Modèles et algorithmes pour le management de nouveaux services de mobilité urbaine et rurale

Aurélien Mombelli




Thèse

Modèles et algorithmes pour le management de nouveaux services de mobilité urbaine et rurale

Présenté par : **Aurélien Mombelli**

Directeurs de thèse :	Rapporteuse et rapporteur :	Examinatrice et examinateur :	soutenance :
Alain Quilliot	**Sonia Vanier**	**Nathalie Grangeon**	**08 Novembre 2024**
Mourad Baiou	**Aziz Moukrim (président)**	**Thierry Garaix**	

1 Rue de la Chebarde, 63178 Aubière

Je veux remercier toutes les personnes qui m'ont aidé pendant ma thèse de quelques manières que ce soit et tout particulièrement, dans un ordre quelconque :

Alain Quilliot, avec qui j'ai beaucoup travaillé durant ces années de thèse, pour son accompagnement.

Mourad Baiou, pour la confiance qu'il m'a donné et les possibilités qu'il m'a offertes.

Trang Vo, ma collègue doctorante qui a commencé sa thèse le même jour que moi, mais surtout une amie avec qui on ne s'ennuie jamais.

Tous mes collègues doctorants et amis, en particulier Chijia Liu.

Béatrice Bourdieu, pour l'aide constante et sa patience. Coline Rimbault, ma compagne, qui m'a soutenu et accompagné toutes ces années.

Ma famille et mon neveu qui vient de naître, qui ont cru en moi (sauf mon neveu qui est trop jeune).

William Guyot-Lena et l'association d'astronomie 4A, avec qui j'ai pu m'échapper et apprendre l'astronomie, mais aussi la géologie et d'autres domaines encore que je ne connaissais pas ou peu.

Emile Rouleau.

Résumé

L'optimisation combinatoire est un domaine des mathématiques dans lequel un problème consiste à trouver une solution optimale dans un ensemble fini d'objets. Elle a des applications cruciales dans de nombreux domaines, notamment les mathématiques appliquées, le génie logiciel, l'informatique théorique et l'apprentissage automatique.

Cette thèse se place dans le champ des travaux en recherche opérationnelle qui cherchent à améliorer le transport des biens et des personnes dans les zones urbaines et rurales. Ce document se concentre sur l'étude de trois problèmes de mobilité dans lesquels seront abordées les contraintes liées aux véhicules autonomes. L'objectif est d'étudier les particularités des problèmes incluant ce type de véhicules, et d'utiliser leurs caractéristiques générales afin de proposer des algorithmes novateurs. Cette étude ne fait partie d'aucun projet industriel particulier et se concentre donc sur les aspects généraux des véhicules autonomes.

Ces problèmes de logistique, auxquels nous souhaitons adjoindre les contraintes liées aux véhicules autonomes, sont nombreux et étudiés depuis longtemps. À commencer par le problème de tournées de véhicules (*Vehicle Routing Problem* - VRP) dans lequel l'objectif est de trouver les tournées les plus rapides afin de livrer un ensemble de clients avec un nombre fixé de véhicules. De ce problème est né de nombreuses variantes comme le problème de tournées de véhicules avec fenêtres de temps (*Vehicle Routing Problem with Time Windows* - VRPTW) qui est une extension du VRP où chaque client ne peut être servi que pendant une période prédéfinie, appelée fenêtre de temps. Cette thèse se concentre sur trois problèmes de logistiques distincts avec un point commun : des mécanismes de synchronisations sont ajoutés pour répondre aux problématiques des véhicules autonomes.

Dans le premier chapitre, nous étudierons un problème de plus court chemin dans un environnement risqué. Ce risque provient des autres véhicules autonomes déjà présent sur place et de leur planning. Il est donc dépendant du temps et nous cherchons à ajouter un nouveau véhicule autonome pour répondre à une tâche non prise en charge par la flotte actuelle. Ce nouveau véhicule devra trouver un chemin jusqu'à sa destination en tenant compte du risque sur son trajet et en adaptant sa vitesse en conséquence. Ce problème a fait l'objet de trois articles [Mombelli et al., 2022, Mombelli et al., 2023a, Mombelli et al., 2024] dont deux en conférence internationale et un article de journal international.

Dans le deuxième chapitre, nous exposerons une idée que nous proposons pour résoudre les problèmes intrinsèquement dépendant du temps : projeter le problème en supprimant la dimension temporelle, résoudre ce problème projeté et construire une solution du problème temporel dont la projection est exactement la solution du problème projeté. Nous appliquerons cette idée à un problème de relocalisation de marchandises sous la forme d'un problème à deux flots (un flot de véhicules et un flot de marchandises) dans lequel la préemption des marchandises est autorisée. La préemption des marchandises est le fait d'autoriser une marchandise à être déchargé par le véhicule qui la transporte afin d'être récupérée par un deuxième véhicule pour continuer son trajet. Cette contrainte, nécessitant la synchronisation des véhicules pour s'échanger des marchandises, est de plus en plus utilisée dans les entrepôts logistiques autonomes et semi-autonomes. Ce problème a fait l'objet d'un article [Mombelli et al., 2023b] dans une conférence nationale.

Dans le troisième chapitre, nous travaillerons sur un problème de synchronisation de deux acteurs, l'un produisant une ressource nécessaire au bon fonctionnement du deuxième. Périodiquement, un transfert de cette ressource doit avoir lieu pour maintenir l'activité du deuxième acteur. Ce problème peut être vu comme l'échange de batteries entre un véhicule autonome électrique et une station de production d'électricité qui va recharger la batterie inutilisée pendant que le véhicule réalise ses tâches. Ce problème a fait l'objet d'un article [Mombelli and Quilliot, 2023] dans une conférence internationale.

Abstract

Combinatorial optimization is a field of mathematics in which a problem involves finding an optimal solution in a finite set of objects. It has crucial applications in many fields, including applied mathematics, software engineering, theoretical computer science, and machine learning.

This thesis is placed in the field of operational research works which seeks to improve the transport of goods and people in urban and rural areas. This document focuses on the study of three mobility problems in which constraints linked to autonomous vehicles will be addressed. The objective is to study the particularities of problems which includes this type of vehicles, and to use their general characteristics in order to propose innovative algorithms. This study is not part of any particular industrial project and therefore focuses on general aspects of autonomous vehicles.

These logistical problems, to which we wish to add the constraints linked to autonomous vehicles, are numerous and have been studied for a long time. Starting with the Vehicle Routing Problem (VRP) in which the objective is to find the fastest routes in order to deliver to a set of customers with a fixed number of vehicles. From this problem many variants were born such as the Vehicle Routing Problem with Time Windows (VRPTW) which is an extension of the VRP where each customer can only be served in a predefined period, called time window. This thesis focuses on three distinct logistics problems with common constraints: synchronization mechanisms are added to address the issues of autonomous vehicles.

In the first chapter, we will study a shortest path problem in a risky environment. This risk comes from the other autonomous vehicles already present on site and their daily planning. It is therefore time-dependent, and we are looking to add a new autonomous vehicle to respond to a task not supported by the current fleet. This new vehicle will have to find a path to its destination by taking into account the risk on it path and adapt its speed accordingly. This problem was the subject of three articles [Mombelli et al., 2022, Mombelli et al., 2023a, Mombelli et al., 2024] including two in an international conference and one international journal article.

In the second chapter, we will present an idea that we propose to solve intrinsically time-dependent problems: project the problem by removing the temporal dimension, solve this projected problem and construct a solution of the temporal problem whose projection is exactly the solution of the projected problem. We will apply this idea to a relocation problem in the form of a two flows problem (a flow of vehicles and a flow of goods) in which preemption of goods is allowed. Preemption of goods is the act of allowing goods to be unloaded by the vehicle transporting it in order to be picked up by another vehicle to continue its journey. This constraint, requiring the synchronization of vehicles to exchange goods, is increasingly used in autonomous and semi-autonomous logistics warehouses. This problem was the subject of an article [Mombelli et al., 2023b] in a national conference.

In the third chapter, we will work on a problem of synchronization of two actors, one producing a resource necessary for the proper functioning of the second. Periodically, a transfer of this resource must take place to maintain the activity of the second actor. This problem can be seen as the exchange of batteries between an autonomous electric vehicle and an electricity production station, which will recharge the unused battery while the vehicle carries out its tasks. This problem was the subject of an article [Mombelli and Quilliot, 2023] in an international conference.

Sommaire

Liste des figures

Liste des tables et tableaux

Liste des algorithmes

Section 1

État de l'art

L'optimisation combinatoire est un sous-domaine de l'optimisation mathématique qui consiste à trouver une solution optimale à partir d'un ensemble fini de possibilités. Il s'agit de l'un des domaines de recherche les plus actifs de ces dernières années puisque des milliers de problèmes de décision réels, ayant un impact significatif sur notre vie quotidienne, peuvent être formulés sous forme de problèmes d'optimisation combinatoire. Les exemples incluent, mais sont non limités à, l'optimisation de ressources dans le domaine de la santé (voir [Jain and Bharti, 2023]), la minimisation des coûts de transport de marchandises (voir [Dumez, 2021]), la minimisation des émissions de gaz à effet de serre d'opérations logistiques (voir [McKinnon, 2011]) et la maximisation du profit total de la sélection d'un portefeuille de projets en entreprise (voir [Belaid, 2011]). Ces problèmes sont souvent caractérisés par leur complexité et leur difficulté, nombre d'entre eux étant classés comme NP-difficiles, et l'obtention de solutions optimales, ou quasi optimales en un temps raisonnable, est au cœur des recherches en optimisation combinatoire.

Bien que les solutions possibles d'un tel problème soient finies, leur nombre augmente de façon exponentielle avec la taille de l'instance, rendant la recherche exhaustive insoluble. Pour bien comprendre cette différence entre la simplicité apparente d'un problème et le nombre de solutions différentes possible, nous prendrons exemple sur le problème du sac à dos (*Knapsack Problem*). L'énoncé du problème est assez simple : plusieurs objets de poids et valeurs différentes sont disponibles et nous souhaitons remplir un sac à dos d'une certaine capacité afin de maximiser la valeur contenue dans le sac. Prenons un exemple, avec un sac à dos capable de supporter 4 kg et quatre objets différents : A (1 kg / 4 €), B (3 kg / 21 €), C (2 kg / 11 €) et D (2 kg / 16 €). C'est ici qu'apparaît le problème puisque le nombre de possibilités est exponentiel ! Plus précisément, il y a 2^N groupes d'objets possibles (avec N le nombre total d'objets). Pour un problème avec quatre objets, tester toutes les possibilités est rapide puisqu'il y en a $2^4 = 16$. De plus, certains groupes sont trop lourds pour notre sac à dos et les ensembles d'objets acceptables sont {A}, {A,B}, {A,C}, {A,D}, {B}, {C}, {C,D} ou {D}. Mais dès que la taille augmente, il n'est plus possible de résoudre ce type de problème en testant toutes les possibilités. Pour seulement vingt objets, il y a déjà un million de possibilités et certains exemples réels traitent plusieurs centaines voir des milliers d'objets et plusieurs sacs à dos en même temps.

1.1 Les méthodes de résolutions

1.1.1 Les programmes linéaires en nombre entiers mixtes

Pour surmonter ce défi, des algorithmes de recherche efficaces ont été développés pour des problèmes d'optimisation combinatoire spécifiques, résultant en des modèles avec plus de contraintes. Toutefois, ces algorithmes nécessitent généralement une compréhension approfondie des caractéristiques du problème étudié et sont non réutilisables d'un problème à l'autre. Une autre approche consiste à établir des méthodes génériques pour résoudre n'importe quel problème d'optimisation combinatoire.

L'une de ces approches génériques est la programmation linéaire en nombres entiers mixtes (*Mixte Integer Linear Programming* - MILP). La totalité des problèmes d'optimisation combinatoire NP peut être

formulée naturellement par une formulation MILP. C'est cette universalité qui a fait et qui fait encore son succès.

Le MILP est une généralisation de la programmation linéaire (*Linear Programming* - LP) dans laquelle toutes les variables sont à valeurs réelles et qui fait partie des problèmes polynomiaux. Ces problèmes LP peuvent être résolus, par exemple, par la méthode du simplex, proposée par George Dantzig en 1947 (voir [Dantzig, 1947]), ou par la méthode des points intérieurs qui est actuellement la méthode avec la meilleure complexité théorique (voir [Jiang et al., 2020]). Mais le MILP est également une généralisation de la programmation linéaire en nombres entiers (*Integer Linear Programming* - ILP) dans laquelle, à l'inverse, toutes les variables sont à valeurs entières et qui fait partie des 21 problèmes NP-complets de Karp (voir [Karp, 1972]). Ce sont des problèmes de combinatoire et de théorie des graphes dont Richard M. Karp a prouvé en 1972 la NP-complétude et a montré qu'ils sont réductibles entre eux. Le MILP est donc appelé «mixte» car il traite à la fois avec des variables continues (à valeur réelle) et discrètes (à valeur entière). Le MILP étant une généralisation du ILP, il est donc également NP-complet.

Pour être plus précis, le MILP consiste à trouver les meilleures valeurs possibles aux variables entières et continues pour optimiser une fonction objectif linéaire sous un ensemble de contraintes linéaires d'égalité et d'inégalité. La linéarité signifie que la fonction objectif et les contraintes sont des fonctions affines des variables.

La principale différence entre le MILP et la programmation linéaire (LP) réside dans la complexité de l'ensemble des solutions. Dans un LP, les variables peuvent prendre n'importe quelle valeur réelle, alors qu'en MILP, tout ou partie de ces variables doivent prendre uniquement des valeurs entières. Restreindre les variables à des nombres entiers introduit des défis supplémentaires puisque, en plus, les solutions entières peuvent ne pas toujours exister ou être uniques.

Pour illustrer cette méthode, nous allons reprendre l'exemple du *Knapsack Problem* présenté précédemment en ajoutant un objet un peu particulier puisque celui-ci est sécable. Autrement dit, il est possible de prendre qu'une portion de cet objet. En formulant le problème de façon un peu plus général, cette variante du problème du sac à dos s'énonce ainsi :

- Nous avons N éléments non sécables indexés par $i = 1, \dots, n$. Chaque élément i possède un poids w_i et une valeur v_i.
- Nous avons un élément sécable d'indice 0, de poids total w_0 et de valeur totale v_0
- Notre objectif est de remplir un sac à dos de capacité maximale W.

Les décisions que nous pouvons prendre sont de mettre, ou non, un objet non sécable dans le sac à dos, mais aussi en quelle proportion l'objet sécable est ajouté. Les décisions sont bien entières (elles sont même binaires) pour les objets non sécables, et réel pour la proportion de l'objet sécable. Notons z_i ces variables (entière pour $i = 1, \dots, n$ et réel pour $i = 0$). Le MILP associé à cette variante du problème de sac à dos peut s'écrire ainsi :

$$\begin{aligned} \textbf{minimiser} \quad & \sum_{i=0}^{n} v_i z_i \\ \textbf{tel que} \quad & \sum_{i=0}^{n} w_i z_i \leq W \\ & z_i \in \{0,1\}, \quad i = 1, \dots, n \\ & z_0 \in [0,1] \end{aligned}$$

1.1.2 L'algorithme du *Branch&Bound*

Une fois qu'un problème d'optimisation combinatoire est formulé sous forme de problème MILP (nous prendrons l'exemple d'un problème de minimisation comme le *Knapsack Problem*), il peut être résolu à l'aide de solveurs génériques, c'est-à-dire indépendants du problème. La méthode de base de résolution des solveurs MILP actuels est le *Branch&Bound*. Elle a été proposée par Ailsa Land and Alison Doig en 1960 pour résoudre les problèmes de décisions discrètes (voir [Land and Doig, 1960]). Ce type de problème

cherche à optimiser une fonction objectif dont les variables peuvent prendre des valeurs discrètes. C'est notamment le cas des ILP et des MILP.

Cette méthode se décompose, comme son nom l'indique, en un schéma de branchement et un schéma de recherche de bornes, permettant une recherche efficace à travers un arbre de solutions candidates. Chaque nœud de l'arborescence représente un sous-problème dérivé du problème d'origine, formé en faisant des choix ou des décisions particulières à chaque nœud de l'arbre. À chaque étape de la recherche, la méthode *Branch&Bound* sélectionne un nœud feuille de l'arbre en fonction de certains critères, souvent déterminés par la meilleure solution connue et les bornes inférieures disponibles. Pour ce nœud sélectionné, un branchement sur ses valeurs est réalisé : il génère deux nœuds enfants ou plus en considérant toutes les décisions ou branches possibles qui peuvent être prises. Par exemple, dans un problème de sac à dos, le branchement binaire implique d'inclure un élément dans le sac à dos (ce qui donnerait un nœud enfant) ou de le laisser de côté (ce qui produirait un second nœud enfant). La méthode *Branch&Bound* répète ce processus de branchement pour chaque nœud nouvellement créé jusqu'à ce que la solution optimale soit trouvée ou que tout l'espace de recherche soit épuisé.

Ensuite, chaque enfant est évalué pour estimer l'intérêt du sous-arbre qu'il contient, autrement dit d'estimer la qualité de la solution qui peut être obtenue à partir d'un nœud donné et de ses enfants sans réellement la calculer. Cette évaluation est faite au travers du calcul d'une borne inférieure, d'où son nom anglais : *Bound* ou *Bounding*. Cette borne inférieure est calculée en relâchant le modèle, c'est-à-dire en considérant que toutes les variables sont réelles. En effet, puisque que ce nouveau problème est plus permissif que la version avec les variables entières, la solution optimale est au mieux la même solution et meilleure sinon, dans le sens où la valeur objectif sera plus petite puisqu'on cherche à minimiser. Ainsi, la résolution du problème relâché sur un nœud donné permet de calculer la borne inférieure de toutes les solutions ayant pris les décisions permettant d'atteindre ce nœud, c'est-à-dire toutes les solutions dans le sous-arbre partant de ce nœud.

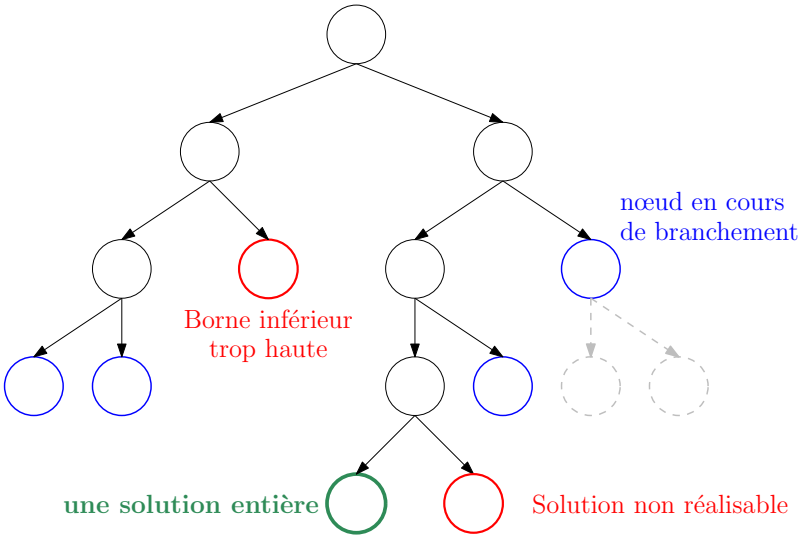

FIGURE 1.1 – Un arbre d'exploration (ici binaire) au milieu de l'exécution du *Branch&Bound*.

Maintenant que les deux morceaux les plus importants de la méthode ont été décrits (voir figure 1.1), voici comment fonctionne la méthode du *Branch&Bound* :

Il reste deux points à aborder pour bien comprendre la méthode (indiqué par *(1)* et *(2)* dans le pseudo-code 1.1) : quelle variable utiliser pour réaliser le branchement et quelles décisions prendre pour cette variable. Pour répondre à la première question, les solveurs de pointe calculent des coûts réduits pour chaque variable en espérant réduire le plus possible la valeur de la fonction objectif et ainsi diminuer le nombre de nœuds explorés au total. Cependant, choisir la prochaine variable dans l'ordre lexicographique est aussi une stratégie valable, bien que moins efficace. En ce qui concerne les valeurs pertinentes pour la

Algorithme 1.1 : Schéma de résolution *Branch&Bound*

Entrées : Un problème de décisions discrètes
Début
Calculer la solution du problème relâché, c'est la racine de l'arbre de recherche du MILP
Tant que L'arbre de recherche n'est pas complètement exploré **Faire**
Choisir le nœud feuille le plus prometteur de l'arbre (avec la borne inférieure la plus petite)
Si La borne inférieure est plus grande que la valeur d'une solution trouvée **Alors**
FIN du *Branch&Bound* puisqu'aucune meilleure solution ne pourra être trouvée.
Fin Si
Choisir une variable sur laquelle réaliser le branchement (1)
Pour toute valeur pertinente possible pour cette variable **Faire** (2)
Calculer la borne inférieure du sous-arbre partant de ce nœud enfant et l'insérer dans l'arbre de recherche
Fin Pour
Fin Tant que
Fin

variable choisie, si la variable est binaire, alors les deux décisions possibles sont de fixer sa valeur à 0 ou à 1. Deux nœuds enfants sont donc créés à partir de chaque nœud d'exploration, générant ainsi un arbre binaire. C'est le cas dans l'exemple du *Knapsack Problem* puisque chaque variable correspond à un objet et sa valeur correspond au fait de prendre, ou non, cet objet dans le sac. Dans le cas de variables entières non bornées, il n'est pas possible de réaliser un branchement sur toutes les valeurs puisqu'il y en a une infinité. Mais nous pouvons nous aider de la valeur de cette variable dans la solution au problème relâché. Dans ce problème, les variables ont une valeur réelle qui est supposée proche de la solution optimale du problème initial. Nous pouvons alors faire un branchement binaire sur les arrondis supérieur et inférieur de la valeur dans la solution du problème relâché. Par exemple, si la valeur relâchée de la variable choisie x est 3.14, alors, le branchement se fait sur les décisions $x = 3$ et $x = 4$.

Ce schéma de résolution très efficace a cependant un problème de mise à l'échelle : plus le modèle contient de variables et de contraintes, et plus le temps de résolution augmente bien sûr, mais il augmente très vite. Au point que certains problèmes peuvent mettre plusieurs jours ou mois à être résolus. Une nouvelle méthode a donc été proposée pour diminuer le nombre de contraintes du problème tout en garantissant l'optimalité de la solution obtenue par *Branch&Bound*. Elle s'appelle la séparation de contraintes. Elle part d'un modèle simplifié dans lequel certaines contraintes ont été omises puis, dans le *Branch&Bound*, les nouveaux nœuds, en plus de l'évaluation de la borne inférieure de la valeur de la fonction optimale, une vérification des contraintes omises est effectuée. Cette vérification peut se faire de plusieurs manières, mais retourne le même format de réponse dans tous les cas : soit la solution en cours de vérification ne viole aucune des contraintes omises, ou bien elle en viole au moins une. Dans le deuxième cas, la contrainte violée trouvée est ajoutée à tous les nœuds du sous-arbre d'exploration correspondant. De cette façon, la contrainte n'est ajoutée que là où elle est utile.

Enfin, une amélioration du *Branch&Bound* a été proposé, toujours dans l'idée d'ajouter des contraintes sur des sous-arbres spécifiques, mais cette fois-ci ce ne sont pas des contraintes du problème. Ce sont des contraintes qui sont valides pour des solutions entières mais violées par la solution fractionnaire en cours d'exploration. Cette méthode s'appelle le *Branch&Cut* (voir [Cordeau, 2006]).

1.1.3 La Programmation Dynamique

La programmation dynamique est une autre méthode générale de résolution introduite pour la première fois par Richard Bellman dans son livre fondateur "Dynamic Programming" publié en 1957 [Bellman, 1957]. Avant l'émergence de la DP, les méthodes traditionnelles reposaient soit sur l'énumération par force brute

coûteuse en calcul, soit sur des approches de type *Branch&Bound* ou heuristiques pour résoudre des problèmes d'optimisation complexes. En reconnaissant les points communs et la récursivité présents dans de nombreux problèmes d'optimisation, Bellman a proposé une méthode très efficace de résolution, à condition que le problème respecte ce qui est maintenant appelé le principe d'optimalité de Bellman :

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

Une politique optimale a la propriété que quels que soient l'état initial et la décision initiale, les décisions restantes doivent constituer une politique optimale au regard de l'état résultant de la première décision.

Un problème respectant le principe d'optimalité de Bellman est dit posséder des «sous-structures optimales». Par exemple, dans le cas d'une recherche d'un plus court chemin dans un graphe acyclique, si le chemin optimal d'un nœud s à un nœud p passe par un nœud intermédiaire u, alors c'est aussi le plus court chemin entre s et u. Il est donc possible de construire le plus court chemin de s à p en construisant les plus courts chemins jusqu'aux nœuds intermédiaires dans un premier temps et garantir l'obtention de la solution optimale du problème global. Ainsi, la méthode est applicable si le problème étudié peut se formuler comme un problème de recherche d'un plus court chemin dans un graphe acyclique construit pour refléter les caractéristiques du problème étudié.

Reprenons maintenant notre problème de sac à dos : il est possible de construire un graphe tel que chaque nœud représente un sous-ensemble d'objets, en plus de deux nœuds fictifs s et p. Un arc n'est possible qu'entre un nœud représentant un sous-ensemble d'objet quelconque et un deuxième nœud représentant le même sous-ensemble auquel un nouvel objet a été ajouté. Cet arc représente donc le fait d'ajouter ce nouvel objet dans le sac-à-dos. Le nœud s représente un sac-à-dos vide et p est connecté à tous les nœuds dont la sélection d'objet respecte la capacité du sac-à-dos. L'arc porte alors la valeur de l'objet qui est ajouté à la sélection du nœud de départ (ou l'opposé de la valeur). Le graphe représentatif de l'exemple énoncé précédemment est présenté dans la figure 1.2.

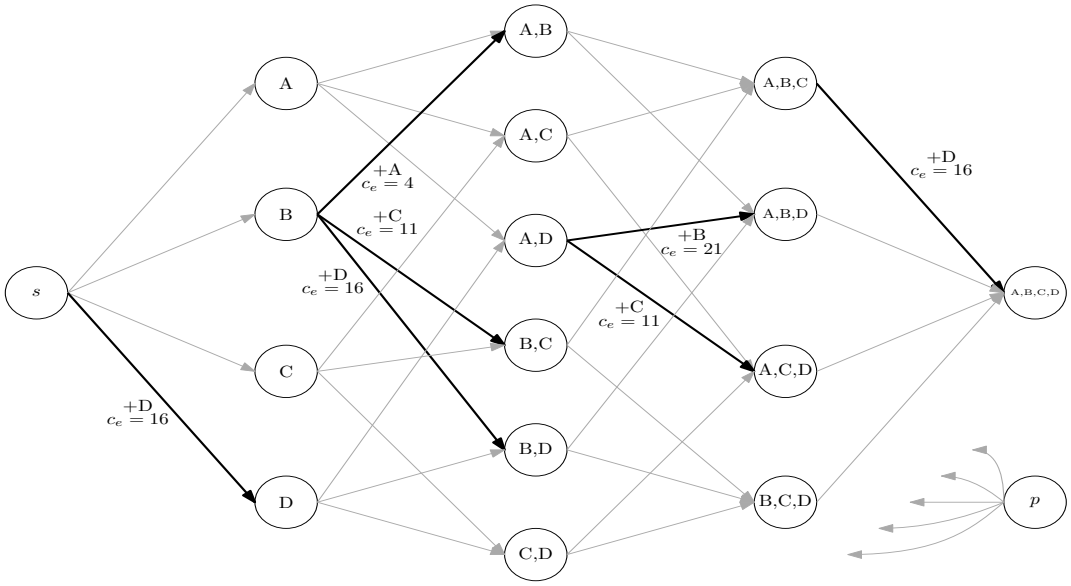

FIGURE 1.2 – Graphe représentant le KP pour l'utilisation d'un schéma de DP (en partie explicité).

L'objectif est alors de chercher le plus long chemin entre s et p (ou le plus court chemin avec l'opposé des valeurs). Un objet est mis dans le sac-à-dos si le plus court chemin passe par le nœud correspondant dans le graphe et un chemin valide vérifie la contrainte de capacité du sac-à-dos. Le KP possède donc bien

des sous-structures optimales et il est solvable par DP. Généralement, un schéma de DP est présenté sous la forme suivante utilisée dans la figure 1.3.

Espace temporel I	$I = \{0, \ldots, n\}$ l'espace temporel de la Programmation Dynamique est un espace dans lequel les solutions sont construites itérativement. Toutes les solutions partielles, à une même date, ont pris un même nombre de décisions. Dans le cas du KP, les solutions à la date i ont sélectionné i objets au total.
Espace des états S	$s = (p, v) \in S$ C'est l'espace regroupant toutes les solutions partielles en cours de construction. Dans le cas du KP, c'est l'état actuel du sac à dos : le poids p et la valeur v des objets sélectionnés jusqu'à présent.
Espace des décisions	$\lambda \in DEC(i, s)$ Sachant que nous sommes à la date i du processus, et que la solution partielle est dans l'état s, les décisions possibles sont regroupées dans l'ensemble $DEC(i, s)$ et permettent de passer à la date $i + 1$. Dans le cas du KP, sachant que nous avons déjà décidé parmi les i premiers objets, la décision λ est de prendre, ou non, le $i + 1^{me}$ objet.
Espace des transitions	$(i, s) \xrightarrow{\lambda} (i + 1, s')$ C'est le processus de mise à jour de l'état. Il dépend fortement du problème. Dans le cas du KP, la transition met à jour les valeurs de poids et de valeur totale du sac à dos pour obtenir le nouvel état après la décision.
Principe de Bellman	Une solution à la date $i + 1$ Seuls les états réalisables non dominés sont gardés : La dominance est une notion indiquant que si un état fait aussi bien qu'un deuxième état, mais est meilleur dans au moins un aspect, alors le premier état domine le deuxième et ce dernier peut être supprimé. Dans le cas du KP, si un état est au plus aussi lourd, avec au moins autant de valeur et est meilleur dans l'un des deux aspects qu'un deuxième état, ce dernier est dominé et peut être supprimé. $\forall (p_1, v_1), (p_2, v_2) \in S$, si $p_1 \leq p_2$ et $v_1 \geq v_2$, (p_2, v_2) est dominé.
Stratégie de recherche	C'est la façon de se déplacer dans l'espace temporel. Le plus souvent, l'espace temporel est traité vers l'avant, c'est-à-dire les solutions sont construites en grossissant au fur et à mesure.
Processus de filtrage	Il est possible d'ajouter des processus de filtrage en plus de celui par domination de Pareto. Ces processus sont fortement dépendants du problème.
Valeur retournée	Une fois la dernière date de l'espace temporelle atteinte, l'état de plus grande valeur objectif est la solution exacte du problème.

FIGURE 1.3 – Schéma générale de Programmation Dynamique et application au KP

Appliquons ce schéma sur notre problème exemple avec uniquement les objets insécables (voir figure 1.4 pour un résumé de l'exécution). Nous prendrons les objets dans l'ordre lexicographique. Pour rappel, la capacité du sac est de 4 kg et les quatre objets sont A (1 kg / 4 €), B (3 kg / 21 €), C (2 kg / 11 €) et D (2 kg / 16 €).

- À la date $i = 0$, le sac à dos est vide, l'état s actuel est donc $(0, 0)$.
  Les décisions possibles pour aller à la date $i = 1$ sont au nombre de quatre.
- À la date $i = 1$, nous avons donc quatre états possibles :
  $s_1 = (1, 4)$, $s_B = (3, 21)$, $s_C = (2, 11)$ et $s_D = (2, 16)$.
  Les décisions possibles depuis ces quatre états sont plus nombreuses, mais beaucoup sont redondantes.
- À la date $i = 2$, nous n'avons que trois états puisque certains ne respectent pas la capacité du sac-à-dos ($s_{B,C}$ et $s_{B,D}$) et $s_{A,C}$ est dominé par $s_{A,D}$:
  $s_{A,B} = (4, 25)$, $s_{A,D} = (3, 20)$ et $s_{C,D} = (4, 27)$.
- À la date $i = 3$, aucun état ne respecte la capacité du sac-à-dos, donc le schéma de DP se termine.

Le meilleur état (celui de plus grande valeur) est $s_{C,D}$, la solution de notre instance de sac à dos est donc un sac à dos plein de valeur totale 27 €.

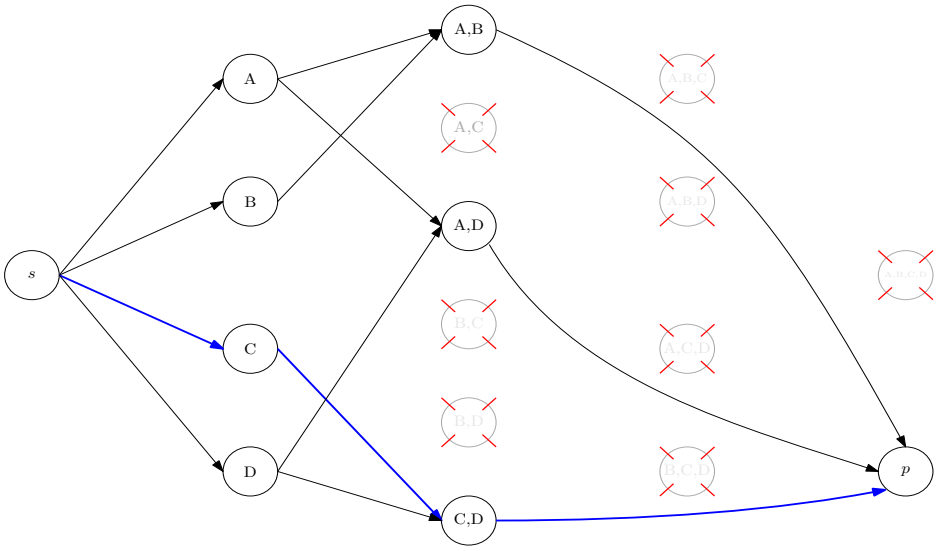

FIGURE 1.4 – Résumé de l'exécution d'un algorithme de DP sur notre instance du KP.

Malgré ses avantages, DP se heurte à certaines limites. Premièrement, il n'est pas toujours adapté aux problèmes importants en raison des besoins en mémoire. À mesure que la taille du problème augmente, la quantité de mémoire nécessaire pour stocker tous les états possibles augmente également. De plus, DP n'est possible que sur un graphe acyclique.

1.1.4 L'algorithme A*

Pour la recherche d'un plus court chemin dans un graphe cyclique, un MILP peut être réalisé, mais il est également possible d'utiliser un algorithme maintenant très connu et parmi les plus utilisés dans le monde dans les applications GPS de nos téléphones portables : l'algorithme A*.

A* (prononcé « A étoile ») est un algorithme de recherche heuristique largement utilisé dans les tâches de recherche de chemin et de parcours de graphes. Il a été proposé par Nils Nilsson en 1968 sous le nom de *Graph Traverser algorithm* (voir [Doran et al., 1966]) puis développé par Bertram Raphael et Peter Hart (voir [Hart et al., 1968]). A* est une amélioration par rapport à l'algorithme de Dijkstra, car il utilise des heuristiques pour guider plus efficacement la recherche vers la solution optimale. En estimant le coût pour atteindre l'objectif à partir d'un nœud donné, A* peut prioriser l'exploration des nœuds les plus proches

de l'objectif, réduisant ainsi considérablement l'espace de recherche nécessaire pour trouver le chemin le plus court.

L'idée centrale de A* réside dans le maintien d'une file d'attente prioritaire de nœuds à explorer ensuite, classés en fonction de leur coût total estimé (autrement dit le coût réel depuis le nœud de départ auquel est additionnée l'estimation heuristique du coût restant pour atteindre l'objectif). À chaque étape, A* explore le nœud dont le coût total estimé est le plus bas et l'étend d'un arc. Si le nœud objectif est trouvé, l'algorithme se termine. Sinon, le processus continue jusqu'à ce que l'objectif soit atteint ou qu'il ne reste plus de nœuds non visités dans la file d'attente prioritaire (voir figure 1.5). L'utilisation de l'heuristique rend l'algorithme A* particulièrement puissant lorsqu'il s'agit de graphes volumineux et d'environnements complexes où trouver le chemin optimal peut s'avérer difficile. Un pseudo-code de l'algorithme A* est présenté dans l'algorithme 1.2.

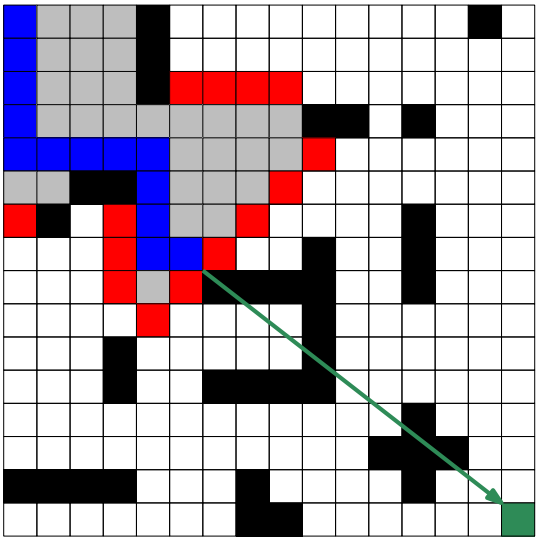

FIGURE 1.5 – Une photo au milieu de l'exécution de l'algorithme A*. Code couleur : les murs sont en noir, les nœuds explorés en gris, les nœuds dans la file à priorité en rouge, le plus court chemin actuel en bleu, et la destination et la valeur heuristique en vert.

Malgré ses atouts, A* présente certaines limites. Premièrement, les performances de A* dépendent fortement de la qualité de la fonction heuristique utilisée. Une heuristique inexacte peut même conduire à des solutions sous-optimales. Deuxièmement, pour les très grands graphes, les besoins en mémoire de l'algorithme A* augmentent considérablement avec le nombre de nœuds. Pour palier à ce problème, diverses méthodes d'approximation et d'optimisation ont été proposées, notamment des stratégies de recherche bidirectionnelle (voir [Sint and de Champeaux, 1977]), de parallélisation (voir [Madduri et al., 2007]) ou d'élagage (voir [Huang et al., 2022]). Enfin, A* suppose que le graphe recherché est entièrement observable et statique, ce qui signifie que tous les arcs ont des poids constants et sont connues à l'avance. Dans des scénarios réels, ces hypothèses pourraient ne pas être vraies, par exemple pour des problèmes d'évacuation de foules lorsque certains arcs peuvent être modifiés ou disparaître en fonction du temps (voir [Pourrahmani et al., 2015]).

1.1.5 La Recherche Locale

Les schémas de recherche locale constituent une classe de méthodes d'optimisation conçues pour améliorer de manière itérative une solution candidate en apportant de petits ajustements à son état actuel, dans le but de trouver une solution optimale localement. Se distinguant par leur simplicité et leur adaptabilité, les algorithmes de recherche locale se sont révélés efficaces pour résoudre un large éventail de problèmes d'optimisation combinatoire.

Algorithme 1.2 : Algorithme A*

Entrées : un graphe $G = (N, E)$ avec :
N est l'ensemble des nœuds, E est l'ensemble des arcs, C_e est le coût de l'arc e,
$H_{u,v}$ est la valeur de l'heuristique de coût restant entre les nœuds u et v,
$Vois(u)$ la liste des couples (e, v) des arcs e partant de u et leur nœud d'arrivée v,
ainsi qu'un nœud origine o et destination d

Début
Soit FP une file à priorité de quadruplets (u, c, h, p) :
(nœud, coût actuel, coût estimé jusqu'à d, parent)
Le trie de FP se fait selon le coût total estimé h
Le parent p nous permettra de reconstituer le chemin recherché entre o et d
Soit L une liste de nœuds visités (initialement vide)

Enfiler $(s, 0, H_{o,d})$ à FP
$FIN = Faux$
Tant que *non* FIN **Faire**
$courant = defiler(FP)$
Ajouter $courant.u$ à L
Si $courant.u = d$ **Alors**
$FIN = Vrai$
Sinon
Pour tout $(e, v) \in Vois(courant.u)$ **Faire**
$c_v = courant.c + C_e$
Si v n'est pas dans L et s'il n'existe pas d'état (v, c, h, p) dans FP tel que $c \leq c_v$ **Alors**
Enfiler $(v, c_v, c_v + H_{v,d}, courant)$
Supprimer l'état (v, c, h, p) de FP s'il existe
Fin Si
Fin Pour
Fin Si
Fin Tant que
Retourner *courant*
Fin

La recherche locale commence par une solution initiale réalisable appelée solution existante. À partir de ce point de départ, une série de mouvements ou d'améliorations sont appliqués, générant des solutions voisines légèrement différentes de l'actuelle. Chaque mouvement évalue la qualité des voisins résultants et sélectionne le meilleur comme nouveau titulaire, en répétant ce processus jusqu'à ce qu'aucune amélioration supplémentaire ne puisse être obtenue. Une solution optimale localement est ainsi obtenue.

Un avantage significatif des algorithmes de recherche locale réside dans leur flexibilité et leur facilité de mise en œuvre par rapport à d'autres méthodes d'optimisation comme le *Branch&Bound* ou la Programmation Dynamique. Elles peuvent gérer des instances à grande échelle de problèmes d'optimisation combinatoire plus efficacement que ces méthodes, ce qui les rend adaptées aux applications du monde réel où les ressources de calcul sont limitées. De plus, les algorithmes de recherche locale peuvent souvent fournir des solutions satisfaisantes qui répondent à des contraintes ou à des objectifs spécifiques, offrant une valeur pratique, même s'ils ne garantissent pas l'obtention de l'optimale globale.

Chaque algorithme de recherche locale est construit pour un problème spécifique, il en existe donc autant qu'il existe de problème. Ils ont cependant un point commun : tous définissent une notion de voisinage d'une solution. Les voisins d'une solution sont eux-mêmes des solutions pouvant être atteints depuis la première grâce à une seule modification. Les modifications possibles sont passées par valeurs à

des opérateurs spécifiques pour chaque problème et appelés opérateurs de voisinage.

Dans le cas de notre problème de sac à dos, à partir d'une solution existante, c'est-à-dire d'un ensemble d'objets sélectionnés, il existe deux opérateurs de voisinage simple : ajouter un objet ou en remplacer un (voir figure 1.6).

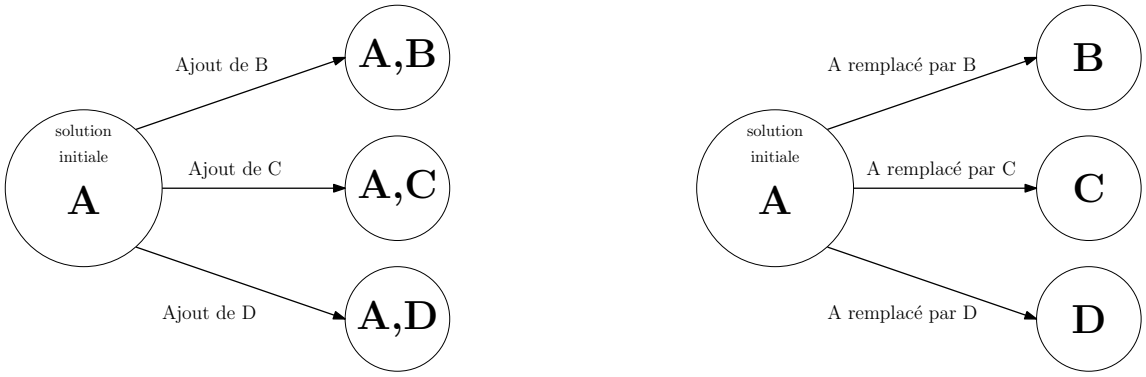

FIGURE 1.6 – Exemple de voisinage pour les opérateurs d'ajout et de remplacement pour le KP.

Il est possible d'ajouter également l'opérateur permettant d'enlever un objet, mais la valeur d'une solution voisine par cet opérateur est toujours plus faible que la solution de départ. Il est important de préciser que ces opérateurs peuvent être plus complexes, comme la recherche d'un cycle améliorant pour la recherche d'un flot maximum par exemple.

Il vient alors la question de savoir quel voisin visiter. En effet, lors de la recherche dans le voisinage d'une solution, trois cas de figure peuvent se produire :

- La solution voisine est réalisable et améliore la valeur de la fonction objectif. C'est le cas de l'ajout de n'importe quel objet dans notre sac à dos initialement vide.
- La solution voisine est réalisable, mais décroit la valeur de la fonction objectif. C'est le cas si l'objet B est remplacé par l'objet A.
- La solution voisine n'est pas réalisable. C'est le cas de l'ajout de l'objet C dans notre sac à dos contenant déjà l'objet B.

Un algorithme de recherche local simple, qualifié de «plus grande descente», consiste à toujours prendre la solution voisine réalisable avec la plus forte amélioration de la valeur de la fonction objectif. Cette méthode permet d'atteindre très rapidement un minimum local, mais n'est pas très efficace en pratique. Un pseudo-code de cette méthode est présenté dans l'algorithme 1.3.

Pour franchir les minima locaux de la fonction objectif et espérer atteindre un meilleur minimum, il faut accepter de dégrader la valeur de la fonction objectif (c'est le cas du Recuit Simulé, voir la section correspondante). De plus, il peut être opportun d'accepter de parcourir des solutions non réalisables pour augmenter la taille de l'espace de recherche et atteindre de nouvelles zones de solutions réalisables autrement inaccessibles. À partir d'une solution non réalisable, un procédé de réparation est appliqué permettant de reconstruire une solution réalisable en minimisant l'impact sur la fonction objectif. Une approche classique consiste à réparer les contraintes violées les unes après les autres ou bien d'ajouter des pénalités correspondantes à la fonction objectif dans le but de guider la recherche de voisinage suivante vers une solution réalisable.

Un autre point faible de cette méthode vient du grand nombre de voisins possible. La recherche exhaustive du meilleur voisin peut alors être contre-productive. Il est alors nécessaire d'utiliser des procédés de filtres efficaces pour affiner l'espace de recherche et se concentrer sur les candidats les plus prometteurs. Par exemple, considérons un problème de voyageur de commerce (*Traveling Salesman Problem* - TSP) à n villes, le nombre total de tournées possibles est $n!$. Évaluer ne serait-ce qu'une petite fraction de ces voisins nécessiterait des ressources de calcul exceptionnelles, rendant une recherche exhaustive irréalisable pour les grandes instances. De plus, à mesure que la taille du problème augmente, la probabilité de rencontrer des voisins de mauvaise qualité augmente, ce qui accroît encore l'importance de disposer de procédés de

Algorithme 1.3 : Algorithme de plus grande descente

Entrées : Une solution initiale s_0
$Obj(s)$ l'opérateur donnant la valeur de la fonction objectif d'une solution s
$Vois(s, val)$ un opérateur de voisinage qui retourne le voisin de s après modification par val
L l'ensemble des valeurs possibles pour l'opérateur de voisinage
Début
$s = s_0$
$FIN = Faux$
Tant que $non\ FIN$ **Faire**
Choisir la meilleure valeur val^* de l'opération de voisinage, c'est-à-dire telle que :
$$Obj\left(Vois(s, val^*)\right) = \min_{val \in L} Obj\left(Vois(s, val)\right)$$
Si $Obj\left(Vois(s, val^*)\right) > Obj(s)$ **Alors**
$FIN = Vrai$
Sinon
$s = Vois(s, val^*)$
Fin Si
Fin Tant que
Retourner s
Fin

filtrage efficaces. Diverses techniques de filtrage ont été proposées au fil des années pour aider à élaguer l'espace de recherche et à réduire le nombre de voisins évalués par les méthodes de recherche locale. Parmi les approches proposées dans la littérature, il y a :

- Estimation par heuristiques : ces fonctions estiment la qualité d'une solution sur la base de caractéristiques facilement calculables. Ils peuvent guider la recherche vers des zones à fort potentiel de l'espace de recherche.
- Limites de distance : en établissant des limites supérieure et inférieure sur la distance entre la solution actuelle et ses voisins, les recherches locales peuvent éviter d'évaluer des voisins qui ne donneront probablement aucune amélioration.
- Relations de dominance : comparer directement les valeurs de la fonction objectif de deux solutions permet d'éliminer les voisins moins bons sans avoir besoin d'évaluations explicites.

En utilisant ces techniques de filtrage et d'autres, les algorithmes de recherche locale peuvent réduire considérablement le nombre de voisins à évaluer, économisant ainsi les ressources de calcul et améliorant leur évolutivité pour des problèmes d'optimisation plus importants.

Malgré leurs mérites, les algorithmes de recherche locale présentent également des limites inhérentes. Leur principale faiblesse vient de leur dépendance à l'égard du point de départ. Une mauvaise solution initiale peut conduire à rester coincé dans un optimum local, empêchant l'algorithme de converger vers l'optimum global. Pour atténuer ce problème, plusieurs variantes d'algorithmes de recherche locale et des métaheuristiques ont été proposées, notamment des procédures aléatoires, le recuit simulé, la recherche tabou et les algorithmes génétiques présentés ci-dessous.

1.1.6 La procédure de recherche adaptative aléatoire gloutonne

La procédure de recherche adaptative aléatoire gloutonne (*Greedy Randomized Adaptive Search Procedure* - GRASP) est une métaheuristique populaire utilisée qui vise à se défaire du problème de la solution initiale. Proposé par Thomas A. Feo et Mauricio G.C. Resende en 1989 [Feo and Resende, 1989], GRASP est une méthode hybride qui intègre des éléments aléatoires et des procédures de recherche locales pour

équilibrer les compromis d'exploration et d'exploitation, améliorant ainsi les chances de trouver la solution optimale au problème étudié.

GRASP utilise une procédure «*multistart*» ou itérative, dans laquelle chaque itération se compose de deux phases consistant en une phase de construction de solutions (c'est l'exploration) et d'une phase de recherche locale (c'est l'exploitation). Au cours de la phase de construction, une solution est construite à l'aide d'un algorithme aléatoire glouton. Si cette solution n'est pas réalisable, il est alors nécessaire d'appliquer une procédure de réparation pour obtenir une solution faisable ou de faire une nouvelle tentative pour construire une solution réalisable. Une fois qu'une telle solution est obtenue, son voisinage est étudié jusqu'à ce qu'un minimum local soit trouvé lors de la phase de recherche locale. La meilleure solution globale est conservée comme résultat final du GRASP. En combinant les deux phases, GRASP évite les pièges des algorithmes purement gloutons tout en conservant les avantages des recherches locales (voir figure 1.7). Un pseudo-code de cette métaheuristique est présenté dans l'algorithme 1.4.

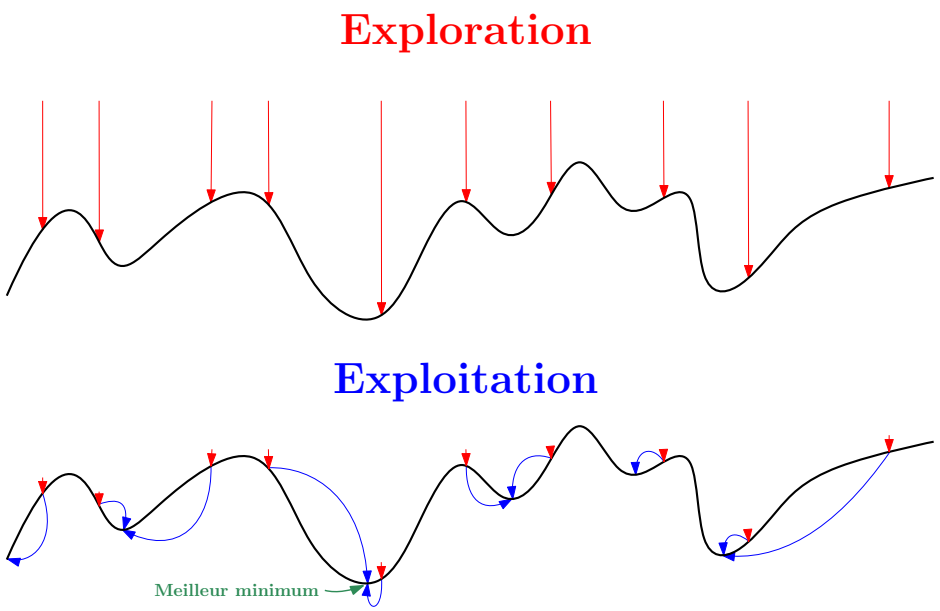

FIGURE 1.7 – Les phases d'exploration et d'exploitation pour la recherche de minima globaux par GRASP.

Algorithme 1.4 : Métaheuristique GRASP

Entrées : $Obj(s)$ l'opérateur donnant la valeur de la fonction objectif d'une solution s
$Generateur()$ un opérateur générant aléatoirement une solution réalisable
$RechercheLocale(s)$ un algorithme de recherche locale partant d'une solution initiale s
K un nombre d'itérations à exécuter
Début
$s_{best} = NIL$
Pour $k = 0, \ldots, K$ **Faire**
$s = Generateur()$
$s = RechercheLocale(s)$
Si $s_{best} = NIL$ ou $Obj(s) < Obj(s_{best})$ **Alors**
$s_{best} = s$
Fin Si
Fin Pour
Retourner s_{best}
Fin

Une force majeure du GRASP réside dans sa capacité à échapper aux optima locaux en intégrant de l'aléatoire dans le processus de construction. Contrairement aux approches purement gloutonnes, qui ont tendance à rester piégées dans des minima locaux en raison de leur focalisation myope sur les gains à court terme, GRASP explore différentes zones de l'espace de recherche, augmentant ainsi la probabilité de découvrir des solutions de meilleure qualité. Même si GRASP offre de nombreux avantages, il reste néanmoins confronté à certains défis. Ses performances dépendent fortement de l'efficacité de l'exploration et de l'étendue du voisinage, ce qui nécessite un réglage minutieux pour chaque application. De plus, étant donné que GRASP ne garantit pas des solutions globalement optimales, il ne répond pas toujours à des exigences strictes en matière d'optimalité.

1.1.7 Le Recuit Simulé

Le Recuit Simulé (*Simulated Annealing* - SA) est une métaheuristique probabiliste inspiré du processus de recuit utilisé dans la métallurgie. Des méthodes très proches ont été proposées entre les années 1970 et 1985 notamment par S. Kirkpatrick, C. Gelatt Jr et M. Vecchi en 1983 qui lui ont donné son nom actuel (voir [Kirkpatrick et al., 1983]). Elle a été conçue pour résoudre des problèmes d'optimisation combinatoire, en particulier ceux présentant des paysages de fonction objectif complexes. En métallurgie, les matériaux subissent des cycles de refroidissement lent et de réchauffage (recuit) qui permettent de minimiser l'énergie interne du matériau. Cette méthode est transposée en optimisation pour trouver les minima d'une fonction.

À la façon d'une recherche locale, une solution candidate est choisie dans le voisinage d'une solution actuelle. La différence de valeur objective entre la solution actuelle et candidate détermine si le mouvement est accepté ou rejeté. Au cours des premières étapes du programme de refroidissement, la température étant encore haute, l'algorithme accepte les mouvements dégradant la valeur de la fonction objectif avec une forte probabilité, favorisant ainsi l'exploration et échappant aux minima locaux. À mesure que le programme de refroidissement progresse, la probabilité d'acceptation diminue, favorisant les mouvements améliorant la valeur de la fonction objectif. Finalement, lorsque la température atteint zéro, l'algorithme converge vers un minimum local. S. Kirkpatrick, C. Gelatt Jr et M. Vecchi ont défini leur probabilité d'acceptation comme étant :

$$P(obj_1, obj_2, T) = \begin{cases} 1 & \text{si } obj_2 < obj_1 \\ \exp(-\frac{obj_2 - obj_1}{T}) & sinon \end{cases}$$

où obj_1 est la valeur de la fonction objectif de la solution courante, obj_2 est celle de son voisin, et $T > 0$ est la température actuelle. Un pseudo-code de cette métaheuristique est présenté dans l'algorithme 1.5.

L'un des principaux avantages du SA réside dans sa capacité à échapper aux optima locaux plus efficacement que les méthodes déterministes. En acceptant des solutions pires avec une probabilité non nulle, le SA permet une meilleure exploration des solutions environnantes, augmentant ainsi les chances de trouver un minimum global.

Malgré ses avantages, le SA est confronté à des défis en termes de taux de convergence et de choix du programme de refroidissement. Un programme de refroidissement lent peut entraîner des durées d'exécution plus longues, mais peut garantir de meilleurs résultats, tandis qu'un programme de refroidissement trop rapide peut conduire à une convergence prématurée vers un optimum local. Plusieurs variantes ont été introduites pour résoudre ces problèmes, notamment des programmes de refroidissement adaptatifs ou cycliques permettant d'alterner des phases d'explorations et d'exploitations.

1.1.8 La Recherche Tabou

La Recherche Tabou (*Tabu Search*) est une autre technique d'optimisation métaheuristique proposé par F. Glover en 1986 (voir [Glover, 1986]). Inspiré par la tendance de la mémoire humaine à se souvenir des actions récentes et à éviter de les répéter, la recherche tabou introduit une mémoire à court terme appelée «liste taboue» pour éviter de revenir sur des solutions précédentes pendant le processus de recherche.

Algorithme 1.5 : Métaheuristique du Recuit Simulé

Entrées : Une solution initiale s_0
$Obj(s)$ l'opérateur donnant la valeur de la fonction objectif d'une solution s
$Vois(s, val)$ un opérateur de voisinage qui retourne le voisin de s après modification par val
L l'ensemble des valeurs possibles pour l'opérateur de voisinage
T_0 une température initiale et $Recuit(T)$ le programme de température
$P(obj_1, obj_2, T)$ la probabilité d'acceptation
Début
$s = s_0$
Tant que $T > 0$ **Faire**
Choisir une valeur $val \in L$ aléatoirement
Soit $p \in [0, 1]$ un nombre aléatoire
Si $p \leq P(Obj(s), Obj\,(Vois(s, val^*))\,, T)$ **Alors**
$s = Vois(s, val^*)$
Fin Si
$T = Recuit(T)$
Fin Tant que
Terminer par une recherche locale gloutonne si nécessaire
Retourner s
Fin

À la façon d'une recherche locale, une solution candidate est choisie dans le voisinage d'une solution actuelle. Cependant, à l'inverse de la recherche locale, cette solution candidate peut détériorer la valeur de la fonction objectif si aucune solution améliorante n'est possible. À chaque itération, la solution visitée est ajoutée à une mémoire à court terme et la Recherche Tabou ne sera pas autorisée à visiter de nouveau une solution présente dans cette liste taboue. En évitant les solutions précédemment visitées, cette méthode encourage la diversification et la découverte de nouvelles régions dans l'espace de recherche.

La taille de la mémoire à court terme permet, si elle est petite, d'améliorer l'exploration de la recherche locale en gardant un algorithme rapide et, si elle est plus grande, d'explorer largement et d'augmenter les chances d'obtenir de bonnes solutions. La polyvalence de la Recherche Tabou et son efficacité ont conduit le développement de plusieurs améliorations et extensions, notamment les stratégies d'exploration guidées. Ces améliorations visent à optimiser davantage le processus de recherche, en réduisant le risque de rester coincé dans les minima locaux et en améliorant les performances globales, comme celle proposée par E. Zachariadis, C. Tarantilis et C. Kiranoudis (voir [Zachariadis et al., 2009]). Ces derniers ont travaillé sur le problème de tournées de véhicules avec contraintes de capacité 2D (*Capacited Vehicle Routing Problem with two-dimensioinal loading constraints* - 2L-CVRP). Dans leur méthode, l'exploration est guidée par des pénalités ajoutées à la fonction objectif. Ces pénalités cherchent à réduire l'utilisation d'arcs longs et coûteux. Pour une solution donnée, un seul arc est pénalisé : celui, parmi tous les arcs utilisés, maximisant une certaine fonction du coût et du nombre de fois où cet arc a été pénalisé. Leur méthode guidée a permis d'améliorer la meilleure solution connue de plusieurs instances au moment de leur publication.

1.1.9 L'algorithme génétique

Les algorithmes génétiques (*Genetic Algorithm* - GA) sont des métaheuristiques évolutionnistes basées sur les principes de la sélection naturelle et de la génétique. Ce schéma d'optimisation est très différent de ceux présentés jusqu'à présent puisque celui-ci traite d'un ensemble de solutions en parallèle et utilise des opérateurs agissant sur des couples de solutions. En particulier, ces algorithmes imitent les processus biologiques de mutation, de croisement et de survie des individus les plus adaptés pour trouver des solutions optimales ou quasi optimales à des problèmes d'optimisation complexes.

- le croisement de deux individus :
  Lors de la création d'un nouvel individu, les gènes de deux parents sont mélangés et assemblés, maintenant ainsi le patrimoine génétique de la génération précédente, mais permettant également la recherche du plein potentiel d'un couple de génome.
- La mutation d'un individu :
  La mutation par un évènement extérieur permet de créer de nouveaux attributs au sein d'une population. Si cette particularité améliore la population et sous de bonnes conditions, la sélection naturelle transmettra cette nouvelle caractéristique aux générations suivantes.
- La sélection naturelle :
  Ce phénomène se caractérise par la survie d'une partie de la population porteuse d'un gène sur une autre non-porteuse, car ce gène permet une meilleure adaptabilité à leur environnement.

Les imitations du vivant dans le domaine de l'informatique remontent à 1950 lorsque A. Turing propose une machine apprenante («*learning machine*») qui utilise des principes évolutionnistes (voir [Turing, 1950]). Mais les algorithmes génétiques tels que nous les utilisons aujourd'hui ont été proposés par J. Holland au milieu des années 1970 (voir [Holland, 1975]) et popularisé par D. Goldberg notamment grâce à son livre «*Genetic Algorithms in Search, Optimization, and Machine Learning*» (voir [Goldberg, 1989]).

Le principe de cette métaheuristique est donc le suivant : un processus de sélection naturelle va être mis en place pour faire évoluer une population d'individus, chacun d'eux représentant une solution réalisable du problème étudié. À chaque itération, une nouvelle population d'individus va être générée en croisant des individus de la génération précédente. Ces derniers n'ont cependant pas tous la même chance d'être sélectionné pour être parent d'un nouvel individu. Cette probabilité dépend de leur capacité d'adaptation (la valeur de la fonction objectif de cette solution). L'individu créé par ce croisement a ensuite une certaine probabilité de muter, apportant de la nouveauté à la future génération. Une fois les nouveaux individus créés, la sélection naturelle permettra la survie d'un certain nombre d'individus, parmi la génération précédente et les nouveaux, formant ainsi la nouvelle génération (voir schéma récapitulatif dans la figure 1.8). Un pseudo-code est présenté dans l'algorithme 1.6.

Algorithme 1.6 : Structure d'un algorithme génétique

Entrées : Une génération initiale G_0
$Crossover(i_1, i_2)$ crée un nouvel individu par croisement entre i_1 et i_2
$Mutation(i)$ provoque une mutation aléatoire de i avec une certaine probabilité
$Selection(G, L)$ est un opérateur de sélection naturelle des individus de G et L
Début
  $G = G_0$
  Évaluer les individus de G
  Tant que Conditions sur G non satisfaites **Faire**
    Soit L la liste des nouveaux individus (initialement vide et de capacité limitée)
    Tant que L n'est pas remplie **Faire**
      Sélectionner deux individus i_1 et i_2 de G selon leur capacité d'adaptation (1)
      $i = Crossover(i_1, i_2)$ (2)
      $Mutation(i)$ (3)
      Ajouter i à L
    Fin Tant que
    $G = Selection(G, L)$ (4)
    Évaluer les individus de G
  Fin Tant que
  Retourner Le meilleur individu de G
Fin

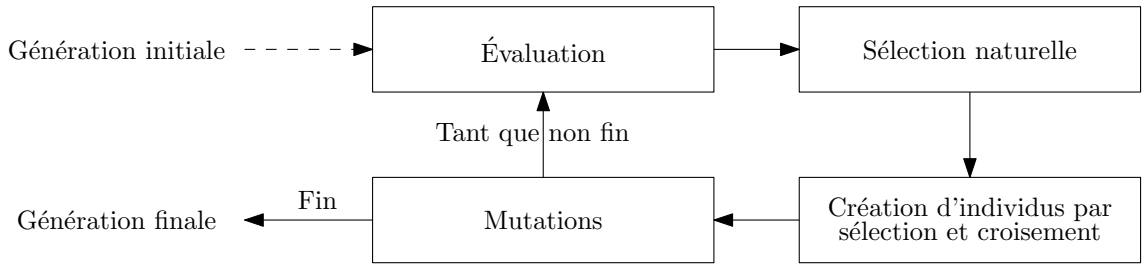

FIGURE 1.8 – Schéma simplifié d'un algorithme génétique.

Il y a cinq points importants à discuter : définir le génome d'une solution et les quatre opérations notées 1, 2, 3 et 4 dans le pseudo-code ci-dessus.

- Un individu, défini par son génome, représente une solution sous une forme simple et dépend très fortement du problème traité. Il en existe plusieurs types comme le codage binaire pour la résolution du KP par exemple. Ici, le génome est composé d'autant de bits qu'il y a d'objets et ils prennent les valeurs 1 si l'objet est dans le sac et 0 sinon. Ou encore, le codage à caractère multiple pour la résolution d'un problème de voyager de commerce par exemple. Dans ce cas, l'identifiant de chaque ville apparait une et une seule fois dans le génome et l'ordre d'apparition définit la tournée.
- La sélection d'un individu pour son croisement avec un second est fait en fonction de sa capacité d'adaptation (la valeur de la fonction objectif de la solution correspondante). Tous les individus de la population se voient alors associer une probabilité d'être sélectionné, proportionnelle à sa capacité d'adaptation par exemple.
- L'opérateur de croisement est, lui aussi, très dépendant du problème étudié, mais le principe reste le même : les deux génomes sont coupés (en deux ou plus), une partie des deux est gardée (l'autre est soit jetée, soit utilisée pour un deuxième individu) puis un individu est construit en se basant le plus possible sur les parties gardées. Par exemple, dans le cas du KP, les k premiers gènes du parent 1 sont gardés à l'identique, puis les gènes du parent 2 sont recopiés à la condition de respecter la capacité du sac-à-dos (et en échangeant le bit sinon).
- L'opérateur de mutation provoque une modification minimale respectant les contraintes du problème. Dans le cas du KP, une mutation peut consister à l'inversion d'un bit. De 1 vers 0, cela ne pose jamais de problème, mais pour inverser de 0 à 1, il faut vérifier la contrainte de capacité et supprimer un objet si elle n'est plus respectée.
- La sélection naturelle peut être déterministe ou avoir un caractère aléatoire. La sélection dite «par rang» est dans le cas de figure et sélectionne les individus avec les meilleures capacités d'adaptation (la valeur de la fonction objectif de la solution correspondante). La sélection «proportionnelle» associe une probabilité de survivre proportionnelle à la capacité d'adaptation de ces individus. Enfin, la sélection «uniforme» donne autant de chance à toute la population.

