'est pas du tout question de présenter un tour d'horizon exhaustif des méthodes de résolution du VRP dans cette thèse, mais de présenter les évolutions majeures dans la résolution du problème.

#### **II.3.5.1 Méthodes de résolution exacte**

Les méthodes exactes peuvent s'appliquer à des problèmes de taille raisonnable.

La première méthode consiste à énumérer toutes les solutions possibles et à mesurer la performance de chaque solution grâce à la fonction objectif. À la fin de l'énumération, la (ou

les) solution(s) optimale(s) est (sont) connu(s).

Une seconde méthode repose sur le principe du Branch and Bound et permet d'obtenir une solution exacte pour des problèmes de taille raisonnable. Dans [Fisher, 1994], Fisher utilise, dans le cadre d'un CVRP, une recherche d'arbre couvrant comportant  $K + n$  arcs où  $K$  est le nombre de véhicules et  $n$  le nombre de clients à visiter. Leur but est de rechercher un tel arbre ayant de plus deux arcs incident au dépôt ainsi que des contraintes supplémentaires liées à la capacité des véhicules et au principe de ne visiter chaque client qu'une et une seule fois. La résolution utilise un algorithme de type branch and bound utilisant une borne inférieure fournie par le problème dual obtenu après dualisation lagrangienne sur les contraintes. Fisher a ainsi résolu des problèmes comportant 100 clients et 10 véhicules.

Dans [Laporte et al., 1985], Laporte et al. étendent le principe développé par Dantzig et al. pour le TSP dans [Dantzig et al., 1954]. Leur algorithme de *Branch-and-Cut* utilise une borne calculée grâce à une méthode de *cutting planes* qui repose sur des inégalités liées aux contraintes d'élimination de sous-tours. Le *Branch-and-Bound* utilisant la borne inférieure calculée précédemment permet de résoudre des instances du CVRP comportant jusqu'à 60 villes. De la même façon, Cornuéjols et Hache ont étendu dans [Cornuejols and Harche, 1993] la méthode des *cutting planes* utilisée dans la résolution du problème de voyageur de commerce afin de résoudre le CVRP ainsi que le Geographical Vehicle Routing Problem (GVRP). Ce dernier problème est une relaxation du CVRP dans lequel les capacités des véhicules sont ignorées. Le GVRP est donc une autre formulation du  $m - TSP$ . Les auteurs ont ainsi résolu de façon exacte un problème comportant 18 clients pour 2, 3 et 4 véhicules.

D'autre part, dans [Bramel and Simchi-Levi, 2001], Bramel et Simchi-Levi utilisent une approche basée sur la couverture ensembliste (*Set Covering*). Ils utilisent une relaxation lagrangienne avec une borne inférieure efficace mais qui demande néanmoins un effort intense en terme de calculs, rendant cette méthode inapplicable même pour des instances de taille raisonnable.

Fukasawa et al. ont défini récemment une méthode de *Branch-and-Cut-and-Price* (voir [Fukasawa et al., 2006]) permettant de résoudre toutes les instances classiques de la littérature jusqu'à 135 clients. Enfin, Baldacci et al. ont développé dans [Baldacci et al., 2008b] une méthode basée sur la formulation de la couverture ensembliste permettant de fournir de meilleures bornes inférieures que la méthode de Fukasawa et al. et de façon plus rapide.

Toutes ces méthodes de résolution exactes ont pour point commun d'être envisageable pour des instances de taille raisonnable. Cependant, elles ne sont que rarement utilisées dans les applications réelles du problème au profit de méthodes approchées permettant d'obtenir des

solutions proches de l'optimal en un temps beaucoup plus court.

### II.3.5.2 Méthodes de résolution approchée

L'utilisation de méthodes approchées est devenue indispensable pour résoudre des instances réelles de problèmes de tournées de véhicules. Le principe est, comme pour les problèmes de voyageurs de commerce, d'orienter la recherche dans l'espace de solutions afin de n'en parcourir qu'une infime partie et ainsi d'obtenir un résultat rapidement et de préférence de bonne qualité. Ces heuristiques sont souvent facilement adaptable aux différentes variantes du problème.

Il existe deux types d'heuristiques. D'une part celle consistant à construire une solution de façon itérative en choisissant à chaque étape le client à insérer dans une route, et d'autre part celles utilisant une solution initiale de qualité médiocre en cherchant à l'améliorer au fil de l'exécution de l'algorithme.

#### II.3.5.2.1 Heuristiques constructivistes :

La première heuristique fut proposée par Dantzig et Ramser en 1959 ([Dantzig and Ramser, 1959]). Elle est basée sur un algorithme de programmation linéaire et permet d'obtenir une solution proche de l'optimal au CVRP. En 1964, Clarke et Wright ont amélioré l'algorithme de Dantzig et Ramser en mettant au point une heuristique gloutonne appelée *Saving Algorithm* (voir [Clarke and Wright, 1964]). Le principe est d'abord de déterminer des "économies" (*savings*)  $s_{ij}$  avec  $s_{ij} = c_{i0} + c_{0j} - c_{ij}$ ,  $\forall i, j \in \{1, \dots, n\}$  and  $i \neq j$  correspondant au coût, en terme de distance, de la fusion de deux tournées en un point. Les fusions de tournées permettant de réaliser la plus grande économie est réalisée jusqu'à ce qu'aucune économie ne puisse être obtenue.

En 1974, Gillet et Miller ont proposé l'heuristique de construction par balayage (*sweep heuristic*, voir [Gillett and Miller, 1974]). Cette heuristique consiste à regrouper les clients de façon à constituer des cercles connectés au dépôt. Dès qu'il est impossible d'ajouter un client à la tournée en respectant les contraintes du problème, un nouveau cercle est créé. Il s'agit d'une construction de type *Cluster First - Route Second*, c'est-à-dire que les clients sont d'abord regroupés en fonction de leur position, puis, dans un second temps, l'ordre de passage des véhicules dans chaque groupe de clients est déterminé. Fisher et Jakumar ont utilisé ce principe dans [Fisher and Jaikumar, 1981], rédigé en 1979 mais publié qu'en 1981. Ils ont résolu le VRP en déterminant les groupes de clients (*clusters*) par la résolution du problème d'affectation généralisé (*Generalized Assignment Problem*) connexe, puis en optimisant chaque tournée en

résolvant le problème de voyageur de commerce associé.

L'approche inverse consiste d'abord à construire une unique tournée regroupant tous les clients, puis à découper cette tournée en sous tournées pour chaque véhicule. On dit alors que la construction est de type *Route First - Cluster Second*. Cette heuristique a été introduite par Beasley en 1983 (voir [Beasley, 1983]). L'auteur génère plusieurs tournées géantes regroupant tous les clients, puis utilise l'algorithme de Floyd (voir [Floyd, 1962]) afin de calculer les meilleures sous-chemins dans chaque tour géant et ainsi de déterminer le découpage optimal en sous-tournées. Beasley souligne donc que l'utilisation de l'algorithme de Floyd a pour conséquence de borner la complexité de sa méthode au cube de la taille du problème :  $O(n^3)$ . Ulusoy a utilisé également une méthode *Route First - Cluster Second* pour résoudre un CVRP (voir [Ulusoy, 1985]). Son algorithme comporte 4 étapes. Tout d'abord, la résolution d'un problème de postier chinois (problème de voyageur de commerce sur les arêtes du graphe au lieu des sommets) permet d'obtenir un tour géant. Ensuite, la deuxième étape consiste à diviser le tour géant en tournées de façon à respecter les contraintes de capacité des véhicules. La troisième étape consiste à résoudre un problème de plus court chemin sur un nouveau graphe résultant de la transformation du graphe précédent où les sommets du nouveau graphe sont les arêtes du tour géant, et les arêtes du nouveau graphe sont les sous-tours obtenus à l'étape précédente. Enfin la dernière phase consiste à améliorer la solution à posteriori.

#### **II.3.5.2.2 Heuristiques d'amélioration :**

Concernant les méthodes par amélioration, les heuristiques utilisées pour le problème de voyageur de commerce se retrouvent également dans la résolution du VRP, comme par exemple  $k-opt$  ( $2-opt$  et  $3-opt$ ). Ces méthodes de recherche locale permettent d'améliorer les résultats fournis par les heuristiques constructivistes comme l'algorithme glouton de Clarke et Wright, ou la méthode d'Ulusoy par exemple. Ainsi, dans [Irñich et al., 2006], Irñich et al. formalisent une méthode générique de recherche locale pour les problèmes de tournées de véhicules. Les résultats montrent qu'un gain important en terme de vitesse de convergence est obtenu en utilisant un voisinage de type  $3-opt$ . Ces méthodes peuvent être utilisées dans plusieurs métaheuristiques basées sur des recherches locales comme la recherche tabou par exemple.

Plus de détails concernant les heuristiques appliquées aux problèmes de tournées de véhicules peuvent-être trouvés dans [Bodin et al., 1983] et [Laporte, 1992]. Plus de précision sur les méthodes de recherche locale, notamment sur les différents voisinages, sont apportés par Funke et al. dans

[Funke et al., 2005]. D'autre part, dans [Laporte et al., 2000], Laporte et al. dressent un tour d'horizon des heuristiques utilisées dans la résolutions des problèmes de tournées de véhicules. Ils classent les méthodes en deux catégories : les heuristiques classiques d'une part, et les heuristiques modernes d'autre part. Cependant la seconde catégorie fait état de la méthode de recherche tabou, qui appartient en réalité à la famille des métaheuristiques. En revanche, leur classification reste valable car en effet les heuristiques ont été développées bien avant les métaheuristiques. La qualité des solutions obtenues par ces dernières surpassent ainsi les résultats des heuristiques.

#### **II.3.5.2.3 Métaheuristiques :**

On distingue deux grandes classes de métaheuristiques pour le VRP. D'une part les méthodes de recherche de trajectoire qui consiste à partir d'une solution initiale, de plus ou moins bonne qualité, pour s'en éloigner en parcourant l'espace des solutions en suivant une trajectoire. D'autre part, les méthodes à population, consistent quant-à-elles à travailler sur un ensemble de solutions (population) et de le faire évoluer de façon itérative. L'intérêt de cette seconde classe est de distribuer l'exploration de l'espace des solutions.

##### **Méthodes à trajectoires :**

Dans les méthodes à trajectoires, dites de recherche locale, il est question de parcourir l'espace des solutions de façon à converger vers un optimum, tout en évitant de bloquer sur une solution localement optimale. Parmi ces méthodes, on peut citer le recuit simulé (*Simulated Annealing*), ou la recherche tabou (*Tabu Search*).

En 1993, dans [Osman, 1993], Osman a introduit une méthode de recuit simulé ainsi qu'une recherche tabou appliqués au CVRP sous contrainte de distance. Son recuit simulé construit des solutions voisines en échangeant un seul arc depuis la solution courante. Ceci permet de ne pas générer de solutions trop éloignées de la solution courante comme c'est le cas lorsqu'une solution est générée de façon aléatoire. L'autre avantage réside également dans la garantie d'une exploration complète des solutions voisines. L'algorithme de recuit simulé de Osman est donc une hybridation entre un recuit simulé classique (où les solutions sont générées de façon aléatoires) et une heuristique de recherche locale ( $k - opt$ ).

La méthode de recherche tabou d'Osman est basée sur celle originelle de Glover (voir

[Glover, 1989, Glover, 1990]). La liste tabou est modélisée par une matrice comportant  $(n + 1)$  lignes et  $m$  colonnes (où  $n$  est le nombre de clients, une ligne supplémentaire permet de représenter le passage par le dépôt (*shift*), et  $m$  est le nombre de véhicules disponibles).

Les 2 méthodes ont été testées sur 17 instances de la littérature comportant de 29 à 199 clients et sur 9 nouveaux problèmes de taille 50, 75, et 100 clients. Les résultats montrent que bien que le recuit simulé soit efficace vis-à-vis des meilleures solutions connues aux problèmes de la littérature, la recherche tabou donne de meilleurs résultats et requiert moins de temps de calcul. En effet, la recherche tabou donne de meilleures solutions à 14 des 17 problèmes et trouve la même solution au 3 autres. Le nombre de véhicules nécessaire est également moindre dans les résultats de la recherche tabou que dans les solutions de la littérature.

Parallèlement à Osman, dans [Gendreau et al., 1994], Gendreau et al. ont proposé l'algorithme *TABURROUTE* basé sur une recherche tabou. La différence avec la méthode d'Osman se situe au niveau du respect des contraintes. En effet, Gendreau et al. autorisent la construction de solutions qui ne respectent pas les contraintes afin de les utiliser comme solution étape, lors de la recherche de la solution optimale. Les solutions ne respectant pas les contraintes de capacité ou de distance sont alors pénalisées mais néanmoins autorisées. Les résultats sont comparables avec ceux d'Osman.

Bien d'autres méthodes tabou ont été appliquées avec succès à différentes variantes du problème de tournées de véhicules comme celles de Cordeau et al. (voir [Cordeau et al., 1997, Cordeau and Laporte, 2001] par exemple. Concernant les problèmes avec collecte et livraisons (PDP et PDP-TW), on peut citer les travaux de Nanry et Barnes [Nanry and Barnes, 2000], Caricato et al. [Caricato et al., 2003], et Codeau et Laporte [J.-F. and G., 2003].

Récemment des méthodes à voisinage variable (*Variable Neighborhood Search*) ont été introduites (voir [Mladenović and Hansen, 1997, Hansen et al., 2010]). L'algorithme comporte deux étapes. Tout d'abord l'application d'une recherche locale permettant de trouver un optimum local. Puis, une fois qu'aucune solution voisine ne permet d'améliorer la qualité de la solution courante, une perturbation est appliquée afin de faire un saut conséquent dans l'espace des solutions. Il existe deux principales heuristiques de descente pour le *VNS* : choisir la solution voisine améliorant le plus la solution courante parmi tous les voisins (*Best Improvement*), ou choisir la première solution voisine améliorant la solution courante (*First Improvement*).

Dans [Polacek et al., 2004], Polacek et al. déclarent être les premiers à appliquer le *VNS* à un VRP. Ils ont montré que l'algorithme était compétitif vis-à-vis de la recherche tabou à la fois

en terme de qualité de solution et de vitesse d'exécution. Néanmoins, Bräysy a développé dans le même temps un même algorithme (voir [Bräysy, 2003]) et sera le premier publier ses résultats. En 2009, Fleszar et al. ont appliqué un *VNS* à l'*Open Vehicle Routing Problem* (OVRP). Dans ce problème les véhicules ne sont pas obligé de revenir au dépôt à la fin de leur tournée. L'algorithme consiste à minimiser d'abord le nombre de véhicules utilisés puis à minimiser la distance parcourue par ces véhicules (voir [Fleszar et al., 2009]).

Dernièrement Kytöjoki et al. ont utilisé un *VNS* appliquée à des instances de grande taille comportant jusqu'à 20000 clients (voir [Kytöjoki et al., 2007]). Leurs résultats montrent qu'une telle méthode est applicable à des problèmes rencontrés dans le monde réel.

La méthode Greedy Randomized Adaptive Search Procedure (GRASP) a été introduite par Feo et Resende en 1989 (voir [Feo and Resende, 1989]). Le principe est de construire une solution en utilisant à la fois une méthode gloutone et une méthode aléatoire. Ainsi, à chaque étape de la construction d'une solution, les éléments pouvant être insérés dans la solution construite sont placés dans une liste qui est triée en fonction d'une fonction objectif. Dans un second temps, l'élément inséré à la solution est tiré au sort parmi les meilleurs éléments présents dans cette liste triée. Le processus est répété jusqu'à ce qu'une solution complète soit construite.

Dans [Kontoravdis and Bard, 1995], Kontoravdis et Bard ont appliqué la métaheuristique GRASP à l'*Open Vehicle Routing Problem with Time-Windows* (OVRP-TW). Afin de tester leur méthode, les auteurs l'ont appliqués avec succès à différentes variantes du VRP-TW comme le *Pickup and Delivery Problem with Time Windows* (PDP-TW).

### Méthodes à population :

Les métaheuristiques à population fonctionnent grâce à des mécanismes d'intelligence collective. Les principales métaheuristiques à population utilisées dans la résolution du VRP sont les algorithmes génétiques et les algorithmes fourmis.

Les premiers algorithmes génétiques développés ont cherché à résoudre les versions du problème utilisant des fenêtres de temps. La très grande majorité des algorithmes génétiques utilisés pour résoudre le VRP sont dits "hybrides" car ils utilisent un opérateur de mutation. C'est cet opérateur qui utilise des heuristiques de recherche locale afin d'optimiser la solution. Par exemple, l'opérateur de mutation le plus simple consiste à inverser deux gènes dans le chromosome et applique ainsi la méthode 2 – *opt*. Les premiers travaux mettant en exergue

les algorithmes génétiques dans la résolution des VRP-TW datent des années 1990. En 1991, Thangiah a présenté dans sa thèse (voir [Thangiah, 1991]) GIDEON, un algorithme génétique appliqué au VRP-TW capable de battre 41 des 56 instances classiques sur lesquelles il a été testé. En 1993, Blanton et Wainwright ont présenté dans [Blanton and Wainwright, 1993] deux opérateurs de croisement adapté au problème de tournées de véhicules avec fenêtres de temps. Le premier opérateur, appelé *MX1*, consiste à parcourir les gènes des deux chromosomes parents *A* et *B* et de déterminer quel est le gène apparaissant en premier dans le vecteur des contraintes de précédence. Ce gène sera placé dans le chromosome fils *AB*. Dans le cas où le gène  $x_i$  sélectionné soit celui de *A* ( $x_i^A$ ), le gène ayant la même valeur  $x_j^B$  (avec  $i \neq j$ ) est inversé dans *B* avec  $x_i^B$  afin de conserver la validité de la solution représentée. Puis le gène suivant des parents sera examiné, etc. Le second opérateur, appelé *MX2*, consiste à comparer deux gènes  $x_i$  et  $x_j$  dans *A* et *B* en commençant à  $i = j = 1$ , puis en plaçant le gène apparaissant en premier dans le vecteur de précédence dans *AB* à l'indice *k*. Si c'est  $x_i^A$  qui est sélectionné, alors le gène de *B* ayant la valeur de  $x_i^A$  sera supprimé et *i* et *k* seront incrémentés pour pouvoir passer à l'indice suivant. En 1996, Potvin et Bengio ont présenté dans [Potvin and Bengio, 1996] l'algorithme GENE-ROUS (GENetic ROUting System) qui utilise deux opérateurs de croisement afin de générer les générations d'individus. Le premier est appelé *Sequence-Based Crossover* (SBX) et croisent deux parties des parents pour obtenir un nouvel individu. Une procédure de réparation est ensuite appliquée pour réparer les chromosomes ne comportant pas une et une seule fois chaque client à servir. Le second opérateur s'appelle *Route-Based Crossover* (RBX) et remplace une partie des gènes d'un individu par ceux d'un autre individu pour créer un nouveau chromosome. La procédure de réparation évoquée précédemment permet de rétablir la validité d'un individu. En 1998, Berger et al. ont proposé dans [Berger et al., 1998] un algorithme génétique hybride utilisant une méthode de recherche locale. Les performances de cet algorithme sont supérieures aux résultats des algorithmes génétiques publiés jusqu'alors. En 1999, Gehring et Homberger ont proposés dans [Gehring and Homberger, 1999, Homberger et al., 1999] (puis d'autres versions en 2001 dans [Gehring and Homberger, 2002] et en 2005 [Homberger and Gehring, 2005]), un algorithme génétique à deux phases (ainsi que sa version parallèle) consistant à d'abord minimiser le nombre de véhicules utilisés puis, dans une seconde phase à minimiser la distance parcourue.

En 2001, Tan et al. ont utilisé une représentation des chromosomes sous forme de chaînes de caractères (contrairement à la représentation binaire utilisée par Thangiah dans [Thangiah, 1991]). En 2004, Berger et Barkaoui ont présenté dans [Berger and Barkaoui, 2004] un algorithme

génétique pour le VRP-TW faisant évoluer deux populations distinctes : l'une cherche à minimiser la distance totale parcourue alors que la seconde essaye de minimiser le nombre de contraintes temporelles non respectées. Là aussi, les performances dépassent ou égalent les meilleurs résultats des algorithmes de la littérature. La version parallèle de l'algorithme permet un facteur de *speed-up* de 5. En 2006, Ombuki et al. ainsi que Tan et al., ont développé respectivement dans [Ombuki et al., 2006] et [Tan et al., 2006], un algorithme génétique multi-objectifs pour le VRP-TW cherchant les solutions Pareto dominantes au problème.

Concernant les problèmes de tournées de véhicule sans contraintes de temps, les premiers algorithmes génétiques datent des années 1990. Dans [Chu and Beasley, 1997], Chu et Baseley ont utilisé une représentation pour le *Generalized Assignment Problem* pouvant être appliquée au VRP. Dans cette formulation, les indices des véhicules sont inscrits dans les gènes. Ainsi, un chromosome sera composé de  $n$  gènes (pour  $n$  clients) et chaque gène aura pour valeur un nombre compris entre 1 et  $m$ . Toutefois, même si cette notation a l'avantage de résoudre le sous-problème de *bin-packing*, elle ne permet pas de connaître l'ordre des visites dans la tournée de chaque véhicule. Malgré les travaux de Chu en 1997, les premiers résultats concernant l'application d'algorithmes génétiques au VRP sans contraintes de temps ne sont apparus qu'à partir de 2003. En effet, dans [Baker and Aye chew, 2003], Baker et Aye chew utilise la représentation de Chu pour les chromosomes et déterminent l'ordre de visite des clients en résolvant  $m$  problèmes de voyageurs de commerce. Leur algorithme n'atteint pas les performances des algorithmes de recherche tabou (notamment le TABROUTE de Gendreau et al) mais permet d'obtenir des solutions de bonne qualité en un temps raisonnable. Berger et Barkaoui ont présenté également dans [Berger and Barkaoui, 2003], une méthode de résolution du CVRP utilisant un algorithme génétique à deux populations. Leur algorithme baptisé HGA-VRP (*Hybrid Genetic Algorithm VRP*), consiste à faire migrer des individus entre les deux populations afin de diversifier la recherche de solutions. Les performances sont comparables à celles obtenues par les méthodes de recherche tabou.

En 2003, Jaszkiewicz et Kominek ont utilisé dans [Jaszkiewicz and Kominek, 2003] une méthode de recherche locale génétique (*Genetic Local Search*) appelée également algorithme mémétique (*memetic algorithm*) qui est en fait une hybridation entre un algorithme génétique et une méthode de recherche locale. Leurs résultats montrent que leur approche hybride obtient de meilleurs résultats que les approches génétiques et la recherche locale utilisées séparément. Une approche similaire a été proposé également par Kubiak dans [Kubiak, 2004].

En 2004, Prins a proposé dans [Prins, 2004], un algorithme génétique hybride pour le DVRP plus performant que les méthodes tabou sur la plupart des 14 instances données par Christofides et meilleur sur 20 instances de grande taille formulées par Golden et al. La méthode consiste à utiliser des chromosomes représentant la liste ordonnée des clients à visiter, sans délimiter les tournées, puis, dans un second temps, de découper le chromosome afin d'obtenir les tournées. Cette approche est de type *route first - cluster second*.

En 2006, Alba et Dorronsoro ont introduit dans [Alba and Dorronsoro, 2006] un algorithme génétique cellulaire appelé *JCell2oli* pour le CVRP. La différence avec un AG classique réside dans la notion de voisinage entre individus. Dans la plupart des AG classiques, lors de la phase de sélection, des individus sont sélectionnés parmi la population globale de façon pseudo-aléatoire puis seule une partie de cette sous-population est conservée. La population de chromosomes de l'AGc d'Alba et Dorronsoro est organisée sous forme d'un tore 2D et utilise un voisinage de Von Neumann. Ainsi lors de la phase de sélection, un individu entre en compétition avec ses 4 voisins (Nord, Est, Sud et Ouest). Cet algorithme est capable d'approcher les résultats des meilleures méthodes de résolution du CVRP.

En 2007, Mester et al. ont proposé dans [Mester et al., 2007] un algorithme génétique hybride utilisant une population à un seul chromosome. La génération du chromosome de la génération suivante est assurée par la mutation du chromosome de la génération courante. L'algorithme permet d'égaler et même d'améliorer 42% des meilleurs résultats de 199 jeux de tests de la littérature. Dans [Mester and Bräsysy, 2007], Mester et Bräsysy ont combinés 3 différentes stratégies de sélection de clients à insérer dans une tournée. La première stratégie consiste à les insérer par proximité avec le véhicule. La seconde consiste à choisir aléatoirement ( $0.2 + 0.5a$ ) clients ( $a$  est un nombre aléatoire uniformément distribué sur  $[0; 1]$ ) puis à prendre les autres en fonction de leur proximité. Enfin, la dernière stratégie consiste à déterminer un rayon de façon aléatoire afin de construire 2 cercles centrés sur le dépôt puis de choisir un nombre aléatoire de clients par proximité par rapport aux cercles. Une des trois stratégies est choisie à chaque itération de l'algorithme afin de faire varier l'exploration de l'espace des solutions. L'algorithme a été testé sur 76 instances de la littérature et a ainsi été capable de donner 70 des 76 meilleurs résultats.

Concernant les problèmes avec collectes et livraisons, les algorithmes génétiques ont été également largement utilisés. Parmi les principaux travaux, on peut citer ceux de Pankratz (voir [Pankratz, 2005]) ainsi que ceux de Ganesh et Narendran (voir [Ganesh and Narendran, 2007]).

Les premiers algorithmes fourmis dédiés aux problèmes de tournées de véhicules datent de la fin des années 1990. Dans [Bullnheimer et al., 1997a, Bullnheimer et al., 1997b], Bullnheimer et al. ont développé un algorithme fourmis basé sur le *Ant System* de Dorigo (voir [Dorigo, 1992]) pour le CVRP avec contrainte de distance. Chaque fourmis construit une tournée puis retourne au dépôt lorsque la capacité du véhicule ou la distance maximale est dépassée. Les résultats montrent que l'algorithme hybride avec une optimisation locale de type  $2 - opt$  permet d'approcher les résultats obtenus par les algorithmes tabou mais toutefois sans les améliorer.

En 1999, Gambardella et al. ont proposé dans [Gambardella et al., 1999] l'algorithme *MACS* pour le VRPTW. L'algorithme utilise deux colonies de fourmis. La première cherche à minimiser le nombre de véhicules utilisés alors que la seconde cherche à minimiser la distance totale parcourue par les véhicules.

En 2000, Doerner et al. ont appliqué la métahéuristique ACO au problème avec collecte et livraison avec fenêtres de temps (PDP-TW). Ils proposent ainsi dans [Doerner et al., 2000] un algorithme fourmis capable de résoudre le *Pickup and Delivery Problem with Time Windows* et montrent qu'il est nécessaire d'adapter l'implémentation de l'algorithme en fonction du problème afin d'obtenir de bons résultats.

Dans [Doerner et al., 2001], Doerner et al. décrivent leur méthode nommée COMPETAnts. Elle consiste à utiliser deux populations de fourmis chacune avec leur propre objectif et de pondérer la priorité d'une population vis-à-vis de l'autre en fonction des besoins du problème. Certaines fourmis appelées "espionnes" (*spies*) utilisent la phéromone de l'autre colonie en plus de la trace de phéromone de leur propre colonie alors que les fourmis traditionnelles n'utilisent que leur propres traces de phéromone. Doerner et al. ont appliqué cet algorithme à un PDP-TW et ont montré que l'utilisation de deux colonies avec le système de communication inter-colonies par mécanismes d'espionnage permet d'obtenir de meilleurs résultats qu'avec une métahéuristique fourmis classique.

Dans [Reimann et al., 2004], Reimann et al. utilisent l'heuristique de Clarke et Wright (voir [Clarke and Wright, 1964]) afin de définir le comportement des fourmis. Une version parallèle de l'algorithme a été proposée par Doerner et al. dans [Doerner et al., 2005]. Bell et McMullen ont indiqué dans [Bell and McMullen, 2004] que les algorithmes fourmis étaient capables de donner des résultats approchant les solutions optimales (à 1% près) à des problèmes de tournées de véhicules. Ils ont également montré que l'utilisation d'une colonie différente pour chaque véhicule utilisant sa propre phéromone permettait d'améliorer les résultats, spécialement pour les instances les plus grandes. Mazzeo et Loiseau ont proposé dans [Mazzeo and Loiseau, 2004],

un algorithme fourmis pour le CVRP. Leur algorithme procède à la construction des tournées par une fourmis qui rentre au dépôt lorsque la capacité du véhicule de la tournée est atteinte. En 2006, Li et Tian ont développé dans [Li and Tian, 2006] un algorithme fourmis similaire pour l'*Open Vehicle Routing Problem*. Enfin, Chen et Ting [Chia-Ho and Ching-Jung, 2005] ont proposé un algorithme fourmis hybride avec une méthode de recuit simulé afin de résoudre le VRPTW. Le principe est d'appliquer la méthode du recuit simulé sur les solutions fournies par l'algorithme fourmis. En 2008, Gajpal et Abad ont appliqué dans [Gajpal and Abad, 2008] un algorithme *MACS* au VRPB. Puis, en 2009 les auteurs ont proposé un algorithme basé sur l'*ACS* (voir [Dorigo and Gambardella, 1997]) à un problème de tournées de véhicules avec collectes et livraisons simultanées (voir [Gajpal and Abad, 2009]).

Plus d'informations concernant les métaheuristiques appliquées aux différents problèmes de tournées de véhicules peuvent être trouvées dans l'excellente revue de littérature de Gendreau et al (voir [Gendreau et al., 2008]). Plus récemment, Vidal et al. ont proposé un tour d'horizon des heuristiques (au sens large) appliquées au VRP (voir [Vidal et al., 2011]). Les métaheuristiques ont permis récemment d'obtenir des solutions d'excellente qualité à des instances du problème de taille importante. Même si les meilleurs résultats ont été obtenus grâce à des algorithmes basés sur la méthode de recherche tabou, d'autres métaheuristiques comme les algorithmes génétiques et les colonies de fourmis artificielles fournissent des solutions intéressantes et ont l'avantage inestimable d'être adaptés à l'utilisation au sein d'un environnement dynamique.

Concernant la version hétérogène du CVRP, un tour d'horizon détaillé des méthodes de résolution peut-être trouvé dans [Baldacci et al., 2008a] et [Berbeglia et al., 2007].

Le problème a donc été largement étudié au cours des 40 dernières années. Néanmoins, beaucoup de méthodes de résolution évoquées précédemment ne peuvent pas s'appliquer à certains problèmes réels. En effet, lorsque les requêtes des clients ne sont connues qu'au cours des tournées, le problème devient dynamique et nécessite une résolution adaptée.

### II.3.6 Problème dynamique

Lorsque certaines informations portant sur les ordres de livraisons, ou sur les caractéristiques du problème, ne sont connues qu'une fois que les tournées des véhicules ont débuté, le problème de tournées de véhicule est dit “dynamique” (*Dynamic Vehicle Routing Problem* : DVRP). Lorsque l'évolution du système comporte une part de prédictibilité, le problème est dit “stochastique” (voir [Gendreau et al., 1996]) et est appelé *Stochastic Vehicle Routing Problem* :

SVRP, *Vehicle Routing Problem with Stochastic Demands* : VRPSD ou encore *Probabilistic Vehicle Routing Problem* : PVRP. Dans ces problèmes il est possible de prévoir l'évolution des demandes de façon probabiliste afin d'adapter les tournées des véhicules à de probables modifications futures. Dans le PVRP les clients ont une probabilité de demander à être visité par un véhicule alors que dans le SVRP la probabilité peut porter sur la demande de visite, la quantité à livrer ainsi que les temps de parcours.

Ainsi, de nombreux problèmes réels sont dynamique. En effet, les itinéraires des véhicules sont désormais calculés dynamiquement en tenant compte du trafic sur les routes par exemple. Plusieurs exemples types de VRP se révèlent dynamique dans la vie réelle. Ainsi, lorsqu'un client appelle en urgence la compagnie de fuel l'hiver, la compagnie prend en compte sa demande en optimisant les tournées courantes de ses camions de livraison afin d'essayer d'intégrer cette nouvelle demande. Dans le cas de la compagnie de taxis, un client peut appeler pour annuler son taxi à la dernière minute, alors que le taxi en question est déjà en route vers l'emplacement du client. La compagnie va alors essayer de rediriger le taxi vers une autre demande afin de minimiser le coût induit par cette annulation.

Ce qui était impossible dans le passé devient réalisable grâce au développement de la technologie et notamment grâce aux téléphones portables (GPRS, 3G) et au système de positionnement par satellite (GPS). Les compagnies sont capable de rediriger leurs véhicules à n'importe quel moment. C'est grâce à ce développement technologique que le problème dynamique de tournées de véhicules est devenu au fil du temps de plus en plus présent dans la littérature scientifique.

Dans [Psaraftis, 1988], Psaraftis identifie 12 points de différence entre le problème statique et le problème dynamique de tournées de véhicules. Il indique que dans le problème dynamique :

1. La dimension temporelle est primordiale ;
2. L'horizon peut être non défini ;
3. L'information sur le futur peut-être imprécise ou inconnue ;
4. Les événements proches de leur terme sont plus importants ;
5. Les mécanismes de mise-à-jour de l'information sont essentiels ;
6. Les décisions de réaffectation, ainsi que de réorganisation peuvent être nécessaires pour garantir la validité de la solution ;
7. Des temps de calculs plus courts sont nécessaire ;
8. La prise en compte des reports infinis est essentielle ;
9. La fonction objectif peut être différente ;

10. Les contraintes de temps peuvent être différentes ;
11. Le degré de flexibilité en terme de variation de la taille de la flotte de véhicules est moindre ;
12. La gestion des files d'attente de demandes peut devenir un facteur important.

La dynamique pousse donc le système à devoir s'adapter suffisamment rapidement afin de limiter l'impact des perturbations. Les méthodes de résolutions doivent donc permettre d'obtenir une solution satisfaisante rapidement.

#### **II.3.6.1 Méthodes de résolution du problème dynamique**

Il n'est pas question ici d'appliquer des méthodes de résolution exactes, à cause du temps de calcul nécessaire pour ces méthodes. Les méthodes heuristiques permettent d'insérer les nouvelles demandes rapidement et obtiennent des résultats satisfaisant. Toutefois, les meilleurs résultats sont obtenus par des métaheuristiques permettant d'allier qualité des solutions et rapidité de calculs.

Le premier algorithme exact à un problème dynamique, en l'occurrence au DARP, a été proposé en 1980 par Psaraftis (voir [Psaraftis, 1980]). Basé sur la programmation dynamique, le programme calcul un nouveau plan de charge pour les véhicules à chaque arrivée de nouvelle demande dans le système. Plus récemment, Chen et Xu ont proposés dans [Chen and Xu, 2006] un algorithme de génération dynamique de colonnes (*DYCOL : DYnamic COLumn*) appliqué au DVRP-TW. Leur méthode permet d'obtenir des solutions exactes plus rapidement qu'avec d'autres méthodes déterministes.

Concernant les heuristiques, les premiers algorithmes d'insertion des nouvelles demandes dans des tournées déjà planifiées ont été développés par Wilson et al. dans les années 1970 dans le cadre du problème de transport à la demande (voir [Wilson and of Technology. Urban Systems Laboratory, 1975 Wilson et al., 1975, Wilson et al., 1977]). Dans le DARP, les horizons de calcul sont très court et la fréquence des demandes est élevée. Ce problème est donc intrinsèquement dynamique. En 1985, Psaraftis a défini l'algorithme *MORSS* (MIT Ocean Routing and Scheduling System) dans [Psaraftis and of Management, 1985] permettant de résoudre un problème dynamique de routage de cargo en situation d'urgence. Madsen et al. ont transformé des procédures d'insertion utilisées dans les problèmes statiques pour les utiliser dans les problèmes dynamique (voir [Madsen et al., 1995]). De même, dans [Swihart and Papastavrou, 1999], Swihart et Papastavrou ont testé plusieurs heuristiques classiques appliquées à un problème dynamique de

collectes et livraisons (*Dynamic Pickup and Delivery Problem* : DPDP). Ainsi, l'heuristique du plus proche voisin consiste dans ce cas à procéder à la collecte du client le plus proche de la position courante du véhicule. Cette méthode donne d'excellents résultats dans plusieurs cas.

En 1998, Savelsbergh et Sol ont proposé dans [Savelsbergh and Sol, 1998] une méthode de *Branch-and-Price* pour résoudre un problème de tournées de véhicules appliquée à une entreprise de transport dans le Benelux.

Plus récemment, les travaux de Yang et al. (voir [Yang et al., 2004]) concernent des problèmes dynamiques avec collectes et livraisons. Le problème statique est d'abord résolu par un programme linéaire en nombre entier, puis les requêtes dynamiques sont insérées en fonction de politiques temps réel.

Concernant les métaheuristiques, les premiers travaux ont été proposés par Gendreau et al. en 1999 (voir [M. et al., 1999]). Ils ont développé un algorithme de recherche tabou utilisant un ensemble de solutions initiales maintenu en fonction de l'évolution dynamique des requêtes. Cet ensemble a été nommé "mémoire adaptative".

Plusieurs algorithmes génétiques ont été développés pour résoudre des problèmes dynamiques de tournées de véhicules, notamment celui de Benyahia et Potvin [Benyahia and Potvin, 1998] appliqué à un DPDP. Dans [Jih and Yung-Jen Hsu, 1999], Jih et Hsu ont développé un algorithme génétique hybride pour un PDP-TW capable de s'adapter à une version dynamique du problème. Dans [Pankratz, 2005], Pankratz propose également un algorithme génétique pour résoudre les sous instances du DPDP-TW. À chaque requête, un nouveau PDP-TW est résolu en utilisant la connaissance acquise lors du calcul des solutions précédentes.

Dans [Montemanni et al., 2005], Montemanni et al. proposent l'algorithme ACS-DVRP. La méthode est basée sur l'Ant Colony System de Dorigo et Gambardella (voir [Dorigo and Gambardella, 1997]) et est appliquée à un problème dynamique de tournées de véhicules. Le principe est de découper le temps en périodes appelées quartiers (slices) comme décrits dans [Kilby et al., 1998] dans lesquels toute nouvelle requête est différée à la fin de la période. Cette approche permet d'éviter de recalculer une solution à chaque événement. Chaque période de temps est vu comme un problème statique et est résolu par un *ACS* et l'information laissée par les fourmis (phéromone) permet de prendre en compte les calculs précédents lors de l'insertion des nouvelles requêtes. Leur méthode a été comparée à une méthode de type *GRASP* et a montré de bons résultats. Les auteurs ont également appliqué l'algorithme à un cas concret concernant la ville de Lugano en Suisse et comportant 50 clients et 10 véhicules. Les résultats sont également commentés par Rizzoli et al. dans [Rizzoli et al., 2007].

La version dynamique des problèmes de tournées de véhicules constitue un pan entier du domaine de recherche avec ses propres spécificités. Ainsi, plusieurs thèses de doctorats lui sont entièrement consacrées ([Larsen, 2000] pour le DVRP, et [Mitrovic-Minic, 2001] pour le PDPD-TW). Des tours d'horizon récents du problème peuvent être trouvés dans [Gendreau et al., 2008], [Larsen et al., 2008], [Berbeglia et al., 2010] et [Pillac et al., 1 10].

## Conclusion

Le problème de tournées de véhicules est une spécialisation du problème des multiples voyageurs de commerce. Dans sa forme la plus classique il consiste à optimiser les tournées d'une flotte de véhicules de façon à servir chaque client en respectant les capacités des véhicules (ainsi que bien souvent une distance maximale par véhicule) tout en minimisant la distance totale parcourue. Certaines versions du problèmes comportent des contraintes temporelles sous forme de fenêtres de temps. Dans sa version dynamique, le problème devient plus difficile à résoudre à cause du besoin d'obtenir une solution rapidement. L'incertitude liée à la dynamique impose l'utilisation de météuristiques permettant d'utiliser les solutions calculées précédemment pour calculer des nouvelles solutions lorsqu'un événement survient. Les méthodes les plus utilisées sont la recherche tabou, les algorithmes génétiques et les algorithmes fourmis.

## II.4 Problèmes d'ateliers

### Introduction

Dans cette section nous introduiront les problèmes d'ateliers. De nombreux problèmes de tournées de véhicules peuvent ainsi être modélisés par des problèmes d'ateliers et notamment par le problème de *Job Shop*. Nous passerons donc en revue les méthodes de résolution utilisées puis nous aborderont la version dynamique du problème en insistant sur les méthodes de résolution basées sur les algorithmes fourmis.

#### II.4.1 Définition et variantes du problème

Un ordonnancement consiste à affecter dans le temps des tâches à des ressources. Les problèmes d'ateliers consistent à produire un ordonnancement qui optimise la fabrication de produits au sein d'une usine de production en affectant des tâches à des machines. Dans ces problèmes,  $m$  machines  $M_j$  ( $\forall j \in [1; m]$ ) doivent effectuer  $n$  tâches (*Jobs*)  $J_i$  ( $\forall i \in [1; n]$ ).

L'objectif est de produire un ordonnancement, c'est-à-dire d'associer chaque tâche à une ou plusieurs périodes de temps sur une ou plusieurs machines, de façon à minimiser ou maximiser un ou plusieurs critères tout en respectant certaines contraintes. Chaque tâche  $J_i$  est divisée en  $n_i$  opérations  $O_{i1}, \dots, O_{in_i}$  et une date de disponibilité (*release date*)  $r_i$  correspondant à la date à partir de laquelle la première opération ( $O_{i1}$ ) de la tâche  $J_i$  peut être effectuée. D'autre part chaque opération  $O_{ij}$  est associée à une liste de machines  $\mu_{ij} \subseteq \{M_1, \dots, M_m\}$  sur lesquelles elle peut être réalisée. Dans le cas où  $\mu_{ij}$  ne comporte pas qu'une seule machine alors le problème est dit *Multi-Purpose Machines* (MPM). Le coût de la réalisation de la tâche  $J_i$  au temps  $t$  est définie par la fonction  $f_i(t)$ . Un ordonnancement est dit réalisable (ou valide) si et seulement si deux intervalles de temps ne se chevauchent pas sur une même machine, si deux intervalles de temps alloué à une même tâche ne se recoupent pas et si les contraintes spécifiques du problème sont respectées. L'optimisation consiste à fournir un ordonnancement minimal c'est-à-dire minimisant un ou plusieurs critères.

#### II.4.1.1 Notation de Graham et al.

Devant le nombre important de types de problèmes d'ateliers, Graham et al. ont introduit dans [Graham et al., 1979] une notation en trois parties :  $\{\alpha|\beta|\gamma\}$  permettant de déterminer et de classer les problèmes les uns par rapport aux autres. Dans cette notation, le champ  $\alpha$  correspond à l'environnement machine du problème,  $\beta$  aux caractéristiques des tâches, et  $\gamma$  indique le (ou les) critère(s) d'optimalité.

##### II.4.1.1.1 Environnement machine ( $\alpha$ )

L'environnement machine peut prendre plusieurs valeurs. Lorsque  $\alpha$  est vide, cela signifie que chaque tâche  $J_i$  doit être effectuée sur une unique machine déterminée. Si  $\alpha = P$  alors les machines sont parallèles et similaires.

Si  $\alpha = Q$ , alors les machines sont parallèles et uniformes c'est-à-dire que la durée d'exécution d'une tâche  $J_i$  sur la machine  $M_j$  dépendra de la vitesse (*speed*)  $s_j$  de  $M_j$ . Si  $\alpha = R$  alors les machines sont parallèles et indépendantes c'est-à-dire que la vitesse de la machine  $M_j$  dépend de la tâche à effectuer et que les vitesses d'exécution de la tâche  $J_i$  dépend également de la machine sur laquelle elle sera effectuée.

Si  $\alpha \in \{G, X, O, J, F\}$  alors le problème est appelé *General Shop* ( $\alpha = G$ ). Ce problème général est caractérisé par le découpage des tâches en opérations. Lorsque  $\alpha = J$  le problème est appelé *Job Shop* et comporte une relation de précédence entre les opérations  $O_{i1}, \dots, O_{in_i}$

d'une tâche  $J_i$ . Ainsi, il n'est pas possible d'effectuer une opération  $O_{il}$  tant que  $O_{ik}$  (avec  $k < l$ ) n'a pas été réalisée. Dans le *Flow Shop* ( $\alpha = F$ ), au moins une opération de chaque tâche doit être effectuée par chaque machine et une machine ne peut traiter qu'une seule tâche à la fois. L'*Open Shop* ( $\alpha = O$ ) reprend les contraintes du *Flow Shop* sans la relation de précédence entre les opérations d'une tâche. Ainsi au moins une tâche de chaque job doit être effectuée sur chacune des machines mais  $O_{il}$  peut être réalisée avant  $O_{ik}$  (avec  $k < l$ ,  $i = 1, \dots, n$ ).

#### II.4.1.1.2 Caractéristiques des tâches ( $\beta$ )

La classification de Graham et al. définit les caractéristiques des tâches grâce à 6 éléments  $\beta_1, \beta_2, \beta_3, \beta_4, \beta_5$  et  $\beta_6$ .

- $\beta_1 = pmtn$  si les tâches sont préemptibles c'est-à-dire si l'exécution des tâches peut être interrompue puis relancée même sur d'autres machines.
- $\beta_2$  décrit la relation de précédence entre les tâches et peut prendre les valeurs *prec*, *intree*, *outtree*, *tree*, *chains*, ou *sp-graph*.
- $\beta_3 = r_i$  si les tâches comportent des dates de disponibilité (*release date*).
- $\beta_4$  décrit la durée d'exécution des tâches ou le nombre d'opérations. Si les tâches ont toutes la même durée d'exécution on dit alors qu'elles ont une durée unitaire et  $\beta_4 = 1$ .
- $\beta_5 = d_i$  si les tâches doivent être réalisées avant une certaine date limite (*due date*).
- $\beta_6$  indique si les tâches doivent être regroupées afin d'être ordonnancées (*batching problem*). Dans ce cas  $\beta_6$  vaut *p-batch* ou *s-batch* selon si la longueur du groupe correspond à la durée maximale des tâches du groupe ou à la somme de ces durées.

D'autres caractéristiques peuvent également être prises en compte et notamment dans les problèmes où l'exécution d'une tâche  $J_i$  par une machine  $M_j$  requiert une durée de mise en place (*setup time*). En effet, dans certains de ces problèmes les durées de mise en place d'une tâche dépend de la tâche exécutée précédemment par la machine. Dans ce cas, on dit que le coût de mise en place est dépendant de la séquence (*sequence dependent setup times*) et  $\beta = ST_{SD}$ . Un tour d'horizon de ces problèmes a été rédigé par Allahverdi et al. (voir [Allahverdi et al., 1999]).

#### II.4.1.1.3 Critères d'optimalité ( $\gamma$ )

Graham et al. définissent deux principaux types de critères d'optimalité. D'une part les critères d'étranglement (*bottleneck objective*) correspondant minimiser la valeur maximale des fonctions de coût du problème et d'autre part, les critères de somme (*sum objective* visant à minimiser la somme des fonctions de coût du problème. Si la fonction de coût n'est pas spécifiée

dans la notation,  $\gamma$  vaudra  $f_{max}$  ou  $\sum f_i$ . La fonction d'évaluation peut concerner :

- la date de fin d'exécution de l'ensemble des tâches (*makespan*) et dans ce cas  $\gamma = C_{max}$
- la durée totale de traitement (*total flow time*) où  $\gamma = \sum C_i$
- la durée totale de traitement pondérée (*weighted total flow time* :  $\gamma = \sum w_i C_i$ )
- le retard algébrique (*lateness*) définit comme la différence entre la date limite de la tâche et sa date de fin d'exécution :  $L_i = C_i - d_i$ . Dans ce cas  $\gamma = \sum L_i$  ou  $\gamma = \max L_i$
- le retard (*tardiness*) définit comme le max entre le retard algébrique et 0 ( $T_i = \max\{0, C_i - d_i\}$ ) :  $\gamma = \sum T_i$  ou  $\gamma = \max T_i$
- l'avance (*earliness*) définit ainsi :  $E_i = \max\{0, d_i - C_i\}$ . Dans ce cas  $\gamma = \sum E_i$  ou  $\gamma = \max E_i$
- le retard algébrique, le retard, ou l'avance pondérés :  $\gamma = \sum w_i L_i$ ,  $\gamma = \sum w_i T_i$ ,  $\gamma = \sum w_i E_i$ .

Tous ces critères sont présents dans la littérature mais il est important de remarquer que la plupart des problèmes rencontrés tentent de minimiser le *makespan*.

#### II.4.1.2 Lien avec les problèmes de tournées de véhicules

Le problème abordé dans cette thèse bien qu'assimilé de prime abord à un problème de tournées de véhicules peut être également modélisé en tant que problème d'atelier. En effet, les véhicules correspondent aux machines et les clients à visiter correspondent à des tâches. La durée de déplacement entre deux clients ou entre le dépôt et un client est assimilée à la durée de mise en place (*setup time*) de la tâche sur la machine. De fait, les durées de mise en place sont dépendant de la séquence (*sequence dependent setup times*). La durée nécessaire aux véhicules pour rentrer au dépôt en fin de tournée correspond au temps d'arrêt de la tâche sur la machine (*removal time*). Chacun de ces problèmes concernent l'optimisation d'une planification et d'une affectation de ressources à des tâches. Néanmoins, les méthodes de résolution peuvent être différente d'un problème à l'autre et c'est pourquoi il peut être intéressant d'essayer de transformer un problème de tournée de véhicule en problème d'atelier (et vice versa). Dans [Beck et al., 2003, Beck et al., 2006], Beck et al. montrent que cette transformation ne conduit pas forcément à l'obtention de meilleurs résultats. Les auteurs indiquent qu'il est nécessaire d'adapter la représentation du problème notamment en ajoutant certaines contraintes.

## II.4.2 Méthodes de résolution

Dans [Brucker, 2007], Brucker a dressé une revue des différents problèmes d'ordonnancement. Nous nous intéresserons ici qu'aux problèmes d'ateliers et notamment au problème de *Job Shop*. La NP-Complétude de ce problème a été démontrée par Lenstra et Rinnooy Kan dans [Lenstra and Kan, 1979] et la difficulté est telle que l'instance proposée par Muth et Thomson en 1963 dans [Muth and Thompson, 1963] comportant 10 machines et 10 tâches n'a été résolue de façon optimale qu'en 1989 (voir [Carlier and Pinson, 1989]). Dans [Jain and Meeran, 1998] et [Blazewicz et al., 1996], respectivement Jain et Meeran et Blazewicz et al. ont dressé un état de l'art des méthodes de résolution de ce problème.

### II.4.2.1 Résolution exacte

Concernant les méthodes déterministes, le premier algorithme efficace a été introduit par en 1954 par Johnson dans [Johnson, 1954]. Cet algorithme permet de résoudre un problème de *Flow Shop* à deux machines. Ce fut le premier algorithme visant à minimiser le *makespan*. D'autres travaux ont traité de problèmes comportant 2 machines (voir [Akers, 1956, Jackson, 1956, Hefetz and Adiri, 1982]), mais restent inapplicables pour des instances plus importantes.

### II.4.2.2 Résolution p-approchée

Lorsqu'il est impossible d'utiliser une méthode exacte, il est possible d'approcher le résultat avec une marge garantie par rapport à l'optimal. Ces algorithmes appelés "p-approchés" (*p-approximated*) consistent en effet à garantir une solution dont le score est supérieur à  $p\%$  du score optimal. Shmoys et al. ont proposé plusieurs algorithmes p-approchés par rapport au pire des cas (voir [Shmoys et al., 1994]) et en 1997 Williamson et al. ont apporté la preuve qu'il n'existe pas d'algorithme polynomial p-approché pour  $p < \frac{5}{4}$  (voir [Williamson et al., 1997]). Jain et Meeran indiquent dans leur tour d'horizon qu'il n'existe aucun algorithmes déterministe ou p-approchés efficaces pour des problèmes comportant au moins 3 machines et 3 tâches. French et al. ont d'ailleurs indiqué dans [French, 1982] qu'aucun algorithme efficace ne sera jamais développé pour la majorité des problèmes d'ordonnancement.

### II.4.2.3 Résolution mathématique

Lorsqu'il n'est pas possible d'utiliser une méthode exacte ou p-approchée, la recherche s'est orientée vers des méthodes énumératives. Le principe est de générer les ordonnancement un par

un en utilisant des procédés d’élagage de l’espace des solutions. En effet, lorsqu’une solution construite est vérifiée comme sous optimale, l’exploration d’une partie de l’espace des solutions devient inutile. Ce procédé est repris par les méthodes de programmation linéaire en nombre entiers (*Mixed Integer Linear Programming*), de relaxation Lagrangienne (*Lagrangian Relaxation*) et de décomposition (*Decomposition Methods*). Toutefois, le nombre de variables générées afin de relâcher les contraintes augmente de façon exponentielle et rend ces procédés applicables uniquement sur de très petites instances. Jain et Meeran ajoutent que ces méthodes donnent de mauvais résultats à cause d’une trop grande déviation par rapport à l’optimum.

Des algorithmes de *Branch and Bound* ont été utilisés afin de résoudre des problèmes d’ateliers. Les deux principales stratégies de branchement sont la *General Active Schedules* : *GAS* (voir [Lageweg et al., 1977]) et la *Settling Essential Conflicts* : *SEC* (voir [Barker and McMahon, 1985]). *GAS* procède à l’énumération des solutions dans un ordre prédéfini alors que *SEC* utilise une heuristique visant à déterminer l’ordre de parcours des opérations deux à deux. Les premiers algorithmes de *Branch and Bound* appliqués aux problèmes d’ateliers ont été ceux de Balas en 1969 (voir [Balas, 1969]) et Florian et al. en 1971 dans [Florian et al., 1971]. Des améliorations successives ont été apportées par Calier et Pinson entre 1988 et 1994 sur le calcul de la borne inférieure de l’algorithme (voir [Pinson, 1988, Carlier and Pinson, 1989, Carlier and Pinson, 1990, Carlier and Pinson, 1994]). Toutes ces approches utilisent un graphe disjonctif afin de représenter les incompatibilités des tâches et des machines. Ces méthodes demandent un temps considérable pour obtenir une solution exacte à des problèmes de taille importante. Néanmoins ils permettent d’obtenir des solutions approchées assez rapidement.

#### II.4.2.4 Heuristiques

Pour obtenir encore plus rapidement des solutions approchées, des heuristiques ont été définies dont deux sont les plus largement répandues dans la littérature.

La première utilise des règles de priorités et ont été utilisées notamment par Baker dans [Baker, 1974], French dans [French, 1982], Morton et Pentico dans [Morton and Pentico, 1993]. Parmi les règles de priorités les plus utilisées on peut citer :

- *First Fit* : les tâches sont exécutées dans l’ordre d’arrivée (indice de la tâche) ;
- *Shortest Processing Time* : les tâches les plus courtes sont prioritaires ;
- *Longest Processing Time* : les tâches les plus longues à exécutées sont traitées en premier ;
- *Most Work Remaining* : les tâches comportant le plus d’opérations non traitées sont

prioritaires.

Une revue de littérature concernant ces règles de priorités peut être trouvée dans [Panwalkar and Iskander, 1986].

La deuxième heuristique est celle du goulot d'étranglement (*bottleneck*) et consiste à réaliser un ordonnancement au plus tôt sur chaque machine afin d'identifier les goulets d'étranglement c'est-à-dire les tâches retardant l'ensemble du processus. Une fois identifiée la tâche bloquante est déplacée dans le plan de charge des machines afin de minimiser le *makespan*. Ce mécanisme d'amélioration locale est répété tant qu'il existe des goulets d'étranglement. Cette procédure appelée *Shifting Bottleneck Procedure* : SBP a été introduite en 1988 dans [Adams et al., 1988] par Adams et al. D'autres procédures fonctionnant sur le même principe ont été proposées par Applegate et Cook (voir [Applegate and Cook, 1991]), Dauzère-Pérès et Lassere (voir [Dauzère-Pérès and Lasserre, 1991]), Balas et al. (voir [Balas et al., 1995]) et par Balas et Vazacopoulos (voir [Balas and Vazacopoulos, 1998]).

#### II.4.2.5 Métaheuristiques

Des métahéuristiques ont également été employées pour résoudre les problèmes d'ateliers comme les méthodes de recuit simulé, de recherche tabou, les algorithmes génétiques ainsi que les algorithmes à colonies de fourmis.

##### II.4.2.5.1 Recuit Simulé (*Simulated Annealing*)

La méthode du recuit a été utilisée pour résoudre le problème du *Job Shop* pour la première fois par Matsuo et al en 1989 (voir [Matsuo et al., 1989]). Ils ont appliqué la métahéuristique pour résoudre un problème à une machine avec comme critère d'optimalité la minimisation du retard pondéré (*weighted tardiness*). Parallèlement, Van Laarhoven et al. ont proposé un recuit simulé pour le problème général de *Job Shop* visant à minimiser le *makespan*. Le recuit simulé permet ainsi d'améliorer les solutions obtenues par les autres méthodes de résolution de l'époque pour des problèmes comportant jusqu'à 30 tâches et 20 machines. En revanche le temps d'exécution est supérieur au temps requis pour les méthodes heuristiques et notamment celle du goulot d'étranglement.

##### II.4.2.5.2 Recherche Tabou (*Tabu Search*)

Dans [Blazewicz et al., 1996], Blazewicz et al. indiquent que le critère d'optimisation dans une recherche locale est de minimiser la dégradation induite par l'affectation d'une tâche dans le plan de charge d'une machine. En 1993, dans [Dell'Amico and Trubian, 1993], Dell'Amico et

Trubian ont trouvé la solution optimale à un problème comportant 5 machines et 20 tâches en seulement deux minutes et trente secondes grâce à une procédure de recherche tabou. L'algorithme le plus connu est nommé *Parallel Tabu Search* de Taillard. Il a été proposé en 1994 dans [Taillard, 1994]. L'originalité de cet algorithme repose sur l'utilisation d'une mémoire de taille variable. Ainsi, la taille de la liste tabou est modifiée dynamiquement toutes les 15 itérations. Combiné à une parallélisation du processus, l'algorithme de Taillard permet d'obtenir de très bonnes solutions pour des instances même très grandes (jusqu'à 20 machines et 100 tâches). En 1995, Barnes et Chambers ont résolu un problème d'ateliers en utilisant la meilleure solution parmi les solutions fournies par 14 heuristiques différentes en tant que solution initiale à une procédure de recherche tabou (voir [Barnes and Chambers, 1995]). La même année, Sun et al. ont développé dans [Sun et al., 1995] une méthode de recherche tabou et ont montré qu'elle était plus performante que l'heuristique du goulot d'étranglement. En 1996, Song et Lee ont utilisé une méthode de recherche tabou pour résoudre un problème d'atelier périodique (*Periodic Job Shop Scheduling Problem* : PJSSP) dans lequel un ensemble de produits sont construits de façon répétée (voir [Song and Lee, 1996]). En 1998, Almeida et Centeno ont appliqué la recherche tabou au problème de Job Shop à 1 seule machine dont le critère d'optimalité est la minimisation de l'avance et du retard (earliness and tardiness) (voir [Almeida and Centeno, 1998]). Une approche similaire a été utilisée par James dans [James, 1997] afin de résoudre la version à plusieurs machines de ce problème. Cependant, selon Blazewicz et al., Les méthodes tabou les plus efficaces pour les problèmes de *Job Shop* sont celles proposées par Nowicki et Smutnicki (voir [Nowicki and Smutnicki, 1996]) et Balas et Vazacopoulos (voir [Balas and Vazacopoulos, 1998]). L'algorithme de Nowicki et Smutnicki a été proposé en 1993 puis publié en 1996 et consiste à utiliser non pas une seule meilleure solution mais une liste des meilleures solutions. Cet algorithme cumule une recherche tabou et un processus de *backtracking*. Ainsi lorsqu'une solution améliorant la meilleure solution courante est trouvée, elle est ajoutée en tête de liste. Lorsque plus aucune meilleure solution n'est trouvée par l'algorithme où que la condition d'arrêt a été rencontrée (temps d'exécution maximum, ou itération maximale), alors l'algorithme est relancé avec une ancienne meilleure solution jusqu'à ce que la liste des meilleures solutions soit vide. Cette méthode a permis d'obtenir la solution optimale à un problème comportant 5 machines et 20 tâches en seulement 3 secondes. L'algorithme de Balas et Vazacopoulos (proposé en 1995 puis publié en 1998) consiste à inverser plusieurs arcs disjonctifs sur un chemin critique à chaque itération. Leur méthode consiste à construire une énumération partielle des solutions sous forme d'arbre où chaque noeud correspond à un ordonnancement, et chaque arc représente l'inversion

de deux opérations sur un chemin critique. Les auteurs ont inclus cette méthode de recherche locale guidée utilisant un arbre de solutions à l'intérieur de la méthode du goulot d'étranglement. La recherche tabou est ainsi utilisée en tant que phase de post-optimisation de l'heuristique. Les résultats obtenus par les auteurs montrent que l'algorithme permet d'obtenir des solutions optimales ou proches de l'optimal de façon très rapide à diverses instances classiques de la littérature. Enfin, en 2001, Schmidt a proposé dans [Schmidt, 2001] une méthode tabou pour résoudre la version du problème où le coût de mise en place est dépendant de la séquence (*Job Shop Scheduling Problem with Sequence Dependent Setup Times* : JSSP-ST<sub>sd</sub>).

#### II.4.2.5.3 Algorithmes génétiques (*Genetic Algorithms*)

La première tentative de résolution d'un problème de *Job Shop* a été réalisée par Davis en 1985 (voir [Davis, 1985]). Depuis, de nombreux algorithmes génétiques ont été élaborés et adaptés aux problèmes de *Job Shop*. Ainsi on peut distinguer les approches utilisant une modélisation des chromosomes par des enchaînements de règles de priorité et celles utilisant la modélisation des chromosomes utilisée pour le problème de voyageur de commerce et qui consiste à définir des permutations de tâches. Concernant la première approche, elle a été utilisée notamment par Della Croce et al. dans [Croce et al., 1995]. Leur algorithme est appelé *Priority-rule based Genetic Algorithm* : P-GA et consiste à déterminer la meilleure séquence de règles de priorités à utiliser pour résoudre le problème. Cette approche est l'une des plus performantes. La seconde représentation a été notamment introduite par Bierwirth dans [Bierwirth, 1995] pour répondre au problème posé par la médiocre performance des opérateurs de croisements (voir [Dorndorf and Pesch, 1995] et [Bierwirth, 1995]). Tout comme pour le problème de voyageurs de commerce, les algorithmes génétiques les plus performants sont en réalité des méthodes hybrides combinant un environnement génétique et une recherche locale. En 1995, Dorndorf et Pesch ont introduit l'algorithme *Shifting Bottleneck Genetic Algorithm* : SB-GA qui comme son nom l'indique utilise l'heuristique du goulot d'étranglement au sein d'un algorithme génétique. Cette méthode donne des résultats comparables à ceux de Della Croce et al. et de façon plus rapide. En 1991, Nakano et Yamada (voir [Nakano and Yamada, 1991]) puis en 1994 Aarts et al. (voir [Aarts et al., 1994]) ont développé des algorithmes génétiques guidés par des méthodes de recherche locale (*Guided Local Search based Genetic Algorithm* : GLS-GA). Plus récemment, Gonçalves et al. ont présenté dans [Gonçalves et al., 2005] un algorithme génétique hybride combinant un algorithme similaire au *Priority-rule based Genetic Algorithm* de Della Croce et al. et une heuristique locale. L'algorithme se montre plus performant que la plupart des

méthodes de la littérature à l'exception de la méthode tabou de Nowicki et Smutnicki (voir [Nowicki and Smutnicki, 1996]).

#### II.4.2.5.4 Optimisation par colonies de fourmis (*Ant Colony Optimization*)

Le premier algorithme de colonies de fourmis utilisé pour résoudre un problème de Job Shop a été proposé par Colorni et al. en 1994 dans [Colorni et al., 1994]. L'algorithme est basé sur le *Ant System* de Dorigo et al. (voir [Dorigo, 1992]). Le principe est de minimiser le *makespan* d'un ordonnancement en modélisant les possibilités d'ordonnancement des opérations des tâches par un graphe complet. Chaque arc est pondéré par deux valeurs : d'une part la trace de phéromone (mémoire), et d'autre part une valeur représentant l'attractivité de l'utilisation de l'arc dans l'ordonnancement (heuristique). Les fourmis colonisent ainsi le graphe et chaque sommet visité est inséré dans une liste tabou. Lorsque la liste est pleine, la qualité de l'ordonnancement est évalué et de la phéromone est déposée sur les arcs utilisés en fonction de la qualité obtenue. Les auteurs ont montré que cette méthode permettait d'obtenir des solutions proches de l'optimal sans pour autant demander de modifications importantes afin de l'adapter spécifiquement au problème. Plus récemment, dans [Udomsakdigool and Kachitvichyanukul, 2011] Udomsakdigool et Kachitvichyanukul ont étendu l'utilisation d'un algorithme fourmis pour résoudre la version multi-critères du problème avec contraintes de temps (*Multi-criteria Job Shop Scheduling Problem with Time Windows*). Le but est ici de minimiser la combinaison linéaire pondérée de la durée de fin d'exécution (*makespan*), la durée moyenne de traitement (*mean flow time*), ainsi que du retard moyen (*mean tardiness*). Les fourmis sont ainsi séparées en 3 colonies utilisant chacune leur propre heuristique représentant chacun des objectifs. Les auteurs ont testé leur algorithme sur des instances comportant jusqu'à 15 tâches et 10 machines et ont montré que l'algorithme permet d'obtenir rapidement des solutions de bonne qualité.

Dans [S G Ponnambalam and Girish, 2005], Ponnambalam et al. ont introduit un algorithme fourmis pour résoudre des instances de grandes taille (20 tâches et 15 machines) du Flexible Job Shop Scheduling Problem (FJSSP). Dans ce problème introduit pour la première fois par Brücker et Schlie (voir [Brucker and Schlie, 1990]), toutes les machines sont capables de réaliser n'importe quelle opération. Le problème est NP-difficile pour des instances comportant plus de 2 machines. Dans [Xing et al., 2010], Xing et al. ont utilisé une méthode hybride d'optimisation pour le FJSSP utilisant à la fois des modèles à base de connaissance, et des modèles fourmis. L'algorithme mémorise les éléments intéressants des itérations précédentes pour guider l'algorithme fourmis vers les meilleures solutions.

Un tour d'horizon des algorithmes utilisés pour résoudre les problèmes de Job Shop avec des coûts de mise en place dépendant de la séquence (JSSP-ST<sub>sd</sub>) peut être trouvé dans [Allahverdi et al., 2008]. Même si les méthodes de résolution évoquées dans les paragraphes précédents sont destinés au problème statique du *Job Shop*, certaines techniques peuvent être utilisées lors de la résolution des versions dynamique du problème.

### **II.4.3 Problème dynamique**

Dans sa version dynamique le problème d'atelier (*Dynamic Job Shop Scheduling Problem* : DJSSP) prend en compte des événements (voir [Ramasesh, 1990]). Cela implique qu'un ordonnancement calculé peut devenir caduque au cours de l'exécution.

#### **II.4.3.1 Identification et politiques de gestion de la dynamique**

Différents événements peuvent se produire. Selon Ouelhadj et Petrovic, ces événements peuvent être liés soit aux ressources, soit aux tâches (voir [Ouelhadj and Petrovic, 2009]). En effet, dans les environnements réels de production, d'une part les machines peuvent tomber en panne ou les temps de mise en place peuvent être variables, et d'autre part les tâches peuvent arriver en retard ou en avance à l'atelier ou peuvent être annulées au dernier moment, etc.

Dans [Ouelhadj and Petrovic, 2009], Ouelhadj et Petrovic établissent un tour d'horizon concernant le DJSSP et indiquent qu'il existe 4 approches pour traiter ce problème :

- ordonnancement complètement réactif;
- ordonnancement prédictif et réactif;
- ordonnancement robuste prédictif et réactif;
- ordonnancement robuste pro-actif.

Dans [Suresh and Chaudhuri, 1993], Suresh et Chaundhuri regroupent ces approches en deux classes : l'ordonnancement réactif d'une part et l'ordonnancement prédictif d'autre part. Dans les systèmes réactifs, l'ordonnancement est calculé en fonction des événements qui surviennent. L'approche mixte (prédictive et réactive) est la plus rencontrée et consiste à calculer un ordonnancement statique avant le début de l'exécution en tenant compte des événements futurs puis de recalculer cet ordonnancement lorsqu'un événement survient pendant l'exécution. Dans les systèmes prédictifs, l'ordonnancement est déterminé de façon à faciliter la gestion des événements futurs potentiels. Chacune de ces approches nécessite des opérations de replanification. Selon Ouelhadj et Petrovic, il existe 3 politiques de replanification dans la littérature :

- la replanification périodique (*periodic rescheduling*);

- la replanification guidée par l'évenement (*event driven rescheduling*) ;
- la replanification mixte (*hybrid rescheduling*).

Les auteurs expliquent que la politique de replanification périodique peut être suffisante pour gérer la dynamique du problème à condition de déterminer de façon précise la taille de la période. Toutefois, les auteurs indiquent que dans chaque article étudié dans leur tour d'horizon, la politique de replanification guidée par l'événement donne de meilleurs résultats que la politique de replanification périodique.

Dans la politique de replanification guidée par l'événement il est question de prendre en compte les nouvelles caractéristiques du problème. Il est donc possible soit de recalculer totalement l'ordonnancement, soit de réparer l'ordonnancement à chaque fois qu'un événement le rend invalide. Cependant, la première méthode se révèle inutilisable en pratique à cause du délai trop court entre deux arrivées d'événements.

#### **II.4.3.2 Méthodes de résolution**

Tout comme pour la version dynamique du problème de tournées de véhicules, il n'est plus possible d'utiliser des méthodes exactes de résolution, qui même dans le cas statiques ne sont quasiment jamais utilisées dans les environnements réels de production. En effet, les délais de calcul d'une nouvelle solution deviendraient supérieurs à l'intervalle de temps entre deux événements. Plusieurs études ont été portées sur des méthodes heuristiques et métaheuristiques, ainsi que sur les techniques utilisées en intelligence artificielle et sur des approches multi-agents pour résoudre le DJSSP.

Dans [Gere, 1966], Gere a introduits des heuristiques à la fois pour le problème statique de *Job Shop* et pour le DJSSP. L'auteur a mesuré l'efficacité de chaque heuristique et a montré que le choix de l'heuristique la plus adaptée au problème est plus important dans la qualité de la solution que le choix de la règle de priorités à utiliser pour les tâches. Gere a conclu que les heuristiques basées sur l'anticipation amélioraient de façon significative la qualité de la solution sans augmenter de façon trop importante le temps de calcul de cette solution.

Les méthodes utilisées en intelligence artificielle sont également applicables au DJSSP comme les algorithmes à base de connaissance, les réseaux de neurones artificiels, les raisonnements à base de cas, la logique floue ou encore les réseaux de Petri.

De récentes approches utilisant des Systèmes Multi-Agents (SMA) ont été développées pour résoudre la version dynamique du problème d'atelier. Dans [Yoo and Müller, 2002], Yoo et Müller ont introduit un système multi-agents utilisant une méthode de recuit simulé pour op-

timiser les solutions proposées par les agents.

Les météahéuristiques constituent les principales méthodes de résolution du DJSSP. Les algorithmes génétiques sont les plus souvent rencontrés dans la littérature. Lin et al. ont proposé dans [Lin et al., 1997] un algorithme génétique pour le DJSSP et ont dans un premier temps comparé les performances de leur algorithme avec les résultats obtenus en utilisant des règles de priorités dans le cadre d'une replanification intégrale. Puis, dans un second temps, ils ont montré que l'utilisation d'une partie du génome de la génération précédente permet une replanification efficace lors de l'arrivée d'un événement. Dans [Qi et al., 2000], Qi et al. ont proposé un algorithme génétique parallèle à plusieurs populations pour résoudre le DJSSP multi-critère. Les auteurs ont implémenté leur algorithme sur la plateforme MATLAB et ont pu ainsi vérifier que leur technique obtenait des résultats de qualité proche de l'optimal tout en nécessitant peu de temps de calcul et en conservant une grande flexibilité face aux critères d'évaluation utilisés. Dans [Madureira et al., 2001], Madureira et al. présentent un algorithme génétique pour résoudre la version dynamique du problème d'ordonnancement à une machine (*Dynamic Single Machine Scheduling Problem* : DSMSP) ainsi qu'un algorithme génétique permettant de résoudre le DJSSP. Leur approche construit des prédictions de planification en fonction des éléments connus au moment du calcul.

Bien que répandus pour résoudre la version statique des problèmes de tournées de véhicules et du problème d'ateliers, les algorithmes d'optimisation par colonies de fourmis ne sont que très peu utilisés dans la résolution de la version dynamique du problème de *Job Shop*. Dans [Vogel et al., 2002], Vogel et al. ont proposé un algorithme fourmis fonctionnant en continu pour résoudre le *Real World Shop Floor Scheduling Problem*. Leur algorithme est comparé aux performances des heuristiques à base de règles de priorités et à un algorithme génétique. Les auteurs concluent que les résultats obtenus n'égalent pas pour le moment ceux de l'algorithme génétique mais que la piste est prometteuse et permettra d'ouvrir la voie aux travaux futurs. Plus récemment, Zhou et al. ont proposés dans [Zhou et al., 2008] un algorithme d'optimisation par colonies de fourmis pour résoudre le DJSSP. Les auteurs ont mesuré les performances de l'algorithme sur deux problèmes comportant les mêmes tâches mais avec des degrés de dynamité différents. Ils ont montré que l'algorithme fourmis était performant dans les deux cas et qu'augmenter le nombre de fourmis ou le nombre d'itérations maximum n'améliorait pas forcément la qualité de la solution fournie.

Les méthodes de résolution adaptées à la dynamique évoquées dans la première partie de ce chapitre ont donc été utilisées avec succès dans la résolution de la version dynamique du

problème de Job Shop. Néanmoins, la littérature reste mince concernant ce problème et des travaux supplémentaires seront nécessaires à la création et à la validation de méthodes efficaces et robustes.

## Conclusion

Dans cette partie nous avons passé en revue les caractéristiques des problèmes d'ateliers et plus précisément du problème de *Job Shop*. Tout comme pour les problèmes de voyageurs de commerce et de tournées de véhicules des méthodes de résolution exacte et approchées ont été utilisées. D'ailleurs, Beck et al. ont montré dans [Beck et al., 2003] que les problèmes de *Job Shop* et de tournées de véhicules étaient similaires. Néanmoins, tout comme pour les problèmes de tournées de véhicules, seules les méthodes approchées sont utilisées dans les problèmes concrets à cause de la complexité du problème et des délais relativement courts rencontrés dans les environnements de production réels. Ainsi les méthodes à base de règles de priorité, l'heuristique du goulot d'étranglement ainsi que les métaheuristiques du recuit simulé, de recherche tabou, les algorithmes génétiques et les algorithmes fourmis sont utilisés pour résoudre les instances de taille importante du JSSP. Concernant la version dynamique du problème, les algorithmes génétiques constituent la méthode la plus efficace permettant de proposer une nouvelle planification sans tout recalculer de zéro. Les algorithmes fournis restent néanmoins une voie à explorer pour résoudre la version dynamique des problèmes d'ateliers.

## Conclusions

Dans ce premier chapitre nous avons passé en revue tous les domaines, tous les principes et toutes les méthodes de résolution touchant à des degrés différents le problème d'optimisation traité dans cette thèse.

Dans la première section nous avons introduit la notion d'optimisation dynamique sous incertitude. Le cadre de l'incertitude dans cette thèse a été défini et nous nous intéresserons uniquement aux systèmes réactifs où la méthode de résolution elle-même sera adaptée à la dynamité et à l'incertitude du problème.

Dans la seconde section nous avons introduit le problème du voyageur de commerce ainsi que ses diverses variantes. Ce problème est à la base des autres problèmes d'optimisation évoqués dans cette thèse. Nous avons dressé un état de l'art des méthodes les plus couramment utilisées afin de résoudre le problème de voyageur de commerce de façon exacte ou ap-

prochée et en détaillant dans ce dernier cas les méthodes de construction, d'amélioration et les métaheuristiques. La version généralisée du problème à  $m$  voyageurs a également été étudiée et correspond au problème de tournées de véhicules dans lequel la capacité et la distance ne sont pas contraints. Nous avons montré qu'il existe deux paradigmes de résolution du problème : d'une part résoudre  $m$  problèmes de voyageur de commerce, et d'autre part résoudre le problème classique issu de la transformation du problème multiple. Là aussi plusieurs méthodes de résolutions existent. Concernant les méthodes exactes, des méthodes de *Branch and Price* et de *Branch and Bound* permettent de résoudre des instances de 120 villes pour 2 à 12 voyageurs. Les méthodes approchées permettent de résoudre des instances plus importantes grâce à des méta-heuristiques (recherche tabou, recuit simulé, réseaux de neurones artificiels, algorithmes génétiques et algorithmes fourmis).

Dans la troisième section nous avons présenté le problème de tournées de véhicules. Dans sa forme la plus classique il consiste à optimiser les tournées d'une flotte de véhicules de façon à servir chaque client en respectant les capacités des véhicules (ainsi que bien souvent une distance maximale par véhicule) tout en minimisant la distance totale parcourue. Certaines versions du problèmes comportent des contraintes temporelles sous forme de fenêtres de temps. Dans sa version dynamique, le problème devient plus difficile à résoudre à cause du besoin d'obtenir une solution rapidement. L'incertitude liée à la dynamique impose l'utilisation de métaheuristiques permettant d'utiliser les solutions calculées précédemment pour calculer des nouvelles solutions lorsqu'un événement survient. Les méthodes les plus utilisées sont la recherche tabou, les algorithmes génétiques et les algorithmes fourmis.

Enfin, dans la quatrième partie nous avons passé en revue les caractéristiques des problèmes d'ateliers et plus précisément du problème de *Job Shop*. Tout comme pour les problèmes de voyageurs de commerce et de tournées de véhicules des méthodes de résolution exacte et approchées ont été utilisées. D'ailleurs, Beck et al. ont montré dans [Beck et al., 2003] que les problèmes de *Job Shop* et de tournées de véhicules étaient similaires. Néanmoins, tout comme pour les problèmes de tournées de véhicules, seules les méthodes approchées sont utilisées dans les problèmes concrets à cause de la complexité du problème et des délais relativement courts rencontrés dans les environnements de production réels. Ainsi les méthodes à base de règles de priorité, l'heuristique du goulot d'étranglement ainsi que les métaheuristiques du recuit simulé, de recherche tabou, les algorithmes génétiques et les algorithmes fourmis sont utilisés pour résoudre les instances de taille importante du JSSP. Concernant la version dynamique du problème, les algorithmes génétiques constituent la méthode la plus efficace permettant de

proposer une nouvelle planification sans tout recalculer de zéro. Les algorithmes fourmis restent néanmoins une voie à explorer pour résoudre la version dynamique des problèmes d'ateliers et c'est pourquoi ils font l'objet de cette thèse.

## **Chapitre III**

# **Ordonnancement et affectation des missions des chariots cavaliers**

Comme définit dans le chapitre précédent, ce problème concerne deux problèmes dépendants : attribuer les missions aux chariots cavaliers et calculer les dates d'exécution de ces missions de façon à minimiser une fonction de coût. Nous détaillerons dans ce chapitre les caractéristiques du problème et notamment l'impact de la dynamique et de l'incertitude vis-à-vis de la résolution, puis nous proposerons deux modélisations possibles faisant le lien avec le contexte théorique détaillé dans le premier chapitre. Enfin nous décrirons les méthodes de résolution élaborées durant cette thèse et permettant de proposer des solutions de façon dynamique. Il sera question d'une méthode de résolution exacte d'une part, puis de plusieurs méthode de résolution approchée d'autre part. Plusieurs heuristiques seront ainsi présentées et deux méthodes utilisant la métaheuristique fourmi seront décrites.

### **III.1 Dynamique et incertitude**

#### **III.1.1 Gestion du temps**

Pour plus de simplicité nous choisissons de considérer le problème sur une période de temps fixée que nous appellerons journée. Il est à noter que dans la plupart des terminaux, l'exploitation ne s'arrête jamais. Cette utilisation continue est la raison pour laquelle une modélisation statique du problème est totalement inapplicable.

Néanmoins, une partie des missions à réaliser sont connues bien à l'avance et peuvent être ainsi plus facilement allouées aux chariots cavaliers. Au contraire d'autres missions ne sont

connues qu'une fois la journée d'exploitation débutée et doivent être ordonnancées en tenant compte des calculs précédents.

Dans le cadre du problème d'affectation des missions aux chariots cavaliers, la notion de degré effectif de dynamicité de Larsen (voir II.1.5) permettant de définir la difficulté induite par la dynamique d'un jeu de données devra être adaptée. En effet, la difficulté à ordonner une mission est associée à l'écart de temps entre la date de l'arrivée de la mission dans le système de planification et la date de début au plus tard de la mission, c'est-à-dire la date au delà de laquelle la fenêtre de temps ne pourra plus être respectée. Ainsi, plus une mission est connue en avance, plus elle est facile à ordonner. En revanche une mission qui est connue au dernier moment doit être immédiatement insérée dans le plan de charge d'un véhicule afin de pouvoir être accomplie dans les temps. Définissons ainsi  $\eta$  le nombre de missions à réaliser,  $t_i$  ( $1 \leq i \leq \eta$ ) la date d'arrivée de la  $i^{\text{ème}}$  mission dans le système et  $T_i$  sa date de début au plus tard. Le degré effectif de dynamicité peut donc être défini selon la formule suivante :

$$(III.1) \quad edod = \frac{\sum_{i=1}^{\eta} \left( \frac{t_i}{T_i} \right)}{\eta}$$

Selon cette formule, le degré effectif de dynamicité peut être supérieur à 1 si une ou plusieurs missions sont connues après leur date de début au plus tard. Dans ce cas, les contraintes de temps ne pourront pas être respectées mais la mission devra quand même être planifiée en cherchant à minimiser le retard.

### III.1.2 Incertitude

En plus de la notion de dynamicité, il existe de l'incertitude sur les données des missions. L'incertitude peut se situer à différents niveaux.

#### III.1.2.1 Incertitude liée aux clients du terminal

Le bassin du Terminal de Normandie est un bassin à marée, c'est-à-dire que la hauteur de l'eau dépend de la marée et donc que les fenêtres de temps d'arrivée et de départ des portes-conteneurs suivent un rythme d'un peu plus de 12 heures. Si un porte-conteneurs manque la marée, ne serait-ce que d'une heure, il devra attendre la marée suivante au large face à l'entrée du port. Dans ce cas, les missions de déchargement/chargement du navire prévues à l'avance seront annulées et devront être planifiées à nouveau afin de prendre en compte les nouveaux

horaires de présence du porte-conteneurs à quai. Ce décalage des missions aura des conséquences sur les autres missions du terminal et une planification globale devra être calculée. D'autre part, si le chargement du navire est retardé dans le terminal, son départ peut être décalé à la marée suivante. Dans ce cas le terminal devra verser des pénalités à l'armateur du navire.

Concernant les camions, les heures d'arrivées sont souvent connues à l'avance mais sont sujettes aux variations du trafic. Ainsi, au Havre les voies d'accès au port empruntent différents ponts-levant. Si les ponts sont abaissés, les camions seront à l'heure. En revanche un pont levé peut retarder les camions de plusieurs dizaines de minutes. D'autres camions ne sont parfois pas prévus dans le registre du terminal et doivent être tout de même servis. Le système d'affectation des missions doit être suffisamment flexible afin de permettre d'intégrer ces missions dans les plans de charge des chariots cavaliers pour répondre aux demandes dynamiques. Si un camion, présent à l'heure sur le terminal n'est pas servi dans les temps par un chariot cavalier, le terminal devra payer des frais de pénalité au transporteur.

Grâce à un réseau ferré développé et à un trafic maîtrisé, les trains sont moins sujets aux retards, ou du moins, ceux-ci sont annoncé à l'avance. En revanche, si les wagons ne sont pas chargés/déchargés à temps, le terminal devra également payer des pénalités.

### **III.1.2.2 Incertitude liée aux chariots cavaliers**

La seconde source d'incertitude est liée aux chariots cavaliers à travers leurs pannes et leurs conducteurs.

#### **III.1.2.2.1 Panne des chariots cavaliers**

Comme détaillé dans le chapitre précédent, un chariot cavalier est un engin mécanique soumis à d'énorme contraintes physiques. Afin d'éviter des pannes, une maintenance régulière et approfondie est effectuée. Ainsi, la totalité de la flotte n'est que rarement disponible pour l'exploitation. Néanmoins, il arrive qu'un chariot cavalier tombe en panne et doive être réparé.

Une panne peut concerner soit la partie fille du chariot, c'est-à-dire le système de levage (*spreader*), soit la partie mère c'est-à-dire le chariot en lui-même (partie moteur, train roulant, système électrique, etc), soit les deux à la fois. En fonction du type de panne la réparation peut être réalisée sur place ou nécessitera un passage par le dépôt. Dans le cas d'une panne empêchant le déplacement autonome du chariot, un remorquage sera nécessaire. Toutes ces opérations de retour au dépôt, réparation, retour sur zone d'exploitation, provoquent un retard à la fois lié à l'indisponibilité du véhicule et à l'inaccessibilité de la travée sur laquelle le véhicule est tombé

en panne.

Si la panne concerne le système de levage, il existe deux cas de figure. D'une part, le conteneur à déplacer n'a pas pu être chargé sur le chariot cavalier et dans ce cas le chariot sera réparé et pendant ce temps la mission sera réaffectée à un autre véhicule. D'autre part, le conteneur a déjà été soulevé par le chariot cavalier et dans ce cas il faudra attendre la réparation du véhicule avant de pouvoir décharger le conteneur à nouveau.