La principale force des algorithmes génétiques réside dans leur capacité à gérer des problèmes d'optimisation de grande dimension avec un grand nombre de variables de décision. Les GA peuvent facilement évoluer pour s'attaquer à des scénarios d'optimisation du monde réel en raison de leur parallélisme inhérent et de leur robustesse par rapport aux optima locaux. De plus, ils offrent une flexibilité dans le codage de diverses représentations de problèmes, y compris des variables continues et discrètes, et peuvent s'intégrer à d'autres techniques d'optimisation telles que le recuit simulé et l'évolution différentielle. Certaines applications notables incluent la conception d'antennes, l'optimisation de l'architecture d'apprentissage automatique et l'optimisation du portefeuille financier. Cependant, malgré leurs atouts, les GA manquent de contrôle précis sur le processus de recherche par rapport aux méthodes basées sur le gradient et peuvent nécessiter un réglage approfondi de paramètres tels que la taille de la population, la probabilité de croisement et les taux de mutation pour obtenir des résultats satisfaisants.

1.1.10 Apprentissage par Renforcement

L'apprentissage par renforcement (*Reinforcement Learning* - RL) est un sous-domaine de l'apprentissage automatique qui permet à des agents d'apprendre à faire de bonnes décisions basées sur des récompenses ou des punitions reçues à la suite leurs actions. Les agents RL apprennent grâce à des interactions par essai-erreur au sein d'un environnement dynamique dont ils ne voient qu'une partie, adaptant leur comportement au fil du temps pour maximiser la somme des récompenses reçues. Contrairement à toutes les méthodes présentées jusqu'à présent, l'apprentissage par renforcement est une méthode statistique et nécessite un grand nombre d'essais pour converger vers une bonne solution. Elle est cependant régulièrement adjointe à des méthodes plus classiques dans le but d'affiner la valeur de certains de leurs paramètres d'entrée.

L'apprentissage par renforcement est caractérisé par son cadre de processus de décision Markov (*Markov Decision Process* - MDP). Un MDP se compose de cinq éléments essentiels : les états, les actions, les probabilités de transition, les politiques et les récompenses. Les agents interagissent avec les environnements en sélectionnant des actions dans chaque état, en recevant des récompenses après avoir effectué ces actions et en observant les transitions qui en résultent vers de nouveaux états. Grâce à des expériences répétées, les agents mettent à jour leur politique, dans le but de maximiser la récompense cumulative à long terme. Plus formellement, le cadre d'un apprentissage par renforcement se caractérise par :

- S un ensemble d'états.
- A un ensemble d'actions réalisable par l'agent.
- $P_a(s, s') = P(S_{t+1} = s' | S_t = s, A_t = a)$ une probabilité de transition de l'état s à l'état s' à la date t lorsque l'agent fait l'action a.
- $R_a(s, s')$ une récompense immédiate après la transition de s à s' par l'action a.
- $\pi : S \times A \to [0, 1], \pi(s, a) = P(A_t = a | S_t = s)$ une politique d'action de l'agent qui tente de maximiser l'espérance des récompenses cumulées à toute date t.

Le principal avantage de l'apprentissage par renforcement est sa capacité à permettre aux agents d'apprendre de manière autonome, sans intervention humaine explicite ni données étiquetées. Cela rend RL particulièrement attrayant dans les scénarios où l'obtention de grandes quantités de données de formation est difficile ou coûteuse. De plus, RL excelle dans la gestion des espaces d'état et d'action continus, permettant son application à un large éventail de problèmes au-delà d'opérations d'optimisation discrètes traditionnelles. Enfin, RL permet le développement d'agents intelligents capables de s'adapter à des environnements changeants et d'apprendre de l'expérience, ce qui en fait un outil puissant pour la recherche et les applications en intelligence artificielle. Par exemple, ChatGPT a reçu un entraînement par renforcement pour terminer son apprentissage.

La question de l'apprentissage de la politique, et en particulier de la mémoire des triplets (s, a, r) (état, action, récompense) est un très vaste sujet et ne sera pas étendu dans le cas général. Dans le cas de l'ajout d'un processus d'apprentissage par renforcement de l'un des paramètres d'entrée d'une méthode précédente, par exemple pour apprendre à sélectionner un bon voisin dans un schéma de Recherche Locale, le cadre est :

- Un état est simplement la solution courante.
- Les actions possibles sont définies par les opérateurs de voisinage : faire une action, c'est choisir un opérateur et une valeur de voisinage.
- Les transitions sont déterministes dans notre cas : une même action depuis le même état donnera toujours le même résultat.
- La récompense immédiate est une des difficultés les plus importantes : elle peut être proportionnelle au gain ou la perte de valeur de la fonction objectif au risque de trop diriger l'agent dans des minima locaux.
- La politique peut être dirigé par différentes modélisations du système étudié :
  - La méthode du « *Q-learning* » proposé par C. Watkins dans sa thèse en 1989 (voir [Watkins, 1989]) dans laquelle il propose d'apprendre explicitement l'espérance des récompenses cumulées dans une matrice Q de taille $|S| \times |A|$. Dans le cas où le nombre d'états est exponentiel, cette méthode

n'est pas efficace.
    - La méthode du «*Deep Q-learning*» tente de répondre à ce problème en remplaçant la matrice par un réseau de neurones artificiels prenant un état en entrant et retournant une estimation de ce que devrait être la colonne de la matrice Q correspondante. Cette méthode a maintes fois prouvé son efficacité, notamment dans l'apprentissage du jeu de go avec le programme «AlphaGo Zero». Ce dernier a réussi à dominer, avec 100 parties gagnées contre 0, son prédécesseur «AlphaGo», le premier programme ayant battu un humain sur le jeu de go, en trois jours d'entraînement seulement.
    - À l'inverse des deux précédentes méthodes, il est possible d'utiliser une modélisation en utilisant les caractéristiques du problème étudié. Dans notre cas par exemple, il est possible de proposer un modèle du système dans lequel l'espérance des récompenses cumulées est linéairement dépendant de la différence des valeurs de la solution précédente et de la solution actuelle : une forte différence montre qu'il est possible d'avoir un gain important en exploitant la puissance de la recherche locale, tandis qu'une différence faible ou négative implique que la solution est actuellement dans un minimum local et nécessite plus d'exploration. Il suffit alors d'apprendre les bons coefficients d'un problème spécifique.
- À partir du modèle choisi, la politique de l'agent peut décider, ou non, de suivre la meilleure décision prédite par ce modèle. Une politique bien connue est la politique «*Epsilon-Greedy*» dans laquelle l'agent choisit de suivre le modèle avec une probabilité de $1 - \epsilon$ ou de choisir une action aléatoire avec une probabilité de ϵ. Généralement, ϵ commence à 1 au début de l'apprentissage et diminue avec les itérations pour apprendre le modèle au début et l'exploiter à la fin.
- La méthode d'apprentissage est également un très vaste sujet et dépend du modèle choisi, un réseau de neurone n'apprenant pas de la même façon qu'une simple matrice ou fonction linéaire. Dans le cas du «*Q-learning*», la matrice est remplie à chaque action prise en ajoutant, dans la case correspondante, la récompense immédiate ainsi que la meilleure espérance de récompense depuis l'état suivant. Ajouté avec un facteur d'actualisation γ, cette espérance future possible permet d'apprendre les récompenses à long terme depuis l'état s (voir algorithme 1.7).

Algorithme 1.7 : Un algorithme d'apprentissage par renforcement par la méthode du Q-learning

Entrées : Q(s,a) initialement nulle
α un coefficient d'apprentissage
γ un facteur d'actualisation
Pour un nombre d'époques fini (l'apprentissage est répété plusieurs fois) **Faire**
Initialiser l'état courant s
Pour tout étapes de l'époque (fini ou jusqu'à ce que s soit un état final **Faire**
Choisir une action a depuis s en utilisant la politique $\epsilon - greedy$
Appliquer a et observer la récompense r et le nouvel état s'
$Q(s,a) = Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(S',a') - Q(s,a)\right)$
$s = s'$
Fin Pour
Fin Pour

Malgré son potentiel, l'apprentissage par renforcement se heurte à des défis importants qui entravent son adoption généralisée. Ces défis incluent la nécessité de disposer de quantités massives de données pour obtenir des résultats satisfaisants, des taux de convergence lents et des difficultés liées à la gestion des distributions multimodales. De plus, le recours de RL à des modèles de récompense précis peut s'avérer problématique lorsque la définition des signaux de récompense corrects est complexe ou ambiguë.

1.2 Les cibles d'applications

1.2.1 Les problèmes de tournées de véhicules

Parmi ces problèmes NP-difficile très étudiés, modélisables en MILP et donc solvables par *Branch&Bound*, il y a le Problème de Tournées de Véhicules (*Vehicle Routing Problem* - VRP [Laporte and Nobert, 1987]). C'est l'un des problèmes d'optimisation combinatoire les plus étudiés grâce au très grand nombre d'applications réelles que modélisent ses variantes. Ce problème peut être défini de la manière suivante : un ensemble de clients doit être visité au moyen d'une flotte fixée de véhicules partant d'un dépôt et la somme des coûts des tournées doit être minimale (voir figure 1.9).

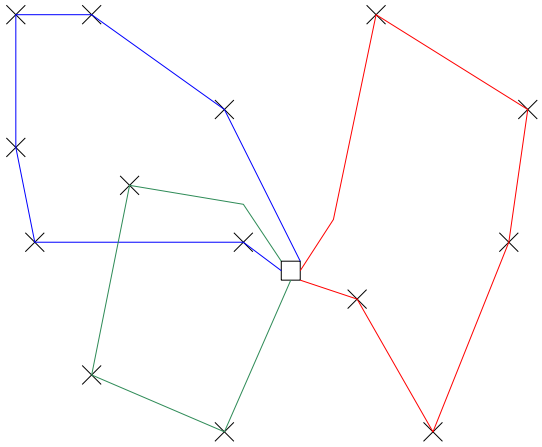

FIGURE 1.9 – Un exemple de tournées de véhicules avec un dépôt central et trois véhicules (bleu, rouge et vert).

Historiquement, le VRP est une version étendue du Problème du Voyageur de Commerce (*Traveling Salesman Problem* - TSP), qui consiste à visiter l'ensemble des clients avec un seul véhicule. Ici, pas de dépôt, l'objectif est de construire un circuit passant par tous les clients et les clients peuvent être visité dans n'importe quel ordre. L'origine de ce problème est assez floue puisque des exemples de circuits ont été retrouvés dans un manuscrit de 1832, mais pas de formulations mathématiques n'ont été écrites avant W. Hamilton et T. Kirkman dans les années 1930 (voir [Biggs et al., 1999]) et nommé, quelques années plus tard par le mathématicien autrichien Karl Menger, *botenproblem* : le problème du messager (voir [Schrijver, 2005]).

Une des difficultés majeures apparaissant lors de la modélisation du TSP en ILP est le nombre exponentiel des contraintes dites de «sous-tours». En effet, ces contraintes garantissent qu'un seul cycle ne sera construit. En dehors des contraintes de sous-tours, un ILP pour résoudre le TSP peut être formulé ainsi :

$$
\begin{aligned}
\textbf{minimiser} \quad & \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} c_{i,j} x_{i,j} & \\
\textbf{tel que} \quad & \sum_{j=1, j \neq i}^{n} x_{i,j} = 1 & i = 1, \ldots, n \\
& \sum_{i=1, i \neq j}^{n} x_{i,j} = 1 & j = 1, \ldots, n \\
& x_{i,j} \in \{0, 1\}, \quad i = 1, \ldots, n, j = 1, \ldots, n &
\end{aligned}
$$

avec $x_{i,j}$ une variable binaire égale à 1 si l'arc entre le nœud i et le nœud j est emprunté par le voyageur de commerce et 0 sinon. L'objectif est de minimiser le coût total de la tournée et les contraintes s'assurent que la tournée passe une et une seule fois par tous les nœuds. Le problème est que, sans autre contrainte, plusieurs tournées peuvent être construites au lieu d'une seule (voir figure 1.10).

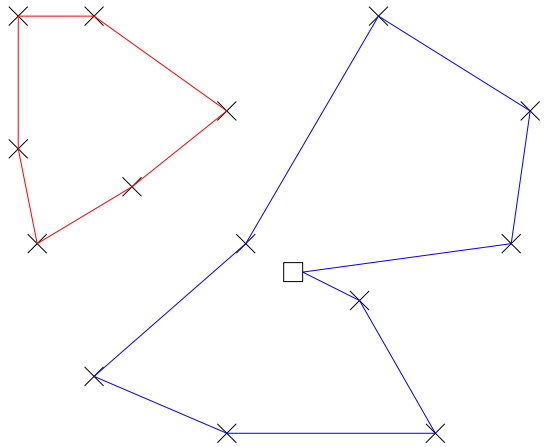

FIGURE 1.10 – Une solution avec un sous-tour. Ce n'est pas une solution valide du TSP.

Il faut alors ajouter les contraintes de sous-tours qui ont été proposées sous différentes formes, comme celle de G. Dantzig, R. Fulkerson et S. Johnson en 1954 (voir [Dantzig et al., 1954]) :

$$\sum_{i \in Q} \sum_{j \in Q, j \neq i} x_{i,j} \leq |Q| - 1 \quad \forall Q \subsetneq \{1, \ldots, n\}, |Q| \geq 2$$

Ici, en prenant Q l'ensemble des nœuds du sous-tour en rouge dans la figure ci-dessus, la somme des $x_{i,j}$ est égale à 6 alors que $|Q| - 1 = 5$. La contrainte est ici violée, ce qui empêche la formation de sous-tours. Cependant, cette contrainte doit être exprimée pour tout sous-ensemble de nœuds, c'est-à-dire 2^n fois. En pratique, elles ne sont jamais exprimées explicitement et une méthode constructive vérifie, pendant la résolution, que toutes les contraintes de sous-tours sont vérifiées de façon intelligente et un petit nombre de contraintes est ajouté au modèle ILP le cas échéant (voir [Balas and Toth, 1983]).

Le TSP puis le VRP a été dérivé de nombreuses fois, chacune introduisant des éléments de complexités supplémentaires. Ces variations ont pour but de créer des modèles toujours plus réalistes de scénarios réels. Parmi les variantes les plus notables, il y a :

- *Capacited VRP (CVRP)* : minimise la distance totale parcourue tout en garantissant que chaque nœud est desservi exactement une fois et en respectant les contraintes de capacité des véhicules (voir [Fischetti et al., 1994]).
- *VRP with Time Windows (VRPTW)* : ajoute des contraintes de fenêtre horaire à chaque nœud, en se concentrant sur la satisfaction des attentes des clients concernant les heures de début et de fin du service (voir [Gambardella et al., 1999]).
- *Dial-a-Ride (DARP)* : fournit des services porte-à-porte flexibles utilisant plusieurs véhicules, en tenant compte des préférences et des priorités des passagers (voir [Madsen et al., 1995]).
- *Periodic VRP (PVRP)* : optimise l'itinéraire pour les livraisons/collectes récurrentes sur un horizon de planification fixe, en tenant compte de facteurs tels que les différentes capacités des véhicules, les fenêtres horaires et les fréquences de service (voir [Gaudioso and Paletta, 1992]).
- *Multi-Depot VRP (MDVRP)* : permet d'avoir plusieurs dépôts au lieu d'un seul, permettant un meilleur placement du centre de distribution et réduisant les coûts de transport globaux (voir [Renaud et al., 1996]).

Au-delà du VRP, il existe de nombreuses extensions et variantes intéressantes qui répondent aux exigences uniques de divers secteurs. Par exemple, le problème de ramassage et de livraison (*Pickup and Delivery Problem* - PDP) qui implique la collecte de marchandises d'un endroit et leur livraison à un autre, ce qui nécessite un acheminement efficace entre les paires origine et destination (voir [Berbeglia et al., 2007]). Ou encore, le problème d'affectation quadratique (*Quadratic Affectation Problem* - QAP) qui peut modéliser un problème d'affectation de procédés de fabrication à des sites physiques. La complexité vient de l'interdépendance des procédés qui oblige à vérifier des contraintes quadratiques (c'est-à-dire non

linéaire) pour le choix de l'affection. L'objectif étant de minimiser les coûts de transport entre les sites, il faut donc bien choisir les différentes affectations pour ne pas trop éloigner les procédés interdépendants. Ce problème, très complexe, combine à la fois les aspects d'itinéraire et d'affectations des procédés (voir [Koopmans and Beckmann, 1957]). Ces formulations avancées de problèmes soulignent encore davantage l'importance du VRP pour relever des défis logistiques complexes, expliquant le grand nombre de travaux sur ce problème.

1.2.2 Les problèmes de dimensionnement des lots

Le problème de dimensionnement des lots (*Lot Sizing Problem* - LSP) est un problème de planification de la production bien connu dans les systèmes de gestion de la production. Il s'agit de déterminer la quantité optimale de produit à fabriquer dans chaque période de production, en tenant compte de facteurs tels que les coûts d'activation de la production et les coûts de stockage, ainsi que les limites de capacité des stocks. L'objectif étant de minimiser le coût total dans un horizon de temps donné et l'ensemble des quantités à produire est appelé un plan de production.

En d'autres termes, l'objectif est de proposer un plan de production permettant de satisfaire la demande en balançant les coûts de production et de stockage. Le plus gros problème étant les coûts d'activation de la production : si une quantité non nulle de produit est fabriquée durant une période, un coût d'activation doit être payé pour cette période. Ce coût est le même quelle que soit la quantité produite (voir figure 1.11).

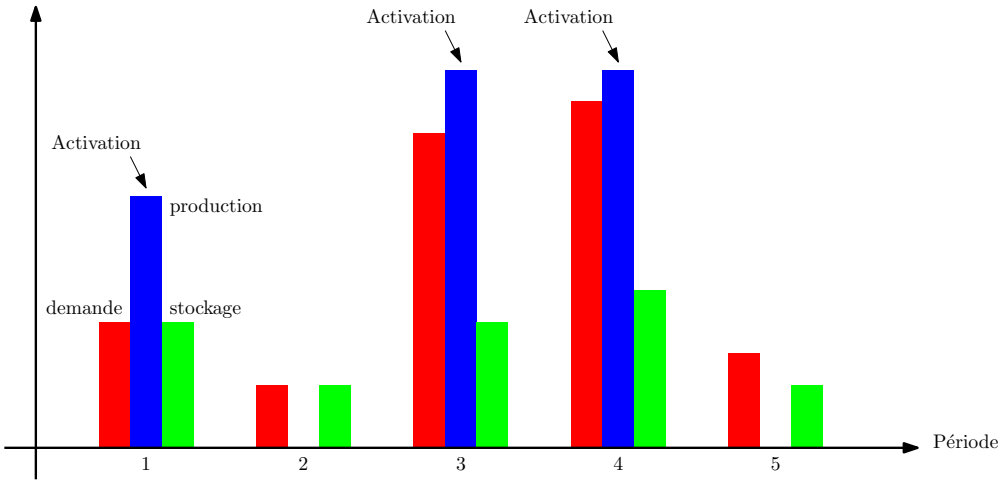

FIGURE 1.11 – Exemple d'instance et de solution réalisable à un LSP. La demande est en rouge, la production est en bleu et l'état du stock à la fin de la période est en vert.

Un LSP très utilisé et étudié est le LSP avec contrainte de capacité (*Capacited LSP* - CLSP). En considérant les variables et données suivantes :

- x_i est une variable réelle qui représente la quantité fabriquée à la période i.
- y_i est une variable binaire qui vaut 1 si la production est démarrée à la période i et 0 sinon.
- q_i est une variable réelle qui représente la quantité dans le stock à la fin de la période i.

- $\{1, \ldots, T\}$ est l'horizon de temps
- q_0 est la quantité de produit initialement dans le stock.
- Q est la capacité maximale du stock.
- P_i est la quantité maximale de production à la période i.
- C_i^x est le coût de production unitaire à la période i.
- C_i^y est le coût d'activation de la production à la période i.
- C_i^Q est le coût de stockage unitaire de la période i.
- d_i est la demande de produit de la période i.

Le CLSP peut être modélisé sous la forme du MILP suivant :

$$
\begin{aligned}
\textbf{minimiser} \quad & \sum_{i=1}^{T} (C_i^x x_i + C_i^y y_i + C_i^Q q_i) & \\
\textbf{tel que} \quad & x_i \leq P_i y_i & i = 1, \ldots, T \\
& q_i = x_i + q_{i-1} - d_i & i = 1, \ldots, T \\
& 0 \leq q_i \leq Q & i = 1, \ldots, T \\
& x_i \in \mathbb{R} & i = 1, \ldots, T \\
& q_i \in \mathbb{R} & i = 1, \ldots, T \\
& y_i \in \{0, 1\} & i = 1, \ldots, T
\end{aligned}
$$

C. Gicquel, M. Minoux et Y. Dallery ont publié en archive ouverte un état de l'art très complet des méthodes de résolution du CLSP (voir [Gicquel et al., 2008]) séparé en plusieurs sections, chacune présentant une variante du CLSP (simple ou multiniveaux avec une ou plusieurs ressources).

1.2.3 Les problèmes d'ordonnancement

Les problèmes d'ordonnancement (*Scheduling Problems*), et en particulier les problèmes d'ordonnancement d'atelier, sont d'autres problèmes de planification tournés, cette fois-ci, sur l'ordre et l'assignation d'opérations à une ou plusieurs machines : Étant donné un ensemble d'opérations, appelé *job*, chacune nécessitant un certain temps de traitement sur une ou plusieurs machines, l'objectif est de déterminer la séquence et l'affectation d'opérations aux machines afin de minimiser la durée de fabrication totale, le retard maximum ou d'autres mesures temporelles ou économiques.

Une vue d'ensemble des principaux problèmes d'ordonnancement a été proposée par H. Toussaint dans sa thèse (voir [Toussaint, 2010]) et est présentée dans la figure 1.12.

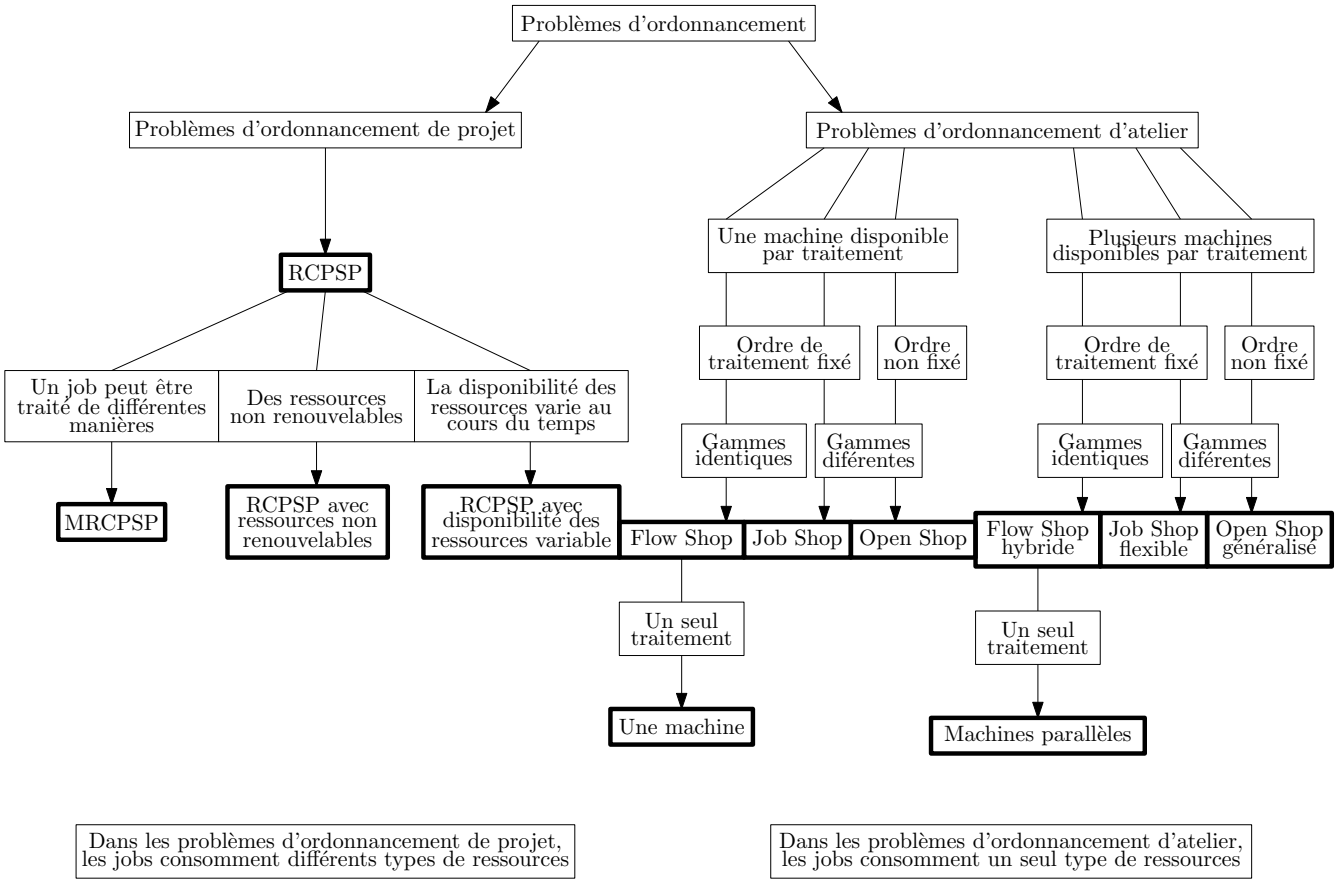

FIGURE 1.12 – Schéma des principaux problèmes d'ordonnancement, reproduit de [Toussaint, 2010]

E. Taillard a réalisé un état de l'art très complet en 1993 pour les problèmes de *Flow Shop*, *Job Shop* et *Open Shop* (voir [Taillard, 1993]). Ce sont les trois problèmes principaux de l'ensemble des problèmes d'ordonnancement d'atelier. Ils se distinguent par les contraintes sur l'ensemble des opérations, appelé «gamme opératoire».

- Le *Flow Shop* est le plus contraint des trois problèmes : Chaque pièce doit passer par toutes les machines dans l'ordre défini par une gamme opératoire commune.
- Dans le *Job Shop*, chaque pièce à sa propre gamme opératoire et l'ordre des opérations doit être respecté. La grande majorité des variantes de *Flow Shop* et de *Job Shop* sont NP-complet comme l'ont montré M. Garey, D. Johnson et R. Sethi en 1976 (voir [Garey et al., 1976]).
- Dans l'*Open Shop*, chaque pièce à sa propre gamme opératoire et les opérations peuvent être faites dans n'importe quel ordre. Le problème d'ordonnancement d'une seule tâche sur plusieurs machines, en supposant des durées d'activation différentes en fonction de la machine précédente, est équivalent au TSP (les villes sont les machines et le voyageur est la tâche), l'*Open shop* est donc également NP-complet.

Une représentation des solutions de problème d'ordonnancement, régulièrement utilisée car simple à comprendre, est le diagramme de Gantt (voir figure 1.13). Cette représentation peut prendre différentes formes en fonction du problème étudié, mais est généralement présentée comme un ensemble de rectangle dans un plan dont l'un des deux axes est temporel. Le deuxième axe peut être : les machines dans le cas d'un problème d'ordonnancement d'atelier, les ressources consommées dans le cas d'un problème d'ordonnancement de projet, etc.

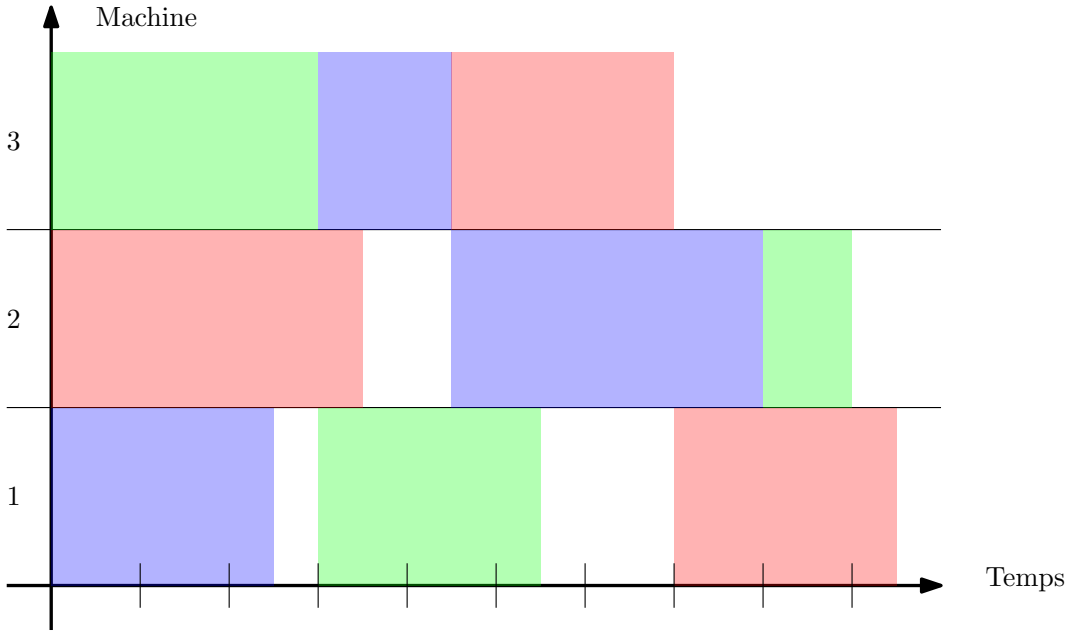

FIGURE 1.13 – Exemple de diagramme de Gantt pour un problème d'*Open Shop* avec trois pièces (rouge, vert et bleu) et trois machines.

1.2.4 La dépendance au temps et les problèmes de synchronisation

Ces problèmes dépendant du temps font référence à des situations dans lesquelles le comportement du système ou la prise de décision doivent prendre en compte le passage du temps. Ils sont omniprésents dans des domaines variés allant des systèmes de contrôle du trafic aux réseaux de communication et aux processus de fabrication, nécessitant une gestion précise des événements et des actions. Lors de leur étude, les problèmes dépendant du temps impliquent la modélisation de la dynamique du système et la conception d'algorithmes capables de s'adapter aux conditions changeantes au fil du temps. Quelques

exemples courants de problèmes dépendant du temps incluent, comme présenté ci-dessus, les problèmes d'évacuation de foules dans des contextes d'urgences, mais aussi l'optimisation du routage de paquets dans un réseau informatique dont les temps de transit sont variables. La résolution de ces problèmes nécessite un examen attentif des aspects temporels, garantissant que le système reste réactif et efficace malgré les changements de l'environnement.

Une première dépendance au temps assez connue est la longueur ou le coût variable des arcs d'un réseau de transport. Ces ralentissements peuvent être causés par des défaillances, comme l'ont traité M. Zhang, R. Batta et R. Nagi dans le cadre d'un problème de transport de produits dans un entrepôt dans lequel des interruptions involontaires d'équipements peuvent se produire (voir [Zhang et al., 2009]). Ces embouteillages peuvent aussi être une donnée d'entrée, comme l'ont été traités G. Lera-Romero et J. Miranda-Bront qui ont proposé une méthode de génération de colonne pour traiter un problème du chemin élémentaire dépend du temps avec contraintes de ressource (voir [Lera-Romero and Miranda-Bront, 2018]). Mais il existe de nombreuses autres raisons (voir [Franceschetti et al., 2017, Ziliaskopoulos and Mahmassani, 1993]).

Il existe également des réseaux de transport qui se modifient en fonction du temps, c'est le cas des problèmes d'évacuations. Par exemple, J. Jarvis et H. Ratliff ont proposé une méthode pour minimiser trois objectifs simultanément sur un réseau dans lequel la capacité de certains arcs peut changer au cours du temps et parfois devenir nulle (voir [Jarvis and Ratliff, 1982]).

Une dernière contrainte dépendante du temps, qui va particulièrement nous intéresser, est la synchronisation des acteurs d'un problème. Un besoin de synchronisation survient lorsque plusieurs entités doivent coordonner leurs actions pour différentes raisons : maintenir la cohérence au sein d'un système, accommoder une panne ou une urgence ou si la préemption des marchandises est autorisée. La préemption introduit une complexité supplémentaire aux problèmes, déjà complexes, des tournées de véhicules et du transport de biens et services. Lorsqu'elle est autorisée, les marchandises ou les personnes transportées peuvent être acheminées par plusieurs véhicules. Une synchronisation entre deux véhicules est absolument nécessaire pour permettre un échange (voir figure 1.14). Ces synchronisations génèrent des temps d'attente qu'il faut minimiser ou contraindre. En 2020, S. Humagain, R. Sinha, E. Lai et P. Ranjitkar ont proposé un état de l'art des méthodes d'optimisation pour réduire le temps de trajet des véhicules d'urgences comme les ambulances, les véhicules de police et les pompiers, quand la préemption est autorisée (voir [Humagain et al., 2020]).

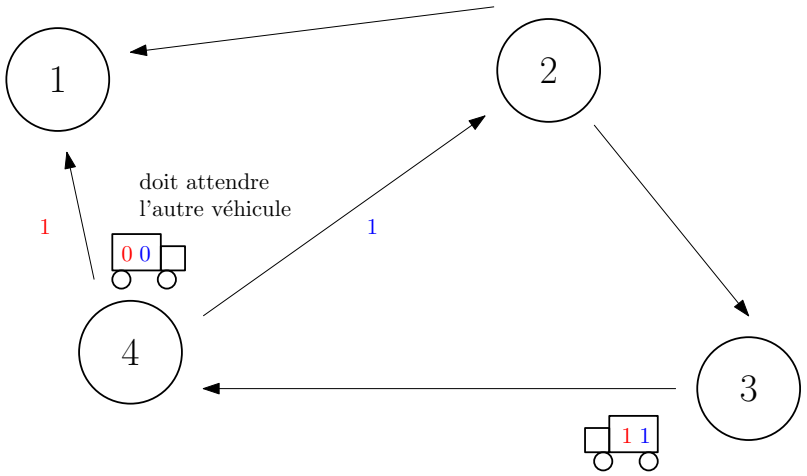

FIGURE 1.14 – Le camion présent sur le nœud 4 doit attendre l'autre véhicule pour échange l'une des deux marchandises et ainsi satisfaire les deux demandes en même temps.

Une des façons de traiter les problèmes dépendant du temps est de les modéliser par un réseau étendu dans le temps (*Time-Expanded Network* - TEN).

1.2.5 Les réseaux *Time-Expanded*

À partir d'un graphe et un horizon temporel donnés, il est possible de construire TEN. C'est un grand réseau statique dans lequel toutes les caractéristiques temporelles sont exprimées, permettant une analyse et une optimisation efficaces à l'aide d'algorithmes graphes traditionnels. Développé par L. Ford and D. Fulkerson à la fin des années 50 (voir [Ford and Fulkerson, 1958]), le modèle TEN a depuis gagné en popularité en raison de sa polyvalence et de son applicabilité dans divers domaines. Il fournit un moyen systématique de représenter et de raisonner sur des systèmes dynamiques confrontés à des contraintes dépendantes du temps.

L'idée centrale du TEN réside dans l'expansion de chaque événement le long de l'axe chronologique en une séquence de nœuds artificiels, reliés par des arcs dirigés. Ces nœuds représentent des états ou des configurations distinctes du système à des moments précis. Les transitions entre ces états sont capturées par les arcs, indiquant les actions ou opérations possibles qui peuvent se produire au cours du processus d'expansion. La structure de réseau résultante conserve les caractéristiques essentielles du système d'origine, mais permet une représentation plus complète. Pour être plus précis (voir figure 1.15) :

- Un nœud du TEN correspond à un couple nœud-date. Être sur un nœud du TEN c'est équivalent à être sur un nœud du graphe initial à une certaine date.
- Un arc du TEN correspond à la traverser d'un arc du graphe initial à une certaine date.

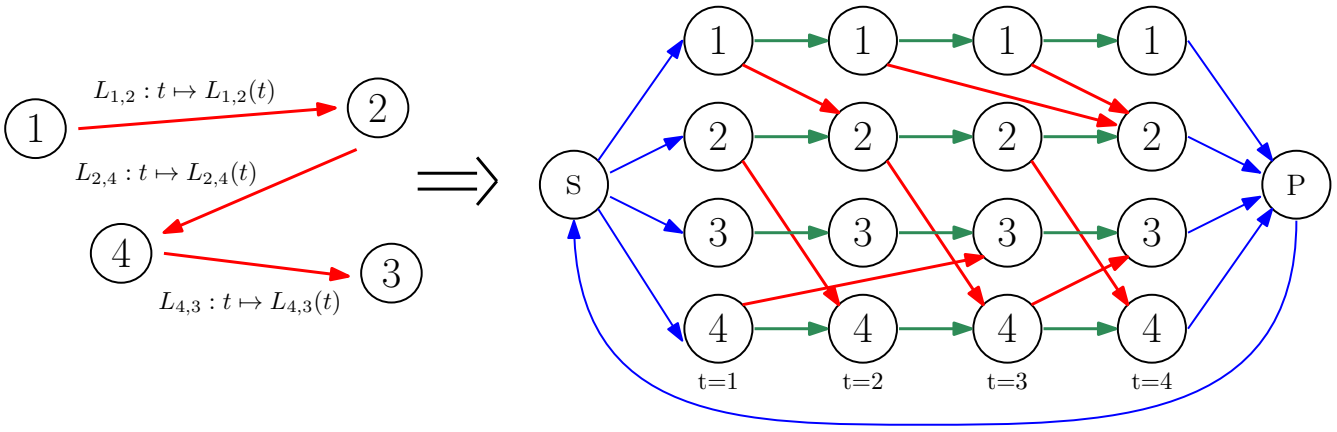

FIGURE 1.15 – Transformation d'un graphe et d'un horizon de temps en un TEN

De ce fait, l'utilisation du TEN permet l'application de concepts de la théorie des graphes tels que l'analyse de l'accessibilité, la détermination du chemin le plus court et l'exploration de l'espace d'état pour résoudre des problèmes complexes de planification et d'ordonnancement dépendant du temps.

Malgré ses avantages, le TEN se heurte à un problème majeur : sa taille. En effet, la construction d'un réseau étendu dans le temps conduit à une augmentation exponentielle de la taille du réseau liée à l'horizon temporel. Ce problème de mise à l'échelle rend difficile l'application du TEN à des systèmes trop grands. En pratique, le TEN est rarement exprimé dans sa totalité, en particulier pour les problèmes en temps continu où ce n'est pas possible. Il est alors nécessaire de mettre en place une résolution permettant la construction du graphe pendant la recherche, comme une résolution par génération de colonnes par exemple.

À l'inverse, L. Ford and D. Fulkerson ont proposé, en même temps que le modèle de réseaux *time-expanded* (voir [Ford and Fulkerson, 1958]), un modèle *Flow Over Time* (FOT). Dans ce modèle, le réseau est figé, mais les flots sont des fonctions du temps. Le temps de transit d'un arc spécifie le temps nécessaire au flot pour se déplacer d'un bout à l'autre de cet arc, mais contrairement aux flots statiques classiques, un FOT spécifie un débit entrant dans un arc pour chaque instant qui doit ressortir avec le même débit après avoir traversé l'arc.

1.2.6 Les problématiques liées aux véhicules autonomes

Une autre application des réseaux temporels, toujours dans le contexte des problèmes de transport de biens et services, est liée aux véhicules autonomes. Puisque, en plus des contraintes liées au transport, de nouvelles difficultés liées aux véhicules s'ajoutent.

Les véhicules autonomes (VA) suscitent une attention considérable depuis ces dernières années en raison de leur potentiel à révolutionner les secteurs du transport et de la logistique. Les VA sont conçus pour fonctionner de manière indépendante, sans intervention humaine directe, en utilisant des capteurs avancés, des systèmes de perception et des algorithmes pour naviguer dans leur environnement en toute sécurité. Cependant, malgré les progrès réalisés dans ce domaine, les VA sont encore confrontés à de nombreux défis et problèmes spécifiques qui doivent être résolus avant d'être largement adoptés.

L'un des principaux obstacles rencontrés par les VA est d'assurer la sécurité autour d'eux. Contrairement aux humains qui s'appuient sur leur intuition et leur expérience, les VA dépendent entièrement de leurs apports sensoriels et de leurs capacités de prise de décision. Garantir la fiabilité et la robustesse des systèmes de perception et de contrôle reste essentiel. Une perception précise de l'environnement nécessite des capteurs sophistiqués, tels que le LiDAR, les caméras et les radars, qui doivent fonctionner ensemble de manière transparente. L'un des grands défis réside dans la gestion des cas marginaux, des conditions météorologiques extrêmes et des situations inattendues susceptibles de dégrader les performances des capteurs (voir [Courcelle et al., 2022]).

Le deuxième défi des VAs est la difficulté de la supervision d'une flotte de VA. La tendance actuelle est à la mise en place d'une architecture de supervision à trois niveaux :

- Le premier niveau, ou niveau embarqué, est défini par les dispositifs de supervision et de détection qui sont embarqués à l'intérieur des véhicules dans le but de résoudre les problèmes liés à la robotique. Ces problèmes spécifiques aux véhicules autonomes peuvent être de différentes natures : les problèmes de suivi de trajectoire ou les procédures de récupération d'objets par exemple. Actuellement, la plupart des efforts de la communauté robotique sont consacrés à ce niveau embarqué (voir [Chen and Englund, 2016, Martínez-Barberá and Herrero-Pérez, 2010]), et concernent principalement les techniques de contrôle optimal et de perception artificielle.
- Le second, ou niveau intermédiaire, est en charge des petites zones partagées et dangereuses, comme les intersections (voir figure 1.16). Les superviseurs locaux gèrent les priorités entre les véhicules autonomes et résolvent les conflits dans ces zones restreintes afin de réguler le flot de VA et d'éviter toute collision VA/VA ou VA/piéton (voir [Chen and Englund, 2016, Philippe et al., 2019]).
- Le troisième, ou niveau global, fait référence à la planification et au routage dynamiques de la flotte, afin de faire en sorte que cette flotte réponde à certaines demandes de logistique interne. Ce niveau doit tenir compte des niveaux inférieurs pour calculer sa propre solution. Par exemple, [Wurman et al., 2008] calcule le plus court chemin grâce à l'algorithme A*, mais affecte chaque tâche à la flotte de véhicules autonomes en utilisant une intelligence artificielle multiagents afin d'éviter au maximum les conflits dans les arcs et aux croisements (voir également [Le-Anh and De Koster, 2006, Vis, 2006, Koes et al., 2005]).

Un véritable défi concernant les véhicules autonomes est la synchronisation de ces différents niveaux de supervision et le contrôle des processus de communication qui leur permettront d'interagir efficacement. Une communication fiable entre les différents niveaux de supervision est cruciale pour assurer la coordination et la synchronisation des tâches effectuées par les véhicules autonomes. Le développement et la mise en œuvre de protocoles et d'algorithmes de communication appropriés seront nécessaires pour atteindre cet objectif. La planification et le contrôle en temps réel nécessitent des calculs complexes impliquant souvent une optimisation multiobjectifs. Parmi ces objectifs, en plus d'indicateurs classiques comme la distance parcourue ou la durée totale du planning, des facteurs tels que les règles de circulation, la sécurité des piétons et la consommation d'énergie sont utilisés dans ce contexte. L'optimisation de ces compromis devient de plus en plus difficile à mesure que la complexité des scénarios augmente. En outre, garantir des considérations éthiques dans les processus de prise de décision, en particulier lorsque l'on est confronté à

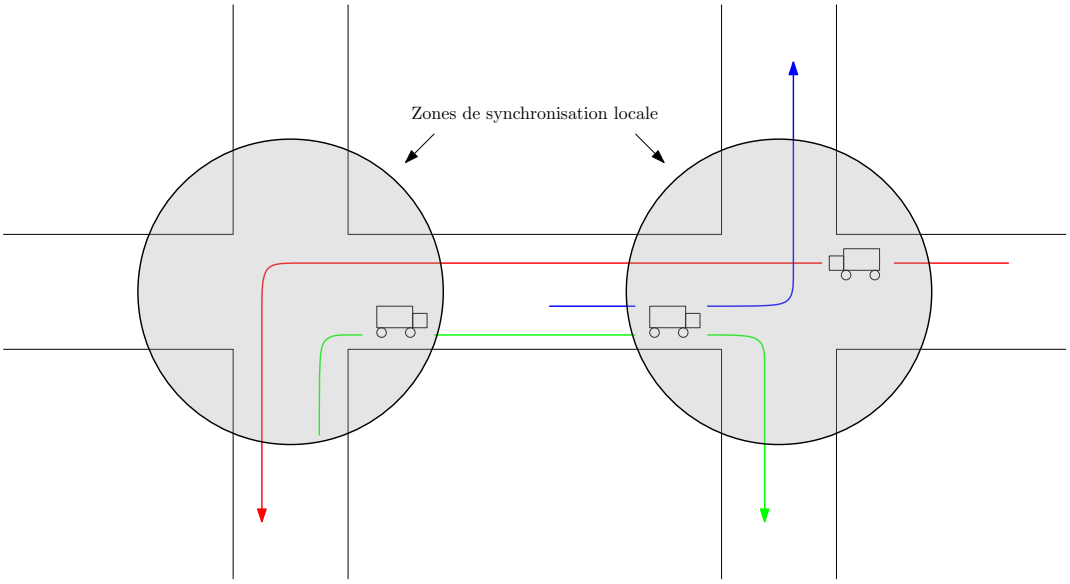

FIGURE 1.16 – Niveau de décision intermédiaire : exemple d'intersections d'allées à double sens.

des situations mettant la vie en danger, présente des dilemmes moraux qui nécessitent un examen attentif. La collaboration internationale entre les gouvernements, les acteurs industriels et les instituts de recherche joueront un rôle essentiel dans l'élaboration du futur paysage de la mobilité autonome.

Les véhicules autonomes soulèvent également plusieurs préoccupations sociétales, notamment l'atteinte à la vie privée en raison de la collecte de données personnelles sur les déplacements des personnes, le recours accru à la technologie pouvant conduire à des modifications d'emplois et les dilemmes éthiques liés à la prise de décision lors d'accidents impliquant des vies humaines. À mesure que ces véhicules deviennent plus répandus, les sociétés doivent faire face à ces défis et établir des lignes directrices pour atténuer les impacts négatifs potentiels tout en maximisant les avantages. La collaboration entre les gouvernements, les experts de l'industrie et le public est essentielle avant l'acceptation généralisée des VA.

Les défis à relever pour avoir des VA pleinement fonctionnels et sûrs sont donc nombreux et de toutes natures : techniques, juridiques et sociétales. Dans le contexte technique, des progrès continus dans les domaines de l'intelligence artificielle, de la vision par ordinateur et de la robotique sont nécessaires pour résoudre les problèmes embarqués. Tandis que la gestion du risque et d'autres facteurs dépendant du temps sont inévitables pour résoudre les problèmes de décisions locaux ou de gestion d'une flotte de VA.

1.3 Contexte universitaire

Mes travaux se sont concentrés sur les problèmes de synchronisation de véhicules et VA dans le contexte du transport de marchandises. Ce champ d'étude est très actif depuis quelques années, notamment à l'Université Clermont Auvergne et le labex IMobS3 (label d'excellence) avec lequel je ne suis pas lié, mais dont les thématiques sont très fortement corrélées aux miennes.

Cette thèse est découpée en trois chapitres, chacun décrivant un problème distinct que j'ai étudié. Le premier est un problème de plus court chemin sécurisé dans le cadre de transport de marchandises par des VA dans un environnement contrôlé. Afin de sécuriser au mieux les déplacements, le chemin et la vitesse d'un tel véhicule doit pouvoir s'adapter au contexte à tout instant. Le deuxième présente une nouvelle méthode pour résoudre des problèmes dépendant du temps en passant par une projection du problème et une reconstruction de la solution a postériori. Le problème de relocation préemptif de marchandises sera utilisé comme exemple pour présenter cette méthode. Enfin, le troisième problème cherche à synchroniser deux acteurs pour transférer des marchandises nécessaires à l'un et produite par l'autre.

Chapitre I

Le Problème du Plus Court Chemin Sécurisé

Dans cette partie, nous étudions l'organisation et les trajets d'une flotte entière de véhicules autonomes dans un entrepôt. Nous décomposons l'étude de cette flotte en plusieurs problèmes imbriqués jusqu'à présenter un problème minimal : l'ajout d'un nouveau robot à une flotte existante dont les tâches et les trajets sont déjà fixés. Ce robot doit se diriger d'un point d'origine à un point de destination, mais le chemin emprunté doit être le plus court et le moins risqué possible. Nous définirons le risque généré par la flotte et le risque pris par le nouveau véhicule nous permettant d'expliquer comment remonter au problème de dimensionnement de la flotte complète.

Le problème minimal est traité à l'aide de deux heuristiques : une résolution où chemin et vitesse sont découplés et une résolution où ils sont croisés. À ces deux méthodes sont associées trois manières de calculer la vitesse des véhicules à tout instant ainsi qu'un apprentissage par renforcement d'un des paramètres communs aux deux méthodes. Puis, nous proposons deux nouvelles méthodes de résolution par apprentissage qui se basent sur l'heuristique gloutonne.

Nous comparons toutes ces méthodes entre elles et en tirons des conclusions.

Table des matières

Liste des notations

Graphe :

$G = (N, A)$	Un graphe de transit
$e = (u, v)$	Un arc entre les noeud u et v
L_e	Le temps de parcours minimal de l'arc e
$L^*_{u,v}$	La longueur du plus court chemin entre u et v
Π^e	La fonction de danger de l'arc e
v_{max_e}	la vitesse maximal autorisée sur l'arc e (si normalisée : égale à 1)

Chemin :

Γ	Un chemin
(o, d)	Un couple (origine, destination)
t	Une date
$t \mapsto v(t)$	Une fonction de vitesse (si normalisée : entre 0 et 1)
H	La fonction liant la vitesse et le risque pris par V
$Risk(\Gamma, v)$	Valeur de risque pris par V en suivant le chemin Γ à la vitesse $t \mapsto v(t)$
$Time(\Gamma, v)$	Temps de parcours de V en suivant le chemin Γ à la vitesse $t \mapsto v(t)$

Section 2

Introduction

Les véhicules autonomes (VA) sont récemment devenus un centre important d'attention dans le domaine de la mobilité. Reste à déterminer ce qu'il est exactement possible de faire avec les nouvelles générations de véhicules autonomes ou semi-autonomes. Puisqu'ils ne sont physiquement liés à aucun type de voie (câble, rail, ...), ils génèrent de nouveaux problèmes de routage, mais aussi de sûreté et de sécurité. Ils offrent une nouvelle approche du transport qui pourrait améliorer la sécurité, réduire les embouteillages et accroître l'efficacité. Cependant, plusieurs défis doivent encore être relevés avant une adoption généralisée. Parmi ces défis, les plus importants sont notamment de développer les technologies de détection avancées, d'assurer la sécurité et la fiabilité sur la route et de répondre aux questions de perception du public. Malgré ces défis, les VA ont le potentiel de révolutionner les transports et de façonner l'avenir de la mobilité urbaine.

Cependant, à moyen terme, la plupart des gens, y compris dans le monde socio-économique et académique, doutent de la perspective de voir de tels véhicules circuler sans aucun contrôle extérieur dans des zones urbaines surpeuplées. Au lieu de cela, ils prévoient que l'utilisation de ces véhicules sera limitée aux zones protégées à des fins spécifiques : déplacement des véhicules en libre accès à l'intérieur de vastes zones de stationnement, opérations de ramassage et de livraison à l'intérieur d'entrepôts (voir [Amazon, 2022, Ren et al., 2018]), ou encore, interventions de sauvetage ou de réparation dans un contexte de catastrophe naturelle pour n'en citer que quelques-uns.

Ce point de vue pose le défi général de la supervision d'une flotte de tels véhicules, dédiée à effectuer des tâches de logistique interne tout en interagissant en toute sécurité avec d'autres acteurs : ouvriers, machines et véhicules standards. Ces problèmes de décision connexes se situent à l'intersection de la Robotique et de la Recherche Opérationnelle. Pour les résoudre, des algorithmes et des techniques avancés issus de divers domaines, notamment l'intelligence artificielle et l'optimisation, sont nécessaires. En outre, le développement et l'intégration de systèmes de navigation et de perception fiables sont essentiels pour garantir le fonctionnement sûr des VAs dans les entrepôts et autres zones contrôlées.

Figure 2.1 – un véhicule autonome

La supervision d'une flotte de véhicules autonomes repose généralement sur une supervision hiérarchique. La tendance actuelle est à la mise en place d'une architecture de supervision à trois niveaux :

- Le premier niveau, ou niveau embarqué, est défini par les dispositifs de supervision et de détection qui sont embarqués à l'intérieur des véhicules dans le but de résoudre les problèmes liés à la robotique (voir [Chen and Englund, 2016, Martínez-Barberá and Herrero-Pérez, 2010])
- Le second, ou niveau intermédiaire, est en charge des petites zones partagées et dangereuses, comme les carrefours (voir [Chen and Englund, 2016, Philippe et al., 2019]).

- Le troisième, ou niveau global, fait référence à la planification et au routage dynamiques de la flotte, afin de faire en sorte que cette flotte réponde à certaines demandes de logistique interne. Ce niveau doit tenir compte des niveaux inférieurs pour calculer sa propre solution (voir [Wurman et al., 2008, Vis, 2006, Koes et al., 2005]).

Un véritable défi concernant les véhicules autonomes est la synchronisation de ces différents niveaux de supervision et le contrôle des processus de communication qui leur permettront d'interagir efficacement. Une communication fiable entre les différents niveaux de supervision est cruciale pour assurer la coordination et la synchronisation des tâches effectuées par les véhicules autonomes. Le développement et la mise en œuvre de protocoles et d'algorithmes de communication appropriés seront nécessaires pour atteindre cet objectif.

Nous traitons ici du niveau de contrôle global, en supposant que ce niveau soit en charge des décisions d'acheminement et de planification des véhicules et qu'un protocole de communication adapté permet la récupération sans erreurs d'informations des autres niveaux. Un véhicule autonome, jusqu'alors inactif, est choisi pour effectuer de nouvelles tâches. À première vue, on peut considérer le problème connexe comme une sorte de *Pickup and Delivery Problem* (PDP) (voir [Berbeglia et al., 2007]), puisqu'une tâche élémentaire consistera pour un véhicule à se déplacer d'une origine à une destination, en effectuant une opération de chargement, de déchargement ou de maintenance. Cette tâche doit être menée à bien rapidement, mais le VA ne doit pas prendre trop de risques. De nombreux articles proposent des techniques pour résoudre les problèmes de plus court chemin contraints ([Lozano and Medaglia, 2013] par exemple). Mais d'autres méthodes peuvent être étudiées : [Ryan et al., 2020] ont enregistré tous les événements dangereux se produisant dans une période donnée sur les routes de Munster en Irlande, puis ont catégorisé les routes sur une échelle de dangerosité en fonction du nombre d'événements survenus dessus. Enfin, ils ont utilisé une somme pondérée du temps et de la catégorie de risque pour calculer un chemin le plus court sûr à l'aide d'un algorithme A*. Dans leur cas, le risque est une mesure des événements de braquage ou de freinage dangereux sur les routes. Nous n'avons pas retenu cette méthode car nous voulions garder la dépendance au temps du risque. Dans ce cas, nous pouvons chercher la solution optimale dans un réseau étendu dans le temps (*Time Expanded Network*) comme l'a fait [Krumke et al., 2014]. Une connexion entre deux nœuds de ce réseau représente le franchissement d'un arc du réseau statique à un instant donné. Ce type de réseaux est utilisé, entre autres applications, pour des problèmes d'évacuation (voir [Park et al., 2009]).

Mais certaines spécificités imposent de nouveaux défis :

- L'horizon temporel des véhicules autonomes ou semi-autonomes est généralement court et les décisions doivent être prises en ligne, ce qui signifie que les processus décisionnels doivent tenir compte de l'infrastructure de communication (voir [Vivaldini et al., 2013]) et de la façon dont le superviseur global peut être fourni, à tout moment, par une représentation de l'état actuel du système et de son évolution à court terme
- Dès que des véhicules autonomes sont concernés, la sécurité est en jeu (voir [Ryan et al., 2020, Pimenta et al., 2017]). Le superviseur global doit calculer et planifier les itinéraires de manière que non seulement les tâches soient exécutées efficacement, mais aussi que les superviseurs locaux et embarqués effectuent leur travail plus facilement pour fluidifier les flux.

Prendre soin de la sécurité nécessite de quantifier le risque induit par l'introduction dans le système de tout véhicule supplémentaire. Pour résoudre ce problème, il faut transformer les données de trafic collectées en temps réel en estimateurs de risque (voir [Ryan et al., 2020, Zhang et al., 2009]). Dans cette étude, nous proposons des estimateurs qui dérivent du trafic, puis nous nous concentrons sur la manière dont ces estimateurs peuvent être utilisés afin de prendre des décisions de routage et d'ordonnancement rapidement et en toute sécurité. On suppose donc qu'au moment où l'on essaie d'acheminer et d'ordonnancer un véhicule donné, on dispose d'une procédure qui, pour tout arc ou le réseau de transits et à tout moment, peut calculer une estimation grossière du risque lié au passage du véhicule sur l'arc à tout instant. Notre objectif devient alors d'ordonnancer la route que le véhicule va suivre, de telle manière que son heure

d'arrivée soit minimale et que l'estimation du risque pris reste bornée par un certain seuil. Notre problème peut alors être vu comme la recherche d'un plus court chemin contraint (voir [Lozano and Medaglia, 2013]). Mais plusieurs caractéristiques rendent la tâche beaucoup plus difficile :

- Nous devons traiter un réseau dépendant du temps (voir [Franceschetti et al., 2017, Krumke et al., 2014, Park et al., 2009])
- Le contexte en ligne nous empêche de nous appuyer sur une machinerie lourde comme celles liées à la programmation linéaire (CPLEX, Gurobi, etc.).
- L'objectif principal de cette étude est de router un VA, mais nous abordons également le problème du dimensionnement de la flotte.

Le chapitre est donc organisé comme suit : dans la section 3, nous décrivons formellement notre modèle et énonçons quelques résultats structurels, notamment sur la notion d'équilibre. Dans la section 4, nous détaillons le problème ainsi que quelques résultats structuraux nous permettant de reformuler le problème. Dans la section 5, nous décrivons la notion de décision et la structure globale de deux heuristiques : l'une basée sur de la programmation dynamique, l'autre sur l'algorithme A*. Dans la section 6, nous montrons comment bien choisir les décisions et comment en profiter pour accélérer les heuristiques. Dans la section 7, nous expliquons comment les techniques d'apprentissage par renforcement peuvent être utilisées pour transformer un algorithme glouton en heuristiques plus efficaces. Nous consacrons la section 8 aux expériences numériques et à la comparaison des différentes méthodes. Nous terminons par une conclusion à la section 9.

Section 3

Problèmes liés au risque induit par l'activité d'une flotte de VA

Nous nous référons ici à une flotte de véhicules autonomes, qui évolue au fil du temps à l'intérieur d'une sorte d'infrastructure industrielle, par exemple un entrepôt, dans le but de réaliser des tâches de logistique interne (stockage et récupération d'articles, maintenance, inventaire...). Ces tâches doivent être effectuées de manière sûre à petits coûts (temps, énergie...). Il s'ensuit que, pour définir le modèle du *Safe Shortest Path Problem* (SSPP), il faut d'abord formaliser la notion de risque.

3.1 fonction de risque et risque induit

Nous supposons que notre flotte de véhicules se déplace à l'intérieur d'un réseau de transit simple planaire $G = (N, A)$, N désignant l'ensemble des nœuds et A l'ensemble des arcs. Ce réseau G est susceptible de représenter par exemple un entrepôt (voir Figure 3.1), ou tout type de zone restreinte, industrielle ou rurale similaire. À tout arc $e = (u, v)$ correspond une longueur L_e et une vitesse maximale v_{max_e} : un véhicule autonome parcourant e n'est pas autorisé à aller plus vite que v_{max_e} en se déplaçant le long de e.

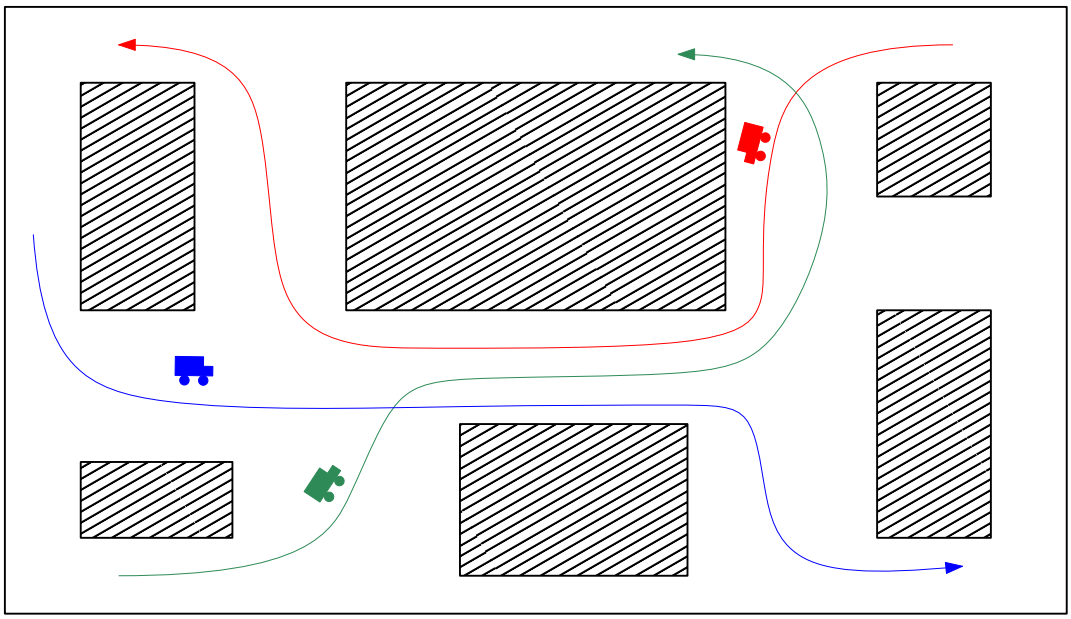

Figure 3.1 – Un réseau de transport similaire à un entrepôt

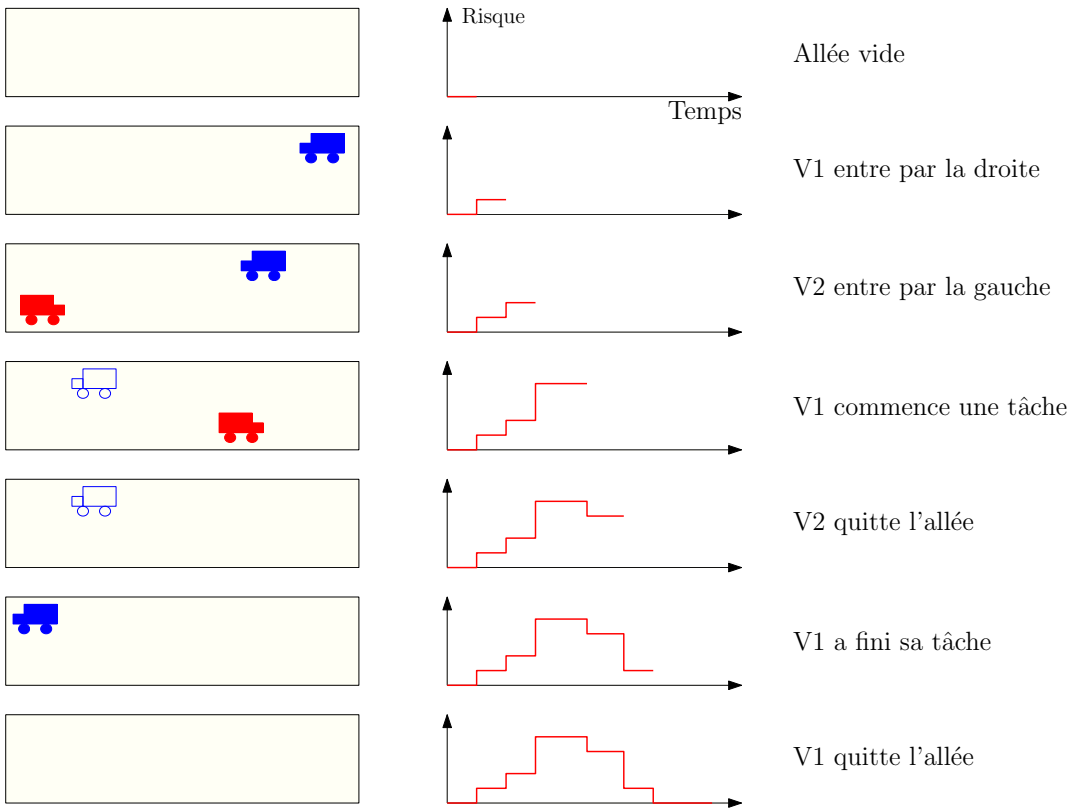

FIGURE 3.2 – Construction de la fonction de risque en fonction des différentes configurations d'une allée.

Une flotte de véhicules autonomes réalise des tâches dans ce réseau. Le superviseur global de cette flotte dispose d'une certaine connaissance des itinéraires suivis par tous les véhicules, de leur horaire et des tâches qu'ils vont accomplir. Cette connaissance lui permet de dériver une fonction d'estimation de risque $\Pi^e(t)$. Il est important de noter que cette fonction $\Pi^e(t)$ n'est pas continue. En effet, il existe, dans une allée, un nombre fini de configurations possibles : vide, deux véhicules en sens inverse, etc. (voir Figure 3.2). À chaque configuration peut être associé un coût estimé des réparations en cas d'accident définit comme tel : pour toute petite valeur dt, $\Pi^e(t).dt$ est une estimation de l'espérance du coût des dommages dans le cas où un nouveau véhicule se déplace à la vitesse maximale v_{max_e} le long de e entre le temps t et le temps $t + dt$. Il s'agit donc de fonctions en escalier évaluées dans une devise (euro, dollars, etc.). Un nouveau véhicule autonome voulant traverser les arcs de ce réseau doit prendre des risques dépendant de l'état des allées sur son trajet. Plus précisément, si le véhicule V se déplace sur l'arc e entre le temps t_1 et le temps t_2, selon la fonction de vitesse $t \mapsto v(t)$, alors le risque pris par ce véhicule est l'**espérance du coût des dommages** entre t_1 et t_2 et est donnée par l'équation 3.1.

$$\int_{t_1}^{t_2} H\left(\frac{v(t)}{v_{max_e}}\right).\Pi^e(t).dt \tag{3.1}$$

Afin d'exprimer le fait qu'une diminution de la vitesse implique une diminution du risque, nous imposons que la fonction H soit telle que $H(v) \ll \frac{v}{v_{max}}$. Dans les sections suivantes, H est défini par $H : v \mapsto \left(\frac{v}{v_{max}}\right)^2$.

Remarque 1. ***Normalisation de la vitesse*** *: On ne s'intéresse ici qu'aux temps de parcours des arcs $e \in A$, et non à leur vraie longueur, au sens géométrique. Nous supposons à partir d'ici, sans perte de généralité, que pour tout arc e, $v_{max_e} = 1$. Il s'agit donc de valeurs de vitesse réduites $v \in [0, 1]$ et L_e désigne le temps de parcours minimal pour l'arc e.*

3.2 Discussion sur les fonctions de risque

Nous allons discuter ici de l'origine des fonctions Π et de la manière dont elles peuvent être dérivées d'observations dans un exemple réel. En fait, la question clé est de savoir comment définir le risque lié à l'activité d'un parc de véhicules autonomes à l'intérieur d'un réseau de transit $G = (N, A)$.

Dans tous les cas, le risque doit être défini ici comme étant lié à l'espérance du coût des dommages qui est induit, sur un arc donné e et pendant un intervalle de temps $[t, t + dt]$, par l'activité de la flotte le long de cet arc. Il s'ensuit que l'enjeu est de déterminer les fonctions de risque $t \mapsto \Pi^e(t), e \in A$. Il est important de noter que l'activité sur les arcs peut impliquer non seulement des véhicules qui se déplacent le long de $e = (i, j)$ mais aussi ceux qui effectuent une activité de stockage ou de récupération, ainsi que les véhicules qui se déplacent en sens inverse $e^{-1} = (j, i)$.

3.2.1 Le cas le plus simple

Ici, le risque est induit par des véhicules identiques en mouvement, qui suivent les arcs de G. Alors, la fonction de risque Π^e devrait prendre la forme $\Pi^e(t) = \Pi^e_n(v_1(t), \ldots, v_n(t))$, où n est le nombre de véhicules qui se déplacent le long de l'arc e au temps t et $v_1, \ldots, v_n$ leur vitesse respective. Puisque nous supposons que tous les véhicules sont identiques, la fonction Π^e_n devrait être symétrique. Pour exprimer l'impact de l'introduction d'un véhicule supplémentaire à une vitesse donnée v, nous pouvons écrire :

$$\Pi^e_{n+1}(v_1, \ldots, v_n, v) = H(v).Q^e(v_1, \ldots, v_n) + \Pi^e_n(v_1, \ldots, v_n)$$

avec $H(v) << v \leq 1$ et $Q^e(v_1, \ldots, v_n)$ symétriques. Cette expression signifie que l'on distingue les dommages qui impliquent explicitement un véhicule supplémentaire $n + 1$ des dommages qui impliquent uniquement d'autres véhicules. Autrement dit, $\Pi^e_n(v_1, \ldots, v_n)$ ne tient compte que des accidents pouvant survenir si le véhicule $n + 1$ n'est pas là et $Q^e(v_1, \ldots, v_n)$ est le surplus de risque induit par l'ajout, dans l'arc e et à la vitesse $v_{max} = 1$, d'un véhicule supplémentaire. Étant donné que ces accidents sont susceptibles d'impliquer non seulement le véhicule supplémentaire, mais également certains véhicules qui circulent déjà à l'intérieur de e, nous devrions également avoir :

$$\sum_{i=1}^{n+1} H(v_i).Q^e(v_1, \ldots, v_i, \ldots, v_n) \geq \Pi^e_{n+1}(v_1, \ldots, v_n, v_{n+1})$$

Les formules ci-dessus peuvent être mises en œuvre de la façon suivante :

$$\Pi^e_n(v_1, \ldots, v_n) = (\prod_{i=1}^{n+1} H(v_i)).Q^e_{uni}(n + 1)$$

où Q^e_{uni} est une fonction croissante de n telle que $Q^e_{uni}(n + 1) - Q^e_{uni}(n)$ est également croissante.

3.2.2 Le cas des véhicules immobiles

De la même façon, en distinguant les dommages impliquant les véhicules déjà présents, les dommages entre le nouveau véhicule et ceux circulant et les dommages entre le nouveau véhicule et ceux immobiles, on obtient :

$$\Pi^e_{n+1,p}(v_1, \ldots, v_{n+1}) = (\prod_{i=1}^{n+1} H(v_i)).Q^e_{imm}(n + 1, p)$$

où n représente le nombre de véhicules qui se déplacent déjà le long de e (auxquels un véhicule est ajouté) et p le nombre de véhicules impliqués dans une activité de stockage/récupération et sont donc immobiles. La fonction Q^e_{imm} doit être croissante en n et p et telle que :

- $Q^e_{imm}(n + 1, p) - Q^e_{imm}(n, p)$ est croissante en n
- $Q^e_{imm}(0, p) = 0$
- $Q^e_{imm}(n, 0) = Q^e_{uni}(n)$

3.2.3 Le cas des véhicules circulant dans les deux sens sans véhicule immobile

On procède de la même façon pour arriver à la formule :

$$\Pi^e_{n+1,q}(v_1,\dots,v_{n+1},v'_1,\dots,v'_q) = (\prod_{i=1}^{n+1} H(v_i)).(\prod_{j=1}^{q} H(u_i)).Q^e_{bi}(n,q)$$

où n désigne le nombre de véhicules qui se déplacent déjà le long de e (auxquels un véhicule est ajouté) et q le nombre de véhicules qui se déplacent sur e^{-1}. La fonction Q^e_{bi} doit être croissante en n et q et telle que :

- $Q^e_{bi}(n+1,q) - Q^e_{bi}(n,q)$ est croissante en n
- $Q^e_{bi}(n,q+1) - Q^e_{bi}(n,q)$ est croissante en q
- $Q^e_{bi}(0,q) = Qe^{-1}{}_{uni}(n)$
- $Q^e_{bi}(n,0) = Q^e_{uni}(n)$

3.2.4 Le cas général

On procède de la même façon pour arriver à la formule :

$$\Pi^e_{n+1,p,q}(v_1,\dots,v_{n+1},v'_1,\dots,v'_q) = (\prod_{i=1}^{n+1} H(v_i)).(\prod_{j=1}^{q} H(u_i)).Q^e_{gen}(n,p,q)$$

où n désigne le nombre de véhicules qui se déplacent déjà le long de e (auxquels un véhicule est ajouté), p le nombre de véhicules impliqués dans une activité de stockage/récupération et q le nombre de véhicules qui se déplacent sur e^{-1}. La fonction Q^e_{gen} doit être croissante en n, p et q et telle que :

- $Q^e_{gen}(n+1,p,q) - Q^e_{gen}(n,p,q)$ est croissante en n
- $Q^e_{gen}(n,p+1,q) - Q^e_{gen}(n,p,q)$ est croissante en p
- $Q^e_{gen}(n,p,q+1) - Q^e_{gen}(n,p,q)$ est croissante en q
- $Q^e_{gen}(0,p,q) = Q^{e^{-1}}_{imm}(q,p)$
- $Q^e_{gen}(n,0,q) = Q^e_{bi}(n,q)$
- $Q^e_{gen}(n,p,0) = Q^e_{imm}(n,p)$
- $Q^e_{gen}(n,0,0) = Q^e_{uni}(n)$
- $Q^e_{gen}(0,p,0) = 0$
- $Q^e_{gen}(0,0,q) = Q^{e^{-1}}_{uni}(q)$

3.3 Le problème de dimensionnement du SSPP

Ces fonctions de risques serviront à calculer le risque pris par un nouveau VA sur son chemin, mais également à l'évaluation du risque globalement pris par la flotte de VAs. En effet, une stratégie de routage optimal pour un certain VA peut entraîner une très forte augmentation du risque pour les VAs déjà routés et partageant une partie de son trajet. Il apparaît donc une dualité entre les stratégies individuelles et les stratégies collectives. Il vient alors la question du dimensionnement : à quel moment, ajouter un VA de plus à la flotte desservirait l'objectif principal, à savoir effectuer des tâches. Autrement dit, dans un horizon de temps T_{max} fixé, si la flotte est trop petite, seules quelques tâches pourront être réalisées. Mais aussi, si la flotte est trop grande, l'entrepôt deviendra trop risqué pour pouvoir circuler normalement et le nombre de tâches réalisable va alors décroitre dû à la congestion grandissante et l'augmentation du risque.

La question est donc de savoir à partir de quel nombre de VAs, le risque engendré par la flotte fait décroitre le nombre de tâches réalisable dans l'horizon de temps. Ou bien, en retournant le problème, pour un nombre de tâches fixé à effectuer dans l'horizon de temps T_{max}, à partir de combien de VAs la flotte n'est plus capable de garder un risque global en dessous d'un seuil de risque R_{max}. Cette version du problème nous impose un certain nombre de points de passage qu'il faut traverser grâce à une collection de trajets $\{(\Gamma_k, v_k), k = 1,\dots,n\}$:

- une route Γ_k permettant d'effectuer une ou plusieurs tâches
- une fonction vitesse $v_k : t \mapsto v_k(t)$ à appliquer sur Γ_k

L'objectif de ce problème est donc de traiter toutes les tâches et de minimiser le risque global dans l'entrepôt. Les problèmes de dimensionnements, dans un cadre général, sont très divers et sont étudiés depuis longtemps (voir [Peiron, 1972]) mais l'ajout des fonctions de risque et de vitesse rend notre problème plus compliqué et peu commun et il se formalise ainsi :

Problème de minimisation du risque global :
Calculer une collection de trajets $\{(\Gamma_k, v_k), k = 1, \ldots, n\}$ *telle que la valeur de risque globale résultante* $\sum_e \int_{[0,T_{max}]} H(v(t))\Pi^e(t)dt$ *soit le plus petit possible.*

La solution optimale de ce problème, pour une taille de flotte fixée, génère un certain risque dans l'entrepôt. Mais, en fonction de la taille de flotte utilisée, il est possible d'obtenir d'autres solutions. Certaines solutions généreront plus de risque que d'autres, en particulier pour de très grandes flottes de VAs. Les flottes trop petites, quant à elles, ne permettront pas d'effectuer toutes les tâches dans l'horizon de temps.

Parmi toutes ces tailles de flotte, celle dont la solution permet la réalisation de toutes les tâches et engendre le moins de risque, a ainsi une taille optimale au regard des objectifs et de l'architecture du réseau utilisé.

3.4 L'équilibre de Wardrop du SSPP

Un équilibre bien connu des réseaux de transport soumis à de fortes congestions est la notion d'équilibre de Wardrop. Plus précisément, si un ensemble de solutions individuelles forment un équilibre de Wardrop, alors aucun VA ne peut changer sa stratégie unilatéralement sans perte. Si un tel équilibre existe, alors toutes les solutions individuelles ont le même coût généralisé (temps et risque) et celui-ci est plus faible que le coût généralisé de tout autres stratégies. Cette notion est légèrement différente de ce que nous avons présenté jusqu'alors puisque que les gains ou pertes de tout changement de stratégie sont individuels (par exemple, si le véhicule k change de stratégie, il subit une augmentation du risque qu'il prend à titre individuel).

Nous avons choisi d'étudier cette notion dans une variante collaborative dans le sens où les conséquences d'un changement de stratégie sont globales à la flotte. Supposons que nous ayons affaire à une mesure du risque qui dérive du cas simple monodirectionnel tel que défini dans la section précédente, et que nous imposions un horizon temporel $[0, T_{max}]$. Nous considérons une flotte de véhicules, avec n véhicules $k = 1, \ldots, n$, dont l'activité est planifiée entre 0 et T_{max}, selon des stratégies de routage $(\Gamma_k, v_k), k = 1, \ldots, n$, où Γ_k est une route permettant d'effectuer une ou plusieurs tâches. Les stratégies de routage $(\Gamma_k, v_k), k = 1, \ldots, n$, permettent de dériver, pour tout arc e, deux catégories de fonctions de risque $t \mapsto \Pi^e_k(t)$ et $t \mapsto \bar{\Pi}^e_k(t)$ définies comme :

- Une collection de fonctions de risque auxiliaires $t \mapsto \bar{\Pi}^e_k(t)$ dont la signification est : $\bar{\Pi}^e_k(t)$ est la fonction de risque, que nous obtenons en supprimant le véhicule k et la stratégie de routage associée (Γ_k, v_k)
- Une collection de fonctions de risque marginal $t \mapsto \Pi^e_k(t)$, obtenue à partir de l'équation 3.2.1 et dont la signification est : si l'on considère tous les véhicules sauf k, et si l'on fait rouler le véhicule k le long de e entre t et $t + dt$ à la vitesse, $v \leq 1$, alors le dommage supplémentaire attendu causé directement ou non par k est égal à $H(v).\bar{\Pi}^e_k(t).dt$

Alors, nous pouvons étendre la notion d'équilibre de Wardrop en définissant un équilibre de risque de Wardrop comme suit :

Une collection de stratégies $\{(\Gamma_k, v_k), k = 1, \ldots, n\}$ définit un équilibre de risque de Wardrop si, pour tout $k = 1, \ldots, n$: (Γ_k, v_k) est une stratégie optimale pour le véhicule k au sens de l'instance du SSPP

impliquant des fonctions de risque marginal $t \mapsto \Pi_k^e(t)$ et un horizon temporel $[0, T_{max}]$.

Ensuite, nous énonçons :

Proposition 1. *Si $T_{max} \geq sup_k\{L^*_{o_k,d_k}\}$, où L^* signifie la distance du chemin le plus court dans le sens de la longueur L, alors il existe un équilibre de risque de Wardrop.* ▼

Preuve 1. Il découle directement de la façon dont nous avons défini $t \mapsto \Pi_k^e(t)$ comme étant lié aux dommages marginaux attendus induits par l'introduction d'un véhicule supplémentaire k dans un réseau de transit où d'autres véhicules $1, \ldots, k-1, k+1, \ldots, n$, sont supposés déjà évoluer. Plus précisément, la fonction $t \mapsto \Pi_k^e(t)$ est définie pour tout k par l'équation 3.2.1 qui la relie aux fonctions de risque $t \mapsto \bar{\Pi}_k^e(t)$.

Considérons alors une solution optimale $\{(\Gamma_k^{opt}, u_k^{opt}), k = 1, \ldots, n\}$ du problème de minimisation du risque global. Notons $t \mapsto \bar{\Pi}_k^{opt^e}(t)$ et $t \mapsto \Pi_k^{opt^e}(t), k = 1, \ldots, n$, fonctions de risque auxiliaires et fonctions de risque marginal associées à cette solution optimale. Ensuite, nous voyons que, pour tout k, $(\Gamma_k^{opt}, u_k^{opt})$, qui respecte l'horizon temporel $[0, T_{max}]$, se minimise sous la contrainte :

$$\sum_e \int_{[0,T_{max}]} \Pi^e(t)dt - \sum_e \int_{[0,T_{max}]} \bar{\Pi}_k^{opt^e}(t)dt = \sum_e \int_{[0,T_{max}]} H(v(t)).\Pi_k^{opt^e}(t)dt$$

et est donc une solution optimale de l'instance du SSPP induite par les fonctions de risque marginal $t \mapsto \Pi_k^{opt^e}(t)$. Si $T_{max} \geq sup_k\{L^*_{o_k,d_k}\}$ alors le problème de la minimisation globale du risque admet clairement une solution réalisable. Il reste à prouver qu'il admet une solution optimale. Nous remarquons que pas plus de n véhicules peuvent être localisés à un instant donné t sur un même arc e, et qu'une fois qu'un véhicule quitte e, il ne revient pas plus tard. Il vient que le nombre de points de rupture d'une fonction $\Pi_k^e(t)$ ne peut excéder n. Alors un argument topologique simple sur la compacité nous permet de vérifier qu'à partir de n'importe quelle suite $\{(\Gamma_k^p, u_k^p), k = 1, \ldots, n, p = 1, \cdots + \infty\}$, on peut extraire une sous-suite convergente au sens simple, et que le théorème de Lebesgue peut être appliqué à cette convergence. Nous en déduisons qu'il doit exister une collection $\{(\Gamma_k^{opt}, u_k^{opt}), k = 1, \ldots, n\}$ qui réalise $Inf_{(\Gamma_k, v_k), k=1,\ldots,n}(\sum_e \int_{[0,T_{max}]} \Pi^e(t)dt)$. ■

Section 4

Le problème du plus court chemin sécurisé

Un véhicule autonome, jusqu'alors inactif, doit maintenant effectuer une nouvelle tâche à l'intérieur de l'entrepôt. Puisqu'il est arrêté au milieu de l'entrepôt suite à sa dernière tâche, il est un danger pour le reste de la flotte. Il faut donc le router vers sa nouvelle destination sans délai. Notre objectif est alors de trouver une décision opérationnelle pour résoudre notre problème du plus court chemin sécurisé en temps réel.

4.1 Objectifs du problème

Pour effectuer cette tâche, il doit passer d'un nœud origine o à un nœud destination p. Il faut déterminer sa trajectoire et ses fonctions de vitesse dans chaque allée de sa trajectoire **tout en connaissant** :

- La structure de l'entrepôt : $G = (N, A)$ un graphe connexe planaire avec N l'ensemble des carrefours et e l'ensemble des arcs (i.e. allées)
- Le temps de parcours minimum L_e de chaque arc e
- La fonction de risque $\Pi^e : t \mapsto Pi^e(t)$ de chaque arc e dérivé de l'activité courante de la flotte de VA
- Le nœud d'origine o et le nœud de destination p.

Alors, **nous voulons obtenir** :

- le chemin Γ de o à p qui sera suivi par le véhicule, ainsi que la date d'entrée t_e de chaque arc e de Γ.
- les fonctions de vitesse $v : t \in [t_e; t_{e+1}] \mapsto v_e(t)$ à appliquer lorsque le véhicule se trouve à l'intérieur de chaque arc e de Γ.

En l'état, nous aurions pu travailler avec un problème multiobjectifs. Cependant, nous voulons calculer une trajectoire et une vitesse telles que le véhicule autonome soit "sûr". Cela signifie qu'une contrainte de valeur de risque maximale est ajoutée. Le responsable d'entrepôt imposera une valeur maximale de risque R_{max} (quantifié en devise, il peut correspondre au coût de remplacement d'un véhicule en cas d'accident) que peut prendre un véhicule autonome pour une tâche. L'objectif est alors de déterminer la décision suivante en temps réel :

SSPP
Calculer le chemin Γ avec les dates d'entrée t_e, de sortie t_{e+1} et les fonctions de vitesse v_e de chaque arc e du chemin tel que :

- *La date d'arrivée en p est minimale.*
- *Le risque global $\sum_{a \in \Gamma} risk\,(t_e, t_{e+1}, v_a) \leq R_{max}$.*

4.2 Quelques résultats structuraux

Tel quel, SSPP ressemble plus à un problème de contrôle optimal (voir [Malikopoulos et al., 2019]) qu'à un problème combinatoire. Mais, comme nous allons le montrer maintenant, nous pouvons imposer des contraintes sur la fonction de vitesse v, qui vont rapprocher le modèle SSPP d'un modèle de décision

discrète. La première proposition (Proposition 2) restreint le type de formes que peut prendre la fonction vitesse :

Proposition 2. *La solution optimale* (Γ, v) *de SSPP peut être choisie de telle sorte que* v *soit constant par morceaux, avec les mêmes points de rupture que la fonction de risque des arcs traversés augmentés des dates de transition entre deux arcs.* ▼

Preuve 2. Supposons que V se déplace le long d'un arc $e = e_i$, et que τ_1, τ_2 soient 2 points de rupture consécutifs dans le sens ci-dessus. Si $v(t)$ n'est pas constant entre τ_1 et τ_2 alors, nous pouvons remplacer $v(t)$ par la valeur moyenne v^* de la fonction $t \mapsto v(t)$ entre τ_1 et τ_2. La valeur de temps $Time(\Gamma, v)$ reste inchangée, tandis que la valeur de risque $Risk(\Gamma, v)$ diminue en raison de la convexité de la fonction H. Nous concluons. ■

Une conséquence est que le risque que prend un VA dans une allée être calculé par morceaux puisque sa vitesse est elle-même par morceau :
Soit t_i et t_{i+1} deux points de rupture consécutifs de la fonction de risque de l'arc courant e. La fonction de vitesse $t \mapsto v(t)$ et le risque $t \mapsto \Pi_e(t)$ sont constants et égale à v et Π_i^e respectivement. Ainsi, le risque pris par V prend la forme de l'équation 4.1.

$$\begin{aligned} risk(t_i, t_{i+1}, v(t)) &= \int_{t_i}^{t_{i+1}} H(v(t))\Pi_e(t).dt \\ &= H(v).(t_{i+1} - t_i).\Pi_i^e \end{aligned} \tag{4.1}$$

La deuxième proposition (Proposition 3) nous permet de connaître la valeur du risque pris dans la solution optimale sous condition sur la fonction vitesse :

Proposition 3. *Si la trajectoire optimale SSPP* (Γ, v) *est telle que* $v(t) \neq 1$ *à un certain* t*, alors* $Risk(\Gamma, v) = R_{max}$. ▼

Preuve 3. Supposons que le chemin Γ soit une séquence $e_1, \ldots, e_n$ d'arcs de G. On procède par induction sur n.

- Premier cas : $n = 1$.
  Supposons que l'assertion ci-dessus soit fausse. Les points de rupture de $e = e_1$, peuvent s'écrire $t_0 = 0, t_1, \ldots, t_Q = Time(\Gamma, v)$, et on peut poser :
  - $q_0 =$ *plus grand* q tel que $v < 1$ entre t_q et t_{q+1} ;
  - $u_0 =$ *vitesse correspondante* ; $l_0 =$ *distance parcourue* par V au temps t_{q_0}.

  Augmentons u_0 de $\epsilon > 0$, tel que $u_0 + \epsilon \leq 1$ et que le surcroît de risque induit pris entre t_{q_0} et t_{q_0+1} ne dépasse pas $R_{max} - Risque(\Gamma, v)$. Puis, au temps t_{q_0+1}, le véhicule V a parcouru une distance $l > l_0$. Si $l < L_e$, alors il poursuit sa route à la vitesse $v = 1$, et arrive donc à la fin de e avant l'instant t_Q, sans avoir dépassé le seuil de risque R_{max}. Nous concluons.
- Deuxième cas : $n > 1$.
  Supposons que l'assertion ci-dessus soit fausse et notons par R_1 le risque pris en fin d'arc e, et par t_1 la valeur temps associée. L'induction appliquée aux arcs $e_2, \ldots, e_n$, et le seuil de risque $R_{max} - R_1$ implique que la vitesse de V est égale à 1 tout le long des arcs $e_2, \ldots, e_n$. Notons $\tau_0 = 0, \tau_1, \ldots, \tau_Q$ les points de rupture de e_1 qui sont compris entre 0 et t_1 et posons $\tau_{Q+1} = t_1$ et :
  - $q_0 =$ *plus grand* q tel que $v < 1$ entre τ_q et τ_{q+1} ;
  - $u_0 =$ *vitesse correspondante* ; $l_0 =$ *distance parcourue* par V au temps t_{q_0+1}.

  Puis on augmente u_0 de $\epsilon > 0$, tel que $u_0 + \epsilon \leq 1$ et que le risque supplémentaire induit entre τ_{q_0} et τ_{q_0+1} ne dépasse pas $(R_{max} - \frac{Risque(\Gamma,v))}{2}$. En se déplaçant à la vitesse $u_0 + \epsilon$ le long de e_1, le véhicule V fait face à deux possibilités : soit il arrive à la fin de e_1 avant le temps τ_{q_0+1} soit il continue de se déplacer depuis le temps τ_{q_0+1} le long de e_1 à la vitesse $v = 1$. Dans tous les cas, il atteint la fin de e_1 à un moment donné $t_1 - \beta, \beta < 0$, avec un risque supplémentaire ne dépassant

pas $(R_{max} - \frac{Risk(\Gamma,v))}{2}$. Ainsi, pour tout $i = 2,\dots,n$ nous calculons la valeur de vitesse u_i telle que se déplacer le long de e_i à la vitesse u_i entre $t_{i-1} - \beta$ et t_{i-1} n'induit pas de risque supplémentaire supérieur à $(R_{max} - \frac{Risk(\Gamma,v))}{2n}$. On applique donc à V la stratégie suivante : se déplacer comme décrit ci-dessus sur l'arc e_1 et ensuite, pour tout $i = 2,\dots,n$, se déplacer le long de e_i à la vitesse u_i entre $t_{i-1} - \beta$ et t_{i-1} puis à la vitesse 1 jusqu'à la fin de e_i. Le surcroît de risque induit par cette stratégie ne peut excéder $(R_{max} - Risque(\Gamma, v))$. D'un autre côté, cette stratégie fait que le véhicule V réalise son parcours strictement avant l'instant t_n. Nous concluons.

■

La dernière proposition (Proposition 4) montre l'indépendance au temps de la dérivée du risque pris par V :

Proposition 4. *Étant donné une trajectoire SSPP optimale (Γ, v), avec $\Gamma = \{e_1,\dots,e_n\}$ et v satisfaisant la proposition 2. Notons par t_i l'heure d'arrivée en fin d'arc e_i. Alors, pour tout $i = 1,\dots,n$, et tout t dans $[t_{i-1}, t_i]$ tel que $v = v(t) < 1$, la quantité $H'(v(t)).\Pi^{e_i}(t)$ est indépendante de t, où $H'(v)$ désigne la dérivée de H en v.* ▼

Preuve 4. Encore une fois, notons par t_i l'heure d'arrivée à la fin de l'arc e_i. Pour un i donné, on dénote par $\tau_1,\dots,\tau_{\bar{i}}$ les points de rupture de la fonction Π_{e_i} qui sont dans l'intervalle $]t_{i-1}, t_i[$, par $\Pi^j_{e_i}$ les valeurs de Π_{e_i} sur l'intervalle $]\tau_j, \tau_{j+1}[$, par $v_0,\dots,v_j,\dots,v_{\bar{i}}$, les valeurs de vitesse du véhicule lorsqu'il quitte ces points de rupture, et par R_i le risque globalement pris par le véhicule lorsqu'il se déplace tout le long de e_i. Grâce à la proposition 3, le vecteur $(u_0,\dots,u_{\bar{q}})$ est une solution optimale du problème d'optimisation convexe suivant : calculer $(v_0,\dots,v_{\bar{i}})$ tel que $\sum_j v_j.(\tau_{j+1} - \tau_j)$ et qui minimise $\sum_j H(v_j)\Pi^j_{e_i}(\tau_{j+1} - \tau_j)$.

Alors, les conditions de Kuhn-Tucker pour l'optimalité du programme d'optimisation convexe différentiable précise qu'il existe $\lambda \geq 0$ tel que : pour tout j tel que $v_j < 1, H'(v_j).\Pi^j_{e_i} = \lambda$. Ainsi, λ ne peut pas être égal à 0. Nous concluons. ■

Remarque 2. *Dans le cas $H(v) = v^2$, l'égalité $H'(v_q)\Pi^{e_i}_q = \lambda$ devient $v_q\Pi^{e_i}_q = \frac{\lambda}{2}$ où $v_q\Pi^{e_i}_q$ est le risque instantané par unité de distance $\frac{dR}{dL}$.*

4.3 Une reformulation *Risque Sur Distance* du modèle SSPP

La remarque 2 nous amène à définir le coefficient de risque par unité de distance pour l'arc e comme étant la valeur $H'(v_q)\Pi^q_e$ impliquée dans la proposition 4. Nous appelons ce coefficient λ^{RD}_e. Cette proposition, combinée à la Proposition 2, nous permet de simplifier significativement le SSPP : nous définissons une *stratégie risque versus distance* comme un couple (Γ, λ^{RD}) où :

- Γ est un chemin, c'est-à-dire une séquence $\{e_1,\dots,e_n\}$ d'arcs, qui relie le nœud d'origine o au nœud de destination d ;
- $\lambda^{RD} = \{\lambda^{RD}_e, e \in \Gamma\}$ est un vecteur de coefficients de risque sur distance associée à tout arc e dans Γ. Ces valeurs permettent de retrouver la vitesse à appliquer sur chaque arc, car $\lambda^{RD}_e = H'(v(t))\Pi_e(t)$ à tout instant t.

Supposons que l'on suive une trajectoire (Γ, v) qui vérifie la proposition 4, et que l'on connaisse la valeur λ^{RD}_e pour tout arc e de Γ.Puisque H est supposé convexe et tel que $H(v) \ll v$, on peut dire que H' admet une fonction réciproque H'^{-1}. Ensuite, à tout instant t, lorsque le véhicule V se trouve à l'intérieur de l'arc e, nous sommes capables de reconstruire la valeur v(t) par l'équation 4.2.

$$v(t) : \begin{cases} H'^{-1}(\frac{\lambda^{RD}_e}{\Pi_e}), & \text{si } H'^{-1}(\frac{\lambda^{RD}_e}{\Pi_e}) < 1 \\ 1, & \text{sinon} \end{cases} \tag{4.2}$$

D'après ceci et la proposition 4, SSPP peut être réécrit comme suit (nous étendons les notations $Risk(\Gamma, v)$ et $Time(\Gamma, v)$ en $Risk(\Gamma, \lambda^{RD})$ et $Time(\Gamma, \lambda^{RD})$:

Reformulation Risque Sur Distance :
Calculer une stratégie Risque Sur Distance (Γ, λ^{RD}) *de sorte que* $Risk(\Gamma, \lambda^{RD}) \leq R_{max}$ *et* $Time(\Gamma, \lambda^{RD})$ *soit le plus petit possible.*

4.4 Discussion sur la complexité

La dépendance au temps du réseau de transit ainsi que la proximité du modèle SSPP avec les modèles Shortest Path Constraint suggèrent que SSPP est un problème complexe. Cependant, identifier la complexité de SSPP n'est pas si simple, puisque nous avons affaire à des variables continues. La complexité dépend aussi de la fonction H, et nous supposons donc ici que $H(v) = v^2$. On peut d'abord vérifier que :

Proposition 5. *SSPP est dans NP.* ▼

Preuve 5. Il est suffisant de traiter le cas où $\Gamma = \{e_1, \ldots, e_n\}$ est fixé. Notons par $\Delta = \{0, d_1, \ldots, d_H\}$ l'ensemble de tous les points de rupture liés aux fonctions $\Pi^e, e \in \Gamma$, et par t_i l'instant auquel le véhicule V arrivera à la fin de e_i (on fixe $t_0 = 0$). Si nous connaissons les valeurs $\{t_1, \ldots, t_n\}$, alors nous pouvons récupérer les valeurs $\lambda_e^{RD}, e \in \Gamma$, par recherche dichotomique. Il s'ensuit que le cœur de notre problème concerne le calcul des valeurs $\{t_1, \ldots, t_n\}$. Pour tout i, la valeur t_i peut être égale à une certaine valeur d_H ou située à l'intérieur d'un intervalle $]d_H, d_{H+1}[$. Afin de faire la distinction entre ces 2 configurations, nous introduisons la fonction suivante σ, qui va caractériser ce positionnement logique des valeurs t_i par rapport à Δ :

- Si $t_i = d_H$ alors on pose $\sigma(i) = (h, 0)$;
- Si $t_i \in]d_H, d_H + 1[$ alors on pose $\sigma(i) = (h, 1)$.

Le nombre de fonctions possibles σ est borné par C_{H^n}. Maintenant, nous remarquons qu'une fois la fonction σ fixée, le problème consiste à calculer les valeurs de celles parmi les variables $\{t_1, \ldots, t_n\}$ qui ne sont pas instanciées, ainsi que les valeurs de vitesse pour tous les intervalles consécutifs définis par les éléments de Δ et par ces valeurs de temps. Ce problème peut être formulé comme un problème d'optimisation cubique, et on peut vérifier que, dans le cas où ce problème a une solution réalisable, alors les équations de Kuhn-Tucker du premier ordre pour l'optimalité locale déterminent exactement un optimum local. Nous concluons. ■

Conjecture 1. *SSPP est NP-Hard.* ▼

Cette conjecture est motivée par le fait que SSPP semble être proche des problèmes de plus court chemin contraints, qui sont le plus souvent NP-Hard [Lozano and Medaglia, 2013]. La difficulté pratique de SSPP peut être capturée à travers l'exemple suivant (Figure 4.1), qui fait apparaître que si (Γ, v) définit une trajectoire SSPP optimale, le risque par valeur de distance $\lambda_e^{RD} = H'(v(t)).\Pi^e$ peut être indépendant de t sur l'arc e comme indiqué dans la proposition 4, mais ne peut pas être considéré comme indépendant de l'arc e.

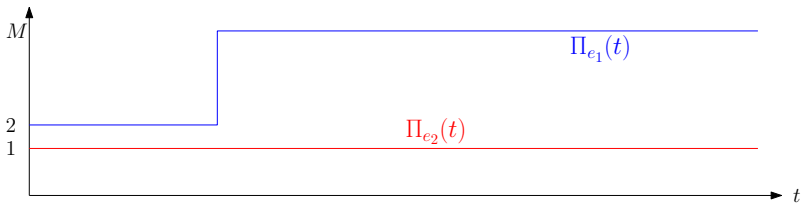

FIGURE 4.1 – Fonctions de risque sur les deux arcs considérés.

Le chemin Γ contient deux arcs, e_1 et e_2, tous deux de longueur 1 et de vitesse maximale 2. La fonction Π^{e_2} est constante et égale à 1. Fonction Π^{e_1} prend la valeur 2 pour $0 \leq t \leq 1$, et une très grande valeur M (par exemple 100) pour $t > 1$ (voir Figure 4.1). $R_{max} = 6$; La fonction H est : $v \mapsto H(v) = v^2$. On voit alors

que le véhicule V doit aller vite tout le long de l'arc e_1, afin de sortir de e_1 avant que cet arc ne devienne très risqué. Cela signifie que sa vitesse est égale à 1 sur e_1, et que sa valeur de risque par unité de distance est égale à 4. Il passe ensuite sur l'arc suivant et freine en termes de risque. C'est-à-dire que sa vitesse reste égale à 1, mais sa valeur de risque par unité de distance diminue à 2. Il est facile de vérifier que cette stratégie de routage est la meilleure, avec $Risk(\Gamma, v) = 6$ et $Time(\Gamma, v) = 2$.

Section 5

Schémas algorithmiques pour la résolution du SSPP

L'objectif est donc de résoudre le problème présenté dans la section précédente à travers le modèle du SSPP. Nous présentons dans cette section deux méthodes de résolution en gardant, comme contrainte principale, l'aspect temps réel nécessaire dans notre contexte. Pour cette raison, nos algorithmes reposent tous sur des notions d'état et de décision qui sont tels que :

Un état est un triplet (i, t, r), où :

- i est un nœud de G où se trouve actuellement le véhicule considéré
- t est le temps passé pour atteindre i
- r est la quantité de risque induite par ce processus de déplacement jusqu'au nœud i.

Une décision est :

- Choisir l'arc $e = (i, j)$ le long duquel le véhicule va se déplacer depuis i
- Choisir une décision λ_e qui va déterminer la fonction de vitesse v le long de l'arc e.

Pour toute la suite de l'étude, nous nous restreignons au cas où $H(v) = v^2$.

5.1 Un algorithme de résolution gloutonne

La reformulation *Risque versus Distance* de la section 4.3 suggère l'utilisation du coefficient λ_e^{RD} comme paramètre de décision λ_e. Un algorithme glouton peut être dérivé de cette reformulation sur le chemin le plus court en termes de distance. C'est-à-dire, pour tous les arcs du plus court chemin en termes de distance, nous supposons que tous les arcs restants sont équivalents et que la valeur de risque par unité de distance sur tous les arcs sont égaux entre eux et donc également à celui du chemin entier. Ainsi, la valeur de risque par unité de distance pour un arc considéré est calculée en fonction de la distance restante à parcourir et du risque encore disponible avant d'atteindre R_{max} (voir algorithme 5.1).

Algorithme 5.1 : Algorithme Glouton RD (Greedy_SSPP)

Entrées : $u_1, \ldots, u_n$ le plus court chemin en termes de distance
Début
  $t_1 = 0$
  $r_1 = 0$
  Pour $i \in \{1, \ldots, n-1\}$ **Faire**
    Parcourir l'arc (u_i, u_{i+1}) avec $\lambda^{RD} = \frac{R_{max} - r_i}{L^*_{u_i,p}}$
      (où $L^*_{u_i,p}$ est la distance du plus court chemin entre u_i et p)
    Mettre à jour t_{i+1} et r_{i+1}.
  Fin Pour
Fin

La décision *Risque versus Distance* possède l'avantage que, à chemin fixé, nous connaissons la moyenne des décisions puisque que nous connaissons le risque pris globalement et la distance totale du chemin parcouru. Mais d'autres types de décision sont possibles. Nous allons en étudier trois différentes au total : *Risque versus Distance*, *Risque versus Temps* et *Distance versus Temps*. Ces trois notions sont indissociables dans l'étude de notre problème et nous permettent d'obtenir des indicateurs que nous allons dériver afin d'obtenir trois types de décisions.

5.2 Schéma de décision

Comme indiqué ci-dessus, une approche naturelle consiste à se référer à la proposition 4 et à considérer $\lambda = \lambda^{RD}$ comme exprimant le coefficient moyen de risque par rapport à la distance à parcourir. Mais une autre approche intuitive est de considérer $\lambda = \lambda^{DT}$ comme exprimant la vitesse moyenne de V le long de e, et d'en déduire de cette manière l'heure d'arrivée sur i_2 de manière simple. Enfin, nous pouvons également considérer que $\lambda = \lambda^{RT}$ exprime la *vitesse de risque* moyenne de V le long de e, ce qui signifie la quantité de risque que V prend par unité de temps au fur et à mesure qu'il avance sur e. Nous allons décrire ici ces trois possibilités (*Risque versus Distance* λ^{RD}, *Risque versus Temps* λ^{RT} et *Distance versus Temps* λ^{DT}), ainsi que la manière dont l'état résultant (i_2, t_2, r_2) peut être déduit de la décision λ et de (i_1, t_1, r_1).

- **Première approche :** *L'approche Risque versus Distance.*

Puisque $H(v) = v^2$, $H'(v(t))\Pi^e(t) = 2u(t)\Pi^e(t)$ pour tout t pendant le parcours de e. Il vient que si λ^{RD} est fixé, la valeur de vitesse $v(t)$ est donnée par : $v(t) = Inf\left(1, \frac{\lambda^{RD}}{\Pi^e(t)}\right)$. L'état résultant (i_2, t_2, r_2) sera obtenu à partir de λ^{RD} et (i_1, t_1, r_1) via le processus itératif suivant : tant que l'arc n'est pas parcouru en entier, calculer la vitesse pour chaque plateau de risque par l'équation ci-dessus ainsi que la distance parcourue et le risque pris avant le prochain plateau (voir Algorithme 5.2).

Algorithme 5.2 : Procédure de transition Risque_Distance

Entrées : L_e la longueur de l'arc e
L'état actuel du véhicule (i_1, t_1, r_1)
$\pi_0, \ldots, \pi_Q$ les dates des points de rupture de Π^e qui sont plus grandes que t_1 et $\Pi^e_0, \ldots, \Pi^e_Q$ leur valeur de Π^e respective.
Début
$t_2 = t_1$, $r_2 = r_1$, $l = 0$ et $q = 0$.
Tant que $L_e < l$ et $q < Q$ **Faire**
$v = \frac{\lambda^{RD}}{\Pi^e_q}$ si < 1 sinon 1
$dt = Inf\left(\frac{Le-l}{v}, \pi_{q+1} - \pi_q\right)$
$t_2 = t_2 + dt$
$l = l + v.dt$
$r_2 = r_2 + v^2\Pi^e_q dt$
$q = q + 1$
Fin Tant que
Si $r_2 < R_{max}$ et $q < Q$ **Alors**
Retourner Réussite
Sinon
Retourner Échec
Fin Si
Fin

- **Deuxième approche :** *L'approche Risque versus Temps.*

Puisque $H(v) = v^2$, nous savons qu'à tout instant t pendant le parcours de e, la *vitesse de risque* associée $\frac{dr}{dt}$ est égale à $v(t)^2\Pi^e(t)$. Il vient que si on fixe λ^{RT} on obtient : $v(t) = Inf\left(1, \left(\frac{\lambda^{RT}}{\Pi^e(t)}\right)^{\frac{1}{2}}\right)$.
L'état résultant (i_2, t_2, r_2) sera obtenu à partir de λ^{RT} et (i_1, t_1, r_1) par le même processus itératif que pour l'approche *Risque versus Distance* (voir Algorithme 5.2).

- **Troisième approche :** *L'approche Distance versus Temps (vitesse moyenne).*

Fixer λ^{DT} revient à fixer le temps t_2 comme suit : $t_2 = t + \frac{Le}{\lambda^{DT}}$. Déterminer la fonction $t \mapsto v(t)$ et la valeur r_2 revient alors à résoudre le programme quadratique de minimisation du risque (voir Algorithme 5.3). Ce programme quadratique est convexe et peut être résolu par l'application directe des formules de Kuhn-Tucker du 1er ordre pour l'optimalité locale. Il vient alors que le vecteur $(\pi_1 - \pi_0, \ldots, \pi_Q - \pi_{Q-1})$ est parallèle au vecteur $(v_1\Pi^e_1(\pi_1 - \pi_0), \ldots, v_Q\Pi^e_Q(\pi_Q - \pi_{Q-1}))$. Il faut donc chercher le coefficient α de proportionnalité qui vérifie la contrainte temporelle du problème, mais s'assurer également que $v_q \leq 1$ pour tous q. Nous présentons une procédure pour trouver le coefficient de proportionnalité α qui s'assure que la fonction de vitesse ne dépasse jamais 1. Pour ce faire, le problème est traité en deux étapes : calculer le coefficient α pour les vitesses $v_q < 1$, puis vérifier s'il y existe $v_q \geq 1$ avec la nouvelle valeur de α. Si oui, bloqué ces vitesses à 1 et recommencer, Si non, la procédure est terminée (voir Algorithme 5.4).

Algorithme 5.3 : Procédure de transition Vitesse_Moyenne

Entrées : $\pi_0 = t_1$, $\pi_1, \ldots, \pi_Q = t_2$ les points de rupture de Π^e qui appartiennent à $[t_1, t_2]$ et $\Pi^e_0, \ldots, \Pi^e_Q$ leur valeur de Π^e respective.
Début
Résoudre le problème quadratique suivant :
Minimiser $\sum_q v_q^2\Pi^e_q(\pi_q - \pi_{q-1})$
tel que $\sum_q v_q(\pi_q - \pi_{q-1}) = t_2 - t_1$
avec $v_1, \ldots, v_Q \in [0, 1]$ les variables de décision.
Fin

5.3 Un algorithme découplé basé sur de la Programmation Dynamique

Pour ce premier algorithme, l'idée est de décomposer la résolution en deux parties puisque la solution recherchée est composée de deux éléments : un chemin et une fonction de vitesse. Nous allons donc chercher itérativement un élément en fixant l'autre. Autrement dit, nous allons d'abord résoudre le problème à chemin fixé (recherche de la fonction de vitesse) puis modifier le chemin en fonction du risque pris sur le chemin précédent. La première étape, l'*Évaluation* à chemin fixé, cherche la solution optimale du problème grâce à un schéma de Programmation Dynamique. La deuxième étape, la *Mise à jour* du chemin, le résultat de l'étape précédente nous donne des informations sur le trajet actuel et nous permet de connaître et d'éviter les zones à risques

Cette heuristique **Dec_SSPP** part de l'idée de simplifier le problème en fixant le chemin. Il ne reste donc que les fonctions de vitesse à déterminer. (voir Algorithme 5.5). Nous allons discuter des étapes d'évaluation et de mise à jour puis nous allons discuter des critères d'arrêts.

- **Première étape - évaluation du chemin courant** : Cette étape d'évaluation permet, par la résolution du problème SSPP à chemin fixé, de connaître la date d'arrivée à sa destination ainsi que les endroits où le VA prend le plus de risque sur le chemin considéré. Cette information sera utile dans la deuxième étape que nous présentons juste après. Dans toute la présentation de la méthode d'évaluation, Γ est donc fixé. Cette étape s'appuie sur une procédure de programmation dynamique **DP_Evaluate** dont les principales caractéristiques sont présentées ci-dessous et résumées dans la figure Figure 5.2.

Algorithme 5.4 : Calcul du coefficient de proportionnalité

Entrées : $\pi_0 = t_1$, $\pi_1, \ldots, \pi_Q = t_2$ les points de rupture de Π^e qui appartiennent à $[t_1, t_2]$ et $\Pi^e_0, \ldots, \Pi^e_Q$ leur valeur de Π^e respective.
Début
$Liste = \{1, \ldots, Q\}$
Pour q tel que $\Pi^e_q = 0$ **Faire**
$v_q = 1$ (la vitesse est maximale s'il n'y a pas de risque).
Bloquer v_q et enlever q de la liste $Liste$
Fin Pour
$l = L_e$ et $r_2 = 0$
Tant que il existe $q \in Liste$ tel que $v_q \geq 1$ **Faire**
Pour tout $q \in Liste$ tel que $v_q \geq 1$ **Faire**
$v_q = 1$
$r_2 = r_2 + (\pi_q - \pi_{q-1})\Pi^e_q$
$l = l - (\pi_q - \pi_{q-1})$
Bloquer v_q et enlever q de $Liste$
Fin Pour
$\frac{1}{\alpha} = \frac{1}{l} \sum_{q \in Liste} \frac{\pi_q - \pi_{q-1}}{\Pi^e_q}$
$v_q = \frac{\alpha}{\Pi^e_q}$
Fin Tant que
Retourner Succès
Fin

Algorithme 5.5 : SSPP - Méthode découplée (Dec_SSPP)

Entrées : Γ le chemin le plus court selon L de o à d
Début
$FIN = Faux$
Tant que NON FIN **Faire**
Première étape : *Évaluer* Γ et obtenir la date de sortie t_i de tout arc e_i de Γ
Seconde étape : *Mettre à jour* Γ
Toujours conserver la meilleure solution
Évaluer le critère d'arrêt
Fin Tant que
Fin

Algorithme 5.6 : Idée générale du schéma de programmation dynamique utilisé

Début
Initialiser $State[0]$ comme $\{(0,0)\}$ et $State[q]$ comme $\{\}$ pour tout $q > 0$
Pour $q = 1, \ldots, n-1$ **Faire**
Générer un ensemble Λ de décisions λ (λ^{RD}, λ^{RT} ou λ^{DT}, voir Section 5.2)
Pour tout $\lambda \in \Lambda$ une décision et $(t, r) \in State[q]$ un état **Faire** (I1)
Calculer (dans le cas où λ est faisable) l'état résultant (t_2, r_2)
S'il n'existe pas (t_1, r_1) dans $State[q+1]$ tel que $t_1 \leq t_2$ et $r_1 \leq r_2$ alors insérer (t_2, r_2) dans $State[q+1]$ et retirer de $State[q+1]$ tout (t_1, r_1) tel que $t_1 \geq t_2$ et $r_1 \geq r_2$
Fin Pour
Fin Pour
Fin

- Notons par $e_1, \ldots, e_n$ les arcs de Γ, et par $i_0, \ldots, i_n$ les nœuds associés. Ainsi, l'espace-temps de **DP_Evaluate** se présente sous la forme naturelle de l'ensemble $\{0, 1, \ldots, n\}$ et d'un état au temps $q = 0, 1, \ldots, n$, est une paire (t, r), où t signifie le moment où le véhicule V arrive en i_q, et r le risque cumulé à ce moment. De toute évidence, l'état initial est $(0, 0)$ et l'état final devrait être n'importe quelle paire (t, r) telle que $r \leq R_{max}$.
- Alors une décision à l'instant q devient une valeur λ $(\lambda^{RD}, \lambda^{DT}, \lambda^{RT})$ et une telle décision induit une transition $(q, t_1, r_1) \to (q + 1, t_2, r_2)$ comme décrit dans la Section 5.2, avec un coût $r_2 - r_1$.

D'après cela, le principe de Bellman peut être appliqué : l'algorithme **DP_Evaluate** parcourt l'espace-temps $\{0, 1, \ldots, n\}$, et, pour tout $q = 0, 1, \ldots, n$, calcule l'ensemble d'états associé $State[q]$, selon l'algorithme 5.6.

Si nous disposons déjà d'une solution **SSPP** réalisable (Γ^*, v^*) avec la valeur t^*, nous pouvons appliquer la règle de filtrage suivante :

- Règle de filtrage basée sur la borne inférieure : Soit (t_2, r_2) l'état impliqué dans l'instruction I1 de l'algorithme 5.6. Si $t_2 + L^*_{i(q+1),d} \geq t^*$ alors l'état (t_2, r_2) peut être tué : on ne l'insère pas dans $State[q + 1]$, car on ne peut pas s'attendre à ce qu'il soit étendu à une meilleure solution que la solution actuelle (Γ^*, v^*).

- **Deuxième étape - mise à jour du chemin** : Il repose sur un pré-processus appliqué au réseau de transit G et fait intervenir un seuil de proximité S_Prox. Pour deux nœuds quelconques i, j de G tels que $L^*_{i,j} \leq S\_Prox$, nous précalculons une collection $Path_{i,j}$ de chemin élémentaire de i à j. Cela nous fournit un opérateur $Detour$, qui agit sur n'importe quel chemin Γ via les paramètres i, j, γ comme suit :
  - i, j sont des nœuds de Γ tels que i précède j dans Γ
  - γ est un chemin dans $Path_{i,j}$
  - $Detour(\Gamma, i, j, \gamma)$ remplace la restriction $\Gamma_{i,j}$ de Γ de i à j par le chemin γ.

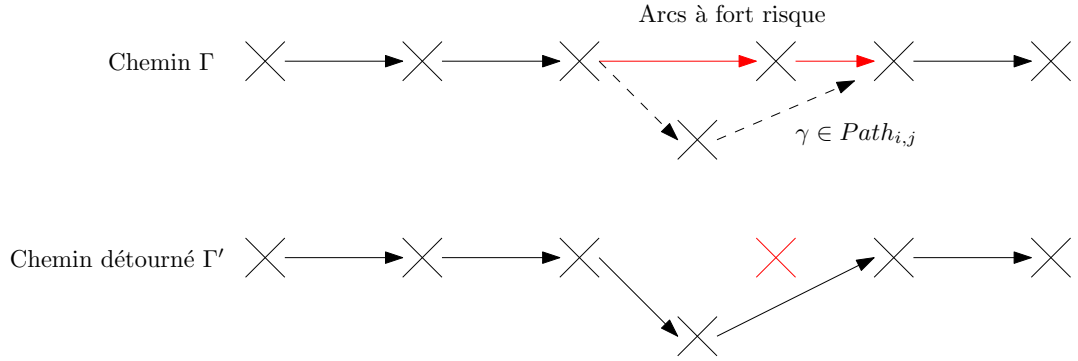

FIGURE 5.1 – Opérateur de détour

Puisque $Detour$ peut admettre un assez grand nombre de valeurs de paramètres (i, j, γ), nous identifions d'abord des couples de nœuds (i, j) dans Γ, tels que le coefficient de ralentissement $\left(\frac{t_j - t_i}{t^*_{i,j}}\right)$ est grand, et choisissez une telle paire (i, j). Ensuite on choisit le chemin γ dans $Path_{i,j}$ sous condition qu'il soit peu encombré entre le temps t_i et le temps t_j, c'est-à-dire qui soit tel que la somme, pour les arcs e de γ des valeurs moyennes de $\Pi^e(t)$ entre l'instant t_i et l'instant t_j est petit.

Remarque 3. *À la première itération, l'algorithme découplé commence par l'évaluation à chemin fixé. Le chemin choisi est le plus court chemin au sens de L.*

Remarque 4. *Si nous sommes capables de générer toutes les décisions susceptibles d'apparaître dans une séquence de décision optimale donnée ainsi que de générer le chemin optimal par l'opérateur Detour, alors l'algorithme* ***Dec_SSPP*** *ci-dessus est optimal.*

Espace temporel I	$I = \{0, \ldots, n\}$ L'ensemble ordonné des nœuds du chemin. Ne pas confondre le temps au sens du véhicule avec l'espace temporel de la programmation dynamique. Ces derniers seront désormais appelés "nœuds".
Espace des états S	$s = (t, r) \in S$ t (resp. r) est une date (resp. somme des risques) réalisable au nœud i.
Espace des décisions DEC(i,s)	$\lambda \in DEC(i, s)$ Au nœud i, v_i est une fonction de vitesse dérivée de λ_i et est réalisable sur l'arc $(i,\ i+1)$ avec t_{i+1} comme date de sortie. Ainsi $t_{i+1} - t_i \geq L_{(i,i+1)}$ et $r_i + risk\,(t_i, t_{i+1}, v_i) \leq R_{max}$.
Espace des transitions	$(i, s) \xrightarrow{\lambda_i} (i+1, s')$ Transition depuis (t, r) vers $(t_{i+1},\ r + risk\,(t, t_{i+1}, v_i))$
Principe de Bellman	Au nœud $i+1$ Seuls les états réalisables non dominés sont gardés : $\forall (t_1, r_1), (t_2, r_2) \in S$, si $t_1 \leq t_2$ et $r_1 \leq r_2$, (t_2, r_2) est dominé.
Stratégie de recherche	Traiter l'espace temporel I vers l'avant et construisez l'espace des états réalisables en conséquence.
Processus de filtrage	sont discutés Section 6.2.

FIGURE 5.2 – **DP_Evaluate** : Schéma de Programmation Dynamique utilisé

Remarque 5. *Nous pouvons transformer l'algorithme* ***Dec_SSPP*** *en un algorithme glouton en supprimant l'étape de mise à jour et en générant $DEC(i, s)$ de telle sorte que sa cardinalité soit 1.*

5.4 Un algorithme basé sur A*

L'algorithme A* [Hart et al., 1968] a été conçu pour gérer la recherche de chemin pour des robots évoluant dans des espaces d'états très grands (éventuellement infinis). Elle peut être adaptée à notre problème, puisque résoudre **SSPP** revient à rechercher un plus court chemin dans un réseau dépendant du temps, dont les nœuds sont des paires (i_1, t_1, r_1) et les arcs correspondent à la transition $((i_1, t_1, r_1) \rightarrow decision\ \ (e = (i_1, i_2), \lambda_e) \rightarrow (i_2, t_2, r_2))$. La décision λ_e peut être choisie, comme décrit dans la Section 5.2, parmi les trois types de décisions λ_e^{RD}, λ_e^{RT} ou λ_e^{DT}. L'algorithme A* s'appuie sur l'évaluation d'une borne inférieure de la valeur recherchée afin d'explorer certains nœuds prioritairement à d'autre. Par exemple, dans un réseau de transport dont la position physique des nœuds est connue, l'algorithme A* peut utiliser la distance à vol d'oiseau depuis un nœud courant vers le nœud de destination. Ainsi, chaque solution trouvée élimine un grand nombre de nœuds en cours d'exploration en comparant leur borne inférieure avec la solution courante : s'il n'est pas possible de faire mieux dans le meilleur des cas, alors il est inutile d'explorer cette branche de possibilités. Cet algorithme est encore très utilisé pour la recherche de chemin car il permet une exploration restreinte en garantissant l'obtention de la solution optimale à la fin du processus.

Pour explorer efficacement, l'algorithme A* se base sur deux listes. La première, ordonnée par la valeur de la borne inférieure de chaque nœud, est composée des nœuds en cours d'exploration. La deuxième garde en mémoire tous les nœuds explorés permettant de supprimer certains nœuds dont on sait qu'ils n'apparaîtront pas dans la solution optimale. Ces deux listes sont :

- Une liste d'expansion LE, qui contient les états (i, t, r), ordonnés selon leur borne inférieure W croissante. La borne inférieure de l'état (i, t, r) est égale à $t + L^*_{i,d}$ (parcourir le plus court chemin restant au sens de L à $v = 1$) et nous fournit une borne inférieure de la meilleure valeur possible

de la solution du **SSPP** (Γ, v) qui prolongerait le chemin nous permettant d'atteindre un état (d, t', r').

- Une liste pivot $LPivot$, contenant les nœuds déjà explorés. Il ne doit pas exister dans LE un élément (i, t, r) qui soit dominé par un autre élément (i, t', r') dans $LPivot \bigcup LE$, c'est-à-dire qui soit tel que $t' \leq t$ et $r' \leq r$.

Alors l'algorithme **A*_SSPP** peut être décrit comme suit : tant qu'il existe au moins un nœud dans la liste d'expansion LE, le garder en mémoire dans $LPivot$, générer des décisions $(e = (i_1, i_2), \lambda_e)$ et insérer tous les nœuds enfants dans LE s'ils sont intéressants (voir Algorithme 5.7).

Algorithme 5.7 : SSPP - Méthode A* (A*_SSPP)

Début
  $Best = +\infty$
  $LPivot = \{\}$
  $LE = \{(o, 0, 0)\}$
  Tant que $LE \neq \{\}$ **Faire**
    $(i_1, t_1, r_1) = Tete(LE)$
    Supprimer (i_1, t_1, r_1) de LE et l'insérer dans $LPivot$
    Pour tout arc $e = (i_1, i_2)$ **Faire**
      Générer l'ensemble Λ_e des décisions λ_e pour l'arc e (voir Section 5.2)
      Pour tout λ dans Λ **Faire**
        $(i_1, t_1, r_1) \rightarrow (e, \lambda) \rightarrow (i_2, t_2, r_2)$
        Si $i_2 = d$ **Alors**
          Insérez (d, t_2, r_2) dans $LPivot$ et mettre à jour $Best$
        Sinon
          $W_2 = t_2 + L*_{i_2,d}$
          Si $\begin{cases} r_2 \leq R_{max} \\ et\ W_2 < Best \\ et\ \nexists (i_2, t', r') \in LPivot \bigcup LE \text{ tel que } t' \leq t_2 \text{ et } r' \leq r_2 \end{cases}$ **Alors**
            Insérez (i_2, t_2, r_2) dans LE et supprimez de LE tout (i_2, t', r') tel que $t_2 \leq t'$ et $r_2 \leq r'$.
          Fin Si
        Fin Si
      Fin Pour
    Fin Pour
  Fin Tant que
  Retourner la solution Γ du **SSPP** liée au meilleur état (d, t, r) dans $LPivot$
Fin

Remarque 6. *Si nous sommes capables de générer toutes les décisions susceptibles d'apparaître dans une séquence de décision optimale donnée, alors l'algorithme* ***A*_SSPP*** *ci-dessus est optimal.*

Remarque 7. *Nous pouvons transformer l'algorithme* ***A*_SSPP*** *en algorithme de plus court chemin en réduisant* Λ *à 1 élément.*

Section 6

Génération heuristique des décisions et procédés de filtrages

Nous présentons, dans cette section, une méthode pour générer des décisions qui peuvent être utilisées dans les trois schémas de décisions λ^{RD}, λ^{RT} et λ^{DT} présentés Section 5.2. Cette méthode n'offre aucune garantie d'obtenir la solution optimale mais permet une vaste exploration et une capacité d'exploitation performante et rapide afin de s'adapter au contexte dynamique propre aux VAs. Nous présentons un procédé générant un petit nombre K de décisions autour d'une valeur connue, puis nous abordons la notion d'apprentissage par renforcement pour trouver une bonne valeur directrice.

Dans un deuxième temps, nous allons présenter une méthode de filtrage des états résultants des transitions réalisées avec chaque décision générée. En effet, si tous les états sont gardés, leur nombre croîtra exponentiellement, entraînant le temps de résolution avec lui. Il faut donc le limiter en filtrant les états. Nous présentons un procédé de filtrage heuristique, puis nous abordons la notion d'apprentissage par renforcement pour reconnaître et filtrer les meilleurs états.

6.1 Générer un nombre limité de décisions

6.1.1 Un processus heuristique de génération de décisions

À partir de l'état $s = (t_i, r_i)$, une infinité de décisions peuvent être générées. Mais la plupart d'entre elles sont inutiles ou pas assez prometteuses pour être prises en compte (l'état résultant est trop lent, trop risqué, plus lent et plus risqué qu'un autre état, etc.). Sachant que la valeur optimale λ_{opt} est inconnue, nous proposons de générer des décisions pour quelques valeurs de λ entre des estimations basses et hautes de λ_{opt} : λ_{inf} et λ_{sup}. La génération de ces décisions sera dirigée par une valeur λ_{moy}. La moitié des décisions seront alors générées à intervalles réguliers entre λ_{inf} et λ_{moy} et l'autre moitié entre λ_{moy} et λ_{sup}.

Ces trois valeurs seront calculées à partir d'informations sur l'instance (R_{max}, $L^*_{o,d}$ et la solution de l'algorithme glouton), d'informations sur l'état courant (r_i le risque pris jusqu'au nœud i, d_i la distance parcourue jusqu'au nœud i et t_i le temps pris pour arriver au nœud i) et d'un nouveau paramètre ρ qui fait le lien entre λ_{inf}, λ_{moy} et λ_{sup}, et détermine donc la taille de l'intervalle d'exploration (voir équation 6.2). Plusieurs choix sont possibles pour lier ces trois valeurs. Nous proposons un lien de multiplication entre eux : λ_{inf} est ρ fois plus petit que λ_{moy} qui est ρ fois plus petit que λ_{sup}. Nous contrôlons alors la quantité de risque que pourra prendre un VA avec ces nouvelles valeurs car la vitesse sera, au pire, multiplié par ρ (dans le cas λ^{RD}) et le risque sera donc multiplié par ρ^2.

En ce qui concerne la valeur de λ_{moy}, qui dirige l'ensemble des décisions qui seront prises, nous utilisons la même méthode utilisée dans l'algorithme glouton : nous considérons que le rapport choisi (*risque sur distance*, *risque sur temps* ou *distance sur temps*) est égale sur tous les arcs restants à parcourir et qu'il est donc égale au rapport en considérant le chemin globalement. Dans le cas *risque sur distance*, la valeur λ^{RD}_{moy} est égale au rapport du risque disponible sur la distance restante à parcourir mais, pour le *risque sur temps* et le *distance sur temps*, il nous manque l'information temporelle. C'est pour cette raison que

nous avons besoin de la solution de l'algorithme glouton. Cette solution nous permet d'obtenir une borne supérieure de la date optimale et nous permet d'estimer λ_{moy}. Nous avons donc que λ^{RT}_{moy} est égale au rapport du risque disponible sur la différence entre date actuelle et solution gloutonne et nous avons que λ^{DT}_{moy} est égale au rapport de la distance restante et de la différence des deux dates (voir équation 6.1).

$$\lambda^{RD}_{moy} = \frac{R_{max} - r_i}{L^*_{o,d} - d_i} \qquad \lambda^{RT}_{moy} = \frac{R_{max} - r_i}{t^{glouton}_p - t_i} \qquad \lambda^{TD}_{moy} = \frac{t^{glouton}_p - t_i}{L^*_{o,d} - d_i} \tag{6.1}$$

$$\lambda_{inf} = \frac{1}{\rho} \cdot \lambda_{moy} \qquad \lambda_{sup} = \rho \cdot \lambda_{moy} \tag{6.2}$$

Une dernière question se pose, maintenant que l'intervalle d'exploration est fixé, quant au nombre de décisions à prendre. Nous avons déjà précisé que les décisions seront prises à intervalles réguliers entre λ_{inf} et λ_{moy} dans un premier temps, puis entre λ_{moy} et λ_{sup}. Nous avons choisi de prendre autant de décisions d'un côté que de l'autre en prenant toujours λ_{moy} comme l'une des décisions. Nous fixons alors le nombre de décisions à prendre à $2K + 1$ éléments où K est un nouveau paramètre. Si nous fixons $K = 0$, alors la décision est λ_{moy}. Si $K = 1$, les trois décisions sont λ_{inf}, λ_{moy} et λ_{sup}. Si $K > 1$, alors la moitié des décisions seront générées à intervalles réguliers entre λ_{inf} et λ_{moy} et l'autre moitié entre λ_{moy} et λ_{sup}. Par exemple, si $K = 3$, 7 décisions sont générées comme présentées sur la figure 6.1.

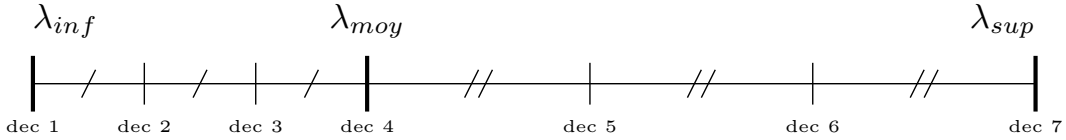

FIGURE 6.1 – Génération des décisions dans l'intervalle $[\lambda_{inf}; \lambda_{moy}]$ et $[\lambda_{moy}; \lambda_{sup}]$

6.1.2 Apprendre à reconnaître les décisions prometteuses

Ces trois valeurs (λ_{inf}, λ_{moy} et λ_{sup}) peuvent donc être calculées grâce aux informations de l'instance (R_{max}, $L^*_{o,d}$ et la solution de l'algorithme glouton) et de l'état courant si aucune autre information n'est disponible mais, d'une exécution à l'autre, il est possible de récupérer des informations. Nous proposons de conserver les décisions λ des résolutions de précédents chemins car ces derniers peuvent se chevaucher avec le chemin de la nouvelle tâche. Ainsi, lorsqu'un nouveau chemin est choisi, les décisions optimales des résolutions précédentes empruntant les mêmes arcs nous permettent de déduire un bon intervalle $[\lambda_{inf}, \lambda_{sup}]$ pour chaque arc. Dans ce cas, le processus de génération de décisions devient indépendant des états courants.

Pour déterminer le meilleur intervalle d'exploration, nous procédons en deux étapes : déterminer $[\lambda_{min}, \lambda_{max}]$, puis trouver λ_{moy} optimal dans cet intervalle. Pour commencer, la plage de décision $[\lambda_{min}, \lambda_{max}]$ est choisi comme le plus petit segment contenant toutes les décisions optimales des résolutions précédentes. Puis la plage est découpée en $2K$ intervalles possédant le même pourcentage de décision. Par exemple, si $K = 2$, nous divisons l'intervalle $[\lambda_{min}, \lambda_{max}]$ en 4 intervalles $[\lambda_{inf}, \lambda^{\frac{1}{4}}]$, $[\lambda^{\frac{1}{4}}, \lambda_{moy}]$, $[\lambda_{moy}, \lambda^{\frac{3}{4}}]$ et $[\lambda^{\frac{3}{4}}, \lambda_{sup}]$ de manière que :

- $\frac{1}{4}$ des décisions optimales λ_i appartiennent à l'intervalle $[\lambda_{inf}, \lambda^{\frac{1}{4}}]$;
- $\frac{1}{4}$ des décisions optimales λ_i appartiennent à l'intervalle $[\lambda^{\frac{1}{4}}, \lambda_{moy}]$;
- $\frac{1}{4}$ des décisions optimales λ_i appartiennent à l'intervalle $[\lambda_{moy}, \lambda^{\frac{3}{4}}]$;
- $\frac{1}{4}$ des décisions optimales λ_i appartiennent à l'intervalle $[\lambda^{\frac{3}{4}}, \lambda_{sup}]$.

Si aucune résolution précédente n'est disponible, un prétraitement peut être appliqué au réseau de transit en générant des chemins aléatoires résolus par un algorithme glouton pour générer une date de sortie et apprendre de toutes ces décisions. La génération de chemins aléatoires peut se terminer après un nombre fixe de générations ou lorsque les valeurs λ semblent se stabiliser.