### **III.1.2.2.2 Facteur humain**

Les chariots cavaliers sont pilotés par des opérateurs qui agissent selon leurs propres décisions. Lorsqu'un conducteur est disponible il demande au système de planification une liste de missions à effectuer. Le système lui fourni ainsi un plan de charge à respecter. Néanmoins, le conducteur peut décider de respecter ce plan ou non. Il indique au système chaque nouvelle mission débutée afin de mettre à jour la base de données et d'avertir les autres conducteurs de chariots cavaliers de son intention (ceci évite de mobiliser deux véhicules pour une même mission). Cependant, il arrive que le conducteur indique réaliser une mission alors qu'il en effectue une autre. De même, il peut oublier d'indiquer la réalisation d'une mission lorsque celle-ci est terminée. Le système considère alors dans le premier cas que le conteneur de la mission a été déplacé, et dans le second cas que le conteneur est toujours au même endroit. Tous ces comportements provoquent des incohérences dans le système qui conduisent bien souvent à des pertes de conteneurs. Des agents sont donc chargés de se déplacer à pieds dans les gigantesques travées du terminal afin de rechercher les conteneurs perdus.

D'autre part, lorsqu'un conducteur valide son intention de réaliser une mission, le système lui propose un itinéraire à suivre afin d'atteindre le point de collecte, puis le point de livraison du conteneur. Dans le cadre de la gestion du trafic sur le terminal, le système de routage peut tenir compte de la présence d'autres véhicules afin de minimiser les temps d'attente en entrée de travée. Lorsque le conducteur débute sa mission, les travées qu'il va emprunter sont ainsi réservées afin d'empêcher un autre chariot de les emprunter au même moment. De même, l'itinéraire est calculé afin de permettre au véhicule d'éviter les zones congestionnées et donc d'arriver à l'heure prévue par le système. Si le conducteur ne respecte pas cet itinéraire, il peut arriver en retard (ou parfois en avance) à l'entrée de la travée ce qui peut provoquer un engorgement du fait du non respect de la période de réservation de cette travée.

En plus de la complexité naturelle du problème, la dynamique et l'incertitude viennent

troubler les solutions calculées. Il est ainsi difficile d'envisager de le résoudre de façon exacte. La durée de validité de la solution ainsi calculée pourrait être inférieure à son temps de calcul. Une solution approchée, suffisamment performante, devra être calculée tout en tenant compte de la dynamique et de la méconnaissance de certaines parties du problème.

## III.2 Modélisations du problème

Le problème peut-être abordé sous plusieurs formes. La première consiste à le modéliser sous forme d'un problème de tournées de véhicules. C'est la représentation la plus naturelle d'un problème utilisant des véhicules. La seconde modélisation est de considérer un problème d'atelier.

### III.2.1 Problème de tournées de véhicules

Comme décrit précédemment, le problème de tournées de véhicules est une spécialisation du problème du voyageur de commerce où une série de lieux doivent être visités par des véhicules de la flotte. Dans ce problème, deux véhicules différents ne peuvent pas visiter une même ville et une ville ne peut-être visitée qu'une seule fois. L'objectif est de minimiser la distance totale parcourue tout en respectant les contraintes de capacité de chaque véhicule.

Concernant l'application aux chariots cavaliers et aux terminaux à conteneurs, le problème se caractérise dans le cas général par des véhicules à capacité unitaire. La non possibilité de transporter plusieurs conteneurs simplifie le problème qui se révèle appartenir au problème des multiples voyageurs de commerce à  $m$  voyageurs (où  $m$  est égal au nombre de véhicules).

La flotte de véhicules peut-être homogène ou hétérogène. Nous considérerons le cas le plus général, c'est-à-dire en conservant la possibilité de rencontrer des véhicules de différents types. Dans ce cas, il est indispensable de distinguer les véhicules les uns des autres. Définissons ainsi  $M$  l'ensemble des voyageurs de commerce (et donc par extension l'ensemble des véhicules) et le  $k^{\text{ème}}$  voyageur  $m_k$  ( $\forall k \in [1; m]$ ).

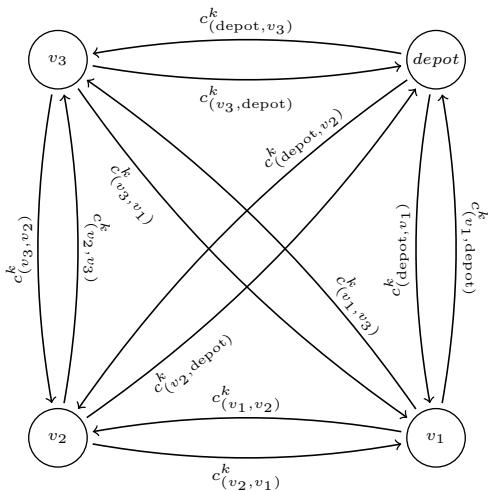
Dans le problème théorique les véhicules (ou les voyageurs de commerce) débutent tous leur tournée du dépôt et y retournent à la fin de leur tournée. Dans notre problème, les tournées peuvent être calculées en plein milieu d'une journée d'exploitation du terminal. Ainsi, les véhicules débutent leur tournée depuis l'emplacement où ils se trouvent s'ils sont inactifs, ou depuis l'emplacement où ils se trouveront à la fin de leur activité courante dans le cas contraire.

D'autre part, une mission correspond à une ville à visiter par l'un des voyageurs de commerce.

Soit  $N$  l'ensemble des villes à visiter et  $n_i$  la ville de la  $i^{\text{ème}}$  mission à effectuer ( $\forall i \in [1; n]$ ) plus  $n_0$  le dépôt. Chaque ville  $i$  doit être visitée en respectant deux fenêtres de temps (*Time Windows*)  $TW_{i,1}$  et  $TW_{i,2}$  respectivement de collecte et de livraison.  $TW_{i,1}^{\min}$  et  $TW_{i,1}^{\max}$  correspondent au minimum et au maximum de la fenêtre de temps  $TW_{i,1}$ .

### III.2.1.1 Structure du graphe

Ce problème est ainsi modélisé par un graphe complet où les noeuds sont les missions (villes) et où les arcs correspondent à l'enchaînement de deux missions. Définissons ainsi  $G^{(t)} = (V, A)$  le graphe complet où  $V$  est l'ensemble des villes (plus le dépôt des véhicules) et  $A$  est l'ensemble des arcs.



**Figure III.1:** Graphe d'un problème de tournées de véhicules de taille 3

La pondération des arcs doit donc représenter le coût de l'enchaînement de deux missions et le coût de passage d'une mission  $n_i$  à  $n_j$  est différent du coût de passage de  $n_j$  à  $n_i$ . Pour cette raison, le graphe est orienté. En effet, ici une mission est un déplacement entre une position de départ et un point de collecte puis entre ce point de collecte et un point de livraison. La distance entre le point de livraison du conteneur de la mission  $n_i$  et du point de collecte du conteneur de la mission  $n_j$  peut-être différente de celle entre le point de livraison de  $n_j$  et le point de collecte de  $n_i$ . Notons au passage que le problème correspond à la version asymétrique du problème des multiples voyageurs de commerce.

### III.2.1.2 Fonction d'évaluation et pondération

Dans le cadre d'un terminal portuaire à conteneurs, l'objectif est de minimiser à la fois les déplacements et le temps de dépassement des fenêtres de temps. La fonction d'évaluation est donc bi-critères. Respecter les fenêtres de temps signifie pour un chariot cavalier de ne pas atteindre un emplacement trop tard, c'est-à-dire après la fin de la fenêtre temporelle. Le cas inverse doit être également pris en compte. En effet, l'objectif pour le chariot cavalier est de minimiser ses déplacements mais aussi de minimiser son temps d'attente. Si un chariot arrive trop tôt sur un emplacement, il devra attendre jusqu'au début de la fenêtre temporelle.

Le retard et l'avance peuvent être considérés en tant que critère unique (retard algébrique ou *lateness*), cependant dans le problème étudié seul le retard engendre le paiement de pénalités. Le retard a une importance relative supérieure à celle de l'avance. La fonction de pondération devra donc être multi-critères et concernera les critères de distance, de retard (*tardiness*) et d'avance (*earliness*). Le retard est défini par l'équation III.4 et l'avance est décrite par l'équation III.5 (voir p. 109). Il est intéressant de noter que la prise en compte de l'avance dans les critères de la fonction objectif permet également d'améliorer la vitesse de convergence des algorithmes de résolution du problème. Ceci permet en effet d'éviter de parcourir des parties importantes de l'espace de recherche des solutions.

Le retard et l'avance sont deux critères correspondant à une durée, alors que le premier critère est une distance. Afin d'intégrer ces trois facteurs dans une même fonction il est nécessaire de les convertir afin d'utiliser une unité commune. Or, la vitesse des différents chariots cavalier étant connue, il est possible de transformer la distance à parcourir en durée de parcours. La pondération dépendra donc également du chariot cavalier empruntant l'arc car la flotte de véhicules est hétérogène. Toutes ces durées seront ainsi indiquées en secondes.

De plus, comme évoqué précédemment, la durée d'un parcours entre deux points du terminal ne dépend pas uniquement de la distance et de la vitesse du véhicule. En effet, les temps d'attente en entrée de travées viennent complexifier la mesure et doivent également être pris en compte. Ainsi  $\text{attente}_{ij}^k(t)$  correspond au temps d'attente du véhicule  $k$  sur le parcours reliant le point de livraison de la mission  $i$  au point de collecte de la mission  $j$  et  $\text{attente}_{jj}^k(t_P^k)$  correspond au temps d'attente de  $k$  entre le point de collecte et de livraison du conteneur de la mission  $j$ . Un système de routage utilisant un mécanisme de réservation d'itinéraire permettra de prévoir les points de ralentissement sur le réseau routier et ainsi de définir les temps d'attentes.

Cette mesure est ainsi dépendante du temps à laquelle le parcours à lieu. La fonction de pondération d'un arc doit donc également prendre en compte le temps, et ce temps dépend lui-même de la date de disponibilité de chariot cavalier. Cette date correspond soit à la date de fin de service de la mission précédente (*Completion Time* :  $C_i^k$ ), soit à la date de fin de réparation (*Repair Completion Time* :  $R^k$ ) si le véhicule est hors service, ou soit au temps courant ( $t^{now}$ ) si le véhicule est disponible et inutilisé.  $C_i^k$  est défini par récurrence en fonction du plan de charge du véhicule  $k$ . La date de fin de la mission  $i$  dépend en effet de la date de fin de la mission précédemment réalisée par le véhicule  $k$ . Soit  $C_{\text{pred}_i^k}^k$  la date de fin de la mission précédente réalisée par  $k$  juste avant de réaliser la mission  $i$ . La date de fin de la mission  $i$  réalisée par  $k$  s'écrit alors selon l'équation suivante :

$$(III.2) \quad C_i^k = \begin{cases} C_{\text{pred}_i^k}^k + \text{duree}_{\text{pred}_i^k, i}^k & \text{si } \text{pred}_i^k \neq \emptyset \\ \text{duree}_{0, i}^k & \text{sinon} \end{cases}$$

La fonction de distance est ainsi définie dans l'équation III.3.

Afin de pondérer l'importance relative de ces trois critères, il est nécessaire d'introduire trois constantes :  $F_1$ ,  $F_2$ , et  $F_3$  indiquant respectivement l'importance du premier, du deuxième et du troisième critère dans la fonction de *Fitness* ( $c_{ij}^k$ ) définie dans l'équation III.6. Ces valeurs devront être fixées en fonction de l'instance du problème à résoudre.

Définissons ainsi :

- $D_{ij}^k$  : le temps en secondes nécessaire au véhicule  $k$  pour se rendre du point de livraison de la mission  $i$  au point de collecte de la mission  $j$ . Si  $i = j$  il s'agit du temps nécessaire au véhicule  $k$  pour se rendre du point de collecte du conteneur au point de livraison. Si  $i = 0$  il s'agit du temps nécessaire pour rallier le point de collecte de la mission  $j$  en partant du dépôt, et à contrario, si  $j = 0$ , il s'agit alors du temps nécessaire pour rentrer au dépôt après avoir effectué la mission  $i$ .
- $T_{ij}^k$  : le retard en secondes lors de l'arrivée du véhicule  $k$  au point de collecte de la mission  $j$  après être parti du point de livraison de la mission  $i$  ;
- $E_{ij}^k$  : l'avance en secondes lors de l'arrivée du véhicule  $k$  au point de collecte de la mission  $j$  après être parti du point de livraison de la mission  $i$  ;
- $c_{ij}^k$  : l'évaluation (coût) de l'enchaînement de la mission  $i$  à la mission  $j$  pour le véhicule  $m_k$ .

et les équations suivantes :

$$(III.3) \quad D_{ij}^k = \frac{distance_{ij} + distance_{jj}}{vitesse_k} + attente_{ij}^k(t) + attente_{jj}^k(t_P^k)$$

avec  $t=C_i^k$  ou  $t=R^k$  ou  $t=t^{now}$  et  $t_P^k$  correspond à la date d'arrivée de  $k$  au point de collecte de  $j$

$$(III.4) \quad T_{ij}^k = \max(0, t_P^k - TW_{j,1}^{\max}) + \max(0, t_D^k - TW_{j,2}^{\max})$$

avec  $t_P^k$  le temps d'arrivée de  $k$  au point de collecte de  $j$  et  $t_D^k$  le temps d'arrivée de  $k$  au point de livraison de  $j$

$$(III.5) \quad E_{ij}^k = \max(0, TW_{j,1}^{\min} - t_P^k) + \max(0, TW_{j,2}^{\min} - t_D^k)$$

avec  $t_P$  le temps d'arrivée de  $k$  au point de collecte de  $j$  et  $t_D$  le temps d'arrivée de  $k$  au point de livraison de  $j$

$$(III.6) \quad c_{ij}^k = D_{ij}^k \cdot F_1 + T_{ij}^k \cdot F_2 + E_{ij}^k \cdot F_3$$

$c_{ij}^k$  définissant le coût de l'enchaînement de la mission  $j$  après la mission  $i$  par le chariot cavalier  $k$ , les arcs du graphe seront donc pondérés par cette valeur.

La fonction globale d'évaluation à minimiser correspondra à l'équation suivante :

$$(III.7) \quad \min \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^m (c_{ij}^k \cdot x_{ij}^k)$$

Avec  $x_{ij}^k$  : variable binaire indiquant si l'arc  $(ij)$  fait partie de la tournée du voyageur  $k$ .

Il peut être intéressant pour le terminal de considérer le dépassement des fenêtres de temps uniquement pour les dépassements engendrant le paiement de pénalités aux clients. Dans ce cas, la fonction de retard devra prendre en compte les variables binaires  $y_j$  et  $z_j$  indiquant si le retard des fenêtres de temps respectivement de collecte et de livraison de la mission  $j$  doivent être comptabilisés.  $y_j = 1$  si un retard au point de collecte de la mission  $j$  engendrera des frais pour le terminal, 0 sinon.  $z_j = 1$  si un retard au point de livraison de la mission  $j$  provoquera le versement de pénalités au client, 0 sinon. La fonction de retard précédemment définie par l'équation III.4 est alors remplacée par l'équation suivante :

$$(III.8) \quad T_{ij}^k = \max(0, (t_P^k - TW_{j,1}^{\max}) \cdot y_j) + \max(0, (t_D^k - TW_{j,2}^{\max}) \cdot z_j)$$

avec  $t_P^k$  le temps d'arrivée de  $k$  au point de collecte de  $j$  et  $t_D^k$  le temps d'arrivée de  $k$  au point de livraison de  $j$

La fonction d'évaluation prend désormais en compte la somme des retards pondérés.

### III.2.1.3 Coût de réalisation d'une mission

Lorsqu'un voyageur de commerce visite une ville  $i$  cela correspond à la réalisation d'une mission par un chariot cavalier dans le contexte du terminal à conteneurs. Cette réalisation comporte ainsi 6 étapes :

- Le chariot se déplace du point de livraison de la mission  $i - 1$  si elle existe, ou de sa position courante sinon, au point de collecte du conteneur de la mission  $i$ ;
- Le chariot attend que le conteneur à collecté soit disponible (si il est arrivé en avance, ou que le camion à déchargé n'est pas encore arrivé, etc);
- Le chariot prend en charge le conteneur ;
- Le chariot se déplace vers le point de livraison de la mission  $j$  ;
- Le chariot attend que le conteneur puisse être livré (s'il est arrivé en avance, que le véhicule du client à livrer n'est pas encore arrivé, etc) ;
- Le chariot dépose le conteneur.

À ces 6 étapes il faut rajouter une étape supplémentaire dans le cas où le chariot à terminé sa tournée. Dans ce cas il doit se rendre au dépôt. La réalisation d'une mission a donc un coût en terme d'utilisation du chariot cavalier qui se retrouve mobilisé de la première étape à la sixième voire à la septième. Le temps d'occupation peut varier en fonction des temps d'attente du chariot et pendant ces temps d'attente, le véhicule ne peut pas être utilisé pour réaliser d'autres missions, ni même d'autres parties de missions (problème de tournées de véhicules avec transbordement). Il est ainsi bon de remarquer qu'il n'existe pas de préemption pour une mission. En effet, une fois que le conteneur de la mission a été pris en charge par le chariot cavalier, la capacité de ce dernier est atteinte et il devra obligatoirement déposer le conteneur avant de pouvoir en charger un autre. Le fait de déposer le conteneur de façon temporaire sur un emplacement afin de pouvoir déplacer un autre conteneur sera alors vu comme une modification de la première mission. Ainsi, l'emplacement et la fenêtre de temps de livraison du conteneur seront modifiés et une nouvelle mission consistant à collecter le conteneur sur l'emplacement temporaire puis à le livrer à l'emplacement initial de livraison sera ajoutée.

Le temps de manutention d'un conteneur est supposé fixe est sera décrit par la constante  $\kappa$ . La durée totale de manutention (prise et dépose) du conteneur par mission sera donc égale à  $2\kappa$ .

La durée minimale<sup>1</sup> de réalisation d'une mission  $i$  sera donc définie par l'équation III.9 où  $t_P^{\text{start}}$  correspond à la date de début de la phase de collecte du conteneur et  $t_D^{\text{start}}$  correspond à la date de début de la phase de livraison, c'est-à-dire à la date de fin de collecte du conteneur.

$$(III.9) \quad duree_i^k = \max\left(d_{i-1,i}^k, TW_{i,1}^{\min} - t_P^{\text{start}}\right) + \max\left(d_{i,i}^k, TW_{i,2}^{\min} - t_D^{\text{start}}\right) + 2\kappa$$

### III.2.1.4 Dynamique du graphe

La durée de parcours d'un chariot cavalier d'un point à un autre du terminal est donc fonction du temps. La pondération des arcs est par conséquent dynamique et évolue ainsi au fil du temps. Mais il ne s'agit pas là de la seule source de dynamique dans le graphe constitué des missions. En effet, seule une partie de ces dernières sont connues à l'avance. Lorsqu'une nouvelle mission arrive dans le système, un sommet est ajouté au graphe ainsi que les arcs permettant de conserver la structure de la clique. La ville correspondante devra être insérée dans une tournée d'un des voyageurs. Au contraire, lorsqu'une mission est annulée, le sommet correspondant est retiré du graphe ainsi que tous les arcs connexes. La ville équivalente sera également supprimée de la tournée du voyageur de commerce qui lui avait été affecté. Une mission peut-être également mise-à-jour. L'une ou plusieurs de ses caractéristiques peuvent être modifiées au fil du temps, comme l'emplacement de collecte et/ou de livraison ou les dates de ses fenêtres de temps. Dans ce cas, c'est le poids des arcs entrants et sortants du sommet correspondant à la mission qui devront être mis-à-jour. Dans tout les cas ces modifications du graphe provoque le besoin de recalculer les tournées des voyageurs de commerce afin de prendre en compte les nouvelles caractéristiques globales du problème.

### III.2.1.5 Formulation mathématique

La formulation mathématique du problème d'ordonnancement et d'affectation de missions aux chariots cavaliers reprend la formulation du problème de tournées de véhicules décrite dans le chapitre II (voir p. 68).

Les seules réelles différences se situent au niveau de la fonction objectif et au niveau de la capacité des véhicules. La fonction d'évaluation doit en effet prendre en compte les coûts des déplacements. La flotte de véhicules étant hétérogène, l'équation II.15 (voir p. 69) est remplacée par l'équation III.10 ci-dessous. D'autre part, ne pouvant déplacer qu'un unique conteneur à

---

1. En supposant que les véhicules clients soient présents sur le terminal dès le début de la fenêtre de temps

la fois  $Q^k = 1$ ,  $\forall 1 \leq k \leq m$ , et chaque mission ne concernant qu'un seul conteneur  $q_i = 1$ ,  $\forall 1 \leq i \leq n$ , la contrainte mathématique de l'équation II.21 (voir p. 69) devient inutile et peut être supprimée.

Le problème s'écrit alors de la façon suivante :

$$(III.10) \quad \min \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^m (c_{ij}^k \cdot x_{ij}^k)$$

$$(III.11) \quad \sum_{i=1}^n \sum_{k=1}^m x_{ij}^k = 1, \forall 1 \leq j \leq n$$

$$(III.12) \quad \sum_{j=1}^n \sum_{k=1}^m x_{ij}^k = 1, \forall 1 \leq i \leq n$$

$$(III.13) \quad \sum_{i=1}^n x_{ip}^k - \sum_{j=1}^n x_{pj}^k = 0, \forall 1 \leq k \leq m; \forall 1 \leq p \leq n$$

$$(III.14) \quad \sum_{i=1}^n x_{i0}^k = 1, \forall 1 \leq k \leq m$$

$$(III.15) \quad \sum_{j=1}^n x_{0j}^k = 1, \forall 1 \leq k \leq m$$

$$(III.16) \quad x_{ij}^k \in \{0, 1\}, \forall 0 \leq i, j \leq n, \forall 1 \leq k \leq m$$

L'équation III.10 correspond à la fonction d'évaluation. Les équations III.11 et III.12 indiquent qu'un véhicule ne peut effectuer qu'une seule fois une même mission et qu'une mission ne peut être effectuée que par un seul véhicule. La contrainte de continuité de la tournée indiquant qu'un véhicule effectuant la livraison d'un conteneur repart de son point de livraison après avoir accompli sa tâche est indiqué par l'équation III.13. Les équations III.14 et III.15 indiquent que les véhicules débutent et terminent leur tournée au dépôt. Enfin, l'équation III.16 définit la variable binaire  $x_{ij}^k$ .

Le problème d'ordonnancement et d'affectation des déplacements de conteneurs aux chariots cavaliers peut donc être modélisé sous la forme d'un problème de voyageurs de commerce. Il est également possible de le modéliser sous forme d'un problème d'atelier.

### III.2.2 Problème d'atelier

La section précédente modélise le problème d'affectation et d'ordonnancement des déplacements de conteneurs aux chariots cavaliers en tant qu'un problème de tournées de véhicules. Néanmoins, un problème d'ordonnancement et d'affectation comme celui-ci peut également être modélisé sous forme d'un problème d'atelier et plus précisément de *Job Shop*.

Sous cette forme, chaque chariot cavalier correspond à une machine et chaque mission de déplacement de conteneur correspond à une tâche. Le problème consiste à déterminer l'ordonnancement (*Schedule*)  $S$  des  $n$  tâches (*Jobs*)  $J_i$  ( $i = 1, \dots, n$ ) sur  $m$  machines  $M_j$  ( $j = 1, \dots, m$ ). Cet ordonnancement est représenté par le plan de charge (*Workload*)  $W_{M_j}$  de chaque machine  $M_j$ . Le plan de charge  $W_{M_j}$  est constitué d'une liste ordonnée de tâches allouées à la machine  $M_j$ .

$$(III.17) \quad \begin{cases} S = \{W_1, \dots, W_m\} & \text{et} \\ W_i = \{J_{\alpha_1}, \dots, J_{\alpha_k}\} & \text{avec } k \leq n, \text{ et } W_i \cap W_j = \emptyset, \forall i \neq j \end{cases}$$

Chaque tâche comporte deux opérations :

- $O_1$  : collecte du conteneur ;
- $O_2$  : livraison du conteneur.

Ces deux opérations concernent la manutention d'un même conteneur, elles doivent donc être réalisées par la même machine (chariot cavalier), et ce en respectant l'ordre  $O_1$  puis  $O_2$ . De plus, entre l'exécution des deux opérations, le véhicule doit se rendre d'un emplacement (point de collecte du conteneur) à un autre (point de livraison) et ce déplacement constitue un délai entre  $O_1$  et  $O_2$  lié à la distance entre les deux emplacements, à la vitesse du véhicule ainsi qu'à l'état du trafic sur le réseau routier du terminal.

#### III.2.2.1 Coût de mise en place

Lorsqu'un chariot cavalier débute la réalisation d'une mission, il doit d'abord se rendre au point de collecte du conteneur. La durée de déplacement peut-être considérée comme un coût de mise en place (*setup time cost*). Ce coût dépend directement de la position du véhicule au début de la mission et cette position dépend du plan de charge du véhicule. En effet, si le chariot cavalier vient de terminer une mission au moment de débuter la mission courante, alors la durée de déplacement entre la position du véhicule et le point de collecte de la mission à réaliser correspond à la distance entre le point de livraison de la mission précédente et le point

de collecte de la mission courante. En revanche, si le véhicule est inutilisé au commencement de la nouvelle mission, cela signifie que le véhicule est soit au dépôt, soit sur le chemin du dépôt. La durée du déplacement est calculée alors entre la position courante du véhicule et le point de collecte de la mission. Les coûts de mise en place dépendent ainsi de l'enchaînement des missions du véhicule concerné (*sequence dependent setup times* :  $ST_{sd}$ ).

Définissons deux tâches  $J_i$  et  $J_j$  chacune composée de deux opérations  $O_i^1$ ,  $O_i^2$  et  $O_j^1$ ,  $O_j^2$ . Chaque opération doit être réalisée à des emplacements précis sur le terminal. Ainsi  $position(O_p^k)$  correspond à l'emplacement (*location*) de l'opération  $p$  ( $p \in \{1, 2\}$ ) de la tâche  $J_k$ . Dans ce cas, si une machine est affectée à deux tâches  $J_i$  et  $J_j$ , dans cet ordre, alors le coût de mise en place sera proportionnel à  $D(position(O_2^i), position(O_1^j))$  où  $D(position, position')$  correspond à la distance du plus court chemin entre les emplacements *position* et *position'*.

La durée de mise en place de la machine sur une tâche est également liée à la vitesse du véhicule correspondant à la machine, ainsi qu'au trafic présent sur le terminal au moment de la réalisation de la mission. En effet, une travée de conteneurs ne pouvant être empruntée que par un seul chariot cavalier à la fois, les temps d'attente en entrée et sortie de travée peuvent augmenter le temps nécessaire à un véhicule pour atteindre un emplacement.

Le coût de mise en place d'une machine  $M_k$  pour la tâche  $J_j$  après avoir réalisé la tâche  $J_i$  est ainsi défini par l'équation III.18 où  $t$  correspond à la date d'exécution de  $M_j$  c'est-à-dire soit à la date de fin de service la mission  $M_i$  (*Completion Time* :  $C_i^k$ ) si elle existe, à la date de disponibilité du véhicule s'il est en panne (*Repair Completion Time* :  $R^k$ ) ou soit au temps courant ( $t^{now}$ ) si le véhicule est inutilisé.

$$(III.18) \quad setup_{ij}^k = \frac{distance_{ij}}{vitesse_k} + attente_{ij}^k(t)$$

### III.2.2.2 Fonction d'évaluation

L'objectif est d'ordonnancer les tâches sur les machines de façon à minimiser la valeur de la fonction d'évaluation. Cette fonction reprend les critères définis pour la modélisation sous forme de problème de tournées de véhicules. Ainsi, la distance totale parcourue par les véhicules ainsi que la somme des retards pondérés sont les deux critères principaux. Dans une moindre mesure, l'avance des véhicules lors de l'arrivée sur les points de collecte et de livraison est également prise en compte afin de guider plus efficacement l'algorithme de résolution dans la recherche de la solution optimale.

Missions	Collecte		Livraison	
$M_1$	00 : 01 : 09	00 : 03 : 17	00 : 03 : 52	00 : 06 : 00
$M_2$	00 : 01 : 32	00 : 04 : 10	00 : 04 : 35	00 : 06 : 47
$M_3$	00 : 07 : 10	00 : 09 : 52	00 : 09 : 14	00 : 12 : 20

Fenêtres de temps ( $hh : mm : ss$ )

Vehicle	Vitesse moyenne (km/h)
$V_1$	20
$V_2$	25

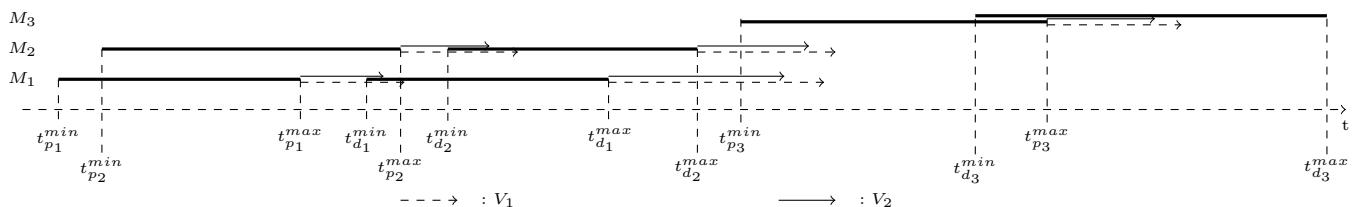
Vitesse des véhicules (km/h)

Origin	Destination		
	Depot	$M_1$	$M_2$
Depot	0	173	334
$M_1$	347	306	642
$M_2$	344	317	413
$M_3$	348	399	396

Distance en mètres (lecture :  $Depot \rightarrow M_1 : 173m$ )

Origine	Destination							
	Depot		$M_1$		$M_2$		$M_3$	
	$V_1$	$V_2$	$V_1$	$V_2$	$V_1$	$V_2$	$V_1$	$V_2$
Depot	00 : 00 : 00	00 : 00 : 00	00 : 00 : 31	00 : 00 : 25	00 : 01 : 00	00 : 00 : 48	00 : 00 : 59	00 : 00 : 47
$M_1$	00 : 01 : 02	00 : 00 : 50	00 : 00 : 55	00 : 00 : 44	00 : 01 : 56	00 : 01 : 32	00 : 01 : 54	00 : 01 : 32
$M_2$	00 : 01 : 02	00 : 00 : 50	00 : 00 : 57	00 : 00 : 46	00 : 01 : 14	00 : 00 : 59	00 : 01 : 13	00 : 00 : 59
$M_3$	00 : 01 : 03	00 : 00 : 50	00 : 01 : 12	00 : 00 : 57	00 : 01 : 12	00 : 00 : 57	00 : 01 : 11	00 : 00 : 57

Temps de parcours en fonction du véhicule (lecture :  $Depot \rightarrow M_1$  pour  $V_1 : 31s$ )



**Figure III.2:** Exemple d'un problème à 2 machines ( $V_1$  et  $V_2$ ) and 3 tâches ( $M_1$ ,  $M_2$  et  $M_3$ ).

### III.2.2.2.1 Distance et durée de parcours

Pour chaque séquence de tâches ( $J_i, J_j$ ) réalisées par une machine  $M_k$ , la distance couverte est égale à :

$$(III.19) \quad \text{distance}(\text{position}(O_1^i), \text{position}(O_2^i)) + \text{distance}(\text{position}(O_1^j), \text{position}(O_2^j))$$

Si  $J_i$  est la première tâche exécutée par la machine, alors la distance vaudra :

$$(III.20) \quad \text{distance}(\text{position}(\text{depot}), \text{position}(O_1^i))$$

Si  $J_j$  est la dernière tâche exécutée, alors une distance supplémentaire doit être ajoutée à la distance totale parcourue correspondant au retour du véhicule au dépôt :

$$(III.21) \quad \text{distance}(\text{position}(O_2^j), \text{position}(\text{depot}))$$

Comme les autres critères d'évaluation d'une solution sont des mesures de temporelles, il est nécessaire de convertir cette distance parcourue en durée de parcours. Cette durée est obtenue

en prenant en compte à la fois la vitesse du véhicule qui a couvert la distance, et à la fois le temps d'attente lié au trafic sur ce parcours pour ce véhicule. La durée est ainsi calculée selon l'équation III.3 (voir p. 109) également utilisée par la modélisation sous forme de tournées de véhicules.

La durée totale du parcours d'un chariot cavalier  $M_k$  liée à la réalisation des  $\text{card}(W_k)$  tâches de son plan de charge  $W_k$  est définie par l'équation suivante où  $\kappa$  définit le temps de pose et de dépose d'un conteneur et est supposé constant :

$$(III.22) \quad D^k = D_{(\text{depot},1)}^k + \sum_{i=2}^{\text{card}(W_k)} \left( D_{(i-1,i)}^k \right) + D_{(\text{card}(W_k),\text{depot})}^k + 2\kappa$$

### III.2.2.2.2 Retard et Avance

Le deuxième et le troisième critère sont liés aux fenêtres de temps associées aux opérations des tâches. Si une mission dépasse une ou ses deux fenêtres de temps, alors un retard est comptabilisé. En fonction du type d'opération concernée, le retard peut engendrer le paiement de pénalités aux clients du terminal. Le retard ainsi comptabilisé devra être pondéré afin de ne prendre en compte uniquement les retards ayant un impact négatif sur le terminal. On définit ainsi  $T_{ij,O_1}$  le retard (*Tardiness*) lié au dépassement de la fenêtre de temps de collecte du conteneur de la mission  $M_j$  après avoir réalisé la mission  $M_i$ , et  $T_{ij,O_2}$  le retard lié au dépassement de la fenêtre de livraison de cette même mission.  $y_j$  et  $z_j$  sont des variables binaires indiquant respectivement si le retard de collecte et de livraison doivent être pris en compte. Le retard total  $T_{ij}^k$  correspondra à la somme des retards de collecte et de livraison (voir III.25).

$$(III.23) \quad T_{ij,O_1}^k = \max \left( 0, (t_P^k - TW_{j,1}^{\max}) \cdot y_j \right)$$

$$(III.24) \quad T_{ij,O_2}^k = \max \left( 0, (t_D^k - TW_{j,2}^{\max}) \cdot z_j \right)$$

$$(III.25) \quad T_i^k j = T_{ij,O_1} + T_{ij,O_2}$$

De la même façon, l'avance (*earliness*) d'un chariot cavalier  $M_k$  pour une mission  $j$  après avoir réalisé la mission  $i$  est définie par la somme de l'avance du chariot cavalier au point de collecte au point de livraison de  $j$  :

$$(III.26) \quad E_{ij,O_1}^k = \max \left( 0, TW_{j,1}^{\min} - t_P^k \right)$$

$$(III.27) \quad E_{ij,O_2}^k = \max \left( 0, TW_{j,1}^{\min} - t_P^k \right)$$

$$(III.28) \quad E_{ij}^k = E_{ij,O_1}^k + E_{ij,O_2}^k$$

La fonction globale d'évaluation est donc définie par l'équation suivante :

$$(III.29) \quad c_{ij}^k = D_{ij}^k \cdot F_1 + T_{ij}^k \cdot F_2 + E_{ij}^k \cdot F_3$$

Le critère de l'avance ne servant qu'à guider l'algorithme de résolution il peut être écarté de la fonction d'évaluation. Le problème d'atelier correspondant au problème d'ordonnancement et d'affectation des missions aux chariots cavaliers est alors défini par la notation de Graham (voir II.4.1.1, p. 86) suivante :  $J|ST_{sd}, R_{sd}| \sum w_j \cdot T_j, \sum distance(i)$ .

### III.2.3 Conclusion

La modélisation sous forme de problème d'atelier reprend exactement les mêmes équations que la modélisation sous forme de tournées de véhicules. Les seules différences se situent dans la formulation du problème et dans le vocabulaire ainsi utilisé. La résolution de l'une ou l'autre modélisation sera la même et visera à minimiser la valeur de la fonction globale d'évaluation proposée dans l'équation III.10 (voir p. 112).

Dans les deux formulations, la dynamique influence les données du modèle et chaque événement se produisant conduira à vérifier la validité de la solution préalablement calculée. Si la solution est devenue inutilisable, une nouvelle solution devra être calculée.

## III.3 Méthodes de résolution

L'objet de cette thèse est de développer une méthode d'optimisation permettant de proposer une solution approchée à un problème dynamique sous incertitude. L'objectif est de proposer une méthode de résolution basée sur les connaissances actuelles des caractéristiques du problème sans prendre en compte les événements futurs. En effet, il ne sera pas question ici d'essayer de prévoir la réalisation ou la non réalisation, ni même la probabilité de réalisation d'un événement, mais plutôt de réagir aux nouvelles caractéristiques afin de toujours proposer une solution réalisable.

Afin de mesurer la performance relative de notre algorithme, une méthode de résolution exacte du problème est également proposée et revêt la forme d'un algorithme de *Branch and Bound*. Cependant, la complexité de ce problème rend inapplicable une telle méthode même avec des instances de taille raisonnable. Des méthodes approchées ont alors été élaborées et utilisent des heuristiques gloutonnes selon différentes politiques. Nous proposons ainsi dans ce chapitre le détail de ces méthodes ainsi que le fonctionnement de nos algorithmes dynamiques. Deux versions sont ainsi élaborées. L'une propose une solution hors-ligne et itérative au problème et doit être exécuté lors de chaque nouvel événement. L'autre est une méthode continue dite en ligne (*on-line*) qui permet de proposer une solution à tout moment.

### III.3.1 Méthode de résolution exacte

La recherche de la solution optimale au problème comporte deux étapes. D'une part la recherche d'une solution valide, puis prouver qu'il n'existe pas de solution plus performante que celle-ci. Pour ce faire, il est nécessaire de parcourir l'espace de recherche. Toutefois, cet espace est immense et une recherche exhaustive des solutions est inapplicable. Afin d'éviter de parcourir inutilement tout l'espace de recherche, une borne supérieure est utilisée. L'initialisation de cette borne est réalisée grâce à une méthode gloutonne, puis elle est mise-à-jour en fonction des nouvelles solutions trouvées. Cette borne correspond à la valeur de la fonction d'évaluation de la meilleure solution trouvée jusqu'alors. Elle indique ainsi que la solution optimale est au moins aussi performante que la meilleure solution trouvée jusque là. Une fois la borne initialisée, un parcours exhaustifs de l'espace des solutions est réalisé. Chaque mission est insérée dans le plan de charge de chaque véhicule selon toutes les combinaisons possibles. Après chaque insertion d'une mission dans un plan de charge, le score de la solution partielle est évalué et s'il dépasse la borne supérieure alors le sous-arbre de recherche ne sera pas exploré. Ceci permet de couper des branches entières de l'arbre de recherche des solutions.

#### III.3.1.1 Exemple

La figure III.2 montre la construction de l'arbre des solutions du problème à 2 machines et 3 missions de l'exemple de la figure III.2 (voir p. 115).

Dans cet exemple, les hypothèses les plus favorables sont considérées :

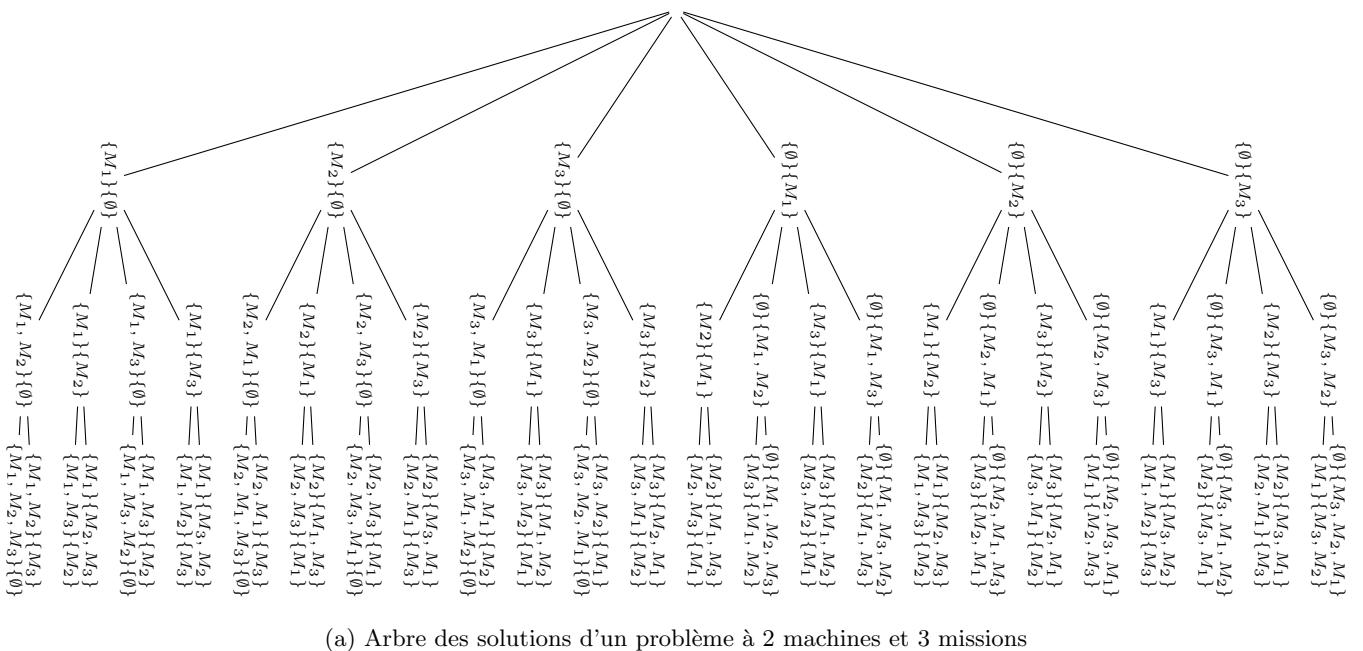
- il n'y a pas de temps d'attente sur le réseau routier du terminal ( $\text{attente}_{ij}^k(t) = 0s$ ) ;
- les véhicules des clients sont sur place dès le début de la fenêtre de temps ;
- tous les retards impliquent des pénalités ( $y_i = z_i = 1, \forall i \in [1; n]$ ) ;

- les temps de pose et de dépose des conteneurs sont considérés nuls ( $\kappa = 0s$ ) ;
  - les véhicules sont stationnés au dépôt au moment de l'exécution de la première mission ;
  - tous les véhicules sont disponibles ;
  - $F_1 = 1$ ,  $F_2 = 10$  et  $F_3 = 2$ .

L'arbre de la figure III.2(a) est construit par un processus de retour sur trace (*backtracking*) permettant de générer toutes les combinaisons possibles, sans toutefois gérer les doublons. Lors de l'évaluation, les branches de l'arbre ayant déjà été évaluées (doublons) seront coupées.