6.2 Le filtrage des états générés

Malgré le petit nombre de décisions utilisées par états, sans limitation de ces derniers, le nombre de nouveaux états est exponentiel et entraîne le temps de calcul avec lui. Cela n'est pas souhaitable dans notre cas car le système doit pouvoir s'adapter en temps réel à toute situation. Il convient donc de limiter le nombre d'états sur chaque nœud pour garder une adaptabilité nécessaire au contexte des VAs.

Dans un premier temps, nous appliquons un filtre logique lié au principe de Bellman qui garantit la suppression d'états n'appartenant pas à la solution optimale. En effet, le principe d'optimalité de Bellman, quand il est applicable, exprime l'idée qu'une solution optimale peut être construite à partir d'une solution optimale d'un sous-problème. Il est facile de se convaincre que ce principe est vrai dans notre cas en considérant deux états au même nœud i et à un arc du nœud de destination $e = (i, d)$. Ces deux états $s_1 = (i, t_1, r_1)$ et $s_2 = (i, t_2, r_2)$ sont tels que $t_1 < t_2$ et $r_1 < r_2$, l'état s_2 est dit *dominé* par s_1. L'état s_1 est une meilleure solution que s_2 comme chemin jusqu'au nœud i et il ne peut exister aucune décision sur l'arc e depuis s_2 qui permettrait d'obtenir une meilleure solution au nœud d que depuis s_1. En effet, une décision depuis s_2 ne peut prendre que $R_{max} - r_2$ unités de risque et cette décision est également possible depuis s_1. Cependant, $t_1 < t_2$ donc, pour la même quantité de risque pris, l'état résultant de la transition de e depuis s_1 arrivera plus tôt que ceux partant de s_2.

Nous avons donc un premier filtrage logique, lié au principe de Bellman, nous permettant de supprimer tout état dominé par un autre. Mais ce filtrage ne permettra pas de maintenir un nombre raisonnable d'états. Nous proposons de garder une limite fixe S_{max} sur le nombre d'états. Nous proposons alors de trier les états selon une fonction d'évaluation que nous présentons ci-dessous, puis nous proposons une méthode pour mettre à jour cette fonction d'évaluation à toutes itérations pour maintenir un processus efficace.

6.2.1 Évaluation et filtrage des états

Après avoir appliqué les différentes transitions ayant généré de nouveaux états au nœud i, la taille de l'ensemble S^i est très grande et dépasse très largement le seuil S_{max} fixé. Afin de filtrer l'ensemble d'états S^i lié à un nœud donné i, plusieurs techniques peuvent être appliquées. On pourra, par exemple, considérer comme équivalents 2 états (t, r) et (t', r') tels que $|t - t'| + |r - r'|$ n'excède pas une petite valeur ϵ à définir. Nous n'allons pas suivre cette approche qui ne garantit pas que nous allons maintenir la cardinalité de S^i en dessous du seuil imposé S_{max}. Nous proposons plutôt d'ordonner les états de S^i selon une fonction d'évaluation. Cette fonction est une simple somme pondérée du temps et du risque, mais la difficulté majeure est d'avoir une pondération pertinente (voir équation 6.3). L'objectif est que l'état apparaissant dans la solution optimale soit le meilleur selon la fonction d'évaluation, c'est-à-dire qu'il soit le minimum de la fonction.

$$eval(s = (i, t, r)) = \omega \cdot t + r \tag{6.3}$$

où ω est une valeur de risque par unité de temps. Intuitivement, ω devrait être égal à $\frac{R_{max}}{t^*}$, où t^* est la valeur de la solution optimale du **SSPP** sur l'arc considéré, et nous souhaitons pouvoir calculer sa valeur en fonction des principales caractéristiques des instances du **SSPP**. Les caractéristiques les plus pertinentes semblent être le seuil de risque R_{max}, la longueur L_Γ du chemin Γ, la moyenne $\overline{\Pi}$ des fonctions $\Pi^e, e \in A$ et la fréquence F des points de rupture de ces fonctions. Nous pouvons remarquer que dans le cas où toutes les fonctions Π^e sont constantes et toutes égales à $\overline{\Pi}$, alors la vitesse optimale v^* va être constante et égale à $\frac{R_{max}}{L^* \overline{\Pi}}$. En effet, si les fonctions Π^e sont toutes constantes (et ne dépendent donc plus du temps), le problème devient équivalent à un problème d'optimisation simple :

$$\begin{aligned} &\text{minimiser } t^* = \frac{L^*}{v^*} \\ &\text{sachant que } H(v^*) \cdot t^* \cdot \overline{\Pi} \leq R_{max} \end{aligned}$$

où, pour rappel, $H(v) = v^2$. Le temps optimal t^* est donc égal à $\frac{L^*}{v^*} = \frac{\overline{\Pi} L^{*2}}{R_{max}}$. Cela nous conduit à initialiser ω de façon à avoir $\omega t^* = R_{max}$ d'après la proposition 3, c'est-à-dire $\omega = \frac{R_{max}^2}{\overline{\Pi} L_\Gamma^2}$ où la longueur du chemin optimal L^* est remplacée par la longueur du chemin courant L_Γ puisque le chemin optimal est inconnu.

Néanmoins, cette façon de calculer ω peut induire des distorsions puisque cette valeur est pertinente pour un chemin entier mais il n'y a aucune raison qu'elle soit la même pour chaque arc. Un manque de souplesse dans la procédure de filtrage associé à une valeur ω non parfaitement ajustée peut donner, pour un nœud donné i, une collection S^i mal équilibrée, dans le sens où les valeurs (t, r) devraient se répartir comme un large ensemble de Pareto.

Dans l'idéal, la valeur de ω est égale au rapport du temps et du risque de l'état correspondant dans la solution optimale. Cette valeur est inconnue mais nous en connaissons une approximation grâce à la méthode *risque versus temps* (voir section 5.2). Nous pouvons alors calculer les valeurs de λ^{RT} pour chaque état de S^i et considérer ω comme étant la moyenne de tous ces λ^{RT}.

Maintenant, si $card(S^i) > S_{max}$, des états doivent être supprimés. Si la valeur ω est une bonne approximation de ω_{opt}, alors les S_{max} états de plus petites valeurs $eval(s)$ sont les meilleurs et eux seuls seront conservés dans S^i. Cependant, si ce n'est pas le cas, un état de valeur $eval(s)$ haute peut être meilleur que l'état de plus basse valeur $eval(s)$. Par conséquent, il est nécessaire de déterminer la qualité d'approximation de ω. Pour ce faire, nous proposons de calculer l'écart des risques des états de S^i par rapport au pourcentage parcouru du chemin $\Gamma_{\%_i}$. L'idée est que si un chemin partiel a parcouru un certain pourcentage du chemin, le risque pris sur ce chemin ne doit pas être loin du même pourcentage (voir équation 6.4). Si ce n'est pas le cas, les valeurs de λ^{RT} et, à fortiori, ω n'ont pas beaucoup de chance d'être de bonnes approximations.

$$\Delta = \frac{\sum_{(i,t,r)\in S^i} \left| \frac{r}{R_{max}} - \Gamma_{\%_i} \right|}{card(S^i)} \tag{6.4}$$

Si la valeur absolue de Δ est élevée, les états générés prennent, en moyenne, trop ou trop peu de risque (ce qui signifie qu'ils peuvent aller plus vite). Il faut, dans ce cas, garder des nœuds moins bien évalués puisqu'ils sont potentiellement meilleurs. Nous proposons de découper l'ensemble S^i en trois tiers (les meilleurs, moyen et les moins bien évalués) et de supprimer un certain nombre de nœuds de manière différente dans chaque tiers de S^i en fonction de la valeur de Δ :

- Si $|\Delta|$ est "haut" (> 0.5) : ω est supposé être une mauvaise approximation : les pires $\left\lfloor \frac{card(S^i) - S_{max}}{3} \right\rfloor$ états sont supprimés de chaque tiers de S^i indépendamment.
- Si $|\Delta|$ est "moyen" : les pires $\left\lfloor \frac{card(S^i) - S_{max}}{2} \right\rfloor$ états sont retirés de l'union des 1er et 2ème tiers de S^i et du 3ème tiers de S^i indépendamment.
- Si $|\Delta|$ est "faible" (proche de 0) : ω est supposé être une bonne approximation : les S_{max} premiers états sont conservés et tous les autres sont supprimés.

6.2.2 Apprendre une bonne valeur

Cependant, il se peut alors que notre technique d'élagage produise des paires (t, r) qui, prises dans leur ensemble, sont soit trop risquées, soit trop prudentes. Afin de contrôler ce type d'effet, nous proposons une méthode pour que la valeur ω devienne auto-adaptative.

Plus précisément, on part de $\omega = \frac{R_{max}^2}{\overline{\Pi} L_\Gamma^2}$ tel qu'expliqué précédemment. Cette valeur est donc initialement indépendante des états, mais elle va évoluer par apprentissage tout au long d'une (ou plusieurs) exécution de nos algorithmes. Afin de mieux l'expliquer, nous nous concentrons sur le cas de l'algorithme **DP_Evaluate**. À une certaine itération de l'algorithme, nous avons traité l'arc e_i et calculé l'ensemble d'états courant S^i, tout en mettant à jour la valeur ω. Après l'application des décisions de $\bigcup_{s\in S^i} DEC(s, i)$ et l'application du

principe de Bellman sur les états résultants $(i+1,t,r)$, la taille de l'ensemble d'états S^{i+1} dépasse S_{max}. Les états $(i+1,t,r)$ de S^{i+1} sont alors triés selon les valeurs de la fonction d'évaluation avec une certaine valeur de ω apprise jusqu'à présent. Idéalement, les états (t,r) ainsi ordonnés devraient faire en sorte que les meilleurs S_{max} états (t,r) soient équilibrés dans le sens où les états risqués et les états prudents sont en nombre équivalent. En d'autres termes, la moyenne des ratios $\frac{r}{R_{max}}$ des S_{max} meilleurs états doit être proche du ratio $\Gamma_{\%_i}$. Si, par exemple, ces valeurs sont centrées significativement au-dessus de ce ratio, alors nous avançons de manière trop risquée et devons faire baisser ω pour donner moins d'importance au temps. À l'inverse, si ces meilleures valeurs sont centrées en dessous de ce ratio, alors nous sommes trop prudents et devons faire augmenter ω pour donner plus d'importance au temps.

Il existe de nombreuses méthodes de modification de ω (pas fixe ou décroissant en fonction de l'écart des ratios de risque et de distance par exemple) et est sûrement dépendant du problème étudié. Nous n'avons cependant étudié qu'un seul problème et nous ne sommes pas en mesure d'en mesurer l'impact correctement. Nous avons ici choisi de multiplier ω par un facteur dépendant de la moyenne des écarts des ratios de risque et de distance. Si l'écart des ratios est négatif, ω doit augmenter donc le facteur doit être supérieur à 1. Inversement, si l'écart des ratios est positif, ω doit baisser donc le facteur doit être inférieur à 1. Nous avons choisi de calculer le facteur comme étant $1 - 0.1 \times moyenne\_ecart$ (voir équation 6.5), car c'est la méthode qui donne globalement les meilleurs résultats pour notre problème. La valeur de ω ainsi obtenu est alors utilisée pour le filtrage des états puis réutilisé pour apprendre la nouvelle valeur lors du filtrage de l'arc suivant.

$$\omega = \omega(1 - 0.1\frac{\sum_{(i,t,r)\in S^i} \frac{r}{R_{max}} - \Gamma_{\%_i}}{card(S^i)}) \tag{6.5}$$

Section 7

Amélioration de solutions par recherches locales

Dans cette section, nous explorons des méthodes bien différentes des trois schémas algorithmiques précédents (**Greedy_SSPP**, **DP_SSPP** et **A*_SSPP**) puisque, ici, nous démarrons avec une solution réalisable et tentons de l'améliorer. Ces méthodes permettront, par la suite, d'obtenir une meilleure borne supérieure à donner à nos deux algorithmes les plus complexes notamment, mais aussi de tenter d'améliorer n'importe quelle solution.

Nous proposons trois méthodes dont deux impliquant de l'apprentissage. La première méthode tente d'améliorer localement la solution en étudiant le changement de fonctions de risque aux intersections. La deuxième méthode considère les décisions comme un ensemble et tente d'améliorer la solution de façon macroscopique en apprenant une direction améliorante. La troisième méthode applique un schéma de descente de gradient aux intersections avec l'idée qu'une solution globale est également une solution locale.

7.1 Prise en compte du changement de fonctions de risque aux intersections

L'idée derrière cet algorithme est la suivante : La décision gloutonne du schéma *Risque versus Distance* peut être légèrement améliorée en étudiant la jonction avec l'arc suivant. Si la fin de l'arc en cours est risquée alors que le début du suivant ne l'est pas, accélérer est peut-être une meilleure solution. Ainsi, la nouvelle solution profite du calme de l'arc suivant et évite le risque de l'arc précédent.

L'algorithme **RechLoc_SSPP** va fonctionner en faisant en sorte que le véhicule V s'entraîne plusieurs fois le long du chemin Γ. Lors du premier entraînement, V suivra $\Gamma = \{e_q, q = 0 \dots Q-1\}$. Ensuite, à chaque arc e_q le long de Γ, V va prendre des décisions qui vont viser à améliorer sa trajectoire actuelle. Les mêmes notations sont utilisées : (t_q, r_q) est l'état au début de l'arc e_q sur lequel t_q est le temps au début de l'arc e_q, r_q est le risque du parcours jusqu'à e_q et la décision standard de l'arc suivant est dépendant de l'état précédent $\lambda^{RD} = \frac{R^{max} - r_q}{\Gamma_{\%q}}$ (voir Section 6). Un état final (t_Q, r_Q) est ajouté et représente ces mêmes informations à la fin du dernier arc e_{Q-1}. Lors des entraînements suivants, nous utiliserons une nouvelle forme d'états : (t_q, r_q, λ_q) où λ_q est la décision prise sur l'arc e_q. La figure 7.1 résume les notations

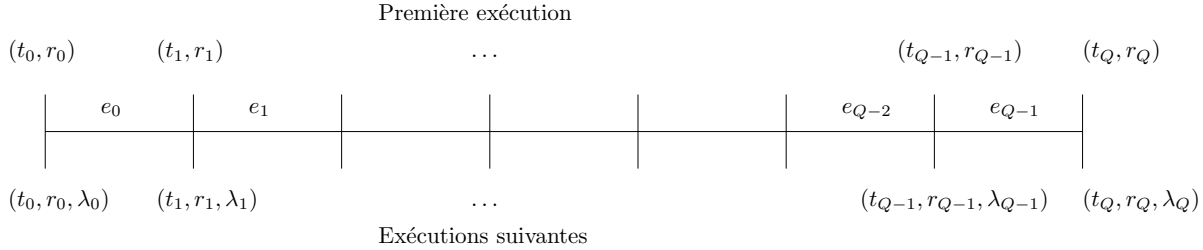

Figure 7.1 – Résumé des notations utilisées pour l'algorithme de recherche locale.

présentées précédemment. Un état final $(t_Q, r_Q, \lambda_Q = null)$ est ajouté et représente ces mêmes informations à la fin du dernier arc e_{Q-1}. Nous nous intéressons alors à la jonction entre deux arcs, c'est-à-dire entre deux états (t_q, r_q, λ_q) et $(r_{q+1}, t_{q+1}, \lambda_{q+1})$. Les fonctions de risque de l'arc e_q légèrement avant t_{q+1} et de l'arc e_{q+1} légèrement après t_{q+1} sont comparées et de leur différence d_Π est déduite une accélération ou ralentissement sur l'arc e_q pour éviter un passage dangereux (voir figure 7.2).

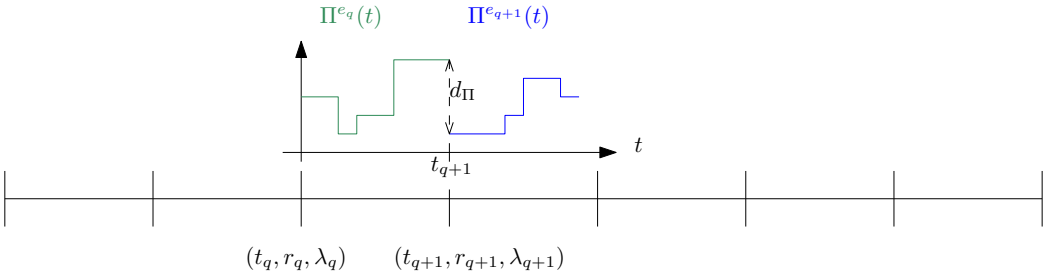

FIGURE 7.2 – Comparaison des fonctions de risque pour réduire le risque localement.

Cependant, il n'y a pas de choix évident quant à la valeur de cette accélération (positive ou négative) :

- accélérer d'une valeur fixe : permet de mieux contrôler les modifications au prix d'un plus grand nombre d'itérations
- accélérer tant que l'arc suivant est moins risqué (ou ralentir tant que plus risqué) : puisque cette méthode ne tient pas compte du risque sur l'arc, elle peut engendrer de trop grandes vitesses.
- accélérer pour minimiser le risque sur l'arc courant. Pour ne pas ralentir la méthode, même si le risque pris par V est quelconque, une dichotomie est supposée suffisante pour trouver une bonne valeur.

Si la méthode de modification se porte sur une valeur fixe, alors nous proposons de vérifier, avant modification, le gain ou la perte de risque si la décision est augmentée ou diminuée d'une petite valeur ϵ. Puis de choisir l'accélération la plus pertinente. Nous calculons donc le changement de risque en cas d'accélération r_q^{left} et en cas de décélération r_q^{right} tel que :

$$r_q^{left} = -\int_{t-dt}^{t} \Pi_t^{e_q} \cdot dt + \int_{t-dt}^{t} \Pi_t^{e_{q+1}} \cdot dt \qquad r_q^{right} = \int_{t}^{t+dt} \Pi_t^{e_q} \cdot dt - \int_{t}^{t+dt} \Pi_t^{e_{q+1}} \cdot dt$$

où $dt = \frac{d_\Pi}{\lambda_q} - \frac{d_\Pi}{\lambda_q + \epsilon}$ pour r_q^{left} et $dt = \frac{d_\Pi}{\lambda_q - \epsilon} - \frac{d_\Pi}{\lambda_q}$ pour r_q^{right}.
Alors :

- Si aucun parmi r_q^{left} et r_q^{right} n'est strictement négatif, alors on applique la décision λ_q à l'arc e_q, et se déplacer vers le nœud final de e_q.
- Si r_q^{left} est strictement négatif (resp. r_q^{right}), et l'autre non, alors nous appliquons la décision $\lambda_q = \lambda_q + \epsilon$ (resp. $\lambda_q = \lambda_q - \epsilon$) où ϵ est une petite valeur arbitraire.
- Si r_q^{left} et r_q^{right} sont strictement positifs, alors nous comparons leurs valeurs et modifions la décision comme dans le cas précédent : si $r_q^{left} \geq r_q^{right}$, alors nous procédons comme dans le cas précédent : avec r_q^{left} nous appliquons la décision $\lambda_q = \lambda_q + \epsilon$, sinon $\lambda_q = \lambda_q - \epsilon$.

L'algorithme se présente alors sous la forme de l'algorithme 7.1 dans lequel le signal d'arrêt est déclenché soit lorsque le nombre d'itérations a atteint un seuil, soit lorsque à la fin d'une itération, la séquence de décision $\lambda_0, \ldots, \lambda_Q$ n'a pas été modifiée.

7.2 Des méthodes apprenantes

Dans cette section, deux approches d'apprentissage seront présentées. La première utilise les modifications de la séquence de décisions vu comme un ensemble insécable que nous appelons direction et

Algorithme 7.1 : Algorithme RechLoc_SSPP

Entrées : Une solution réalisable
Début
$Sol_{best} = \{(t_q, r_q, \lambda_q), q = 0 \dots Q\}$ les valeurs de la solution initiale
$T_{best} = t_Q$.
Tant que signal d'arrêt non déclenché **Faire**
Modifier les valeurs λ_q le long de Γ itérativement avec la méthode choisie (accélération fixe, tant que moins risqué ou par minimisation).
Si $t_Q < T_{best}$ **Alors**
Mettre à jour Sol_{best} et T_{best}
Fin Si
Vérifier la condition d'arrêt
Fin Tant que
Fin

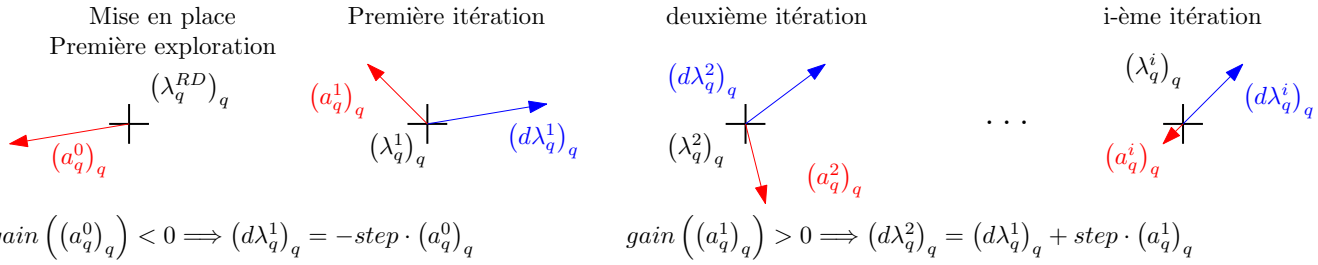

FIGURE 7.3 – Schéma de la méthode d'apprentissage d'une bonne direction de modification.

se déplace dans l'ensemble des solutions selon que la dernière direction améliore ou non la solution. La seconde approche repose sur le fait qu'une solution globale est aussi une solution locale, modifiant ainsi les décisions vers des solutions locales.

7.2.1 Apprendre une direction améliorante

L'algorithme **Direction_SSPP** que nous allons maintenant présenter introduit le principe d'apprentissage sous la forme d'une recherche de direction améliorante. Cet algorithme fonctionnera en faisant en sorte que le véhicule V s'entraîne plusieurs fois le long du chemin Γ. La première fois, V suivra Γ de taille Q en appliquant les décisions obtenues par un des algorithmes présentés précédemment. On note, à chaque arc e_q le long de Γ, (t_q, r_q, λ_q) respectivement le risque, le temps de parcours au début de l'arc e_q et la décision prise sur l'arc e_q.

L'idée ici est de modifier légèrement toutes les décisions en même temps, nous appelons cela une direction $dir = (d\lambda_1, \dots, d\lambda_{Q-1})$, et d'évaluer leur impact. À la prochaine exécution, si la direction avait un gain positif (c'est-à-dire arrivée plus tôt à la fin de Γ), la même direction sera reprise à l'itération suivante, sinon la direction inverse sera prise. Une fois la direction sélectionnée, un pas d'amplitude *step* est effectué. La combinaison d'une direction est d'une amplitude est nommée *mouvement* dans la suite du document. De plus, l'exploitation de l'apprentissage étant inexistant sans exploration : nous ajoutons un mouvement de direction aléatoire $dir_{alea} = (a_1, \dots, a_{Q-1})$ et d'amplitude décroissante avec les itérations.

Pour garantir une solution réalisable et efficace, la décision sur le dernier arc est systématiquement recalculée selon la reformulation *Risque versus Distance*. La méthode se présente donc sous la forme suivante :

1– Initialiser les décisions avec un des algorithmes présentés précédemment.
2– Choisir aléatoirement une direction, appliquer le mouvement associé (amplitude dépendant de l'itération) et calculer la dernière décision.

3– Évaluer la nouvelle solution.
4– Si la dernière direction évaluée a un gain positif (a amélioré la solution), garder cette solution.
5– Sinon, revenir à l'état précédant et appliquer le mouvement inverse.
6– Évaluer la condition d'arrêt, augmenter le compteur des itérations et revenir à l'étape 2.

Un schéma résumé de la méthode est présenté figure 7.3 et l'algorithme est présenté en pseudo-code dans l'algorithme 7.2 où le calcul des valeurs (t_q, r_q, λ_q) (instruction (I1)) est présenté dans l'algorithme 7.3.

Algorithme 7.2 : Algorithme Direction_SSPP

Entrées : Une première exécution de l'algorithme glouton présenté dans la Section 4.3
Début
Cette première exécution nous fournit une solution réalisable Sol_{best}
ainsi qu'une valeur de temps t_{best}.
$step = 0.5$
Tant que signal d'arrêt non déclenché **Faire**
Calculer les valeurs (t_q, r_q, λ_q) le long de Γ (I1)
Si $t_Q < t_{best}$ **Alors**
Mettre à jour Sol_{best} et t_{best}
$r = r^{max} - \sum_q r_q$
Fin Si
$step = 0.8 \cdot step$
Vérifier la condition d'arrêt
Fin Tant que
Fin

Algorithme 7.3 : (I1) Calcul des valeurs (t_q, r_q, λ_q)

Entrées : $step$ l'amplitude courante, $\left(d\lambda_q^i\right)_q$ le mouvement appliqué à l'exécution précédente et $gain$ le gain associé au mouvement aléatoire précédent.
Début
Pour tout $q = 0, \ldots, Q-1$ **Faire**
$d\lambda_q^{i+1} = d\lambda_q^i + sign(gain) \cdot step \cdot a_q^i$
Appliquer la transition et passer à l'arc suivant
Fin Pour
Appliquer la transition sur le dernier arc selon la reformulation RD.
Évaluer le gain sans le mouvement aléatoire
Tirer une direction aléatoirement $\left(a_q^{i+1}\right)_q$
Pour tout $q = 0, \ldots, Q-1$ **Faire**
$d\lambda_q^{i+1} = d\lambda_q^{i+1} + a_q^{i+1}$
Appliquer la transition et passer à l'arc suivant
Fin Pour
Appliquer la transition sur le dernier arc selon la reformulation RD.
Évaluer le gain du mouvement aléatoire pour la prochaine itération
Fin

7.2.2 Apprendre les décisions à partir de critères locaux

Comme l'algorithme **Direction_SSPP**, le **LocMin_SSPP** que nous allons présenter maintenant passera plusieurs fois sur Γ. Cependant, cette fois, les décisions seront modifiées en essayant de minimiser le risque localement. Ensuite, les décisions seront légèrement augmentées pour que le véhicule accélère et utilise le risque qui a été récupéré localement.

Pour ce faire, les arcs seront pris deux par deux. La date d'entrée du premier arc et la date de sortie du deuxième arc sont considérées comme fixes et le problème local de minimisation du risque entre ces deux dates entraînera des modifications des décisions. Pour ce faire, nous étudions la dérivée des fonctions de risque et appliquons une méthode de descente de gradient, à la différence qu'ici, la performance locale est étudiée itérativement. Une fois la descente de gradient effectuée à toutes les intersections, la performance globale est calculée pour ne toujours garder que la meilleure solution.

Là encore, nous n'allons pas chercher la solution exacte car nous voulons un algorithme très rapide. Nous allons plutôt essayer de faire tendre la dérivée de la fonction d'optimisation vers 0. Ainsi, lors de la transition d'un arc pour une décision spécifique λ, les dérivées G_t^λ de t par rapport à λ et G_r^λ la dérivée de r par rapport à λ seront calculées. Comme pour la fonction de vitesse, $(G_t^\lambda, G_r^\lambda)$ peut être calculé sur chaque plateau de risque :

- pour tout plateau de risque avant la fin de l'arc, $G_t^\lambda = 0$ et $G_r^\lambda = G_r^\lambda + 2.v.\Pi.dT.G_v^\lambda$
- pour le dernier plateau de risque, $G_t^\lambda = G_v^\lambda.\frac{dL}{v^2}$ et $G_r^\lambda = G_r^\lambda + 2.v.\Pi.dT.G_v^\lambda + v^2.\Pi.G_v^\lambda$

où Π (resp. dT) est la valeur (resp. durée) du plateau de risque, G_v^λ est la dérivée de v par rapport à λ c'est-à-dire 0 si $v = 1$ et $\frac{1}{\Pi}$ dans le cas contraire et dL est la distance restante avant la fin de l'arc.

À la fin de l'arc et au début du suivant, les dérivées (G_t^T, G_r^T) de t et r par rapport à T, la date de sortie du premier arc (et la date d'entrée du suivant), peut être calculé par l'Equation 7.1.

$$\begin{aligned} G_t^T &= \left(\frac{v_1}{v_2} - 1\right) \\ G_r^T &= v_1.(v_2.\Pi_2 - v_1.\Pi_1) \end{aligned} \tag{7.1}$$

où Π_1 (resp. Π_2) la valeur de risque juste avant (resp. juste après) T et v_1 (resp. v_2) est la vitesse juste avant (resp. juste après) T en fonction de la décision λ, Π_1 et Π_2.

À partir de $(G_t^{\lambda_1}, G_r^{\lambda_1})$ du premier arc, $(G_t^{\lambda_2}, G_r^{\lambda_2})$ du second arc et (G_t^T, G_r^T), les vraies dérivées de $(G_t^{\lambda_1}, G_r^{\lambda_1})$ peuvent être calculées par l'équation 7.2.

$$\begin{aligned} G_t^{\lambda_1} &= G_t^{\lambda_2} + (G_t^{\lambda_1} + G_t^T) \\ G_r^{\lambda_1} &= G_r^{\lambda_2} + (G_r^{\lambda_1} + G_r^T) \end{aligned} \tag{7.2}$$

Enfin, la direction de descente $(\Delta^{\lambda_1}, \Delta^{\lambda_2})$ est calculée grâce à $G_t^{\lambda_1}$, $G_r^{\lambda_1}$, $G_t^{\lambda_2}$ et $G_r^{\lambda_2}$ en projetant $(G_r^{\lambda_1}, G_r^{\lambda_2})$ sur le plan orthogonal à $(G_t^{\lambda_1}, G_t^{\lambda_2})$ et en normalisant le vecteur résultant comme dans l'Equations 7.3 :

$$\begin{aligned} (\Delta^{\lambda_1}, \Delta^{\lambda_2})^* &= (G_r^{\lambda_1}, G_r^{\lambda_2}) - \frac{< (G_r^{\lambda_1}, G_r^{\lambda_2}), (G_t^{\lambda_1}, G_t^{\lambda_2}) >}{||(G_t^{\lambda_1}, G_t^{\lambda_2})||^2}.(G_t^{\lambda_1}, G_t^{\lambda_2}) \\ (\Delta^{\lambda_1}, \Delta^{\lambda_2}) &= \frac{(\Delta^{\lambda_1}, \Delta^{\lambda_2})^*}{||(\Delta^{\lambda_1}, \Delta^{\lambda_2})^*||} \end{aligned} \tag{7.3}$$

Mais maintenant que le risque a diminué, la vitesse peut être augmentée et réutiliser le risque que nous avons récupéré précédemment. Nous proposons de multiplier la décision λ_q par le quotient du risque précédent et du nouveau. Alors, si on récupère effectivement du risque, le quotient sera supérieur à 1 et égal à 1 si on ne le fait pas.

Là encore, l'amplitude des modifications sera décroissante comme dans l'algorithme 7.2. Ainsi, pour minimiser localement le risque, plusieurs itérations du processus de minimisation sont appliquées (3 par exemple) et le calcul des valeurs de $(t_q, r_q, \lambda_q, t_q)$ suivra l'algorithme 7.4.

Algorithme 7.4 : Calcul des valeurs $(t_q, r_q, \lambda_q, t_q)$

Entrées : *step* l'amplitude courante
Début
$q = 0$
Tant que $q \leq Q - 1$ **Faire**
r^{prev} le risque sur l'arc q avant toute modification
Pour 3 itérations (choix arbitraire) **Faire**
Calculer $(G_t^{\lambda_1}, G_r^{\lambda_1})$ et $(G_t^{\lambda_2}, G_r^{\lambda_2})$ sur les arcs q et $q+1$
Calculer la direction de descente $(\Delta^{\lambda_1}, \Delta^{\lambda_2})$
$\lambda_q = \lambda_q - step.\Delta^{\lambda_1}$
$\lambda_{q+1} = \lambda_{q+1} - step.\Delta^{\lambda_2}$
Calculer le nouveau risque r sur l'arc q
Fin Pour
$\lambda_q = \frac{r^{prev}}{r}\lambda_q$
Appliquer la transition avec la nouvelle valeur λ_q
Fin Tant que
Appliquer la transition sur le dernier arc selon la reformulation RD.
Fin

Section 8

Expériences numériques

But : Nous réalisons des expériences numériques dans le but d'étudier le comportement des algorithmes statiques **DP_Evaluate**, **Dec_SSPP** et **A*_SSPP** de la Section 5. Nous portons une attention particulière à la dépendance de ces algorithmes au choix du mode de décision (Risque versus Distance, Risque versus Temps, Distance versus Temps).

Contexte technique : Les algorithmes ont été implémentés en C++17 sur un processeur Intel i5-9500 à 4,1 GHz. Les temps CPU sont en millisecondes.

Table 8.1 – Tableau des paramètres des instances

id	$\|M\|$	Freq	$\overline{R}$	α	L^*	Glouton
1	4	0.2	2	0.4	59	142.32
2	5	0.25	1.9	1	55	82.47
3	6	0.19	2	1.5	63	72.72
4	4	0.43	2	0.4	67	141.15
5	6	0.6	1.9	1	61	118.89
6	7	0.42	2	1.5	68	93.34
7	6	0.16	1.9	0.4	104	317.61
8	7	0.18	1.9	1	96	142.3
9	6	0.18	2	1.5	93	102.79
10	7	0.41	2	0.4	102	194.87
11	6	0.45	2	1	104	185.33
12	8	0.32	2	1.5	101	129.06
13	11	0.16	1.9	0.4	130	251.43
14	8	0.15	2	1	126	219.55
15	7	0.19	2	1.5	142	172.05
16	6	0.33	1.9	0.4	138	351.25
17	9	0.3	1.9	1	133	199.06
18	7	0.36	2	1.5	140	175.94

Instances : Nous avons généré des réseaux (N, A) sous forme de graphe planaire aléatoire créé par une triangulation de Delaunay. Ces graphes sont résumés par leur nombre $|N|$ de nœuds et leur nombre $|A|$ d'arcs. Les valeurs de longueur $L_e, e \in A$, sont uniformément réparties entre 3 et 10. La fonction H est prise comme fonction $v \mapsto H(v) = v^2$. Les fonctions Π^e sont générées en fixant un horizon de temps T_{max}, une fréquence moyenne $Freq$ des points de rupture t_i^e et une valeur moyenne $\overline{R}$ pour la valeur $\Pi^e(t)$: Plus précisément, les valeurs Π^e sont générées dans un ensemble fini $\{2\overline{R}, \frac{3\overline{R}}{2}, \overline{R}, \frac{\overline{R}}{2}, 0\}$. Quant au seuil R_{max}, nous remarquons que si les fonctions Π^e sont constantes de valeur $\overline{R}$ et si nous suivons un chemin Γ de longueur L_{diam}, le diamètre du réseau G, à la vitesse $\frac{1}{2} = \frac{v_{max}}{2}$, alors le risque espéré est $\frac{L_{diam}\overline{R}}{2}$. Il en

résulte que nous générons R_{max} comme une quantité $\alpha\frac{L_{diam}\overline{R}}{2}$, où α est un nombre compris entre 0,2 et 2. Enfin, puisqu'une instance est aussi déterminée par l'origine/le couple (o,p), nous notons L^* la valeur $L^*_{o,p}$. **Table 8.1** présente un groupe de 18 instances avec leurs caractéristiques et la valeur de temps obtenue par l'algorithme glouton présenté dans la Section 4.3. La valeur obtenue par cet algorithme étant un paramètre de deux des trois méthodes de calcul de vitesse, nous la présentons comme un paramètre de l'instance.

8.1 Résultats liés au comportement de DP_SSPP

Nous appliquons **DP_SSPP** en testant le rôle des paramètres $\lambda = \lambda^{RD}, \lambda^{RT}, \lambda^{DT}$, ainsi que S_{max} et ρ. Ainsi, pour chaque instance, nous calculons :

- dans la **Table 8.2** : la valeur de temps T_DP^{mode}, le pourcentage de risque R_DP^{mode} de R_{max}, et les temps de calcul CPU^{mode} (en millisecondes.), induit par l'application de **DP_SSPP** sur le plus court chemin entre o et p avec $\lambda^{mode} = \lambda^{RD}, \lambda^{RT}, \lambda^{DT}$, $S_{max} = +\infty$, $G_{max} = 21$ et $\rho = 8$.
- dans la **Table 8.3** : la valeur de temps T_DP^{mode}, le pourcentage de risque R_DP^{mode} à partir de R_{max}, et les temps de calcul CPU^{mode} (en millisecondes.), induit par l'application de **DP_SSPP** sur le plus court chemin entre o et p avec $\lambda^{mode} = \lambda^{RD}, \lambda^{RT}, \lambda^{DT}$, $S_{max} = 21$, $G_{max} = 5$ et $\rho = 4$.
- dans la **Table 8.4** : pour le mode spécifique λ^{RD}, le nombre associé $\#S$ d'états par nœud i, ainsi que la valeur temporelle T^{RD}, quand $S_{max} = 3, 7, 11, 15, 21$, $G_{max} = 5$ et $\rho = 4$.
- dans la **Table 8.5** : Pour le mode spécifique λ^{RD}, le nombre moyen $\#S$ d'états par nœud i, ainsi que la valeur temporelle T^{RD}, lorsque $S_{max} = +\infty$, $G_{max} = 21$ et $\rho = 1, 5, 4$.

TABLE 8.2 – **DP_SSPP** - Impact de λ^{mode}, avec $Smax = +\infty$, $Gmax = 21$ et $\rho = 8$

id	T^{RD}	R^{RD}	T^{RT}	R^{RT}	T^{DT}	R^{DT}
1	112.3	98.3	118.5	99.9	126.2	78.9
2	66.1	96.1	68.1	99.1	68	87.8
3	63	84.1	63	84.1	72.3	49.5
4	129.4	99.8	136.6	89.7	131.7	96.3
5	72	97.2	70.2	98.2	75.3	92.7
6	78.7	99.8	77.6	98.1	76.2	99.6
7	279.1	97.3	278.1	99.5	284.6	99.2
8	116	96.3	116.2	94.3	118.5	94.9
9	93.2	98.9	93.2	98.9	93.2	98.2
10	167.4	99.3	176.7	92.6	174.7	99.8
11	123.5	92.8	123.4	93.5	123.2	92.6
12	110.3	99.8	109.6	99.7	111	99.9
13	208.1	97.4	208.6	97.9	226.5	99.1
14	148.2	98	148.1	99.7	148.9	97.6
15	157.3	96.9	161	89.7	162	89
16	254.8	99.9	259.9	90.1	251.5	95.4
17	155.7	99.5	155.1	99.9	161.1	97.4
18	146.1	99.8	147	98.5	146.5	99.5

Commentaires du tableau 8.2 :

Il est important de noter que la longueur du plus court chemin de l'instance 3 est de 63 unité de temps. Par conséquent, les solutions qui vont à pleine vitesse n'atteignent pas R_{max}. Sur cette même instance, la méthode DT a été particulièrement prudente au début du chemin puisque cette zone est très risquée, accumulant du retard qu'elle n'a pas pu rattraper même à pleine vitesse sur la fin du trajet qui n'est pas très risqué.

La méthode *RD* fonctionne toujours mieux, ou est proche de la meilleure solution, comme attendu en raison de la reformulation *RD*, voir Section 4.3.

la méthode *DT* est conçue pour minimiser ses risques, elle peut alors se terminer sans atteindre R_{max} plus souvent que les deux autres méthodes.

Sur l'instance 11, les trois méthodes se terminent sans atteindre R_{max}, nous pouvons donc en déduire que le dernier arc n'est pas risqué autour de la date d'arrivée.

TABLE 8.3 – **DP_SSPP** - Impact de λ^{mode}, avec $Smax = 21$, $Gmax = 5$ et $\rho = 4$

id	T^{RD}	R^{RD}	cpu^{RD}	T^{RT}	R^{RT}	cpu^{RT}	T^{DT}	R^{DT}	cpu^{DT}
1	132.8	99.6	0.27	140	93.9	0.28	142.3	87.8	0.53
2	72.3	99.5	1.57	78.7	96.1	0.46	79.6	88.5	1.19
3	63	84.1	1.22	63	84.1	1.14	72.6	55.1	1.04
4	137.1	91.7	0.19	140.3	96.6	0.34	159.1	91.3	0.18
5	76.2	98.2	1.1	85.6	96.6	1.34	90.2	99.4	0.98
6	81.4	99.9	1.36	78	99.6	1.72	78.2	97.8	1.99
7	280.5	98	1.21	283.3	98.4	1.01	287.5	82.4	1.12
8	120.2	99.2	1.5	121.2	96.5	1.44	125.8	99.6	1.03
9	93.2	97.4	0.88	94.6	93.5	1.06	95.7	97.2	0.99
10	187.2	94.7	1.08	188.9	97.6	1.07	194.8	95.8	1.11
11	124.9	96.2	1.08	124.5	93.8	1.14	125.1	95.9	1.74
12	111.2	99.2	2.84	115.6	98.4	3.43	116	95.9	1.5
13	212	97.6	1.74	214.4	94.4	3.2	235.3	89	1.5
14	149.2	99	1.31	153.5	97.7	1.4	156.9	93.5	1.18
15	158.2	96.3	1.04	161.2	90.6	0.87	172	96.1	0.83
16	276.2	92.1	0.87	292.4	80.7	10.35	351.2	79.4	1.01
17	161	99.8	1.61	167.9	93.5	1.61	168	95	2.65
18	148.9	98.5	1.11	148.6	98.4	1.16	153.4	99.2	1.8

Commentaires du tableau 8.3 :

Pas de surprise par rapport à l'expérience précédente : avec les mêmes paramètres mais en fixant le nombre d'états, la méthode *RD* reste la plus efficace globalement en trouvant la meilleure solution des trois méthodes ou une solution très proche.

Il y a une baisse générale de la qualité des solutions puisqu'une partie des trajets est perdu du fait du nombre fini d'états explorés.

Par contre, le schéma de programmation dynamique devient très rapide et n'est pas très loin de la solution trouvée sans limites d'états.

TABLE 8.4 – **DP_SSPP** - Impact de $Smax$, avec λ^{RD}, $Gmax = 5$ et $\rho = 4$

S_{max}	3		7		11		15		21	
id	T^{RD}	$\#S$	T^{RD}	$\#S$	T^{RD}	$\#S$	T^{RD}	$\#S$	T^{RD}	$\#S$
1	142.3	1.1	141	2.45	140	3.4	140	3.75	132.8	3.95
2	82.3	2.5	72.3	4.09	78.3	5.5	78.3	6.45	72.3	7.55
3	63	4	63	7.75	63	10.05	63	11.65	63	12.95
4	187	1.45	150.1	3.55	140.8	4.34	140.3	4.84	137.1	5.4
5	109.1	3.6	109.1	6.4	90.2	8.8	90.2	10.15	76.2	12.3
6	111.1	4.15	111.1	8.3	110.8	11.3	82.8	14.05	81.4	18.75
7	445.6	2.1	316.7	3.73	316.1	4.66	286.9	6.6	280.5	7.93
8	171.5	2.36	146.6	4.8	120.2	6.93	120.2	8.63	120.2	10.63
9	101.2	1.9	95.9	3.86	95.8	5.73	95.7	7.66	93.2	9.26
10	190.1	2.2	187.6	3.93	187.2	5.06	187.2	6.86	187.2	8.53
11	200	2.33	139.6	4.2	124.1	6.33	124.9	7.2	124.9	8.5
12	144.4	2.86	140.1	5.76	137.2	7.73	111.7	9.63	111.2	12.4
13	281.9	3.55	313.6	7.45	222.2	8.6	221.7	11.62	212	13.25
14	207.3	2.4	156.9	4.17	156.9	6	156.9	7.47	149.2	9.52
15	160.8	1.57	159.9	3.6	158.2	5.77	158.2	6.17	158.2	8.92
16	353.6	1.5	332.4	2.85	304.6	4	304	5.09	276.2	6.32
17	205.9	2.85	176.1	5.32	167.9	7.57	167.9	9.17	161	11.1
18	190.4	1.92	148.9	3.7	148.9	4.92	148.9	6.72	148.9	8.12

Commentaires du tableau 8.4 :

Les résultats montrent que, plus le nombre d'états gardé est grand, plus la solution est de qualité. Une forte amélioration pour S_{max} entre 3 et 7 est à noter, ce qui est cohérent puisque 3 états sont trop peu pour obtenir une solution de qualité.

TABLE 8.5 – **DP_SSPP** - Impact de ρ, avec λ^{RD}, $Smax = +\infty$ et $Gmax = 21$

ρ	1.5		4	
id	T^{RD}	$\#S$	T^{RD}	$\#S$
1	113.4	236.75	103.2	75.5
2	79.7	370.3	62	221.15
3	66.2	118.75	63	254.75
4	130.6	70.7	127.9	62.9
5	81.9	151.8	71.2	174.7
6	80.4	723.8	76.1	623.45
7	282.8	233.06	275.7	203.4
8	118.5	550.13	115.8	331.63
9	95.8	61.73	93.2	170.86
10	183.2	222.36	176.5	293.76
11	127.3	91.2	123.2	210.36
12	112.9	542	110.3	511.86
13	212.7	675.02	207.7	473
14	159.2	188.55	147.8	254.62
15	161.2	311.05	157.5	205.32
16	264.7	127.55	250.8	254.25
17	174.2	395.87	155.4	523.22
18	154.8	131.3	146	160.9

Commentaires du tableau 8.5 :

Lorsque les décisions sont concentrées vers la valeur λ^{RD} (c'est-à-dire $\rho = 1.5$), une dégradation de la solution est à noter. Nous pouvons en déduire que les décisions peuvent prendre des valeurs très éloignées d'un arc à l'arc, ce qui nous permet de mieux comprendre pourquoi la méthode DT fonctionne mal puisque celle-ci lisse ses décisions par la minimisation du risque local.

8.2 Résultats liés au comportement de A*_SSPP et Dec_SSPP

Nous testons la capacité de **A*_SSPP** et de **Dec_SSPP** à attraper la solution optimale et observons les caractéristiques du chemin résultant. Les instances utilisées sont présentées avec leurs paramètres dans **Table 8.6**. Nous nous appuyons sur $\lambda = \lambda^{RD}$, $S_{max} = 21$, $G_{max} = 5$ et $\rho = 4$. Pour chaque instance, nous calculons :

- dans la **Table 8.7** : La valeur de temps T_A*, le pourcentage de risque R_A* par rapport à R_{max}, le temps de calcul *cpu* (en s.), le nombre $\#T$ de nœuds visités, le nombre $\#S$ d'états générés, et l'écart *dev* entre la longueur du chemin résultant Γ et L^*, induit par **A*_SSPP** avec $\lambda^{mode} = \lambda^{RD}$, $S_{max} = 10$ et $\rho = 4$
- dans la **Table 8.8** : La valeur de temps T_Dc, le pourcentage de risque R_Dc par rapport à R_{max}, le temps de calcul *cpu* (en s.), le nombre $\#T$ d'essais, le nombre $\#S$ des états générés, et l'écart *dev* entre la longueur du chemin résultant Γ et L^*, qui dérive de l'application de **Dec_SSPP** avec $\lambda^{mode} = \lambda^{RD}$, $S_{max} = 10$ et $\rho = 4$.

Table 8.6 – Tableau des paramètres globaux des instances

id	$\|N\|$	$\|M\|$	Freq	$\overline{R}$	α	#SP	Glouton
1	20	87	0.2	2	0.4	59	142.3
2	20	87	0.25	1.9	1	55	82.4
3	20	87	0.19	2	1.5	63	72.7
4	20	87	0.43	2	0.4	67	141.1
5	20	90	0.6	1.9	1	61	118.8
6	20	90	0.42	2	1.5	68	93.3
7	30	147	0.16	1.9	0.4	104	317.6
8	30	138	0.18	1.9	1	96	142.3
9	30	144	0.18	2	1.5	93	102.7
10	30	153	0.41	2	0.4	102	194.8
11	30	147	0.45	2	1	104	185.3
12	30	150	0.32	2	1.5	101	129
13	40	201	0.16	1.9	0.4	130	251.4
14	40	204	0.15	2	1	126	219.5
15	40	204	0.19	2	1.5	142	172
16	40	201	0.33	1.9	0.4	138	351.2
17	40	201	0.3	1.9	1	133	199
18	40	204	0.36	2	1.5	140	175.9

TABLE 8.7 – **A*_SSPP** - λ^{RD}, $Smax = 21$, $Gmax = 5$ and $\rho = 4$

id	T_A*	R_A*	$\#T$	cpu	$\#S$	dev
1	102.9	98.5	19	7.86	52.2	16.9
2	67.5	97.1	18	3.8	27.25	0
3	63	84.1	19	4.5	42.25	0
4	117.6	99	19	6.19	40.45	0
5	66.7	79.9	19	9.95	62.15	0
6	69.1	99.8	19	5.89	44.2	0
7	144.8	99.2	29	9.15	48.4	3.84
8	104.7	99	29	10.97	56.66	2.08
9	95	88.2	28	9.16	53	2.15
10	181.1	98.8	29	8.29	35.5	0
11	118.9	96.3	29	13.56	57.3	1.92
12	108.3	98	29	17.62	69.93	1.98
13	180.5	75.4	37	15.16	57.3	9.23
14	150.7	99.2	39	21.13	70.45	3.17
15	143.4	97.5	35	19.91	60.17	0
16	186.4	99.4	35	14.11	42.72	0
17	146.5	96.9	35	17.56	61.72	1.5
18	143.6	98.3	39	25.04	76	0

TABLE 8.8 – **Dec_SSPP** - λ^{RD}, $Smax = 21$, $Gmax = 5$ and $\rho = 4$

id	T_Dc	R_Dc	$\#T$	cpu	$\#S$	dev
1	118	97.9	11	5.96	5.35	6.77
2	67.4	97.1	5	2.83	6.25	0
3	63	84.1	6	5.24	8.19	0
4	140.3	96.6	3	1.67	4.7	0
5	69.8	92	20	26.76	12.25	0
6	70.8	95.5	14	14.83	10.6	0
7	159.9	79.9	27	27.47	10.3	11.5
8	119.5	92.4	13	13.59	6.13	4.16
9	95	88.2	9	9.71	8.46	2.15
10	187.2	94.7	8	9.86	8.03	0
11	124.9	96.2	7	7.03	7.43	0
12	114.1	95.1	15	22.21	10.73	5.94
13	187.2	97.8	53	114.4	11.12	1.53
14	151.1	99.6	15	21.46	7.42	3.17
15	143.7	97.9	22	32.62	6.92	0
16	252.2	92.5	5	5.05	3.75	0
17	155.4	89.7	16	26.01	8.15	0
18	148.9	98.5	7	8.24	6.47	0

Commentaires des tableaux 8.7 et 8.8 :

Nous pouvons voir que **A*_SSPP** et **Dec_SSPP** sont proches mais **Dec_SSPP** est en retard dans la plupart des cas. Ceci est dû au fait que **Dec_SSPP** explore plus de chemins mais génère moins d'états et obtient donc de moins bons résultats en utilisant le même chemin que **A*_SSPP**.

8.3 Résultats liés aux caractéristiques des solutions

Encore une fois, nous nous appuyons sur **DP_Evaluate**, et nous cherchons des informations sur l'ensemble de décision Γ. Nous utilisons $\lambda = \lambda^{RD}$, $S_{max} = 21$, $G_{max} = 3$ et $\rho = 4$. Nous considérons les instances dans leur ensemble et calculons :

- dans la **Table 8.9** : Pour chaque valeur de λ, le pourcentage de décisions optimales liées à λ dans les séquences de décisions optimales calculées par **DP_Evaluate**
- dans la **Table 8.10** : Le rang moyen mR et le rang maximum MR des états (T, R) impliqués dans ces trajectoires optimales, dans les ensembles d'états $STATE[i], i = 1, \ldots, n$, selon l'ordre induit par les valeurs croissantes de T, estimé comme dans le tableau 8.3. mS et MS font respectivement référence à la taille moyenne de l'ensemble d'états $State[i], i = 1, \ldots, n$, et sa taille maximale.

TABLE 8.9 – Répartition des décisions optimales - λ^{RD}, $Smax = +\infty$, $Gmax = 21$ et $\rho = 8$

id_λ	1	2	3	4	5	6	7
%	16.6	0.4	1.2	1.2	1.2	2.4	1.2
id_λ	8	9	10	11	12	13	14
%	1.4	0.9	1.6	7.4	15.2	11.3	7.9
id_λ	15	16	17	18	19	20	21
%	4.8	3.3	3.3	2.4	1.9	2.4	11

TABLE 8.10 – Ensemble de décision - λ^{RD}, $Smax = 21$, $Gmax = 5$ et $\rho = 4$

id	mR	MR	mS	MS
1	3	9	19	35
2	2.8	6	21.8	36
3	0	0	21.66	40
4	3.75	9	18.75	34
5	2.83	7	22.5	33
6	5.85	10	28.28	58
7	1.33	3	20.83	38
8	6.28	12	17.57	25
9	2.16	3	28	49
10	4.85	23	30.28	74
11	3.66	9	27.16	48
12	2.37	9	24.5	51
13	6.09	21	26.36	47
14	4.62	19	18.62	34
15	0	0	25	45
16	6	18	24.16	45
17	6.44	16	25.44	39
18	3.42	8	26.42	39

Commentaires du tableau 8.9 et 8.10 :

Nous pouvons voir que les décisions optimales sont souvent générées grâce aux valeurs moyennes de λ (de la $9^{\text{ème}}$ à la $15^{\text{ème}}$ valeur) et, dans certains scénarios, les valeurs d'extrême gauche et d'extrême droite sont utilisées. Ces résultats sont en cohérence avec les résultats observés lorsque la valeur de ρ est diminuée.

8.4 Résultats liés au processus d'apprentissage statistique

À présent, nous allons utiliser **DP_Evaluate** avec le processus d'apprentissage de la Section 7 en testant le rôle du paramètre $\lambda = \lambda^{RD}, \lambda^{RT}, \lambda^{DT}$. Donc, pour chaque instance, nous calculons dans la **Table 8.11** : la valeur de temps T_DP^{mode}, la valeur de risque R_DP^{mode}, et les temps de calcul CPU^{mode} (en millisecondes.), induits par application de **DP_SSPP** sur le plus court chemin en termes de distance entre o et p avec $\lambda^{mode} = \lambda^{RD}, \lambda^{RT}, \lambda^{DT}$, $S_{max} = 9$, $G_{max} = 3$ et $\rho = 4$.

TABLE 8.11 – **DP_SSPP** avec apprentissage - Impact de λ^{mode}, avec $Smax = 9$, $Gmax = 3$ et $\rho = 4$

id	T^{RD}	R^{RD}	cpu^{RD}	T^{RT}	R^{RT}	cpu^{RT}	T^{DT}	R^{DT}	cpu^{DT}
1	104.6	98.5	0.22	118.6	98.4	0.19	142.3	87.8	0.36
2	75	97.9	0.29	96.4	99.8	0.41	71.6	98.2	0.68
3	63	84.1	0.62	63	84.1	0.68	0	0	0
4	130.4	98.6	0.26	138.2	99.8	0.25	195.7	69.7	0.24
5	86.4	86.7	0.4	81.6	98.4	0.54	81.5	99.6	0.96
6	81.5	99.9	0.59	0	0	0	82.1	97.5	0.98
7	280.5	99.7	0.37	284.7	98.8	0.4	284.6	88.6	0.63
8	126.6	99.5	0.48	130.6	98.2	0.36	121.4	99.7	0.85
9	93.2	97.7	0.75	93.2	97.6	0.42	96	82.5	0.77
10	181.5	99	0.63	186.4	99.3	0.55	190.4	97.8	0.83
11	126.5	96.2	0.54	127.1	99.8	0.53	121.5	97.6	2.1
12	136.8	76.1	0.57	0	0	0	138.4	99.3	1.08
13	218.9	99.7	0.82	214	95.8	0.81	249.8	99.2	0.98
14	154.1	99.1	0.57	155.7	91.4	0.55	158.1	87.3	0.82
15	158.6	97.8	0.5	165.4	87.8	0.57	161.5	96.3	0.62
16	294.9	95.1	0.4	332.6	87.5	0.84	346.4	95.3	0.38
17	162.4	98.9	3.47	163.8	98.5	0.7	172	91.4	1.91
18	149.1	98.9	0.48	148.9	99.7	0.46	151.2	99.9	0.86

Commentaires du tableau 8.11 :

Du fait du petit nombre d'états générés (G_{max}) et conservés (S_{max}), la méthode est la plus rapide de toutes et elle fonctionne aussi bien qu'avec $S_{max} = 11$ et $G_{max} = 5$ (voir tableau 8.4). L'apprentissage permet donc de récupérer des décisions et états qui n'auraient pas été gardés autrement et améliore globalement la méthode.

8.5 Résultats liés aux algorithmes gloutons améliorés

Enfin, nous allons comparer les différentes approches basées sur l'algorithme glouton : la recherche locale et l'apprentissage par renforcement. Pour chaque instance, nous calculons dans la **Table 8.12** : la valeur de temps T^{alg}, la valeur de risque R^{alg}, et les temps de calcul CPU^{alg} (en millisecondes.), induits par application de **RechLoc_SSPP** et **AGlouton_SSPP** sur le plus court chemin entre o et p en comparaison avec l'algorithme glouton. *Rl* désigne la recherche locale, *Ar* désigne l'apprentissage par renforcement et *Gl* désigne l'algorithme glouton.

TABLE 8.12 – Méthodes gloutonnes améliorés - Recherche locale Rl et Apprentissage par renforcement Ar comparés à l'algorithme glouton sans amélioration Gl.

id	T^{Rl}	R^{Rl}	cpu^{Rl}	T^{Ar}	R^{Ar}	cpu^{Ar}	T^{Gl}
1	141.7	67	0.5	142.3	76.1	0.31	142.3
2	81.6	84.3	0.63	78.2	100	0.36	82.4
3	68.9	61.4	0.87	63	84.1	0.43	72.7
4	140.3	88.1	0.48	141.1	95	0.32	141.1
5	118.7	94.8	0.84	107.2	85	0.7	118.8
6	86.9	82.4	0.87	84.5	97.7	0.58	93.3
7	294.8	62.7	0.83	317.6	94.9	1.67	317.6
8	142.2	99.9	0.83	137.4	98	0.56	142.3
9	102.6	83.2	0.64	95.8	96.7	0.53	102.7
10	194.6	84.6	0.85	194.8	84.6	0.71	194.8
11	147.1	66.3	0.74	141.9	80.8	0.57	185.3
12	129	88	1.05	128.7	97.3	0.68	129
13	248.1	100	1.48	251.4	100	0.77	251.4
14	158.6	67.3	0.97	162.7	69.9	0.61	219.5
15	172	75.2	0.78	157.9	96.6	0.52	172
16	293.8	71.7	0.77	305.5	83.1	0.48	351.2
17	191.4	93.4	1.08	199	92.6	0.74	199
18	166	86.4	0.8	151.5	97.4	0.6	175.9

Commentaires du tableau 8.12 :

En fonction de la fréquence des changements de la fonction de risque de chaque instance, certaines méthodes permettent une forte amélioration, pour les instances avec la plus grande fréquence, aucune amélioration n'est visible. En effet, une amélioration locale modifie trop la date de la décision suivante qui ne correspond alors plus avec la quantité de risque à cette date. Nous pouvons remarquer une légère amélioration par rapport à l'algorithme glouton simple, mais les résultats ne sont pas à la hauteur des algorithmes **A*_SSPP** et **Dec_SSPP**. Ils peuvent néanmoins être utilisés pour améliorer la borne supérieure lors de l'utilisation de ces deux derniers algorithmes.

Section 9

Conclusion

Dans ce chapitre, nous avons étudié un problème de plus court chemin avec des contraintes de risque dépendantes du temps. Nous avons construit des méthodes de résolution sous l'angle d'exigences de calcul rapides.

Après avoir étudié le risque induit par l'activité d'une flotte de véhicules autonomes, nous nous sommes concentrés sur l'impact de l'ajout d'un nouveau véhicule autonome dans une flotte préexistante. Ce nouveau véhicule doit parcourir l'entrepôt pour réaliser une tâche non prise en charge par la flotte actuelle. L'objectif est donc de trouver un chemin daté vers sa destination et une fonction vitesse à tout instant, en prenant en compte le planning et le trajet des véhicules de la flotte et le risque qu'ils induisent.

Après avoir défini le risque que prend un véhicule en fonction de sa vitesse, nous avons reformulé le problème grâce à des résultats structuraux, notamment sur le risque pris par un véhicule allant à une certaine vitesse. De cette reformulation, nous avons pu construire trois méthodes de calcul de la vitesse sur un arc donné, que nous avons nommées *Risque versus Distance*, *Risque versus Temps* et *Distance versus Temps*.

Ces calculs de vitesse ont été intégrés à deux schémas de résolution de notre problème de Plus Court Chemin Sécurisé :

- Le premier est une méthode itérative qui découple la résolution en une modification locale de chemin puis une recherche de vitesses grâce à un schéma de programmation dynamique.
- Le deuxième est basé sur l'algorithme A* où la construction du chemin se fait en parallèle du calcul de la vitesse.

À ces méthodes, ont été ajouté des processus pour limiter le nombre d'états qui font perdre les garanties d'optimalités mais qui permettent de satisfaire l'exigence de calcul rapide.

Pour finir, trois méthodes simples ont été proposées pour améliorer une solution existante :

- La première tente d'améliorer la fonction de vitesse en regardant les intersections du chemin. Si l'arc suivant est moins risqué, il peut être mieux d'accélérer pour profiter de cette zone plus sécurisée à cette même date.
- La deuxième considère l'ensemble des décisions sur un chemin comme étant une direction et tente de l'améliorer en évaluant l'impact d'une modification aléatoire.
- La troisième se base sur l'idée qu'une solution globale est aussi une solution locale. Les arcs sont alors pris deux par deux et une descente de gradient est appliquée. Puis les vitesses sur ces deux arcs sont augmentées pour récupérer la fraction de risque qui a été libéré par la minimisation.

Par la suite, ces méthodes pourront être reprise en partie pour résoudre le problème de dimensionnement d'une flotte de véhicule autonome en fonction de la configuration d'un entrepôt.

Chapitre II

Un Problème de bi-flot préemptif

En autorisant la préemption des marchandises et la dépendance au temps du réseau de transport, les problèmes ressemblant au *Pickup and Delivery Problem* deviennent très complexes et peu étudiés. Pourtant, ces problèmes sont très souvent rencontrés par les acteurs du monde socio-économiques (relocalisation des vélos en libre-service d'une métropole par exemple). Dans ce contexte, la résolution du problème doit être faite un nombre limité de fois par jour, pour autant, la taille des instances empêche la résolution directe du problème.

Nous proposons une simplification du problème en le projetant hors de la dimension temporelle avant de résoudre ce sous-problème et de récupérer le temps grâce à deux heuristiques : l'une basée sur des méthodes de simulations et l'autre sur des heuristiques de flots dans un graphe adapté.

Table des matières

Liste des notations

Graphe statique :

$G = (N, E)$	Un graphe de transit statique
$e = (u, v)$	Un arc entre les noeud u et v
L_e	Le temps de parcours de l'arc e
C_e^x	Le coût de l'arc e pour les véhicules
C_e^y	Le coût de l'arc e pour les marchandises
x	Le flot de véhicules statique
y	Le flot de marchandises statique
q^d	Un vecteur de triplés (origine, destination, quantité) dans le cas distinct
q^i	Un vecteur de demandes et de stocks dans le cas indistinct
CAP	La capacité des véhicules

Graphe dépendant du temps :

$G^T = (N^T, E^T)$	Un graphe de transit dépendant du temps
$u^T = (u, t) \in N^T$	Un nœud de N^T est formé d'un nœud u du graphe statique et d'une date t
$e^T = (u^T, v^T) \in E^T$	Un arc entre les noeud u^T et v^T
L_{e^T}	Le temps de parcours de l'arc e^T (peut être une fonction dépendante du temps)
T_{max}	L'horizon de temps
X	Le flot de véhicules dépendant du temps
Y	Le flot de marchandises dépendant du temps
s	La source des flots
p	Le puits des flots

Notations complémentaires :

S	Un sous-ensemble de nœuds de N
S^c	l'ensemble complémentaire de S
$\delta^+(S)$	L'ensemble des arcs sortants de S
$\delta^-(S)$	L'ensemble des arcs entrants en S
$\overline{\delta}(S)$	L'ensemble des arcs ayant au moins un noeud dans S
$\delta^*(S)$	L'ensemble des arcs interne à S (allant de S dans S

Section 10

Introduction

Les systèmes de transport font partie des éléments les plus complexes et importants de la société moderne, facilitant la circulation des personnes et des biens essentiels au développement économique et à l'amélioration de la qualité de vie globale. Cependant, à mesure que l'urbanisation continue de s'accélérer, le secteur des transports est confronté à un éventail croissant de défis, notamment dans le contexte de la circulation des ressources qui nous intéresse dans ce chapitre. Depuis de nombreuses années (voir [Schrijver, 2002]), les problèmes de transport de marchandises n'ont fait que se diversifier et s'intensifier tant au niveau des applications (voir [Li et al., 2013]) qu'au niveau de la recherche. Parmi les questions les plus étudiées se posent celles de la congestion, de la gestion du trafic localement, de l'optimisation des itinéraires et, plus récemment, des mobilités durables. Dans ce dernier contexte, les contraintes de transport de marchandises sont restées principalement inchangées mais l'objectif a connu un changement notable. Il n'est plus question de livrer le plus vite possible ou au moindre coût économique mais plutôt de se poser la question du nombre de véhicules strictement nécessaires et de considérer les coûts écologiques (carburant, pollution, etc.).

Ce changement de paradigme dans le domaine des transports met en évidence le besoin croissant de méthodologies innovantes pour répondre à des problèmes dont la dépendance au temps n'est plus secondaire, en particulier lorsque le transport autorise la préemption des marchandises (voir [Humagain et al., 2020]). La préemption permet l'interruption ou la réaffectation des ressources, pour répondre à des tâches plus prioritaires par exemple, un concept qui s'est révélé très prometteur pour améliorer la réactivité et l'adaptabilité des réseaux de transport. La réaffectation de marchandises implique qu'un second véhicule doit récupérer tout ou partie du chargement d'un premier et un mécanisme de synchronisation, ou de précédence au minimum, doit être mis en place.