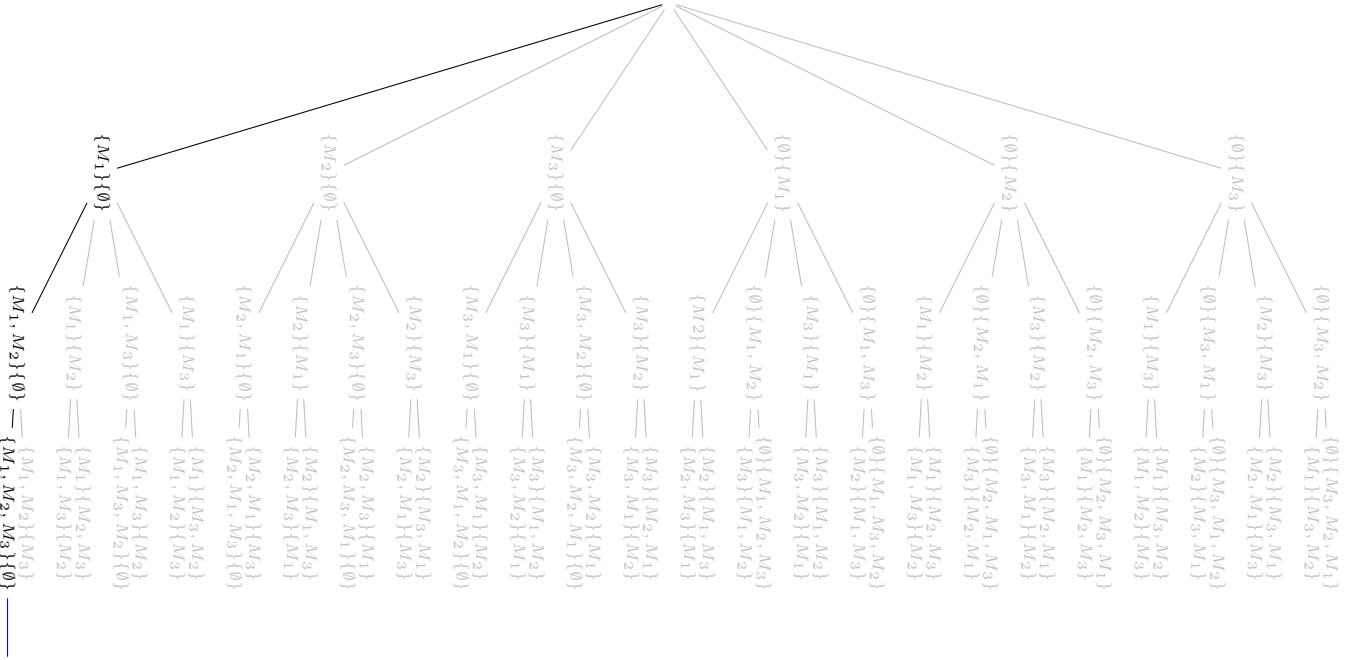
Il est intéressant de noter que malgré la trivialité de cet exemple, il existe déjà 24 combinaisons possibles d'ordonnancement et d'affectation des missions aux chariots cavaliers.

La figure III.2(b) décrit l'initialisation de la borne supérieure réalisé en affectant toutes les missions au premier véhicule. En prenant en compte les hypothèses précédente, la branche  $\{M_1, M_2, M_3\}\{\emptyset\}$  de l'arbre est construite et permet d'initialiser la borne à 1829. Le calcul des chemins se poursuit donc avec cette borne jusqu'à trouver une nouvelle borne supérieure (voir III.2(c)) : ici 1211. Les calculs se poursuivent et la borne supérieure de 1211 se trouve dépassée lors du calcul du sous-arbre  $\{M_2, M_1\}\{\emptyset\}$  dont le score est de 2084. Les sous-branches de cette partie de l'arbre sont donc élaguées (voir III.2(d)). La recherche de borne supérieure est poursuivie jusqu'à ce que toutes les branches de l'arbre aient été soit évaluées soit élaguées. La solution optimale est donc :  $\{M_2, M_3\}\{M_1\}$ , dont le score de 1186 est indiqué sur le sous arbre rouge (voir `reffig :arbreBB(e)`).

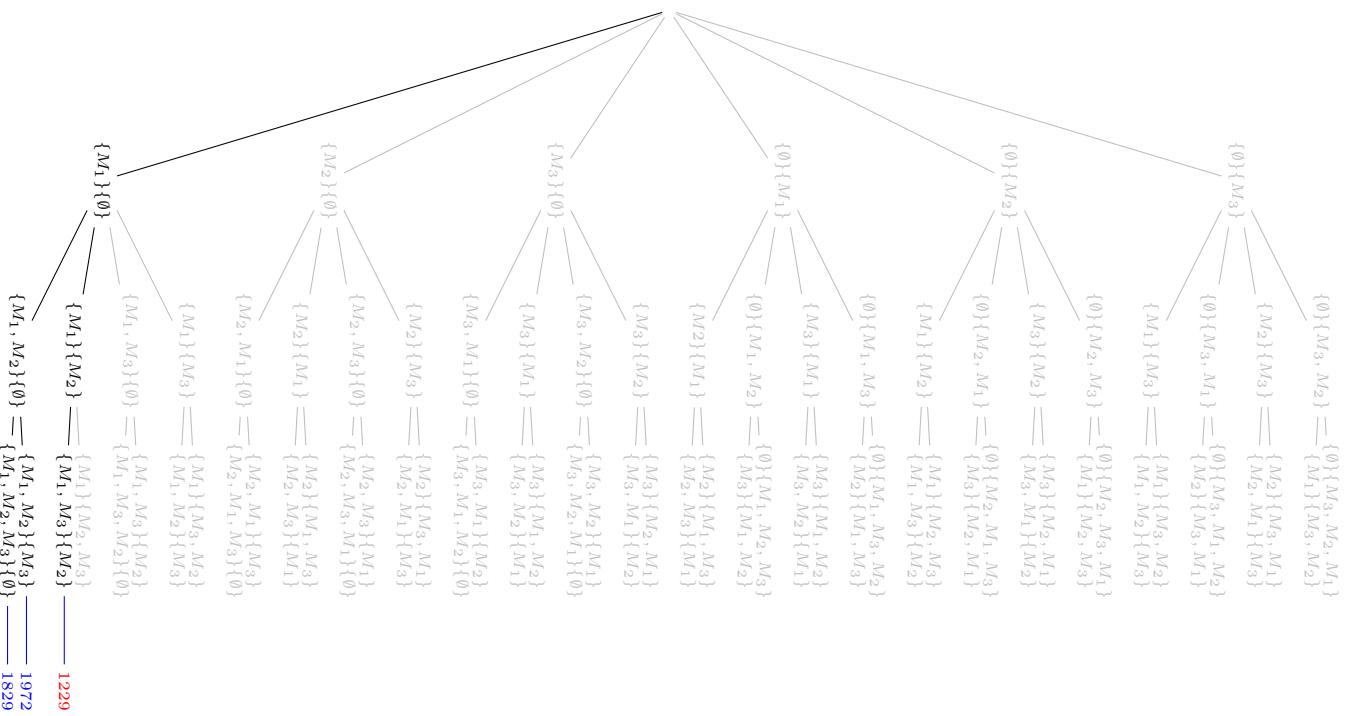


Dans l'exemple précédent, la première borne trouvée est relativement fine. Néanmoins, elle

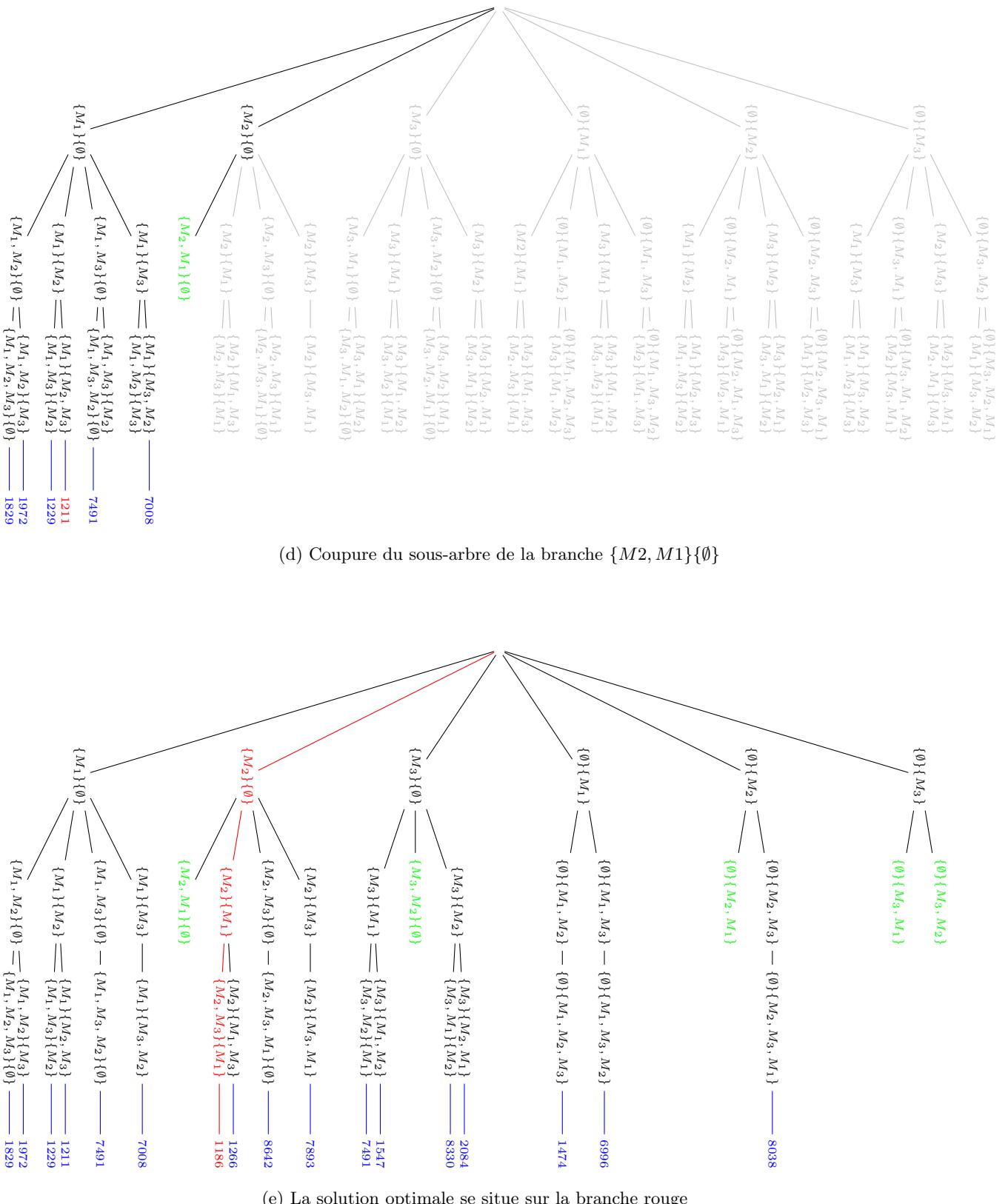
aurait pu être bien plus large et par conséquent requérir l'exploration d'un nombre supérieur de sous-arbres. Une initialisation fine de la borne supérieure peut donc influencer positivement la performance de l'algorithme (vitesse de convergence). D'autre part, l'arbre de cet exemple est parcouru en profondeur dans l'ordre préfixe. L'évaluation est donc réalisée dès le premier sommet (racine  $\rightarrow \{M1\}\{\emptyset\}$ ), ce qui permet d'arrêter l'évaluation d'une branche dont la partie supérieure dépasse la borne courante.



(b) Initialisation de la borne supérieure



(c) recherche d'une nouvelle borne supérieure : ici 1229



**Figure III.2:** Exemple de fonctionnement de l'algorithme de résolution exacte

### III.3.1.2 Algorithmes

L'algorithme 1 et la fonction d'évaluation présentés p.124 décrivent le fonctionnement de la méthode de séparation et évaluation. Cette méthode de résolution utilise l'arbre des solutions construit par un mécanisme de retour sur trace exhaustif (*backtracking*). La seconde étape est l'évaluation de la borne supérieure initiale et est réalisée grâce à une des heuristiques présentées dans la section suivante (voir III.3.2 p.124). Une fois l'initialisation terminée, l'évaluation des sous-arbres débute par la racine de l'arbre. La fonction  $\text{evaluation}(A, S)$  consiste donc à évaluer les fils de la racine de l'arbre  $A$ . Si un fils possède un score supérieur à la borne courante, sa branche est éliminée. Dans le cas contraire, la descente se poursuit récursivement dans son sous-arbre. Si le fils est une feuille et que son score est inférieur à la borne supérieure courante, alors cela signifie qu'il constitue une nouvelle borne. Une fois que tous les fils de la racine de l'arbre ont été évalués, la fonction d'évaluation renvoie le chemin qui relie la racine à une feuille et qui possède le score optimal.

Le processus de *backtracking* générant plusieurs fois les même combinaisons, il est nécessaire de maintenir une structure de mémorisation des solutions déjà évaluées. Ainsi, lors de l'évaluation d'un fils, si celui-ci a déjà été évalué, il est éliminé. Sinon il est ajouté à la mémoire et il est évalué.

Il est possible de paralléliser les appels à la fonction d'évaluation à condition de synchroniser

les accès à la borne supérieure ainsi qu'à la mémoire.

---

**Algorithme 1 :** Algorithme d'évaluation séparation

---

**Résultat** : Feuille de l'arbre ayant le score le plus faible

**1 début**

```
2   A ← initialiserArbre();  
3   S ← initialiserBorne(A);  
4   S ← evaluation(A,S);  
5   retourner S;
```

**6 fin**

---

---

**Fonction** evaluation( $A, S$ )

---

**Données :**

$A$  : sous-arbre à évaluer

$S$  : borne supérieure courante

**1 début**

```
2   pour chaque fils  $f$  de  $A$  faire  
3       fScore ← score( $f$ );  
4       si  $fScore \leq score(S)$  alors  
5           si  $f$  est une feuille alors  
6               |   S ←  $f$ ;  
7           sinon  
8               |   S ← evaluation( $f, S$ );  
9           fin  
10      fin  
11  fin  
12  retourner  $S$ 
```

**13 fin**

---

### III.3.2 Méthodes de résolution approchées

Les méthodes de résolution approchées permettent de trouver rapidement une solution en sacrifiant la garantie d'optimalité de cette dernière. La qualité de la solution ainsi trouvée dépendra fortement de la stratégie utilisée pour son élaboration. Les principales stratégies utilisées sont la méthode des plus proches voisins, la méthode aléatoire et la méthode de répartition de charge. Enfin une méthode, basée sur la méthode gloutonne a été développée afin d'améliorer

la qualité de la solution de cette dernière tout en modérant le temps de calcul.

### III.3.2.1 Algorithme d'affectation aléatoire

Cette méthode consiste à déterminer les plans de charge des véhicules de façon aléatoire. À chaque étape une mission est tirée au sort, puis un véhicule est également choisi aléatoirement. La mission est insérée à la fin du plan de charge du véhicule choisi. Le processus est répété jusqu'à ce que toutes les missions aient été affectées.

Il est clair qu'aucune évaluation ne guide le choix des missions et des véhicules, ni concernant l'affectation, ni vis-à-vis de l'ordre de l'ajout des missions dans les plans de charge. L'inconvénient de cette méthode est donc de bien souvent produire des solutions de piètre qualité. En revanche, l'absence de calculs conduit une grande rapidité d'obtention d'une solution.

Vu la rapidité d'obtention d'une solution grâce à cette méthode, il est envisageable d'améliorer la qualité de la solution produite. Il est ainsi possible, par exemple, de combiner cette méthode avec l'heuristique *2-opt* de Lin et Kernighan (voir [Lin and Kernighan, 1973]), ou dans le cas général *k-opt*, ou encore de produire plusieurs solutions et de ne conserver que la meilleure.

### III.3.2.2 Politique de répartition de charge

Cette méthode consiste à répartir les tâches sur les différentes ressources. Ainsi, les missions sont triées par date de début au plus tard, puis tour-à-tour affectées à un chariot cavalier différent. Lorsque tous les véhicules ont obtenu une mission, l'opération est répétée jusqu'à ce que toutes les missions aient été affectées.

Cette méthode permet d'éviter de dépasser les fenêtres de temps des missions en répartissant les missions proches temporellement sur différents chariots cavaliers. En revanche, elle provoque une augmentation de la distance parcourue car un grand nombre de véhicules sont utilisés et donc doivent rentrer au dépôt à la fin de leur tournée, alors que les trajets vers le dépôt sont minimisé en enchaînant plusieurs missions sur un même véhicule.

Au niveau de la complexité, cette méthode utilise un tri qui sera l'élément critique de performance de l'algorithme. Plus le tri sera efficace, plus la solution sera obtenue rapidement. De façon générale, les solutions sont obtenues très rapidement et sont de meilleure qualité que celles produites par la méthode aléatoire. Il est également possible de combiner cette méthode avec l'heuristique *k-opt* ou d'utiliser de l'aléatoire dans le choix de l'ordre des véhicules, ou de combiner cette dernière méthode afin de générer plusieurs solutions et de ne garder que la plus performante.

### **III.3.2.3 Algorithme glouton : méthode des plus proches voisins**

La méthode des plus proches voisins consiste à déterminer les enchaînements de missions dont les coûts sont les plus faibles. Les couples (ressource,tâche) sont triés par coût, puis le couple de poids minimal est choisi. L'affectation se poursuit jusqu'à ce que toutes les missions aient été attribuées. Ici, la notion de proximité correspond donc, non pas à la distance entre les missions, mais bien à la valeur de la fonction d'évaluation associées à l'enchaînement des missions.

Cette méthode gloutonne permet d'obtenir rapidement des solutions sans garantie d'optimalité. Il s'agit d'une optimisation locale liée à chaque nouvelle affectation de mission. L'optimisation globale résultant de la somme des optimisations locales est bien souvent de mauvaise qualité. La méthode développée dans le paragraphe suivant permet d'améliorer la qualité de la solution obtenue par la méthode des plus proches voisins.

### **III.3.2.4 Méthode gloutonne élaborée**

Cette heuristique consiste à déterminer pour chaque mission à affecter, à la fois le véhicule à associer ainsi que la position d'insertion de la mission dans le plan de charge du véhicule, de façon à minimiser la valeur de la fonction d'évaluation. Ainsi, pour chaque mission, toutes les combinaisons possibles d'insertion sont calculées et la mission est insérée dans le plan de charge du véhicule et à l'indice de coût minimal. Il s'agit également d'optimisation locale car le résultat dépend de l'ordre dans lequel les missions sont considérées, mais qui fournit des solutions de meilleure qualité que la simple méthode des plus proches voisins. Toutefois, le temps de calcul d'un tel ordonnancement est bien supérieur à celui des méthodes heuristiques évoquées précédemment du fait du nombre de combinaisons à tester. Elle ne sera donc pas utilisable pour des instances de grande taille.

## **III.3.3 Méthodes dynamiques**

Les méthodes heuristiques permettent d'obtenir une solution au problème en privilégiant la rapidité de calcul à la qualité. La complexité du problème, même dans le cas statique, rend inapplicable une approche de résolution exacte. Dans le cas dynamique il sera nécessaire de recalculer la solution à chaque changement d'une caractéristique du problème. Les méthodes de résolution doivent donc être en mesure de calculer une solution suffisamment rapidement pour rester valide au moment de son application. Parallèlement au besoin de réactivité, la méthode

de résolution doit fournir une solution dont la qualité est proche de la solution optimale.

Pour toutes ces raisons, cette thèse introduit une méthode de résolution dynamique basée sur une métaheuristique. Une méthode à base de colonies de fourmis est utilisée pour produire l'ordonnancement et l'affectation de missions aux chariots cavaliers. Deux versions de l'algorithme ont été mises au point : l'une permettant d'obtenir une solution de façon itérative et l'autre de façon continue.

La première méthode est dite hors-ligne (*off-line*) et permet de calculer une solution dès qu'une caractéristique du problème a changé. La solution ainsi obtenue est complète. La seconde méthode, est une méthode en ligne (*on-line*) permettant de calculer continuellement une solution au problème. Cette solution peut ici être partielle et les missions non-affectées à l'instant  $t$  le seront à l'instant  $t + t'$ .

### III.3.3.1 Méthode de résolution hors-ligne à colonies de fourmis

Cette méthode résout de façon dynamique le problème de voyageurs de commerce associé au problème d'ordonnancement et d'affectation de missions. La métaheuristique reprend le comportement naturel des individus d'une colonie de fourmis. Ici, chaque fourmi artificielle de la colonie représente un voyageur de commerce, c'est-à-dire un chariot cavalier. Il y a donc  $m$  fourmis dans la colonie qui évolue sur un graphe.

#### III.3.3.1.1 Graphe

Soit le graphe complet  $G^{(t)} = (V, A)$  où :

- $V$  est l'ensemble des sommets et est constitué des villes à visiter (missions à réaliser)  $v_1 \dots v_n$  plus le sommet représentant le dépôt des véhicules  $v_0$  ;
- $A$  est l'ensemble des arcs  $(v_i, v_j)$  indiquant la visite de la ville  $v_j$  immédiatement après  $v_i$ .

La section III.2.1 (voir p. 105) décrit la construction d'un tel graphe.

Chaque fourmi artificielle est initialisée sur le noeud de dépôt et l'algorithme prend en compte :

- soit la position courante du chariot cavalier correspondant s'il est inactif au moment du calcul de la solution ;
- soit la position future du véhicule ainsi que sa date de disponibilité s'il est occupé.

### III.3.3.1.2 Gestion de la phéromone

L'algorithme proposé fonctionne sur un mécanisme de collaboration/compétition entre les fourmis. L'objectif pour une fourmi est de lutter contre les autres fourmis afin de coloniser des missions à réaliser pour le chariot cavalier correspondant. Il s'agit de l'objectif local. Au niveau de la colonie, l'objectif est de répartir les missions de façon ordonnée sur les différentes fourmis afin de minimiser une fonction d'évaluation. Pour ce faire, chaque fourmi possède sa propre marque de phéromone. Ainsi la quantité de phéromone de la fourmi  $k$  sur l'arc  $(v_i, v_j)$  au temps  $t$  est noté  $\tau_{ij}^k(t)$ . La quantité de phéromone des autres fourmis est noté  $\hat{\tau}_{ij}^k(t)$ .

$$(III.30) \quad \hat{\tau}_{ij}^k(t) = \left( \sum_{l=1, l \neq k}^m (\tau_{ij}^l(t)) \right)$$

Grâce à cette distinction des traces de phéromone, il est possible pour une fourmi d'être attirée par les chemins comportant sa phéromone, et repoussée par ceux comportant des traces de phéromone étrangère.

#### Renforcement des pistes

Lorsque toutes les missions ont été colonisées par les fourmis, la solution ainsi créée est évaluée. Une politique de marquage de type élitiste est alors réalisée, c'est-à-dire que seuls les chemins de la meilleure solution sont marqués en phéromone par les fourmis. Les chemins des autres solutions sont ainsi "oubliés" par la colonie ce qui permet de minimiser le bruit créé par ces solutions au cours de la recherche. Ainsi  $\Delta_{ij}^k(t)$  indique la quantité de phéromone déposée par la fourmi  $k$  sur l'arc  $(v_i, v_j)$  au temps  $t$  et est défini par l'équation suivante :

$$(III.31) \quad \Delta_{ij}^k(t) = \begin{cases} \frac{1}{c_{ij}^k(t)} & \text{si le score du chemin à marquer est un nouveau record} \\ 0 & \text{sinon} \end{cases}$$

#### Évaporation

Afin d'empêcher l'algorithme de rester piégé dans une solution localement optimale, les traces de phéromone sont continuellement évaporées. Cette évaporation est réalisée après chaque solution calculée. Le paramètre  $\rho$  ( $\rho \in [0; 1]$ ) permet de définir le taux de conservation de la phéromone. Quand  $\rho = 1$ , il n'y a pas d'évaporation.

La quantité de phéromone de la fourmi  $k$  présente au temps  $t$  sur un arc  $(v_i, v_j)$  est définie par l'équation III.32. Cette quantité est bornée dans l'intervalle  $[\lambda; +\infty[$  afin de garantir la présence de phéromone sur chaque arc à tout moment.

$$(III.32) \quad \tau_{ij}^k(t) = \max(\lambda, \rho \cdot \tau_{ij}^k(t-1)) + \Delta_{ij}^k(t)$$

### III.3.3.1.3 Comportement des fourmis

Une fourmi se déplace sur le graphe  $G$  grâce à la règle de transition décrite dans le paragraphe suivant. À chaque fois qu'une fourmi  $k$ , présente sur le noeud  $i$ , doit choisir une destination, les probabilités de choix des destinations possibles ( $p_{ij}^k$ ) sont calculées. Un tirage au sort proportionnel détermine alors la destination choisie.

À chaque déplacement d'une fourmi sur le graphe, le coût du chemin courant est évalué et le sommet choisi est rendu inaccessible pour les autres fourmis. On peut ainsi remarquer que c'est ce procédé de gestion des sommets à l'échelle globale qui permet d'obtenir une solution complète c'est-à-dire une solution dont les chemins contiennent toutes les missions à réaliser. En revanche, cette gestion globale est contraire au fonctionnement traditionnel des algorithmes fourmis dont la caractéristique principale repose sur le comportement totalement décentralisé des fourmis.

Lorsque plus aucun déplacement n'est possible, c'est-à-dire que toutes les villes ont été visitées, les fourmis sont replacées à leur position initiale sur le graphe et la solution est évaluée. La fonction d'évaluation du chemin global prend en compte les fonctions d'évaluation du chemin de chaque fourmi. Toutefois, les paramètres  $F_1$ ,  $F_2$  et  $F_3$  servant à guider la recherche locale, il peut être utile d'utiliser des valeurs différentes lors de l'évaluation de la solution globale. Pour cette raison, l'évaluation de l'enchaînement total des missions par les chariots cavaliers  $c(t)$  est calculée ainsi par l'équation suivante :

$$(III.33) \quad c(t) = \sum_{i=0}^n \sum_{j=0, j \neq i}^n \sum_{k=1}^m (c'_{ij}^k(t) \cdot x_{ij}^k(t))$$

Où  $c'_{ij}^k(t)$  se définit comme tel :

$$(III.34) \quad c'_{ij}^k(t) = d_{ij}^k(t) \cdot F'_1 + l_{ij}^k(t) \cdot F'_2 + e_{ij}^k(t) \cdot F'_3$$

Dans le cas où le chemin ainsi construit obtient un meilleur score que le meilleur chemin trouvé jusqu'alors, la phéromone est déposée. Sinon aucune phéromone n'est déposée par les

fourmis. L'évaporation est réalisée à chaque fois qu'une nouvelle solution a été obtenue par les fourmis.

### Règle de transition pseudo-aléatoire proportionnelle

Une fourmi choisit aléatoirement une mission à réaliser parmi les missions non-affectées selon les probabilités calculées par l'équation III.35 :

$$(III.35) \quad p_{ij}^k(t) = \frac{\left[\tau_{ij}^k(t)\right]^\alpha \cdot \left[1 + \frac{1}{c_{ij}^k(t)}\right]^\beta \cdot \left[1 + \frac{\tau_{ij}^k(t)}{\sum_{l=1}^m \tau_{ij}^l(t)}\right]^\gamma}{\sum_{s \in S_i} \left(\left[\tau_{is}^k(t)\right]^\alpha \cdot \left[1 + \frac{1}{c_{is}^k(t)}\right]^\beta \cdot \left[1 + \frac{\tau_{is}^k(t)}{\sum_{l=1}^m \tau_{is}^l(t)}\right]^\gamma\right)}$$

Cette règle de transition pseudo-aléatoire proportionnelle indique la probabilité pour qu'un voyageur  $k$  choisisse la ville  $j$  immédiatement après avoir visité la ville  $i$  au temps  $t$  (et donc parmi les successeurs  $s$  de  $i$  ( $s \in S_i$ )). Cette règle prend en compte trois facteurs :

- $\left[\tau_{ij}^k(t)\right]$  : la quantité de phéromone de la fourmi  $k$  présente sur l'arc  $(v_i, v_j)$  ;
- $\left[1 + \frac{1}{c_{ij}^k(t)}\right]$  : l'inverse du coût d'enchaînement de la visite de la ville  $j$  après la ville  $i$  pour le voyageur  $k$  au temps  $t$  ;
- $\left[1 + \frac{\tau_{ij}^k(t)}{\sum_{l=1}^m \tau_{ij}^l(t)}\right]$  : la part de phéromone de la fourmi  $k$  sur l'arc  $(v_i, v_j)$  au temps  $t$ .

Le premier terme de l'équation (quantité de phéromone de la fourmi  $k$  sur l'arc  $(v_i, v_j)$ ) permet d'indiquer la qualité de l'insertion de l'arc dans la solution construite. Plus la qualité de la solution précédemment construite est importante et plus la quantité de phéromone déposée sur les arcs sera élevée (voir équation III.31 p.128). Ce terme permet aux fourmis de communiquer les unes avec les autres. Cette stigmergie modélise également une forme de mémoire en permettant aux fourmis d'accéder à une connaissance de l'évaluation passée de la qualité d'une section de chemin. De part cette communication, les fourmis vont être en mesure de collaborer les unes avec les autres.

La fonction de coût est décrite par l'équation III.6 (voir p.109) et prend en compte la durée de déplacement des voyageurs, le retard pondéré ainsi que l'avance. C'est la partie heuristique de l'équation. Elle permet de guider la recherche vers des optimums locaux et ainsi d'accélérer la vitesse de convergence de l'algorithme. Mis à part de l'emploi de l'aléatoire ainsi que de l'utilisation de la phéromone, cette partie de la formule correspond à l'heuristique des plus proches voisins (voir III.3.2.3 p.126).

Le troisième et dernier terme :  $\left[ 1 + \frac{\tau_{ij}^k(t)}{\sum\limits_{l=1}^m \tau_{ij}^l(t)} \right]$  n'est pas présent dans la loi de transition traditionnelle d'un ACO (voir II.5 p.43). Il introduit un processus de compétition entre les fourmis. Ainsi, plus il y aura de phéromone étrangère sur un arc et plus la fourmi sera repoussée. En revanche si la part de phéromone de la fourmi est importante alors elle sera attirée par cet arc.

Les constantes  $\alpha$ ,  $\beta$ , et  $\gamma$  ( $0 \leq \alpha, \beta, \gamma < +\infty$ ) permettent de pondérer l'importance relative des trois termes de la formule. L'inverse du coût de l'enchaînement de deux missions ainsi que la part de phéromone de la fourmi sur la phéromone totale présente sur l'arc sont bornés sur  $]0; 1]$ . Leur valeur est donc augmentée afin d'être comprise sur  $[1; 2]$  afin que l'application de la puissance  $\beta$  et  $\gamma$  donne un résultat sur l'intervalle  $]1; +\infty[$ . Le dénominateur de l'équation permet ensuite d'obtenir un résultat compris dans l'intervalle  $[0; 1]$ .

### III.3.3.1.4 Algorithme général

L'algorithme principal (voir Algorithme 2) prend en paramètre la liste des véhicules ainsi que le graphe des missions à ordonner et à affecter. À chaque itération de l'algorithme, un véhicule est tiré au sort se déplace en fonction de la loi de transition III.35 jusqu'à ce que toutes les missions aient été attribuées. Ensuite, les scores des plans de charge ainsi déterminés sont additionnés pour obtenir le score total de l'ordonnancement. Si ce score est le meilleur obtenu jusqu'à présent, il est sauvegardé et les fourmis déposent de la phéromone sur leurs chemins respectifs. Enfin la phéromone est évaporée et le processus est réitéré pendant  $\Theta$  itérations.

### III.3.3.1.5 Gestion de la dynamique

Cette méthode de résolution du problème est dite hors-ligne car elle doit être exécutée à chaque fois qu'une caractéristique du problème change. Chaque événement survenant sur le terminal modifie les données du problème et une solution calculée au temps  $t$  peut ne plus être valable au temps  $t'$ . Ainsi, les entrées de l'algorithme sont modifiés à chaque nouvelle exécution.

En revanche, même si les coûts des arcs ou les sommets du graphe, ou encore le nombre ou les caractéristiques des véhicules changent, les traces de phéromone présentes sur les arcs du graphe sont conservées. Cette conservation permettra de guider les fourmis dans leur nouvelle recherche de solution. En effet, malgré les modifications des caractéristiques du problème, une partie de la solution précédemment calculée peut rester valide.

C'est ce mécanisme de conservation de phéromone qui rend la météahéuristicque fourmi

---

**Algorithme 2 :** méthode de résolution dynamique hors-ligne

---

Résultat :

Plan de charge des machines

Données :

M : liste des véhicules,

V : sommets du graphe,

A : arcs du graphe

1 **début**

2     *meilleurScore*  $\leftarrow +\infty$ ;

3     *meilleurChemin*  $\leftarrow \emptyset$ ;

4     *etape*  $\leftarrow 1$ ;

5     **tant que** *etape*  $\leq \Theta$  **faire**

6         *P*  $\leftarrow N$ ;

7         *chemins*  $\leftarrow \emptyset$ ;

8         **pour chaque** *fourmi k de M faire*

9             *initialiserPosition()*;

10             *chemin<sup>k</sup>*  $\leftarrow \emptyset$ ;

11             *chemins*  $\leftarrow \text{chemin}^k$ ;

12         **fin**

13         **tant que** *P*  $\neq \emptyset$  **faire**

14             *k*  $\leftarrow alea(M)$ ;

15             *j*  $\leftarrow choisirDestination(k, P)$ ;

16             *chemin<sup>k</sup>*  $\leftarrow j$ ;

17             *P*  $\leftarrow P - j$ ;

18         **fin**

19         *score*  $\leftarrow sommeScore(chemins)$ ;

20         **si** *score*  $< meilleurScore$  **alors**

21             *meilleurScore*  $\leftarrow score$ ;

22             *deposerPheromone(chemins)*;

23         **fin**

24         *evaporation(N)*;

25         *etape*  $\leftarrow etape + 1$ ;

26         **fin**

27     **retourner** *meilleurChemin*;

28 **fin**

adaptée à la résolution de problèmes dynamiques.

### III.3.3.1.6 Paramètres

Le défaut majeur de la météuristiche fourmi réside dans le nombre important de paramètre ainsi que dans l'incidence de leur valeur vis-à-vis de la qualité de la solution trouvée et de sa vitesse d'obtention. La méthode de résolution hors-ligne utilise les paramètres suivants :

- $\alpha$  : importance relative de la trace de phéromone lors du choix de la destination ;
- $\beta$  : importance relative de l'heuristique de pondération lors du choix de la destination ;
- $\gamma$  : importance relative du processus de répulsion lors du choix de la destination ;
- $\lambda$  : quantité minimale de phéromone présente sur chaque arc ;
- $\rho$  : taux de conservation de la phéromone après chaque itération de l'algorithme ;
- $F_1$  : importance relative de la durée des trajets dans l'évaluation de l'enchaînement de deux missions ;
- $F_2$  : importance relative du retard pondéré dans l'évaluation de l'enchaînement de deux missions ;
- $F_3$  : importance relative de l'avance dans l'évaluation de l'enchaînement de deux missions ;
- $F'_1$  : importance relative de la durée des trajets dans l'évaluation globale d'une solution ;
- $F'_2$  : importance relative du retard pondéré dans l'évaluation globale d'une solution ;
- $F'_3$  : importance relative de l'avance dans l'évaluation globale d'une solution ;
- $\Theta$  : nombre d'itérations de la boucle principale de l'algorithme à effectuer à chaque exécution.

La valeur idéale pour chacun de ses paramètres se détermine de manière empirique. Néanmoins, il est possible de prédéfinir des valeurs initiales en prenant en compte les relations mathématiques entre ces paramètres.

Le paramètre  $\Theta$  permet de contrôler le rapport entre la qualité de la solution et la durée du calcul. Un  $\Theta$  important conduira à de meilleures solutions mais également à des calculs plus longs. Il peut-être nécessaire de réduire la valeur de  $\Theta$  dans le cas où la dynamique du système est trop importante et provoque des modifications sur les caractéristiques du problème avant que l'algorithme ait le temps de proposer une solution. Pour cela il peut être utile de connaître le temps de calcul d'une itération et d'utiliser les connaissances sur la dynamique du système (statistiques, degré effectif de dynamique, etc.) afin de déterminer le nombre d'itérations optimal. Ce temps de calcul d'une itération dépendra, mis à part des performances de la machine utilisée

pour le calcul, de la taille du problème, c'est-à-dire du nombre de missions à ordonner ainsi que du nombre de véhicules à utiliser.

Le calcul du temps moyen de trajet par mission permet d'initialiser les paramètres  $F_1$ ,  $F_2$  et  $F_3$ . En effet, ces paramètres contrôlant une importance relative, il est nécessaire d'évaluer les parts de retard et d'avance tolérées par mission afin de les initialiser. Par exemple, si la durée moyenne du déplacement d'une mission est de 2 minutes et qu'on tolère 30 seconde de retard et 6 minutes d'avance par mission, on pourra définir :

- $F_1 = 4$  (2 minutes =  $4 * 30$  secondes)
- $F_2 = 1$
- $F_3 = 12$  (6 minutes =  $12 * 30$  secondes)

Ces paramètres étant pris en compte dans l'évaluation heuristique du poids d'un arc, ils sont utilisés dans la règle de transition proportionnelle et cette valeur est pondérée par le paramètre  $\beta$ . Une mission consistant à déplacer un conteneur, le temps de réalisation ne peut être nul. Dans l'hypothèse d'une discréétisation à la seconde, la valeur de la partie droite du deuxième terme de la loi de transition est comprise entre 0 et 1. Puis la valeur est incrémentée afin d'appartenir à l'intervalle  $[1; 2]$ . Mis à la puissance  $\beta$ , le résultat est défini sur  $[1; +\infty]$ .

La partie droite du dernier terme de la règle de transition étant un ratio, il est compris entre 0 et 1. Là encore, la valeur est incrémentée pour être définie sur  $[1; 2]$ , puis sur  $[1; +\infty]$  après la mise à la puissance  $\gamma$ .

Le paramètre  $\lambda$  doit être défini sur  $[1; +\infty]$  afin d'assurer que le premier terme de la règle de transition puisse être défini sur  $[1; \infty[$ .

Le paramètre  $\rho$  représente la part de phéromone conservée après chaque évaporation et se définit sur  $[0; 1]$ . Pour une même valeur de  $\rho$ , plus la trace de phéromone présente sur un arc sera importante et plus la quantité évaporée sera grande. L'utilisation d'un fort taux d'évaporation combiné à une faible présence de phéromone conduira à l'inefficacité du système de communication entre les fourmis. Au contraire, une évaporation faible alliée à une présence forte de phéromone provoquera l'inefficacité du processus d'évaporation et les arcs seront saturés de phéromones. La conséquence sera que les anciennes meilleures solutions, devenues invalides avec le temps, continueront d'attirer les fourmis, au détriment de nouvelles solutions potentielles.

Toutes ces valeurs conditionnent grandement l'efficacité de l'algorithme et leur définition requiert de nombreux ajustement du fait des interactions entre les paramètres. Ils constituent le point faible de la métaheuristique fourmi en général, et ici de la méthode fourmi appliquée au problème d'ordonnancement et d'affectation.

### **III.3.3.2 Variante de la méthode de résolution hors-ligne à colonies de fourmis**

#### **III.3.3.2.1 Algorithme**

Une seconde approche hors-ligne utilisant une modélisation du problème sous forme de problème de voyageurs de commerce consiste à considérer une seule colonie de fourmis capables de déposer des phéromones de différentes couleurs. Tout comme pour le modèle précédent, une couleur représente un véhicule. Dans cette variante, chaque fourmi construit une solution globale au problème, c'est-à-dire un chemin contenant toutes les missions du problème. Le graphe des missions est construit de la même manière que pour la méthode de résolution hors-ligne de base. La seule différence concerne le noeud représentant le dépôt qui est déconnecté du reste du graphe lorsque tous les véhicules disponibles ont été utilisés pour construire la solution.

En effet, cette méthode repose sur le principe suivant : au début de l'algorithme, un véhicule est choisi parmi la liste des véhicules disponibles<sup>2</sup>. Puis, chaque choix de destinations sur le graphe des missions permet de construire la solution de ce véhicule jusqu'à ce que la destination soit le noeud du dépôt. Dès lors, un autre véhicule est choisi et l'algorithme se poursuit tant qu'il reste des missions à planifier. Dès que tous les véhicules ont été utilisés, le noeud de dépôt est déconnecté du reste du graphe et l'algorithme se déroule jusqu'à ce que toutes les missions aient été planifiées.

Lorsque toutes les missions sont planifiées, la solution globale obtenue est évaluée et conservée si elle obtient le meilleur score, c'est-à-dire la fitness la plus faible. Le processus d'évaporation se déroule alors sur le graphe des missions. Enfin, chaque fourmi dépose de la phéromone colorée sur les arcs du graphe de missions faisant partie de sa solution. Une couleur est utilisée par véhicule.

Ce procédé est décrit dans l'algorithme 3 p.136. La meilleure solution obtenue constitue le plan de charge qui sera affecté aux chariots cavaliers.

---

2. Ce choix peut très bien être aléatoire ou au contraire défini selon une règle déterminée.

---

### Algorithme 3 : Seconde méthode de résolution dynamique hors-ligne

---

Résultat :

Plan de charge des machines

Données :

M : liste des véhicules,

V : sommets du graphe,

A : arcs du graphe,

K : colonie de fourmis

```
1 début
2   meilleurScore ← +∞;
3   meilleurChemin ← {∅};
4   etape ← 1;
5   tant que etape ≤ Θ faire
6     meilleurScoreetape ← +∞;
7     meilleurCheminetape ← {∅};
8     connecterDepot(V,A);
9     initialiserVehicules(M);
10    pour chaque fourmi k de K faire
11      chemink ← {∅};
12      m ← choisirVehicule(M);
13      M ← M \ {m};
14      si M = {∅} alors
15        V ← V \ {depot};
16      fin
17      tant que V ≠ {∅} faire
18        vdest ← choisirMission(V);
19        si vdest = depot alors
20          chemink ← chemink ∪ cheminkm;
21          m ← choisirVehicule(M);
22          M ← M \ {m};
23          si M = {∅} alors
24            V ← V \ {depot};
25          fin
26        fin
27        deposerPheromone(v,vdest);
28        cheminkm ← cheminkm ∪ vdest;
29        v ← vdest;
30        V ← V \ {v};
31      fin
32      cheminkm ← cheminkm ∪ {depot};
33      cheminsk ← cheminsk ∪ cheminkm;
34      si score(cheminsk) > meilleurScoreetape alors
35        meilleurScoreetape ← score(cheminsk);
36        meilleurCheminetape ← cheminsk;
37      fin
38    fin
39    pour chaque fourmi k de K faire
40      deposerPheromone(cheminsk);
41    fin
42    si meilleurScore > meilleurScoreetape alors
43      meilleurScore ← meilleurScoreetape;
44      meilleurChemin ← meilleurCheminetape;
45    fin
46    evaporation();
47  fin
48  retourner meilleurChemin;
49 fin
```

---

### III.3.3.2.2 Paramétrage

Le paramétrage de cette variante est strictement identique à celui de la méthode initiale de résolution hors-ligne. En effet, la méta-heuristique reste la même, un processus de renforcement des pistes de phéromones utilisant une coloration afin d'implémenter un mécanisme de collaboration/compétition entre les chariots cavaliers. Les paramètres sont donc les suivants :

- $\alpha$  : importance relative de la trace de phéromone lors du choix de la destination ;
- $\beta$  : importance relative de l'heuristique de pondération lors du choix de la destination ;
- $\gamma$  : importance relative du processus de répulsion lors du choix de la destination ;
- $\lambda$  : quantité minimale de phéromone présente sur chaque arc ;
- $\rho$  : taux de conservation de la phéromone après chaque itération de l'algorithme ;
- $F_1$  : importance relative de la durée des trajets dans l'évaluation de l'enchaînement de deux missions ;
- $F_2$  : importance relative du retard pondéré dans l'évaluation de l'enchaînement de deux missions ;
- $F_3$  : importance relative de l'avance dans l'évaluation de l'enchaînement de deux missions ;
- $F'_1$  : importance relative de la durée des trajets dans l'évaluation globale d'une solution ;
- $F'_2$  : importance relative du retard pondéré dans l'évaluation globale d'une solution ;
- $F'_3$  : importance relative de l'avance dans l'évaluation globale d'une solution ;
- $\Theta$  : nombre d'itérations de la boucle principale de l'algorithme à effectuer à chaque exécution.