Les approches traditionnelles d'optimisation des transports ne parviennent souvent pas à prendre en compte cette contrainte pour de multiples raisons (taille des modèles, temps de calcul, etc.). C'est notamment le cas lors de l'utilisation de réseaux *Time-Expanded* (voir [Krumke et al., 2014] pour un

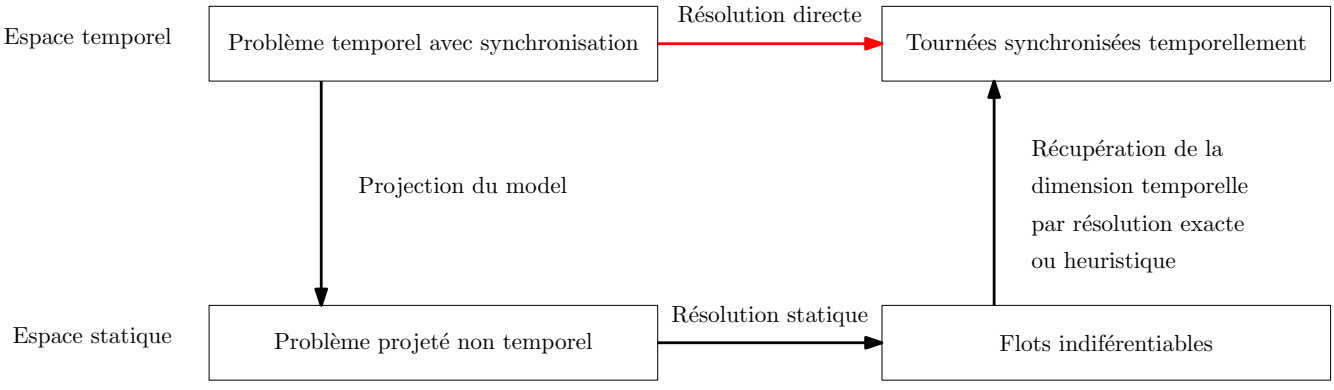

Figure 10.1 – Résolution en passant par une projection statique.

exemple). Dans ce type de réseaux, un arc entre deux nœuds du réseau *Time-Expanded* représente le franchissement d'un arc à un instant donné. Les réseaux *Time-Expanded* ont notamment été utilisés pour des problèmes d'évacuations (voir [Park et al., 2009]) où certains arcs sont franchissables à certains moments et infranchissable à d'autre, l'objectif étant de faire traverser un flux de personnes à travers le réseau. La dimension temporelle étant intrinsèque, les contraintes de précédences ou de synchronisation sont très facilement mises en place. L'inconvénient majeur est la taille d'un tel réseau, car un arc doit être copié pour toutes les dates auxquelles il peut être franchi. Si la dimension temporelle n'est pas discrétisée, le problème devient encore plus difficile.

Puisque la résolution directe n'est pas envisageable, nous proposons de supprimer la dimension temporelle dans un premier temps. Le problème de la synchronisation sera donc éludé pour résoudre en priorité l'acheminement des marchandises. Puis, la dimension temporelle est récupérée lors de la construction de tournées répondant au problème initial (voir Figure 10.1).

Ce chapitre débute avec l'introduction du problème temporelle ainsi qu'une modélisation en programme linéaire. Nous présenterons ensuite la version projetée du problème et les nouvelles contraintes inhérentes aux réseaux de transport statiques. Puis, nous présenterons un nouveau modèle statique dans l'idée de palier les défauts du modèle projeté avec une discussion sur l'efficacité d'un tel modèle. Dans un quatrième temps, nous verrons comment récupérer la dimension temporelle de manière exacte à partir d'une solution projetée. Nous présenterons ensuite deux heuristiques pour remplacer ce dernier algorithme. Enfin, nous présenterons des résultats numériques et comparerons les différentes méthodes pour la résolution du problème temporel.

Section 11

Un modèle *Time-Expanded* pour un problème de relocation préemptif

11.1 Le problème de relocation préemptif de marchandises

Le problème que nous avons étudié se base sur un cas de redistribution de véhicules partagés hétérogènes dans une agglomération. Des stations de dépose sont installées à plusieurs endroits de la zone couverte et les usagers louent un véhicule (vélos ou trottinettes par exemple) d'une station à une autre, payant au prorata du temps d'utilisation. Ces utilisateurs font des trajets en partie prévisibles : depuis les zones résidentielles vers les zones commerciales et industrielles le matin et le trajet inverse le soir. Mais les stations doivent également accommoder tout autre déplacement non prévisible. Un prestataire, pour maximiser son attractivité et son profit, doit permettre l'emprunt du plus grand nombre de véhicules possible depuis n'importe quelles stations vers n'importe quelles stations, cela nécessite donc de réapprovisionner et de libérer de la place lorsque ces stations sont encombrées.

Ces surplus de véhicules peuvent survenir à tout instant, d'autant que les usagers ne réservent pas en avance leur trajet. Un tel contexte en ligne ajoute un degré de complexité et, par mesure de simplification, nous considérons que les besoins de relocalisation sont pris à un instant donné. Nous considérons donc les surplus et besoins de véhicules qu'à un certain moment, peu importe les changements de stocks liés à l'utilisation des véhicules partagés pendant le transport des surplus.

Dans l'objectif de raccourcir les trajets et tournées des véhicules de transport, la préemption des marchandises (véhicules partagés à relocaliser) est autorisée. Il est donc possible qu'un camion de transport charge un certain nombre de marchandises, parcourt une partie du chemin de cette cargaison puis dépose, à une station intermédiaire ou directement dans un autre camion de transport, tout ou partie de sa cargaison. Dans ce cas, il faut qu'un premier véhicule passe avant ou au même moment qu'un deuxième véhicule.

De plus, afin de rester le plus général possible, nous présenterons tout au long de ce chapitre le cas de marchandises distinctes à relocaliser depuis plusieurs origines vers plusieurs destinations. Ce cas général peut être spécialisé de deux manières et se rapprocher de problèmes bien connus :

- Le cas de marchandises distinctes ne possédant qu'une seule origine et destination, à la manière d'un problème *Dial a Ride* [Ho et al., 2018, Li et al., 2013].
- Le cas de marchandises indistinctes (un seul type de marchandises) possédant plusieurs origines et destinations, à la manière d'un problème de *Balanced Transportation* [Amaliah et al., 2022, Vignaux and Michalewicz, 1991].

Le problème est donc de transporter toutes les marchandises (depuis les surplus vers les zones en déficit) dans un horizon de temps fixe. L'objectif étant de minimiser un coût économique mélangeant le coût des trajets des véhicules, le coût d'indisponibilité des marchandises en transport et un coût unitaire pour chaque véhicule utilisé (avec α_1, α_2 et α_3 les coefficients de pondération). En fonction de cette pondération, l'accent peut être mis sur l'optimisation des trajets ou sur la minimisation du nombre total de véhicules nécessaires. L'idée est que l'ajout d'un véhicule à la flotte ne doit être fait que si l'impact sur les tournées est significatif.

Ainsi, un certain nombre de véhicules homogènes de capacité CAP parcourent un réseau de transport planaire $G = (N, E)$ pour relocaliser K types de marchandises dans un horizon de temps $T_{max} > 0$. Les véhicules de transport partent tous d'un nœud dépôt $u_d \in N$ et coûtent C^V unitairement dès qu'ils sont utilisés. Ils parcourent les arcs dont la durée de traversée est donnée par le vecteur $L = \{L_e, e \in E\}$. Aussi noté $L_{u,v}$ avec $e = (u, v)$, L_e peut être fixe ou bien être une fonction dépendante du temps $L_e : t \mapsto L_e(t)$. Les véhicules transportent donc les marchandises à travers le réseau de transit en payant un coût de passage donné par le vecteur $C^x = \{C^x_e \geq 0, e \in E\}$ et les marchandises paient un coût d'indisponibilité donné par le vecteur $C^y = \{C^y_e \geq 0, e \in E\}$. Les origines et destinations des marchandises de type $k = 0, \ldots, K$ sont données sous forme d'un vecteur de quantités $\left(q^k_u\right)_u \in N$ définies sur les nœuds $u \in N$. Ce vecteur est tel que $\sum_{u \in N} q^k_u = 0$ où $q^k_u > 0$ si u a un surplus et $q^k_u < 0$ si u a un besoin de marchandises k. Dans la suite de ce chapitre, ce problème est nommé le **problème de relocation préemptif (PRP : *Preemptive Relocation Problem*)**.

De plus, nous définissons quelques notations qui nous aideront tout au long de ce chapitre :

- S un ensemble de nœuds quelconque ($S \subset N$)
- S^c l'ensemble complémentaire de S ($S^c = N \backslash S$)
- $\delta^+(S) \subset E$ l'ensemble des arcs sortants de S
- $\delta^-(S) \subset E$ l'ensemble des arcs entrant en S
- $\bar{\delta}(S) \subset E$ l'ensemble des arcs ayant au moins un nœud dans S
- $\delta^*(S) \subset E$ l'ensemble des arcs interne à S (allant de S dans S)
- $l_e = \frac{L_e}{T_{max}}$

On a donc que $\bar{\delta}(S) \backslash \delta^*(S) = \delta^+(S) \cup \delta^-(S)$.
Aussi, pour tout nœud u, nous confondrons les notations $\delta(u)$ avec $\delta(\{u\})$ pour améliorer la lisibilité.

11.2 Un graphe *Time-Expanded*

Un graphe *Time-Expanded* $G^T = (N^T, E^T)$ est une construction qui permet de représenter un graphe à tout instant d'un horizon de temps. Un nœud d'un graphe *Time-Expanded* est donc une paire $(u, t) \in N^T$ avec $u \in N$ et $t \in [0, T_{max}]$. Il existe alors deux types d'arcs dans l'ensemble E^T : ceux représentant le déplacement sur un arc de E, appelés arcs transverses, et ceux représentant le fait de ne pas se déplacer, appelés arcs horizontaux.

Dans le premier cas, la traversée d'un arc $(u, v) \in E$ se traduit par des arcs dans E^T de la forme $((u, t), (v, t+L_{u,v}(t)))$ tant que l'horizon de temps est vérifié. Dans le deuxième cas, le fait de rester immobile sur un nœud $u \in N$ se traduit par des arcs dans E^T de la forme $((u, t_1), (u, t_2))$ avec $0 \leq t_1 < t_2 \leq T_{max}$.

Les graphes *Time-Expanded* tels que présenté ci-dessus permettent de représenter toutes dates et sont donc très précis. Ils modélisent, de ce fait, un grand nombre de problèmes dépendant du temps, mais sont très complexes d'utilisation. Puisqu'il y a une infinité de valeurs temporelles possible, il y a une infinité de variables dans un programme linéaire équivalent. Il n'est donc possible de le résoudre qu'avec des méthodes très couteuses pour la résolution, comme la génération de colonnes.

Une façon de modéliser plus simplement les graphes *Time-Expanded* est de discrétiser les valeurs temporelles possibles. Le programme linéaire équivalent possède alors un nombre fini de variables et peut être mis en place plus simplement. Cette méthode n'en reste pas moins une simplification et possède un coût sur la précision des solutions, croissant avec la taille de la discrétisation. En effet, un simple exemple permet de mettre en évidence des erreurs qui s'accumulent et qui peuvent, à la fin, provoquer un écart important par rapport à la solution optimale. Prenons, par exemple, un pas de temps de 2 minutes pour notre discrétisation. la traversée d'un certain arc prend 4 minutes et 50 secondes : discrétisé, la traversée doit-elle être considérée comme faisant 2 ou 3 unités de temps ? Dans les deux cas, un écart d'une demi-unité est observé et, dans le cas d'une unité de 2 minutes, ce demi-écart répété sur chaque arc du trajet peut générer une erreur de l'ordre de 10 minutes et empêcher une synchronisation réelle des

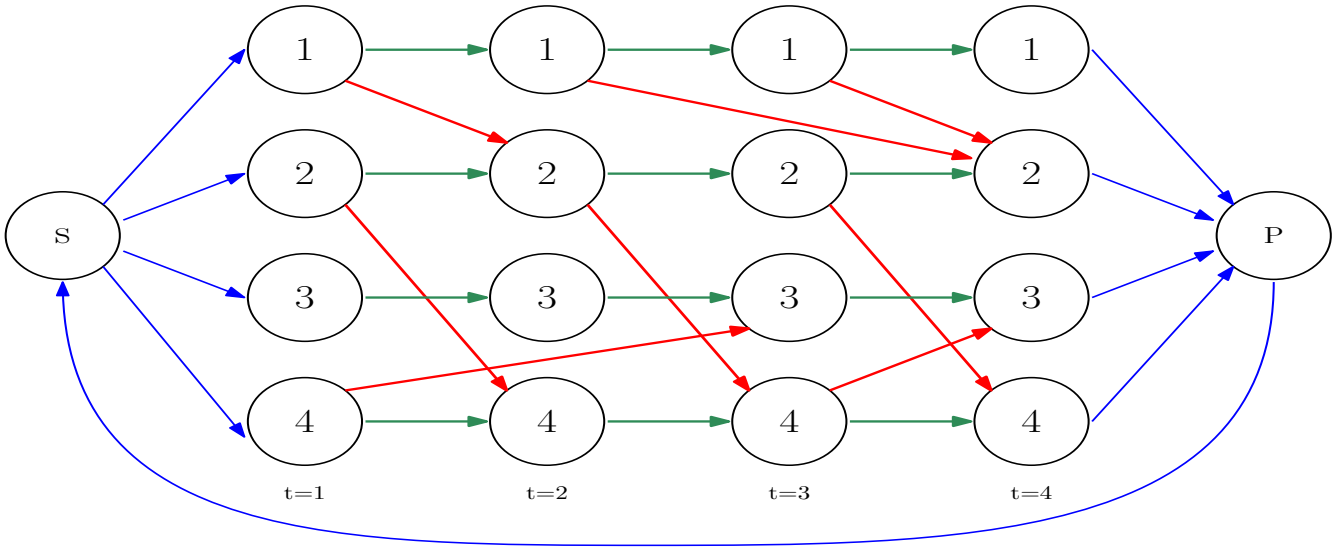

FIGURE 11.1 – Un exemple de réseau *Time-Expanded* avec $T_{max} = 4$ unités de temps.

véhicules de transport lors de leur tournée. Bien sûr, il est possible de diminuer la taille des unités de temps, mais cela augmente en proportion le nombre de variables du programme linéaire équivalent.

Puisque nous ne voulons qu'une base de comparaison pour les méthodes présentées dans les sections suivantes et non une réelle valeur optimale, nous avons utilisé un graphe *Time-Expanded* discrétisé. Les données de coûts (C^x, C^y et C^V) et de longueur d'arcs (L) sont aussi nécessaires à la création de ce graphe *Time-Expanded* pour modéliser l'objectif du problème PRP. Nous ajoutons, pour finir, deux nœuds s et p qui permettent de compléter les lois de Kirchhoff et de compter le nombre de véhicules de transport utilisés au total. La Figure 11.1 permet de visualiser un exemple d'un graphe *Time-Expanded* où $T_{max} = 4$ unités de temps où l'ensemble des nœuds N^T et l'ensemble des arcs E^T sont tels que :

- Un nœud de N^T peut être une source s, un puits p ou une paire $(u,t), u \in N, 0 \leq t \leq T_{max}$.
- Un arc de E^T est : (les couleurs sont à voir sur la figure)

un arc d'entrée	$e = (s,(u,0))$	où $C_e^X = 0$	et $C_e^Y = 0$	en bleu
un arc de sortie	$e = ((u,T_{max}),p)$	où $C_e^X = 0$	et $C_e^Y = 0$	en bleu
un arc retour	$e = (p,s)$	où $C_e^X = \alpha_1 C^V$	et $C_e^Y = 0$	en bleu
un arc horizontal	$e = ((u,t),(u,t+1))$	où $C_e^X = 0$	et $C_e^Y = 0$	en vert
un arc transverse	$e = ((u,t),(v,t+L_{u,v}(t)))$	où $C_e^X = \alpha_2 C_e^x$	et $C_e^Y = \alpha_3 C_e^y$	en rouge

avec α_1, α_2 et α_3, les coefficients de pondération du problème PRP.

11.3 Un modèle linéaire bi-flots : le *Time-Expanded 2 Flows Problem*

Nous proposons de décomposer la solution en deux flots : un flot de véhicules de transport et un flot marchandises. Le flot de véhicules de transport est porté par un vecteur d'entiers positifs $X = (X_e)_{e \in E}$ et le flot de marchandises est porté par un vecteur d'entiers positifs $Y = \sum_{k=0}^{K} Y^k$. Ce dernier se décompose en plusieurs vecteurs de rationnels positifs $Y^k = (Y_e^k)_{e \in E}$ qui représentent le flot des différents types de marchandises $k = 0, \ldots, K$. Il est à noter que, si les surplus, besoins et capacités des camions sont entiers, le flot Y le sera aussi. Ces deux flots X et Y sont définis, pour autoriser la préemption, sur un graphe *Time-Expanded* $G^T = (N^T, E^T)$ basé sur un réseau de transport initial $G = (N, E)$ tel que présenté ci-dessus. Nous allons maintenant présenter un modèle linéaire du problème PRP, en commençant par la fonction objectif :

$$\text{minimiser} \sum_{e \in E} C_e^X X_e + C_e^Y Y_e \qquad \text{(T.*)}$$

Lois de Kirchhoff
Pour tout nœud $u \in N^T$

$$\sum_{e \in \delta^+(u)} X_e - \sum_{e \in \delta^-(u)} X_e = 0 \tag{T.1}$$

$$\sum_{e \in \delta^+(u)} Y_e^k - \sum_{e \in \delta^-(u)} Y_e^k = 0, \ \forall k \tag{T.2}$$

Contraintes liées aux demandes
En début de journée, les sommets avec un surplus de marchandises sont fournis par le nœud source : pour tout arc d'entrée $e = (s, (u, 0))$

$$X_e = 0, \forall e = (s, (u, 0)) \in E^T \text{ tel que } u \neq u_d \tag{T.3}$$

Cette contrainte signifie qu'aucun camion ne peut partir d'un autre nœud que le dépôt.

$$Y_e^k = \max(0, q_u^k), \forall e = (s, (u, 0)) \in E^T \tag{T.4}$$

En fin de journée, les sommets demandeurs doivent fournir au puits les produits nouvellement arrivés : pour tout arc de sortie $e = ((u, T_{max}), p)$

$$X_e = 0, \forall e = ((u, T_{max}), p) \in E^T \text{ tel que } u \neq u_d \tag{T.5}$$

Cette contrainte signifie qu'aucun camion ne peut terminer à aucun nœud autre que le dépôt.

$$Y_e = \max(-q_u, 0), \forall e = ((u, T_{max}), p) \in E^T \tag{T.6}$$

Contraintes liants véhicules et marchandises
Si les marchandises sont transportées à travers un arc, elles doivent être transportées par suffisamment de véhicules. Pour tout arc $e = ((u_1, t_1), (u_2, t_2))$ tel que $u_1 \neq u_2$ et $t_2 = t_1 + L_e(t_1)$:

$$CAP.X_e \geq Y_e \tag{T.7}$$

Et c'est tout ! L'utilisation d'un graphe *Time-Expanded* présente de nombreux avantages car il ne contient pas de cycles et presque toutes les contraintes sont utilisées lors de la création du graphe (et n'apparaissent donc pas dans le programme linéaire).

Nous nommons *Time-Expanded 2 Flows Problem (TE2FP)* le modèle linéaire du problème PRP :

(TE2FP)

$$\begin{aligned}
\textbf{minimiser} \quad & \sum_{e \in E} C_e^X X_e + C_e^Y Y_e & & & (T.*)\\
\textbf{tel que} \quad & \sum_{e \in \delta^+(u)} X_e - \sum_{e \in \delta^-(u)} X_e = 0 & & & (T.1)\\
& \sum_{e \in \delta^+(u)} Y_e^k - \sum_{e \in \delta^-(u)} Y_e^k = 0 & & \forall k = 0, \dots, K & (T.2)\\
& X_e = 0, & & \forall e = (s, (u, 0)) \in E^T, u \neq u_d & (T.3)\\
& Y_e^k = \max(0, q_u^k), & & \forall e = (s, (u, 0)) \in E^T & (T.4)\\
& X_e = 0, & & \forall e = ((u, T_{max}), p) \in E^T, u \neq u_d & (T.5)\\
& Y_e = \max(-q_u, 0), & & \forall e = ((u, T_{max}), p) \in E^T & (T.6)\\
& CAP.X_e \geq Y_e, & & \forall e \in E^T & (T.7)
\end{aligned}$$

Section 12

Le *Projected 2 Flows Problem* : projection du TE2FP sur le graphe initial

12.1 Résoudre le problème projeté et retrouver le temps a posteriori

Deux raisons principales nous poussent à éliminer de la dimension temporelle :

- En pratique, la résolution du modèle temporel est beaucoup trop lourde et imprécise du fait de la discrétisation du temps. Aucune solution viable ne peut être obtenue en un temps raisonnable par la résolution du TE2FP.
- le problème projeté se rapproche des problèmes très connus et largement étudiés de *Pickup and Delivery*[Berbeglia et al., 2007, Hernández-Pérez and Salazar-González, 2009] et de *Balanced Transportation* [Vignaux and Michalewicz, 1991, Amaliah et al., 2022], sur lesquels nous pouvons nous baser. Cependant, même sans la dimension temporelle, de nouvelles contraintes doivent être ajoutées permettant de diriger les solutions et espérer récupérer plus facilement une solution du PRP à partir d'une solution du problème projeté que nous appelons *Projected 2 Flows Problem*.

Cette fois, les variables de décisions sont nommées $x = (x_e)_{e \in E}$, un vecteur d'entiers positifs représentant le flot de véhicules de transport (à lier à X dans le TE2FP) et $y = \sum_{k=0}^{K} y^k$ le flot de marchandises qui se décompose en plusieurs vecteurs de rationnels positifs $y^k = (y_e^k)_{e \in E}$ (à lier à Y^k dans le TE2FP).

Pour projeter le modèle du TE2FP sur le graphe initial, il faut commencer par projeter la fonction objectif, puis les 3 groupes de contraintes présentes dans le TE2FP :

La fonction objectif

Le coût des déplacements et le coût d'indisponibilité des marchandises ne changent pas. Cependant, le nombre de véhicules n'est pas connu : seule une borne inférieure peut être calculé comme étant la somme des temps de trajets divisée par l'horizon de temps. Par exemple, si la somme des trajets fait 100 unités de temps et l'horizon de temps est de 30 unités, alors il faut 4 véhicules au minimum. En notant x^V la variable entière représentant la borne inférieure du nombre de véhicules nécessaires, la fonction objectif s'exprime ainsi :

$$\text{minimiser } \alpha_1 \sum_{e \in E} C_e^x x_e + \alpha_2 \sum_{e \in E} C_e^y \sum_{k=0}^{K} y_e^k + \alpha_3 C^V x^V \tag{P.*}$$

et la borne inférieure se calcule grâce à la contrainte ci-dessous.

$$x^V \geq \frac{\sum_{e \in E} L_e x_e}{T_{max}} = \sum_{e \in E} l_e x_e \tag{P.1}$$

Lois de Kirchhoff et **Contraintes liées aux demandes**

Pour tout nœud $u \in N$

$$\sum_{e \in \delta^+(u)} x_e - \sum_{e \in \delta^-(u)} x_e = 0 \tag{P.2}$$

$$\sum_{e \in \delta^+(u)} y_e^k - \sum_{e \in \delta^-(u)} y_e^k = q_u^k, \ \forall k = 0, \ldots, K \tag{P.3}$$

Contraintes liants véhicules et marchandises
Si les marchandises sont transportées à travers un arc, elles doivent être transportées par suffisamment de véhicules. Pour tout arc e :

$$CAP.x_e \geq y_e \tag{P.4}$$

Cependant, il y a un groupe de contraintes majeur à ajouter : ce graphe projeté possède des cycles. Il faut donc ajouter des contraintes de sous-tours.

Contraintes de sous-tours étendues
Il est possible, à cette étape, d'ajouter un lien avec la dimension temporelle tout en supprimant les sous-tours. L'idée est la suivante : Le nombre de camions entrants dans un sous-ensemble S de nœuds doit être suffisant pour réaliser tous les trajets à l'intérieur et à la frontière de S dans l'horizon de temps. Par exemple, si la somme des trajets à l'intérieur de S est de 11 unités et l'horizon de temps et $T_{max} = 4$, trois camions doivent entrer en S.

$$x(\delta^-(S)) \geq \sum_{e \in \bar{\delta}(S)} l_e.x_e, \quad \forall S \subset N \backslash \{u_d\} \tag{P.5}$$

Le modèle du *Projected 2 Flows Problem (P2FP)* peut donc se résumer comme ci-dessous.

minimiser $\alpha_1 \sum_{e \in E} C_e^x x_e + \alpha_2 \sum_{e \in E} C_e^y \sum_{k=0}^{K} y_e^k + \alpha_3 C^V x^V$ $(P.*)$

tel que

(P2FP)

$$x^V \geq \sum_{e \in E} l_e x_e \tag{P.1}$$

$$\sum_{e \in \delta^+(u)} x_e - \sum_{e \in \delta^-(u)} x_e = 0 \qquad \forall u \in N \tag{P.2}$$

$$\sum_{e \in \delta^+(u)} y_e^k - \sum_{e \in \delta^-(u)} y_e^k = q_u^k, \qquad \forall u \in N, \forall k = 0, \ldots, K \tag{P.3}$$

$$CAP.x_e \geq y_e, \qquad \forall e \in E \tag{P.4}$$

$$x(\delta^-(S)) \geq \sum_{e \in \bar{\delta}(S)} l_e.x_e, \qquad \forall S \subset N \backslash \{u_d\} \tag{P.5}$$

12.2 Séparation des contraintes de sous-tours étendues (P.5)

L'objectif est, ici, de séparer les contraintes de sous-tours étendues du *Projected 2 Flows Problem*, car elles sont en nombre exponentiel et il n'est pas nécessaire de toutes les écrire. Ces contraintes seront alors séparées grâce à une procédure adaptée appliquée aux nœuds du *Branch&Bound* utilisé pour résoudre le problème P2FP.

Supposons le flot x réel, solution réalisable du P2FP hors contraintes de sous-tours étendues, et soit S un ensemble de nœuds ne contenant pas le dépôt. Les contraintes de sous-tours étendues peuvent être

réécrites de la façon suivante :

$$\begin{aligned}
&\sum_{e\in\delta^-(S)} x_e \geq \sum_{e\in\bar{\delta}(S)} l_e.x_e, && S\subset N\backslash\{u_d\}\\
\Leftrightarrow &\sum_{e\in\delta^+(S^c)} x_e \geq \sum_{e\in\bar{\delta}(S)} l_e.x_e, && S\subset N\backslash\{u_d\}\\
\Leftrightarrow &\sum_{e\in\delta^+(S^c)} x_e \geq \sum_{e\in E} l_e.x_e - \sum_{e\in\delta^*(S^c)} l_e.x_e, && S\subset N\backslash\{u_d\}\\
\Leftrightarrow &\sum_{e\in\delta^+(S^c)} x_e + \sum_{e\in\delta^*(S^c)} l_e.x_e - \sum_{e\in E} l_e.x_e \geq 0, && S\subset N\backslash\{u_d\}
\end{aligned}$$

Cette nouvelle forme d'équations nous permet de ne considérer que les sous-ensembles contenant le dépôt ($S\subset N\backslash\{u_d\}$, donc S^c contient le dépôt). Ainsi, un algorithme de flot maximum dans un graphe adapté (présenté juste après) permet de trouver une coupe $\{S,S^c\}$ associée à la contrainte (P.5) de plus petite valeur. Un tel algorithme est alors utilisé dans une procédure de séparation de contraintes présentée dans l'Algorithme 12.1.

Algorithme 12.1 : Séparation des contraintes de sous-tours étendues

Entrées : x un flot réel, solution réalisable du problème relâché.
Calculer une coupe $\{S,S^c\}$ (voir ci-dessous)
Obtenir la valeur de la contrainte associée
Si la valeur est négative **Alors**
Ajouter au modèle la contrainte de sous-tours étendue associée à S
Fin Si

Nous allons, à présent, présenter un graphe adapté nous permettant de calculer la coupe $\{S,S^c\}$ associée à la contrainte (P.5) de plus petite valeur.

x étant connu, $\sum_{e\in E} l_e.x_e$ est facilement calculable et ne dépendant d'aucune coupe. Nous cherchons à construire un graphe tel que la valeur du flot maximum soit égale à la plus petite valeur de $\sum_{e\in\delta^+(S^c)} x_e + \sum_{e\in\delta^*(S^c)} l_e.x_e$. De cette façon, une seule application d'un algorithme de flot maximum serait suffisante pour vérifier toutes les contraintes.

Théorème 1. *Soit $G'=(N\cup\{puit\},E')$ où E' est l'ensemble des arcs tels que :*
- *pour chaque arc $e=(u,v)\in E$, (u,v) est dans E' avec une capacité de $l_e.x_e$*
- *pour chaque arc $e=(u,v)\in E$, $(u,puit)$ est dans E' avec une capacité de $x_e-l_e.x_e$*

Le flot maximum de G' entre le dépôt et le puits est égale à la plus petite valeur de $\sum_{e\in\delta^+(S^c)} x_e + \sum_{e\in\delta^*(S^c)} l_e.x_e$ et les nœuds accessibles depuis le dépôt (il existe un chemin ne possédant aucun arc saturé) forment l'ensemble S^c recherché.

Un exemple du passage du graphe G au graphe G' est présenté Figure 12.1.

Preuve 6. Dans ce graphe G', le flot a trois possibilités : soit il sature un arc (u,v) de G et sature également son équivalent $(u,puit)$, soit il ne sature pas l'arc (u,v) mais sature $(u,puit)$, soit il ne sature aucun des deux. Dans le premier cas, l'arc (u,v) fait partie des arcs sortant de S^c (c'est-à-dire $\delta^+(S^c)$) et le flot saturant les deux arcs est de valeur x_e. Dans le deuxième cas, (u,v) est un arc intérieur de S^c (c'est-à-dire $\delta^*(S^c)$) et le flot sur $(u,puit)$ est de valeur $l_e.x_e$. Tout le flot est alors comptabilisé puisque, dans le troisième cas, les arcs entrants en u sont tous saturés, sinon le surplus serait utilisé pour saturer $(u,puit)$ et ce n'est pas le cas. De plus, le flot étant inférieur à la valeur de toute coupe, il est donc inférieur à la coupe de plus petite valeur. ■

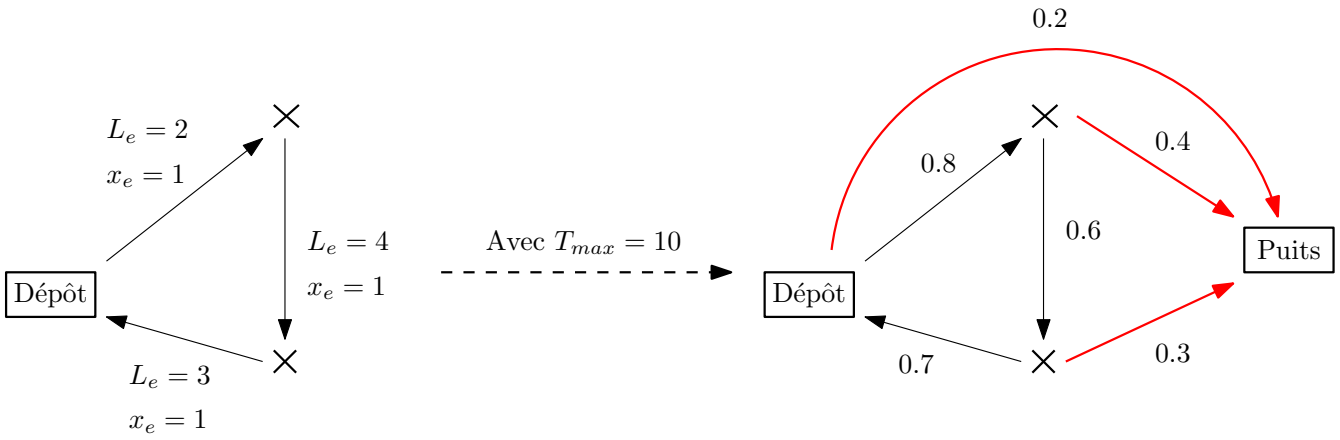

FIGURE 12.1 – Passage du graphe G au graphe G'.

La recherche d'un flot maximum peut être réalisé par l'algorithme de Ford-Fulkerson de complexité $O(F.E)$ où F est la valeur du flot maximal. Une recherche simple d'un chemin améliorant, nécessaire à l'algorithme de Ford-Fulkerson, peut être réalisé par une recherche en largeur d'un arbre de visite depuis le dépôt. Cependant, nous pouvons légèrement modifier l'algorithme de flot maximum pour tenir compte des spécificités de notre problème : si la valeur du flot dépasse $\sum_{e \epsilon E} l_e.x_e$, l'algorithme s'arrête puisqu'aucune contrainte n'est violée. Si, par contre, un flot max est trouvé, alors l'ensemble des nœuds accessibles depuis le dépôt S^c (c'est-à-dire s'il existe un chemin ne possédant aucun arc saturé) permet de générer une contrainte violée. Le pseudo code est présenté dans l'Algorithme 12.2.

Algorithme 12.2 : Algorithme de Ford-Fulkerson modifié

Entrées : x un flot réel, solution du problème relâché.
$\Delta = \sum_{e \epsilon E} l_e.x_e$
Début
Créer un graphe où x correspond aux capacités des arcs
Commencer avec un flot initial à 0
$F_{max} = 0$
Tant que $F_{max} < \Delta$ **et** il existe un chemin améliorant du dépôt vers i **Faire**
Ajouter le flot de ce chemin au graphe
La valeur du flot est ajouté à F_{max}
Fin Tant que
S^c est l'ensemble des nœuds accessibles depuis le dépôt
c'est-à-dire s'il existe un chemin ne possédant aucun arc saturé
$S = N \backslash \{S\}$
Fin

12.3 Les marchandises doivent suivre des chemins *réalisables*

Le problème du TE2FP n'a pas beaucoup de chance de permettre de récupérer la dimension temporelle à partir de la solution du P2FP tel que défini ci-dessus. Une première idée, pour faire tendre cette solution vers une vraie projection de la solution du TE2FP recherchée, est de s'assurer que le flot de marchandise puisse être respecté dans l'horizon de temps. Autrement dit, nous vérifions qu'un ou plusieurs véhicules, ne respectant pas obligatoirement la projection, puisse récupérer les marchandises, acheminer la cargaison via le flot de marchandises et revenir au dépôt dans le temps imparti. Si une telle situation n'est pas possible, des contraintes sont ajoutées au P2FP.

Nous proposons de chercher, pour chaque type $k = 0, \ldots, K$ de marchandises, une décomposition du flot en une somme de chemins *réalisables*, puis de vérifier que la distance du dépôt à l'origine de la marchandise, plus le temps du trajet du chemin, plus la distance de la destination de la marchandise au dépôt, est possible dans l'horizon de temps.

Définition 1. *Soit u un nœud fournisseur ($q_u > 0$) et v un nœud demandeur ($q_v < 0$), un chemin γ de u vers v est réalisable si, et seulement si, avec L_u^* la distance du dépôt à u et L_v^* la distance de v au dépôt :*

$$L_u^* + \sum_{e \in \gamma} L_e + L_v^* \leq T_{max}$$

Pour faire la décomposition et la vérification en même temps, nous proposons de vérifier que le flot de marchandises est décomposable en chemins *réalisables* d'un nœud fournisseur ($q_u > 0$) à un nœud demandeur ($q_u < 0$). Ainsi le problème de décomposition s'écrit, avec y le flot de marchandises, Γ l'ensemble des chemins *réalisables* et $y_\gamma = \left\{ \begin{array}{l} 1 \ si \ e \in \gamma \\ 0 \ sinon \end{array} , e \in E \right\}$:

$$(P) \quad t.q. \quad \begin{array}{r} Existe\text{-}t\text{-}il \ \lambda_\gamma, \\ y = \sum_{\gamma \in \Gamma} \lambda_\gamma . y_\gamma \\ \lambda_\gamma \geq 0 \end{array}$$

12.3.1 Séparation des contraintes de chemins *réalisables*

Γ est bien l'ensemble de tous les chemins possibles et pas uniquement des plus courts chemins. Si T_{max} est suffisamment grand, Γ contient la totalité des chemins possibles des nœuds fournisseurs aux nœuds demandeurs. Il est possible de résoudre ce problème directement en mettant en place un algorithme de génération de colonnes ou, dans sa version duale, par un algorithme de génération de coupes.

Dans ces méthodes, un sous-ensemble de $\Gamma_0 \subset \Gamma$ est généré initialement et s'agrandit au fur et à mesure des itérations. L'objectif étant de résoudre le problème en ne générant que des chemins utiles et ainsi accélérer la résolution sans perdre l'optimalité de la solution. Bien entendu, à une itération i donnée, si Γ_i ne contient pas les " bons " chemins, les contraintes sont impossibles à satisfaire. Il faut donc traiter ce sous-problème en parallèle du *Branch&Bound* afin de vérifier si la solution ne satisfait effectivement pas les contraintes ou si Γ_i doit être augmenté d'un chemin *réalisable*.

La résolution dans un solveur d'un problème avec génération de coupes est plus simple et plus rapide que celle d'un problème avec génération de colonnes. L'un étant le problème dual de l'autre, nous allons transformer notre problème en son dual et le donner à un solveur. Ainsi, en notant μ_e les variables duales, le problème devient :

$$(D) \quad t.q. \quad \begin{array}{r} min \ y.\mu, \\ \mu.y_\gamma \geq 0 \\ \mu_e \in IR \end{array}$$

Il vient également, par le lemme de Farkas, que si $\mu.y < 0$, alors le problème primal n'admet pas de solution et le problème peut être reformulé en :

$$(D') \quad \begin{array}{lll} Existe\text{-}t\text{-}il \ \mu, & & \\ t.q. & \mu.y < 0 & (1) \\ & \mu.y_\gamma \geq 0 & (2) \\ & \mu_e \in IR & (3) \end{array}$$

12.3.2 Recherche de chemins *réalisables* à ajouter

Comme dit dans la Section 12.3.1, les chemins *réalisables* ne vont pas tous être utilisés. Le problème va être traité avec les chemins $\gamma \in \Gamma_0$ initialement puis, si le problème (D') a une solution, une vérification est effectuée par savoir s'il existe un nouveau chemin *réalisable* γ_1 tel que $\mu.\gamma_1 < 0$. Si un tel chemin est trouvé, il est ajouté à l'ensemble des chemins $\Gamma_1 = \Gamma_0 \cup \{\gamma_1\}$, la contrainte associée est donc ajoutée à (D') et la résolution est relancée. Si aucun chemin n'est trouvé, alors le problème (P) n'admet effectivement pas de solution et le flot de marchandises n'est pas décomposable. La coupe $\mu.y \geq 0$ est donc ajoutée au P2FP et sa résolution est relancée.

Supposons alors que (D') ait une solution avec l'ensemble de chemins Γ_i. Une première méthode pour trouver un chemin *réalisable* γ_{i+1} tel que $\mu.y_{\gamma_{i+1}} < 0$ est de résoudre le problème suivant :

$$
(Ch) \quad \begin{aligned}
&\textit{Existe-t-il } \gamma_{i+1} \\
&t.q. \quad \mu.\gamma_{i+1} < 0 \\
&\gamma_{i+1}(\delta^-(u)) = \gamma_{i+1}(\delta^+(u)) \leq 1, \ \forall u \in \{u, q_u = 0\} \\
&\gamma_{i+1}(\delta^+(\{u, q_u > 0\})) = 1 \\
&\gamma_{i+1}(\delta^-(\{u, q_u < 0\})) = 1 \\
&\gamma_{i+1_e} = 0, \ \forall e \in \{e, f_e = 0\} \\
&\sum_{u \in \{u, q_u > 0\}} L(\gamma_u)\gamma_{i+1}(\delta^+(u)) + L(\gamma_{i+1}) + \sum_{v \in \{v, q_v < 0\}} L(\gamma_v)\gamma_{i+1}(\delta^-(v)) \leq T_{max} \\
&\gamma_{i+1_e} \in \{0, 1\}
\end{aligned}
$$

Cependant, n'importe quelle solution conviendrait. Nous proposons d'utiliser un algorithme de plus court chemin pour trouver plus rapidement une solution de (Ch). En particulier, nous proposons d'appliquer l'algorithme A* en utilisant μ_e comme distance, mais l'exécution est arrêtée dès qu'une solution à somme des μ négative est trouvée. La borne inférieure utilisée est la distance vers le nœuds demandeur v le plus proche (c'est-à-dire tel que $q_v < 0$).

Section 13

Un nouveau modèle pour remplacer le P2FP

Pour rappel, le réseau de transit initial est noté $G = (N, E)$ où N est l'ensemble des nœuds avec u_d le dépôt de véhicules et E et l'ensemble des arcs. Chaque nœud u possède une demande si $q_u^k < 0$, un surplus si $q_u^k > 0$ ou aucun des deux si $q_u^k = 0$ pour chaque marchandise $k = 0, \ldots, K$. Chaque arc e possède une longueur L_e, un coût (énergétique ou économique) pour les véhicules C_e^x et un coût d'indisponibilité pour les marchandises C_e^y.

Nous souhaitons résoudre le problème projeté grâce à un flot de véhicules et un flot de marchandises tels que les véhicules doivent partir du dépôt, faire leur tournée et revenir au dépôt avant une date limite T_{max}. Ces derniers possèdent une capacité CAP. Chaque véhicule utilisé possède un coût d'utilisation unitaire C^V.

Cependant, la résolution par le P2FP de l'instance présentée figure 13.1 pose des problèmes. En effet, la solution optimale, présentée figure 13.2, n'est pas transformable en une solution du PRP : La marchandise est livrée en v avant d'être récupérée en w (voir figure 13.3). L'objectif est de cette section est de présenter un nouveau modèle pour palier à ce problème, entre autres.

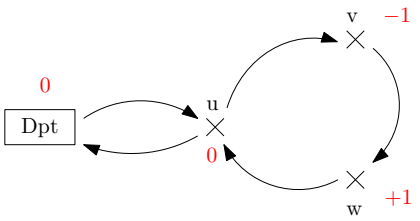

Figure 13.1 – Un réseau de transit.

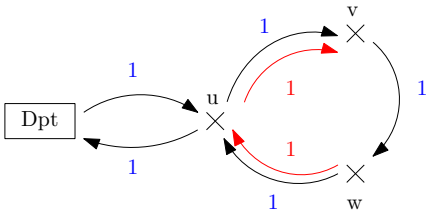

Figure 13.2 – Solution du P2FP sur le graphe représenté par la figure 13.1.

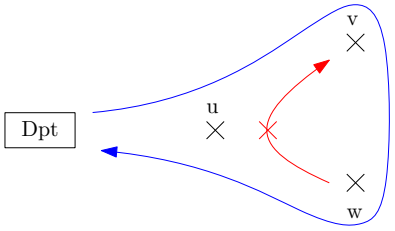

Figure 13.3 – Un faux cycle.

13.1 Des copies du graphe pour différencier les passages sur un même nœud

Le problème vient du cycle apparent w-u-v-w du flot de véhicules. La tournée est bien $Dpt - u - v - w - u - Dpt$, mais, sans informations temporelles, il est impossible de savoir ce qu'il se passe en u. Pour empêcher les faux cycles d'apparaître dans la solution, nous proposons de remplacer le graphe initial par un nouveau graphe $G^R = (N^R, E^R)$ et d'y construire un modèle linéaire interdisant les cycles dessus.

Pour autoriser plusieurs traversées d'un même nœud sans créer de cycle, il faut différencier les passages en ce nœud. Nous proposons de copier chaque nœud $u \in N$ plusieurs fois $u^0, u^1, \ldots, u^R \in N^R$ représentant les différents passages d'un véhicule sur le nœud u (voir l'exemple figure 13.4). Le nombre R de copie est arbitraire et, bien sûr, cette représentation ne permet d'exprimer que les solutions dans lesquelles les véhicules ne font pas plus de R passages en chaque nœud. Aussi, pour savoir qu'un véhicule est passé r fois sur un nœud, il est nécessaire qu'il soit identifiable. Dans ce modèle, les véhicules ne sont donc plus

mélangés en un unique flot, mais auront chacun un flot unitaire propre et le nombre de véhicules sera fixé arbitrairement à une valeur que nous nommons B.

De plus, les véhicules pourront traverser R fois un même nœud grâce à ses différentes copies, mais il n'y a pas de contrainte de précédence entre deux copies. Cela veut dire qu'un véhicule peut passer, par exemple, par u^2 puis u^R puis u^0. Cette liberté permet la préemption des marchandises : si un premier véhicule est déjà passé par u^0 mais qu'il doit retrouver un second véhicule en u, le transfert de marchandises aura lieu sur u^1, même si le deuxième véhicule n'est jamais passé par u avant. Cette idée oblige la construction d'un nombre conséquent d'arcs dans E^R puisque, pour un arc $(u, v) \in E$, toutes les copies du nœud u sont reliées à toutes les copies du nœud v. Il y a donc $|E| \times R^2$ arcs dans ce modèle (voir figure 13.5).

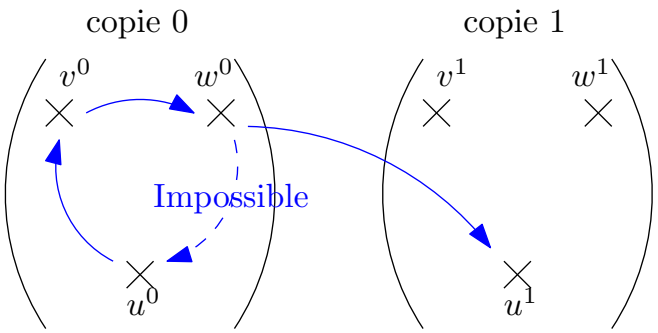

FIGURE 13.4 – Un exemple de flot d'un véhicule (les arcs ne sont pas tous représentés).

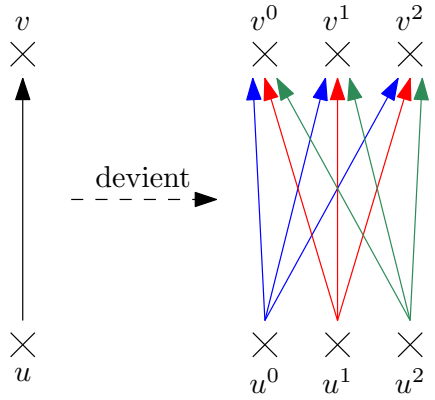

FIGURE 13.5 – Modification de l'arc (u, v) dans le premier modèle.

De plus, quatre nœuds fictifs sont ajoutés à E^R : une source $S_{\bar{x}}$ et un puits $P_{\bar{x}}$ pour le flot de véhicules, et une source $S_{\bar{y}}$ et un puits $P_{\bar{y}}$ pour le flot de marchandises.

Les arcs à ajouter pour le flot de véhicules sont (voir figure 13.6) :

- de la source vers u_d^0 : $(S_{\bar{x}}, u_d^0)$ de coût nul
- d'un dépôt u_d^r $(r > 0)$ vers le puits : $(u_d^r, P_{\bar{x}})$ de coût nul
- et du puits vers la source : $(P_{\bar{x}}, S_{\bar{x}})$ de coût nul pour boucler les tournées.

Les arcs à ajouter pour le flot de marchandises sont (voir figure 13.7) :

- des arcs transversaux : (u^r, u^{r+1}) de coût nul
- de la source vers u^0 : $(S_{\bar{y}}, u^0)$ de coût nul
- d'un nœud u^R vers le puits : (u^R, S_p) de coût nul
- et du puits vers la source : $(P_{\bar{y}}, S_{\bar{y}})$ de coût nul

Ici, plusieurs approches sont possibles, mais nous avons choisi celle ajoutant le moins d'arcs au réseau. Ici, les arcs ajoutés sont au nombre de $(|N| - 1) \times (R - 1)$ au maximum (dépend du nombre de demandes non nulles).

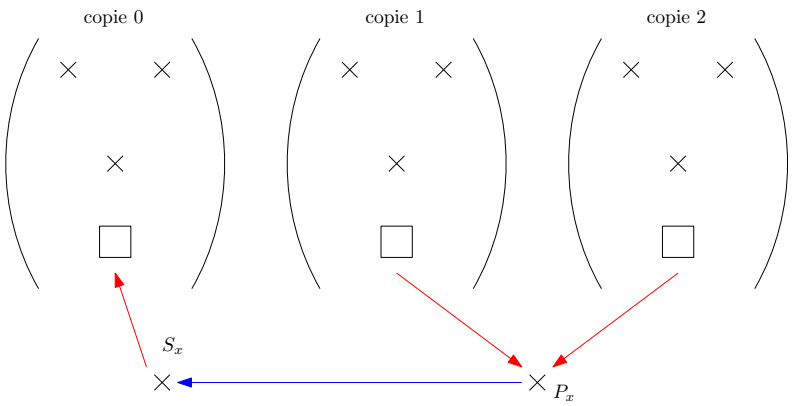

FIGURE 13.6 – Arcs fictifs des copies de u_d vers $S_{\bar{x}}$.

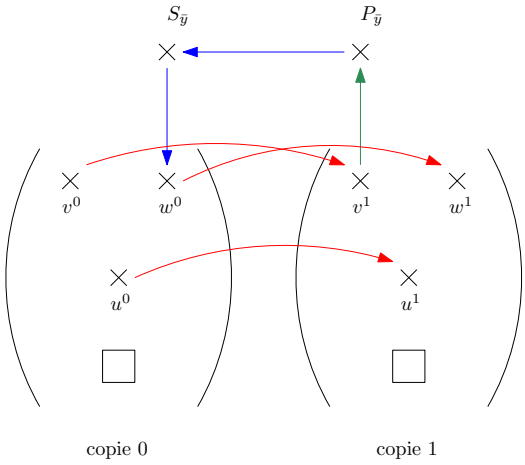

FIGURE 13.7 – Ajout d'arcs transversaux.

Dans ce nouveau graphe $G^R = (N^R, E^R)$, la solution optimale du P2FP (qui n'est pas une solution du PRP) n'existe pas dans ce nouveau modèle (voir figure 13.8) car le flot de marchandises est interrompu et ne respecte alors pas les lois de Kirchhoff. Ce modèle est donc plus restrictif tout en permettant d'exprimer toutes les solutions sans cycles apparents.

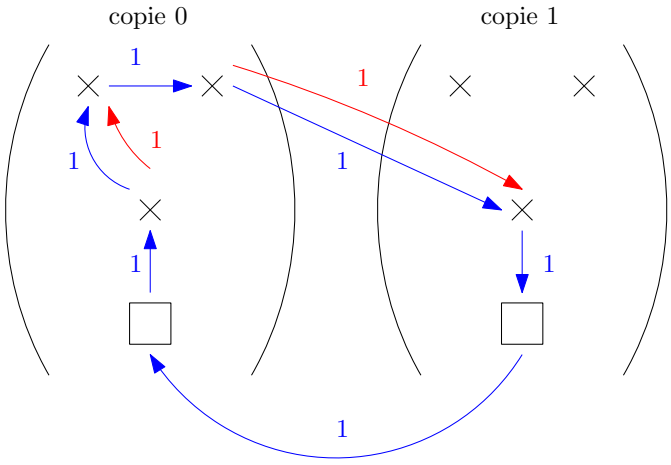

FIGURE 13.8 – Flots équivalents à ceux de la figure 13.2. Dans ce modèle, le flot de marchandises est impossible.

Autant pour le nombre de cycles que pour le nombre de camions, il est possible de ne pas fixer leur valeur à l'avance et de traiter le problème par génération de colonnes. Le modèle est d'abord résolu pour une certaine valeur de R et B, puis, si le modèle est insoluble, une nouvelle copie ou un nouveau véhicule est ajouté en fonction des contraintes bloquantes (voir [Desrosiers and Lübbecke, 2005]). Cette méthode de résolution étant très lourde, nous nous contentons de choisir arbitrairement des valeurs dépendant de chaque instance. Par exemple, il est possible d'estimer le nombre de véhicules nécessaire à partir de la quantité totale à transporter, de la capacité des camions et de l'horizon de temps.

13.2 Un nouveau modèle de projection du PRP

Soit les variables de décisions suivantes :

- Des vecteurs unitaires $\bar{x}^b = \left(\bar{x}^b_e \in \{0,1\}\right)_{e \in E^R}$ représentant le flot de chaque véhicule $b = 0, \ldots, B$ sur chaque arc $e \in E^R$.

- Des vecteurs de réels positifs $\bar{y}^k = \left(\bar{y}_e^k\right)_{e \in E^R}$ représentant le flot de marchandises $k = 0, \ldots, K$ sur chaque arc $e \in E^R$.

Nous ajoutons les vecteurs auxiliaires $\bar{x} = \sum_{b=0}^{B} \bar{x}^b$ et $\bar{y} = \sum_{k=0}^{k} \bar{y}^k$ pour simplifier certaines équations.

La fonction objectif
Elle est équivalente à celle du P2FP :

$$\text{minimiser } \alpha_1 \sum_{e \in E} C_e^x \bar{x}_e + \alpha_2 \sum_{e \in E} C_e^y \bar{y}_e^k \tag{C.*}$$

Lois de Kirchhoff sur $\bar{x}$

$$\bar{x}^b(\delta^-(u^r)) = \bar{x}^b(\delta^+(u^r)), \qquad \forall u^r \in N^R; b = 0, \ldots, B \tag{C.1}$$

Lois de Kirchhoff sur $\bar{y}$

$$\bar{y}(\delta^-(u^r)) = \bar{y}(\delta^+(u^r)), \qquad \forall u^r \in N^R \tag{C.2}$$

Demandes des nœuds
Les contraintes des demandes des nœuds s'expriment par un flot depuis la source $S_{\bar{y}}$ ou vers le puits $P_{\bar{y}}$:

$$\begin{aligned} \bar{y}_{(S_{\bar{y}}, u^0)} &= q_u, \qquad \forall u \in N \text{ t.q. } q_u > 0 \\ \bar{y}_{(u^0, P_{\bar{y}})} &= -q_u, \qquad \forall u \in N \text{ t.q. } q_u < 0 \end{aligned} \tag{C.3}$$

Recouvrement de $\bar{y}$ par $\bar{x}^b$
Bien sûr, les marchandises doivent être transportées par suffisamment de véhicules. Les contraintes de recouvrements s'expriment donc ainsi :

$$R.\bar{x}_e \geq \bar{y}_e, \qquad \forall e \in E^R \tag{C.4}$$

Un véhicule ne peut passer deux fois sur le même nœud
L'objectif premier de ce modèle est d'empêcher un véhicule de créer de "faux" tours. Les véhicules sont donc forcés à ne passer qu'une seule fois maximum sur chaque nœud du nouveau réseau. Les contraintes de passage s'expriment donc ainsi :

$$\bar{x}^b(\delta^-(u^r)) \leq 1, \qquad \forall u^r \in N^R; b = 0, \ldots, B \tag{C.5}$$

Temps de parcours d'une tournée
Puisque les véhicules sont distincts, il est possible de calculer le temps de parcours de sa tournée. Cette durée doit rester inférieure à la date limite T_{max}. Les contraintes de temps de parcours des tournées s'expriment donc ainsi :

$$\sum_{e \in E^R} L_e.\bar{x}_e^b \leq T_{max}, \qquad b = 0, \ldots, B \tag{C.6}$$

Élimination des sous-tours
Ici encore, les contraintes d'*Extended no sub-tour* sont nécessaires et appliquées au réseau entier. Les contraintes *Extended no sub-tour* s'expriment donc ainsi :

$$T_{max}. \sum_{k \in K} \bar{x}^b(\delta^-(S)) \geq \sum_{k \in K} L(\bar{\delta}(S)).\bar{x}^b(\bar{\delta}(S)), \qquad \forall S \subset N^R; b = 0, \ldots, B \tag{C.7}$$

Le modèle linéaire complet associé à ce nouveau graphe G^R est nommé *Cycle-Extended 2 Flows Problem (CE2FP)* et est présenté ci-dessous.

$$\textbf{minimiser} \quad \alpha_1 \sum_{e \in E} C_e^x \sum_{b=0}^{B} \bar{x}_e^b + \alpha_2 \sum_{e \in E} C_e^y \sum_{k=0}^{K} \bar{y}_e^k \qquad (C.*)$$

tel que

$$\bar{x}^b(\delta^-(u^r)) = \bar{x}^b(\delta^+(u^r)), \quad \forall u^r \in N^R; b = 0, \dots, B \qquad (C.1)$$

$$\bar{y}(\delta^-(u^r)) = \bar{y}(\delta^+(u^r)), \quad \forall u^r \in N^R \qquad (C.2)$$

$$\bar{y}_{(S_{\bar{y}}, u^0)} = q_u, \quad \forall u \in N \text{ t.q. } q_u > 0 \qquad (C.3)$$

$$\bar{y}_{(u^0, P_{\bar{y}})} = -q_u, \quad \forall u \in N \text{ t.q. } q_u < 0$$

(CE2FP)

$$R.\bar{x}_e \geq \bar{y}_e, \quad \forall e \in E^R \qquad (C.4)$$

$$\bar{x}^b(\delta^-(u^r)) \leq 1, \quad \forall u^r \in N^R; b = 0, \dots, B \qquad (C.5)$$

$$\sum_{e \in E^R} L_e.\bar{x}_e^b \leq T_{max}, \quad b = 0, \dots, B \qquad (C.6)$$

$$T_{max}. \sum_{k \in K} \bar{x}^b(\delta^-(S)) \geq \sum_{k \in K} L(\bar{\delta}(S)).\bar{x}^b(\bar{\delta}(S)), \quad \forall S \subset N^R; b = 0, \dots, B \qquad (C.7)$$

13.3 Discussions

Il y a deux questions importantes à se poser : peut-on projeter toutes solutions du PRP dans ce modèle ? Et, réciproquement, une solution de ce modèle est-elle transformable en une solution du PRP ? Malheureusement, bien que les problèmes au sein d'une tournée aient été réglés, les échange de marchandises entre deux véhicules restent opaques. En effet, il n'est pas possible de savoir les dates des deux véhicules lors d'un transfert pour une simple raison : les marchandises et les véhicules ne sont pas liés. Il n'est pas possible de savoir que le véhicule b transporte une certaine quantité de marchandises k : si un autre véhicule prend le même arc, le flot est mélangé et il est impossible de les distinguer. Il n'est donc pas possible de dater les marchandises et cela engendre des problèmes. La figure 13.9 montre un exemple dans lequel les deux tournées sont de longueur égale à l'horizon de temps, mais le transfert de marchandises, qui n'est pas visible dans ce modèle, entre les deux véhicules oblige l'un des deux à attendre 1 unité de temps et fais dépasser l'horizon de temps.

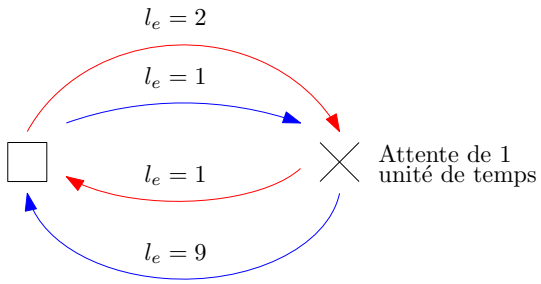

FIGURE 13.9 – Le véhicule dont la tournée dure 10 unités de temps doit attendre 1 unité de temps pour récupérer des marchandises du deuxième véhicule, dépassant l'horizon de temps si $T_{max} = 10$.

Ainsi, cette modélisation, bien que permettant d'exprimer plus de solutions, ne permet pas non plus de ne trouver que des solutions du PRP, d'autant qu'il y a une forte limitation sur le nombre de cycles et de véhicule au maximum. Nous avons donc décidé de ne pas la tester telle quelle, mais nous avons repris l'idée qu'un véhicule ne fera qu'un nombre limité de cycles lors de la récupération de la dimension temporelle à partir d'une solution projetée.

Section 14

Reconstruction d'une solution du PRP à partir du P2FP

Dans cette section, nous supposons résolu le *Projected 2 Flows Problem* sur le graphe $G = (V, A)$ et cherchons à reconstruire une solution du PRP à partir de la solution optimale du P2FP. Nous avons les variables suivantes et leur valeur :

- x_{uv} un entier représentant le nombre de véhicules traversant l'arc entre les nœuds u et v.
- y^k_{uv} un réel représentant la quantité de marchandise k traversant l'arc entre les nœuds u et v.

Nous avons projeté le modèle initialement dans le but de rechercher la projection se la solution du PRP au travers de ce modèle projeté. Dans un premier temps, nous allons considérer que c'est bien le cas, c'est-à-dire que la solution du P2FP est bien la projection de la solution optimale du PRP. Nous appelons *Strong Lift* ce problème de reconstruction de la solution du PRP. Cependant, comme précisé dans les discussions lors de la section sur la projection, il n'y a pas de raison que la solution optimale du P2FP soit exactement la projection de la solution optimale du PRP. Et il est même possible, nous en avons discuté dans la section précédente, que la solution optimale du P2FP ne soit la projection d'aucune solution du PRP. Néanmoins, nous avons ajouté des contraintes sur le flot de marchandises, assurant sa faisabilité dans l'horizon de temps. Pour rappel, ces contraintes que nous avons nommées contraintes de décomposition, vérifie que le flot de marchandises se décompose en chemins *accteptable*, c'est-à-dire qu'il existe un tour qui part du dépôt, parcourt le chemin avec la marchandise, puis revient au dépôt sans dépasser l'horizon de temps. Alors, en ne respectant que la projection du flot de marchandise, nous pouvons, à coup sûr, reconstruire une solution réalisable du PRP. Nous appelons *Weak Lift* ce problème de reconstruction plus permissif, mais il n'y a pas de garantie de reconstruire la solution optimale du PRP.

Pour résumer, nous allons présenter deux problèmes :

- Le *Strong Lift* dans lequel l'objectif est de reconstruire les flots X et Y sur le graphe G^T tels que x et y sont les projections de X et Y sur le graphe G.
- Le *Weak Lift* dans lequel l'objectif est de reconstruire les flots X et Y sur le graphe G^T tels que y est la projection de Y sur le graphe G.

Pour illustrer les différents algorithmes et heuristiques présentés dans cette section, nous utiliserons l'exemple présenté Figure 14.1 avec des marchandises distinctes. Le dépôt des véhicules est le nœud carré et $T_{max} = 40$. Pour des raisons de simplicité de l'exemple, la capacité des véhicules est de 1 et la durée de traversée des arcs ne varie pas au cours du temps. Il y a deux marchandises (resp. en rouge et en vert). Le nombre positif (resp. négatif) désigne l'origine et la quantité (resp. la destination et la quantité) de chaque marchandise. La solution optimale du P2FP de cette instance est présentée Figure 14.2. Dans cette solution, pour pouvoir acheminer à temps la marchandise rouge, le véhicule doit passer par l'origine de la marchandise verte. Ce véhicule en profite donc pour avancer la marchandise d'un arc pour pouvoir la donner au second véhicule qui est passé par l'autre chemin moins coûteux. Le premier véhicule ne va cependant pas attendre le second, il dépose la marchandise verte, prend la rouge et repart immédiatement. Trois unités de temps plus tard, le second véhicule arrive pour récupérer la marchandise verte et part

l'acheminer à sa destination.

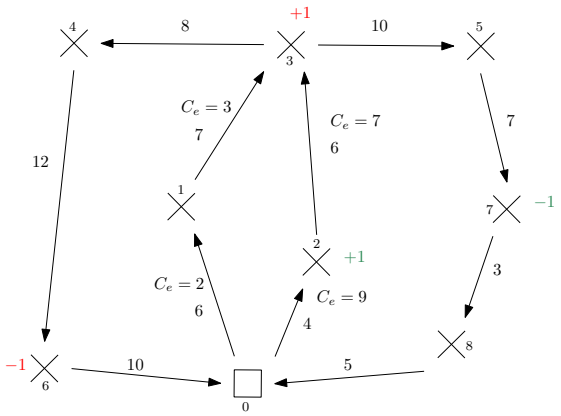

FIGURE 14.1 – Graphe d'exemple étudié

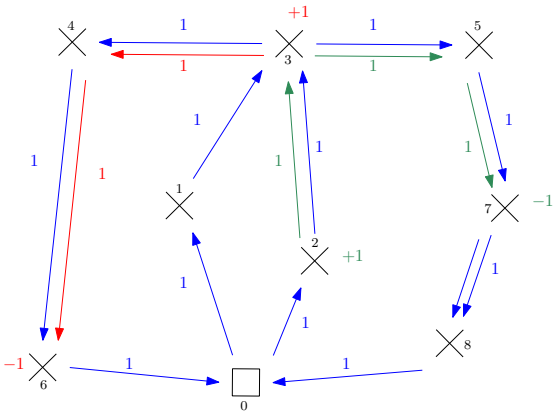

FIGURE 14.2 – Solution du P2FP sur l'instance Figure 14.1

14.1 Un programme linéaire pour résoudre le *Strong Lift*

Dans cette section, nous proposons un graphe sur lequel nous pouvons définir un programme linéaire en nombres entier pour résoudre le problème du *Strong Lift*. Nous reprenons l'idée que, pour pouvoir exprimer des tournées et ainsi pouvoir les dater, il faut désagréger le flot de véhicules. À la différence du modèle présenté dans la section précédente, nous connaissons le nombre de passages de véhicules sur chaque arc grâce à la valeur de x dans la solution projetée. En effet, dans le *Strong Lift*, le flot x est exactement la projection de X sur le graphe G et, de la même façon, le flot de marchandise y est exactement la projection de Y. Il suffit alors de définir les tournées et de distribuer les marchandises aux différents passages des véhicules. Cela est fait en regardant ce qu'il se passe à l'intérieur d'un nœud, c'est-à-dire en identifiant et en liant les entrées et les sorties d'un même nœud. Pour ce faire, le graphe $G^{SL} = (N^{SL}, E^{SL})$ que nous proposons possède autant de copies de chaque arc que la valeur du flot de véhicules x sur cet arc dans la solution du P2FP. Cela nous permet de désagréger et d'identifier le flot de véhicules. Pour finir la reconstruction d'une solution du PRP, autrement dit pour construire des tournées, il faut lier les différentes copies des arcs en regardant ce qu'il se passe à l'intérieur de chaque nœud.

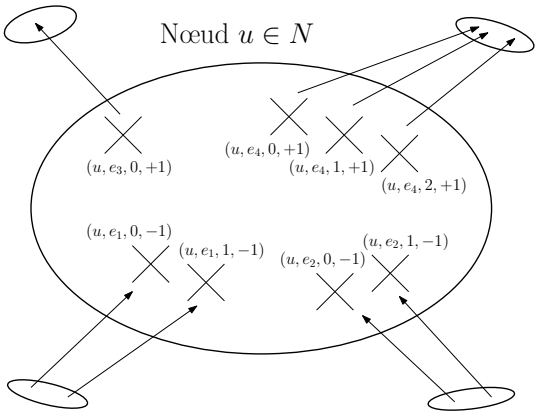

FIGURE 14.3 – Un nœud $u \in N$ est copié pour chaque départ et arrivée d'arcs.

Pour pouvoir accommoder ces copies d'arcs, chaque nœud de N devient regroupement de plusieurs nœuds dans N^{SL} (voir Figure 14.3) : un nœud pour le départ de chaque copie d'un arc et un nœud pour chaque arrivée. Chacun de ces arcs peut être identifié par un triplé unique (u, e, id) avec $u \in N$, $e \in E$ tel que $e = (u, v)$ ou $e = (v, u)$ et id est l'identifiant de la copie de l'arc. Pour des raisons de simplification,

nous identifierons les nœuds par un quadruplé $(u, e, id, \{+1, -1\})$ où la quatrième valeur est +1 si $e = (u, v)$ (un arc sortant de u) et -1 si $e = (v, u)$ (un arc entrant en u).

Un arc $e^{SL} \in E^{SL}$ peut, quant à lui, appartenir à l'un des deux groupes suivants (voir figure 14.4) :

- les arcs *copies* : avec $e = (u, v) \in E$ $(u, e, i, +1) \to (v, e, i, -1)$
- les arcs *traversants* : $(u, e, i, -1) \to (u, e', i', +1)$

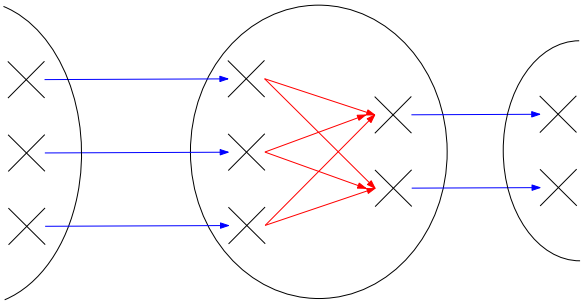

FIGURE 14.4 – Les deux catégories d'arcs : *copies* en bleu et *traversants* en rouge.

La longueur $L_{e^{SL}}$ (en temps) et le coût $C_{e^{SL}}$ des arcs *copies* sont les mêmes que pour le graphe G. Les arcs *traversants* et *parcourants* n'ont ni longueur, ni coût.

14.1.1 Variables de décisions et variables auxiliaires

Les variables de décisions associées au flot marchandises sont :

- Des vecteurs de réels positifs $Y^k = \left(Y_e^k\right)_{e \in E^{SL}}$ représentant le flot de marchandises $k = 0, \ldots, K$ sur chaque arc *copies* de E^{SL}.
- Des vecteurs de réels positifs $Y^{k*} = \left(Y_e^{k*}\right)_{e \in E^{SL}}$ représentant le flot de marchandises $k = 0, \ldots, K$ sur chaque arc *traversant* de E^{SL}.

Nous ajoutons les vecteurs auxiliaires $Y = \sum_{k=0}^{K} Y^k$ et $Y^* = \sum_{k=0}^{k} Y^{k*}$ pour simplifier certaines équations.

Les variables de décisions associées au flot véhicules sont :

- Un vecteurs de variables unitaires $X = (X_e \in \{0, 1\})_{e \in E^{SL}}$ représentant le flot de véhicules sur chaque arc $e \in E^{SL}$.

À noter que X est toujours égale à 1 sur les arcs *copies* puisqu'il doit se projeter exactement sur x.

Les variables associées au temps sont :

- T un vecteur de réels indexés sur les nœuds de N^{SL}.

Au total, il y a $2 \times card(E^{SL}) + card(N^{SL}) = 4 \times card(E^{SL}$ variables. Ce n'est pas beaucoup dans la mesure où les arcs copiés sont uniquement les arcs apparaissant dans la solution optimale du P2FP et qu'en plus certaines variables du vecteur X sont déjà fixées à 1.

14.1.2 Les Contraintes du Strong Lift

La fonction objectif est simple puisque les arcs porteurs de coût sont obligatoires et les décisions ne portent que sur des arcs de coût nul. La seule décision importante est de déterminer le nombre de véhicules nécessaire. Afin de calculer ce nombre de véhicule, il nous faut définir une tournée :

Une tournée est définie par un départ depuis l'un des nœuds sortant du dépôt u_d $((u_d, e, i, +1) \in N^{SL})$ et une arrivée à l'un des nœuds entrant au dépôt $((u_d, e, i, -1) \in N^{SL})$. Cependant, un véhicule peut effectuer un passage par le dépôt sans s'y arrêter. Pour les différencier, lors d'un passage par le dépôt, un arc traversant est utilisé. Dans le cas d'un départ ou d'une arrivée, le flot apparaît ou disparaît sans respecter la loi de Kirchhoff (voir Figure 14.5).

Ainsi, pour calculer le nombre de tournées, et donc le nombre de véhicules, il faut faire la différence entre la quantité de flot sortant (ou entrant) depuis n'importe quel nœud copie du dépôt u_d $((u_d, e, i, +1) \in N^{SL})$

et la quantité de flot traversant l'intérieur du nœud dépôt (indiquant que le véhicule est de passage et il ne faut pas le compter plusieurs fois).

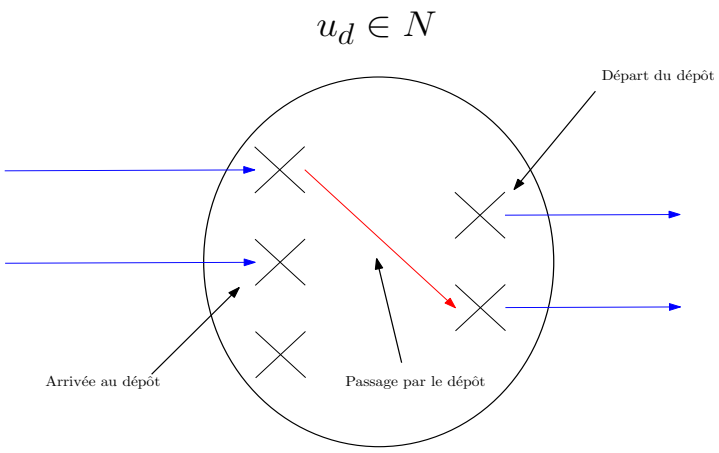

FIGURE 14.5 – Différence entre passage et départ/arrivée de véhicules au dépôt.

La fonction objectif est donc :

$$\sum_{u^{SL}=(u_d,e,i,+1)\in N^{SL}} \left(\sum_{e^{SL}\in\delta^+(u^{SL})} X_{e^{SL}} - \sum_{e^{SL}\in\delta^-(u^{SL})} X_{e^{SL}} \right) \tag{SL.*}$$

Les contraintes associées aux marchandises sont :

- Y^* permet la conservation du flot marchandise :
  Ici, l'idée est que les marchandises arrivant sur un nœud u (c'est-à-dire sur un nœud $u^{SL} = (u, e, i, -1)$) sont plus nombreuses que celles qui traversent le nœud car il peut y avoir une demande qui obligent les marchandises à s'arrêter (SL.1.1). Et, de la même façon, la quantité de marchandises qui part du nœud u (c'est-à-dire d'un nœud $u^{SL} = (u, e, i, +1)$) peut être plus importante que la quantité qui a traversé le nœud puisqu'il peut y avoir un surplus des marchandises ajouté au flot sortant (SL.1.2).

$$\sum_{e^{SL}\in\delta^-(u^{SL})} Y^r_{e^{SL}} \geq \sum_{e^{SL}\in\delta^+(u^{SL})} Y^{r*}_{e^{SL}}, \quad \forall u^{SL} = (u, e, i, -1) \in N^{SL}, \forall r = 0, \dots, R \tag{SL.1.1}$$

$$\sum_{e^{SL}\in\delta^-(u^{SL})} Y^{r*}_{e^{SL}} \leq \sum_{e^{SL}\in\delta^+(u^{SL})} Y^r_{e^{SL}}, \quad \forall u^{SL} = (u, e, i, +1) \in N^{SL}, \forall r = 0, \dots, R \tag{SL.1.2}$$

- Y^* permet la satisfaction des demandes :
  En reprenant l'idée précédente, mais cette fois en regardant la somme de tout ce qui entre, traverse et sort d'un nœud u, alors nous connaissons exactement la quantité de demande et de surplus.

$$\sum_{\substack{e^{SL}\in\delta^-(u^{SL}) \\ u^{SL}=(u,e,i,-1)\in N^{SL}}} Y^r_{e^{SL}} = \sum_{\substack{e^{SL}\in\delta^+(u^{SL}) \\ u^{SL}=(u,e,i,-1)\in N^{SL}}} Y^r_{e^{SL}} + \max(0, q^r_u), \quad \forall r = 0, \dots, R \tag{SL.2.1}$$

$$\sum_{\substack{e^{SL}\in\delta^-(u^{SL}) \\ u^{SL}=(u,e,i,+1)\in N^{SL}}} Y^{r*}_{e^{SL}} + \max(0, -q^r_u) = \sum_{\substack{e^{SL}\in\delta^+(u^{SL}) \\ u^{SL}=(u,e,i,+1)\in N^{SL}}} Y^r_{e^{SL}}, \quad \forall r = 0, \dots, R \tag{SL.2.2}$$

- Y doit se projeter exactement sur y sur tous les arcs porteurs de flot véhicule :

$$\sum_{e^{SL} \text{ copie de } e} Y^r_{e^{SL}} = y^r_e, \quad \forall r = 0, \dots, R, \forall e \in E \text{ t.q. } x_e > 0 \tag{SL.3}$$

Les contraintes associées au temps sont :

- Les dates au départ du dépôt sont positives et doivent être inférieures à la date limite à l'arrivée :

$$T_{u^{SL}} \geq 0, \quad \forall u^{SL} = (u_d, e, i, +1) \in N^{SL} \tag{SL.4}$$

$$T_{u^{SL}} \leq T_{max}, \quad \forall u^{SL} = (u_d, e, i, -1) \in N^{SL} \tag{SL.5}$$

- La date à l'arrivée d'un arc *copie* doit respecter le temps de trajet de l'arc :

$$T_{v^{SL}} \geq T_{u^{SL}} + L_e, \quad \forall e^{SL} = (u^{SL}, v^{SL}) \text{ copie de } e = (u, v) \tag{SL.6}$$

- La date sur le nœud d'arrivée d'un arc *traversant* doit permettre d'attendre toutes les marchandises :

$$T_{v^{SL}} \geq T_{u^{SL}} \quad \text{si} \quad Y^*_{e^{SL}} > 0, \quad \forall e^{SL} = (u^{SL}, v^{SL}) \in \textit{traversants} \tag{SL.7}$$

- La date sur le nœud d'arrivée d'un arc *parcourant* doit permettre d'attendre tous les véhicules :

$$T_{v^{SL}} \geq T_{u^{SL}} \quad \text{si} \quad X_{e^{SL}} = 1, \quad \forall e^{SL} = (u^{SL}, v^{SL}) \in \textit{parcourants} \tag{SL.8}$$

Les contraintes associées aux véhicules sont :

- Les véhicules vérifient les lois de Kirchhoff en tout nœud de N^{SL} sauf au dépôt

$$\sum_{e^{SL} \in \delta^-(u^{SL})} X_{e^{SL}} = \sum_{e^{SL} \in \delta^+(u^{SL})} X_{e^{SL}}, \quad \forall u^{SL} = (u, e, i, \pm 1) \in N^{SL} \text{ t.q. } u \neq u_d \tag{SL.9}$$

- Au dépôt, la somme de tout ce qui entre est égale à la somme de tout ce qui sort (SL.10.1), mais il faut également vérifier que les véhicules qui traverse sont bien arrivés d'un arc entrant (SL.10.2) et ressortent part un arc sortant (SL.10.3).

$$\sum_{e^{SL} \in \delta^-(u^{SL})} X_{e^{SL}} = \sum_{e^{SL} \in \delta^+(v^{SL})} X_{e^{SL}}, \quad \forall^{u^{SL}=(u_d,e,i,-1)\in N^{SL}}_{v^{SL}=(u_d,e,i,+1)\in N^{SL}} \tag{SL.10.1}$$

$$\sum_{e^{SL} \in \delta^-(u^{SL})} X_{e^{SL}} \geq \sum_{e^{SL} \in \delta^+(u^{SL})} X_{e^{SL}}, \quad \forall u^{SL} = (u_d, e, i, -1) \in N^{SL} \tag{SL.10.2}$$

$$\sum_{e^{SL} \in \delta^-(u^{SL})} X_{e^{SL}} \leq \sum_{e^{SL} \in \delta^+(u^{SL})} X_{e^{SL}}, \quad \forall u^{SL} = (u_d, e, i, +1) \in N^{SL} \tag{SL.10.3}$$

- Pour faciliter la résolution, la contrainte ci-dessous est ajoutée car un véhicule doit obligatoirement traverser les arcs copies.

$$X_{e^{SL}} = 1, \quad \forall e^{SL} \in \textit{copies} \tag{SL.11}$$

- Un véhicule ne peut pas transporter plus que sa capacité :

$$Y_{e^{SL}} < X_{e^{SL}}.CAP, \quad \forall e^{SL} \in \textit{copies} \tag{SL.12}$$

14.1.3 Un MILP pour résoudre le problème du *Strong Lift*

Un programme linéaire pour résoudre le problème du *Strong Lift* peut donc s'écrire comme suit :
avec M une très grande valeur et $Z \in \{0, 1\}$ un vecteur de variables auxiliaires indicé sur les arcs traversants pour les contraintes conditionnelles (SL.7) et (SL.8),

(Strong Lift)
minimiser $\sum_{u^{SL}=(u_d,e,i,+1)\in N^{SL}} \left(\sum_{e^{SL}\in\delta^+(u^{SL})} X_{e^{SL}} - \sum_{e^{SL}\in\delta^-(u^{SL})} X_{e^{SL}} \right)$

tel que

$$\sum_{e^{SL}\in\delta^-(u^{SL})} Y^r_{e^{SL}} \geq \sum_{e^{SL}\in\delta^+(u^{SL})} Y^{r*}_{e^{SL}}, \quad \forall u^{SL}=(u,e,i,-1)\in N^{SL},\ \forall r=0,\dots,R \qquad (SL.1.1)$$

$$\sum_{e^{SL}\in\delta^-(u^{SL})} Y^{r*}_{e^{SL}} \leq \sum_{e^{SL}\in\delta^+(u^{SL})} Y^{r}_{e^{SL}}, \quad \forall u^{SL}=(u,e,i,+1)\in N^{SL},\ \forall r=0,\dots,R \qquad (SL.1.2)$$

$$\sum_{\substack{e^{SL}\in\delta^-(u^{SL})\\ u^{SL}=(u,e,i,-1)\in N^{SL}}} Y^r_{e^{SL}} - \max(0,q^r_u) = \sum_{\substack{e^{SL}\in\delta^+(u^{SL})\\ u^{SL}=(u,e,i,-1)\in N^{SL}}} Y^r_{e^{SL}}, \quad \forall r=0,\dots,R \qquad (SL.2.1)$$

$$\sum_{\substack{e^{SL}\in\delta^-(u^{SL})\\ u^{SL}=(u,e,i,+1)\in N^{SL}}} Y^{r*}_{e^{SL}} + \max(0,-q^r_u) = \sum_{\substack{e^{SL}\in\delta^+(u^{SL})\\ u^{SL}=(u,e,i,+1)\in N^{SL}}} Y^r_{e^{SL}}, \quad \forall r=0,\dots,R \qquad (SL.2.2)$$

$$\sum_{e^{SL}\text{ copie de }e} Y^r_{e^{SL}} = y^r_e, \quad \forall r=0,\dots,R, \forall e\in E \text{ t.q. } x_e>0 \qquad SL.3$$

$$T_{u^{SL}} \geq 0, \quad \forall u^{SL}=(u_d,e,i,+1)\in N^{SL} \qquad (SL.4)$$

$$T_{u^{SL}} \leq T_{max}, \quad \forall u^{SL}=(u_d,e,i,-1)\in N^{SL} \qquad (SL.5)$$

$$T_{v^{SL}} \geq T_{u^{SL}} + L_e, \quad \forall e^{SL}=(u^{SL},v^{SL}) \text{ copie de } e=(u,v) \qquad (SL.6)$$

$$Z_{e^{SL}} \geq \tfrac{1}{M}.Y^*_{e^{SL}}, \quad \forall e^{SL}=(u^{SL},v^{SL})\in \textit{traversants} \qquad (SL.7.1)$$

$$T_{u^{SL}} - M.(1-Z_{e^{SL}}) \leq T_{v^{SL}}, \quad \forall e^{SL}=(u^{SL},v^{SL})\in \textit{traversants} \qquad (SL.7.2)$$

$$T_{u^{SL}} + L_{e^{SL}} - M.(1-X_{e^{SL}}) \leq T_{v^{SL}}, \quad \forall e^{SL}=(u^{SL},v^{SL})\in \textit{parcourants} \qquad (SL.8)$$

$$\sum_{e^{SL}\in\delta^-(u^{SL})} X_{e^{SL}} = \sum_{e^{SL}\in\delta^+(u^{SL})} X_{e^{SL}}, \quad \forall u^{SL}=(u,e,i,\pm1)\in N^{SL} \text{ t.q. } u\neq u_d \qquad (SL.9)$$

$$\sum_{e^{SL}\in\delta^-(u^{SL})} X_{e^{SL}} = \sum_{e^{SL}\in\delta^+(v^{SL})} X_{e^{SL}}, \quad \forall u^{SL}=(u_d,e,i,-1)\in N^{SL},\ \forall v^{SL}=(u_d,e,i,+1)\in N^{SL} \qquad (SL.10.1)$$

$$\sum_{e^{SL}\in\delta^-(u^{SL})} X_{e^{SL}} \geq \sum_{e^{SL}\in\delta^+(u^{SL})} X_{e^{SL}}, \quad \forall u^{SL}=(u_d,e,i,-1)\in N^{SL} \qquad (SL.10.2)$$

$$\sum_{e^{SL}\in\delta^-(u^{SL})} X_{e^{SL}} \leq \sum_{e^{SL}\in\delta^+(u^{SL})} X_{e^{SL}}, \quad \forall u^{SL}=(u_d,e,i,+1)\in N^{SL} \qquad (SL.10.3)$$

$$X_{e^{SL}} = 1, \quad \forall e^{SL}\in \textit{copies} \qquad (SL.11)$$

$$Y_{e^{SL}} < X_{e^{SL}}.CAP, \quad \forall e^{SL}\in \textit{copies} \qquad (SL.12)$$

En pratique, ce programme ne fonctionne pas très souvent à cause des temps d'attentes non prévus par la solution projetée, mais aussi à cause des cycles apparents que nous avons essayé de traiter dans la section précédente. Ainsi, dès qu'un transfert de marchandises entre deux véhicules a lieu et qu'un véhicule doit attendre, ou lorsqu'une marchandise est déposée avant d'être récupéré dans la solution projetée, alors le programme linéaire ci-dessus est infaisable.

14.2 Un programme linéaire pour résoudre le *Weak Lift*

Dans le cas du *Weak Lift*, le flot de marchandises forme un arbre et la quantité d'arcs copiés est significativement plus petite que pour le *Strong Lift*. Cependant, à l'inverse de ce dernier, aucune information n'est ici disponible sur le nombre de passages des véhicules sur chaque arc. Nous connaissons néanmoins la quantité à transporter et donc une borne inférieure du nombre de passages nécessaires $\left\lceil \frac{y_e}{CAP} \right\rceil$.

Nous proposons donc un graphe $G^{WL} = (N^{WL}, E^{WL})$, similaire au graphe G^{SL} présenté précédemment dans la mesure où, ici encore, les nœuds sont copiés pour chaque départ et arrivée des arcs et sont identifiés par un quadruplé $(u, e, id, \{+1, -1\})$ (voir Figure 14.3). Cependant, les arcs copiés e ne sont que ceux portant du flot de marchandises non nul $y_e > 0$ et en quantité $\left\lceil \frac{y_e}{CAP} \right\rceil$ puisque, dans le *Weak Lift*, seul le flot y est la projection de Y.

Il faut alors ajouter un nœud source $s \in N^{WL}$, un nœud puits $p \in N^{WL}$ et des arcs pour permettre le déplacement à vide des véhicules de transports. Ces nouveaux arcs, correspondant au cheminement d'un véhicule d'un nœud à un autre, sont très nombreux puisqu'ils partent de tous nœuds $(u, e, id, -1) \in N^{WL}$ et s vers tous nœuds de $(u, e, id, +1) \in N^{WL}$ et p.

Alors, les arcs de E^{WL} peuvent être répartis en trois catégories : les arcs *copies* et *traversants* comme dans le graphe G^{SL}, et les nouveaux arcs *parcourants*.

Hypothèse : *Pour vérifier les contraintes de temporalités, les véhicules prennent toujours le chemin le plus rapide quand ils sont vides.*

Cette hypothèse a pour but de simplifier le modèle en supposant un comportement uniforme des véhicules. Si cette hypothèse est fausse, il suffit de créer plusieurs copies des arcs *parcourants* avec toutes les valeurs Pareto optimales possibles pour ce chemin, mais cela augmente considérablement le nombre d'arcs. Avec cette hypothèse, seul un arc e^{WL} est créé pour le plus court chemin de longueur $L_{e^{WL}}$ (en temps) et de coût $C_{e^{WL}}$.

Pour résumer, un arc $e^{SL} \in E^{SL}$ peut appartenir à l'un des trois groupes suivants (voir figure 14.6) :

- les arcs *copies* : avec $e = (u, v) \in E$
  $(u, e, i, +1) \to (v, e, i, -1)$
- les arcs *traversants* :
  $(u, e, i, -1) \to (u, e', i', +1)$
- les arcs *parcourants* :
  - $s \to (u, e, i, +1)$
  - $(u, e, i, -1) \to p$
  - $p \to s$
  - $(u, e, i, -1) \to (v, e', i', +1)$
    Les arcs *traversants* sont inclus (c'est-à-dire $u = v$ possible).

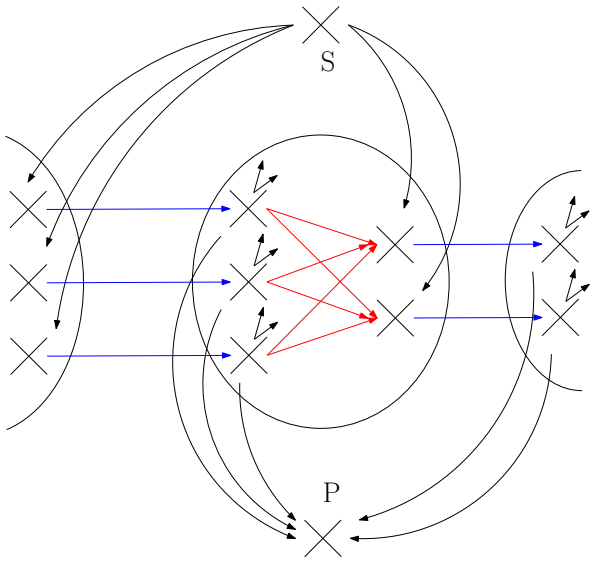

FIGURE 14.6 – Les trois catégories d'arcs : *copies* en bleu, *traversants* en rouge et *parcourants* en noirs (non exhaustif).

Pour rappel, le coût de l'arc (p, s) est C^V qui est le coût unitaire par véhicule utilisé.

14.2.1 Variables de décisions et variables auxiliaires

Ici encore, les variables de décisions associées au flot marchandises sont :

- Y un vecteur d'entiers indexé sur les arcs *copies* représentant la quantité de marchandises passant sur un arc de G.
- Y^* un vecteur d'entiers indexé sur les arcs *traversants* représentant la préemption des marchandises sur un nœud de G.

Ces variables sont décomposables en une somme des vecteurs de chaque marchandise, autrement dit $Y = \sum_{r \in R} Y^r$ et $Y^* = \sum_{r \in R} Y^{r*}$.

Les variables de décisions associées au flot véhicules sont :

- X un vecteur d'entiers indexé sur les arcs *copies* et *parcourants* représentant le parcours des différents véhicules dans le graphe G. Ces variables sont toutes en $\{0, 1\}$ sauf sur l'arc (p, s) pour laquelle elle est entière non bornée.

Il y a, là aussi, une particularité : X est toujours égale à 1 sur les arcs copies puisqu'il doit permettre le transport de toutes les marchandises.

Les variables auxiliaires associées au temps sont :

- T un vecteur de réels indexés sur les nœuds de N^{SL}.

14.2.2 Les Contraintes du Weak Lift

La fonction objectif est plus complexe que pour le *Strong Lift* puisque le coût du transport de marchandises n'est, ici, pas fixe. Cependant, il est plus facile de terminer le nombre de véhicules utilisés en utilisant le flot sur l'arc $e^{WL} = (p, s)$.

$$\alpha_1 \sum_{\substack{e^{WL} \in E^{WL} \\ parcourants}} C_{e^{WL}} X_{e^{WL}} + \alpha_3 C^V X_{(p,s)} \tag{WL.*}$$

Les contraintes associées aux marchandises sont (identiques au *Strong Lift*) :

- Y^* permet la conservation du flot marchandise :

$$\sum_{e^{WL} \in \delta^-(u^{WL})} Y^r_{e^{WL}} \geq \sum_{e^{WL} \in \delta^+(u^{WL})} Y^{r*}_{e^{WL}}, \quad \forall u^{WL} = (u, e, i, -1) \in N^{WL}, \forall r = 0, \ldots, R \tag{WL.1.1}$$

$$\sum_{e^{WL} \in \delta^-(u^{WL})} Y^{r*}_{e^{WL}} \leq \sum_{e^{WL} \in \delta^+(u^{WL})} Y^r_{e^{WL}}, \quad \forall u^{WL} = (u, e, i, +1) \in N^{WL}, \forall r = 0, \ldots, R \tag{WL.1.2}$$

- Y^* permet la satisfaction des demandes :

$$\sum_{\substack{e^{WL} \in \delta^-(u^{WL}) \\ u^{WL} = (u,e,i,-1) \in N^{WL}}} Y^r_{e^{WL}} = \sum_{\substack{e^{WL} \in \delta^+(u^{WL}) \\ u^{WL} = (u,e,i,-1) \in N^{WL}}} Y^r_{e^{WL}} + \max(0, q^r_u), \quad \forall r = 0, \ldots, R \tag{WL.2.1}$$

$$\sum_{\substack{e^{WL} \in \delta^-(u^{WL}) \\ u^{WL} = (u,e,i,+1) \in N^{WL}}} Y^{r*}_{e^{WL}} + \max(0, -q^r_u) = \sum_{\substack{e^{WL} \in \delta^+(u^{WL}) \\ u^{WL} = (u,e,i,+1) \in N^{WL}}} Y^r_{e^{WL}}, \quad \forall r = 0, \ldots, R \tag{WL.2.2}$$

- Y doit se projeter exactement sur y sur tous les arcs porteurs de flot véhicule :

$$\sum_{e^{WL} \text{ copie de } e} Y^r_{e^{WL}} = y^r_e, \quad \forall r = 0, \ldots, R, \forall e \in E \text{ t.q. } x_e > 0 \tag{WL.3}$$

Les contraintes associées au temps sont (identiques au *Strong Lift* sauf pour s et p) :

- La date de la source est égale à 0 et celle du puits doit être inférieur à la date limite :

$$T_s = 0 \tag{WL.4}$$

$$T_p \leq T_{max} \tag{WL.5}$$

- La date sur le nœud d'arrivée d'un arc *copie* doit respectant le temps de trajet de l'arc :

$$T_{v^{WL}} \geq T_{u^{WL}} + L_{e^{WL}}, \quad \forall e^{WL} = (u^{WL}, v^{WL}) \in copies \tag{WL.6}$$

- La date sur le nœud d'arrivée d'un arc *traversant* doit permettre d'attendre toutes les marchandises :

$$T_{v^{WL}} \geq T_{u^{WL}} \quad \text{si} \quad \sum_{r \in R} Y^{*r}_{e^{WL}} > 0, \quad \forall e^{WL} = (u^{WL}, v^{WL}) \in traversants \tag{WL.7}$$

- La date sur le nœud d'arrivée d'un arc *parcourant* doit permettre d'attendre tous les véhicules :

$$T_{v^{WL}} \geq T_{u^{WL}} + L_{e^{WL}} \quad \text{si} \quad x_{e^{WL}} = 1, \quad \forall e^{WL} = (u^{WL}, v^{WL}) \in parcourants \tag{WL.8}$$

Les contraintes associées aux véhicules sont :

- Les véhicules vérifient les lois de Kirchhoff en tout nœud de G^{WL}

$$\sum_{e^{WL} \in \delta^-(u^{WL})} X_{e^{WL}} = \sum_{e^{WL} \in \delta^+(u^{WL})} X_{e^{WL}}, \quad \forall u^{WL} = (u, e, i, \pm 1) \in N^{WL} \tag{WL.9}$$

Pour faciliter la résolution, la contrainte ci-dessous est ajoutée, car un véhicule doit obligatoirement transporter les marchandises sur les arcs copies.

$$X_{e^{WL}} = 1, \quad \forall e^{WL} \in \mathit{copies} \tag{WL.10}$$

- Un véhicule ne peut pas transporter plus que sa capacité :

$$Y_{e^{WL}} < X_{e^{WL}}.CAP, \quad \forall e^{WL} \in \mathit{copies} \tag{WL.11}$$

14.2.3 Un MILP pour résoudre le problème du *Weak Lift*

Un programme linéaire pour résoudre le problème du *Strong Lift* peut donc s'écrire comme suit :
avec M une très grande valeur et $Z \in \{0,1\}$ un vecteur de variables auxiliaires indicé sur les arcs traversants pour les contraintes conditionnelles (WL.7) et (WL.8),

(Weak Lift)
minimiser $\alpha_1 \sum_{\substack{e^{WL} \in E^{WL} \\ \mathit{parcourants}}} C_{e^{WL}} X_{e^{WL}} + \alpha_3 C^V X_{(p,s)}$

tel que

$$\sum_{e^{WL} \in \delta^-(u^{WL})} Y^r_{e^{WL}} \geq \sum_{e^{WL} \in \delta^+(u^{WL})} Y^{r*}_{e^{WL}}, \quad \substack{\forall u^{WL}=(u,e,i,-1) \in N^{WL} \\ \forall r=0,\dots,R} \tag{WL.1.1}$$

$$\sum_{e^{WL} \in \delta^-(u^{WL})} Y^{r*}_{e^{WL}} \leq \sum_{e^{WL} \in \delta^+(u^{WL})} Y^{r}_{e^{WL}}, \quad \substack{\forall u^{WL}=(u,e,i,+1) \in N^{WL} \\ \forall r=0,\dots,R} \tag{WL.1.2}$$

$$\sum_{\substack{e^{WL} \in \delta^-(u^{WL}) \\ u^{WL}=(u,e,i,-1) \in N^{WL}}} Y^r_{e^{WL}} - \max(0, q^r_u) = \sum_{\substack{e^{WL} \in \delta^+(u^{WL}) \\ u^{WL}=(u,e,i,-1) \in N^{WL}}} Y^r_{e^{WL}}, \quad \forall r = 0, \dots, R \tag{WL.2.1}$$

$$\sum_{\substack{e^{WL} \in \delta^-(u^{WL}) \\ u^{WL}=(u,e,i,+1) \in N^{WL}}} Y^{r*}_{e^{WL}} + \max(0, -q^r_u) = \sum_{\substack{e^{WL} \in \delta^+(u^{WL}) \\ u^{WL}=(u,e,i,+1) \in N^{WL}}} Y^r_{e^{WL}}, \quad \forall r = 0, \dots, R \tag{WL.2.2}$$

$$\sum_{e^{WL} \text{ copie de } e} Y^r_{e^{WL}} = y^r_e, \quad \forall r = 0, \dots, R, \forall e \in E \text{ t.q. } x_e > 0 \tag{WL.3}$$

$$T_s = 0 \tag{WL.4}$$

$$T_p \leq T_{max} \tag{WL.5}$$

$$T_{v^{WL}} \geq T_{u^{WL}} + L_{e^{WL}}, \quad \forall e^{WL} = (u^{WL}, v^{WL}) \in \mathit{copie} \tag{WL.6}$$

$$Z_{e^{WL}} \geq \tfrac{1}{M}.Y^*_{e^{WL}}, \quad \forall e^{WL} = (u^{WL}, v^{WL}) \in \mathit{traversants} \tag{WL.7.1}$$

$$T_{u^{WL}} - M.(1 - Z_{e^{WL}}) \leq T_{v^{WL}} \quad \forall e^{WL} = (u^{WL}, v^{WL}) \in \mathit{traversants} \tag{WL.7.2}$$

$$T_{u^{WL}} + L_{e^{WL}} - M.(1 - X_{e^{WL}}) \leq T_{v^{WL}}, \quad \forall e^{WL} = (u^{WL}, v^{WL}) \in \mathit{parcourants} \tag{WL.8}$$

$$\sum_{e^{WL} \in \delta^-(u^{WL})} X_{e^{WL}} = \sum_{e^{WL} \in \delta^+(u^{WL})} X_{e^{WL}}, \quad \forall u^{WL} = (u, e, i, \pm 1) \in N^{WL} \tag{WL.9}$$

$$x_{e^{WL}} = 1, \quad \forall e^{WL} \in \mathit{copies} \tag{WL.10}$$

$$Y_{e^{WL}} < X_{e^{WL}}.CAP, \quad \forall e^{WL} \in \mathit{copies} \tag{WL.11}$$

Le *Weak Lift* a significativement plus de chance de trouver une solution que le *Strong Lift* dans la mesure où, pendant la résolution du P2FP, nous avons vérifié que le flot de marchandises est bien décomposable en chemins *réalisables*. Ainsi, ce modèle s'affranchit des deux problèmes majeurs apportés par la projection :

- Les marchandises ne peuvent être livrées avant d'être récupérées puisque les trajets des véhicules ne sont plus imposés.
- Les temps d'attente ne sont acceptés que si la tournée reste dans l'horizon de temps.

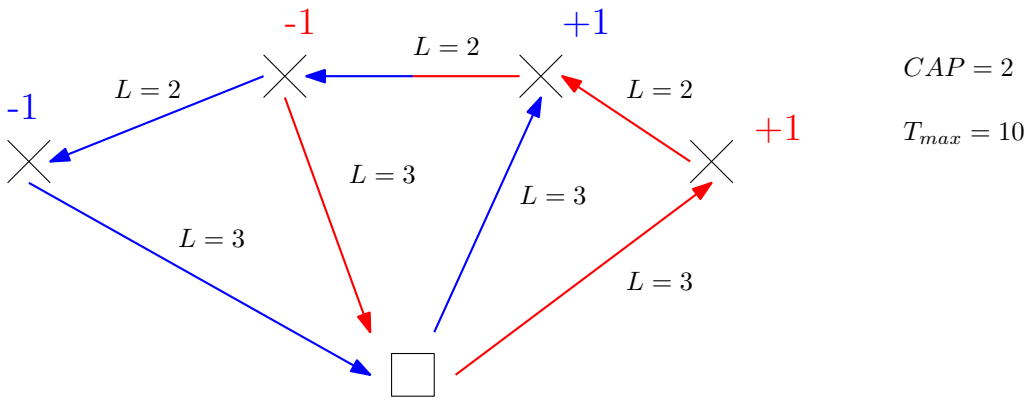

FIGURE 14.7 – Deux passages sont nécessaires sur un arc avec une seule copie.

De plus, nous sommes assurés de l'existence d'une solution temporelle grâce à la décomposition du flot de marchandises. Dans le pire des cas, la solution de ce modèle utilisera autant de camion que de chemins *réalisables* de cette décomposition.

Il reste toutefois une approximation qui nous empêche d'obtenir une solution de ce modèle à coup sûr : le nombre de copies pour chaque arc $e \in E$. En effet, ils sont au nombre de $\lceil \frac{y_e}{CAP} \rceil$ dans notre modèle et cette borne peut entraîner une infaisabilité (voir figure 14.7).

Section 15

Deux méthodes pour la résolution du problème du *Weak Lift*

Cependant, le graphe nécessaire au programme linéaire à un nombre d'arcs important et la résolution, bien que plus rapide que la résolution directe du TE2FP, reste lente. Il est nécessaire de développer des méthodes pour résoudre efficacement le *Lift problem*. Nous en proposons deux : l'une part du graphe utilisé pour la résolution exacte et l'autre explore le graphe temporel à la manière d'une simulation.

15.1 Première méthode : remplacer le programme linéaire

Reprenons le problème du *Weak Lift* depuis le début : la solution optimale du P2FP nous donne un flot de marchandises agrégé qui est la projection du flot de marchandise recherché comme solution du PRP. Ainsi, nous partons du flot y de marchandises de la solution projetée et du graphe G^{WL} comme présenté dans la section précédente et nous souhaitons construire les tournées datées des véhicules transportant les marchandises en respectant y.

Basée sur le même graphe que le programme linéaire présenté dans la section précédente, cette méthode permet de trouver une solution réalisable rapidement en décomposant la résolution en deux parties. La première est la décomposition du flot de marchandises en distinguant les différentes unités de flot à l'intérieur de chaque nœud (c'est-à-dire sur les arcs *traversants*). Cette étape permet de calculer des fenêtres de temps $[\tau_u^-(e), \tau_u^+(e)]$ sur chacun des arcs *copies* e ainsi qu'une relation d'ordre $e < e'$ entre les arcs *copies*. Cette relation indique que l'arc e doit être visité avant l'arc e' mais n'oblige pas qu'elle soit respectée par le même véhicule. Puis vient la seconde étape qui est la résolution d'un problème de *Vehicle Routing Problem with Time Windows* (VRPTW) avec relation d'ordre pour construire les tournées des véhicules. Cette relation n'a pas été étudiée conjointement au VRPTW dans la littérature, et une méthode type *Insertion-Removal* ou un algorithme glouton du VRPTW peut être adapté pour notre problème.

15.1.1 Décomposition du flot de marchandises

Dans un premier temps, il faut identifier les différentes unités de marchandises pour pouvoir les affecter à des tournées. Il n'est pas nécessaire de faire une décomposition en flots unitaires puisque certaines marchandises vont suivre le même trajet, mais pour pouvoir affecter les différentes marchandises aux véhicules, il faut les désagréger. Bien que, pendant la résolution du P2FP, des coupes nous assurent de la décomposabilité du flot de marchandises, aucune décomposition n'est récupérée avec la solution. De plus, même s'il est possible d'en récupérer une, elle traite les différents types de marchandises séparément alors qu'un seul passage peut permettre de transporter plusieurs types de marchandises en même temps.

En partant du flot projeté de marchandises y, nous souhaitons donc construire le flot de marchandises sur le graphe G^{WL}. Celui-ci est en deux parties :

- Le flot de marchandises Y_{e_i} circulant sur les arcs copies e_i et dont la somme du flot sur toutes les copies d'un même arc e est égale au flot projeté y_e.

- Le flot de marchandises $Y^*_{e_t}$ circulant sur les arcs traversants e_t à l'intérieur d'un nœud qui permet de construire la décomposition, la relation d'ordre entre les arcs copies et les fenêtres de temps.

Trouver une décomposition du flot de marchandise, c'est être capable de suivre chaque marchandise de son origine à une de ses destinations tout en respectant le flot projeté dans la solution du P2FP. La difficulté est, comme expliqué dans la section précédente, de savoir ce qu'il se passe à l'intérieur d'un nœud. Autrement dit, chaque unité de marchandise entrante en un nœud doit être affecté à une unité sortante. De cette affectation découlera une relation d'ordre entre les arcs *copies*, les fenêtres de temps sur chacun de ces arcs et nous permettra de propager les unités de flots sur les arcs sortants. L'affectation sur un nœud dépend donc de l'affectation des nœuds en amont de celui-ci.

Pas question cependant d'essayer de trouver la solution sur tous les nœuds en même temps, mais plutôt de résoudre nœud par nœud pour trouver des solutions locales en un temps de calcul raisonnable, puis d'itérer jusqu'à trouver une solution réalisable globalement. Mais cela n'est possible qu'en connaissant le flot entrant au nœud considéré. Pour résoudre ce problème, nous proposons de réaliser l'affectation en partant des nœuds ayant du surplus de marchandises en ne tenant pas compte des autres entrées (puisqu'elles ne sont pas encore connues) et de réaliser l'affectation des nœuds en suivant la propagation des marchandises. Cela implique que l'affectation sera effectuée plusieurs fois sur les nœuds jusqu'à ce que toutes les valeurs du flot projeté y soient respectées. Nous proposons de réaliser l'affectation en suivant un parcours d'arbre en largeur initialisé avec les nœuds avec un surplus de marchandises et agrandit si un des flots sortants est différent de la précédente affectation.

Ainsi, nous allons nous concentrer sur le **problème de l'affection des marchandises au sein d'un nœud u**. L'idée est que chaque marchandise entrant en u (connue) doit être affecté à une sortie respectant le flot de marchandises de la solution du P2FP. L'affectation d'une unité de marchandise d'un arc *copie* entrant e à un arc *copie* sortant e' implique la relation d'ordre $e < e'$ (e doit être visité avant e') et les fenêtres de temps doivent être propagées en conséquence. À la fin de cette étape, toute marchandise doit pouvoir être suivi de son origine, à travers chaque nœud et jusqu'à l'une de ses destinations.

Affecter efficacement les marchandises entrantes en un nœud u aux arcs sortants de ce nœud, c'est affecter en prenant en compte les dates de passages des marchandises. Autrement dit, il faut s'assurer de la réalisabilité de l'affection dans l'horizon de temps. Nous proposons de construire une méthode sur le principe suivant : les marchandises pouvant arriver en premier seront redirigées sur les chemins nécessitant le plus de temps de trajet. Cela vaut aussi pour les marchandises en surplus (qui sont donc les premières quoi qu'il arrive) et les demandes (qui doivent être satisfaites le plus tôt possible). Avec cette méthode, nous essayons de garder les fenêtres de temps $[\tau^-_u(e), \tau^+_u(e)]$ les plus larges possible en affectant les entrées avec la plus faible valeur $\tau^-_u(e)$ aux sorties avec la plus faible valeur $\tau^+_u(e)$. La figure 15.1 montre un exemple d'affectation avec deux types de marchandises différents. Nous affectons alors les marchandises dans l'ordre croissant des valeurs $\tau^-_u(e)$ et dans l'ordre lexicographique quand plusieurs entrées ont la même valeur. Nous présentons un pseudo-code de cette idée dans l'algorithme 15.1.

Les bornes inférieures et supérieures des fenêtres de temps sont propagées *en pire cas* pendant l'affectation des marchandises. Ainsi, si deux arcs e et e' entrant au nœud u, avec leur valeur respective $\tau^-_u(e)$ et $\tau^-_u(e')$, sont affectés à la même sortie e'', alors $\tau^-_u(e'') = \max(\tau^-_u(e), \tau^-_u(e'))$ (voir figure 15.2). Et, de la même façon, si un arc e entrant au nœud u est affecté à deux arcs e' et e'' sortant de u avec leur valeur respective $\tau^+_u(e')$ et $\tau^+_u(e'')$, alors $\tau^+_u(e) = \min(\tau^+_u(e'), \tau^+_u(e''))$ (voir figure 15.3). Les fenêtres de temps sont initialisées par la longueur du plus court chemin du dépôt au nœud u arrivant par l'arc e pour $\tau^-_u(e)$, et, de façon similaire, l'horizon de temps auquel est soustraite la longueur du plus court chemin depuis le nœud u jusqu'au dépôt partant par l'arc e pour $\tau^+_u(e)$. La propagation des fenêtres de temps après l'affectation des marchandises sur l'instance de la figure 15.1 est présentée dans la figure 15.4

Une fois que les marchandises ont été affectées localement en utilisant l'algorithme présenté ci-dessus, les fenêtres de temps reflètent la réalisabilité de la décomposition du flot de marchandises trouvé. Si toutes les fenêtres de temps sont non vides, la décomposition est valide et la méthode d'affectation s'arrête là. Sinon, un nœud, dont l'une des fenêtres de temps est impossible, est choisi et l'affection de ce nœud est recalculée avec les nouvelles fenêtres de temps. Cette procédure est répétée jusqu'à l'obtention d'une

décomposition réalisable.

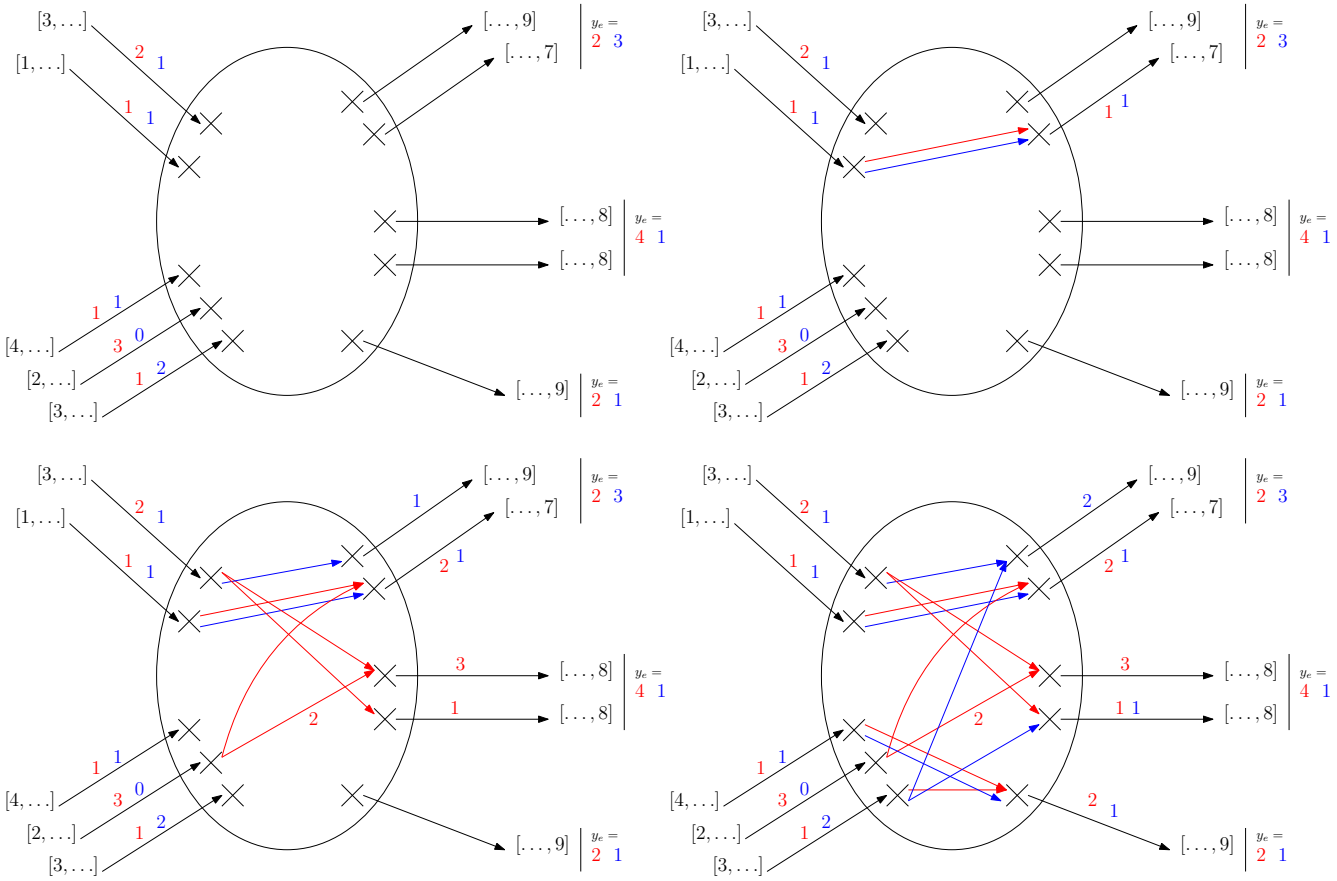

FIGURE 15.1 – Exemple d'affectation, avec $CAP = 3$, des entrées d'un nœud à ses sorties en suivant la règle proposée. Seules les étapes 0, 1, 3, et 5 sont présentées ici.

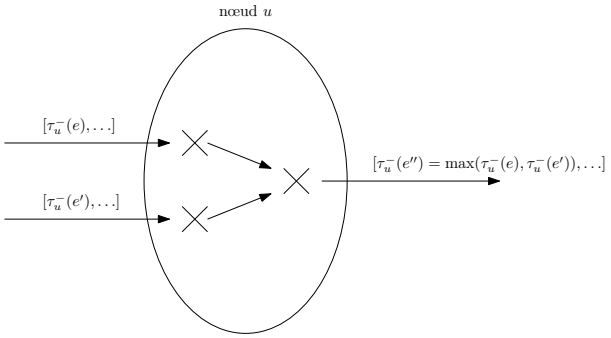

FIGURE 15.2 – Propagation des valeurs $\tau_u^-(e)$.

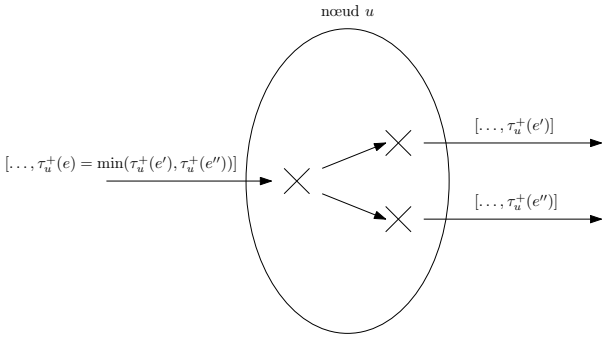

FIGURE 15.3 – Propagation des valeurs $\tau_u^+(e)$.

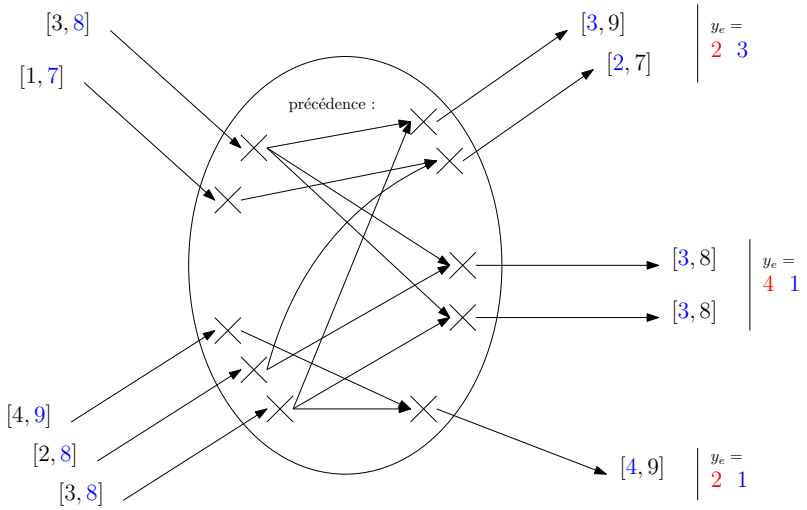

FIGURE 15.4 – Le résultat de la propagation des fenêtres de temps sur l'exemple d'affectation figure 15.1.

Algorithme 15.1 : Affectation des marchandises au sein d'un nœud u

Entrées : les valeurs de fenêtres de temps initiales $[\tau_u^-(e), \tau_u^+(e)]$
Les valeurs du flot y sur les arcs de E dans la solution projeté
Début
Pour tout arc *copie* entrant e_i dans l'ordre des $\tau_u^-(e_i)$ croissant **Faire**
Pour $r = 1, \ldots, R$ **Faire**
Soit e'_j le premier arc *copie* sortant, dans l'ordre des $\tau_u^+(e'_j)$ croissant, tel que :
la quantité de marchandises $y^r_{e'}$ n'est pas satisfaite et l'arc *copie* e'_j n'est pas saturé
c'est-à-dire $\sum_j Y^r_{e'_j} < y^r_{e'}$ et $\sum_{r=1,\ldots,R} Y^r_{e'_j} < CAP$
Affecter le maximum possible : $q = min(y^r_{e'} - \sum_j Y^r_{e'_j}, CAP - \sum_{r=1,\ldots,R} Y^r_{e'_j})$
Affecter q marchandises de type r à l'arc *traversant* $e_t = (u, e, i, -1) \to (u, e', j, +1)$:
$Y^{r*}_{e_t} = Y^{r*}_{e_t} + q$
$Y^r_{e'_j} = Y^r_{e'_j} + q$
Ajouter la relation d'ordre $e_i < e'_j$
Calculer les nouvelles fenêtres de temps de e_i et e'_j.
Fin Pour
Fin Pour
Propager les fenêtres de temps qui ont été modifiées de chaque côté par deux parcours en largeur.
(un parcours pour $\tau_u^-(e)$ et un parcours pour $\tau_u^+(e)$).
Retourner Les valeurs de flots Y en sortie et Y^* à l'intérieur du nœud, ainsi que la relation d'ordre et les fenêtres de temps.
Fin

15.1.2 Résolution du VRPTW avec relation d'ordre

Les problèmes VRPTW ont été largement étudiés, mais notre problème est une variante très spécifique puisqu'elle impose des relations de précédence sur les clients (les arcs *copies*), même lorsque ces clients ne sont pas pris en charge par les mêmes véhicules. De plus, la fonction objectif étant le reflet de celle du PRP, elle implique à la fois le nombre de véhicules et un coût économique des tournées. Cela veut donc dire que nous ne connaissons pas a priori le nombre de tournées que nous devons construire, ce qui rend le VRPTW avec relation d'ordre plus difficile encore.

Nous proposons de résoudre ce problème grâce à une méthode gloutonne qui pourra être rendu aléatoire pour être utilisé dans un schéma de type GRASP (*Greedy Randomized Adaptive Search Procedure*). L'idée de cette méthode gloutonne est basée sur les méthodes d'insertions bien connues pour résoudre les problèmes de VRP et VRPTW (voir [Janssens et al., 2015, Lu and Dessouky, 2006]). Dans le cas du problème du voyageur de commerce (*Traveling Salesman Problem* - TSP), ces méthodes partent d'une tournée vide et insèrent les clients itérativement dans cette tournée. Dans notre cas, les clients sont les arcs *copies* puisqu'ils correspondent aux marchandises qui doivent être transportées. Nous cherchons donc à construire des tournées permettant de servir tous les clients (transporter toutes les marchandises) en respectant leur fenêtre de temps et la relation d'ordre entre eux. Les tournées doivent revenir au dépôt dans l'horizon de temps et permettent de minimiser la fonction objectif équivalente à celle du PRP.

Les deux difficultés des méthodes d'insertions sont le choix du client à insérer et l'endroit, dans la tournée, où l'insérer. Il existe plusieurs réponses à ces deux questions :

- Le choix du prochain client à insérer peut être fait : de façon arbitraire, en choisissant celui qui minimise le coût d'insertion à l'itération considérée ou encore, en choisissant le client le plus proche d'une tournée en cours par exemple.
- L'emplacement où insérer le client choisi peut être, lui aussi, choisi arbitrairement (peu efficace) ou être l'emplacement minimisant le coût à l'itération considérée par exemple.

Notre problème possède cependant des informations supplémentaires que nous pouvons utiliser. Les clients sont, dans notre cas, les arcs copies puisqu'il s'agit des marchandises à transporter et l'ensemble de ces arcs possède une relation d'ordre à respecter. Ainsi, le champ de recherche du prochain client et de l'emplacement dans la tournée est considérablement réduit :

- Tous les clients sont envisageables, mais certains clients doivent être visités avant certains autres à cause de la relation d'ordre. Traiter ces clients en priorité permet de minimiser les vérifications pendant l'étape d'insertion. Nous proposons que le prochain client à insérer soit un client minimal au sens de la relation d'ordre (c'est-à-dire un client e tel qu'il n'existe pas de client non inséré e' qui vérifie $e' < e$).
- Les emplacements possibles pour de tels clients sont : dans une nouvelle tournée, à la fin d'une tournée existante (puisque c'est une position toujours valide), ou entre deux clients e et e' d'une même tournée tels que $e < e$ et $e' \not< e$. Cette dernière possibilité peut être vérifiée rapidement en rebroussant la tournée jusqu'à la première position qui ne vérifie pas cette condition puisqu'aucun autre, avant celle-ci, ne pourra la vérifier. Toutefois, pour garder un algorithme efficace, nous ne vérifierons que les deux premières possibilités (la création d'une nouvelle tournée et l'ajout à la fin d'une tournée existante). Nous proposons alors d'insérer le nouveau client à l'emplacement valide (qui respecte les fenêtres de temps) qui minimise le coût d'insertion.

La méthode que nous proposons se résume donc à : tant qu'il reste des arcs *copies* à insérer, sélectionner un arc minimal au sens de la relation d'ordre, vérifier les emplacements possibles en calculant le coût en cas d'insertion, insérer à l'emplacement de coût minimal, propoger les nouvelles fenêtres de temps et répéter jusqu'à ce que tous les arcs *copies* aient été insérés dans une tournée (voir Algorithme 15.2).

Cette méthode gloutonne peut être rendu aléatoire de plusieurs façons : l'instruction (I1) peut facilement être dérivée pour devenir aléatoire et l'instruction (I2) peut être assouplie pour sélectionner aléatoirement parmi les deux ou trois choix de coût minimum par exemple. De cette façon, il est possible d'exploiter cette procédure dans un schéma algorithmique de type GRASP.

Algorithme 15.2 : méthode gloutonne d'insertion

Entrées : Les fenêtres de temps induites par la relation d'ordre sur les nœuds
Début
Initialiser une première tournée $\{(u_d, t = 0)\}$
Tant que il existe un arc *copie* de E^{WL}, non affecté à une tournée **Faire**
Soit e un tel arc, minimal au sens de la relation d'ordre (I1)
Calculer les coûts associés à l'ajout de e à la fin des tournées actuellement en construction (quand l'insertion est valide)
Calculer le coût associé à la création et l'ajout de e à une nouvelle tournée
Choisir l'option de coût minimum respectant les fenêtres de temps (I2)
Propager la nouvelle valeur $\tau_u^-(e)$ associée à l'insertion de e par un parcours en largeur
Fin Tant que
Retourner Les tournées ainsi créées et la valeur objective de cette solution
Fin

15.2 Seconde méthode basée simulation

L'idée de cette méthode est de simuler les véhicules sur le graphe G et les faire évoluer pour résoudre le *Weak Lift*. Ainsi, nous partons des flots de véhicules x et de marchandises y de la solution projetée et nous souhaitons construire les tournées datées des véhicules transportant les marchandises respectant y. Ne sachant pas s'il est possible de réaliser plusieurs tournées avec un même véhicule, nous considérerons que la valeur totale de flot x sortant du dépôt correspond au nombre de véhicules total. Cette méthode se concentre d'abord sur le problème du *Strong Lift* puisque celui-ci est plus fort que le *Weak Lift* et est donc plus simple à résoudre ici, car les choix sont plus limités. Puis, si le *Strong Lift* est impossible à résoudre, le flot de véhicules x de la solution projetée est augmenté et la méthode est relancée avec ce nouveau flot. Cette deuxième exécution permet donc de chercher une solution au *Weak Lift* lorsque le *Strong Lift* n'est pas possible.

Cependant, malgré le choix de résoudre le *Strong Lift* pour restreindre les possibilités, une projection peut encore correspondre à plusieurs solutions temporelles. Il faudra, de plus, faire de bons choix pour ne pas bloquer la simulation et avoir des chances de trouver une solution temporelle. Notamment, si un véhicule a le choix entre plusieurs routes (toutes respectant la projection), laquelle doit-il prendre ? Dans cette section, nous proposons des critères pour trier les routes permettant également d'équilibrer la durée des différentes tournées.

15.2.1 Simulation de tournées de véhicules

Pour résoudre le problème du *Strong Lift*, c'est-à-dire la reconstruction les tournées complètes avec les dates de passages et les durées d'attentes dont la projection est exactement la solution du problème projeté, nous proposons une méthode qui se concentre sur les trajets des véhicules et y intègre les flux de marchandises. La méthode est alors adaptée pour continuer lorsque la simulation se bloque, autrement dit l'algorithme résout le problème du *Weak Lift* si le problème du *Strong Lift* est impossible.

Un état du système

L'espace dans lequel évolue la simulation se décompose en 2 parties :

- L'espace temporel : c'est un axe sur lequel avancent les différents véhicules au cours de la simulation.
- L'espace physique : il correspond à l'emplacement des véhicules ainsi que les flots restant des véhicules et des marchandises qui n'ont pas encore été traitées.

Il est important de noter que le graphe va rétrécir à chaque décision prise pendant la simulation puisqu'il porte les arcs qui n'ont pas encore été traités.

Un état du système, dans cet espace, est alors composé de cinq vecteurs :

- $\left((u_i, t_i, (c_i^k)_{k=\{1,\dots,K\}})\right)_{i=\{1,\dots,x_{ps}\}}$: Un vecteur de triplets indexé sur les véhicules correspondant à la position, la date actuelle et la quantité des différentes marchandises dans le chargement de chaque véhicule.
- $\left((q_u^k)_{k=\{1,\dots,K\}}\right)_{u\in N}$: Un vecteur de vecteurs indexé sur les nœuds dans lequel figure la quantité de toutes les marchandises disponibles (si quantité positive) ou demandées (si quantité négative) de chaque nœud.
- $(x_e)_{e\in E}$: Un vecteur indexé sur les arcs dans lequel figure le nombre restant de passages de véhicules de chaque arc.
- $\left((y_e^k)_{k=\{1,\dots,K\}}\right)_{e\in E}$: Un vecteur de vecteurs indexé sur les arcs dans lesquels figurent les quantités restantes de marchandises à chaque arc.

Un état de la simulation peut donc ressembler à la figure 15.5.

Une décision en deux temps

Faire avancer la simulation, c'est faire se déplacer les véhicules pour qu'ils transportent les marchandises. Faire une itération de la simulation, c'est donc faire avancer un véhicule le long d'un arc. Une décision se compose alors de deux parties : choisir un véhicule et choisir un arc sur lequel il va se déplacer. Dans l'état de la simulation présenté dans la figure 15.5, une décision peut être le véhicule 2 et l'arc (0,1) (voir figure 15.6).

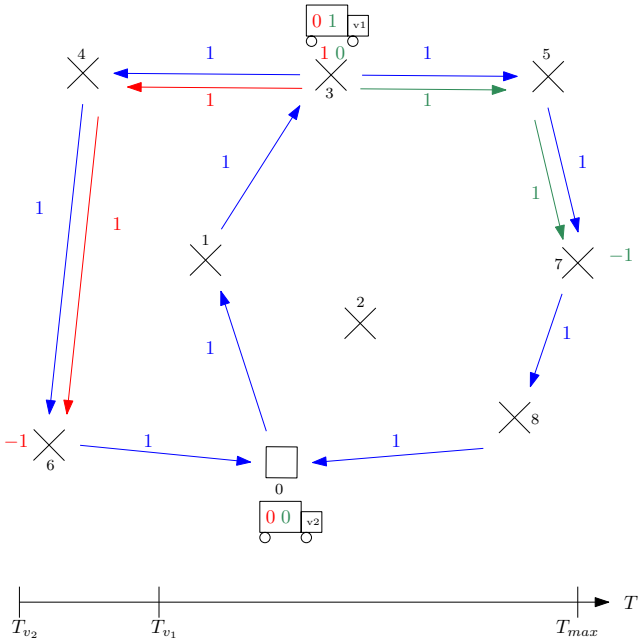

FIGURE 15.5 – Un exemple d'un état de la simulation sur l'instance présentée figure 15.16.

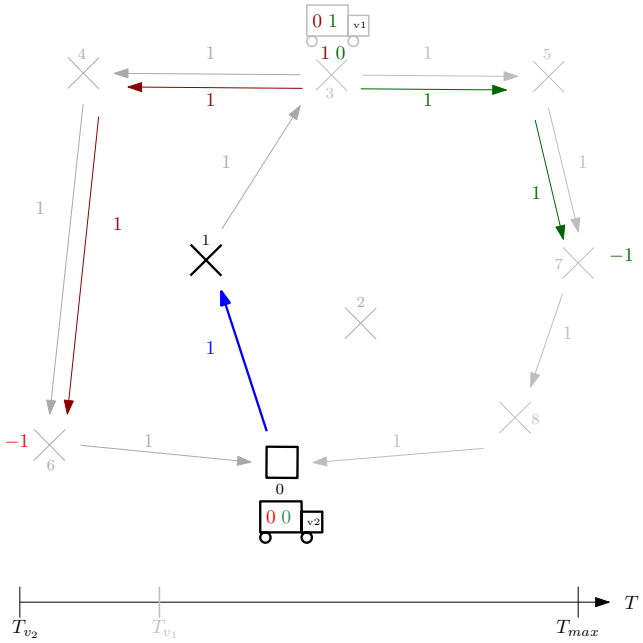

FIGURE 15.6 – Un exemple d'une décision à partir de l'état présenté figure 15.5.

Le choix du véhicule se fait en considérant le premier véhicule dans l'ordre temporel (c'est-à-dire le véhicule i^* tel que $t_{i^*} = \min_{i=\{1,\dots,x_{ps}\}} t_i$). Il n'est pas possible de choisir un véhicule ne respectant pas cette condition, car cela permet d'être certain qu'un véhicule est bloqué (aucun véhicule ne peut le débloquer avant sa date). Ce choix nous permet de toujours avoir un état réalisable de la simulation sans jamais avoir à remettre en cause nos décisions. Ces dernières peuvent ne pas être optimales, mais l'état résultant de cette décision est toujours possible. Cette condition sur le choix du véhicule permet également d'équilibrer les tournées en empruntant prioritairement les chemins les plus longs afin de laisser aux véhicules tardifs les arcs retournant au dépôt. Si plusieurs véhicules sont possibles, le choix peut être fait arbitrairement

puisque les véhicules sont identiques.

Une fois le véhicule choisi, le choix de l'arc se fait parmi tous les arcs sortant du nœud dans lequel est stationné le véhicule. Les arcs sont considérés dans l'ordre décroissant de leur distance au dépôt (la longueur de l'arc additionnée à la longueur du plus court chemin entre le nœud d'arrivé et le dépôt). Nous proposons de choisir le premier arc qui ne soit pas impossible à traverser : respecter le *Strong Lift* c'est respecter exactement (ni plus ni moins) le flot de marchandises avec exactement le nombre de passages donné par le flot de véhicule. Cela impose, pour qu'un véhicule puisse traverser un arc, une borne supérieure des marchandises qu'il peut faire transiter sur l'arc, mais également une borne inférieure. Par exemple, si les véhicules ont une capacité de 1 et que 3 unités de marchandises doivent être transportées par 3 véhicules, chacun de ces derniers doit transporter 1 unité pour que le *Strong Lift* soit respecté. Nous avons donc deux conditions sur le choix d'un arc e : une borne supérieure (équation 15.1) et une borne inférieure (équation 15.2) sur la quantité de marchandises $c^k_{i^*}$ à transporter, en plus de la contrainte de capacité des véhicules.

$$\sum_{K} c^k_{i^*} \leq y^k_e, \ \forall k = 0, \ldots, K \tag{15.1}$$

$$\sum_{k=0} y^k_e - c^k_{i^*} \leq CAP(x_e - 1), \tag{15.2}$$

Si la borne supérieure n'est pas respectée, le véhicule peut déposer des marchandises sur le nœud sur lequel il est actuellement. Si la borne inférieure n'est pas respectée, le véhicule doit récupérer des marchandises du nœud ou des autres camions qui sont sur le même nœud à la même date que lui. Si aucun arc n'est possible (le véhicule ne peut respecter la borne inférieure d'aucun arc sortant), le véhicule se met en attente : un autre véhicule doit venir sur le même nœud que lui avec des marchandises qui lui permettront de se débloquer.