### III.3.3.3 Méthode de résolution en-ligne à colonies de fourmis

Cette méthode se rapproche de la méthode décrite dans le paragraphe précédent. Il est également question de métaheuristique fourmi ainsi que de phéromone colorée. Ici il y aura autant de colonies de fourmis qu'il y a de voyageurs dans le problème des voyageurs de commerce connexe. Ainsi, une colonie est modélisée par une couleur et chaque fourmi de la colonie dépose de la phéromone de la couleur de la colonie. Les fourmis d'une colonie sont attirées par les pistes contenant de la phéromone de leur couleur, et repoussées par les pistes contenant de la phéromone des autres couleurs.

Cet algorithme a été élaboré dans le but de proposer rapidement une solution au problème, peu importe son degré de dynamité. Pour ce faire, il est nécessaire d'intégrer cette dynamité dans le modèle même de la méthode de résolution, afin d'éviter de recalculer complètement une solution à partir de zéro.

La méthode hors-ligne décrite dans le paragraphe précédent étant déjà rapide et utilisant également la métaheuristique fourmi, la seule possibilité d'accélération de la résolution du problème réside dans la structure du graphe sur lequel évoluent les fourmis. Il ne sera pas question ici de modéliser toutes les solutions possibles. Le graphe ne sera donc pas complet et seules les solutions dont le coût est “raisonnable” seront envisagées. On parlera de tâches compatibles pour indiquer qu'une tâche peut être réalisée après une autre par la même ressource. Deux tâches seront ainsi compatibles si les fenêtres de temps permettent théoriquement de réaliser les deux missions sans dépassement.

### **III.3.3.3.1 Description générale du graphe**

Définissons le graphe dynamique et orienté  $G^{(t)} = (V, A)$  où  $V$  est l'ensemble des noeuds et  $A$  l'ensemble des arcs. Lors de l'initialisation, chaque tâche à ordonner (ou ville à visiter selon la modélisation) est modélisée par un noeud  $v_i \in V$  et les arcs  $a_{ij} = (v_i, v_j) \in A$  représentent la possibilité pour les machines (ou les voyageurs de commerce) d'enchaîner la tâche  $J_j$  (ou la ville  $v_j$ ) après  $J_i$  ( $v_i$ ) sans dépasser les fenêtres de temps. Ainsi, un arc est ajouté entre deux noeuds  $v_i$  et  $v_j$  si pour au moins une ressource (voyageur)  $k$ ,  $TW_{i,2}^{\max} + setup_{ij}^k < TW_{j,1}^{\min}$ . Au cours du processus de résolution, dès qu'une mission est réalisée par un chariot cavalier, le sommet correspondant est supprimé du graphe et la structure du graphe est corrigée comme décrit dans les paragraphes suivants afin de maintenir les propriétés du modèle.

En plus des tâches à planifier, deux sommets sont ajoutés au graphe : le noeud de source :  $v_{source}$ , et le noeud de fin :  $v_{fin}$ . Le noeud source est relié à chaque noeud du graphe dont le degré entrant est nul. Le degré entrant de  $v_{source}$  est également nul. Les sommets du graphe dont le degré sortant est nul sont connectés au noeud  $v_{fin}$ , qui possède quant à lui uniquement des arcs entrant.

### **III.3.3.3.2 Colonies et véhicules**

Dans ce modèle, une colonie de fourmis représente un véhicule (machine ou voyageur selon la modélisation). Le véhicule doit choisir la meilleure séquence de missions à réaliser, c'est-à-dire celle qui minimise à la fois la distance parcourue, le retard pondéré et l'avance. Ainsi, les séquences de missions pour différents véhicules sont modélisées par des chemins distincts sur le graphe.

Les fourmis d'une colonie doivent coloniser le graphe des missions de façon à trouver des chemins qui minimisent la valeur de la fonction d'évaluation. Quand une fourmi atteint un som-

met (correspondant à une mission), elle dépose de la phéromone sur celui-ci. Cette phéromone sera utilisée pour guider les autres fourmis de la colonie vers le noeud et pour repousser les fourmis des autres colonies vers d'autres noeuds. Comme chaque colonie agit de la même façon, les sommets sont répartis entre les différentes colonies et cette distribution tend à minimiser la fonction d'évaluation globale.

Chaque colonie est modélisée par une couleur. De cette façon, chaque noeud du graphe est coloré par la couleur de la colonie dont la phéromone est la plus présente sur ce sommet. La solution globale est obtenue en construisant les meilleurs chemins pour chaque couleur. Chaque chemin  $P^k$  de couleur  $k$  est construit en partant de  $v_{source}$ . Lors de la construction du chemin, à partir d'un noeud  $v_i$  de couleur  $k$ , le prochain sommet est choisi parmi les successeurs de  $v_i$  de couleur  $k$  :  $S_i^k$ . Le noeud choisi est celui comportant la plus grande quantité de phéromone de couleur  $k$ . Le processus est répété jusqu'à ce que  $v_{fin}$  ait été atteint ou que  $S_i^k = \emptyset$ .

### III.3.3.3.3 Pondération des arcs

Un arc  $a_{ij}$  relie le sommet  $v_i$  à  $v_j$  et correspond à l'enchaînement de la mission  $j$  après la réalisation de la mission  $i$ . Cet arc est pondéré par le coût de l'enchaînement des deux missions sur la même machine. Ce poids doit prendre en compte à la fois la distance parcourue par le véhicule ainsi que le retard pondéré et l'avance engendrés par l'enchaînement des missions.

Les critères de la fonction d'évaluation sont les mêmes que pour le modèle hors-ligne de résolution dynamique. La distance correspond au temps de parcours entre le point de livraison du conteneur de la mission  $i$  et le point de collecte du conteneur de la mission  $j$  et est définie par l'équation III.3 (voir p. 109). Le retard pondéré et l'avance sont décrits par les équations III.8 et III.5 (voir p. 109, et p. 109). Les variables  $F_1$ ,  $F_2$  et  $F_3$  permettent de pondérer les critères les uns vis-à-vis des autres dans la fonction locale d'évaluation (voir eq. III.6 p. 109).

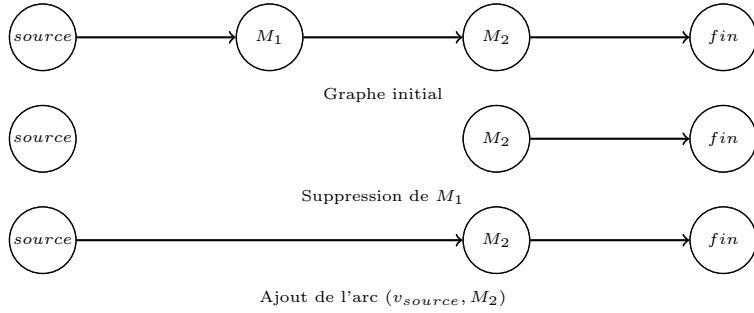
La flotte de véhicules pouvant être hétérogène, chaque arc devra contenir autant de poids qu'il y a de machines (voyageurs) dans le problème puisque les temps de parcours diffèrent d'un véhicule à l'autre si leur vitesse est différente.

Pour calculer les poids d'un arc, il est nécessaire de distinguer deux cas selon les extrémités de cet arc :

- $v_j \neq v_{fin}$  : le poids de l'arc est donnée par l'équation précédente.
- $v_j = v_{fin}$  : le poids correspond uniquement au temps de parcours entre le point de livraison de la mission  $j$  et le dépôt à la date de livraison prévue de la mission  $j$  ;

### III.3.3.3.4 Maintient de la structure du graphe

Les sommets  $v_{source}$  et  $v_{fin}$  étant respectivement reliés uniquement aux sommets sans prédécesseur et sans successeur, il est nécessaire de modifier le graphe pour maintenir ces propriétés lors de l'ajout et de la suppression de sommets. Dans le cas décrit par la figure III.3 où un sommet  $v_i$  est l'unique prédécesseur de  $v_j$  et où  $v_{source}$  est relié à  $v_i$  et  $v_j$  est connecté à  $v_{fin}$ , lorsque la mission modélisée par le sommet  $v_i$  sera exécutée par un véhicule, le sommet  $v_i$  sera supprimé du graphe et un arc reliant  $v_{source}$  à  $v_j$  sera ajouté. Dans la même situation où  $v_j$  est le seul successeur de  $v_i$ , lorsque  $v_j$  est réalisée par un véhicule,  $v_j$  est supprimé du graphe et un arc reliant  $v_i$  au sommet  $v_{fin}$  est ajouté au graphe. Lorsqu'un sommet  $v_l$  est ajouté sur le graphe entre  $v_{source}$  et  $v_i$ ,  $v_i$  possède alors deux successeurs  $v_{source}$  et  $v_l$ . L'arc  $(v_{source}, v_i)$  est alors supprimé et un arc est ajouté entre  $v_{source}$  et  $v_l$ . De la même façon, si  $v_l$  devient le successeur de  $v_j$  et que  $v_j$  est le seul prédécesseur de  $v_l$ , alors l'arc  $(v_j, v_{fin})$  est supprimé et un arc est ajouté entre  $v_l$  et  $v_{fin}$ .

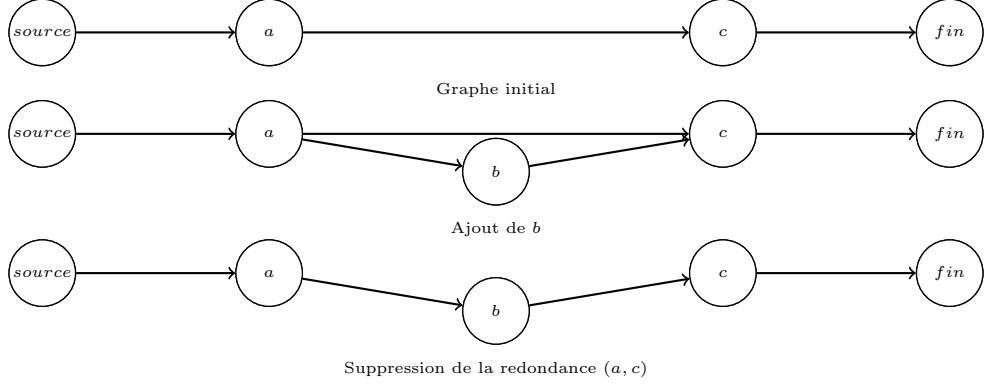


**Figure III.3:** Suppression du sommet  $M_1$  : ajout d'un arc entre les sommets  $source$  et  $M_2$

Modélisé par un graphe complet, l'espace de recherche est immense. Un tel graphe permet de représenter toutes les solutions possibles au problème alors qu'une grande partie de ces solutions sont de très mauvaise qualité. Les informations apportées par les fenêtres de temps permettent pourtant d'explorer uniquement les solutions les plus performantes. Ainsi, il est possible de réduire la taille de l'espace de recherche en analysant le voisinage des nœuds dans le graphe. En effet, lorsqu'un nœud est ajouté au graphe, il est possible de prendre en compte la notion de précédence entre les missions pour éviter la redondance des arcs dans le graphe.

Dans le cas décrit par la figure III.4, où les arcs  $(a, b)$ ,  $(a, c)$  et  $(b, c)$  se trouvent sur le graphe après l'ajout du sommet  $b$ , l'arc  $(a, c)$  sera supprimé afin d'assurer que le véhicule réalisant la mission  $a$  réalise ensuite la mission  $b$  avant d'effectuer  $c$ . Sans cette suppression, le cas où  $c$  serait réalisé avant  $b$  serait envisagé. En effet, chaque mission étant supprimée du graphe lorsqu'elle est réalisée, la mission  $b$  se trouverait réalisable après l'exécution de  $a$  puis de  $c$ . Or, les fenêtres

de temps permettent à un même véhicule de réaliser  $a$ , puis  $b$ , puis  $c$  sans dépassement mais pas  $a$  puis  $c$  puis  $b$ .



**Figure III.4:** Suppression des arcs redondants

### III.3.3.3.5 Gestion de la dynamique

Dans la version dynamique du problème, les missions peuvent être insérées, annulées ou modifiées et le graphe doit être en mesure d'évoluer en fonction de ces modifications. De plus, la flotte de véhicules pouvant également être concerné par des événements dynamique (pannes, trafic, comportement humain...), les informations doivent être mises-à-jour sur le graphe afin de modéliser efficacement les colonies correspondantes.

Lorsqu'une nouvelle mission  $i$  est insérée dans l'ordonnanceur, un noeud  $v_i$  est ajouté au graphe. De nouveaux arcs reliant  $v_i$  aux autres sommets du graphe sont créés en respectant la structure décrite dans le paragraphe précédent. Si l'insertion de  $v_i$  dans le graphe conduit à la création de deux arcs  $(v_{source}, v_i)$  et  $(v_i, v_j)$  et si l'arc  $(v_{source}, v_j)$  existe déjà, alors ce dernier doit être supprimé. Le procédé est le même concernant le noeud de fin. Cette procédure est nécessaire pour obliger les véhicules à effectuer chaque mission. Dans le cas contraire, la meilleure solution trouvée par l'algorithme consisterait à n'effectuer aucune mission car ainsi la fonction d'évaluation serait nulle et donc minimale.

Lorsqu'une mission est annulée ou terminée, le sommet correspondant doit être retiré du graphe. Ses arcs incidents sont alors également supprimés, mais il est possible que de nouveaux arcs doivent être ajoutés afin de connecter les prédecesseurs du noeud supprimé à ses successeurs, en suivant les règles décrites précédemment.

Quand une mission est mise-à-jour, le sommet correspondant ainsi que ses arcs incidents sont supprimés du graphe. Puis le noeud est inséré de nouveau et les nouveaux arcs créés prennent ainsi en compte les nouvelles caractéristiques de la mission.

Quand un véhicule commence à réaliser une mission, il doit la mener à son terme à moins de tomber en panne. Dans ce dernier cas, la mission doit être mise-à-jour car le point de collecte du conteneur peut avoir changé si le véhicule avait commencé à déplacer le conteneur avant de tomber en panne. Si le véhicule devient indisponible, les fourmis de la colonie correspondante sont réinitialisées au sommet source et doivent y rester tant que le véhicule ne redevient pas disponible. Dans le même temps, le processus d'évaporation aura fait disparaître progressivement la phéromone de cette colonie permettant ainsi aux autres colonies de coloniser les missions concernées.

Si le véhicule ne tombe pas en panne, il doit achever sa mission. Pour représenter cette contrainte les fourmis de la colonie du véhicule débutent leur parcourt depuis le noeud de la mission courante. La phéromone des autres fourmis sur ce noeud est évaporée afin de le rendre inaccessible aux autres colonies.

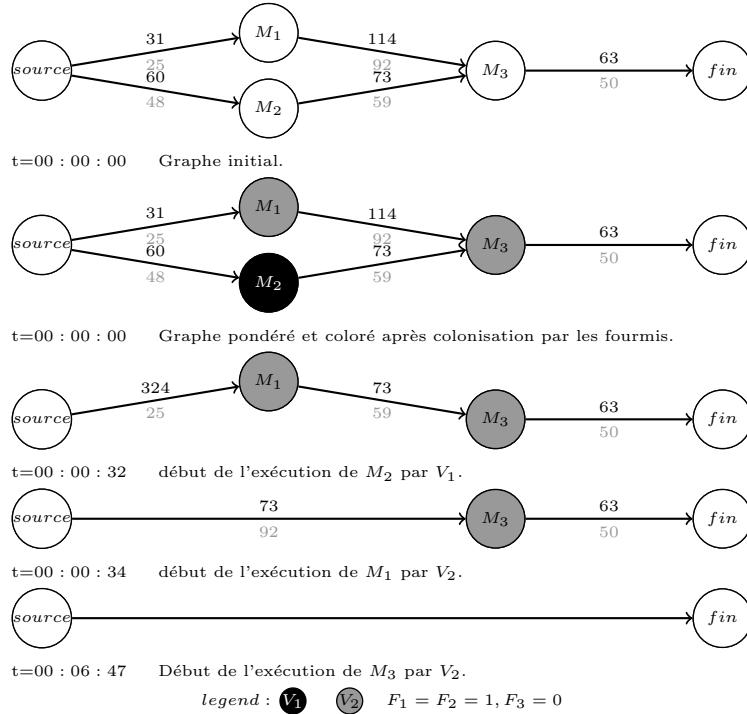
### **III.3.3.3.6 Gestion de la phéromone**

La phéromone permet d'indiquer aux individus quels ont été les choix réalisés par d'autres fourmis et également quel fut la qualité du chemin construit en suivant ces choix. L'utilisation de phéromone colorée permet de répartir les individus sur le graphe des missions et ainsi de distribuer les tâches aux différentes ressources. Un système de dépôt de phéromone sur les arcs du graphe permet de retracer le chemin complet d'une fourmi du noeud source au noeud fin. Toutefois, un tel système ne permet pas de faire fonctionner le mécanisme de compétition entre les colonies. En effet, un sommet peu être atteint par une fourmi à partir de chacun de ses arcs entrants. Il existe ainsi plusieurs chemins menant à un sommet. Des fourmis de colonies différentes peuvent ainsi accéder à un sommet par des chemin différents sans être repoussées par la phéromone étrangère.

Le système utilisé ici consiste à déposer de la phéromone sur les sommets du graphe au lieu des arcs. Ainsi, lorsqu'un sommet est atteint par une fourmi, les différentes phéromones sont prises en compte, peu importe la provenance de la fourmi. Ce procédé à en revanche l'inconvénient d'empêcher de retracer le chemin complet d'une fourmi du noeud source au noeud fin, c'est pourquoi les missions ne peuvent pas être affectées en prenant en compte les chemins dans le graphe. Chaque affectation sera donc uniquement déterminée par la couleur dominante de la phéromone sur le sommet correspondant à la mission à affecter.

Par exemple, le graphe de la figure III.5 reprend les données de l'exemple de la figure III.2 et montre la coloration du graphe par les deux colonies de fourmis. Ici, la phéromone de la

colonie du véhicule  $V_1$  est majoritaire sur le sommet  $M_2$  alors que la phéromone de la colonie du véhicule  $V_2$  est majoritaire sur les noeuds  $M_1$  et  $M_3$ . La figure indique également l'évolution du graphe dans le temps. À l'initialisation la structure du graphe indique bien que  $M_1$  et  $M_2$  ne peuvent pas être enchaînées par un même véhicule. La colonisation du graphe par les fourmis permet de distribuer les missions aux chariots cavaliers qui ensuite les exécutent. Ainsi, d'abord  $M_2$  est réalisée par  $V_2$ , puis parallèlement  $M_1$  est exécutée par  $V_1$ . Enfin,  $V_2$  effectue la mission  $M_3$ .



**Figure III.5:** Graphe dynamique de l'exemple de la figure III.2 (voir p.115)

Hormis l'emplacement de dépôt de la phéromone, le fonctionnement reste identique à celui de l'algorithme hors-ligne. Ainsi il y a deux mécanismes : l'un de renforcement (*positive feedback*) et l'autre d'évaporation (*negative feedback*). L'équation III.36 décrit la quantité de phéromone déposée par la fourmi  $k$  sur le sommet  $j$  en venant de  $i$  au temps  $t$  et l'équation III.37 indique la quantité de phéromone de la couleur de la fourmi  $k$  qui sera présente sur le sommet  $i$  au temps  $t$ .

$$(III.36) \quad \Delta_{ij}^k(t) = \frac{1}{c_{ij}^k(t)}$$

$$(III.37) \quad \tau_j^k(t) = \max \left( \lambda, \rho \cdot \tau_j^k(t-1) \right) + \sum_{i \in P_j} \left( \Delta_{ij}^k(t) \right)$$

La quantité de phéromone présente sur un sommet  $i$  au temps  $t$  de couleur différente de celle de la colonie  $k$  est décrite par l'équation suivante :

$$(III.38) \quad \hat{\tau}_i^k(t) = \left( \sum_{l=1, l \neq k}^m (\tau_i^l(t)) \right)$$

### III.3.3.3.7 Règle de transition pseudo-aléatoire proportionnelle

La règle de transition reste identique à celle utilisée par l'algorithme hors-ligne, mis à part l'emplacement de la phéromone qui se trouve dorénavant sur les sommets. Une fourmi choisit donc aléatoirement une mission à réaliser parmi les missions non-affectées selon les probabilités calculées par l'équation III.39.

$$(III.39) \quad p_{ij}^k(t) = \frac{\left[ \tau_j^k(t) \right]^\alpha \cdot \left[ 1 + \frac{1}{c_{ij}^k(t)} \right]^\beta \cdot \left[ 1 + \frac{\tau_j^k(t)}{\sum_{l=1}^m \tau_j^l(t)} \right]^\gamma}{\sum_{s \in S_i} \left( \left[ \tau_s^k(t) \right]^\alpha \cdot \left[ 1 + \frac{1}{c_{is}^k(t)} \right]^\beta \cdot \left[ 1 + \frac{\tau_s^k(t)}{\sum_{l=1}^m \tau_s^l(t)} \right]^\gamma \right)}$$

Afin d'empêcher la colonisation continue du graphe par les colonies il est nécessaire de déterminer un seuil d'acceptation de la transition. En effet, si il existe dans le problème plus de véhicules que de missions à affecter, alors il est impossible d'attribuer une mission à chaque véhicule. Ce seuil est fixé par la constante  $\Delta$ . Si la probabilité de la destination choisie est inférieure à  $\Delta$ , alors la fourmi est replacée sur le nœud source.

### III.3.3.3.8 Solution *anytime*

À n'importe quel moment les plans de charge calculés par l'algorithme peuvent-être obtenus. Ainsi la solution est constituée de  $m$  chemins dans le graphe de différentes couleurs. Le graphe des missions étant orienté et acyclique, chaque chemin peut-être reconstruit en partant du noeud source et en choisissant parmi les sommets accessibles colorés par la même couleur que le chemin construit, celui qui possède la plus grande quantité de phéromone. La construction est complète lorsque le noeud fin est atteint ou lorsqu'aucun sommet de la couleur du chemin n'est accessible.

### III.3.3.3.9 Renforcement du chemin courant

Lorsqu'un véhicule commence à réaliser une mission d'un chemin, un processus de renforcement de la piste de phéromone de la couleur de la colonie du véhicule intervient sur la totalité du chemin. La trace de phéromone de cette couleur est augmentée afin d'empêcher que la couleur des sommets change trop rapidement pendant que la solution précédemment calculée est réalisée. Les sommets concernés deviennent moins attractifs pour les autres colonies. La constante  $\Lambda$  définit ainsi la quantité de renforcement et le processus suit l'algorithme 4.

---

#### Algorithme 4 : Renforcement du chemin d'une solution en cours de réalisation

---

```
1 début
2   |   pour chaque sommet  $i$  du chemin faire
3   |   |    $\tau_i^k(t) = \tau_i^k(t - 1) + \Lambda$ 
4   |   fin
5 fin
```

---

### III.3.3.3.10 Algorithme général

L'algorithme principal (voir Algorithme 5) prend en paramètre la liste des véhicules ainsi et que le graphe des missions à ordonner et à affecter. À chaque itération de l'algorithme, chaque fourmi de chaque colonie choisie une destination parmi les sommets voisins de sa position courante sur le graphe. Elle retourne au nœud source si aucun voisin n'est accessible, ou dans le cas contraire se rend sur le sommet choisi et y dépose de la phéromone. Après que chaque fourmi de chaque colonie se soit déplacée, le processus d'évaporation est effectué et les solutions

sont mises-à-jour.

---

**Algorithme 5 :** méthode de résolution dynamique en-ligne

---

Données :

M : liste des véhicules,

V : sommets du graphe,

A : arcs du graphe

1 **début**

```
2   pour chaque fourmi f de chaque colonie k faire
3     destination ← choisir_destination(f);
4     position_courante ← position(f);
5     si destination =  $\emptyset$  alors
6       retourner_au_noeud_source(f);
7     sinon
8       déplacement(f,destination);
9       déposer_pheromone(f,position_courante,destination);
10    fin
11   fin
12  pour chaque sommet v  $\in$  V faire
13    évaporation(v);
14  fin
15  pour chaque colonie k faire
16    mise_a_jour_chemin(k);
17  fin
18 fin
```

---

### III.3.3.3.11 Paramètres

L'algorithme de résolution en-ligne reprend tous les paramètres de la méthode hors-ligne à l'exception de  $\Theta$  qui était utilisé dans l'autre méthode pour définir le nombre d'itérations à réaliser. Ici, il n'est plus question de limiter le temps de calcul mais bien de calculer en continu les solutions. Trois autres paramètres sont nécessaires au fonctionnement de cette méthode. D'abord  $\eta$  qui définit le nombre d'individus à utiliser par colonie. Ensuite  $\Delta$  qui modélise la pression de l'environnement en forçant la fourmi à mourir si la part de sa phéromone présente sur le noeud choisi est trop faible. Enfin,  $\Lambda$  sert à définir la quantité de phéromone déposée sur

un chemin lorsqu'une mission de celui-ci est commencée par un véhicule.

On peut distinguer, dans la liste ci-dessous, deux classes de paramètres : d'une part ceux qui concernent le choix de la destination pour la fourmi ( $\alpha$ ,  $\beta$ ,  $\gamma$ ,  $\Delta$ ,  $F_1$ ,  $F_2$  et  $F_3$ ), et d'autre part ceux qui régissent la gestion de la phéromone ( $\lambda$ ,  $\Lambda$  et  $\rho$ ).

- $\alpha$  : importance relative de la trace de phéromone lors du choix de la destination ;
- $\beta$  : importance relative de l'heuristique de pondération lors du choix de la destination ;
- $\gamma$  : importance relative du processus de répulsion lors du choix de la destination ;
- $\eta$  : nombre de fourmis par colonie ;
- $\Delta$  : pression de l'environnement ;
- $\lambda$  : quantité minimale de phéromone présente sur chaque arc ;
- $\Lambda$  : quantité de phéromone de renforcement de chemin ;
- $\rho$  : taux de conservation de la phéromone après chaque itération de l'algorithme ;
- $F_1$  : importance relative de la durée des trajets dans l'évaluation de l'enchaînement de deux missions ;
- $F_2$  : importance relative du retard pondéré dans l'évaluation de l'enchaînement de deux missions ;
- $F_3$  : importance relative de l'avance dans l'évaluation de l'enchaînement de deux missions ;
- $F'_1$  : importance relative de la durée des trajets dans l'évaluation globale d'une solution ;
- $F'_2$  : importance relative du retard pondéré dans l'évaluation globale d'une solution ;
- $F'_3$  : importance relative de l'avance dans l'évaluation globale d'une solution.

## Conclusions

Dans ce chapitre nous avons décrit les différentes modélisations ainsi que plusieurs méthodes de résolution développées au cours de cette thèse afin de résoudre le problème d'ordonnancement et d'affectation des déplacements de conteneurs aux chariots cavaliers.

Le problème correspond à un problème de tournées de véhicules et plus précisément au problème des multiples voyageurs de commerce, et peut également être modélisé sous forme de problème d'atelier. La notation de Graham (voir II.4.1.1, p. 86) décrivant ce problème est la suivante :  $J|ST_{sd}, R_{sd}| \sum w_j \cdot T_j, \sum distance(i)$ .

Malgré la complexité du problème, une méthode de résolution exacte a été élaborée et repose sur le principe de l'évaluation séparation (*Branch-and-Bound*). Cette méthode a été parallélisée

et permet de résoudre de façon exacte de très petites instances du problème (3 véhicules et 10 missions).

Des méthodes approchées ont été mises au point afin de résoudre des instances plus importantes du problème (20 véhicules et plus de 100 missions). Ces méthodes sont basées sur des heuristiques ou des méta-heuristiques.

Ainsi les heuristiques de répartition de charge, de l'aleatoire et des plus proches voisins ont été utilisées. Une version élaborée de la méthode des plus proches voisin a également été présentée dans ce chapitre et se montre bien plus performante que les autres heuristiques. Néanmoins, la qualité moyenne des solutions produites par ces heuristiques est très faible et le rapport vitesse de calcul / qualité est peu intéressant.

Afin de produire des solutions de qualité satisfaisante en un temps raisonnable, deux méthodes de résolutions basées sur la méta-heuristique fourmi ont été proposées. L'une est dite en-ligne (*on-line*) et l'autre est dite en-ligne (*on-line*). La méthode en-ligne permet de calculer une solution au temps  $t$  après avoir effectuer un certain nombre d'itérations de l'algorithme. La seconde méthode permet d'obtenir une solution à chaque instant (*anytime solution*) mais à l'inconvénient de potentiellement fournir des solutions partielles.

Toutes ces méthodes ont été testées grâce à un simulateur de terminal portuaire à conteneurs développé durant cette thèse et décrit dans le chapitre suivant. Les expérimentations seront décrites et les résultats seront analysés dans le dernier chapitre de ce manuscrit.

# Chapitre IV

## Simulateur de terminal à conteneurs

### Introduction

Ce chapitre présente *D<sup>2</sup>CTS* : un simulateur de terminal portuaire à conteneurs conçu et développé durant cette thèse. Les problèmes d'ordonnancement et d'affectation ainsi que de routage dynamique sont des problématiques théoriques ici inscrites dans un contexte concret. Les terminaux portuaires à conteneurs sont des structures privées difficilement abordables. Ils fonctionnent en continu et il est donc impossible de procéder à des tests grandeure nature sur une journée d'exploitation. Un simulateur permet ainsi de réaliser ces mesures de performance dans un environnement virtuel le plus réaliste possible avant d'hypothétiquement passer à la mise en place à l'échelle réelle.

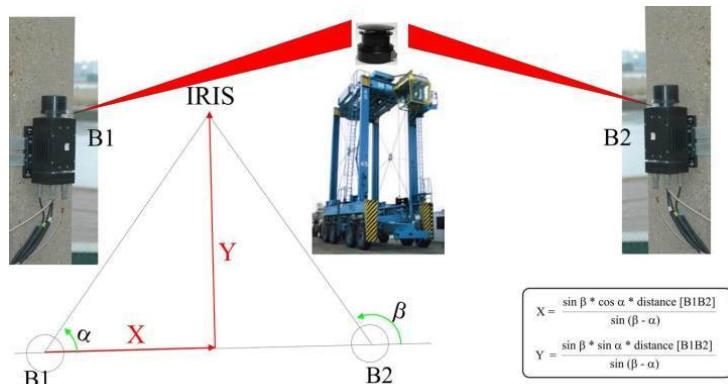
Le programme a été élaboré lors de la participation du LITIS au projet CALAS qui est l'acronyme de *CArrier LAser tracking System*. Le projet consiste à élaborer une technologie de localisation des engins de manutention capable de fonctionner à n'importe quel endroit du terminal. En effet, la technologie de géolocalisation actuelle utilise des satellites (GPS : *Global Positioning System*) afin de déterminer les coordonnées d'un émetteur. Or, le signal des satellites traversant mal le métal, les véhicules qui se trouvent sous les portiques de déchargement ou dans les travées de conteneurs ne sont pas repérés.

La société *Laser Data Technology Terminal* (LDTT) a mis au point une technologie de géolocalisation utilisant un rayon laser et exploitant la caractéristique physique principale des chariots cavaliers : leur hauteur. En effet, les chariots cavaliers sont les engins mobiles autonomes les plus élevés du terminal. Il est donc possible de déterminer leur position grâce à un signal horizontal émis à la hauteur du sommet des chariots cavaliers afin d'être en mesure de localiser les véhicules équipés à n'importe quel endroit du terminal. Le système est composé d'un réseau

d'émetteurs/récepteurs laser (*InfraRed Intelligent Sensors*) IRIS répartis sur le terminal (voir figure IV.1). D'autres bornes IRIS sont installées sur les chariots cavaliers et permettent de réaliser une triangulation du signal infrarouge (voir figure IV.2).



**Figure IV.1:** Réseau de bornes laser implantées sur le Terminal de Normandie (source : <http://www.ldtt-fr.com>)



**Figure IV.2:** Triangulation du signal infrarouge entre les bornes IRIS du terminal et celle d'un chariot cavalier (source : <http://www.ldtt-fr.com>)

Après plusieurs années de développement et de tests réels, cette technologie se montre performante et fiable et permet de connaître en temps réel la position des engins de manutention au sein du terminal. Cette information est la condition *sine qua non* à toute recherche d'optimisation dynamique des activités des engins de manutention. Grâce à la position des véhicules il est ainsi possible d'optimiser dynamiquement le routage des chariots cavaliers et de prendre en compte les durées de parcours au sein du terminal. Ceci permet par conséquent, d'optimiser l'activité des chariots cavaliers tout en contrôlant le suivi de leurs opérations. En effet, lors-

qu'un conteneur est chargé ou déposé par un chariot cavalier, un signal contenant la position du véhicule est envoyé au système. Ainsi, le système connaît la position de prise du conteneur (et par conséquent le conteneur chargé) ainsi que sa position de dépôse. Ces informations permettent d'éviter les pertes de conteneurs au sein du terminal.

La partie LITIS du projet consistait à proposer des méthodes d'optimisation dynamique des activités des chariots cavaliers en utilisant l'information fournie par le système de géolocalisation laser.

## IV.1 Spécifications

### IV.1.1 Technologie

Le simulateur est écrit en JAVA. Cette technologie a été choisie pour sa souplesse et sa puissance. Elle permet le développement du cœur de l'application, de la vue et du contrôleur et assure l'interopérabilité des systèmes et des plates-formes. Une base de données MySQL est également utilisée afin de permettre la communication avec l'interface 3D de notre partenaire *EADS Astrium*. L'accès en écriture et en lecture à cette base est réalisé par un service web. Le terminal étant constitué de multiples entités en fortes interactions, les calculs sont distribuables sur plusieurs unités grâce à la technologie *RMI* de JAVA.

### IV.1.2 Objectifs

Le simulateur doit permettre de représenter la structure d'un terminal à conteneurs (blocs, travées, emplacements, carrefours, routes, quais, voies ferrées, etc.) ainsi que ses composants (portiques de berge, portes conteneurs, chariots cavaliers, etc.). Il doit également permettre de modéliser son activité (arrivée/départ des clients (camions, trains, navires), déplacement de conteneurs par les chariots cavaliers, chargement/déchargement des clients par les chariots cavaliers et les portiques). Pour cela, le temps doit être pris en compte par le simulateur. Il a été décidé de discréteriser la représentation du temps dans le simulateur pour être en mesure de se soustraire de l'influence de la (ou les) machine(s) d'exécution et pour synchroniser efficacement les composants. Un pas de temps devra être déterminé avant le début de chaque simulation et déterminera à la fois la précision temporelle et la durée des calculs de la simulation.

### IV.1.3 Flexibilité

Afin de permettre de mesurer la performance de plusieurs méthodes d'optimisation, le simulateur doit être adapté au développement de ce multiples algorithmes. Il doit permettre d'ajouter, de modifier ou de supprimer rapidement et facilement une méthode d'optimisation. De même, les données concernant les composants et la structure du terminal doivent être aisément modifiables. Ainsi, il a été choisi de décrire toutes les données nécessaire à la fois à la configuration du programme et aux composants du terminal dans des fichiers XML dont la structure est décrite dans les sections suivantes.

## IV.2 Modélisation du système

Un terminal portuaire à conteneurs est un système complexe. En effet, on peut définir un système complexe comme un groupe très important d'entités en interactions et dont le comportement ne peut pas, à lui seul, définir l'état futur du système. De plus, ce système est ouvert sur l'extérieur, il est donc soumis à des flux entrant et sortant d'information générant des perturbations sur l'état du système. Un terminal portuaire peut alors être vu comme la superposition de plusieurs sous-systèmes en interactions les uns avec les autres. Parmi ces sous-systèmes on trouve le réseau routier, les espaces de stockages, les véhicules ainsi que leur système de mobilité, et enfin les éléments stockés c'est-à-dire les conteneurs. Le simulateur a donc pour objectif de modéliser le plus fidèlement possible ces éléments ainsi que leurs interactions.

### IV.2.1 Graphe routier

Les carrefours et les routes composant le réseau routier du terminal décrivent un graphe  $G = (V, A)$  où  $V$  est l'ensemble des sommets et  $A$  est l'ensemble des arcs. Les arcs sont orientés et lorsque la circulation d'un véhicule est possible dans les deux sens entre deux carrefours  $v_i$  et  $v_j$  alors il existe deux arcs  $(v_i, v_j)$  et  $(v_j, v_i)$ . Chaque arc du graphe comporte un poids modélisant la distance séparant les deux sommets de l'arc. Un carrefour routier permet de relier au moins deux routes.

Afin de modéliser des routes sinueuses, la notion de point de route a été introduite. Un point de route est un sommet permettant de relier deux arcs d'une même route. Ainsi une même route comporte 2 carrefours et  $n$  points de routes. Ce procédé permet de simplifier le calcul des plus court chemins en réduisant le nombre d'arcs dans le graphe.

La figure IV.3 montre la vue d'une partie du graphe routier d'un terminal dans le simulateur.



**Figure IV.3:** Exemple de routes comportant des points de route sur le Terminal de Normandie

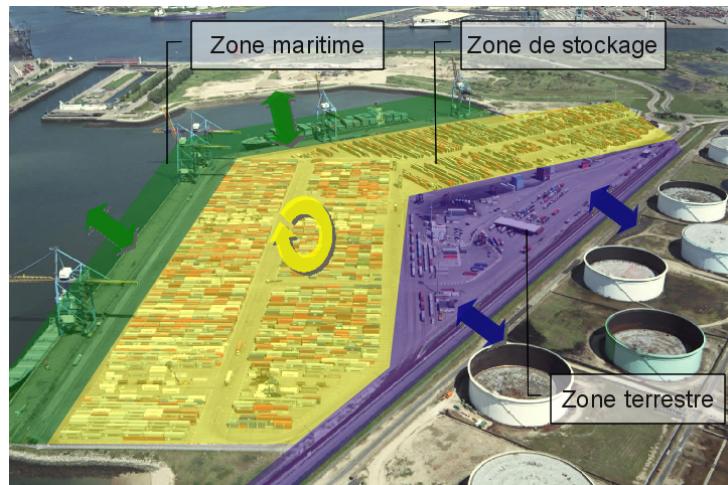
Les travées sont des routes réservées aux chariots cavaliers sur lesquelles ces engins ne peuvent pas se croiser. Une travée est modélisée par un arc First In, First Out (FIFO) relié aux routes par des points de travées. Un point de travée est un point de route reliant à la fois deux arcs et une ou plusieurs travées. Avec cette modélisation il est possible de représenter le réseau routier de n'importe quel terminal à conteneurs.

#### IV.2.2 Réseau de stockage

Avant le début du projet CALAS, le terminal était composé de 2 principaux sous-systèmes. D'une part le réseau routier permettant aux véhicules de se déplacer à l'intérieur du terminal et d'autre part le réseau de stockage. Ce dernier est composé de 3 zones (voir fig. IV.4) :

- La **zone maritime** (*Quay side*) permettant de charger ou de décharger la cargaison des navires ;
- La **zone de stockage interne** (*Yard*) dédiée au stockage temporaire des conteneurs en attente de transit ;
- La **zone terrestre** (*Land side*) permettant de charger ou de décharger la cargaison des véhicules terrestres, c'est-à-dire des camions et des trains.

Chacune des zones est en interaction avec le réseau routier et comporte différents type d'engins de manutention. Des grues de quai sont utilisées dans la zone maritime pour charger/décharger les navires. Dans le cas d'un déchargement, les conteneurs sont posés sur le quai par la grue puis un chariot cavaliers se charge de transporter le conteneur vers sa destination,



**Figure IV.4:** Les 3 types de zones du Terminal de Normandie

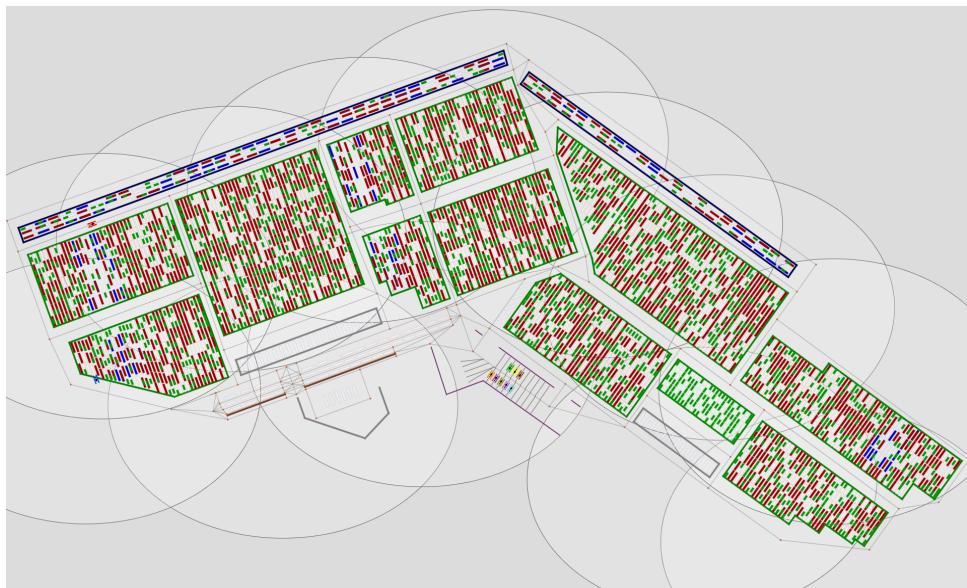
c'est-à-dire soit la zone de stockage, soit la zone terrestre, ou soit, dans le cas d'un cabotage, à un autre endroit sur la zone maritime. Dans le cas d'un chargement c'est le chariot cavalier qui amène le conteneur au pied de la grue. Concernant la zone de stockage et la zone terrestre, ce sont les chariots cavaliers qui sont affectés directement au chargement/déchargement des camions ou des trains et au transport des conteneurs entre les zones. Ces véhicules sont capables de transporter un conteneur sur 3 voire 4 niveaux pour les plus récents. Ils peuvent ainsi se déplacer avec un conteneur dans une travée de 3 étages pour les meilleurs et 2 étages pour les autres.