Transition d'un état à l'autre

Une transition démarre après avoir choisi un véhicule et un arc. Pour que le déplacement soit possible, le véhicule doit récupérer une quantité de marchandises respectant deux conditions : la quantité de chaque marchandise ne doit pas dépasser celui prévu sur l'arc et le reste des marchandises prévues doivent pouvoir être acheminées par les véhicules restants.

- Dans la Figure 15.7, les véhicules ont une capacité de 3. Le véhicule ne peut prendre qu'un maximum de 2 unités de marchandises actuellement, mais les 4 unités restantes ne pourront pas être transportées par un seul véhicule si celui-ci traverse l'arc avec seulement 2 unités de marchandises. Le déplacement est donc impossible à cette date au sens du *Strong Lift*.

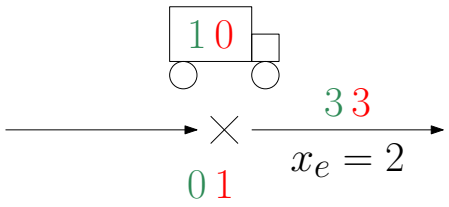

FIGURE 15.7 – Déplacement multiple impossible à cette date : le véhicule se met en attente.

- Si le reste des marchandises est transportable par les traversées restantes, la transition se fait comme sur la Figure 15.8.

FIGURE 15.8 – Déplacement multiple possible à cette date et transition sur l'arc.

Bien sûr, cette règle s'applique également s'il ne reste qu'un véhicule devant traverser l'arc et, dans ce cas, toutes les marchandises prévues, et seulement elles, doivent être récupérées par le véhicule avant sa traversée dans le cas du *Strong Lift* (voir Figure 15.9 et Figure 15.10).

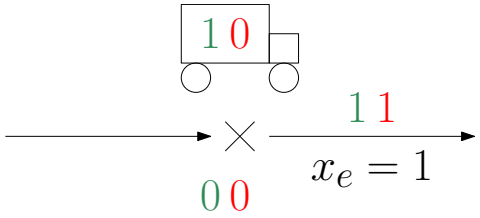

FIGURE 15.9 – Déplacement simple impossible à cette date : le véhicule se met en attente.

FIGURE 15.10 – Déplacement simple possible à cette date et transition sur l'arc.

Une fois l'arc traversé, les véhicules résolvent la demande de marchandises sur le nœud d'arrivée (voir Figure 15.11).

FIGURE 15.11 – Résolution de la demande de marchandises d'un nœud.

Pour continuer sur notre exemple, à partir de la décision présenté figure 15.6, la transition est un déplacement simple possible, et se traduit par le nouvel état présenté figure 15.12.

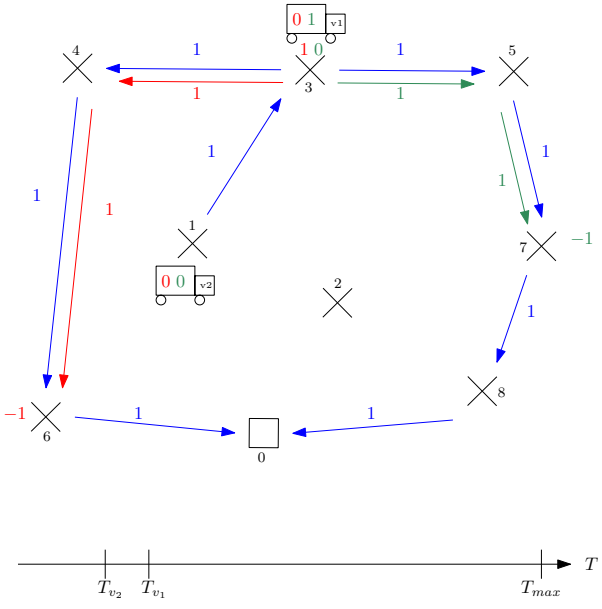

FIGURE 15.12 – Etat résultant de la décision présenté figure 15.6.

Aussi, comme dit précédemment, si aucun arc n'est possible à partir du véhicule sélectionné, alors ce dernier se met en attente. Cela veut dire que personne avant lui n'a pu résoudre son problème et il a besoin d'attendre l'arrivée d'un autre véhicule pour qu'il puisse être débloqué. Pour ce faire, le véhicule ne sera plus considéré lors du choix du prochain véhicule à faire avancer. Il ne pourra être débloqué que lorsqu'un autre véhicule arrive sur le même nœud que celui sur lequel il est. Lorsque cela se produit, le véhicule précédemment en attente sera réintroduit dans la simulation avec la même date que celle du deuxième véhicule qui vient d'arriver sur son nœud.

Choix de l'arc sortant quand plusieurs sont possibles

Plusieurs comparaisons sont envisageables pour prioriser un arc par rapport à un autre. Cependant, ce choix a de grandes conséquences et peut mener à un blocage de la méthode. Un blocage potentiel peut venir du fait qu'un véhicule décide de prendre un chemin ne lui permettant pas de revenir au dépôt avant la fin de l'horizon de temps. Nous proposons alors de faire prendre le chemin le plus long vers le dépôt au véhicule arrivant en premier, puis le deuxième plus long au deuxième véhicule et ainsi de suite.

Pseudo-code de la méthode proposé

La simulation se déroulera donc de la manière suivante (voir algorithme 15.3) :

- Le véhicule choisi au début de chaque itération est le premier dans l'ordre temporel (c'est-à-dire le véhicule i^* tel que $t_{i^*} = \min_{i=\{1,\dots,x_{ps}\}} t_i$) qui ne soit pas en attente.
- Si aucun arc, en partant du nœud courant du véhicule choisi, n'est possible (voir règles 15.7 et 15.9), mettre en attente le véhicule sur le nœud courant.
- Choisir un arc possible. Si au moins deux tournées sont possibles, choisir celle qui part le plus loin du dépôt, puis appliquer la règle 15.10 ou 15.8 sur ce chemin.
- Satisfaire les demandes sur le trajet et prendre les surplus de marchandises quand c'est possible.
- Appliquer la règle de résolution des demandes 15.11 sur le nœud d'arrivée.
- Si un véhicule arrive sur un nœud où des véhicules sont en attentes, débloquer ces derniers à la date du véhicule venant d'arriver.
- La simulation se termine lorsque tous les véhicules sont en attentes ou que tous les arcs aient été visités.

15.2.2 Adaptation pour le problème du *Weak Lift*

Deux vérifications sont nécessaires à cette étape : si certains véhicules ne sont pas bloqués mais n'ont pas pu rentrer au dépôt par manque de temps et si certains véhicules sont encore en attente et n'ont pas pu être débloqués.

Dans le premier cas, cela signifie qu'il manque un véhicule. Nous ajoutons alors un véhicule en augmentant la valeur de x_{ps} de 1.

Dans le deuxième cas, nous proposons trois façons de débloquer les marchandises qui n'aurait pas été transporté par la simulation. L'une d'entre elles est une méthode d'insertion dans les tournées précédemment créées. Les deux autres modifient les valeurs de la solution projetée et nécessitent de relancer la méthode du *Strong Lift* sur ces nouvelles valeurs.

Une insertion à postériori

Une fois la méthode terminée, des tournées datées sont proposées. Cependant, certains arcs porteurs de flot véhicules x et de flot marchandises y n'ont pas été insérés dans une tournée et forme un cycle puisque x respecte les lois de Kirchhoff.

L'idée est d'insérer tout ou partie des cycles d'arcs porteurs de flots à l'intérieur des tournées passant grâce à une méthode d'insertion ne dépendant pas du flot de véhicule.

Algorithme 15.3 : méthode Lift

Entrées :

$(q_u^k)_{k=\{1,\dots,K\}}$ les quantités de marchandises disponibles ou demandées par le nœud u
x_e le nombre de passages sur l'arc e
$(y_e^k)_{k=\{1,\dots,K\}}$ les quantités de marchandises nécessaires pour l'arc e
$Distances_e$ la distance du plus court chemin de l'arc e au dépôt

Début

$(Attentes_u)_{u\epsilon N}$ est un vecteur de files d'attentes vides indicées sur les nœuds du graphe G.
Initialiser tous les véhicules au dépôt et à la date 0.

$FIN = Faux$
Tant que $FIN == Faux$ **Faire**
i^* est l'indice du premier véhicule (u_{i^*}, t_{i^*}) dans l'ordre temporel ($t_{i^*} = \min_{i=\{1,\dots,x_{ps}\}} t_i$)
qui ne soit pas en attente
Si plusieurs véhicules vérifient la condition, prendre dans l'ordre lexicographique parmi eux.

Si Aucun véhicule n'est trouvé (car tous en attente) **Alors**
$FIN = Vrai$
Sinon
Choisir l'arc e^* partant de u_{i^*} qui maximise
la distance au dépôt parmi les arcs partant de u_{i^*} et dont la transition est possible :
e^* est tel que $x_{e^*} > 0$ et $Distances_{e^*} = \min_{e=(u,v)\epsilon E,\ u=u_{i^*},\ e\ possible} Distances_e$

«e possible» $\Leftrightarrow \sum_k y_e^k - \min(CAP, \sum_k q_u^k + c_u^k) \leq (x_{tp} - 1) \times CAP$

Si Un tel arc est trouvé **Alors**
Parcourir l'arc e^* :
$x_{e^*} = x_{e^*} - 1$
Échanger des marchandises entre véhicules et en prendre ou en laisser sur le nœud
afin d'avoir $\max(CAP, y_{e^*})$ marchandises au total
Mettre à jour la date du véhicule i^* et déposer les marchandises qui sont en demandes en v.
Les véhicules dans la file d'attente $Attente[v]$ sont débloqués à la date d'arrivée de i^*.
Sinon
Ajouter le véhicule i^* à $Attentes[u^*]$.
Fin Si
Fin Si

Fin Tant que
$BLOQUE = Faux$
Si L'une des files d'attentes $Attente[u]$ est non vide **Alors**
$BLOQUE = Vrai$
Fin Si
Retourner La valeur de $BLOQUE$ et les tournées ainsi créées.
Fin

Cette méthode ajoute donc au minimum deux unités de flot de véhicule et en retire au minimum deux également (voir Figure 15.13).

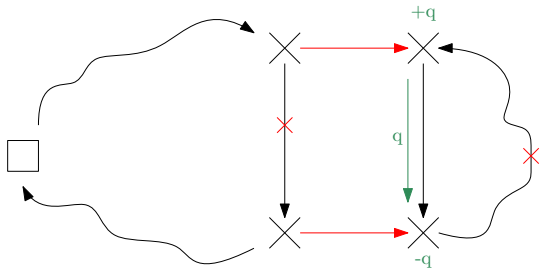

FIGURE 15.13 – Insertion à postériori d'une partie d'une tournée porteuse de flot marchandise dans une tournée passant par le dépôt.

Cette méthode a besoin de faire deux décisions :

- Quel cycle non desservi s'associe avec quelle tournée ?
- Quels sont les quatre points de jonction entre le cycle et la tournée ?

Modification du départ d'une tournée

Si une certaine marchandise n'a pu être transporté, c'est qu'aucun véhicule n'a pu atteindre son nœud de départ. L'une des tournées est alors forcée de commencer sur ce nœud en ajoutant une unité de flot de véhicule sur tous les arcs du plus court chemin entre le dépôt et le nœud considéré (voir Figure 15.14). Puis, la méthode du *Strong Lift* est relancé avec cette nouvelle solution.

Cependant, pour respecter la conservation du flot, il faut supprimer une valeur de flot parmi les autres arcs sortants du dépôt. Laquelle et pour quelles raisons ?

Ajout de flot sur les arcs porteurs de flot marchandises

L'idée est que le nœud de départ de la marchandise est peut-être atteignable en passant par les arcs porteurs de ce même flot. Le véhicule doit donc y passer deux fois (une fois pour atteindre le nœud de départ, et une autre pour faire transiter la marchandise). Il faut donc ajouter une unité de flot véhicule à tous les arcs porteurs de cette marchandise ainsi qu'une unité de flot sur les arcs du plus court chemin entre le nœud de fin et de départ de la marchandise pour vérifier les lois de Kirchhoff (voir Figure 15.15). Puis, la méthode du *Strong Lift* est relancé avec cette nouvelle solution.

C'est cette dernière solution qui a été choisie et implémentée.

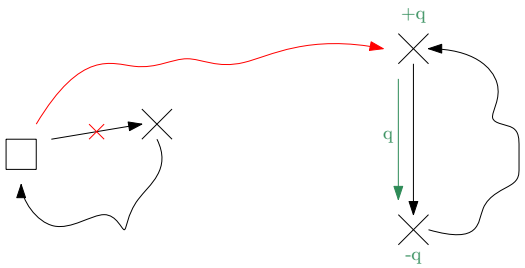

FIGURE 15.14 – Résolution de la demande de marchandises d'un nœud.

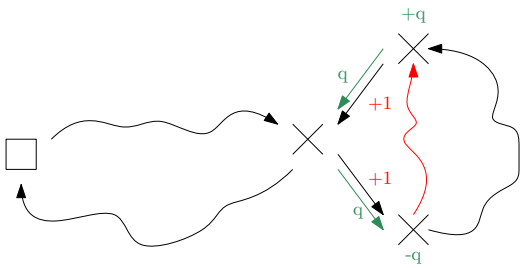

FIGURE 15.15 – Ajout de flot véhicule sur les arcs porteurs de la marchandise.

15.2.3 Simplification du graphe en «tournées partielles»

On ne tient plus compte, ici, des éventuels nœuds source s et puits p ajoutés au modèle du *Projected 2 Flows Problem* pour sa résolution, sauf la valeur de x_{ps} qui représentera ici le nombre total de tournées qui seront construites. Il est possible de modifier le nombre de tournées dans un deuxième temps, mais

pour la résolution du *Strong Lift*, nous utilisons cette variable comme une donnée du problème. Plusieurs informations sont alors nécessaires pour appliquer la méthode que nous proposons :

- le graphe support de x qui peut être simplifié en agrégeant des arcs qui se suivent dont le degré entrant et sortant des nœuds intermédiaires sont tous égaux à 1 (autrement dit, les véhicules n'ont pas le choix de l'arc suivant).
- les véhicules doivent respecter certaines conditions avant de pouvoir emprunter un arc ou une agrégation d'arcs (notamment concernant le nombre de traversées total sur cet arc dans la solution projeté).

Définition 2. *Une Tournée Partielle est une suite de nœuds liés par des arcs dans le graphe support de x dont les nœuds intermédiaires (ni le premier, ni le dernier) sont de degrés entrant et sortant égaux à 1.*

Une tournée partielle peut donc contenir un seul arc et respecter la définition peu importe les degrés des 2 nœuds de cet arc.

Ainsi, les informations nécessaires sont :

- Une liste de tournées partielles appelée TP.
- Pour chaque tournée partielle :
  - La quantité de toutes les marchandises nécessaires à l'entrée de la tournée partielle.
  - Le nombre de passages de véhicules sur cette tournée partielle.
  - Une valeur ou un ensemble de valeurs permettant de comparer et de trier les tournées partielles

Calcul des tournées partielles

Les tournées partielles peuvent être calculées par l'Algorithme 15.4 grâce à la liste d'entier $nbChoix$ indexé sur les nœuds $u \in N$ du graphe G où $nbChoix[u]$ représente le nombre de trajets différents possible depuis le nœud u, c'est-à-dire :

$$nbChoix[u] = \sum_{v,(u,v)\in E} \begin{cases} 1 & si\ x_{uv} > 0 \\ 0 & sinon \end{cases}$$

Algorithme 15.4 : Calcul des tournées partielles

Entrées : Les variables x_{uv} ainsi que la liste d'entiers $nbChoix$.
TP est une liste de tournées partielles initialement vide.
Début
  Pour tout les variables x_{uv} **Faire**
    Chercher dans TP la première tournée partielle qui :
      finit par u si $nbChoix[u] = 1$ (1)
      ou commence par v si $nbChoix[v] = 1$ (2)
    Si une tournée partielle est trouvée **Alors**
      Insérer v à la fin si (1) ou u au début si (2) de la tournée partielle trouvée
    Sinon
      Créer une nouvelle tournée partielle dans TP et y insérer u puis v
    Fin Si
  Fin Pour
  Retourner TP
Fin

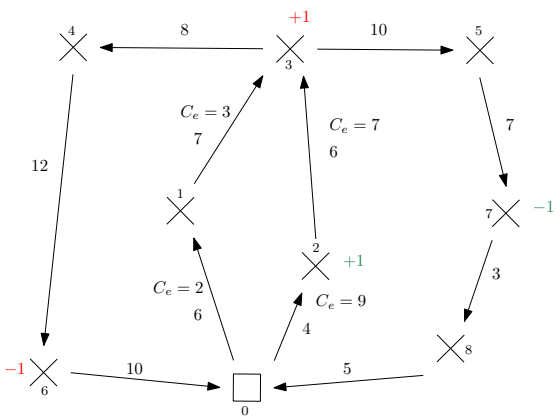

FIGURE 15.16 – Rappel du graphe d'exemple étudié

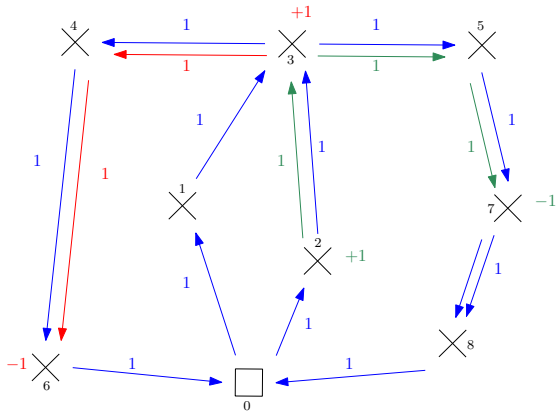

FIGURE 15.17 – Rappel de la solution du P2FP sur l'instance Figure 15.16

Dans notre exemple présenté dans les figures 14.1 et 14.2 (et rappelé dans les figures 15.16 et 15.17), cela se traduit par la création des tournées partielles suivantes (avec $nbChoix = [2, 1, 1, 2, 1, 1, 1, 1, 1]$) :

- 0-1-3
- 0-2-3
- 3-4-6-0
- 3-5-7-8-0

Conditions d'entrées des tournées partielles

La quantité de chaque marchandise nécessaire pour pouvoir emprunter chaque tournée partielle est exactement égale à Y_{uv}^k où k est la marchandise considérée et uv est le premier arc de la tournée partielle :

- $Y_{01}^0 = 0$ et $Y_{01}^1 = 0$
- $Y_{02}^0 = 0$ et $Y_{02}^1 = 0$
- $Y_{34}^0 = 1$ et $Y_{34}^1 = 0$
- $Y_{35}^0 = 0$ et $Y_{35}^1 = 1$

Le nombre de passages de véhicules sur chaque tournée partielle est exactement égale à x_{uv} où uv est n'importe quel arc de la tournée partielle :

- $x_{01} = 1$
- $x_{02} = 1$
- $x_{34} = 1$
- $x_{35} = 1$

Section 16

Expériences numériques

But : Nous réalisons des expériences numériques dans le but d'étudier le comportement des algorithmes de l'instance temporelle à sa solution. Cela comprend quatre expériences : La résolution directe par le graphe dépendant du temps, la projection puis la résolution du *Lift* de manière exacte, la projection puis l'heuristique d'insertion et enfin, la projection puis l'heuristique de simulation. Les quatre méthodes seront comparées et une discussion sera apportée.

Les résultats des méthodes dans le cas de marchandises distinctes sont très similaires au cas de marchandises indistinctes. Nous nous concentrerons sur cas de marchandises distinctes et en présentant des cas particuliers, notamment la décomposition du flot de marchandises dans le modèle projeté.

Contexte technique : Les algorithmes ont été implémentés en C++17 sur un processeur Intel i5-9500 à 4.1 GHz. Les temps CPU sont en secondes.

id	$\|N\|$	$\|E\|$	T_{max}	K
1	5	16	50	2
2	5	16	50	2
3	5	14	50	2
4	5	14	50	2
5	10	42	100	5
6	10	42	100	5
7	10	44	100	5
8	10	42	100	5
9	15	66	150	7
10	15	70	150	7
11	15	68	150	7
12	15	72	150	7

id	$\|N\|$	$\|E\|$	T_{max}	K
13	20	98	200	10
14	20	100	200	10
15	20	100	200	10
16	20	102	200	10
17	25	130	250	12
18	25	122	250	12
19	25	128	250	12
20	25	120	250	12
21	30	160	300	15
22	30	156	300	15

Table 16.1 – Identifiants et tailles des instances

Instances : Nous avons généré des réseaux (N, E) sous forme de graphe planaire aléatoire créé en générant des points dans un espace à 2 dimensions borné de taille $|N|$ par $|N|$. Les arcs sont ensuite générés en utilisant une triangulation de Delaunay sur l'ensemble de ces points 2D. Les valeurs de longueur $L_e, e = (u, v) \in E$ sont calculées en prenant la norme euclidienne entre u et v. Les valeurs de coût des arcs $C_e, e = (u, v) \in E$ sont calculées en prenant la norme 1 entre u et v. Les paramètres de la fonction objective α_1, α_2 et α_3 sont fixés à 0.5, 0.5 et 1. La Table 16.1 présente un groupe de 15 instances avec leurs caractéristiques principales : leur nombre $|N|$ de nuds, leur nombre $|E|$ d'arcs, l'horizon de temps T_{max}, le nombre de marchandises K.

16.1 Résultat lié à la résolution du *Preemptive Relocation Problem*

Nous réalisons les quatre expériences pour résoudre les instances et obtenir une solution temporelle :

- La résolution directe par le *Time-Expanded 2 Flows Problem* dans la Table 16.2 : le temps de calcul *CPU* (en secondes) bornée à 3600 secondes, la valeur de la meilleure solution entière trouvée *Sol* et la borne inférieure *LB* obtenue par CPLEX lors de la résolution directe du TE2FP.
- La projection puis la résolution du *Weak Lift* de manière exacte dans la Table 16.3 : le temps de calcul *CPU* (en secondes) bornée à 3600 secondes, la valeur de la meilleure solution entière trouvée *Sol*, la borne inférieure *LB* obtenue par CPLEX.
- La projection puis l'heuristique d'insertion dans la Table 16.4 : le temps de calcul *CPU* (en secondes) bornée à 3600 secondes, la valeur de la solution *Sol*.
- La projection puis l'heuristique de simulation dans la Table 16.5 : le temps de calcul *CPU* (en secondes) bornée à 3600 secondes, le pourcentage de marchandises non acheminées *%left* ainsi la valeur de la solution *Sol* si toutes les marchandises sont acheminées et une estimation sinon. Dans le dernier cas, lorsque certaines marchandises n'ont pas été acheminées, la valeur de la fonction objectif est précédée par ~ est est calculé en proportion de la quantité de marchandises restantes : $\sim V = \frac{V}{1-\%left}$

id	CPU	Sol	LB
1	0.03	33	33
2	4.86	74.5	74.5
3	3600	73	71.34
4	3600	67.5	63.5
5	3600	244	230.5
6	3600	328.5	311.81
7	3600	243.5	233
8	3600	307.5	281.1
9	3600	744.01	697.14
10	3600	904.02	845.22
11	3600	593.01	551.9
12	3600	1042.02	985.28

id	CPU	Sol	LB
13	3600	913.5	855.59
14	3600	1134.5	1090.5
15	3600	1128.02	1090.5
16	3600	737.01	699.5
17	3600	1130.5	1005.33
18	3600	.	.
19	3600	.	1555.24
20	3600	.	1451.61
21	3600	.	.
22	3600	.	.

TABLE 16.2 – Résultats de la première expérience (résolution directe)

id	CPU	Sol	LB
1	0.011	33	33
2	0.002	75.01	75.01
3	0.002	73.04	73.04
4	0.002	67.54	67.54
5	0.011	253.07	253.05
6	0.011	338.18	337
7	0.021	245.09	245.07
8	0.021	308.62	308.59
9	0.07	758.15	753.5
10	0.1	916.78	904.5
11	0.081	597.22	597.16
12	0.13	1046.29	1032

id	CPU	Sol	LB
13	0.191	918.2	916.5
14	0.061	1135.3	1135
15	0.051	1135.3	1135
16	0.081	737	737
17	1.79	1159.8	1141.5
18	0.26	1866.5	1835
19	0.3	1645.5	1629
20	0.531	1617.5	1617
21	3563.671	.	2299
22	2.5	2335.5	2325

TABLE 16.3 – Résultats de la deuxième expérience (projection puis résolution exacte par MILP)

id	CPU	Sol
1	0.019	33
2	0.011	75.01
3	0.011	73.04
4	0.01	67.54
5	0.021	254.05
6	0.022	338.5
7	0.031	245.07
8	0.033	313.59
9	0.074	758.5
10	0.094	926.5
11	0.092	597.26
12	0.132	1052

id	CPU	Sol
13	0.206	926.5
14	0.078	1136
15	0.068	1135.5
16	0.093	737
17	1.8	1161.5
18	0.27	1866.5
19	0.31	1649
20	0.55	1618
21	3564.3	2299
22	3.05	2345

TABLE 16.4 – Résultats de la troisième expérience (résolution par l'heuristique d'insertion)

id	CPU	Sol	%left
1	0.011	33	0
2	0.002	75	0
3	0.002	73	0
4	0.002	67.5	0
5	0.011	255	0
6	0.011	338.5	0
7	0.021	245	0
8	0.021	317.5	0
9	0.061	758.5	0
10	0.081	~1057.3	25
11	0.081	598	0
12	0.121	~1075.9	40

id	CPU	Sol	%left
13	0.191	928.5	0
14	0.061	1139	0
15	0.051	1139	0
16	0.081	737	0
17	1.781	1163.5	0
18	0.251	~1952	50
19	0.291	~1970.9	94
20	0.531	1618	0
21	3600	.	100
22	2.501	~2192.7	55

TABLE 16.5 – Résultats de la quatrième expérience (résolution par l'heuristique de simulation)

Commentaires :
La première information visible est que la résolution par MILP, que cela soit l'expérience 1 ou 2, est beaucoup trop lourde. En particulier pour la résolution du *Time-Expanded 2 Flows Problem* qui bloque dès les instances de 5 nœuds. La résolution du *Weak Lift* exact est un petit peu plus efficace, mais bloque à partir des instances de 10 nœuds. Néanmoins, les différentes résolutions du *Weak Lift* (expériences 2, 3 et 4) ne trouve pas toujours la solution optimale du PRP (voir instances 5 et 9). Les détails seront précisés dans d'autres expériences, mais il est important de noter que la résolution du *Projected 2 Flows Problem* a pris 3600 secondes pour l'instance 16. De ce fait, toutes les expériences se trouvent avec des temps de calculs importants. Dernière information, la méthode de simulation est la plus rapide, mais bloque sur certaines instances. Aucune caractéristique évidente ne peut expliquer pourquoi une instance bloque la simulation et pas une autre, bien que ce phénomène semble s'accentuer avec la taille des instances.

Maintenant, la question est de savoir à quoi est dû le décalage de valeur objectif lorsqu'il y en a un. (lors du *Projected 2 Flows Problem* ou des méthodes de reconstruction).

16.2 Résultats liés aux différents algorithmes isolés les uns des autres

Pour bien comprendre les résultats précédents, il est nécessaire de bien distinguer l'implication des différents algorithmes. Nous présentons alors les résultats de la résolution directe du TE2FP, de la résolution

du P2FP, de la résolution exacte du *Weak Lift*, de l'heuristique d'insertion à partir de la solution projetée et, enfin, de l'heuristique de simulation à partir de la solution projetée. Pour chacun des cinq algorithmes, nous calculons :

- dans la Table 16.6 : un rappel des résultats de la résolution direct du *Time-Expanded 2 Flows Problem* (voir table 16.2).
- dans la Table 16.7 : le temps de calcul *CPU* (en secondes) bornée à 3600 secondes, la valeur de la meilleure solution entière trouvée *Sol*, la borne inférieure *LB* obtenue par CPLEX et le nombre de coupes de sous-tours étendus $\underset{enst}{Cut}$, le nombre initial de coupes de décomposition des marchandises en chemins *réalisables* avant le début de la résolution $\underset{Dec}{Init}$ et le nombre de coupes de décomposition ajouté pendant la résolution $\underset{Dec}{Cut}$.
- dans la Table 16.8 : le temps de calcul *CPU* (en secondes) bornée à 3600 secondes, la valeur de la meilleure solution entière trouvée *Sol* et la borne inférieure *LB* obtenue par CPLEX lors de la résolution MILP du *Weak Lift*.
- dans la Table 16.9 : le temps de calcul *CPU* (en secondes), la valeur de la solution entière trouvée *Sol* et le pourcentage de marchandises non acheminées $\%left$ en cas de blocage lors de la résolution du *Weak Lift* par l'heuristique d'insertion.
- dans la Table 16.10 : le temps de calcul *CPU* (en secondes) et la valeur de la solution entière trouvée *Sol* lors de la résolution du *Weak Lift* par l'heuristique de simulation.

id	CPU	Sol	LB
1	0.03	33	33
2	4.86	74.5	74.5
3	3585.78	73	71.34
4	3591.98	67.5	63.5
5	3597.08	244	230.5
6	3597.42	328.5	311.81
7	3595.05	243.5	233
8	3598.56	307.5	281.1
9	3593.57	744.01	697.14
10	3598.91	904.02	845.22
11	3598.58	593.01	551.9
12	3598.62	1042.02	985.28

id	CPU	Sol	LB
13	3580.97	913.5	855.59
14	3598.37	1134.5	1090.5
15	3599.02	1128.02	1090.5
16	3598.44	737.01	699.5
17	3597.42	1130.5	1005.33
18	3595.33	.	.
19	3597.5	.	1555.24
20	3590.5	.	1451.61
21	3596.89	.	.
22	3598.29	.	.

TABLE 16.6 – Rappel des résultats de la résolution directe du *Time-Expended 2 Flows Problem*

id	CPU	Sol	*LB*	$\underset{enst}{Cut}$	$\underset{Dec}{Init}$	$\underset{Dec}{Cut}$
1	0.01	33	33	0	4	0
2	0.001	74	74	0	4	0
3	0.001	73	73	0	4	0
4	0.001	67.5	67.5	1	6	0
5	0.01	241	241	1	16	1
6	0.01	328	327.98	1	36	0
7	0.02	240	240	2	20	0
8	0.02	307.5	307.5	0	49	1
9	0.06	738	738	4	30	0
10	0.08	886.5	886.5	2	56	0
11	0.08	591	591	1	70	1
12	0.12	1041	1040.9	1	48	0

id	CPU	Sol	*LB*	$\underset{enst}{Cut}$	$\underset{Dec}{Init}$	$\underset{Dec}{Cut}$
13	0.19	903.5	903.5	2	72	0
14	0.06	1129	1129	0	90	0
15	0.05	1119	1119	0	90	0
16	0.08	737	737	0	90	1
17	1.78	1123.5	1123.4	6	100	2
18	0.25	1855.5	1855.3	1	156	0
19	0.29	1635	1634.9	3	154	0
20	0.5	1617	1616.9	1	99	1
21	3563.67	2313	2307.5	41	168	50
22	2.5	2314	2313.9	2	168	2

TABLE 16.7 – Résultats de la résolution du *Projected 2 Flows Problem*

id	CPU	Sol	LB
1	0.001	33	33
2	0.001	75.01	75.01
3	0.001	73.04	73.04
4	0.001	67.54	67.54
5	0.001	253.07	253.05
6	0.001	338.18	337
7	0.001	245.09	245.07
8	0.001	308.62	308.59
9	0.01	758.15	753.5
10	0.02	916.78	904.5
11	0.001	597.22	597.16
12	0.01	1046.29	1032

id	CPU	Sol	LB
13	0.001	918.24	916.5
14	0.001	1135.33	1135
15	0.001	1135.33	1135
16	0.001	737	737
17	0.01	1159.8	1141.5
18	0.01	1866.5	1835
19	0.01	1645.5	1629
20	0.001	1617.5	1617
21	0.001	0	2299
22	0.01	2335.5	2325

TABLE 16.8 – Résultats de la résolution MILP du *Weak Lift* à partir de la solution projetée

id	CPU	Sol
1	0.009	33
2	0.01	75.01
3	0.01	73.04
4	0.009	67.54
5	0.011	254.05
6	0.012	338.5
7	0.011	245.07
8	0.013	313.59
9	0.014	758.5
10	0.014	926.5
11	0.012	597.26
12	0.012	1052

id	CPU	Sol
13	0.016	926.5
14	0.018	1136
15	0.018	1135.5
16	0.013	737
17	0.02	1161.5
18	0.019	1866.5
19	0.018	1649
20	0.02	1618
21	0.667	2299
22	0.55	2345

TABLE 16.9 – Résultats de l'heuristique d'insertion à partir de la solution projetée

id	CPU	Sol	%left
1	0.001	33	0
2	0.001	75	0
3	0.001	73	0
4	0.001	67.5	0
5	0.001	255	0
6	0.001	338.5	0
7	0.001	245	0
8	0.001	317.5	0
9	0.001	758.5	0
10	0.001	~1057.3	25
11	0.001	598	0
12	0.001	~1075.9	40

id	CPU	Sol	%left
13	0.001	928.5	0
14	0.001	1139	0
15	0.001	1139	0
16	0.001	737	0
17	0.001	1163.5	0
18	0.001	~1952	50
19	0.001	~1970.9	94
20	0.001	1618	0
21	0.001	0	100
22	0.001	~2192.7	55

TABLE 16.10 – Résultats de l'heuristique de simulation à partir de la solution projetée

Commentaires :
Comme précisé plus tôt, l'instance 16 n'est pas résolu exactement par le P2FP en 3600 secondes. Dans cette instance, le dépôt est sur un bord du graphe et le nombre de coupes nécessaires est grand, réduisant le temps passé à l'exploration de l'arbre du *Branch&Bound*. Pour finir, le temps de calcul des méthodes de résolutions par insertion et par simulation sont clairement visibles ici, prouvant bien que les deux méthodes sont très efficaces bien que la simulation reste bloquée, faute d'avoir ajouté les trajets nécessaires. Pour rappel, cette dernière méthode part de l'idée que le flot de véhicules projeté n'est pas très loin des trajets dans la solution du PRP et ajoute des routes en suivant certaines règles. Il est certainement possible de trouver de meilleurs résultats en utilisant d'autres règles.

Section 17

Conclusion

Dans ce chapitre, nous avons étudié la possibilité de résoudre un problème intrinsèquement dépendant du temps et passant par une version projetée. Nous nous sommes concentrés sur un problème de relocation de marchandises avec préemption appelé *PRP*. Ce type de problème nécessite une synchronisation des véhicules de transports pour pouvoir se transférer des marchandises.

La résolution directe de ce problème par un modèle linéaire sur un graphe *Time-Expanded* est très vite limitée et ne permet pas une résolution efficace. Mais la résolution du problème projeté sur le graphe initial est très rapide et permet d'obtenir beaucoup d'informations sur une solution proche de la solution optimale du *PRP*. En particulier si nous considérons le flot de marchandises comme étant la projection fidèle du même flot dans la solution du *PRP*. Nous avons appelé ce problème le *Weak Lift* et nous avons proposé trois méthodes pour reconstruire les tournées de véhicules permettant d'acheminer ces marchandises. La première est un modèle linéaire sur un graphe approprié permettant d'obtenir la solution optimale du *Weak Lift*. La deuxième est une méthode d'insertion dans le même graphe que le modèle linéaire précédemment utilisé. Et la dernière est une méthode de simulation sur le graphe initial.

Les trois méthodes ont des points faibles, mais donnent toutes des résultats identiques ou proches. Les deux points faibles les plus importants sont : la résolution exacte du *Weak Lift* est rapidement mise en défaut par la taille des instances et la résolution par simulation ne permet pas toujours d'acheminer toutes les marchandises. Seule la méthode d'insertion possède les avantages des deux autres méthodes, mais sans garanties d'optimalités.

Par la suite, il peut être intéressant d'appliquer cette idée sur un autre problème pour vérifier ses performances sur différentes classes de problèmes dépendant du temps.

Chapitre III

Le Problème de dimensionnement et planification de deux lignes de production avec transferts synchronisés

Enfin, nous traitons un problème lié à la production par un sous-traitant de ressources qu'un producteur final va utiliser pour réaliser une séquence de travaux dans un horizon temporel. Tous deux sont dotés de capacités de stockage limitées, ce qui les oblige à synchroniser leurs processus respectifs de production et de consommation. Le problème de production de ressources qui en résulte apparaît, du point de vue du producteur de ressources, comme un problème de *Lot Sizing* à plusieurs étapes, où chaque étape doit être délimitée par des transferts qui deviennent le cœur de la décision. Un tel problème peut généralement se poser dans des contextes dans lesquels la ressource est une sorte d'énergie renouvelable (hydrogène, photovoltaïque, ...) et les travaux nécessitent que cette énergie soit stockée dans des réservoirs ou des batteries. Nous allons nous concentrer sur le problème de synchronisation entre ces deux types d'acteurs. Dans ce chapitre, nous décrivons d'abord le problème que nous nommons SLSS (*Synchronized Lot Sizing/Scheduling*) de manière précise et définissons un modèle MILP, amélioré avec des coupes valides supplémentaires, et résolvons les variables de décision $\{0, 1\}$ tandis que les autres variables sont gérées par la génération de coupes de Benders. Enfin, nous montrons que notre problème SLSS peut être vu comme la recherche d'un chemin dans un espace de transfert partiellement ordonné spécifique et le traite en s'appuyant sur une adaptation de l'algorithme A* pour la recherche dans de très grands espaces d'états.

Table des matières

Liste des notations

Eater :

M	Le nombre de jobs à réaliser
$j = 0, \ldots, M-1$	Les indices des jobs
t_j	La durée du job j
r_j	Le coût en ressources du job j
C^F	La capacité de stockage de Eater
C_0^F	La quantité de ressources initialement dans le stock de Eater

Feeder :

N	Le nombre de périodes de production
$i = 0, \ldots, N-1$	Les indices des périodes
p	La durée des périodes (les périodes sont identiques en durée)
R_i	La quantité de ressource produite en période i
$Cout_i^{Prod}$	Le coût de production en période i
$Cout_i^{Act}$	Le coût d'activation de la production en période i
α	Pondération du coût de production dans la fonction objectif
C^E	La capacité de stockage de Feeder
C_0^E	La quantité de ressources initialement dans le stock de Feeder
z	Le vecteur de variables des périodes activées
y	Le vecteur de variables des activations de la production

Temporalité et Transferts :

T	La date de fin du dernier job
β	Pondération de la variable T dans la fonction objectif
x	Le vecteur de variables des transferts exécutés
δ	Le vecteur de variables des transferts exécutés vu par Eater
γ	Le vecteur de variables des transferts exécutés vu par Feedere
m	Le vecteur de variables des quantités transportées
C^{Trans}	La quantité maximale transportable
d_j	Le temps nécessaire à la préparation d'un transfert après le job j
ϵ_j	La quantité supplémentaire de ressources nécessaires à un transfert après le job j
$\mu(j_1, j_2)$	La quantité de ressources consommées entre deux transferts en j_1 et j_2
$\Delta(j_1, j_2)$	Le temps minimum entre deux transferts en j_1 et j_2
Λ	L'ensemble des transferts sous forme de couples (i, j)
$\tau_m(j)$	L'indice de la première période possible pour faire un transfert au job $j-1$
$\tau_M(j)$	L'indice de la dernière période possible pour faire un transfert au job j
$\Gamma(j)$	L'ensemble des jobs réalisables avec un entrepôt plein à la fin du job j

Section 18

Introduction

Le *Lot Sizing* se pose lorsqu'il faut planifier la production en profitant des effets d'échelle, qui correspondent généralement à des coûts marginaux décroissants et à des coûts d'activation élevés (voir [Clark et al., 2011, Drexl and Kimms, 1997]). Le temps est généralement structuré en périodes, et une partie d'une stratégie *Lot Sizing* consiste à concentrer la production sur des périodes bien choisies afin de minimiser les coûts induits par l'activation de la production. Dans des contextes standard, la production doit être livrée aux clients à la fin de certaines périodes fixes, et entre-temps doit être stockée de telle sorte que l'on doit tenir compte à la fois des contraintes de capacité de stockage et des coûts de stockage. Les problèmes qui en résultent peuvent se présenter comme des problèmes de sac à dos impliquant une dimension temporelle (voir [Kellerer et al., 2004]). Ils sont souvent traités par des algorithmes pseudo-polynomiaux comme la programmation dynamique, ce qui donne, dans de nombreux cas, des FPTAS (*Fully Polynomial Time Approximation Scheme*) [Woeginger, 2000].

Dans le problème du *Multi-dimensional Lot Sizing* (voir [Song et al., 2008]) nous avons affaire à une production hétérogène et à des contraintes de capacité liées à différentes caractéristiques (poids, volume, ...). Le *Multi-level Lot Sizing* (voir [Colson et al., 2005, Goisque and Rapine, 2017, Hung and Chien, 2000]) signifie plusieurs acteurs agissant sur un même système (par exemple un producteur principal et plusieurs sous-traitants). Ces acteurs peuvent être partiellement indépendants les uns des autres, ce qui pose le problème de la collaboration qui peut être gérée en s'appuyant sur la Théorie des Jeux Coopératifs (voir [Thomson and Moulin, 1990]) ou sur des reformulations multiobjectifs. Une méthode de résolution souvent appliquée exécute plusieurs itérations des schémas de résolution de ces problèmes permettant de tenir compte des changements contextuels, comme dans le *Lot Sizing* en plusieurs étapes (voir [Kaminsky and Simchi-Levi, 2003]). La notion de collaboration implique, non seulement que les conséquences sont distribuées, mais également que les décisions sont prises individuellement en collaboration avec les autres acteurs. À l'inverse, les processus centralisés n'ont qu'un seul décisionnaire et distribuent les décisions. Cela permet la recherche de stratégies visant l'amélioration de l'objectif global sans se préoccuper des conséquences individuelles. Dans tous les cas, les problèmes résultants sont NP-difficiles.

Or, il peut arriver qu'un tel processus de production s'inscrive dans le cadre d'une interaction entre le producteur et un ou plusieurs consommateur(s), tous deux pourvus d'un agenda propre et tenus de se synchroniser du fait de contraintes de capacité. Ce sera le cas si le producteur est un sous-traitant qui fournit au(x) consommateur(s) des ressources que ces consommateurs vont utiliser pour réaliser leurs propres tâches de service ou de production. Considérons par exemple (voir Figure 18.1) un producteur local d'énergie $Feeder$ qui peut produire de l'énergie tout au long de N périodes $i = 0, \dots, N-1$ de durée identique mais un montant R_i et un coût C_i différent. Ce producteur d'énergie interagit avec un fournisseur de services $Eater$ qui a besoin de cette énergie pour effectuer des travaux $j = 0, \dots, M-1$, chaque travail nécessitant des unités de ressources et des unités de temps. $Feeder$ doit assurer une production suffisante, donc il doit décider d'un vecteur de production $z = (z_i \in \{0,1\}, i = 0, \dots, N-1)$, solution du problème $Feeder\_Lot\_Sizing$:

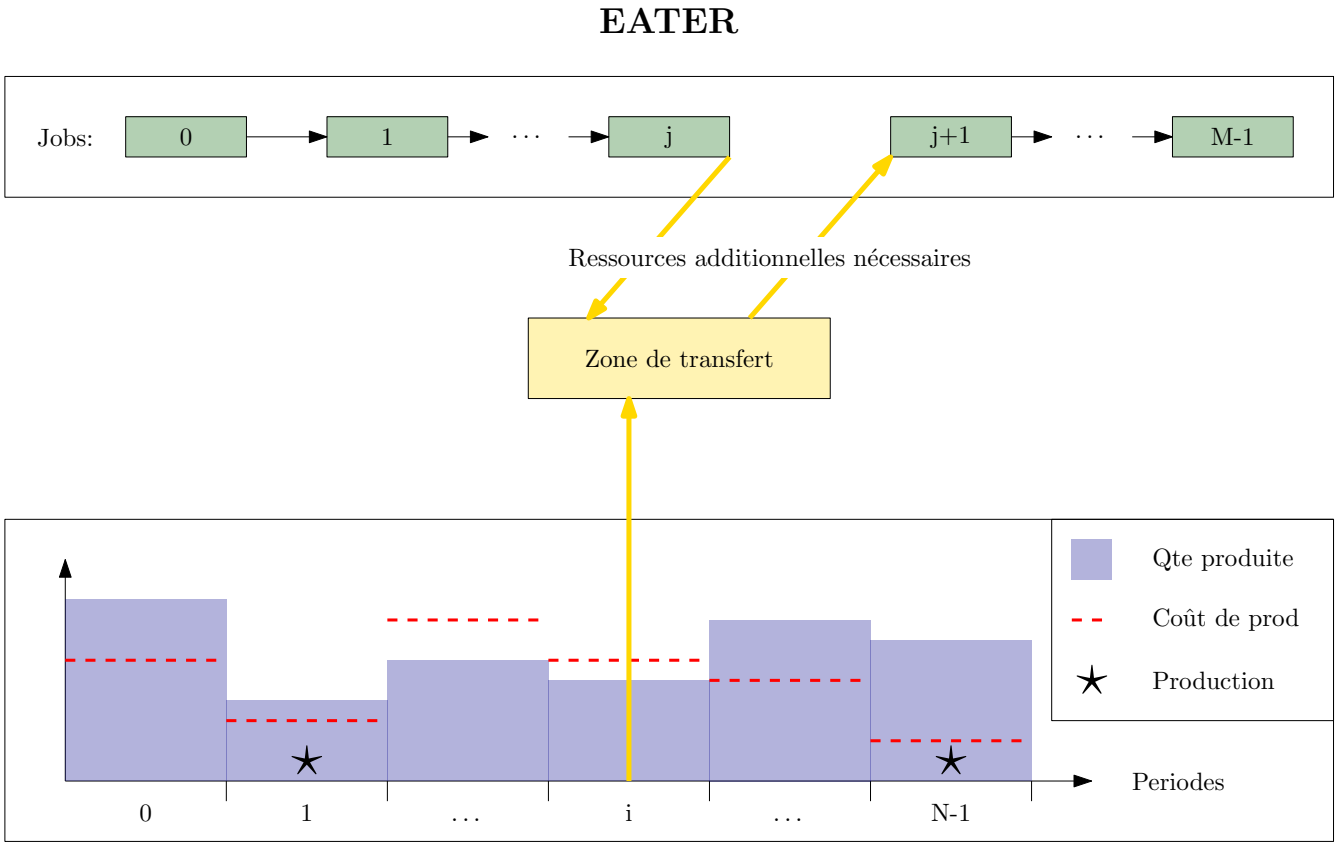

FIGURE 18.1 – processus d'interaction *Feeder*/*Eater*

$$\text{Minimiser } \sum_i C_i.z_i$$
$$\text{tel que } \sum_i R_i.z_i \geq Total\_Cons$$

où $Total\_Cons$ est la quantité totale d'énergie demandée par *Eater*.

D'un autre côté, le problème de *Eater* concerne la façon dont il planifie les tâches $j = 0, \ldots, M-1$, tout en optimisant son propre critère de performance (par exemple le makespan) et en respectant les contraintes temporelles et les contraintes de ressources : chaque fois que *Eater* commence à effectuer un travail j, il doit charger (dans un réservoir ou dans une batterie) le travail avec l'énergie nécessaire.

Cependant, pour mettre en œuvre ce processus de production/consommation, *Feeder* et *Eater* doivent s'accorder sur les périodes i, et les dates t auxquelles *Feeder* transfère une quantité d'énergie L à *Eater*. Un tel transfert $\omega = (i, t, m)$ peut nécessiter un coût, une durée et une quantité d'énergie supplémentaires de la part des deux joueurs et peut aussi dans de nombreux cas interdire la production d'énergie ou le traitement des tâches. Nous obtenons alors une description complète de notre processus si nous connaissons, outre le vecteur de production *Feeder* z et le calendrier de décision *Eater*, la collection Ω de tous les transferts d'énergie $\omega = (i, t, m)$ qui font se rencontrer les deux acteurs.

Or, il arrive aussi généralement que, dans un tel contexte, *Feeder* et *Eater* stockent l'énergie disponible à l'intérieur d'installations de stockage aux capacités limitées, respectivement CF et CE. Il s'ensuit que notre modèle *Feeder_Lot_Sizing* se décompose en une séquence d'étapes *LotSizing*, délimitée par les périodes i telles que les transferts ont lieu. Soit dit en passant, le problème de planification de *Eater* se divise également en étapes de planification, délimitées par les dates t auxquelles les transferts ont lieu. Ainsi, la synchronisation des processus *Feeder* et *Eater* associés devient notre problème clé.

Le problème décrit ci-dessus est une sorte de problème complexe *Lot Sizing and Job Scheduling* à deux (multi)niveaux avec des étapes flexibles et des contraintes de synchronisation. Elle se pose dans des contextes liés non seulement à la gestion de l'énergie (voir [Biel and Glock, 2016, Irani and Pruhs, 2005, Moon and Park, 2014, Pechmann and Schöler, 2011]), mais aussi à la coopération en temps réel entre capteurs et robots, véhicules électriques et installations de recharge (voir [Erdelic et al., 2019, Macrina et al., 2020]) ou entre *supply chain management* et *asset management* (voir [Adulyasak et al., 2015, Labbé et al., 2020, Sargut and Romeijn, 2007, Taillandier et al., 2017]). Les niveaux connexes n'identifient pas de manière naturelle un leader et un suiveur, et posent ainsi la question *centralisé versus collaboratif.*

Mais il s'agit clairement d'un problème complexe, et nous allons donc définir de manière formelle un problème que nous nommons **SLSS** : *Synchronized Lot Sizing/Scheduling*, que nous pouvons considérer comme un problème central pour la gestion synchronisée de processus multiacteurs tels que décrit précédemment. Bien que la question *centralisé versus collaboratif* soit cruciale en pratique, nous nous en tiendrons ici au paradigme centralisé et nous nous intéresserons aux caractéristiques structurelles et algorithmiques de notre modèle. Cette analyse structurelle fera apparaître le rôle central joué par les *transferts* à l'intérieur de notre problème. Nous montrerons qu'ils doivent induire des chaînes à l'intérieur d'un ensemble partiellement ordonné Γ et induire des contraintes $No_Antichain$ à l'intérieur de nos formulations de programmation linéaire en nombres entiers (ILP). Nous étudierons ensuite la façon dont la décomposition de Benders (voir [Colson et al., 2005, Kleinert et al., 2021]) peut être appliquée pour éliminer les variables non combinatoires à l'intérieur de ces formulations ILP. Enfin, nous montrerons qu'il est possible de résoudre notre problème **SLSS** en appliquant un algorithme de recherche de chemin à l'ensemble partiellement ordonné Γ de toutes les transactions de transfert possibles.

La partie est donc organisée comme suit. Nous fournissons d'abord une description formelle du problème **SLSS** de base, et discutons brièvement de quelques variantes. Dans la section suivante, nous proposons une première formulation de programmation linéaire mixte en nombres entiers (MILP) orientée *Branch&Cut* de **SLSS** qui induit la séparation de contraintes $No_Antichain$. Nous décrivons ensuite la façon dont nous pouvons améliorer cette formulation avec des contraintes valides supplémentaires, limitées aux variables combinatoires et gérées par l'application de la décomposition de Benders. La section suivante est consacrée à notre contribution principale, à savoir un algorithme de recherche de chemin qui nous permet d'affirmer que **SLSS** est pseudo-polynomial. Nous terminons par une brève discussion sur la question de la collaboration.

Section 19

Problème détaillé

19.1 Présentation du problème

Le problème que nous avons étudié est théorique, mais est une généralisation d'un problème de changement de batterie pour un véhicule électrique. Le véhicule a des tâches à réaliser et est alimenté avec une batterie pendant qu'une deuxième est rechargée par un producteur d'électricité verte (énergie renouvelable : photovoltaïque, éolien, ...). L'échange d'une batterie vide pour une batterie pleine permet un très court temps de recharge du côté véhicule et permet ainsi de pouvoir reprendre les tâches à réaliser sans attendre.

Du côté du véhicule, nous supposons que l'ordre des tâches est fixé et doivent être réalisé dans un horizon de temps. Cet ordre peut faire partie, dans un premier temps, d'un problème d'affectation ou de VRP pour coordonner une flotte, puis notre problème est étudié pour permettre la réalisabilité de l'ensemble des tâches sous contrainte énergétiques.

Du côté du producteur d'électricité, la quantité produite par plage horaire est hors de contrôle du producteur (par la météo par exemple) et le prix de vente (qui influence directement le coût relatif de production) est fixé par les tendances du marché et n'est donc pas modifiable non plus par le producteur. Le seul choix qu'il possède est l'activation ou non de la production qui implique l'ajout d'un coût d'activation économique, en particulier pour l'éolien. Aussi, pour ne pas détériorer les batteries à cause de recharges trop profondes, nous imposons qu'il ne soit pas possible de produire plus que la capacité totale des batteries à disposition.

Au moment du transfert de la batterie vide du véhicule contre une batterie chargée par producteur électrique, aucun des deux acteurs ne peut effectuer de tâches. En effet, le véhicule doit rejoindre le point de recharge et la batterie chargée est mise à disposition et ne peut plus être rechargé pendant ce temps. Bien sûr, un tel transfert nécessite également l'utilisation d'électricité pour le déplacement du véhicule vers le lieu d'échange. Il y a donc un léger coût supplémentaire du côté du véhicule.

L'objectif du problème est alors d'assurer la réalisation de toutes les tâches du véhicule dans l'horizon de temps. Il est donc nécessaire de produire suffisamment d'électricité et de garder le véhicule chargé durant cet horizon de temps. Nous imposons par ailleurs que le niveau des batteries en fin d'horizon de temps chez les deux acteurs soient revenus à leur niveau de début d'horizon de temps. Nous supposons finalement qu'il n'y a pas d'incertitudes sur les entrées du problème.

Le problème théorique que nous proposons d'étudier est une généralisation dans la gestion de la batterie des deux acteurs : ce seront des stocks de capacités différentes permettant de stocker un produit intermédiaire. Ce produit intermédiaire peut être transféré en tout ou partie et la quantité que le véhicule n'a pas utilisée peut être gardée et n'est pas rendu au producteur comme dans le cas d'une batterie. Pour retrouver le cas présenté plus haut, les stocks devront être de capacités égales et une contrainte sur la quantité à transférer doit être ajoutée (la quantité à transférer est égale à la différence des deux stocks au moment du transfert).

19.1.1 Problème formalisé

- ***Eater* (véhicule réalisant des tâches) :**
  Jobs j=0,...,M−1, à réaliser séquentiellement : chaque job j requiert $\boldsymbol{t_j}$ unités de temps, et consomme $\boldsymbol{r_j}$ produits intermédiaires (*ressources*). La capacité de stockage du produit intermédiaire de cet acteur est $\boldsymbol{C^F}$. Sa réserve initiale est $\boldsymbol{C_0^F}$. Elle doit être au moins égale C_0^F à la fin de l'horizon de temps.
- ***Feeder* (production électrique) :**
  Il produit au cours de **périodes i=0,...,N−1**, identiques et de durées $\boldsymbol{p}$. À chaque période, il peut produire au plus $\boldsymbol{R_i}$ unités, pour un coût de $\boldsymbol{Cout_i^{Prod}}$. L'activation en période i de la production (passage d'une période oisive à une période active) coûte $\boldsymbol{Cout_i^{Act}}$. La capacité de stockage de ce producteur est $\boldsymbol{C^E}$. Sa réserve initiale est $\boldsymbol{C_0^E}$. Elle doit être au moins égale C_0^E à la fin du processus.
- **Transferts :**
  Un transfert de ressources entre les deux acteurs a lieu au cours d'une période i, entre un job j et son successeur $j+1$. Le producteur intermédiaire ne peut pas produire durant la période i. Le surplus de temps requis pour le *Eater* est $\boldsymbol{d_j+p}$ et le surplus de produit intermédiaire est $\boldsymbol{\epsilon_j}$. Ce surplus de consommation est pris de la réserve du *Eater*.
- **Objectif :**
  Réaliser les jobs au moindre coût et en minimisant la durée : $\alpha.(\sum_i (Cout_i^{Prod}.z_i + Cout_i^{Act}.y_i) + \beta.T$, avec $\boldsymbol{T}$ la date de fin du dernier job $M-1$, $\boldsymbol{z}$ et $\boldsymbol{y}$ désignent respectivement les périodes d'activité et d'activation du producteur *Feeder*, et α et β sont des coefficients de proportionnalités entre le coût économique et le temps total du processus.

Avant de continuer avec un cadre MILP formel de ce problème et son analyse structurelle, nous allons présenter un exemple et discuter brièvement de ce que pourraient être des variantes du cadre ci-dessus.

19.1.2 Un exemple

Prenons $M = 5$, $N = 7$, $p = 5$, $C^E = 6$, $C^F = 10$, $C_0^E = 3$, $C_0^F = 2$, $\alpha = 2$, $\beta = 1$, ainsi que les vecteurs t, d, r, ϵ, R, $Cout^{act}$, $Cout^{prod}$ tels que présentés Table 19.1.

Eater

j	0	1	2	3	4
r	2	4	1	2	3
ϵ	1	1	2	1	2
t	3	4	2	1	7
d	1	2	1	1	2

Feeder

i	0	1	2	3	4	5	6
R	5	2	4	5	4	2	5
***C**^{act}*	4	1	5	2	6	1	8
***C**^{prod}*	5	5	6	5	3	4	6

Transferts

i	1	4
j	0	2
m	7	8

TABLE 19.1 – Une instance du problème et une solution réalisable

Ensuite, nous obtenons une solution réalisable en posant :

- Deux transferts $(1,0,7),(4,2,8)$:
  À la période 1, le *Feeder* transfère 7 unités de ressource au *Eater* à la fin du job 0, et à la période 4, il transfère 8 unités de ressource à la fin du job 2.
- Les périodes actives sont $z = (1,0,1,1,0,1,0)$ et les périodes d'activations sont $y = (1,0,1,0,0,1,0)$.
- Les heures de début des jobs sont $(0,10,14,25,26)$ (temps d'attente respectif à la fin des jobs : $(1,0,3,0,0)$).

Cette solution induit le coût économique suivant $Cout = Cout^{act} + Cout^{prod} = (4+5+2+1)+(5+6+5+4) = 32$ et la date de fin est $T = 33$. Le coût global est donc $\alpha.Cout + \beta.T = 2 \times 32 + 33 = 97$.

19.1.3 Notations additionnelles

Les notations suivantes sont des outils pour simplifier les contraintes présentées par la suite.

Consommation de ressources et de temps entre deux transferts :

Pour tout couple (j_1, j_2) de jobs telles que $0 \leq j_1 < j_2 \leq M-1$, on note $\boldsymbol{\mu(j_1, j_2)}$ la quantité de ressources consommées entre deux transferts entre j_1 et j_1+1, puis entre j_2 et j_2+1. On a donc : $\mu(j_1, j_2) = \epsilon_{j_2} + \sum_{j_1+1 \leq j \leq j_2} r_j$. On étend cette définition à $j_1 = -1$ et $j_2 = M$:

- $\mu(-1, j) = \mu_j^0$ exprime la ressource consommée entre le début du processus et le premier transfert.
- $\mu(j, M) = \mu_j^*$ exprime la ressource consommée entre le dernier transfert et la fin du processus, sans le terme ϵ_j.

De la même façon, on note $\boldsymbol{\Delta(j_1, j_2)}$ le temps nécessaire entre deux transferts entre j_1 et j_1+1, puis entre j_2 et j_2+1. On a donc : $\Delta(j_1, j_2) = d_{j_2} + \sum_{j_1+1 \leq j \leq j_2} t_j$. On étend cette définition à $j_1 = -1$ et $j_2 = M$:

- $\Delta(-1, j) = D_j$ exprime le temps minimal entre le début du processus et le premier transfert.
- $\Delta(j, M) = D_j^*$ exprime le temps requis entre le dernier transfert et la fin du processus, sans le terme d_j.

Ensemble ordonné $(\boldsymbol{\Lambda}, \ll)$:

On considère alors l'ensemble $\boldsymbol{\Lambda}$ des couples (i, j), $0 \leq i \leq N-1$, $0 \leq j \leq M-1$, et on définit sur cet ensemble la relation $\ll$ suivante : $(i_1, j_1) \ll (i_2, j_2)$ si et seulement si $(i_2 - i_1 - 1).p \geq \Delta(j_1, j_2)$. La relation $\ll$ définit une relation d'ordre partiel sur Λ. On complète Λ et $\ll$ de façon naturelle avec les couples $(-1, -1)$ et (N, M), respectivement éléments minimum et maximum dans Λ pour $\ll$. Une $\ll$-*Antichaine* est tout sous-ensemble de Λ formés de couples (i, j) 2 à 2 incomparables au sens de $\ll$.

Cette relation nous permet de vérifier la faisabilité d'un ensemble de transferts, ou plutôt, leur incompatibilité. Par exemple, les couples $(1, 2)$ et $(2, 1)$ forment une $\ll$-Antichaine (et sont donc incompatibles), peu importe l'instance étudiée. En effet, si un transfert a lieu entre la période 1 et le job 2, il n'est pas possible d'effectuer un transfert entre la période 2 et le job 1.

Autres notations :

- L'indice de la première période possible pour faire un transfert entre le job j et le job $j+1$: $\tau_m(j) = \left\lceil \frac{\sum_{k \leq j} t_k + d_j}{p} \right\rceil$
- L'indice de la dernière période possible pour faire un transfert entre le job j et le job $j+1$: $\tau_M(j) = (N-1) - \left\lceil \frac{\Delta(j, M))}{p} \right\rceil$

Ces deux notations τ_m et τ_M vont nous servir pour déterminer les couples (i, j) auxquels un transfert n'est jamais possible. Dans le premier cas, $i < \tau_m(j)$ veut dire que la somme des durées de tous les jobs jusqu'à j, même s'ils sont faits sans pause, est plus grande que la date à la période i (c'est-à-dire $p.i$) et il est donc impossible de synchroniser le job j et la période i. De la même façon, $i > \tau_M(j)$ veut dire que, si un transfert a lieu à la période i et après le job j, il ne restera pas assez de temps pour réaliser tous les jobs restant avant la fin de l'horizon de temps.

- $C^{Trans} = min(C^E, C^F)$

C^{Trans} est la valeur maximale transportable qui ne fait pas déborder le stock de *Eater* s'il est vide.

- L'ensemble des jobs réalisables si un transfert est réalisé juste après le job j : $\forall j \geq 0, \quad \Gamma(j) = \{k > j \mid \mu(j, k) \leq C^{Trans}\}$ et $\Gamma(-1) = \{k \mid \mu(-1, k) \leq C_0^F\}$.

Cet ensemble $\Gamma(j)$ est l'ensemble des jobs réalisables sans interruption en supposant qu'un maximum de produit ait été transféré juste après le job j.

19.2 Discussion sur les variantes du SLSS

Selon le contexte d'application, le modèle ci-dessus peut devoir être adapté :

Modes de production du LotSizing

La fonctionnalité de dimensionnement des lots est contenue dans le coût d'activation. Ce coût d'activation incite le producteur à regrouper ses périodes actives en intervalles (groupes de périodes consécutives). Cela correspond à ce qui se passe avec la production d'hydrogène solaire, lorsque le soleil détermine la production à une période donnée, et lorsque les coûts sont dus à l'électrolyse (coûts variables) tandis que les coûts fixes sont dus à l'intervention humaine. Il aurait pu être possible d'utiliser un mode de dimensionnement des lots plus standard, où le niveau de production à toute période fait partie de la décision, le coût lié à une telle période étant un coût fixe augmenté d'un coût suivant une fonction concave du niveau de production. Aussi, on aurait pu s'appuyer sur un calage des fonctions de coût, impliquant des opérations d'achat et de vente, comme cela peut se produire dans le cas de la gestion de la puissance produite par une plateforme photovoltaïque.

Modes de transfert :

Une opération de transfert peut ne pas être liée à une période exactement. Elle peut nécessiter moins d'une période, ou au contraire plusieurs périodes (par exemple lorsque la ressource est de l'énergie électrique qui doit être chargée dans des batteries). Aussi, le transfert et la production peuvent ne pas être exclusifs, mais impliquer un coût supplémentaire pour le producteur de la ressource qui, en pratique, devra superviser une telle opération de transfert.

Planification de tâches :

Par souci de simplicité, nous avons choisi ici de restreindre la décision de l'ordonnanceur de jobs aux heures de début des jobs. Bien sûr, l'ordonnancement des jobs peut faire partie du problème, qui deviendra un problème à deux niveaux à traiter en étudiant plus spécifiquement l'aspect collaboratif. Dans un tel cas, le planificateur doit faire face à un type de problème de *Bin Packing*. Le problème du SLSS est alors le sous problème d'un tel problème bi-niveaux dans lequel une affectation est réalisée et chaque ensemble de tâches doit être évalué (faisabilité, coûts économiques et date de fin du processus).

19.3 Un programme linéaire en nombre entier mixte

Dans la suite de cette section, nous allons définir un MILP pour résoudre le problème du SLSS. Nous commençons par décrire les notations relatives aux entrées et sorties du SLSS avec un exemple d'instance résolue ainsi que quelques notations complémentaires permettant de simplifier les futures contraintes. Puis, nous présenterons les variables entières et leurs contraintes associées que nous appelons *structurelles*. Ensuite, nous présenterons les variables réelles et leurs contraintes associées que nous appelons *non structurelles*. L'union de ces deux types de variables et contraintes formeront le MILP permettant de résoudre le SLSS. Nous terminerons cette section en exprimant de nouvelles contraintes *structurelles* pour durcir l'ensemble et permettre d'approcher la solution en omettant les variables et contraintes *non structurelles* (pour ne résoudre qu'un ILP).

19.3.1 Variables et Contraintes Structurelles

Les variables sont séparées en deux catégories complémentaires : les variables *structurelles* et *non structurelles*. La première catégorie, les variables *structurelles*, désigne toutes les variables entières permettant de décider quand produire des ressources, quand réaliser un transfert de ces mêmes ressources, ainsi que la variable indiquant la date de fin du processus global. Ce sont elles qui conditionnent la valeur de la fonction objectif, tant en termes de coûts économiques que de date de fin. Les variables *non structurelles* sont des variables réelles et indiquent la quantité à transférer si un transfert a lieu. Ces contraintes permettent de s'assurer de la faisabilité de la solution, mais n'influent pas sur la fonction objectif. Les contraintes sont naturellement découpées en deux catégories similaires : les contraintes *structurelles* (ne dépendant que de

variables *structurelles*) et les contraintes *non structurelles* (dans lesquels des variables *non structurelles* sont impliquées).