Chaque zone est composée de pavés. Un pavé (ou bloc) est un regroupement de travées de conteneurs. Chaque pavé comporte donc un certain nombre de travées. Une travée est reliée au réseau routier par des points de travée et représente donc une route particulière qui ne peut être empruntée que par les chariots cavaliers. Elles comportent une série d'emplacements de différentes longueurs pour stocker les conteneurs. Les emplacements où se garent les camions pour être chargés/déchargés sont ainsi modélisés par une travée ne comportant qu'un seul emplacement. Les wagons de trains sont eux aussi modélisés par des travées ne pouvant contenir qu'un seul étage de conteneurs. Pour ces deux types de véhicules, les emplacements sont disponibles si le camion ou le train est en place. Dans le cas contraire, les chariots cavaliers ne peuvent effectuer le chargement ou le déchargement du conteneur.

#### IV.2.3 Système de géolocalisation laser (LDTT)

Suite au projet CALAS, le terminal de Normandie s'est vu doté d'un troisième sous-système : la géolocalisation des engins de manutention. En effet, un réseau de bornes laser a été déployé

sur le terminal dans le but de mesurer avec précision la position des véhicules. Ces bornes communiquent avec un serveur central afin de transmettre les coordonnées des chariots. Ce système a été modélisé en simulant la détection des chariots par les bornes selon une certaine portée. Une fois détecté, la borne laser envoie la position du véhicule au système central qui met à jour les informations connues sur ce chariot. De cette façon, si un véhicule se trouve en dehors de la couverture du réseau de bornes laser, le système ne reproduit que sa dernière position connue. La figure IV.5 montre une vue 2D du Terminal de Normandie dans le simulateur, et où chaque cercle représente la portée d'une borne laser.



**Figure IV.5:** Vue 2D du simulateur montrant le Terminal de Normandie ainsi que le système de localisation laser

#### IV.2.4 Mobilité au sein du terminal

Dans la modélisation choisie ici, seuls les chariots cavaliers sont mobiles. En effet, ils sont les seuls véhicules à pouvoir déplacer les conteneurs au sein du terminal. La mobilité des autres véhicules (navires, camions et trains) ne comporte que les actions d'arrivée et de départ. Ainsi, un camion est soit sur son emplacement de chargement/déchargement, soit en dehors du terminal. Toutefois, d'autres véhicules peuvent être ajoutés facilement en décrivant leur comportement dans le simulateur.

Les chariots cavaliers sont donc les seuls véhicules à pouvoir emprunter les travées de conteneurs. Cependant, ils ne peuvent pas se croiser à l'intérieur de celles-ci. C'est cet élément qui caractérise les arcs (ou arêtes) de type travée. En effet, les travées sont *First-In-First-Out* (pre-

mier entré, premier sorti), c'est-à-dire que les véhicules sortent de la travée dans le même ordre qu'ils y sont entrés. Pour éviter des blocages, les chariots cavaliers n'ont pas l'autorisation d'emprunter une travée déjà occupée par un autre chariot. Si la travée est prise, ils devront attendre à l'entrée de celle-ci jusqu'à ce qu'elle soit libérée.

Sur les routes, les chariots peuvent se croiser et se doubler. Néanmoins, les dépassements ne sont pas modélisés fidèlement, c'est-à-dire que les différentes voies de circulation ne sont pas modélisées et que les collisions ne sont pas prises en compte. Ce procédé permet de simplifier la modélisation sans pour autant dégrader la qualité de la simulation. En effet, suite à une collision, les véhicules deviendront simplement indisponibles (en panne) durant un certain temps.

Le comportement des conducteurs des chariots est complexe à reproduire. En théorie, ces derniers choisissent une mission à effectuer parmi celles que le système leur propose et se rendent sur les emplacements de collecte et de livraison de conteneurs selon l'itinéraire affecté par le système. En réalité les conducteurs s'échangent des missions entre eux et choisissent leur propre itinéraire. Ces comportements ont été modélisés par des événements de non respect d'affectation et d'itinéraire. De cette façon, il est possible d'introduire de la dynamicité dans la simulation et surtout de pouvoir la quantifier facilement.

#### IV.2.5 Temporalité

L'objectif étant d'étudier l'impact de la dynamicité sur l'évolution du système, la modélisation du temps est essentielle. Il a été décidé de se placer dans le cas discret, avec un pas de temps réglable. Ainsi, il est possible de modéliser les événements liés à la dynamique.

Le simulateur permettant de préciser le pas de temps pour une simulation, il sera possible d'étudier l'influence du choix du pas de temps sur les résultats des simulations au niveau macro. La figure IV.6 montre deux captures d'écran d'une simulation à 1 pas de temps d'intervalle. Ici le véhicule mauve situé dans le quart inférieur droit de l'image continue son déplacement. La vitesse du véhicule étant de 27km/h, il aura effectué 15 mètres pendant les 2 secondes simulées.



(a) Capture d'écran du simulateur à t=3m33s



(b) Capture d'écran du simulateur à t=3m35s

**Figure IV.6:** Exemple de représentation discrète du temps, ici avec un pas de temps de 2 secondes par itération

#### **IV.2.6 Événements**

Les événements interviennent au cours du temps et possèdent donc un marqueur temporel de déclenchement. Il existe des événements de différentes natures :

- arrivée de mission : une nouvelle mission est connue du système ;
- annulation de mission : une mission déjà connue est retirée du système ;
- arrivée de véhicule (bateau, camion, train) : un véhicule est arrivé sur son/ses emplacements ;
- départ de véhicule (bateau, camion, train) : un véhicule a quitté son/ses emplacements ;
- panne de chariot cavalier : un chariot est indisponible ;
- fin de panne de chariot cavalier : un chariot indisponible redevient disponible ;
- non respect d'affectation de mission d'un chariot cavalier : le conducteur d'un chariot a choisi une mission qui ne lui était pas destinée ;
- non respect d'itinéraire d'un chariot cavalier : le conducteur d'un chariot a choisi un itinéraire différent de celui proposé par le système ;
- perte de conteneur : un conteneur ne se trouve pas à l'emplacement indiqué par le système.

L'objectif est de reproduire un terminal portuaire à conteneurs tant dans son contenu que dans son comportement. Chaque sous-système du terminal a donc été modélisé ainsi que les interactions, à la fois à l'intérieur et entre ces sous-systèmes.

### **IV.3 Collecte des données**

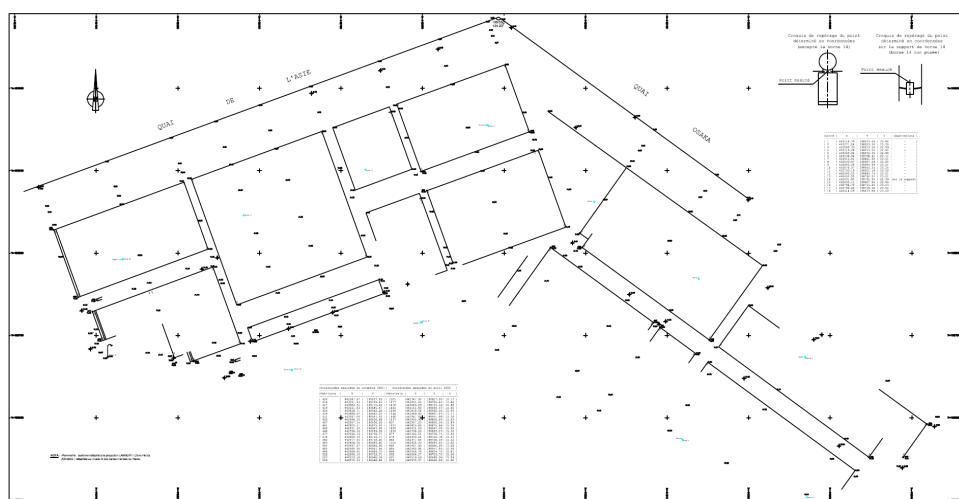
Une fois le modèle établi, il est nécessaire de collecter des données afin de décrire un terminal existant. Le choix s'est porté sur le Terminal de Normandie, un des terminaux du Port Autonome du Havre directement impliqué dans le projet CALAS.

La difficulté rencontrée quant à l'obtention d'information concernant les données du terminal est l'une des raisons pour lesquelles le développement d'un simulateur a été la solution retenue pour de mettre au point et tester la performance de nos méthodes. Une grande partie du temps alloué au projet a donc été consacrée à la collecte d'informations sur :

- Les chariots cavaliers : dimensions, vitesse, comportement ;
- Le fonctionnement du terminal : différentes zones d'échange, zone de stockage, engins de manutention ;
- Le réseau routier du terminal :

- coordonnées des carrefours ;
  - routes ;
  - travées ;
  - coordonnées des emplacements conteneurs dans les travées ;
  - zone de dépôt des engins de manutention.

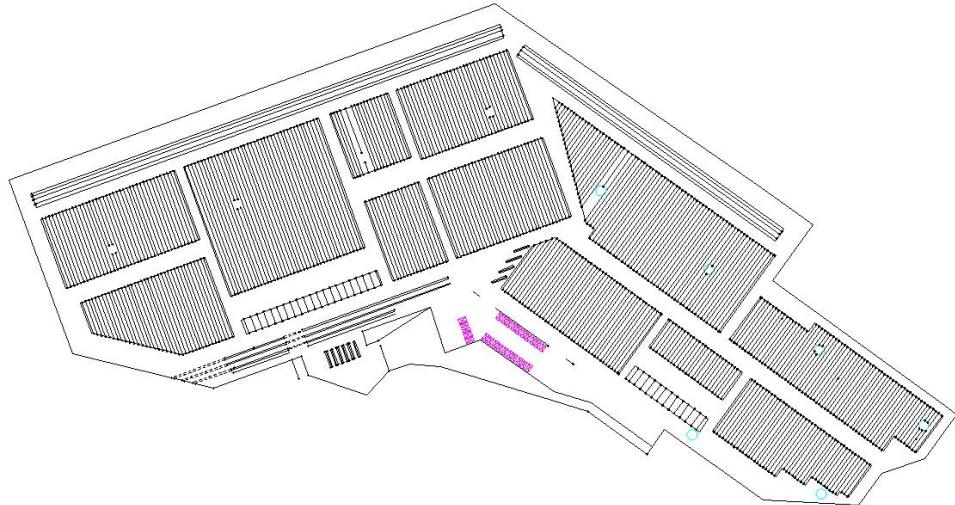
Les coordonnées des carrefours, routes, travées et emplacements conteneurs du Terminal de Normandie ont été obtenus grâce à un plan sommaire du terminal fourni par nos partenaires du projet CALAS (voir figure IV.7) et au site internet de cartographie [wikimapia.org](http://wikimapia.org). Ce site permet de mesurer de façon relativement précise des distances sur des photos satellites d'une grande définition. Les coordonnées indiquées sur le plan fourni par nos partenaires sont exprimés en suivant la projection conique conforme de Lambert (zone centre : Lambert II). Un degré de ce système correspond à 100km. Cette équivalence permet de calculer aisément les coordonnées des points manquant du plan grâce aux distances mesurées sur le site [wikimapia.org](http://wikimapia.org).



**Figure IV.7:** Plan sommaire du Terminal de Normandie

La figure IV.8 montre le plan obtenu grâce aux recroisement des informations du plan sommaire et de wikimapia.org. Le Terminal de Normandie, loin d'être le plus grand au monde, comporte tout de même 1170 carrefours, 170 routes, 531 travées et 3499 emplacements conteneurs.

D'autre part, les images satellites ont également permis de collecter des données sur les chariots cavaliers et notamment leur lieu de stationnement (dépôt). Des données concernant les chariots cavaliers ont également été récupérées sur les sites des constructeurs afin de connaître leurs principales dimensions :



**Figure IV.8:** Plan du Terminal de Normandie obtenu après collecte et recoupelement des données

- longueur hors tout ;
  - largeur hors tout ;
  - hauteur hors tout ;
  - longueur des panneaux latéraux ;
  - largeur interne (entre les deux panneaux latéraux) ;
- ainsi que des informations sur la vitesse moyenne de fonctionnement, les temps de manutention, etc. Toutes ces informations sont décrites dans des fichiers au format XML.

#### IV.4 Structuration des données

Les données des simulations sont décrites dans des fichiers XML. L'Extensible Markup Language (langage extensible de balisage 5) permet de définir de façon lisible des caractéristiques et des comportements dans de simple fichiers texte et sont également rapidement interprétables par le programme. Il existe différents fichiers de configurations nécessaires au fonctionnement du simulateur :

- le fichier de déploiement de l'application ;
- le fichier de configuration du système de localisation laser (position et portée des bornes) ;
- le fichier de description des véhicules ;
- le fichier de configuration du terminal (zones de stockage, réseau routier) ;
- le fichier d'initialisation du terminal permettant de décrire la position des conteneurs au démarrage de la simulation ;
- les fichiers d'événements (missions, pannes de véhicule ou de borne laser, arrivées ou

départs des véhicules clients, etc).

#### IV.4.1 Réseau routier

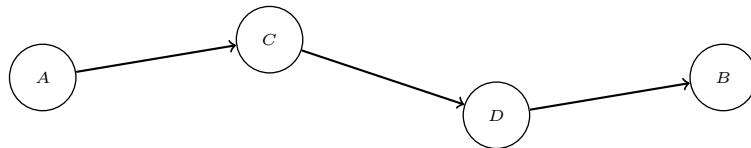
Les routes et les carrefours d'un terminal portuaire à conteneurs constituent un graphe. Ainsi, les nœuds du graphe sont des carrefours et les routes sont des arcs. La structure XML doit donc permettre de décrire ce graphe. Les coordonnées sont exprimées en mètres selon la projection conique de Lambert.

```
<crossroad id='' x='' y=''/>
<road
  id=''
  origin='crossroadId'
  destination='crossroadId'
  [directed='boolean']
/>
```

**Figure IV.9:** Code XML nécessaire à la description d'un carrefour et d'une route

Un arc étant représenté par une droite entre deux nœuds, l'objet roadpoint (point de route) a été introduit pour permettre de modéliser des routes sinueuses. Ainsi une route est une liste d'arcs dont le premier part d'un nœud et le dernier se termine par un autre noeud. Tous les autres arcs de cette route relient des points de route. La figure IV.10 donne un exemple de code XML décrivant une route reliant le carrefour A au carrefour B et utilisant deux points de route, ainsi que le graphe ainsi généré.

```
<crossroad id='A' x='0' y='1' > </crossroad>
<crossroad id='B' x='3' y='1' > </crossroad>
<road id=' ' origin='A' destination='B'>
<roadpoint id='C' x='1' y='2' > premier point </roadpoint>
<!--... autres points de route eventuels -->
<roadpoint id='D' x='2' y='0' > dernier point </roadpoint>
</road>
```



**Figure IV.10:** Exemple de description XML d'un arc comportant des points de route

#### IV.4.2 Zones de stockage

Il existe deux types de zones sur un terminal à conteneurs. D'une part, les zones d'échanges (quais, zone ferroviaire et zone routière) et les zones de stockage d'autre part. Une zone est modélisée par la balise `<pave id='' type='{BOAT, ROAD, TRAIN, STOCK}'></pave>`. À l'intérieur

```

<pave id="L" type="ROAD">
<coordinate id="L-NW" x="443051.6250" y="198654.4978"/>
<coordinate id="L-NE" x="443138.2976" y="198591.7532"/>
<coordinate id="L-SE" x="443127.3772" y="198576.6273"/>
<coordinate id="L-SW" x="443040.6241" y="198639.2604"/>
<wall from="L-NW" to="L-NE"/>
<wall from="L-NE" to="L-SE"/>
<wall from="L-SE" to="L-SW"/>
<wall from="L-SW" to="L-NW"/>
</pave>

```

**Figure IV.11a:** Code XML d'une zone de chargement/déchargement de camions

de cette balise se trouvent les coordonnées des contours de la zone. La balise `<wall from=' ' to=' '>` `</wall>` permet de relier ces points.



**Figure IV.11b:** Rendu 2D de cette zone dans *D<sup>2</sup>CTS*

Les travées sont également décrites à l'intérieur de la balise `<pave>` de la zone concernée. Les points d'entrée et de sortie des travées sur les routes sont des points de routes spécifiques décrits par la balise `<laneCrossroad>` et sont insérés à l'intérieur de la balise `<road>` de la route permettant d'accéder à la travée. La balise `<lane id=' ' origin=' ' destination=' ' slots=' '>` permet de décrire une travée grâce à l'identifiant, le point d'entrée, le point de sortie et la description des emplacements conteneurs de cette travée. Cette dernière information est donnée sous la forme : *tailleConteneurEnPieds \* nombreD'emplacementsDeCetteTaille*

Par exemple, une travée comportant 2 emplacements de 40 pieds puis 1 emplacement de 20 pieds sera décrite en XML de la façon suivante :

```
<lane id='> origin='> destination='> slots='40*2-20*1' />
```

De plus, les coordonnées du premier et du dernier emplacement de la travée peuvent être précisées grâce aux paramètres `in='coordX,coordY'` et `out='coordX,coordY'`. Dans le cas contraire, ces coordonnées sont calculées de façon à conserver la même distance entre les conteneurs ainsi qu'entre le premier emplacement et le point d'entrée de la travée et entre le dernier emplacement et le point de sortie de la travée. Si les coordonnées `in` et `out` sont précisées alors les emplacements contenus entre le premier et le dernier sont répartis de façon régulière entre ces deux emplacements. Les emplacements de trains et de camions sont modélisés par des travées spécifiques. Ces travées sont décrites par la balise `<exchangeLane>` en XML. Les paramètres sont les mêmes que pour une travée classique (`<lane>`). La figure IV.12 donne un exemple d'utilisation de ces balises.

```

<pave id="L" type="ROAD">
<coordinate id="L-NW" x="443051.6250" y="198654.4978"/>
<coordinate id="L-NE" x="443138.2976" y="198591.7532"/>
<coordinate id="L-SE" x="443127.3772" y="198576.6273"/>
<coordinate id="L-SW" x="443040.6241" y="198639.2604"/>
<wall from="L-NW" to="L-NE"/>
<wall from="L-NE" to="L-SE"/>
<wall from="L-SE" to="L-SW"/>
<wall from="L-SW" to="L-NW"/>
<road id="c32-c38" origin="c32" destination="c38">
<laneCrossroad id="L_S_1" x="443037.4695" y="198627.9259"/>
<laneCrossroad id="L_S_2" x="443044.1513" y="198623.1196"/>
<laneCrossroad id="L_S_3" x="443050.8331" y="198618.3134"/>
<laneCrossroad id="L_S_4" x="443057.5149" y="198613.5071"/>
<laneCrossroad id="L_S_5" x="443064.1967" y="198608.7009"/>
<laneCrossroad id="L_S_6" x="443070.8785" y="198603.8946"/>
<laneCrossroad id="L_S_7" x="443077.5602" y="198599.0884"/>
<laneCrossroad id="L_S_8" x="443084.2420" y="198594.2822"/>
<laneCrossroad id="L_S_9" x="443090.9238" y="198589.4759"/>
<laneCrossroad id="L_S_10" x="443097.6056" y="198584.6697"/>
<laneCrossroad id="L_S_11" x="443104.2874" y="198579.8634"/>
<laneCrossroad id="L_S_12" x="443110.9692" y="198575.0572"/>
<laneCrossroad id="L_S_13" x="443117.6509" y="198570.2510"/>
</road>
<exchangeLane id="L-1/13" origin="L_N_1" destination="L_S_1" slots="45*1"
    in="443054.9578,198652.0845" out="443043.9608,198636.8514"/>
<exchangeLane id="L-2/13" origin="L_N_2" destination="L_S_2" slots="45*1"
    in="443061.6258,198647.2580" out="443050.6341,198632.0335"/>
<exchangeLane id="L-3/13" origin="L_N_3" destination="L_S_3" slots="45*1"
    in="443068.2929,198642.4315" out="443057.3074,198627.2155"/>
<exchangeLane id="L-4/13" origin="L_N_4" destination="L_S_4" slots="45*1"
    in="443074.9600,198637.6050" out="443063.9807,198622.3976"/>
<exchangeLane id="L-5/13" origin="L_N_5" destination="L_S_5" slots="45*1"
    in="443081.6271,198632.7785" out="443070.6540,198617.5797"/>
<exchangeLane id="L-6/13" origin="L_N_6" destination="L_S_6" slots="45*1"
    in="443088.2942,198627.9520" out="443077.3273,198612.7617"/>
<exchangeLane id="L-7/13" origin="L_N_7" destination="L_S_7" slots="45*1"
    in="443094.9613,198623.1255" out="443084.0007,198607.9438"/>
<exchangeLane id="L-8/13" origin="L_N_8" destination="L_S_8" slots="45*1"
    in="443101.6284,198618.2990" out="443090.6740,198603.1259"/>
<exchangeLane id="L-9/13" origin="L_N_9" destination="L_S_9" slots="45*1"
    in="443108.2956,198613.4724" out="443097.3473,198598.3080"/>
<exchangeLane id="L-10/13" origin="L_N_10" destination="L_S_10" slots="45*1"
    in="443114.9627,198608.6459" out="443104.0206,198593.4900"/>
<exchangeLane id="L-11/13" origin="L_N_11" destination="L_S_11" slots="45*1"
    in="443121.6298,198603.8194" out="443110.6939,198588.6721"/>
<exchangeLane id="L-12/13" origin="L_N_12" destination="L_S_12" slots="45*1"
    in="443128.2969,198598.9929" out="443117.3672,198583.8542"/>
<exchangeLane id="L-13/13" origin="L_N_13" destination="L_S_13" slots="45*1"
    in="443134.9640,198594.1664" out="443124.0405,198579.0362"/>
</pave>

```

**Figure IV.12:** Exemple de code XML de description d'une travée

#### IV.4.3 Conteneurs

Le simulateur tient compte de l'hypothèse selon laquelle il existe 3 types de conteneurs sur le Terminal de Normandie. D'abord les conteneurs de 20 pieds : 1 TEU (twenty feet equivalent unit) ; les conteneurs de 40 pieds : 2 TEU ; et enfin les conteneurs de 45 pieds : 2.25 TEU qui sont des conteneurs réfrigérés. Les dimensions de ces conteneurs sont les suivantes (normes ISO) :

Type	Longueur (m)	Largeur (m)	Hauteur (m)
20 pieds	6,058	2,438	2,591
40 pieds	12,192	2,438	2,591
45 pieds	13,716	2,438	2,896

Pour pouvoir décrire un conteneur en XML il faut donner l'identifiant du conteneur et son type (en TEU). Il est également possible de spécifier sa position. Cette dernière est définie par un pavé, une travée, un emplacement, un niveau et enfin un alignement. Le niveau correspond à l'étage auquel le conteneur est stocké sur l'emplacement. L'étage 0 correspond au rez-de-chaussé (le conteneur est posé sur le sol du terminal). Il est possible de stocker deux niveaux de conteneurs dans les travées de stockage et un seul niveau sur les trains et les camions. L'alignement du conteneur permet de définir si le conteneur est centré sur son emplacement, c'est-à-dire que l'espace entre les extrémités de l'emplacement et le conteneur est le même, ou si le conteneur est aligné sur un coté de l'emplacement. La valeur '`origin`' correspond à un alignement le plus proche du point d'entrée de la travée de l'emplacement. La valeur '`destination`' correspond quant à elle à l'alignement le plus proche du point de sortie de la travée contenant l'emplacement de stockage.

```
<container id="LWCU 429852 1" teu="2.25">
  <containerLocation pave="Asie" lane="Asie-3/3" slot="Asie-3/3-34" level="0"
    align="center"/>
</container>
```

**Figure IV.13:** Exemple de code XML de description d'un conteneur

#### IV.4.4 Missions

Une mission est un déplacement de conteneur au sein du terminal. Elle comporte 2 phases. Tout d'abord la phase de collecte du conteneur par le chariot cavalier, c'est la phase de *pickup*. Dans cette phase le chariot effectue le trajet de son point de départ vers la position courante du conteneur. Il doit respecter une fenêtre de temps pour pouvoir le récupérer. Une fois le chariot en position et le conteneur récupéré, la deuxième phase, celle de livraison du conteneur (*delivery*) commence. Cette phase consiste à amener le conteneur sur son emplacement de destination, tout en respectant également une fenêtre de temps. Une fois la destination atteinte, le conteneur est déchargé du chariot et déposé sur son emplacement de destination. La mission est alors terminée. Pour pouvoir décrire ces différentes opérations la structure XML de la balise `<mission>` se compose d'un identifiant, de l'identifiant du conteneur concerné et du type de mission. Une mission peut être de type :

- STAY : un conteneur doit être déplacé d'une zone de stockage du terminal à une autre et donc va rester dans le terminal ;
- IN : un conteneur provenant d'un véhicule externe (train, camion ou bateau) doit être

- déplacé dans une zone de stockage du terminal et donc va entrer dans le terminal ;
- OUT : un conteneur d'une zone de stockage doit être chargé sur un véhicule extérieur (train, camion ou bateau) et donc va sortir du terminal ;
  - IN\_AND\_OUT : un conteneur doit être déchargé d'un véhicule externe puis être chargé sur un autre véhicule externe. Dans ce type de mission, le conteneur ne passe pas par une zone de stockage.

Chaque type comporte un identifiant numérique :

Type	Identifiant
STAY	0
IN	1
OUT	2
IN_AND_OUT	3

Il est possible de spécifier dans le paramètre '`kind`' de la balise `<mission>` soit le type de mission, soit son identifiant. Une fois ces trois paramètres fournis, il faut définir les fenêtres de temps de collecte (*pickup*) et de livraison (*delivery*) et l'emplacement de destination du conteneur. La balise `<timewindow>` correspond à la description d'une fenêtre de temps et comporte les paramètres `start` et `end` correspondant respectivement à l'heure de début et de fin de cette fenêtre de temps. Le temps est donné sous la forme `hh:mm:ss.ms`. La destination du conteneur pour la mission est définie par la balise `<containerLocation>` décrite précédemment.

```
<mission id="m24" container="TXIU 696005 5" kind="0">
  <timewindow start="00:10:39.49" end="00:15:59.24"/>
  <timewindow start="00:12:05.23" end="00:18:07.85"/>
  <containerLocation pave="N" lane="N-25/42" slot="N-25/42-0" level="1" align="center"/>
</mission>
```

**Figure IV.14:** Exemple de code XML de description d'une mission

#### IV.4.5 Bornes de positionnement laser

Le système de géolocalisation laser doit être décrit dans un fichier indépendant. Dans ce fichier la description du système comportera plusieurs informations. Une borne a un identifiant ('`id`'), une position ('`x`', '`y`' et '`z`') et une portée ('`rangeX`', '`rangeY`' et '`rangeZ`') indiquée en mètres.

Si un véhicule se trouve à portée de la borne, alors celle-ci va capter son signal et renvoyer la position au simulateur.

```

<laserheads>
  <laserhead id="1" x="443118.75" y="198573.65" z="22.84" rangeX="125" rangeY="100"
    rangeZ="40"/>
  <laserhead id="2" x="443271.54" y="198503.34" z="23.30" rangeX="125" rangeY="100"
    rangeZ="40"/>
</laserheads>

```

**Figure IV.15:** Exemple de code XML de description d'un réseau composé de deux bornes laser

#### IV.4.6 Description des véhicules

Les informations permettant de décrire les types de chariots cavaliers présents sur le terminal ainsi que les instances de ces types doivent être regroupées dans un fichier. Il est nécessaire de décrire les types (ou modèles de chariots), puis chacun des véhicules.

##### IV.4.6.1 Types de chariots cavaliers

Un type de chariot cavalier donne les informations permettant de décrire les attributs d'un chariot de ce modèle. Il comporte :

- un identifiant ;
- la largeur ('width') en mètres ;
- la hauteur ('height') en mètres ;
- la longueur ('length') en mètres ;
- la largeur intérieure ('innerWidth') en mètres correspondant à l'espace libre entre les roues gauches et les roues droites du chariot ;
- la longueur intérieure ('innerLength') en mètres correspond à la longueur entre les deux barres transversales du chariot ;
- les longueurs entre ces barres transversales et les extrémités du chariot ('backOverLength' et 'frontOverLength') en mètres ;
- la largeur de la cabine ('cabWidth') en mètres ;
- la vitesse du chariot ('speed') en mètres par seconde ;
- la vitesse en marche arrière ('reverseSpeed') en mètres par seconde ;
- la vitesse en travée ('laneSpeed') en mètres par seconde ;
- la vitesse de demi-tour du poste de pilotage ('turnBackTime') en secondes ;
- la compatibilité du chariot avec les conteneurs ('compatibility'). Cette dernière est décrite de façon binaire. Le premier bit, bit de poids fort, correspond aux conteneurs de 20 pieds, le second aux conteneurs de 40 pieds, et enfin le troisième représente les conteneurs de 45 pieds. Un chariot compatible avec les conteneurs de 40 pieds uniquement aura

comme paramètre de compatibilité la valeur '010'. Dans le cas d'un chariot compatible avec tous les conteneurs (bras de levage réglable), il est possible de spécifier soit la valeur '111' soit la valeur 'all'.

```
<type
  id="standard"
  width="5"
  height="15.4"
  length="10"
  innerWidth="3"
  innerLength="6"
  backOverLength="1"
  frontOverLength="3"
  cabWidth="2"
  speed="8.33"
  reverseSpeed="8.33"
  laneSpeed="5"
  turnBackTime="4"
  compatibility="all"
/>
```

**Figure IV.16:** Exemple de code XML de description d'un type de chariot cavalier

#### IV.4.6.2 Instances de chariots cavaliers

Une fois que les modèles de chariots ont été décrit, il est possible de donner la description de chaque chariot. Un chariot possède un identifiant, un type, un emplacement de stationnement dans le dépôt, une couleur, une machine pour la distribution, une route de départ, une position sur cette route sous forme de taux (entre 0 et 1, où 0 est le point d'origine de l'arc et 1 est le point de destination de l'arc), une direction sur cette route sous forme d'un booléen (`true` signifie que le chariot se dirige dans le sens de l'arc, vers le point de destination). Le routage du chariot cavalier est indiqué dans la balise `<routing>` à l'intérieur de la balise du chariot. Il est nécessaire de spécifier le type du routage et la machine qui devra se charger des calculs (`host`).

```
<straddleCarrier id="cav1" type="standard" slot="Central_1" color="red"
  host="localhost" locationRoad="c2-c5" locationPourcent="0.5" direction="true">
  <routing type="RDijkstra" host="localhost"/>
</straddleCarrier>
<straddleCarrier id="cav2" type="standard" slot="Central_2" color="blue"
  host="localhost" locationRoad="B-4/38" locationPourcent="0.8" direction="true">
  <routing type="RDijkstra" host="localhost"/>
</straddleCarrier>
```

**Figure IV.17:** Exemple de code XML de description de deux chariots cavaliers

#### IV.4.7 Description de l'ordonnanceur de missions

L'optimisation de l'affectation et de l'ordonnancement des missions est l'un des objectifs de la participation du LITIS au projet CALAS. Le simulateur a été conçu de manière à permettre

l'utilisation de diverses politiques et algorithmes d'ordonnancement et notamment ceux évoqués dans le chapitre précédent (voir III p101).

La classe abstraite `MissionScheduler` donne la signature des méthodes `precompute()`, `compute()`, et `apply()`. La méthode `precompute()` est appelée par le contrôleur temporel avant l'exécution d'une itération de la simulation. Puis la méthode `compute()` contient le code d'une itération de l'algorithme d'ordonnancement et doit être appelée à l'intérieur de la méthode `precompute()`. Enfin, la méthode `apply()` est exécutée à la fin de chaque itération de la simulation. Le principe est que chaque objet puisse effectuer ses calculs sur les mêmes données, puis les résultats sont mis en application quand tous les calculs sont terminés.

La classe abstraite contient également toutes les structures de données nécessaire à l'élaboration de l'ordonnancement (liste des ressources et des tâches) ainsi que des files d'événements. Les structures concernant les événements concernent :

- les missions et les véhicules à ajouter dans l'ordonnanceur ;
- les missions et les véhicules à supprimer de l'ordonnanceur ;
- les missions et les véhicules à mettre à jour (fenêtre de temps, position, vitesse, etc).

Chaque accès en écriture sur ces structures déclenche la prise en compte de leur contenu lors de l'exécution de la méthode `precompute()` à l'itération suivante.

D'autre part, les paramètres de la fonction d'évaluation ( $F'_1$ ,  $F'_2$ ,  $F'_3$ ) d'un ordonnancement doit également être défini dans la classe `MissionScheduler`. En revanche, les autres paramètres dépendent de l'implémentation de l'algorithme et seront donc définis dans les classes filles.

Chaque algorithme d'ordonnancement sera donc modélisé par une classe héritant de `MissionScheduler` et contenant la définition des méthodes abstraites. L'implémentation est donc simplifiée au maximum. Chaque classe fille devra contenir un identifiant unique indiqué dans la variable `rmiBindingName` permettant au *parser XML* de déterminer quel algorithme instancier.

Il est donc impératif de modifier le code du *parser XML* lors de l'ajout d'un algorithme. Cette modification concerne la classe `util.parser.XMLNetworkConfigurationParser` et plus particulièrement la méthode `startElement(String uri, String localName, String qName, Attributes atts)`. Il est nécessaire d'ajouter le nouveau type à prendre en compte ainsi que les paramètres de l'algorithme. Les modules d'ordonnancement développés avec le simulateur permettront au développeur de réaliser facilement les modifications nécessaires.

## IV.4.8 Événements

Le simulateur a pour objectif de modéliser un terminal portuaire à conteneurs dans son ensemble : sa structure (réseau routier, zones de stockages, composants...) et également sa dynamique (arrivée de véhicules, de missions, pannes de chariots cavaliers...). Pour cela la balise `<event>` permet de définir une heure de prise en compte du contenu à l'intérieur de la balise par le système. Il devient alors possible de décrire des éléments et de déclencher leur arrivée à une date spécifique, simulant ainsi la dynamique du simulateur. Toute balise `<event>` comportera un type (paramètre '`type`') et une heure de déclenchement (paramètre '`time`'). D'autres paramètres peuvent être ajoutés en fonction du type de l'événement.

### IV.4.8.1 Arrivée de véhicule

Pour décrire l'arrivée d'un véhicule (train, camion ou bateau) sur le terminal il faut indiquer dans la balise `<event>` que le type d'événement est une arrivée de véhicule '`vehicleIn`' et également indiquer les travées sur lesquelles le véhicule est arrivé (paramètre '`lanes`', séparation des valeurs par une virgule). À l'intérieur de cette balise `<event>` il est possible de décrire les conteneurs acheminés par le véhicule. Tous ces conteneurs seront alors stockés sur les travées lorsque la date de l'événement sera atteinte.

```
<event time="0:20:0" type="vehicleIn" lanes="train1_1/4,train1_4/4">
  <container id="SZWU 075947 3" teu="2.0">
    <containerLocation pave="train" lane="train1_1/4" slot="train1_1/4-0" level="0"
      align="origin"/>
  </container>
  <container id="GPMU 632388 2" teu="1.0">
    <containerLocation pave="train" lane="train1_4/4" slot="train1_4/4-1" level="0"
      align="center"/>
  </container>
  <container id="XOPU 972968 1" teu="2.0">
    <containerLocation pave="train" lane="train1_1/4" slot="train1_1/4-2" level="0"
      align="center"/>
  </container>
  <container id="ZGHU 515875 8" teu="1.0">
    <containerLocation pave="train" lane="train1_1/4" slot="train1_1/4-3" level="0"
      align="center"/>
  </container>
  <container id="TVLU 759330 5" teu="1.0">
    <containerLocation pave="train" lane="train1_4/4" slot="train1_4/4-0" level="0"
      align="destination"/>
  </container>
</event>
```

**Figure IV.18:** Exemple de code XML de description de l'événement associé à l'arrivée d'un train

#### IV.4.8.2 Départ de véhicule

Le départ d'un véhicule est décrit de la même façon que pour son arrivée. Le type de l'événement devient '`vehicleOut`'. Le paramètre '`lanes`' permet de connaître l'emplacement de départ du véhicule. La liste des conteneurs devant être emportés par le véhicule est spécifié à l'intérieur de la balise `<event>` par des balises `<container>`. Si le véhicule ne contient pas l'intégralité des conteneurs spécifiés au moment du départ, alors il devra attendre pour partir.

```
<event time="0:30:0" type="vehicleOut" lanes="train3_1/4,train3_4/4">
  <container id="IBMU 639824 4"/>
  <container id="PIFU 715392 7"/>
  <container id="IKOU 345806 4"/>
  <container id="WDTU 864334 6"/>
  <container id="LYUU 819837 0"/>
  <container id="DNDU 050568 4"/>
  <container id="MKGU 520946 8"/>
  <container id="ANZU 209354 8"/>
  <container id="UGGU 129115 4"/>
  <container id="EAIU 487169 1"/>
  <container id="PXEU 295047 2"/>
  <container id="XSSU 356087 6"/>
  <container id="SIFU 857986 3"/>
  <container id="WYMU 125212 4"/>
  <container id="FIDU 063172 1"/>
</event>
```

**Figure IV.19:** Exemple de code XML de description de l'événement associé au départ d'un train

#### IV.4.8.3 Arrivée de mission

Une nouvelle mission peut être connue après le départ de la simulation grâce à la balise `<event>` en spécifiant le type '`newMission`'. La mission en question est alors décrite à l'intérieur de la balise `<event>` comme une mission classique.

```
<event time="00:08:26" type="newMission">
  <mission id="m52" container="EORU 945264 3" kind="0">
    <timewindow start="00:16:24.27" end="00:24:36.41"/>
    <timewindow start="00:18:02.41" end="00:27:03.62"/>
    <containerLocation pave="I" lane="I-4/73" slot="I-4/73-0" level="1" align="center"/>
  </mission>
</event>
```

**Figure IV.20:** Exemple de code XML de description de l'événement associé à l'arrivée d'une mission

#### IV.4.8.4 Affectation de mission à un chariot cavalier

Pour simuler le choix d'un conducteur de chariot cavalier dans les missions, il est possible de décrire l'affectation de ces missions par la balise `<event>` avec le paramètre `type` valant

'**affectMission**'. Le paramètre '**mission**' correspond alors à l'identifiant de la mission à affecter, le paramètre '**straddleCarrier**' est quant à lui, l'identifiant du chariot cavalier affecté à cette mission.

```
<event time="0:10:0" type="affectMission" mission="m10" straddleCarrier="cav9"/>
```

**Figure IV.21:** Exemple de code XML de description de l'événement associé à l'affectation de la mission "m10" au chariot cavalier "cav9"

#### IV.4.8.5 Panne d'un chariot cavalier

Les pannes sont décrites par la valeur '**straddleFailure**' du paramètre '**type**'. Trois autres paramètres sont alors nécessaires :

- l'identifiant du chariot en panne : **straddleId**;
- le type de la panne : '**failureType**' ;
- la durée de la panne : '**duration**'.

Le type de panne peut prendre trois valeurs : '**move**', '**spreader**', ou **both**. Le premier type concerne une panne empêchant au chariot cavalier de se déplacer alors que le second type concerne une panne du système de manutention. Le dernier type correspond à l'association des deux pannes.

```
<event
  time="00:14:00"
  type="straddleCarrierFailure"
  straddleCarrierID="straddleCarrier_1"
  failureType="both"
  repairDuration ="00:15:00"
/>
```

**Figure IV.22:** Exemple de code XML de description d'une panne d'un chariot cavalier

#### IV.4.8.6 Panne d'une borne laser

Les pannes des bornes laser sont décrites par la valeur '**laserHeadFailure**' du paramètre '**type**'. Trois autres paramètres sont également nécessaires :

- l'identifiant de la borne concernée par la panne : **laserHeadID** ;
- le taux de fonctionnement de la borne : '**range**'. Un taux de 0 indique une panne complète de la borne. Un taux supérieur à 1 indique une élévation de la portée ;
- la durée de la panne : '**duration**'.

```

<event
  time="00:07:00"
  type="laserHeadFailure"
  laserHeadID="1"
  range="0.1"
  duration="00:10:00"
/>

```

**Figure IV.23:** Exemple de code XML de description de panne d'une borne laser qui fonctionnera à 10% de sa capacité pendant 10 minutes

Les données sont donc structurées de façon hiérarchique grâce au format XML, ce qui permet de les lire rapidement et de générer les objets correspondant dans le simulateur. Le moteur de gestion de ces données est l'une des parties du logiciel.

## IV.5 Architecture logicielle

### IV.5.1 Modularité

Le simulateur développé a été écrit en JAVA. Ce langage permet une grande flexibilité à la fois dans le développement (langage objet) et dans l'utilisation puisqu'il est multi-plateformes. Le programme a été conçu pour être modulaire et permettre ainsi de développer des parties du logiciel sans avoir besoin de modifier de façon conséquente le noyau de l'application. Cette architecture permet une grande souplesse dans l'évolution du projet, mais requiert une certaine discipline de développement voire, dans certain cas, une baisse de performance du programme lors de l'exécution. Néanmoins, le système simulé étant complexe et lui même décomposé en sous-systèmes, un découpage de l'application en modules apparaît primordial.

### IV.5.2 Distribution

Cette architecture modulaire a facilité la distribution de l'application permettant ainsi de répartir la charge de calculs sur différentes machines. La technologie RMI (*Remote Method Invocation*) est utilisée afin d'appeler les méthodes des objets distants. Ainsi chaque objet distribuable possède une interface (au sens Java du terme) permettant aux autres objets de pouvoir communiquer avec cet objet distant. La distribution de l'application est paramétrable grâce à un fichier de configuration écrit en XML, de la même façon que pour les données du simulateur (voir IV.4 p160). Ce fichier comporte les informations nécessaires au lancement du simulateur.

#### IV.5.2.1 Librairies tierces

Les chemins vers les librairies nécessaires au fonctionnement de l'application doivent être décrits dans le *classpath* sur chaque machine utilisée afin que les machines virtuelles Java puissent y accéder.

Ces librairies sont :

- ***Graphstream*** : une librairie permettant de manipuler et de représenter des graphes dynamiques (voir [Dutot et al., 2007]) sous licence CeCILL-C et GNU, disponible à l'adresse <http://graphstream-project.org/>. Cette application est composée de 3 sous-librairies :
  - *gs-core-version.jar* : noyau de l'application ;
  - *gs-algo-version.jar* : librairie comportant les algorithmes les plus répandus (et bien d'autres !) concernant les graphes ;
  - *gs-ui-version.jar* : librairie nécessaire à la représentation graphique des graphes ;
- ***common.io.jar*** : une librairie fournie par *Apache* à cette adresse : <http://commons.apache.org/io>.

#### IV.5.2.2 Répartition des composants

Une fois la configuration système établie il faut avant tout définir le serveur RMI de l'application ainsi que son port. La balise `<networkConfiguration>` remplit cette fonction et comporte deux paramètres : le nom de la machine servant d'annuaire et le port d'accès au service d'annuaire RMI sur cette machine. La troisième étape de configuration réseau consiste à définir la répartition des composants de l'application. Chaque composant est décrit par la balise `<remoteObject>` et prend en paramètre le nom RMI du composant (`rmiBindingName`) et le nom de la machine hôte (`host`) sur laquelle le composant sera distribué.

Il existe 8 composants différents pour le simulateur :

- Le terminal en lui même : '`TerminalImpl`' , c'est le noyau de l'application, il est en très forte interaction avec les autres composants ;
- Les consoles distantes : '`display`' , ce sont des consoles permettant de recevoir un affichage texte distant du déroulement de la simulation ;
- L'interface graphique 2D : '`JTerminal`' , c'est un composant *Swing* utilisant *GraphStream* permettant de dessiner le terminal et son évolution au cours du temps ;
- Le système de localisation laser : '`LaserData`' , c'est le composant qui est chargé de capter les positions des chariots cavaliers et qui renvoie ces données au simulateur ;

- Le moteur du simulateur : '**TimeScheduler**', c'est le composant chargé de la gestion des itérations, le temps étant discrétilisé ;
- La télécommande de gestion du simulateur : '**TimeController**', c'est un composant Swing permettant à l'utilisateur de lancer, arrêter, mettre en pause le simulateur, ainsi que de régler le pas de temps ;
- Le *parser XML* : '**XMLTerminalComponentParser**', c'est le composant permettant de décoder les fichiers XML et de créer les objets qui y sont décrits ;
- Le système d'optimisation : '**MissionScheduler**', c'est le composant chargé d'affecter les missions aux chariots cavaliers.

Il est également possible de définir la graine du générateur de nombres aléatoires grâce à la balise `<random/>`. Cette fonctionnalité permet de pouvoir reproduire à chaque exécution les même valeurs aléatoires et ainsi mettre de côté ce facteur lors des tests des différents algorithmes par exemple.

#### **IV.5.3 Déscripteurs du contenu du terminal**

Enfin, la dernière étape consiste à indiquer les liens vers les fichiers de configuration du système de localisation laser (voir IV.4.5 p166), de configuration du terminal (voir IV.4.1 p161 et IV.4.2 p161) et de configuration des chariots cavaliers (voir IV.4.6 p167).

#### **IV.5.4 Base de données**

EADS/Astrium, partenaire le projet CALAS, a été chargée de développer une vue en trois dimensions du terminal portuaire à conteneurs. Grâce à cette partie de l'application, un utilisateur peut piloter manuellement un ou plusieurs chariots cavaliers. La vue 3D communique avec le simulateur dans le but de représenter les données issue du simulateur dans la vue 3D d'une part, et d'autre part afin que le simulateur intègre les événements déclenchés par la vue 3D. Nous avons travaillé en collaboration afin de fournir les informations sur la position des éléments simulés au cours du temps. Afin de permettre de transmettre ces informations, notre simulateur communique avec une base de données et y stocke des informations de la simulation. Ainsi, le programme est capable d'indiquer à notre partenaire l'évolution du terminal au cours du temps afin qu'ils puissent répercuter les changements de position sur leur application. La connexion à la base de données est configurée dans le fichier de configuration réseau comme décrit dans la figure IV.25.

La lecture et l'écriture de données dans la base est réalisée à travers des scripts PHP hébergés

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<document>
    <!--Serveur RMI-->
    <networkConfiguration hostname="localhost" port="2000"/>
    <!--Terminal-->
    <remoteObject rmiBindingName="TerminalImpl" host="localhost"/>
    <!--Console distante-->
    <remoteObject rmiBindingName="display" host="localhost"/>
    <!--IHM-->
    <remoteObject rmiBindingName="JTerminal" host="localhost" id="JTerminal1"/>
    <!--Système de geolocalisation laser-->
    <remoteObject rmiBindingName="LaserData" host="localhost"/>
    <!--Système de gestion du temps-->
    <remoteObject rmiBindingName="TimeScheduler" host="localhost"/>
    <!--Contrôleur de simulation-->
    <remoteObject rmiBindingName="TimeController" host="localhost"/>
    <!--Parseur XML-->
    <remoteObject rmiBindingName="XMLTerminalComponentParser" host="localhost"/>
    <!--Système d'optimisation-->
    <remoteObject rmiBindingName="MissionScheduler" type="TSPScheduler"
        t="1.0" l="5.0" e="0.0"
        alpha="10.0" beta="10.0" gamma="0.0"
        rho="0.00001" Q="10" sync="5000"
        F1="1.0" F2="1000.0" F3="10.0"
        host="localhost" out="localhost"
    />
    <!--Graine aléatoire-->
    <random seed='100' />
    <!--Système de géolocalisation laser-->
    <laserSystemFile file="xml/bornesTN-LARGE_RANGE.xml"/>
    <!--Configuration du terminal (structure, initialisation et événements)-->
    <terminalFile file="xml/results/10missions/TN_33_0_0.xml"/>
    <!--Straddle Carriers-->
    <clientFile file="xml/results/10missions/4vehicles/vehicles-4.xml"/>
</document>

```

**Figure IV.24:** exemple de code XML nécessaire à la description d'un terminal

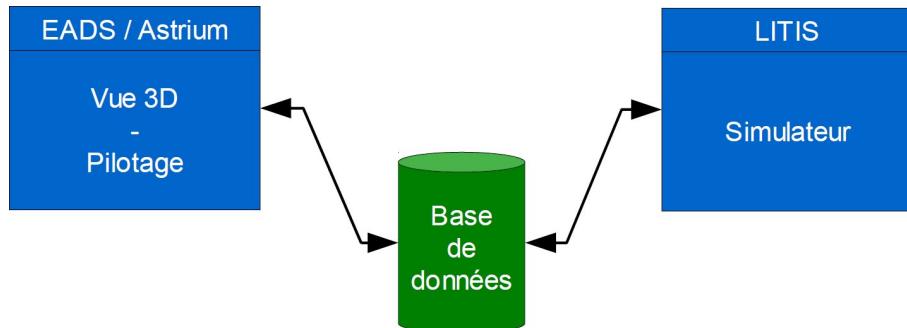
```

<database
    name='dbname',
    server='servername',
    port='serverportnumber',
    user='username',
    password='password'
/>

```

**Figure IV.25:** configuration de l'accès à la base de données du simulateur

sur un serveur web ayant accès à la base puis formate le résultat dans un tableau à l'intérieur d'un fichier XML. La figure IV.27 donne un exemple d'accès en lecture à la base de données.



**Figure IV.26:** Communications entre la vue 3D et le simulateur

```

<?php
try {
    $host = $_POST['host'];
    $dbName = $_POST['dbName'];
    $user = $_POST['user'];
    $password = $_POST['password'];

    $request = $_POST['request'];
    $request = stripslashes($request);
    $pdo_options[PDO::ATTR_ERRMODE] = PDO::ERRMODE_EXCEPTION;
    $bdd = new PDO("mysql:host=$host;dbname=$dbName", $user, $password, $pdo_options);
    $reponse = $bdd->query($request);
?>
<table>
<?php while ($donnees = $reponse->fetch()){ ?>
    <tr>
        <?php
            $i=0;
            while($i < count($donnees)-1) {
        ?>
            <td>
                <?php echo $donnees[$i]; $i=$i+1; ?>
            </td>
            <?php } ?>
        </tr>
        <?php } ?>
    </table>
<?php
    $reponse->closeCursor();
}
catch(Exception $e) {
    die('Erreur : '.$e->getMessage());
}
?>

```

**Figure IV.27:** script PHP réalisant une requête sur la base de données

## IV.6 Modules proposés

Le simulateur développé propose une liste extensible de modules.

### IV.6.1 Module de mobilité des chariots cavaliers

Les chariots peuvent être soit pilotés par une IA, soit par un pilote grâce à un joystick au travers de l'interface 3D de EADS / Astrium. Concernant la partie manuelle, les actions effectuées par le pilote sont inscrites dans la base de données et sont ensuite lues par le simulateur qui reproduit les actions sur le modèle. Aucune vérification d'applicabilité n'est alors effectué puisque cette action est réalisée dans le programme d'EADS / Astrium. L'IA développée dans le module automatique permet de reproduire le comportement du véhicule en terme de déplacements et d'opérations. Les déplacements consistent à respecter le réseau routier et les zones de stockage du terminal ainsi que d'éviter les collisions entre les véhicules. À chaque itération du simulateur, un véhicule contrôlé par l'IA calcule sa future position en fonction de sa position courante, de sa destination et de sa vitesse. La direction à prendre lui est donnée par le module de routage.

### IV.6.2 Module de routage des véhicules

Ce module permet de calculer des plus court chemins entre deux points du terminal. Plusieurs algorithmes sont ainsi implémentés et des interfaces sont proposées afin de permettre d'ajouter facilement d'autres algorithmes.

#### IV.6.2.1 Floyd Warshall : All Pair Shortest Path

Cet algorithme calcule une seule fois tous les plus court chemins et sauvegarde les résultats pour que leur récupération soit rapide. La complexité est en  $O(n^3)$  et le temps d'exécution peut donc être assez long. Cet algorithme est donc inutilisable dans des graphes trop dynamiques.

#### IV.6.2.2 Dijkstra

Cet algorithme calcule les plus court chemins entre un sommet et tous les autres sommets du graphe. Avec un graphe de m arcs et n noeuds la complexité est de  $O[(m + n) * \ln(n)]$ .

#### IV.6.2.3 A \* (A Star)

Il calcule un plus court chemin entre deux sommets du graphe grâce à une heuristique. C'est un algorithme très rapide mais la qualité de la solution ainsi que sa vitesse d'obtention dépend de

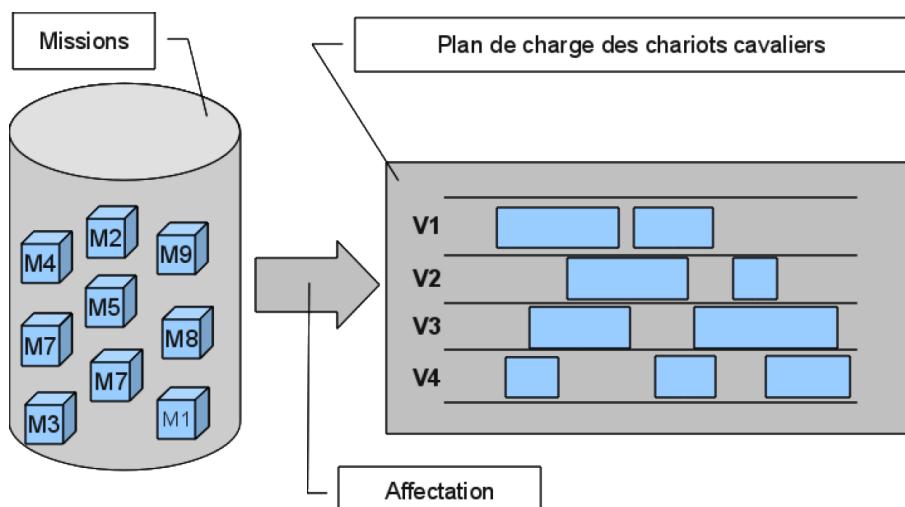
l'adaptation de l'heuristique à la topologie du graphe. La complexité de cet algorithme dépend également de l'heuristique utilisée. Une seule heuristique a été pour le moment implémentée et réalise une estimation du temps de parcours entre deux sommets en calculant le ratio entre la distance (heuristique de Manhattan) et la vitesse du chariot.

#### IV.6.2.4 Dijkstra avec prise en compte des temps d'attente en entrée de travée

Cet algorithme calcule les plus court chemins en temps entre un sommet et tous les autres sommets du graphe avec prise en compte des blocages dans les travées (arcs FIFO) (voir I.2.5 p26) grâce à un mécanisme de réservation de chemins. La complexité de cet algorithme est semblable à celle de Dijkstra.

#### IV.6.3 Module d'ordonnancement des missions

Ce module permet de calculer l'ordonnancement des missions à réaliser sur le terminal. Les missions sont placées dans un pool de tâches à exécuter par les chariots cavaliers. Ces derniers représentent donc des ressources. Le calcul doit tenir compte des fenêtres de temps des missions afin de proposer une solution acceptable pour les clients du terminal (navires, trains ou camions).



**Figure IV.28:** Affectation des missions aux chariots cavaliers

Dans le soucis de réduire les coûts d'exploitation du terminal ainsi que de maintenir une qualité de service suffisante pour les clients, le système doit prendre en compte plusieurs paramètres lors de l'affectation des missions de chargement/déchargement aux chariots cavaliers. Ainsi, la distance à effectuer et le temps de parcours lié à une mission va avoir un impact direct sur le coût de la mission pour le chariot cavalier (consommation de carburant, temps occupé)

ainsi que sur le respect des fenêtres de temps pour le client. En effet, si le chariot cavalier arrive trop tard sur le lieu de collecte ou de livraison, le client devra l'attendre. Cette attente a un coût et le client peut réclamer des indemnités au terminal. En revanche, si le chariot cavalier arrive en avance sur le lieu de collecte/livraison, alors il devra attendre le client devenant ainsi indisponible pour d'autres tâches dans le même temps. Plusieurs algorithmes sont développés afin de répondre à ces problématiques. Ils peuvent être statiques ou dynamiques. Les algorithmes statiques sont performants mais demandent de recalculer entièrement une solution dès qu'une mission est insérée ou supprimée ou qu'une ressource devient indisponible. En revanche, les algorithmes dynamiques proposent des solutions "acceptables" à tout moment.

Chaque algorithme d'ordonnancement et d'affectation présenté dans le chapitre III a été implémenté dans le simulateur. Le module d'ordonnancement contient donc les algorithmes décrits dans le tableau suivant :

Algorithme	rmiBindingName
Branch-and-Bound (voir III.3.1 p118)	BranchAndBoundScheduler
Aléatoire (voir III.3.2.1 p125)	RandomScheduler
Répartition de charge (voir III.3.2.2 p125)	LinearScheduler
Gloutonne : heuristique des plus proches voisins (voir III.3.2.3 p126)	GreedyScheduler
Gloutonne élaborée (voir III.3.2.4 p126)	GreedyOptScheduler
Métaheuristique fourmi hors-ligne (voir III.3.3.1 p127)	OfflineACOScheduler
Variante de la métáheuristique fourmi hors-ligne (voir III.3.3.2 p135)	OfflineACOScheduler2
Métaheuristique fourmi en-ligne (voir III.3.3.3 p137)	OnlineACOScheduler

#### IV.6.4 Module de représentation 2D du terminal

Ce module permet de visualiser en 2 dimensions l'évolution du terminal. Il contient trois parties :

- la vue ;
- la partie informative ;
- la partie contrôle ;
- la partie ordonnancement.

##### IV.6.4.1 Vue

La partie vue représente le graphe routier du terminal, le réseau de stockage, les conteneurs et les chariots cavaliers. Cette vue utilise la librairie GraphStream (voir <http://graphstream-project.org>) permettant de dessiner entièrement ces éléments de façon très simple grâce à des

feuilles de style CSS. Le module développé propose un **Listener** sur les modifications du terminal et permet à l'utilisateur de définir des actions en cas d'ajout/suppression de routes ou de conteneurs, de déplacement de véhicule, etc. De cette façon, à tout moment, la vue 2D représente graphiquement et avec exactitude l'état du système.

#### IV.6.4.2 Informations

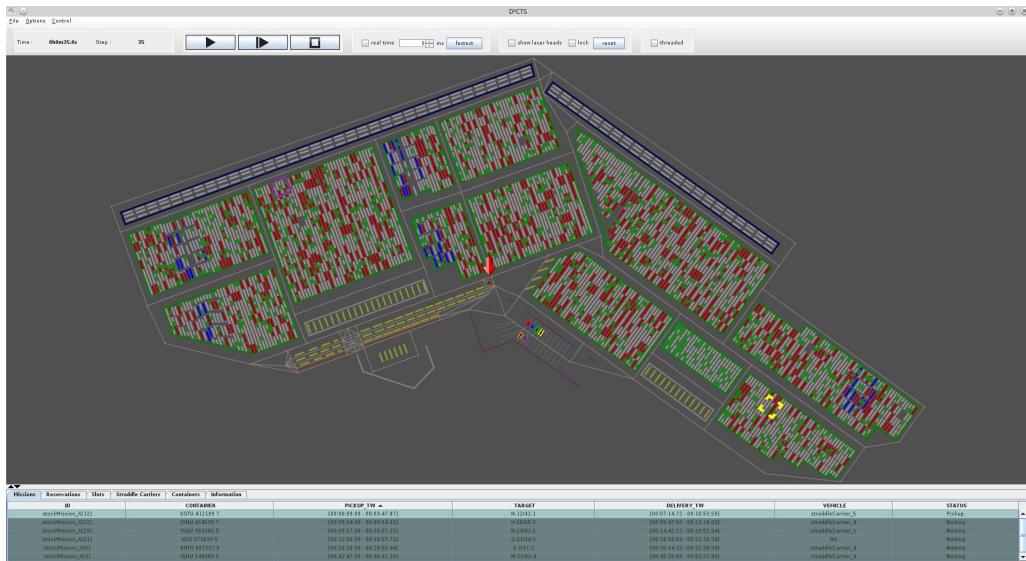
La partie informative du module de représentation 2D contient des informations sous forme de texte à propos de l'état du terminal. On y trouve des informations sur les missions connues du systèmes (conteneur concerné, destination, fenêtres de temps, état, chariot cavalier affecté...), sur les conteneurs (position), sur les véhicules (position, état, plan de charge) ou sur les travées du terminal (contenu des emplacements). Lorsqu'un élément est sélectionné par l'utilisateur dans cette vue, les éléments de la vue 2D concernés par ces informations sont mis en évidence graphiquement. Cette vue est développée en JAVA au travers de la librairie **Swing**.

#### IV.6.4.3 Contrôleur

La partie contrôle concerne le déroulement de la simulation. Elle permet à l'utilisateur de définir le pas de temps, puis de démarrer la simulation. Une fois lancée, il peut également la mettre en pause et avancer itération par itération, relancer la simulation ou quitter le programme. D'autre part, dans le cas où un chariot se trouve dans l'impossibilité de prendre/déposer un conteneur (conteneur bloqué sous un autre, problème d'alignement, problème d'étage, emplacement plein, etc), une boîte de dialogue apparaît et demande à l'utilisateur de régler ce problème. Pour cela, la boîte de dialogue propose des solutions et l'utilisateur n'a plus qu'à faire son choix.

#### IV.6.4.4 Ordonnanceur

La partie ordonnancement concerne la performance des affectations des missions aux chariots cavaliers. Selon les critères établis dans le chapitre précédent, cette performance dépend de la distance couverte par les véhicules, du retard lié au dépassement des fenêtres de temps (*tardiness*), ainsi que de la durée d'attente des véhicules aux points de collecte et de livraison (*earliness*). Lorsque l'algorithme de résolution le défini, des données supplémentaires peuvent être affichées, comme le graphe de résolution par exemple pour la méthode de résolution en-ligne utilisant la métaheuristique fourmi.



**Figure IV.29:** Capture de la partie principale du module graphique du simulateur (en haut le contrôleur, au centre la vue du terminal, en bas la partie informative)

#### IV.6.5 Module de transformation XML (*parser*)

Ce module permet de transformer les données XML concernant la simulation en informations. Il se compose de 3 parties :

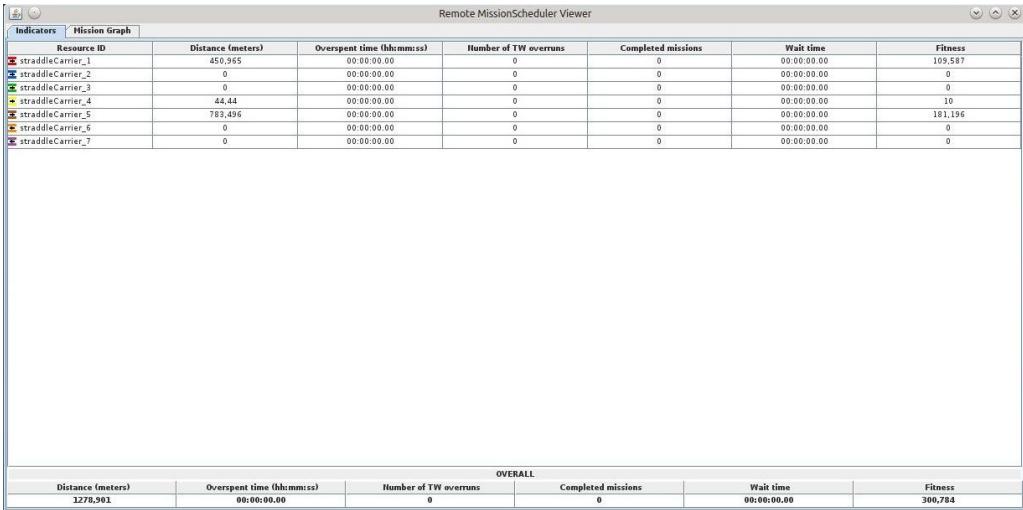
- Le *parser* de configuration réseau : transforme les informations décrites dans le fichier de distribution (voir IV.5.2.2 p174) ;
- Le *parser* de configuration du terminal : transforme selon le format défini (voir IV.5.3 p175) les éléments XML en objets informatiques afin de décrire les composants du terminal ainsi que son contenu ;
- Le *parser* de communication avec les chariots cavaliers : permet de définir les actions à réaliser par l'IA des chariots cavaliers en fonctions de messages XML.

C'est dans ce module que sont définies les règles d'écritures des balises. Un modèle différent peut donc être facilement implémenté en redéfinissant ces classes.

#### IV.6.6 Modules de génération de données

Les générateurs proposés sont des modules permettant de générer des fichiers XML. Il existe pour le moment 3 générateurs de données :

- le générateur d'état initial ;
- le générateur de missions ;
- le générateur d'événements.



**Figure IV.30:** Partie ordonnancement du module graphique du simulateur : informations sur la performance de l'algorithme

#### IV.6.6.1 Générateur d'état initial

Ce générateur permet de créer des conteneurs et de les répartir entre les différents emplacements de stockage du terminal. Il utilise les fichiers de configuration du terminal afin de définir la structure du terminal et de connaître son contenu. Il charge donc le graphe routier et le réseau de stockage avant de commencer à générer les conteneurs. La génération se fait selon différents paramètres. Il faut en effet fournir au générateur le nombre de conteneurs de 20 pieds, de 40 pieds et de 45 pieds à créer. De plus, il faut indiquer le nom de la machine sur laquelle le générateur est lancé ainsi que les fichiers de configuration réseau et de terminal à utiliser. Enfin il faut définir le nom du fichier XML à générer.

#### IV.6.6.2 Générateur de missions

Ce générateur permet de créer des missions pour les chariots cavaliers. Une mission consiste à déplacer un conteneur à l'intérieur du terminal. Selon le type de mission à définir, le générateur va créer ou non un véhicule (train, bateau, ou navire). Il détermine également si la mission à créer sera de type IN ou OUT puis, en fonction du résultat décidera du conteneur à déplacer. Si le conteneur entre dans le terminal (mission de type IN) alors il sera créé. Ensuite le générateur défini un emplacement de destination pour la livraison du conteneur. Une fois ces informations déterminées le générateur détermine les fenêtres de temps de la mission ainsi que les fenêtres de temps des véhicules concernés. Pour cela il se base sur le temps moyen de parcours entre le point de collecte et de livraison du conteneur. Les fenêtres de temps sont ensuite décalées selon



**Figure IV.31:** Partie ordonnancement du module graphique du simulateur : graphe des missions de l'algorithme de résolution en-ligne

une marge de tolérance d'écart de temps. Ce paramètre est fourni en entrée de l'algorithme.

#### IV.6.6.3 Générateurs d'événements

Le générateur d'événements permet de générer un fichier XML contenant des balises `<event></event>` afin de décrire des perturbations sur le système. Il est possible de générer des pannes sur les véhicules, des non respect d'itinéraires par les conducteurs ou de non respect d'affectation de mission. Il est également possible de générer des variations de portée des bornes lasers. Ces événements sont définis en fonction de taux indiqués en entrée du générateur.

L'objectif final de ces générateurs est de pouvoir les configurer en fonction de lois statistiques. Ainsi, les modules de générations sont composés d'interfaces permettant aux utilisateurs de redéfinir les fonctions de générations selon les besoins.

## IV.7 Perspectives

Les deux premières années de travail sur le projet CALAS ont consisté à modéliser les composants d'un terminal portuaire ainsi que leur dynamique. Le simulateur informatique de terminal a été élaboré autour de l'informatisation du plan du terminal de Normandie grâce à une longue phase de collecte de données.

Le simulateur développé a été conçu pour mettre à l'épreuve différentes approches d'ordonnancement et de routage pour obtenir des résultats permettant d'évaluer la qualité des diverses solutions. Toutefois, il est possible d'utiliser le simulateur pour concevoir des méthodes

de résolution à tous les problèmes d'optimisation liés aux terminaux à conteneurs. Il est ainsi possible de mettre à l'épreuve des approches de résolution des problèmes de :

- la structure du terminal ;
- l'allocation des berges aux navires ;
- l'allocation des grues de quai ;
- les plans de chargement des navires ;
- le transbordement ;
- la gestion des conteneurs vides ;
- la gestion des effectifs ;
- le stockage (allocation des travées et des blocs) ;
- les transferts (quai - stockage, stockage - stockage, stockage - terre) ;
- le routage des véhicules ;
- l'allocation des engins de manutention.

Pour le moment, le simulateur n'a été utilisé que pour modéliser les 2 derniers problèmes, néanmoins il est capable de façon intrinsèque de répondre aux autres problématiques.

## Conclusions

*D<sup>2</sup>CTS* est un simulateur de terminal portuaire à conteneurs conçu et développé durant cette thèse dans le cadre du projet CALAS. Il permet de mettre à l'épreuve les méthodes de résolution élaborées également durant cette thèse grâce à un environnement réaliste. Cet outil utilise la technologie JAVA pour fonctionner ainsi qu'une description XML des différents éléments du terminal. Ces deux langages permettent une grande souplesse tant au niveau de l'exécution qu'au niveau des développements futurs. Un certain nombre de fonctionnalités ont été développées avec le simulateur et notamment des modules de routage et d'ordonnancement de missions. Les tests de performance de ce dernier module feront l'objet du chapitre suivant.



# Chapitre V

## Expérimentations et résultats

### Introduction

Cette partie regroupe les différentes expérimentations réalisées grâce au simulateur.

Il s'agit de répondre aux questions suivantes à propos des algorithmes présentés dans le chapitre III.3 :

- les algorithmes se comportent-ils comme prévu ?
- la qualité des solutions calculées est-elle satisfaisante ?
- quelle est l'impact de la dynamicité du problème sur la qualité de la solution calculée ?
- quel est le temps de calcul nécessaire pour déterminer une solution ?

Ces questions portent sur la performance des algorithmes sous plusieurs points de vue. D'abord concernant l'optimisation elle-même, grâce à une fonction d'évaluation (*fitness*) commune à tous les algorithmes la qualité des différentes solutions peuvent être comparées. Puis, en terme de vitesse de convergence, c'est-à-dire du temps de calcul nécessaire pour obtenir une solution proche de la meilleure solution trouvée par l'algorithme il est possible de déterminer si un algorithme trouve des meilleures solution plus rapidement que d'autres méthodes de résolution. Ensuite en terme de robustesse, c'est-à-dire de tolérance à la dynamique, il est possible de comparer la qualité des solutions proposés par les différentes méthodes de résolution en fonction de degrés de dynamicité différents et de déterminer dans quelle mesure les algorithmes répondent aux changements d'environnement. En d'autres termes, nous cherchons à répondre à la question suivante : la solution proposée est-elle proche de l'optimale peu importe les caractéristiques du problème au moment du calcul ? Enfin, dans une moindre mesure, il est question de performance calculatoire, c'est-à-dire du temps CPU nécessaire à l'algorithme pour terminer ses calculs.

Comme discuté dans les chapitres II.3.3 et II.4.2, et peu importe que le problème soit

modélisé sous forme de problème de tournées de véhicules ou de problème d'atelier, il reste NP-Difficile. Cela implique bien-sûr de ne pouvoir calculer la solution optimale dans des problèmes de taille suffisante. Par conséquent, il sera impossible de déterminer si une solution proposée par l'un des algorithmes est optimale ou non. Nous nous baserons alors sur les solutions proposées par les différents algorithmes pour un même problème afin d'établir une comparaison permettant de distinguer les performances de chaque algorithme.

La première partie de ce chapitre présente le protocole de test mis en place afin de répondre aux questions ci-dessus. La seconde partie présente les résultats de l'expérimentation. Enfin, la troisième partie présente une discussion sur les résultats.

## V.1 Protocole de test

Il s'agit de déterminer dans quelle mesure les algorithmes proposés dans les chapitres précédents permettent de résoudre le problème.

La difficulté d'une instance du problème d'ordonnancement et d'affectation des missions aux chariots cavaliers dépend de deux facteurs principaux :

- sa taille : le nombre de tâches à planifier ainsi que le nombre de ressources disponibles.
- sa dynamicité : moins les caractéristiques du problème changent et plus la solution optimale au problème sera facile à atteindre. Cette dynamicité sera déterminée grâce au degré effectif de dynamicité (*edod*) de Larsen (voir II.1.5).

Le problème étant NP-difficile, une première étape sera de s'assurer que pour un problème trivial, une solution optimale est trouvée par chaque algorithme. Cette solution sera calculée par un algorithme du Branch and Bound comme décrit dans la partie III.3.1. Ensuite, pour chaque instance de test, un algorithme glouton permettra de comparer la solution des algorithmes à une solution étalon.

Les méthodes de résolutions implémentées dans le simulateur seront utilisées pour ces tests :

- Algorithme d'évaluation et séparation (voir III.3.1 p.118)
- Algorithme d'affectation aléatoire (voir III.3.2.1 p.125)
- Algorithme de répartition de charge (voir III.3.2.2 p.125)
- Méthode gloutonne élaborée (voir III.3.2.4 p.126)
- Deux méthodes hors-ligne à colonies de fourmis (voir III.3.3.1 p.127 et III.3.3.2 p.135)
- Méthode en-ligne à colonies de fourmis (voir III.3.3.3 p.137)

Les tests consisteront à générer de façon pseudo-aléatoire une série de dix scénarios de tests

de tailles différentes. Pour chaque scénario, des versions avec des degrés différents de dynamicité seront joués. Les métriques utilisées seront celles de la fonction d'évaluation des algorithmes à savoir, la distance parcourue par les chariots cavaliers, la durée de dépassement des fenêtres de temps, le nombre de ces dernières, ainsi que du temps d'attente des chariots cavaliers. Enfin, la durée des calculs nécessaires sera également mesurée afin de déterminer le ratio entre la qualité de la solution proposée et le temps de calcul nécessaire pour l'obtenir.

## V.1.1 Scénarios de test

### V.1.1.1 Taille des instances

Le tableau suivant dresse la liste des scénarios générés par le simulateur. Les scénarios n°1 et n°2 permettent de vérifier que les solutions proposées répondent bien aux problématiques (respectivement) d'ordonnancement et d'affectation. Les solutions proposées par les algorithmes sont comparées à la solution optimale de chaque instance du problème. Dans la même optique, les scénario n°3 et n°4 permettent de mesurer la qualité des solutions fournies par les algorithmes au problème combiné d'ordonnancement et d'affectation. Ici aussi, les solutions proposées seront comparées à la solution optimale. Le scénario n°3 est un problème moins complexe dans la mesure où un nombre suffisant de ressources sont disponibles. Le scénario n°4 permet ainsi de mesurer le comportement des algorithmes en cas de stress lié au manque de ressources tout en conservant la possibilité de comparer le résultat avec la solution optimale. Les scénarios n°5 et n°6 sont des instances moyennes du problème. L'objectif est de mesurer l'impact de l'ajout d'une ressource dans le comportement des algorithmes. Tout comme le scénario n°4, le scénario n°7 permet de contrôler le comportement des algorithmes lorsque le nombre de ressources est faible en comparaison du nombre de tâches à ordonner et planifier. Les scénarios n°8, n°9 et n°10 correspondent aux instances les plus importantes du problème et ont pour but de contrôler les performances techniques des algorithmes vis-à-vis de la taille du problème.

### V.1.1.2 Période de simulation

Il est nécessaire que la période de simulation soit la même pour chaque scénario. Sans cette période commune, il serait impossible de comparer les tailles des instances. En effet, dans le cas dynamique où les missions ne sont pas connues en début de simulation, le scénario n°exécuté sur une période de 9h serait moins complexe à résoudre que le scénario n°3 sur une période de 30 minutes.

<b>ID</b>	<b>Missions</b>	<b>Véhicules</b>	<b>Description</b>
n°1	3	1	instance triviale, sans problème d'affectation
n°2	1	2	instance triviale, sans problème d'ordonnancement
n°3	2	10	peu de missions sont à planifier alors qu'un grand nombre de ressources sont disponibles (comparable à l'optimum).
n°4	5	2	instance de petite taille où une solution optimale est calculable.
n°5	10	4	instance moyenne du problème à petite échelle.
n°6	10	5	instance moyenne du problème permettant de mesurer l'impact de l'ajout d'une ressource dans le calcul des algorithmes : qualité de la solution / performance calculatoire. Les missions sont identiques à celles du scénario 5.
n°7	20	5	instance avec une lourde charge pour les véhicules afin de mesurer la capacité de l'algorithme à résister à des scénarios critiques.
n°8	25	10	instance moyenne à grande échelle.
n°9	30	10	seconde instance moyenne à grande échelle.
n°10	35	10	troisième instance moyenne à grande échelle.

**Table V.1:** Scénarios de tests

Chaque simulation se déroulera donc sur une période de deux heures afin d'être en mesure de déterminer, en fonction des degrés de dynamicité, le comportement des algorithmes en fonction de la taille du problème.

#### V.1.1.3 Degré de dynamicité

Afin de contrôler la réaction des algorithmes vis-à-vis des changements dynamiques surveillant au cours de la journée, il est nécessaire de jouer chaque scénario selon différents degrés (et degrés effectifs) de dynamicité. En fonction d'un degré de dynamicité déterminé, la distribution des missions de chaque scénario décrit dans le tableau ?? diffère. Le tableau ?? présente ces différentes distributions.

## **V.2 Résultats**

## **V.3 Analyse et discussion**

### **Conclusions**

Conclusion sur les épérimentations et résultats

Scenario :

dod	edod	Branch & Bound	Linear	Greedy	Random	Offline	Offline2	Online
0	0							
0.25	0							
0.25	0.25							
0.25	0.5							
0.25	0.75							
0.25	1							
0.5	0							
0.5	0.25							
0.5	0.5							
0.5	0.75							
0.5	1							
0.75	0							
0.75	0.25							
0.75	0.5							
0.75	0.75							
0.75	1							
1	0							
1	0.25							
1	0.5							
1	0.75							
1	1							

**Table V.2:** Distribution de la dynamicité

# **Conclusions**

Conclusions



# Bibliographie

- [Aarts et al., 1994] Aarts, E. H. L., van Laarhoven, P. J. M., Lenstra, J. K., and Ulder, N. L. J. (Spring 1994). A computational study of local search algorithms for job shop scheduling. *ORSA Journal on Computing*, 6(2) :118–125.
- [Adams et al., 1988] Adams, J., Balas, E., and Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Manage. Sci.*, 34(3) :391–401.
- [Ahuja et al., 2003] Ahuja, R. K., Orlin, J. B., Pallottino, S., and Scutellà, M. G. (2003). Dynamic shortest paths minimizing travel times and costs. *NETWORKS*, 41 :205.
- [Akers, 1956] Akers, S. B. (1956). Letter to the editor—a graphical approach to production scheduling problems. *Operations Research*, 4(2) :244–245.
- [Alba and Dorronsoro, 2006] Alba, E. and Dorronsoro, B. (2006). Computing nine new best-so-far solutions for capacitated vrp with a cellular genetic algorithm. *Inf. Process. Lett.*, 98(6) :225–230.
- [Allahverdi et al., 1999] Allahverdi, A., Gupta, J. N., and Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega*, 27(2) :219 – 239.
- [Allahverdi et al., 2008] Allahverdi, A., Ng, C., Cheng, T., and Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3) :985 – 1032.
- [Almeida and Centeno, 1998] Almeida, M. T. and Centeno, M. (1998). A composite heuristic for the single machine early/tardy job scheduling problem. *Computers & Operations Research*, 25(7–8) :625 – 635.
- [Angénol et al., 1988] Angénol, B., de La Croix Vaubois, G., and Texier, J.-Y. L. (1988). Self-organizing feature maps and the travelling salesman problem. *Neural Networks*, 1(4) :289 – 293.

- [Applegate, 2006] Applegate, D. (2006). *The Traveling Salesman Problem : A Computational Study*. Princeton Series in Applied Mathematics. Princeton University Press.
- [Applegate and Cook, 1991] Applegate, D. and Cook, W. (Spring 1991). A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3(2) :149–156.
- [Aras et al., 1999] Aras, N., Oommen, B. J., and Altinel, I. K. (1999). The kohonen network incorporating explicit statistics and its application to the travelling salesman problem. *Neural Netw.*, 12(9) :1273–1284.
- [Baker and Aye chew, 2003] Baker, B. M. and Aye chew, M. (2003). A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 30(5) :787 – 800.
- [Baker, 1974] Baker, K. (1974). *Introduction to sequencing and scheduling*. Wiley.
- [Balas, 1969] Balas, E. (November/December 1969). Machine sequencing via disjunctive graphs : An implicit enumeration algorithm. *Operations Research*, 17(6) :941–957.
- [Balas et al., 1995] Balas, E., Lenstra, J. K., and Vazacopoulos, A. (1995). The one-machine problem with delayed precedence constraints and its use in job shop scheduling. *Management Science*, 41(1) :94–109.
- [Balas and Toth, 1985] Balas, E. and Toth, P. (1985). Branch and bound methods. In et al., E. L., editor, *The Traveling Salesman Problem*, pages 361–401. John Wiley and Sons Ltd : Chichester.
- [Balas and Vazacopoulos, 1998] Balas, E. and Vazacopoulos, A. (1998). Guided local search with shifting bottleneck for job shop scheduling. *Manage. Sci.*, 44(2) :262–275.
- [Baldacci et al., 2008a] Baldacci, R., Battarra, M., and Vigo, D. (2008a). Routing a heterogeneous fleet of vehicles. In Golden, B., Raghavan, S., Wasil, E., Sharda, R., and Voß, S., editors, *The Vehicle Routing Problem : Latest Advances and New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces Series*, pages 3–27. Springer US.
- [Baldacci et al., 2008b] Baldacci, R., Christofides, N., and Mingozzi, A. (2008b). An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115 :351–385. 10.1007/s10107-007-0178-5.
- [Barker and McMahon, 1985] Barker, J. R. and McMahon, G. B. (1985). Scheduling the general job-shop. *Management Science*, 31(5) :594–598.
- [Barnes and Chambers, 1995] Barnes, J. W. and Chambers, J. B. (1995). Solving the job shop scheduling problem with tabu search. *IIE Transactions*, 27(2) :257–263.

- [Barricelli, 1957] Barricelli, N. (1957). *Symbiogenetic Evolution Processes Realized by Artificial Methods*.
- [Beasley, 1983] Beasley, J. E. (1983). Route first–cluster second methods for vehicle routing. *Omega*, 11(4) :403–408.
- [Beck et al., 2003] Beck, J. C., Prosser, P., and Selensky, E. (2003). Vehicle routing and job shop scheduling : What’s the difference ? In *ICAPS*, pages 267–276.
- [Beck et al., 2006] Beck, J. C., Prosser, P., and Selensky, E. (2006). A case study of mutual routing-scheduling reformulation. *J. of Scheduling*, 9(5) :469–491.
- [Bektas, 2006] Bektas, T. (2006). The multiple traveling salesman problem : an overview of formulations and solution procedures. *Omega*, 34(3) :209–219.
- [Bell and McMullen, 2004] Bell, J. E. and McMullen, P. R. (2004). Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics*, 18(1) :41 – 48.
- [Bellman, 1962] Bellman, R. (1962). Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1) :61–63.
- [Bellmore and Hong, 1974] Bellmore, M. and Hong, S. (1974). Transformation of multisalesman problem to the standard traveling salesman problem. *J. ACM*, 21(3) :500–504.
- [Bentley, 1990] Bentley, J. L. (1990). Experiments on traveling salesman heuristics. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, SODA ’90, pages 91–99, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.
- [Benyahia and Potvin, 1998] Benyahia, I. and Potvin, J.-Y. (1998). Decision support for vehicle dispatching using genetic programming. *Systems, Man and Cybernetics, Part A : Systems and Humans, IEEE Transactions on*, 28(3) :306 –314.
- [Berbeglia et al., 2007] Berbeglia, G., Cordeau, J.-F., Gribkovskaia, I., and Laporte, G. (2007). Static pickup and delivery problems : a classification scheme and survey. *TOP : An Official Journal of the Spanish Society of Statistics and Operations Research*, 15(1) :1–31.
- [Berbeglia et al., 2010] Berbeglia, G., Cordeau, J.-F., and Laporte, G. (2010). Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1) :8–15.
- [Berger and Barkaoui, 2003] Berger, J. and Barkaoui, M. (2003). A hybrid genetic algorithm for the capacitated vehicle routing problem. In Cantú-Paz, E., Foster, J., Deb, K., Davis, L., Roy, R., O’Reilly, U.-M., Beyer, H.-G., Standish, R., Kendall, G., Wilson, S., Harman, M.,

- Wegener, J., Dasgupta, D., Potter, M., Schultz, A., Dowsland, K., Jonoska, N., and Miller, J., editors, *Genetic and Evolutionary Computation — GECCO 2003*, volume 2723 of *Lecture Notes in Computer Science*, pages 198–198. Springer Berlin / Heidelberg.
- [Berger and Barkaoui, 2004] Berger, J. and Barkaoui, M. (2004). A parallel hybrid genetic algorithm for the vehicle routing problem with time windows. *Computers & Operations Research*, 31(12) :2037 – 2053.
- [Berger et al., 1998] Berger, J., Salois, M., and Begin, R. (1998). A hybrid genetic algorithm for the vehicle routing problem with time windows. In Mercer, R. and Neufeld, E., editors, *Advances in Artificial Intelligence*, volume 1418 of *Lecture Notes in Computer Science*, pages 114–127. Springer Berlin / Heidelberg.
- [Bierwirth, 1995] Bierwirth, C. (1995). A generalized permutation approach to job shop scheduling with genetic algorithms. *OR Spektrum*, 17 :87–92.
- [Blanton and Wainwright, 1993] Blanton, Jr., J. L. and Wainwright, R. L. (1993). Multiple vehicle routing with time and capacity constraints using genetic algorithms. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 452–459, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [Blazewicz et al., 1996] Blazewicz, J., Domschke, W., and Pesch, E. (1996). The job shop scheduling problem : Conventional and new solution techniques. *European Journal of Operational Research*, 93(1) :1–33.
- [Bodin et al., 1983] Bodin, L., Golden, B., Assad, A., and Ball, M. (1983). Routing and scheduling of vehicles and crews - the state of the art. volume 10, pages 63–212. Computers and Opertions Research.
- [Bramel and Simchi-Levi, 2001] Bramel, J. and Simchi-Levi, D. (2001). The vehicle routing problem. chapter Set-covering-based algorithms for the capacitated VRP, pages 85–108. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.
- [Bräysy, 2003] Bräysy, O. (2003). A reactive variable neighborhood search for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 15 :347–368.
- [Brucker, 2007] Brucker, P. (2007). *Scheduling Algorithms*. Springer, 5th edition.
- [Brucker and Schlie, 1990] Brucker, P. and Schlie, R. (1990). Job-shop scheduling with multi-purpose machines. *Computing*, 45 :369–375. 10.1007/BF02238804.
- [Budinich and Rosario, 1996] Budinich, M. and Rosario, B. (1996). A self-organizing neural

network for the traveling salesman problem that is competitive with simulated annealing. *Neural Computation*, 8 :416–424.

[Bullnheimer et al., 1997a] Bullnheimer, B., Hartl, R. F., and Strauss, C. (1997a). Applying the ant system to the vehicle routing problem.

[Bullnheimer et al., 1997b] Bullnheimer, B., Hartl, R. F., and Strauss, C. (1997b). An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research*, 89 :319–328.

[Bullnheimer et al., 1997c] Bullnheimer, B., Hartl, R. F., and Strauß, C. (1997c). A new rank based version of the ant system - a computational study. *Central European Journal for Operations Research and Economics*, 7 :25–38.

[Caricato et al., 2003] Caricato, P., Ghiani, G., Grieco, A., and Guerriero, E. (2003). Parallel tabu search for a pickup and delivery problem under track contention. *Parallel Comput.*, 29(5) :631–639.

[Carlier and Pinson, 1989] Carlier, J. and Pinson, E. (1989). An algorithm for solving the job-shop problem. *Manage. Sci.*, 35(2) :164–176.

[Carlier and Pinson, 1990] Carlier, J. and Pinson, E. (1990). A practical use of Jackson's preemptive schedule for solving the Job-Shop problem.

[Carlier and Pinson, 1994] Carlier, J. and Pinson, E. (1994). Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, 78(2) :146 – 161. *je :title;Project Management and Scheduling;ce :title;*.

[Carter and Ragsdale, 2006] Carter, A. E. and Ragsdale, C. T. (2006). A new approach to solving the multiple traveling salesperson problem using genetic algorithms. *European Journal of Operational Research*, 175(1) :246 – 257.

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

[Chia-Ho and Ching-Jung, 2005] Chia-Ho, C. and Ching-Jung, T. (2005). A hybrid ant colony system for vehicle routing problem with time windows. *Journal of the Eastern Asia Society for Transportation Studies*, 6 :2822–2836.

[Christofides, 1976] Christofides, N. (1976). Worst-case analysis of a new heuristic for the traveling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University.

- [Christofides et al., 1981] Christofides, N., Mingozzi, A., and Toth, P. (1981). Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical Programming*, 20(1) :255–282.
- [Chu and Beasley, 1997] Chu, P. and Beasley, J. (1997). A genetic algorithm for the generalised assignment problem. *Computers & Operations Research*, 24(1) :17 – 23.
- [Clarke and Wright, 1964] Clarke, G. and Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4) :pp. 568–581.
- [Colorni et al., 1994] Colorni, A., Dorigo, M., Maniezzo, V., and Trubian, M. (1994). Ant System for job-shop scheduling. *Belgian Journal of Operations Research, Statistics and Computer Science*, 34(1) :39–53.
- [Cordeau et al., 1997] Cordeau, J. F., Gendreau, M., and Laporte, G. (1997). A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30(2) :105–119.
- [Cordeau and Laporte, 2001] Cordeau, J. F. and Laporte, G. (2001). A unified tabu search heuristic for vehicle routing problems with time windows.
- [Cornuejols and Harche, 1993] Cornuejols, G. and Harche, F. (1993). Polyhedral study of the capacitated vehicle routing problem. *Mathematical Programming*, 60 :21–52. 10.1007/BF01580599.
- [Croce et al., 1995] Croce, F. D., Tadei, R., and Volta, G. (1995). A genetic algorithm for the job shop problem. *Computers & Operations Research*, 22(1) :15 – 24. jce :title;Genetic Algorithms;ce :title;.
- [Dantzig, 1963] Dantzig, G. (1963). *Linear Programming and Extensions*. Princeton University Press.
- [Dantzig et al., 1954] Dantzig, G., Fulkerson, R., and Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Operations Research*, 2 :393–410.
- [Dantzig and Ramser, 1959] Dantzig, G. B. and Ramser, J. H. (1959). The Truck Dispatching Problem. *Management Science*, 6(1) :80–91.
- [Dauzère-Pérès and Lasserre, 1993] Dauzère-Pérès, S. and Lasserre, J.-B. (1993). A modified shifting bottleneck procedure for job-shop scheduling. *International Journal of Production Research*, 31(4) :923–932.
- [Davis, 1985] Davis, L. (1985). Job shop scheduling with genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 136–140, Hillsdale, NJ, USA. L. Erlbaum Associates Inc.

- [Dell'Amico and Trubian, 1993] Dell'Amico, M. and Trubian, M. (1993). Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, 41 :231–252. 10.1007/BF02023076.
- [Deneubourg et al., 1983] Deneubourg, J., Pasteels, J., and Verhaeghe, J. (1983). Probabilistic behaviour in ants : A strategy of errors ? *Journal of Theoretical Biology*, 105(2) :259 – 271.
- [Doerner et al., 2005] Doerner, K., Hartl, R., and Lucka, M. (2005). A parallel version of the d-ant algorithm for the vehicle routing problem. *International Workshop on Parallel Numerics*, pages 20–23.
- [Doerner et al., 2000] Doerner, K., Hartl, R., and Reimann, M. (2000). Ant colony optimization applied to the pickup and delivery problem. *Management*, (April).
- [Doerner et al., 2001] Doerner, K., Hartl, R. F., and Reimann, M. (2001). Are competants more competent for problem solving ? - the case of a multiple objective transportation problem.
- [Dorigo, 1992] Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy.
- [Dorigo and Blum, 2005] Dorigo, M. and Blum, C. (2005). Ant colony optimization theory : A survey. *Theoretical Computer Science*, 344(2-3) :243–278.
- [Dorigo and Gambardella, 1997] Dorigo, M. and Gambardella, L. M. (1997). Ant Colony System : A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, 1(1) :53–66.
- [Dorigo et al., 1991] Dorigo, M., Maniezzo, V., and Colomi, A. (1991). Positive feedback as a search strategy. Technical report.
- [Dorndorf and Pesch, 1995] Dorndorf, U. and Pesch, E. (1995). Evolution based learning in a job shop scheduling environment. *Computers & Operations Research*, 22(1) :25 – 40. jce :title;Genetic Algorithmsj/ce :titlej.
- [Durbin et al., 1989] Durbin, R., Szeliski, R., and Yuille, A. (1989). An analysis of the elastic net approach to the traveling salesman problem. *Neural Comput.*, 1(3) :348–358.
- [Durbin and Willshaw, 1987] Durbin, R. and Willshaw, D. (1987). An analogue approach to the travelling salesman problem using an elastic net method. *Nature*, 326 :689–691.
- [Dutot et al., 2007] Dutot, A., Guinand, F., Olivier, D., and Pigné, Y. (2007). Graphstream : A tool for bridging the gap between complex systems and dynamic graphs. In *EPNACS : Emergent Properties in Natural and Artificial Complex Systems*.

- [Favata and Walker, 1991] Favata, F. and Walker, R. (1991). A study of the application of kohonen-type neural networks to the travelling salesman problem. *Biological Cybernetics*, 64 :463–468. 10.1007/BF00202610.
- [Feo and Resende, 1989] Feo, T. A. and Resende, M. G. C. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2) :67 – 71.
- [Fisher, 1994] Fisher, M. L. (July/August 1994). Optimal solution of vehicle routing problems using minimum k-trees. *Operations Research*, 42(4) :626–642.
- [Fisher and Jaikumar, 1981] Fisher, M. L. and Jaikumar, R. (1981). A generalized assignment heuristic for vehicle routing. *Networks*, 11(2) :109–124.
- [Fleszar et al., 2009] Fleszar, K., Osman, I. H., and Hindi, K. S. (2009). A variable neighbourhood search algorithm for the open vehicle routing problem. *European Journal of Operational Research*, 195(3) :803 – 809.
- [Florian et al., 1971] Florian, M., Trepant, P., and McMahon, G. (1971). An implicit enumeration algorithm for the machine sequencing problem. *Management Science*, 17(12) :B–782–B–792.
- [Floyd, 1962] Floyd, R. W. (1962). Algorithm 97 : Shortest path. *Commun. ACM*, 5(6) :345–.
- [Fogel, 1990] Fogel, D. B. (1990). A parallel processing approach to a multiple traveling salesman problem using evolutionary programming. In *Proceedings on the Fourth Annual Parallel Processing Symposium*, pages 318–326.
- [França et al., 1995] França, P. M., Gendreau, M., Laporte, G., and Müller, F. M. (1995). The m-traveling salesman problem with minmax objective. *Transportation Science*, 29(3) :267–275.
- [Fraser, 1957] Fraser, A. S. (1957). Simulation of genetic systems by automatic digital computers. II. Effects of linkage on rates of advance under selection. *Australian Journal of Biological Science*, 10 :492–499.
- [French, 1982] French, S. (1982). *Sequencing and scheduling : an introduction to the mathematics of the job-shop*. Ellis Horwood series in mathematics and its applications. E. Horwood.
- [Fukasawa et al., 2006] Fukasawa, R., Longo, H., Lysgaard, J., Aragão, M. P. d., Reis, M., Uchoa, E., and Werneck, R. F. (2006). Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming*, 106 :491–511. 10.1007/s10107-005-0644-x.

- [Funke et al., 2005] Funke, B., Grünert, T., and Irnich, S. (2005). Local search for vehicle routing and scheduling problems : Review and conceptual integration. *Journal of Heuristics*, 11 :267–306. 10.1007/s10732-005-1997-2.
- [Gajpal and Abad, 2008] Gajpal, Y. and Abad, P. L. (2008). Multi-ant colony system (macs) for a vehicle routing problem with backhauls. *European Journal of Operational Research*, 196(1) :102–117.
- [Gajpal and Abad, 2009] Gajpal, Y. and Abad, P. L. (2009). An ant colony system (acs) for vehicle routing problem with simultaneous delivery and pickup. *Computers & OR*, 36(12) :3215–3223.
- [Gambardella et al., 1995] Gambardella, L. M., Dorigo, M., and Bruxelles, U. L. D. (1995). Ant-q : A reinforcement learning approach to the traveling salesman problem. pages 252–260. Morgan Kaufmann.
- [Gambardella et al., 1999] Gambardella, L. M., Taillard, E., and Agazzi, G. (1999). Macs-vrptw : A multiple ant colony system for vehicle routing problems with time windows. Technical report.
- [Ganesh and Narendran, 2007] Ganesh, K. and Narendran, T. (2007). Cloves : A cluster-and-search heuristic to solve the vehicle routing problem with delivery and pick-up. *European Journal of Operational Research*, 178(3) :699 – 717.
- [Gardner, 1970] Gardner, M. (1970). The fantastic combinations of John Conway’s new solitaire game “life”. *Scientific American*, 223 :120–123.
- [Gavish and Srikanth, 1986] Gavish, B. and Srikanth, K. (1986). An optimal solution method for large-scale multiple traveling salesmen problems. *Oper. Res.*, 34(5) :698–717.
- [Gehring and Homberger, 1999] Gehring, H. and Homberger, J. (1999). A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In *Proceedings of EUROGEN99–Short Course on Evolutionary Algorithms in Engineering and Computer Science*, pages 57–64.
- [Gehring and Homberger, 2002] Gehring, H. and Homberger, J. (2002). Parallelization of a two-phase metaheuristic for routing problems with time windows. *Journal of Heuristics*, 8 :251–276. 10.1023/A :1015053600842.
- [Gendreau et al., 1994] Gendreau, M., Hertz, A., and Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10) :1276–1290.

- [Gendreau et al., 1996] Gendreau, M., Laporte, G., and Seguin, R. (1996). Stochastic vehicle routing. *European Journal of Operational Research*, 88(1) :3–12.
- [Gendreau et al., 2008] Gendreau, M., Potvin, J. Y., Bräysy, O., Hasle, G., and Løkketangen, A. (2008). Metaheuristics for the vehicle routing problem and its extensions : a categorized bibliography. In Golden, B., Raghavan, S., and Wasil, E., editors, *The Vehicle Routing Problem - Latest Advances and New Challenges*. Springer Verlag, Heidelberg.
- [Gere, 1966] Gere, W. S. (1966). Heuristics in job shop scheduling. *Management Science*, 13(3) :167–190.
- [Geurts, 1998] Geurts, F. (1998). Ants'98 - from ant colonies to artificial ants : First international workshop on ant colony optimization. *AI Commun.*, 11(3,4) :241–242.
- [ghazali Talbi, 2001] ghazali Talbi, E. (2001). Métaheuristiques pour l'optimisation combinatoire multi-objectif : Etat de l'art. Technical report.
- [Gillett and Miller, 1974] Gillett, B. E. and Miller, L. R. (March/April 1974). A heuristic algorithm for the vehicle-dispatch problem. *Operations Research*, 22(2) :340–349.
- [Glover, 1989] Glover, F. (1989). Tabu Search — Part I. *ORSA Journal on Computing*, 1(3) :190–206.
- [Glover, 1990] Glover, F. (1990). Tabu search - part ii. *INFORMS Journal on Computing*, 2(1) :4–32.
- [Glover and McMillan, 1986] Glover, F. and McMillan, C. (1986). The general employee scheduling problem : an integration of ms and ai. *Comput. Oper. Res.*, 13(5) :563–573.
- [Golden et al., 1977] Golden, B. L., Magnanti, T. L., and Nguyen, H. Q. (1977). Implementing vehicle routing algorithms. *Networks*, 7(2) :113–148.
- [Gonçalves et al., 2005] Gonçalves, J. F., de Magalhães Mendes, J. J., and Resende, M. G. (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, 167(1) :77 – 95.
- [Goss et al., 1989] Goss, S., Aron, S., Deneubourg, J., and Pasteels, J. (1989). Self-organized shortcuts in the Argentine ant. *Naturwissenschaften*, 76(12) :579–581.
- [Graham et al., 1979] Graham, R., Lawler, E., Lenstra, J., and Rinnooy (1979). Optimization and approximation in deterministic sequencing and scheduling : a survey. *Annals of discrete mathematics*, 5 :287–326.

- [Grassé, 1959] Grassé, P.-P. (1959). La reconstruction du nid et les coordinations interindividuelles chez bellicositermes natalensis et cubitermes sp. la théorie de la stigmergie : essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6.
- [GuoXing, 1995] GuoXing, Y. (1995). Transformation of multidepot multisalesmen problem to the standard travelling salesman problem. *European Journal of Operational Research*, 81(3) :557 – 560.
- [Gutjahr, 2000] Gutjahr, W. J. (2000). A graph-based ant system and its convergence. *Future Generation Comp. Syst.*, 16(8) :873–888.
- [Hansen et al., 2010] Hansen, P., Mladenović, N., Brimberg, J., and Pérez, J. A. M. (2010). Variable neighborhood search. In Gendreau, M., Potvin, J.-Y., Hillier, F. S., and Price, C. C., editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 61–86. Springer US.
- [Hefetz and Adiri, 1982] Hefetz, N. and Adiri, I. (1982). An efficient optimal algorithm for the two-machines unit-time jobshop schedule-length problem. *Mathematics of Operations Research*, (7) :354–360.
- [Held and Karp, 1961] Held, M. and Karp, R. M. (1961). A dynamic programming approach to sequencing problems. In *Proceedings of the 1961 16th ACM national meeting*, ACM '61, pages 71.201–71.204, New York, NY, USA. ACM.
- [Holland, 1975] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*, volume Ann Arbor. University of Michigan Press.
- [Homberger and Gehring, 2005] Homberger, J. and Gehring, H. (2005). A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. *European Journal of Operational Research*, 162(1) :220 – 238. jce :title;Logistics : From Theory to Application;ce :title;.
- [Homberger et al., 1999] Homberger, J., Gehring, H., Hagen, F., Wirtschaftsinformatik, L., D-Hagen, and Deutschland, B. (1999). Two evolutionary metaheuristics for the vehicle routing problem with time windows. *INFOR*, 37 :297–318.
- [Hopfield and Tank, 1985] Hopfield, J. and Tank, D. (1985). Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52 :141–152.
- [Hopfield, 1982] Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8) :2554–2558.

- [Hsu et al., 1991] Hsu, C. Y., Tsai, M. H., and Chen, W. M. (1991). A study of feature-mapped approach to the multiple travelling salesmen problem. In *Proc. Int. Symp. on Circuits and Systems*, volume II, pages 1589–1592, Piscataway, NJ. IEEE Service Center.
- [Huang and Chu, 2003] Huang, W.-C. and Chu, C.-Y. (2003). Cost comparison of straddle carrier direct and relay systems in container terminals. *Journal of Marine Science and Technology*, 11(4) :197–204.
- [Irnich et al., 2006] Irnich, S., Funke, B., and Grünert, T. (2006). Sequential search and its application to vehicle-routing problems. *Computers & Operations Research*, 33(8) :2405 – 2429.
- [J et al., 1992] J, G., J, P., and I, B. (1992). Exact solution of multiple traveling salesman problems. In *Combinatorial optimization*, volume F82 of *NATO ASI Series*, pages 291–292.
- [J.-F. and G., 2003] J.-F., C. and G., L. (2003). A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B : Methodological*, 37(6) :579–594.
- [Jackson, 1956] Jackson, J. R. (1956). An extension of johnson’s results on job-lot scheduling. *Naval Research Logistics Quarterly*, (3) :201–224.
- [Jain and Meeran, 1998] Jain, A. S. and Meeran, S. (1998). A state-of-the-art review of job-shop scheduling techniques.
- [James, 1997] James, R. (1997). Using tabu search to solve the common due date early/tardy machine scheduling problem. *Computers & Operations Research*, 24(3) :199 – 208.
- [Jaskiewicz and Kominek, 2003] Jaskiewicz, A. and Kominek, P. (2003). Genetic local search with distance preserving recombination operator for a vehicle routing problem. *European Journal of Operational Research*, 151(2) :352 – 364. *je :title; Meta-heuristics in combinatorial optimization;ce :title;*
- [Jih and Yung-Jen Hsu, 1999] Jih, W.-R. and Yung-Jen Hsu, J. (1999). Dynamic vehicle routing using hybrid genetic algorithms. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 1, pages 453 –458 vol.1.
- [Johnson, 1990] Johnson, D. S. (1990). Local optimization and the traveling salesman problem. In *Proceedings of the seventeenth international colloquium on Automata, languages and programming*, pages 446–461, New York, NY, USA. Springer-Verlag New York, Inc.
- [Johnson and Mcgeoch, 1997] Johnson, D. S. and Mcgeoch, L. A. (1997). *The Traveling Salesman Problem : A Case Study in Local Optimization.*

- [Johnson, 1954] Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1) :61–68.
- [Jonker and Volgenant, 1988] Jonker, R. and Volgenant, T. (1988). An improved transformation of the symmetric multiple traveling salesman problem. *Oper. Res.*, 36(1) :163–167.
- [Junjie and Dingwei, 2006] Junjie, P. and Dingwei, W. (2006). An ant colony optimization algorithm for multiple travelling salesman problem. In *Proceedings of the First International Conference on Innovative Computing, Information and Control - Volume 1*, ICICIC '06, pages 210–213, Washington, DC, USA. IEEE Computer Society.
- [Kahng and Reda, 2004] Kahng, A. B. and Reda, S. (2004). Match twice and stitch : a new tsp tour construction heuristic. *Operations Research Letters*, 32(6) :499 – 509.
- [Kilby et al., 1998] Kilby, P., Prosser, P., and Shaw, P. (1998). Dynamic vrps : A study of scenarios. Technical report.
- [Kim and Park, 2004] Kim, K. H. and Park, Y.-M. (2004). A crane scheduling method for port container terminals. *European Journal of Operational Research*, 156(3) :752 – 768.
- [Király and Abonyi, 2011] Király, A. and Abonyi, J. (2011). Optimization of multiple traveling salesmen problem by a novel representation based genetic algorithm. In *Intelligent Computational Optimization in Engineering*, pages 241–269.
- [Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220 :671–680.
- [Kohonen, 1982] Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43 :59–69. 10.1007/BF00337288.
- [Kontoravdis and Bard, 1995] Kontoravdis, G. and Bard, J. F. (Winter 1995). A grasp for the vehicle routing problem with time windows. *ORSA Journal on Computing*, 7(1) :10–23.
- [Kruskal, 1956] Kruskal, J. B. (1956). On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings of the American Mathematical Society*, 7(1) :48–50.
- [Kubiak, 2004] Kubiak, M. (2004). Systematic construction of recombination operators for the vehicle routing problem. *Foundations of Computing and Decision Sciences*, 29(3).
- [Kytöjoki et al., 2007] Kytöjoki, J., Nuortio, T., Bräysy, O., and Gendreau, M. (2007). An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. *Computers & Operations Research*, 34(9) :2743 – 2757.
- [Lageweg et al., 1977] Lageweg, B. J., Lenstra, J. K., and Rinnooy Kan, A. H. G. (1977). Jobshop scheduling by implicit enumeration. *Management Science*, 24(4) :441–450.

- [Laporte, 1992] Laporte, G. (1992). The vehicle routing problem : An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3) :345 – 358.
- [Laporte et al., 2000] Laporte, G., Gendreau, M., Potvin, J.-Y., and Semet, F. (2000). Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research*, 7(4-5) :285–300.
- [Laporte and Nobert, 1980] Laporte, G. and Nobert, Y. (1980). A Cutting Planes Algorithm for the m-Salesmen Problem. *The Journal of the Operational Research Society*, 31(11) :1017–1023.
- [Laporte et al., 1985] Laporte, G., Nobert, Y., and Desrochers, M. (September/October 1985). Optimal routing under capacity and distance restrictions. *Operations Research*, 33(5) :1050–1073.
- [Laporte et al., 1988] Laporte, G., Nobert, Y., and Taillefer, S. (1988). Solving a family of multi-depot vehicle routing and location-routing problems. *Transportation Science*, 22(3) :161–172.
- [Larrañaga et al., 1999] Larrañaga, P., Kuijpers, C., Murga, R., Inza, I., and Dizdarevic, S. (1999). Genetic algorithms for the travelling salesman problem : A review of representations and operators. *Artificial Intelligence Review*, 13 :129–170. 10.1023/A :1006529012972.
- [Larsen, 2000] Larsen, A. (2000). *The Vehicle Routing Problem*. PhD thesis, Department of Mathematical Modelling, Technical University of Denmark.
- [Larsen et al., 2008] Larsen, A., Madsen, O. B., and Solomon, M. M. (2008). Recent developments in dynamic vehicle routing systems. In Golden, B., Raghavan, S., Wasil, E., Sharda, R., and Voß, S., editors, *The Vehicle Routing Problem : Latest Advances and New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces Series*, pages 199–218. Springer US.
- [Lenstra and Kan, 1979] Lenstra, J. and Kan, A. R. (1979). Computational complexity of discrete optimization problems. In P.L. Hammer, E. J. and Korte, B., editors, *Discrete Optimization I Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium*, volume 4 of *Annals of Discrete Mathematics*, pages 121 – 140. Elsevier.
- [Leung et al., 2004] Leung, K.-S., Jin, H., and Xu, Z. (2004). An expanding self-organizing neural network for the traveling salesman problem. *Neurocomputing*, 62 :267–292.
- [Li and Tian, 2006] Li, X. and Tian, P. (2006). An ant colony system for the open vehicle routing problem. In Dorigo, M., Gambardella, L., Birattari, M., Martinoli, A., Poli, R., and

- Stützle, T., editors, *Ant Colony Optimization and Swarm Intelligence*, volume 4150 of *Lecture Notes in Computer Science*, pages 356–363. Springer Berlin / Heidelberg.
- [Lin and Kernighan, 1973] Lin, S. and Kernighan, B. W. (1973). An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Operations Research*, 21(2) :498–516.
- [Lin et al., 1997] Lin, S.-C., Goodman, E. D., and III, W. F. P. (1997). A genetic algorithm approach to dynamic job shop scheduling problem. In *ICGA*, pages 481–488.
- [Little et al., 1963] Little, J., Murty, K., Sweeney, D., and Karel, C. (1963). An algorithm for the travelling salesman problem. *Operations Research*, 11 :972–989.
- [Lund et al., 1996] Lund, K., Oli, and Rygaard, J. M. (1996). Vehicle Routing Problems with Varying Degrees of Dynamism.
- [M. et al., 1999] M., G., G., L., C., M., and E.D., T. (1999). A tabu search heuristic for the heterogeneous fleet vehicle routing problem. *Computers and Operations Research*, 26(12) :1153–1173.
- [Madsen et al., 1995] Madsen, O., Ravn, H., and Rygaard, J. (1995). A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. *Annals of Operations Research*, 60 :193–208. 10.1007/BF02031946.
- [Madureira et al., 2001] Madureira, A., Ramos, C., and do Carmo Silva, S. (2001). A genetic approach for dynamic job-shop scheduling problems.
- [Malek et al., 1989] Malek, M., Guruswamy, M., Pandya, M., and Owens, H. (1989). Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. *Annals of Operations Research*, 21 :59–84. 10.1007/BF02022093.
- [Martin et al., 1991] Martin, O., Otto, S. W., and Felten, E. W. (1991). Large-step markov chains for the traveling salesman problem. *Complex Systems*, 5 :299–326.
- [Matsuo et al., 1989] Matsuo, H., Juck SUH, C., and Sullivan, R. S. (1989). A controlled search simulated annealing method for the single machine weighted tardiness problem. *Annals of Operations Research*, 21 :85–108. 10.1007/BF02022094.
- [Mazzeo and Loiseau, 2004] Mazzeo, S. and Loiseau, I. (2004). An ant colony algorithm for the capacitated vehicle routing. *Electronic Notes in Discrete Mathematics*, 18(0) :181 – 186. jxocs :full-name;Latin-American Conference on Combinatorics, Graphs and Applicationsjxocs :full-name;.
- [McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5(4) :115–133.

- [Menger, 1928] Menger, K. (1928). Ein theorem über die bogenlänge, anzeiger. In *Mathematisch-naturwissenschaftliche Klasse* 65, page 264–266. Akademie der Wissenschaften in Wien.
- [Mester and Bräysy, 2007] Mester, D. and Bräysy, O. (2007). Active-guided evolution strategies for large-scale capacitated vehicle routing problems. *Computers & Operations Research*, 34(10) :2964 – 2975.
- [Mester et al., 2007] Mester, D., Bräysy, O., and Dullaert, W. (2007). A multi-parametric evolution strategies algorithm for vehicle routing problems. *Expert Systems with Applications*, 32(2) :508 – 517.
- [Metropolis et al., 1953] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6) :1087–1092.
- [Miller et al., 1960] Miller, C. E., Tucker, A. W., and Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *J. ACM*, 7(4) :326–329.
- [Minsky and Papert, 1969] Minsky, M. and Papert, S. (1969). *Perceptrons : An Introduction to Computational Geometry*. The MIT Press.
- [Mitrovic-Minic, 2001] Mitrovic-Minic (2001). *The Dynamic Pickup and Delivery Problem with Time Windows*. PhD thesis, Simon Fraser University.
- [Mitrovic-Minic and Krishnamurti, 2006] Mitrovic-Minic, S. and Krishnamurti, R. (2006). The multiple tsp with time windows : vehicle bounds based on precedence graphs. *Operations Research Letters*, 34(1) :111–120.
- [Mladenović and Hansen, 1997] Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11) :1097 – 1100.
- [Modares et al., 1999] Modares, A., Somhom, S., and Enkawa, T. (1999). A self-organizing neural network approach for multiple traveling salesman and vehicle routing problems. *International Transactions in Operational Research*, 6(6) :591–606.
- [Montemanni et al., 2005] Montemanni, R., Gambardella, L., Rizzoli, A., and Donati, A. (2005). Ant colony system for a dynamic vehicle routing problem. *Journal of Combinatorial Optimization*, 10 :327–343. 10.1007/s10878-005-4922-6.
- [Morton and Pentico, 1993] Morton, T. E. and Pentico, D. W. (1993). *Heuristic Scheduling Systems*. John Wiley Interscience, New York.

- [Muth and Thompson, 1963] Muth, J. and Thompson, G. (1963). *Industrial scheduling*. Prentice-Hall international series in management. Prentice-Hall.
- [Nakano and Yamada, 1991] Nakano, R. and Yamada, T. (1991). Conventional genetic algorithm for job shop problems. In *ICGA*, pages 474–479.
- [Nanry and Barnes, 2000] Nanry, W. P. and Barnes, J. W. (2000). Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Research Part B : Methodological*, 34(2) :107 – 121.
- [Neumann, 1966] Neumann, J. V. (1966). *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA.
- [Nowicki and Smutnicki, 1996] Nowicki, E. and Smutnicki, C. (1996). A fast taboo search algorithm for the job shop problem. *Manage. Sci.*, 42(6) :797–813.
- [Ombuki et al., 2006] Ombuki, B., Ross, B. J., and Hanshar, F. (2006). Multi-objective genetic algorithms for vehicle routing problem with time windows. *APPLIED INTELLIGENCE*, 24 :2006.
- [Orda and Rom, 1990] Orda, A. and Rom, R. (1990). Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *J. ACM*, 37(3) :607–625.
- [Osman, 1993] Osman, I. H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41 :421–451. 10.1007/BF02023004.
- [Ouelhadj and Petrovic, 2009] Ouelhadj, D. and Petrovic, S. (2009). A survey of dynamic scheduling in manufacturing systems. *J. of Scheduling*, 12(4) :417–431.
- [Pankratz, 2005] Pankratz, G. (2005). A grouping genetic algorithm for the pickup and delivery problem with time windows. *OR Spectrum*, 27 :21–41. 10.1007/s00291-004-0173-7.
- [Panwalkar and Iskander, 1977] Panwalkar, S. S. and Iskander, W. (January/February 1977). A survey of scheduling rules. *Operations Research*, 25(1) :45–61.
- [Park, 2001] Park, Y.-B. (2001). A hybrid genetic algorithm for the vehicle scheduling problem with due times and time deadlines. *International Journal of Production Economics*, 73(2) :175 – 188.
- [Pillac et al., 1 10] Pillac, V., Gendreau, M., Guéret, C., and Medaglia, A. (2011-10). A review of dynamic vehicle routing problems. Technical Report CIRRELT-2011-62. 29 pages, available at <https://www.cirrelt.ca/DocumentsTravail/CIRRELT-2011-62.pdf>.

- [Pinson, 1988] Pinson, E. (1988). *Le problme de job-shop*. PhD thesis.
- [Polacek et al., 2004] Polacek, M., Hartl, R. F., Doerner, K., and Reimann, M. (2004). A variable neighborhood search for the multi depot vehicle routing problem with time windows. *Journal of Heuristics*, 10 :613–627. 10.1007/s10732-005-5432-5.
- [Potvin and Bengio, 1996] Potvin, J.-Y. and Bengio, S. (1996). The vehicle routing problem with time windows part ii : Genetic search. *INFORMS Journal on Computing*, 8(2) :165–172.
- [Powell et al., 1995] Powell, W. B., Jaillet, P., and Odoni, A. (1995). Chapter 3 stochastic and dynamic networks and routing. In M.O. Ball, T.L. Magnanti, C. M. and Nemhauser, G., editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, pages 141 – 295. Elsevier.
- [Prim, 1957] Prim, R. C. (1957). Shortest connection networks and some generalizations. *Bell System Technology Journal*, 36 :1389–1401.
- [Prins, 2004] Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12) :1985 – 2002.
- [Psaraftis and of Management, 1985] Psaraftis, H. and of Management, S. S. (1985). *Analysis and Solution Algorithms of Sealift Routing and Scheduling Problems*. Working paper (Sloan School of Management). Massachusetts Institute of Technology, Alfred P. Sloan School of Management.
- [Psaraftis, 1980] Psaraftis, H. N. (1980). A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2) :130–154.
- [Psaraftis, 1988] Psaraftis, H. N. (1988). *Dynamic Vehicle Routing Problem*, pages 223–248. Elsevier Science Publishers B.V.
- [Psaraftis, 1995] Psaraftis, H. N. (1995). Dynamic vehicle routing : Status and prospects. *Annals of Operations Research*, 61 :143–164. 10.1007/BF02098286.
- [Qi et al., 2000] Qi, J. G., Burns, G. R., and Harrison, D. K. (2000). The application of parallel multipopulation genetic algorithms to dynamic job-shop scheduling. *The International Journal of Advanced Manufacturing Technology*, 16 :609–615. 10.1007/s001700070052.
- [Ramasesh, 1990] Ramasesh, R. (1990). Dynamic job shop scheduling : A survey of simulation research. *Omega*, 18(1) :43 – 57.

- [Ray et al., 2007] Ray, S., Bandyopadhyay, S., and Pal, S. (2007). Genetic operators for combinatorial optimization in tsp and microarray gene ordering. *Applied Intelligence*, 26 :183–195. 10.1007/s10489-006-0018-y.
- [Rayward-Smith, 1994] Rayward-Smith, V. (1994). A unified approach to tabu search simulated annealing and genetic algorithms. In *Unicom Seminars Ltd (ed) Adaptive Computing and Information Processing*, volume I, pages 55–78, London. Unicom Seminars Ltd.
- [Reimann et al., 2004] Reimann, M., Doerner, K., and Hartl, R. F. (2004). D-ants : Savings based ants divide and conquer the vehicle routing problem. *Computers & Operations Research*, 31(4) :563 – 591.
- [Ritter et al., 1992] Ritter, H., Martinetz, T., and Schulten, K. (1992). *Neural computation and self-organizing maps : an introduction*. Computation and neural systems series. Addison-Wesley.
- [Rizzoli et al., 2007] Rizzoli, A., Montemanni, R., Lucibello, E., and Gambardella, L. (2007). Ant colony optimization for real-world vehicle routing problems. *Swarm Intelligence*, 1 :135–151. 10.1007/s11721-007-0005-x.
- [Rosenblatt, 1958] Rosenblatt, F. (1958). The Perceptron : probabilistic model for information storage and organization in the brain. *Psychological Review*, 65 :386–408.
- [Russell, 1977] Russell, R. A. (1977). An Effective Heuristic for the M-Tour Traveling Salesman Problem with Some Side Conditions. *Operations Research*, 25(3) :517–524.
- [Ryan et al., 1998] Ryan, J. L., Bailey, T. G., Moore, J. T., and Carlton, W. B. (1998). Reactive tabu search in unmanned aerial reconnaissance simulations. In *Proceedings of the 30th conference on Winter simulation*, WSC '98, pages 873–880, Los Alamitos, CA, USA. IEEE Computer Society Press.
- [S G Ponnambalam and Girish, 2005] S G Ponnambalam, N. J. and Girish, B. S. (2005). An ant colony optimization algorithm for flexible job shop scheduling problem.
- [Savelsbergh and Sol, 1998] Savelsbergh, M. and Sol, M. (1998). Drive : Dynamic Routing of Independent Vehicles. *Operations Research*, 46(4) :474–490.
- [Schmidt, 2001] Schmidt, K. (2001). Using tabu search to solve the job shop scheduling problem with sequence dependent setup times.
- [Schrijver, 2005] Schrijver, A. (2005). On the history of combinatorial optimization (till 1960). In *in Handbooks in Operations Research and Management*. Elsevier.

- [Shmoys et al., 1994] Shmoys, D. B., Stein, C., and Wein, J. (1994). Improved approximation algorithms for shop scheduling problems. *SIAM J. Comput.*, 23(3) :617–632.
- [Somhom et al., 1999] Somhom, S., Modares, A., and Enkawa, T. (1999). Competition-based neural network for the multiple travelling salesmen problem with minmax objective. *Computers & Operations Research*, 26(4) :395 – 407.
- [Song et al., 2003] Song, C.-H., Lee, K., and Lee, W. D. (2003). Extended simulated annealing for augmented tsp and multi-salesmen tsp. In *Proceedings of the international joint conference on neural networks*, volume 3, pages 2340–2343.
- [Song and Lee, 1996] Song, J.-S. and Lee, T.-E. (1996). A tabu search procedure for periodic job shop scheduling. *Computers & Industrial Engineering*, 30(3) :433 – 447. jce :title;IE in Korea|/ce :title;.
- [Steenken et al., 2004] Steenken, D., Voß, S., and Stahlbock, R. (2004). Container terminal operation and operations research - a classification and literature review. *OR Spectrum*, 26 :3–49. 10.1007/s00291-003-0157-z.
- [Stützle and Hoos, 2000] Stützle, T. and Hoos, H. H. (2000). Max-min ant system. *Future Generation Comp. Syst.*, 16(8) :889–914.
- [Sun et al., 1995] Sun, D., Batta, R., and Lin, L. (1995). Effective job shop scheduling through active chain manipulation. *Comput. Oper. Res.*, 22(2) :159–172.
- [Suresh and Chaudhuri, 1993] Suresh, V. and Chaudhuri, D. (1993). Dynamic scheduling—a survey of research. *International Journal of Production Economics*, 32(1) :53 – 63.
- [Swihart and Papastavrou, 1999] Swihart, M. R. and Papastavrou, J. D. (1999). A stochastic and dynamic model for the single-vehicle pick-up and delivery problem. *European Journal of Operational Research*, 114(3) :447 – 464.
- [Taillard, 1994] Taillard, É. D. (1994). Parallel taboo search techniques for the job shop scheduling problem. *INFORMS Journal on Computing*, 6(2) :108–117.
- [Tan et al., 2006] Tan, K. C., Chew, Y. H., and Lee, L. H. (2006). A hybrid multiobjective evolutionary algorithm for solving vehicle routing problem with time windows. *Comput. Optim. Appl.*, 34(1) :115–151.
- [Tang et al., 2000] Tang, L., Liu, J., Rong, A., and Yang, Z. (2000). A multiple traveling salesman problem model for hot rolling scheduling in shanghai baoshan iron & steel complex. *European Journal of Operational Research*, 124(2) :267 – 282.

- [Thangiah, 1991] Thangiah, S. R. (1991). *Gideon : a genetic algorithm system for vehicle routing with time windows*. PhD thesis, Fargo, ND, USA. UMI Order No. GAX91-27716.
- [Toth and Vigo, 2001] Toth, P. and Vigo, D., editors (2001). *The vehicle routing problem*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.
- [Troyon, 1988] Troyon, M. (1988). *Quelques heuristiques et résultats asymptotiques pour trois problèmes d'optimisation combinatoire*. PhD thesis, Lausanne.
- [Udomsakdigool and Kachitvichyanukul, 2011] Udomsakdigool, A. and Kachitvichyanukul, V. (2011). Ant colony algorithm for multi-criteria job shop scheduling to minimize makespan, mean flow time and mean tardiness. *International Journal of Management Science and Engineering Management*, 6(2) :117–123.
- [Ulusoy, 1985] Ulusoy, G. (1985). The fleet size and mix problem for capacitated arc routing. *European Journal of Operational Research*, 22(3) :329–337.
- [Vakhutinsky and Golden, 1994] Vakhutinsky, A. and Golden, B. (1994). Solving vehicle routing problems using elastic nets. In *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*, volume 7, pages 4535 –4540 vol.7.
- [Vallivaara, 2008] Vallivaara, I. (2008). A team ant colony optimization algorithm for the multiple travelling salesmen problem with minmax objective. In *Proceedings of the 27th IASTED International Conference on Modelling, Identification and Control*, MIC '08, pages 387–392, Anaheim, CA, USA. ACTA Press.
- [Vidal et al., 2011] Vidal, T., Crainic, T., M., G., and Prins, C. (2011). Heuristiques pour les problèmes de tournées de véhicules multi-attributs. Technical report, CIRRELT.
- [Vogel et al., 2002] Vogel, A., Fischer, M., Jaehn, H., and Teich, T. (2002). Real-world shop floor scheduling by ant colony optimization. In *Proceedings of the Third International Workshop on Ant Algorithms*, ANTS '02, pages 268–273, London, UK, UK. Springer-Verlag.
- [Wacholder et al., 1989] Wacholder, E., Han, J., and Mann, R. C. (1989). A neural network algorithm for the multiple traveling salesmen problem. *Biological Cybernetics*, 61 :11–19. 10.1007/BF00204755.
- [Widrow, 1960] Widrow, B. (1960). An adaptive 'Adaline' neuron using chemical 'memistors'. Technical report.
- [Williamson et al., 1997] Williamson, D. P., Hall, L. A., Hoogeveen, J. A., Hurkens, C. A. J.,

- Lenstra, J. K., Sevast'janov, S. V., and Shmoys, D. B. (March/April 1997). Short shop schedules. *Operations Research*, 45(2) :288–294.
- [Wilson et al., 1977] Wilson, N., Colvin, N., of Technology. Center for Transportation Studies, M. I., Administration, U. S. U. M. T., and Authority, R. G. R. T. (1977). *Computer control of the Rochester dial-a-ride system*. Massachusetts Institute of Technology, Center for Transportation Studies.
- [Wilson et al., 1975] Wilson, N., of Technology. Dept. of Civil Engineering, M. I., and Administration, U. S. U. M. T. (1975). *Advanced Dial-a-ride Algorithms : Interim Report*. R75-27. The Department.
- [Wilson and of Technology. Urban Systems Laboratory, 1971] Wilson, N. and of Technology. Urban Systems Laboratory, M. I. (1971). *Scheduling Algorithms for a Dial-A-Ride System*. TR (Massachusetts Institute of Technology. Urban Systems Laboratory). Massachusetts Institute of Technology, Urban Systems Laboratory.
- [Xing et al., 2010] Xing, L.-N., Chen, Y.-W., Wang, P., Zhao, Q.-S., and Xiong, J. (2010). A knowledge-based ant colony optimization for flexible job shop scheduling problems. *Applied Soft Computing*, 10(3) :888 – 896.
- [Yang et al., 2004] Yang, J., Jaillet, P., and Mahmassani, H. (2004). Real-time multivehicle truckload pickup and delivery problems. *Transportation Science*, 38(2) :135–148.
- [Yoo and Müller, 2002] Yoo, M.-J. and Müller, J.-P. (2002). Using multi-agent system for dynamic job shop scheduling. In *ICEIS*, pages 517–525.
- [Zhou et al., 2008] Zhou, R., Lee, H. P., and Nee, A. Y. C. (2008). Applying ant colony optimisation (aco) algorithm to dynamic job shop scheduling problems. *IJMR*, 3(3) :301–320.