Rappelons que $J = \{0, \ldots, M-1\}$ désigne l'ensemble des jobs du *Eater* et $I = \{0, 1, \ldots, N-1\}$ désigne l'ensemble des périodes du *Feeder*. Les variables et contraintes *structurelles* du problème SLSS sont alors, de façon naturelle :

Variables Structurelles :

- $y = (y_i, i = 0, \ldots, N-1)$ une variable booléenne qui représente l'activation de la production de ressources
  donc $y_i = \begin{cases} 1 & \text{si } \textit{Feeder} \text{ active sa production à la période } i \\ 0 & \text{sinon} \end{cases}$
- $z = (z_i, i = 0, \ldots, N-1)$ une variable booléenne qui représente l'activité de la production de ressources
  donc $z_i = \begin{cases} 1 & \text{si } \textit{Feeder} \text{ produit en période } i \\ 0 & \text{sinon} \end{cases}$
- $x = (x_{i,j}, i = 0, \ldots, N-1, j = 0, \ldots, M-1)$, une variable booléenne qui représente les transferts
  donc $x_{i,j} = \begin{cases} 1 & \text{si un transfert a lieu à la période } i \text{ et après le job } j \\ 0 & \text{sinon} \end{cases}$
- T qui représente la date finale de fin du traitement des jobs (*Eater*) puisque nous voulons minimiser un temps d'attente uniquement.

Il s'agit là des variables *cœur* du problème. Des variables auxiliaires leur sont adjointes :

- $\delta = (\delta_i, i = 0, \ldots, N-1)$ une variable booléenne qui représente les transferts vus au niveau des périodes
- $\gamma = (\gamma_j, j = 0, \ldots, M-1)$ une variable booléenne qui représente les transferts vu par les jobs.

Contraintes Structurelles :

La fonction objectif vient comme suit :

- **Fonction Objectif :**

$$min \quad \alpha . \sum_i (Cout_i^{Prod} . z_i + Cout_i^{Act} . y_i) + \beta . T \tag{*}$$

Les contraintes viennent comme suit :

- **Sur T :**
  - La date de fin est supérieure à la date de fin du dernier job :
    Cette date de fin est égale à la somme des durées des jobs restant après la date du dernier transfert.

$$\forall j, \quad T \geq \sum_i x_{i,j} . (p(i+1) + \Delta(j, M)) \tag{S.1}$$

- **Production de ressources :**
  - Pour produire en période 0, la production doit être activée :

$$y_0 - z_0 \geq 0 \tag{S.2}$$

  - Pour produire en période i, si rien n'est produit en période $i-1$, la production doit être activée :

$$\forall i \geq 1, \quad y_i \geq z_i - z_{i-1} \tag{S.3}$$

  - Si un transfert est exécuté en période i, la production est arrêtée :

$$\forall i \geq 0, \quad z_i + \delta_i \leq 1 \tag{S.4}$$

- **Synchronisation :**
  - Calcul des variables représentant les transferts vus par les périodes et par les jobs :

$$\forall i, \quad \delta_i = \sum_j x_{i,j} \tag{S.5}$$

$$\forall j, \quad \gamma_j = \sum_i x_{i,j} \tag{S.6}$$

  - Un transfert au job j n'est possible qu'entre deux périodes :

$$\forall (i,j) \textit{ tel que } (i > \tau_M(j)) \wedge (i < \tau_m(j)), \quad x_{i,j} = 0 \tag{S.7}$$

    Du fait de l'horizon de temps, au job j, il n'est pas possible de faire un transfert avant une certaine période $\tau_m(j)$ puisqu'il n'y a pas assez de temps pour réaliser les jobs $0, \ldots, j-1$. De même, au job j, il n'est pas possible de faire un transfert après une certaine période $\tau_M(j)$ parce qu'il ne restera pas assez de temps pour réaliser les jobs $j+1, \ldots, M-1$.

  - Un transfert interdit certains autres :

$$\forall E \text{ une } \ll\text{-Antichaine}, \quad \sum_{(i,j)\in E} x_{i,j} \leq 1 \tag{S.8}$$

19.3.2 Variables et Contraintes *Non Structurelles*

Le deuxième ensemble de variables nous permettant de compléter le modèle est l'ensemble des variables *non structurelles* $m_{i,j}$ qui fournissent les quantités de ressources associées aux transferts. Les contraintes *non structurelles* associées sont alors :

- **Capacités :**

$$\forall i, j, \quad m_{i,j} \leq C^{Trans} x_{i,j} \tag{NS.1}$$

- **Production :**
  - La production jusqu'à la période i est plus grande que la quantité totale transférée avant cette même période, moyennant le stock initial. Cette contrainte nous permet d'avoir une borne inférieure de la quantité produite jusqu'à la période i.

$$\forall i, \quad \sum_{j,k\leq i} m_{k,j} - \sum_{k<i} z_k.R_k \leq C_0^E \tag{NS.2}$$

  - La production jusqu'à la période i ne peut pas dépasser la capacité du stock de *Feeder*, après avoir soustrait la quantité transférée avant cette même période i. Cette contrainte nous permet d'avoir une borne supérieure de la quantité produite jusqu'à la période i.

$$\forall i, \quad \sum_{k\leq i} z_k.R_k - \sum_{j,k\leq i} m_{k,j} \leq C^E - C_0^E \tag{NS.3}$$

  - La production sur la totalité des périodes est au moins aussi grande que la quantité totale transportée. Cette contrainte nous permet de garantir que le stock à la fin de l'horizon de temps sera revenu à son état initial.

$$\sum_i z_i.R_i - \sum_{i,j} m_{i,j} \geq 0 \tag{NS.4}$$

- **Jobs**

- La quantité totale transportée jusqu'au job j (inclus) est suffisante pour réaliser les jobs jusqu'à ce même job j, moyennant le stock initial et la quantité supplémentaire nécessaire aux transferts. Cette contrainte nous permet d'avoir une borne inférieure de la quantité à transporter entre le job j et $j+1$.

$$\forall j < M, \quad \mu(-1,j) - C_0^F \leq \sum_{i,k \leq j} m_{i,k} - \sum_{k \leq j} \gamma_k.\epsilon_k \tag{NS.5}$$

$$\mu(-1,M) \leq \sum_{i,j} m_{i,j} - \sum_{j} \gamma_j.\epsilon_j \tag{NS.6}$$

- La quantité totale transportée jusqu'au job j (inclus) ne peut pas dépasser la capacité du stock de *Eater*, après avoir soustrait la quantité utilisée jusqu'à ce même job j. Cette contrainte nous permet d'avoir une borne supérieure de la quantité à transporter entre le job j et $j+1$.

$$\forall j, \quad \sum_{i,k \leq j} m_{i,k} - \sum_{k<j} \gamma_k.\epsilon_k \leq C^F - C_0^F + \mu(-1,j) \tag{NS.7}$$

19.3.3 Renforcement des Contraintes *Structurelles*

Les contraintes *structurelles* et *non structurelles* forment un programme hétérogène, mélangeant variables entières et réels. Ce genre de mélange détériore la qualité de la relaxation linéaire, notamment à cause de la contrainte (NS.1), et nous souhaitons ajouter des contraintes valides pour améliorer l'efficacité du solveur. Dans un deuxième temps, nous pourrons essayer d'homogénéiser le modèle en supprimant les contraintes *non structurelles* et en ne gardant que les contraintes *structurelles* et les contraintes *renforcées*. À ce moment, les contraintes *non structurelles* seront séparées par un schéma de Benders, mais nous en parlerons plus en détail dans une section dédiée.

Si les contraintes *non structurelles* sont supprimées, les contraintes *structurelles* seules sont insuffisantes pour obliger l'activation de la production du *Feeder*. Nous proposons de nouvelles contraintes valides sur les variables entières *structurelles* pour mettre des bornes inférieures et supérieures sur la quantité à produire.

- Avant de tomber à court de produit intermédiaire, il y aura au moins 1 transfert :

$$\forall j, \quad \sum_{k \epsilon \Gamma(j)} \gamma_k \geq 1 \tag{R.1}$$

- La production avant la dernière période possible pour un transfert au job j doit être au moins égale à la demande jusqu'au job $j+1$ inclus, moyennant les capacités initiales et le coût des transferts :

$$\forall j, \quad C_0^E + \sum_{k<\tau_M(j)} z_k.R_k \geq \sum_{k<j} \gamma_k.\epsilon_k + \mu(-1,j+1) - C_0^F \tag{R.2}$$

- Chaque transfert ne peut pas transporter plus que C^{Trans} produits. Puisque le nombre de transferts est connu, une borne supérieure de la production vient donc ainsi :

$$\forall i, \quad \sum_{k \leq i} z_k.R_k \leq C_0^E + C^{Trans}.(\sum_{k \leq I} \delta_k) \tag{R.3}$$

- La production ne peut être concentrée dans une période réduite pour ne pas dépasser la capacité de l'entrepôt de *Feeder* dans le cas où ce dernier est vide au début de la période i_1 :

$$\forall i_1, i_2 \; tel \; que \; i_1 < i_2, \quad \sum_{i_1 \leq k \leq i_2} z_k.R_k \leq C^{Trans}.(1 + \sum_{i_1 \leq k \leq i_2} \delta_k) \tag{R.4}$$

- Nous ne connaissons pas les quantités transportées, mais nous connaissons les besoins de *Eater* ainsi que la quantité de produit consommée par les transferts. Une borne inférieure de la production nous permettant de remettre, à la fin de l'horizon de temps, les stocks à leur état initial, vient ainsi :

$$\sum_i z_i.R_i \geq \sum_j (r_j + \epsilon_j.\gamma_j) \tag{R.5}$$

Le problème résultant n'est pas équivalent au problème avec les variables et contraintes *non structurelles*, mais permet d'approcher la solution lorsque ces dernières sont omises.

19.3.4 Deux programmes linéaires pour la résolution du problème du SLSS

Le programme linéaire minimal du problème du SLSS est donc formé par la concaténation des variables et contraintes *structurelles* et *non structurelles* :

(SLSS) :

minimiser $\alpha.\sum_i (Cout_i^{Prod}.z_i + Cout_i^{Act}.y_i) + \beta.T$

tel que

$$T \geq \sum_i x_{i,j}.(p(i+1) + \Delta(j,M)) \quad \forall j = 0,\dots,M-1 \qquad (S.1)$$

$$y_0 - z_0 \geq 0 \qquad (S.2)$$

$$y_i \geq z_i - z_{i-1} \quad \forall i = 1,\dots,N-1 \qquad (S.3)$$

$$z_i + \delta_i \leq 1 \quad \forall i = 0,\dots,N-1 \qquad (S.4)$$

$$\delta_i = \sum_j x_{i,j} \quad \forall i = 0,\dots,N-1 \qquad (S.5)$$

$$\gamma_j = \sum_i x_{i,j} \quad \forall j = 0,\dots,M-1 \qquad (S.6)$$

$$x_{i,j} = 0 \quad \forall (i,j) \text{ t.q. } (i > \tau_M(j)) \wedge (i < \tau_m(j)) \qquad (S.7)$$

$$\sum_{(i,j)\in E} x_{i,j} \leq 1 \quad \forall E \text{ une } \ll\text{-Antichaine} \qquad (S.8)$$

$$m_{i,j} \leq C^{Trans} x_{i,j} \quad \forall i = 0,\dots,N-1, \forall j = 0,\dots,M-1 \qquad (NS.1)$$

$$\sum_{j,k\leq i} m_{k,j} - \sum_{k<i} z_k.R_k \leq C_0^E \quad \forall i = 0,\dots,N-1 \qquad (NS.2)$$

$$\sum_{k\leq i} z_k.R_k - \sum_{j,k\leq i} m_{k,j} \leq C^E - C_0^E \quad \forall i = 0,\dots,N-1 \qquad (NS.3)$$

$$\sum_i z_i.R_i - \sum_{i,j} m_{i,j} \geq 0 \qquad (NS.4)$$

$$\mu(-1,j) - C_0^F \leq \sum_{i,k\leq j} m_{i,k} - \sum_{k\leq j} \gamma_k.\epsilon_k \quad \forall j = 0,\dots,M-1 \qquad (NS.5)$$

$$\mu(-1,M) \leq \sum_{i,j} m_{i,j} - \sum_j \gamma_j.\epsilon_j \qquad (NS.6)$$

$$\sum_{i,k\leq j} m_{i,k} - \sum_{k<j} \gamma_k.\epsilon_k \leq C^F - C_0^F + \mu(-1,j) \quad \forall j = 0,\dots,M-1 \qquad (NS.7)$$

Proposition 6. *Les contraintes structurelles et non structurelles sont nécessaires et suffisantes pour décrire le problème du SLSS.*

Preuve 7. Il est facile de se convaincre que les contraintes *structurelles* et *non structurelles* sont nécessaires grâce aux explications fournies pour chaque contrainte. Il est cependant plus compliqué de montrer qu'ensembles, elles sont suffisantes pour décrire le problème.

L'information manquante dans le programme linéaire, grâce à laquelle nous pouvons complètement décrire les solutions du problème, sont les valeurs des stocks à tout instant. Puisque que les variables décrivent la production, les transferts et les dates de début des jobs, si les contraintes permettent de garantir que la valeur des stocks est toujours possible, alors l'ensemble des contraintes sont suffisantes.

La valeur du stock de *Feeder* à la fin de chaque période i est égale à la somme du stock initial et de la production jusqu'à la période i (incluse) à laquelle les quantités transférées sont soustraites (c'est-à-dire $C_0^F + \sum_{k\leq i} z_k.R_k - \sum_{j,k\leq i} m_{k,j}$). La valeur de ce stock est toujours positive grâce à la contrainte (NS.2) et est inférieur à la limite C^F du stock grâce à la contrainte (NS.3). La contrainte (NS.4) nous garantit que la valeur du stock de *Feeder* reviendra, à la fin de l'horizon de temps, à son niveau initial.

De même, la valeur du stock de *Eater* à la fin de chaque job j (transfert inclus) est égale à la somme du stock initial et des quantités transférées à laquelle la consommation des jobs et des transferts est soustraite (c'est-à-dire $C_0^E + \sum_{i,k\leq j} m_{i,k} - \mu(-1,j) - \sum_{k\leq j} \gamma_k.\epsilon_k$). La valeur de ce stock est toujours positive grâce à la contrainte (NS.5) et est inférieur à la limite C^E du stock grâce à la contrainte (NS.7). La contrainte (NS.6) nous garantit que la valeur du stock de *Eater* reviendra, à la fin de l'horizon de temps, à son niveau initial.

Une solution de ce programme linéaire est donc toujours une solution du problème énoncé et inversement. ■

Cependant, comme expliqué plus tôt, le programme mixte (SLSS) possède des variables hétérogènes qui impactent négativement la relaxation linéaire. Une façon d'améliorer la résolution est d'ajouter des contraintes valides, ici appelées *renforcées*, et de résoudre le problème durci :

(SLSS Durci) :

minimiser $\alpha.\sum_i (Cout_i^{Prod}.z_i + Cout_i^{Act}.y_i) + \beta.T$

tel que

$$T \geq \sum_i x_{i,j}.(p(i+1) + \Delta(j,M)) \quad \forall j = 0,\dots,M-1 \quad (S.1)$$
$$y_0 - z_0 \geq 0 \quad (S.2)$$
$$y_i \geq z_i - z_{i-1} \quad \forall i = 1,\dots,N-1 \quad (S.3)$$
$$z_i + \delta_i \leq 1 \quad \forall i = 0,\dots,N-1 \quad (S.4)$$
$$\delta_i = \sum_j x_{i,j} \quad \forall i = 0,\dots,N-1 \quad (S.5)$$
$$\gamma_j = \sum_i x_{i,j} \quad \forall j = 0,\dots,M-1 \quad (S.6)$$
$$x_{i,j} = 0 \quad \forall (i,j) \text{ t.q. } (i > \tau_M(j)) \wedge (i < \tau_m(j)) \quad (S.7)$$
$$\sum_{(i,j)\in E} x_{i,j} \leq 1 \quad \forall E \text{ une } \ll\text{-Antichaine} \quad (S.8)$$

$$m_{i,j} \leq C^{Trans} x_{i,j} \quad \forall i = 0,\dots,N-1, \forall j = 0,\dots,M-1 \quad (NS.1)$$
$$\sum_{j,k\leq i} m_{k,j} - \sum_{k<i} z_k.R_k \leq C_0^E \quad \forall i = 0,\dots,N-1 \quad (NS.2)$$
$$\sum_{k\leq i} z_k.R_k - \sum_{j,k\leq i} m_{k,j} \leq C^E - C_0^E \quad \forall i = 0,\dots,N-1 \quad (NS.3)$$
$$\sum_i z_i.R_i - \sum_{i,j} m_{i,j} \geq 0 \quad (NS.4)$$
$$\mu(-1,j) - C_0^F \leq \sum_{i,k\leq j} m_{i,k} - \sum_{k\leq j} \gamma_k.\epsilon_k \quad \forall j = 0,\dots,M-1 \quad (NS.5)$$
$$\mu(-1,M) \leq \sum_{i,j} m_{i,j} - \sum_j \gamma_j.\epsilon_j \quad (NS.6)$$
$$\sum_{i,k\leq j} m_{i,k} - \sum_{k<j} \gamma_k.\epsilon_k \leq C^F - C_0^F + \mu(-1,j) \quad \forall j = 0,\dots,M-1 \quad (NS.7)$$

$$\sum_{k\in\Gamma(j)} \gamma_k \geq 1 \quad \forall j = 0,\dots,M-1 \quad (R.1)$$
$$C_0^E + \sum_{k<\tau_M(j)} z_k.R_k \geq \sum_{k<j} \gamma_k.\epsilon_k + \mu(-1,j+1) - C_0^F \quad \forall j = 0,\dots,M-1 \quad (R.2)$$
$$\sum_{k\leq i} z_k.R_k \leq C_0^E + C^{Trans}.(\sum_{k\leq I} \delta_k) \quad \forall i = 0,\dots,N-1 \quad (R.3)$$
$$\sum_{i_1\leq k\leq i_2} z_k.R_k \leq C^{Trans}.(1 + \sum_{i_1\leq k\leq i_2} \delta_k) \quad \forall i_1, i_2 \text{ tel que } i_1 < i_2 \quad (R.4)$$
$$\sum_i z_i.R_i \geq \sum_j (r_j + \epsilon_j.\gamma_j) \quad (R.5)$$

19.4 Séparation des Contraintes d'antichaine

Cependant, pour résoudre les problèmes du (SLSS) et (SLSS Durci), il est nécessaire de séparer les contraintes de $\ll$-antichaine (S.8) car elles sont en nombre exponentiel. Nous allons donc vérifier, pour un nœud du *Branch&Bound* donné, si la solution courante viole une des contraintes et, dans ce cas, ajouter une contrainte violée au programme linéaire.

Afin de déterminer si une solution fractionnaire viole au moins une contrainte de $\ll$-antichaine (S.8), on peut procéder en cherchant un flot de coût minimum dans un graphe où un certain ensemble d'arc représente une antichaine. Ce graphe peut être défini comme $G^{\ll} = (\Lambda, E^{\ll})$. Λ est l'ensemble des nœuds représentés par des couples (i, j) dont $(-1, -1)$ et (N, M). $E^{\ll}$ est l'ensemble des arcs reliant deux nœuds (i_1, j_1) et (i_2, j_2) si et seulement si $(i_1, j_1) \ll (i_2, j_2)$, en rappelant que tout couple (i, j) est tel que $(-1, -1) \ll (i, j) \ll (N, M)$.

Un vecteur $X = (x_{i,j})_{i=0,\dots,N-1,j=0,\dots,M-1}$ étant donné, la contrainte d'antichaine sera violée si on ne peut pas trouver de flot F qui envoie, au plus, une unité de flot de $(-1, -1)$ à (N, M) et qui soit tel qu'en tout (i, j) différent de $(-1, -1)$ et (N, M), le flot F entrant en (i, j) est au moins égale à $x_{i,j}$.

Cette approche peut s'implémenter via une formulation PLNE du problème de flot, mais la résolution sera extrêmement lente, car le graphe $G^{\ll}$ a un nombre de sommets en $(N.M)^2$ et est différent à chaque nœud du *Branch&Bound*. La résolution d'un algorithme exact induit une lourdeur trop importante pour une recherche de coupe et la résolution du *Branch&Bound* en sera très fortement pénalisé. Il faut donc implémenter un algorithme de flot réalisable pour pouvoir séparer rapidement et efficacement ces contraintes.

La méthode la plus simple est alors de procéder comme pour une recherche de stable de poids max dans un graphe. L'idée est de sélectionner l'un des deux ou trois nœuds de plus grande valeur et d'explorer les sous-problèmes récursivement. L'ensemble des nœuds de *backtracking* de cette exploration correspond alors à une suite d'éléments $(i_1, j_1), \dots, (i_k, j_k)$ telle que $x_{i_1,j_1} \geq x_{i_2,j_2} \geq \cdots \geq x_{i_k,j_k}$ et que les éléments $(i_1, j_1), \dots, (i_k, j_k)$ constituent une antichaine dans Λ. On génère alors une antichaine avec l'algorithme 19.1. Sans limite de nombre de nœuds explorés, cet algorithme est exact, mais nous ne prendrons en compte que les deux nœuds de plus grande valeur en supposant que l'une d'elles apparaisse toujours dans la solution optimale. Nous explorons donc l'ensemble des solutions grâce à un arbre de décision binaire.

Algorithme 19.1 : Recherche d'une antichaine violée

Entrées : Q la profondeur de l'arbre de *backtracking*
Entrées : L la liste des $(x_{i,j})_{i,j}$ triée par ordre décroissant
Entrées : A une antichaine initialement vide
  $left = L[0]; right = L[1]$
  Retirer $L[0]$ et $L[1]$ de L
  $(Trouve, A) =$ **Arbre**$(A, Q-1, L, left)$ [voir alg 19.2]
  Si non Trouve **Alors**
    $(Trouve, A) =$ **Arbre**$(A, Q-1, L, right)$ [voir alg 19.2]
  Fin Si
  Si Trouve **Alors**
    Ajouter la contrainte $\sum_{c \in A} x_c \leq 1$ au problème principal
  Fin Si

Afin d'accélérer la méthode, et ainsi de ne pas trop pénaliser la résolution du problème du SLSS, l'exploration de l'arbre de décision s'arrêtera à une profondeur arbitraire de 5. Cela veut dire que nous prendrons cinq décisions successives parmi les deux nœuds de plus grande valeur de chaque sous-problème, et l'antichaine est terminée en sélectionnant itérativement le nœud de plus grande valeur parmi les nœuds restants, et jamais le deuxième.

Algorithme 19.2 : Fonction récurrente pour la recherche d'antichaines violées

Fonction Arbre(A, Q, L, x_c)
  Ajouter c à A
  Retirer x_c et tous les x comparables à x_c de la liste L

  Si $Q = 0$ **Alors**
    Tant que L non vide **Faire**
      $x_c = L[0]$
      Ajouter c à A
      Retirer x_c et tous les x comparables à x_c de la liste L
    Fin Tant que
    Retourner $(\sum_{c \in A} x_c > 1,\ A)$
  Fin Si

  Si $Q > 0$ **Alors**
    $left = L[0]; right = L[1]$
    retirer $L[0]$ et $L[1]$ de L
    $(Trouve, A) =$ **Arbre**$(A, Q-1, L, left)$ [récursion]
    Si non Trouve **Alors**
      $(Trouve, A) =$ **Arbre**$(A, Q-1, L, right)$ [récursion]
    Fin Si
    Retourner $(Trouve, A)$
  Fin Si

Fin Fonction

Cette recherche arborescente d'antichaine violée est pratique lorsque que les valeurs sont réelles mais, dans le cas d'une solution entière, cet algorithme n'est pas nécessaire. Pour vérifier si une solution entière respecte bien toutes les contraintes d'antichaines, il suffit de vérifier les transferts dans l'ordre temporel croissant et vérifier qu'ils forment bien une chaîne au sens de $\ll$. En effet, avoir des valeurs entières signifie que nous avons un ensemble de transferts. Il suffit alors de vérifier que cet ensemble forme une chaîne au sens de $\ll$. Si ce n'est pas le cas, il suffit de prendre deux transferts incompatibles et d'ajouter la contrainte d'antichaine correspondante aux coupes du problème.

Section 20

Application d'un schéma de Benders

Une autre manière d'accélérer la résolution est d'homogénéiser le programme en utilisant un schéma de décomposition de Benders [Benders, 1962]. Ce schéma permet de découpler un problème en deux si les contraintes ont une structure en blocs, tout en gardant les garanties d'optimalités. Le problème constitué par le premier bloc de contraintes est utilisé comme problème principal, et le second bloc de contraintes est vérifié pendant le *Branch&Bound* du problème principal. Si au moins une contrainte du second bloc est violée, une coupe appelée coupe de Benders est créée et ajoutée au problème principal.

Cette décomposition nous permet de résoudre le problème du SLSS par la résolution de deux problèmes homogènes : le problème principal ne garde que les variables *structurelles* (et donc que les contraintes *structurelles* et les contraintes *renforcées*) et le problème secondaire vérifie les contraintes sur les variables *non structurelles*. Si des contraintes violées sont trouvées lors de la résolution de ce sous-problème, des coupes seront ajoutées au problème en variables entières. Le problème avec les variables et contraintes *structurelles* et *renforcées* est appelé le problème maître (SLSS Maître) et le sous-problème avec les variables et contraintes *non structurelles* est appelé le problème esclave et est vérifié pendant la résolution par *Branch&Bound* du problème maître.

Pour être plus précis, nous allons réaliser une décomposition de Benders du problème du SLSS en partant de la formulation suivante :

(SLSS)

$$\begin{aligned} &min\ C.V \\ &t.q.\ A.V \geq a \\ &\quad B.V + Q.m \geq b \end{aligned}$$

où V regroupe toutes les variables entières (x, y, z, δ, γ).
Le problème maître est donc la restriction de ce problème aux seules variables entières V, c'est-à-dire :

(SLSS Maître) :

$$\begin{aligned} &min\ C.V \\ &t.q.\ A.V \geq a \end{aligned}$$

Autrement dit, le problème maître est la restriction du (SLSS) aux contraintes *structurelles* et *renforcées* :

(SLSS Maître) :

$$\textbf{minimiser} \quad \alpha.\sum_i (Cout_i^{Prod}.z_i + Cout_i^{Act}.y_i) + \beta.T$$

$$\begin{aligned} \textbf{tel que} \quad & T \geq \sum_i x_{i,j}.(p(i+1) + \Delta(j,M)) && \forall j = 0,\dots,M-1 && (S.1) \\ & y_0 - z_0 \geq 0 && && (S.2) \\ & y_i \geq z_i - z_{i-1} && \forall i = 1,\dots,N-1 && (S.3) \\ & z_i + \delta_i \leq 1 && \forall i = 0,\dots,N-1 && (S.4) \end{aligned}$$

$$\delta_i = \sum_j x_{i,j} \qquad \forall i = 0,\dots,N-1 \qquad (S.5)$$

$$\gamma_j = \sum_i x_{i,j} \qquad \forall j = 0,\dots,M-1 \qquad (S.6)$$

$$x_{i,j} = 0 \qquad \forall (i,j) \text{ t.q. } (i > \tau_M(j)) \wedge (i < \tau_m(j)) \qquad (S.7)$$

$$\sum_{(i,j)\in E} x_{i,j} \leq 1 \qquad \forall E \text{ une } \ll\text{-Antichaine} \qquad (S.8)$$

$$\sum_{k\in\Gamma(j)} \gamma_k \geq 1 \qquad \forall j = 0,\dots,M-1 \qquad (R.1)$$

$$C_0^E + \sum_{k<\tau_M(j)} z_k.R_k \geq \sum_{k<j} \gamma_k.\epsilon_k + \mu(-1,j+1) - C_0^F \qquad \forall j = 0,\dots,M-1 \qquad (R.2)$$

$$\sum_{k\leq i} z_k.R_k \leq C_0^E + C^{Trans}.(\sum_{k\leq I} \delta_k) \qquad \forall i = 0,\dots,N-1 \qquad (R.3)$$

$$\sum_{i_1\leq k\leq i_2} z_k.R_k \leq C^{Trans}.(1 + \sum_{i_1\leq k\leq i_2} \delta_k) \qquad \forall i_1, i_2 \textit{ tel que } i_1 < i_2 \qquad (R.4)$$

$$\sum_i z_i.R_i \geq \sum_j (r_j + \epsilon_j.\gamma_j) \qquad (R.5)$$

Le problème esclave, quant à lui, considère l'ensemble de variables V comme fixé et cherche à vérifier si une des contraintes omises dans le problème maître est violée. Le problème esclave cherche donc à trouver une solution réalisable du (SLSS) connaissant les valeurs de l'ensemble V, c'est-à-dire chercher des valeurs de m qui respectent les contraintes *non structurelles*.

(SLSS Esclave) :

existe-t-il m

$$t.q.\ Q.m \geq b - B.V$$

Pour être plus précis, en remplaçant Q, b, B et V, le problème esclave est :

(SLSS Esclave) :

existe-t-il m

tel que

$$m_{i,j} \leq C^{Trans} x_{i,j} \qquad \forall i = 0,\dots,N-1, \forall j = 0,\dots,M-1 \qquad (NS.1)$$

$$\sum_{j,k\leq i} m_{k,j} \leq C_0^E + \sum_{k<i} z_k.R_k \qquad \forall i = 0,\dots,N-1 \qquad (NS.2)$$

$$\sum_{j,k\leq i} m_{k,j} \geq \sum_{k\leq i} z_k.R_k - C^E + C_0^E \qquad \forall i = 0,\dots,N-1 \qquad (NS.3)$$

$$\sum_{i,j} m_{i,j} \leq \sum_i z_i.R_i \qquad (NS.4)$$

$$\sum_{i,k\leq j} m_{i,k} \geq \mu(-1,j) - C_0^F + \sum_{k\leq j} \gamma_k.\epsilon_k \qquad \forall j = 0,\dots,M-1 \qquad (NS.5)$$

$$\sum_{i,j} m_{i,j} \geq \mu(-1,M) + \sum_j \gamma_j.\epsilon_j \qquad (NS.6)$$

$$\sum_{i,k\leq j} m_{i,k} \leq C^F - C_0^F + \mu(-1,j) + \sum_{k<j} \gamma_k.\epsilon_k \qquad \forall j = 0,\dots,M-1 \qquad (NS.7)$$

Si le problème esclave a une solution, elle est de valeur 0 (trouver une solution réalisable revient à minimiser une fonction objectif vide et est donc toujours de valeur 0), alors la valeur du problème dual est également 0. Si le problème esclave n'a pas de solution, les variables duales λ sont telles que $Q^T.\lambda \leq 0$ et $\lambda.(b - B.V) > 0$. Pour éviter d'avoir une solution avec des valeurs trop petite (et inutilisable car trop proche de la précision du solveur utilisé), nous allons résoudre le problème dual directement en ajoutant une condition pour obtenir des valeurs significatives de λ :

existe-t-il λ

$$t.q.\ (b - B.V)^T.\lambda \geq 1$$

$$Q^T.\lambda \leq 0$$

Si ce problème a une solution, alors le problème esclave n'en a pas et la coupe $(b - B.V)^T.\lambda \leq 0$ peut être ajoutée au problème maître.

En pratique, la matrice Q est la même pour toutes les résolutions du problème esclave, seules les valeurs de $b - B.V$ changent en fonction de la solution courante. Il est donc très facile et rapide d'implémenter le programme linéaire à chaque fois qu'il faut le vérifier. Aussi, puisque les variables sont toutes réelles et que nous ne cherchons à savoir que si une solution existe, la résolution est très rapide et peut être utilisé pendant le *Branch&Bound* du problème maître sans être trop lourd à vérifier.

Section 21

Recherche de chaîne dans un ensemble approprié d'états

En regardant le problème sous un autre angle, il est possible de se convaincre qu'une solution est principalement déterminée par les opérations de transfert : entre deux transferts, une borne inférieure et supérieure des ressources nécessaires sont connues et il existe un nombre limité d'ensembles de productions de coûts optimaux. De plus, l'ensemble des transferts définit une chaîne dans l'ensemble ordonné Λ incluant les deux éléments $(-1,-1)$ et (N,M), voir Figure 21.1. Il vient alors qu'un algorithme de plus court chemin dans l'ensemble Λ est donc capable de trouver la solution optimale. Nous allons construire un algorithme de type A*.

21.1 Formulation de notre problème comme une recherche de chaîne dans un ensemble ordonné

Soit $L^{string} = \{(i_0,j_0) = (-1,-1),(i_1,j_1),(i_2,j_2),\ldots,(i_S,j_S),(i_{S+1},j_{S+1}) = (N,M)\}$ une chaîne de transferts. Pour tout $s = 0,\ldots,S$, soit W_s la quantité de ressources produites entre le transfert (i_s,j_s) et le transfert (i_{s+1},j_{s+1}), avec les conventions suivantes :

- Pour $s = 0$, W_0 désigne la quantité de ressources produites avant le premier transfert.
- Pour $s = S$, W_S désigne la quantité de ressources produites après le dernier transfert.

Résoudre le problème du SLSS revient à calculer la chaîne L^{chaine} et la séquence $W = \{W_0,\ldots,W_S\}$ telles que :

- Chaque W_s est réalisable et a une valeur de coût Q_s pour produire cette quantité
- $\sum_s W_s \geq \sum_s \mu(j_s,j_{s+1})$
- $\forall\ s = 1,\ldots,S,\ W_s \leq C^F$
- $\forall\ s = 1,\ldots,S,\ C_0^E + C_0^F + \sum_{k=0,\ldots,s-1} W_k \geq \sum_{k=0,\ldots,s} \mu(j_k,j_{k+1})$
- $\forall\ s = 1,\ldots,S,\ C_0^E + C_0^F + \sum_{k=1,\ldots,s} W_k - \sum_{k=0,\ldots,s-1} \mu(j_k,j_{k+1}) \leq C^E + C^F.$
- La fonction objectif de notre problème $(\alpha.\sum_{s=0,\ldots,S} Q_s + \beta.T)$ est minimale.

Une attention particulière est portée à la première transition (en partant de $(-1,-1)$) et à la dernière transition (en arrivant à (N,M)) :

- Pour que la première transition soit valide, les deux contraintes suivantes doivent être satisfaites :
  - la décision W_0 est telle que $W_0 \leq C^F - C_0^F$.
  - $C_0^E \geq \mu(j_0,j_1)$
- Pour que la dernière transition soit valide, l'état doit satisfaire la contrainte suivante :
  - $C^E \geq \mu(j_S,M) + C_0^E$

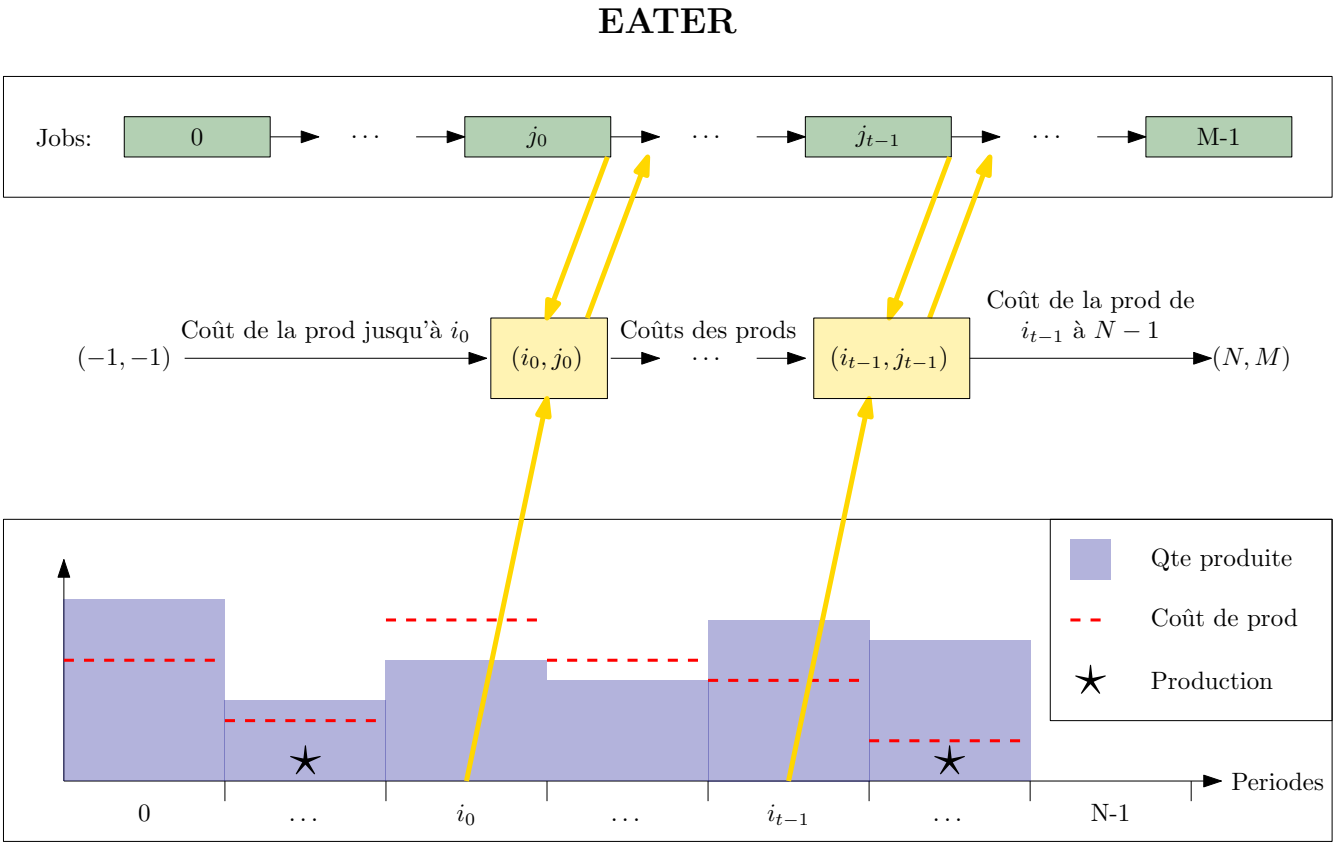

FIGURE 21.1 – Chaîne de transferts dans le processus *Feeder/Eater*

21.2 Pré-traitement

Afin de lister toutes les valeurs W_s possibles et de calculer le coût minimum possible pour produire cette quantité, nous avons utilisé un algorithme de programmation dynamique pour construire une table TAB, indexée sur les couples d'indices (i_1, i_2) avec $-1 \leq i_1 < i_2 \leq N$, tel que $TAB[i_1, i_2]$ contient une liste de couples $(R \leq C^T, z)$ rangés par valeurs croissantes de R.

Ici, z est la solution optimale de la restriction du problème de production aux périodes i telles que $i_1 < i < i_2$, puisque la production recherchée est exactement R :

Calculez z avec des valeurs dans $\{0,1\}$ et avec la sémantique usuelle, telle que :

- Minimiser $Q = \sum_i Cout_i^{Prod}.z_i + \sum_i Cout_i^{Act} y_i$
- $\sum_i R_i.z_i = R$
- Pour tout $i = i_1 + 1, \ldots, i_2 - 1$,
  $y_i = 1 \Leftrightarrow (z_i = 1) \wedge (z_{i-1} = 0)$
- $z_{i_1} = 0$

Pour rappel, la production entre i_1 et i_2 est entourée de deux transferts et donc nécessairement $z_{i_1} = z_{i_2} = 0$.

21.3 Algorithme de plus court chemin

Le problème peut alors être résolu grâce à un algorithme de recherche de chemin dans un espace d'état adapté. Nous présenterons, dans cette section, cet espace d'état dans lequel nous chercherons la solution, puis nous présenterons l'algorithme de recherche heuristique A* que nous avons déjà utilisé dans un chapitre précédent.

Espace d'état Σ :

Un état de Σ est un triplet $\sigma = (i, j, \omega)$ tel que : $(i, j) \in \Lambda$ est un transfert possible et $0 \leq \omega \leq C^E + C^F$ est la somme des quantités de produit présent dans les deux stocks. Pour rappel, un transfert est valide si $\tau_m(j) \leq i \leq \tau_M(j)$, c'est-à-dire si la date $p.i$ au début de la période i est suffisamment tardive pour réaliser les jobs 0 à j et laisse suffisamment de temps pour réaliser les jobs $j+1$ à $M-1$.

Il existe également plusieurs états virtuels dans $\boldsymbol{\Sigma}$:

- un état source $(-1, -1, C_0^E + C_0^F)$ qui représente l'état initial du problème SLSS,
- des états puits (N, M, ω) tel que $\omega \geq C_0^E + C_0^F$ qui représente des états finaux possibles vérifiant les contraintes de valeur de stocks en fin d'horizon de temps.

Aussi, pour un transfert (i, j) donné, toutes les valeurs de ω sont valides tant qu'elles sont entre 0 et $C^E + C^F$. Cependant, certaines de ces valeurs ne sont pas accessibles depuis l'état source $(-1, -1, C_0^E + C_0^F)$, c'est-à-dire qu'il n'existe pas de chemin de l'état source vers cet état.

Un arc entre deux états σ_1 et σ_2 :

Pour décrire ces chemins que nous cherchons, il nous faut définir les arcs entre deux états.
Un arc entre $\sigma_1 = (i_1, j_1, \omega_1)$ et $\sigma_2 = (i_2, j_2, \omega_2)$ est commandée par une décision W et existe si :

(1) $(i_1, j_1) \ll (i_2, j_2)$
(2) W doit être dans la liste $TAB(i_1, i_2)$, avec une valeur de coût de production Q
(3) $\mu(j_1, j_2) \leq C^{Trans}$
(4) $\omega_1 \geq \mu(j_1, j_2)$
(5) $W \leq C^F$
(6) $\omega_2 = \omega_1 + W - \mu(j_1, j_2) \geq 0$

Les contraintes temporelles sont toujours respectées car sinon, l'état n'existe pas.

Les transitions depuis l'état source $(-1, -1, C_0^E + C_0^F)$ vers un état (i, j, ω) sont cependant légèrement différentes :

(1) $(-1, -1) \ll (i, j)$ est toujours vrai
(2) W doit être dans la liste $TAB(-1, i)$, avec une valeur de coût de production Q
(3) $\mu(-1, j) \leq C_0^E$
(4) $C_0^F + W \leq C^F$
(5) $\omega = C_0^E + C_0^F + W - \mu(-1, j) \geq 0$

Le chemin le plus court dans cet espace d'état Σ :

Le chemin optimal que nous recherchons dans l'espace Σ est celui qui minimise la fonction objectif du problème SLSS :

$$\alpha \bar{Q} + \beta T$$

où $\bar{Q}$ est la somme des coûts de production Q liés aux transitions successives et $T = p.\bar{i} + \Delta(\bar{j}, M)$ est la date de fin du dernier job avec $(\bar{i}, \bar{j}, \bar{\omega}) \to (N, M, \omega)$ la dernière transition du chemin considéré.

Recherche du plus court chemin

L'algorithme de recherche heuristique A* est, ici encore, utilisé pour son efficacité et sa garantie d'optimalité (dépendant de l'heuristique guide). Il utilise une file à priorité dans laquelle sont gardés les états en cours d'exploration. Cette file est triée par ordre croissant de la somme de leur valeur actuelle et de la valeur calculée par l'heuristique guide qui donne une borne inférieure de la valeur restante jusqu'à l'état final (voir ci-après). Les états les plus pertinents seront utilisés en priorité pour continuer l'exploration.

Au cours de l'exécution de A*, à partir du meilleur état (premier de la file à priorité), toutes les transitions possibles sont évaluées et les états résultants sont insérés au bon endroit dans la file à priorité. L'algorithme continu tant qu'il reste des états prometteurs (dont leur valeur guide est plus petite que toutes les solutions trouvées jusqu'à présent). Si la liste d'exploration est vide ou que la borne inférieure de la valeur des états non explorés est plus grande que la meilleure solution, l'algorithme s'arrête.

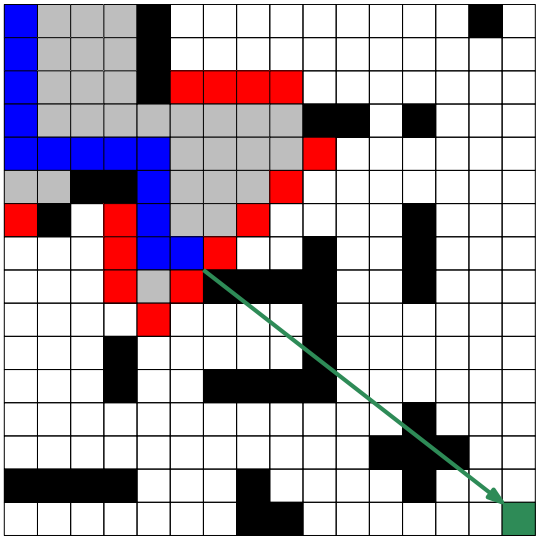

FIGURE 21.2 – Une photo au milieu de l'exécution de l'algorithme A*. Code couleur : les murs sont en noir, les états explorés en gris, les états dans la file à priorité en rouge, le plus court chemin actuel en bleu, et la destination et la valeur heuristique guide en vert.

Guidage d'exploration (borne inférieure de la valeur restante) :

Pour calculer une borne inférieure de la valeur restante jusqu'à un état puits (N, M, ω), nous procédons comme suit : puisque nous connaissons la quantité restante à produire, à savoir $\mu(j, M) + C_0^F + C_0^E - \omega$, il suffit de prendre le coût de production calculé en prétraitement pour une quantité au plus égale au besoin restant. Pour essayer d'avoir la borne inférieure la plus serrée possible, nous prendrons le coût $Cout^{LB}$ associé à la production de la plus grande quantité qui soit inférieure ou égale au besoin restant. La borne inférieure d'un état peut donc être calculée de la même manière que la valeur objectif du problème : $LB(\sigma = (i, j, \omega)) = \alpha.Cout^{LB} + \beta.T$ où $T = p.i + \Delta(j, M)$ est la date de fin si aucun autre transfert n'a lieu après (sans vérifier si c'est possible).

Proposition 7. *L'algorithme A*, ainsi présenté, résout optimalement le SLSS.* ▼

Preuve 8. De la même manière que le MILP, cet algorithme ne travaille pas explicitement avec les valeurs des stocks de *Eater* et *Feeder*. Toutes les autres contraintes sont respectées et la fonction à minimiser est la même, donc si les deux stocks sont valides à tout instant, alors A* résout bien le SLSS.

La première transition (à partir de l'état source $(-1, -1, C_0^E + C_0^F)$) est toujours valide. Les transitions suivantes partent du principe qu'un maximum de produit transite de *Feeder* à *Eater* à chaque transfert. Puisque chaque transfert coûte cher, peu de transferts seront réalisés. Il y a donc deux cas de figure que

nous allons présenter en utilisant deux nouvelles notations : ω^F et ω^E sont la quantité de produit dans le stock de *Feeder* et *Eater* respectivement.

Nous allons regarder ce qu'il se passe lors d'une transition entre deux états $\sigma_1 = (i_1, j_1, \omega_1)$ et $\sigma_2 = (i_2, j_2, \omega_2)$:

- Si $C^E \geq C^F$, alors $C^{Trans} = C^F$ et, après le transfert, le stock de *Feeder* sera vide (c'est-à-dire $\omega^F = 0$) et la totalité des produits est dans le stock de *Eater* ($\omega^E = \omega_2$). Lors de la recherche de la transition suivante :
  - La contrainte (1) s'assure que le couple suivant n'est incompatible avec aucune transition précédente, car la relation est transitive.
  - La contrainte (2) s'assure que la quantité produite est réalisable avec un coût minimal.
  - La contrainte (3) s'assure que, si le stock de *Eater* était vide au départ ($\omega^E = 0$), la quantité après le transfert ne peut pas être supérieur à C^{Trans}, la quantité utilisable est donc C^{Trans} au plus. Cependant, cette contrainte n'est pas très importante dans ce cas de figure puisque la contrainte (4) est plus restrictive.
  - La contrainte (4) s'assure que la quantité de produit utilisable est ω^E au plus.
  - La contrainte (5) s'assure que la quantité produite ne fait pas déborder le stock de *Feeder*.
  - La contrainte (6) s'assure que la quantité de produit après le transfert est compatible avec l'état post-transition.
- Si $C^E < C^F$, alors $C^{Trans} = C^E$ et, après le transfert, le stock de *Eater* sera plein (c'est-à-dire $\omega^E = C^E$) et une partie des produits est toujours dans le stock de *Feeder* ($\omega^F = \omega_2 - C^E$). Lors de la recherche de la transition suivante :
  - La contrainte (1) s'assure que le couple suivant n'est incompatible avec aucune transition précédente, car la relation est transitive.
  - La contrainte (2) s'assure que la quantité produite est réalisable avec un coût minimal.
  - La contrainte (3) est ici plus contraignante que la contrainte (4) puisque $C^{Trans} = C^E$. Ainsi, la contrainte s'assure que la quantité consommée est plus petite que la capacité totale du stock.
  - La contrainte (4) sera toujours vrai dans ce cas à cause de la contrainte (3)
  - La contrainte (5) s'assure que la quantité produite ne fait pas déborder le stock de *Feeder*.
  - La contrainte (6) s'assure que la quantité de produit après le transfert est compatible avec l'état post-transition.

Nous avons donc des valeurs de stocks valides à tout instant, des valeurs de productions valides et optimales à tout instant et les garanties d'optimalités de la méthode nous assure que l'algorithme A* ainsi présenté résout optimalement le problème du SLSS. ■

Section 22

Expériences numériques

22.1 Génération d'instances

Quantités et coûts de production
Afin d'imiter le marché de l'électricité, le montant et les coûts de production seront générés de la même manière que dans la Figure 22.1 : les périodes de production sont regroupées en super-périodes de même longueur. Chaque super-période se voit attribuer une classe de quantité de production (*Faible*, *Moyen* ou *Eleve*) et de coût de production (*Bas*, *Moyen* ou *Eleve*). Ces deux classes sont indépendantes l'une de l'autre, il peut y avoir une quantité faible à coût élevé comme une quantité élevée à coût faible.

Ensuite, la quantité et le coût réel de production de chaque période seront générés selon les classes de leur super-période. Nous choisissons d'associer un montant aléatoire uniformément généré dans un intervalle $[\frac{1}{2}.Mean_{classe}^{Quantite}, \frac{3}{2}.Mean_{classe}^{Quantite}]$ et un coût aléatoire généré uniformément dans un intervalle $[\frac{1}{2}.Mean_{classe}^{Cout}, \frac{3}{2}.Mean_{classe}^{Cout}]$ à chaque période. Enfin, pour s'assurer de ne pas surcharger le système, les demandes de ressources sont générées de telle sorte que $Mean^{Demande} = 0.75 * Mean^{Production}$.

Tailles de réservoir
Pour avoir au moins deux transferts, nous proposons de choisir C^T tel que $\frac{\mu(-1,M)}{C^T} \in [2,3]$. Ensuite, la taille de l'autre réservoir peut être calculée telle que $\frac{C^E}{C^F} \in [0.5,2]$.

Coûts d'activation
Pour forcer la solution à avoir une production groupée mais sans surcharger le coût total, nous proposons de générer les coûts d'activation tels que $\frac{Cout_i^{Act}}{Cout_i^{Prod}} \in [2\frac{N}{\#tr}, 3\frac{N}{\#tr}]$ où $\#tr$ est une approximation du nombre de transferts nécessaires, à savoir $\#tr = \frac{\mu(-1,M)}{C^T}$.

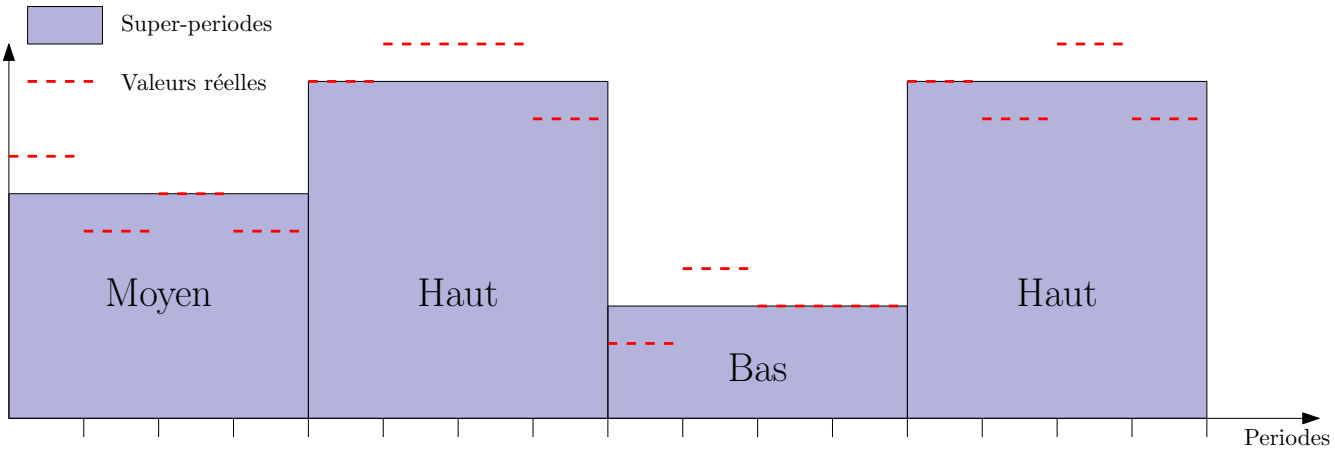

Figure 22.1 – Super-périodes et valeur réelle des périodes (super-périodes de taille 4). Schéma valable pour les coûts et pour les quantités de production.

Période et durée des jobs

Leurs valeurs ont peu d'impact tant qu'il y a suffisamment de périodes pour tous les jobs. Nous avons choisi de toujours normaliser les périodes (périodes toujours de longueur 1) et de générer les durées des jobs de sorte que la somme des durées de tous les jobs est égale au trois quarts de l'horizon de temps.

Prétraitement d'antichaînes

Un certain nombre de contraintes d'antichaînes sont ajoutées, par prétraitement, aux différents programmes linéaires. L'heuristique que nous avons choisie nous assure que tous les couples (i, j) apparaissent dans une contrainte d'antichaîne au moins. Pour ce faire, nous construisons $N \times M$ antichaînes en commençant par un couple (i, j) différent pour chacune. Puis, nous insérons le premier couple, dans l'ordre lexicographique, (i', j') incompatible avec tous les couples déjà présent dans l'antichaîne considérée. Nous itérons jusqu'à ce que l'antichaîne soit maximale et nous insérons la contrainte correspondante si l'antichaîne, ainsi créée, n'a pas déjà été inséré dans le modèle.

22.2 Expériences numériques

But : Nous avons utilisé les instances présentées dans la Table 22.1 pour effectuer des expériences numériques dans le but d'étudier le comportement du MILP (avec et sans contraintes renforcées), la version de Benders, l'approximation ILP et l'algorithme de type A^*.
Contexte technique : Les algorithmes ont été implémentés en C++17 sur un processeur Intel i5-9500 à 4,1 GHz. Les temps CPU sont en secondes.

Instances : Table 22.1 présente un package de 12 instances avec leurs caractéristiques les plus importantes générées comme présenté dans la Section 22.1. n est le nombre de périodes du *Feeder*, m est le nombre de jobs du *Eater*, p est la durée des périodes, $\#tr$ est l'approximation du nombre de transferts utilisée pour le calcul des coûts d'activation et de la valeur C^T, $\frac{C^E}{C^F}$ est le rapport des capacités des deux entrepôts et *nb_antichaines* est le nombre d'antichaînes générées en prétraitement et ajoutées au modèle avant sa résolution.

Table 22.1 – Tableau des paramètres d'instance

id	1	2	3	4	5	6	7	8	9	10	11	12
n	20	20	20	40	40	40	60	60	60	80	80	80
m	11	14	8	10	26	28	16	27	29	59	49	49
p	3	3	4	2	2	4	3	4	3	4	4	3
$\#tr$	4	5	5	7	10	10	10	15	15	14	20	20
C^T	14	49	12	12	34	142	34	69	42	136	87	87
$\frac{C^E}{C^F}$	2	2	0.5	2	2	0.5	2	2	0.5	2	2	0.5
nb_antichaines	211	260	140	390	992	1097	950	1585	1688	4672	3883	3885

□ Résultats liés aux deux résolutions des MILPs exacts

Nous résolvons les modèles exacts **(SLSS)** et **(SLSS Durci)** et calculons pour chacun d'eux :

Dans la Table 22.2 : La valeur objectif *obj*, la valeur de la relaxation linéaire au nœud racine *relax*, le nombre de coupes d'antichaine $\#AC$, les temps CPU (en secondes.) CPU, et le nombre de nœuds exploré par le Branch&Bound $\#nodes$, calculé par la résolution du MILP **(SLSS)**.

Dans la Table 22.3 : La valeur objectif *obj*, la valeur de la relaxation linéaire au nœud racine *relax*, le nombre de coupes d'antichaine $\#AC$, les temps CPU (en secondes.) CPU, et le nombre de nœuds exploré par le Branch&Bound $\#nodes$, calculé par la résolution du MILP **(SLSS Durci)**.

Table 22.2 – Résolution MILP exacte

id	1	2	3	4	5	6	7	8	9	10	11	12
obj	143	134	78.5	271.4	113.9	192.5	115.4	264.9	129	263.4	211	223.9
relax	67.4	80.7	64.1	64.2	63.2	173.7	49.1	132.3	76	173.2	116.6	173.5
#AC	3	2	0	5	9	2	8	3	11	8	3	0
CPU	4.03	8.89	0.14	49.46	185.31	10.24	496	1313	609	16699	35691	6684
#nodes	865	548	0	460	6010	273	17354	15090	8481	36330	346064	6245

Commentaire sur le tableau 22.2 :

Nous pouvons remarquer, sans surprise, que la résolution directe du problème **(SLSS)** prend beaucoup de temps, en particulier parce que la valeur de la relaxation linéaire est très éloignée de la valeur de la solution entière.

Ensuite, les instances 3, 6, 9, et 12, parce que la taille de l'entrepôt de *Feeder* est plus grande que celui de *Eater*, la résolution est simplifiée : la production est plus souvent concentrée est le problème de la production est donc plus simple. Pour ces instances et pour les mêmes raisons, nous remarquons également que la valeur de la relaxation linéaire est plus proche de la valeur de la solution entière.

Notons aussi que le nombre important d'antichaînes générées initialement permet de bien couvrir les besoins. Seule une poignée supplémentaire est nécessaire pour chaque instance.

Table 22.3 – Résolution MILP renforcée

id	1	2	3	4	5	6	7	8	9	10	11	12
obj	143	134	78.5	271.5	113.9	192.5	115.4	264.9	128.9	263.4	210.9	223.9
relax	114.3	107.2	68.4	71.2	75.1	174.8	49.1	132.3	79.2	179.9	116.6	173.5
CPU	1.64	1.92	0.37	74.5	127	9.69	217	265	172	2612	23584	4442
#nodes	193	275	0	1140	4817	110	11066	6015	3489	17203	231000	4096

Commentaire sur le tableau 22.3 :

Les contraintes de renforcements ajoutées au **(SLSS)** permettent d'améliorer la valeur de la relaxation linéaire et donc une réduction drastique du nombre de nœuds du *Branch&Bound*, réduisant ainsi le temps de résolution.

Malgré cette amélioration, la résolution reste difficilement appliquable à de grandes instances.

□ Résultats liés au MILP approché

Dans la Table 22.4, nous calculons : La valeur objectif *obj*, le nombre de coupes d'antichaine $\#AC$, les temps CPU (en secondes.) CPU, et le nombre de nuds explorés par le Branch&Bound $\#nodes$, calculés par la résolution du MILP **(SLSS Approx)**.

Table 22.4 – Résolution ILP approximative

id	1	2	3	4	5	6	7	8	9	10	11	12
obj	103.9	109.4	77	209.9	111.9	189	104.9	256.4	121	261.4	203.9	221
#AC	1	0	0	2	6	2	0	1	0	2	18	0
CPU	0.66	0.88	0.28	5.58	36.95	2.91	22.5	81.3	75.9	314	682	1131
#nodes	0	0	0	230	1680	80	620	2947	329	13816	23987	294

□ Résultats liés à la décomposition de Benders et à la résolution de l'algorithme A*

Nous testons enfin la décomposition de Benders **(SLSS Benders)** et l'algorithme A* qui sont toutes deux des méthodes exactes. Pour ces deux algorithmes, nous calculons :

Dans la Table 22.5 : La valeur objectif *obj*, le nombre de coupes d'antichaine $\#AC$, le nombre de coupes de Benders $\#Benders$, les temps CPU (en secondes.) CPU, et le nombre de nuds explorés par la Branche &Cut $\#nodes$, calculé par la résolution de l'ILP (SLSS Benders)

Dans la Table 22.6 : La valeur objective *obj*, la dernière borne inférieure trouvée LB, la longueur du plus court chemin (c'est-à-dire le nombre de transferts) $\#T$, les temps CPU (en secondes.) CPU, et le nombre de nœud exploré par l'algorithme A^* $\#nodes$

Table 22.5 – Résolution MILP de Benders

id	1	2	3	4	5	6	7	8	9	10	11	12
obj	143	134	78.5	271.4	113.9	192.5	115.4	264.9	129	263.4	211	223.9
#AC	0	3	0	15	22	1	7	9	18	25	4	8
#Benders	8	231	3	703	1312	24	48	2102	2046	6409	9971	197
CPU	3.62	4.08	0.36	66.1	192	10	215	650	293	2217	15634	1070
#nodes	530	1000	0	1247	4636	187	7545	11478	5103	59611	67664	2536

Commentaire sur le tableau 22.5 :

Du fait de l'absence d'une grande partie des contraintes, le nombre de nœuds a tendance à être plus important que pour le **(SLSS)** et le **(SLSS Durci)**, néanmoins, pour les mêmes raisons, le temps de calcul est beaucoup plus faible. La vérification n'étant pas systématique et le nombre de contraintes étant plus faible, en particulier pour les grandes instances, le **(SLSS Benders)** est le plus efficace des trois programmes linéaires. Il reste cependant lent (plus de quatre heures pour l'instance 11).

Table 22.6 – Résolution de l'algorithme A*

id	1	2	3	4	5	6	7	8	9	10	11	12
obj	143	134	78.5	271.5	114	192.5	115.5	265	129	263.4	211	224
#T	5	4	3	4	4	3	5	5	3	4	4	3
CPU	0.01	0.01	0	0.03	3.95	23	0.61	22.01	44.44	893.5	347.39	624.7
#nodes	550	713	76	980	8051	9178	2676	18178	9123	138046	48878	98577
nodes(%)	17.8	5.1	3.9	20.4	22.7	5.7	8.1	16.2	12.4	21.5	14.3	28.9

Commentaire sur le tableau 22.6 :

C'est la méthode exacte la plus rapide. Elle résout l'instance la plus difficile (instance 11) en cinq minutes contre presque 10 heures pour le **(SLSS)** malgré le nombre conséquent de nœud explorés.

Nous pouvons remarquer, à ce propos, la différence de temps de résolution entre l'instance 5 et 6 qui ont presque le même nombre de nœuds exploré, mais l'instance 6 est plus lente. Cette différence s'explique par le nombre conséquent de nœud à ajouter aux files à priorités. En effet, l'insertion dans la file triée prendre du temps, l'instance 6 possédant un plus grand nombre de nœuds explorable, il est normal qu'elle prenne plus de temps.

□ Résultats liés à la décomposition de Benders sans contraintes d'antichaînes ajoutées initialement.

Enfin, nous testons enfin la décomposition de Benders **(SLSS Benders)** sans qu'aucune contraintes d'antichaîne n'ait été calculé en prétraitement, nous calculons donc :

Dans la Table 22.7 : Le nombre de coupes d'antichaine $\#AC$, le nombre de coupes de Benders $\#Benders$, la différence avec le temps CPU de la résolution de Benders précédente (en secondes.) Δ_{CPU}, et le nombre de nuds explorés par la Branche &Cut $\#nodes$, calculé par la résolution de l'ILP (SLSS Benders)

TABLE 22.7 – Résolution MILP de Benders

id	1	2	3	4	5	6	7	8	9	10	11	12
#AC	108	482	10	1715	1969	6	27	995	1619	4998	779	106
#Benders	77	514	25	3994	4237	13	12	2580	3774	4687	12878	249
Δ_{CPU}	-0.4	+21	+0.2	+678	+742	-6	+817	+836	+1663	+33634	+20250	+673
#nodes	425	3242	0	54493	22226	79	33982	23059	28325	137282	270700	3944

Commentaire sur le tableau 22.7 :

Nous remarquons également que le nombre de coupes d'antichaînes ajoutées correspond à peu près au nombre d'antichaînes calculées en prétraitement. Cependant, l'augmentation considérable du nombre de nœuds explorés du fait de l'absence initiale de ces contraintes génère une forte augmentation du temps de résolution. La résolution avec la décomposition de Benders sans contrainte d'antichaîne initiale devient alors moins efficace que la résolution du **(SLSS)** avec contraintes d'antichaînes initiales.

Section 23

Conclusion

Dans ce chapitre, nous avons étudié un problème de synchronisation entre deux acteurs dont l'un produit une ressource nécessaire au bon fonctionnement du deuxième. Cette ressource doit être transférée régulièrement pour permettre au second acteur de mener à bien ses tâches sans perte de temps.

Nous avons proposé une formulation MILP en deux parties, nommées *Structurante* et *Non Structurante*, et proposé des contraintes supplémentaires sur les variables entières, que nous avons nommées contraintes *Renforcées*. Nous avons également montré comment séparer les contraintes d'antichaînes lors de la résolution du MILP

Nous avons ensuite proposé d'appliquer un schéma de décomposition de Benders en séparant les variables entières en réelles. Cette reformulation permet à la fois d'améliorer la valeur de la relaxation linéaire, mais également de réduire le nombre de nœuds du *Branch&Bound* et a permis d'accélérer considérablement la résolution.

Enfin, nous avons construit une méthode basée sur l'algorithme A* en montrant que la solution peut être complètement définie par les différents transferts et que ces derniers forment une chaîne dans l'ensemble des transferts possibles. La recherche d'une telle chaîne peut donc être réalisée par un algorithme de recherche de chemin dans un espace d'état approprié. Cette méthode s'est révélée être la plus efficace tout en gardant les garanties d'optimalités.

La partie *Scheduling* du problème (le problème du *Eater*) est cependant très simple puisque l'ordre des jobs est fixe. Elle peut cependant être vue comme un sous-problème d'un problème de planification d'une flotte de véhicules autonomes qui peut être étudié par la suite.

Section 24

Conclusion générale

Les travaux en recherche opérationnelle qui cherchent à améliorer le transport des biens et des personnes sont nombreux, mais ces mêmes problèmes abordés dans le contexte des véhicules autonomes sont récents et moins étudiés. Durant ma thèse, je me suis concentré sur l'étude de trois problèmes de mobilité intrinsèquement dépendant du temps dans lesquels ont été abordées les contraintes liées aux véhicules autonomes. Mon objectif était d'étudier les particularités des problèmes incluant ce type de véhicules, et d'utiliser leurs caractéristiques générales afin de proposer des algorithmes novateurs. Ainsi, cette thèse s'est concentré sur trois problèmes de logistiques distincts avec un point commun : des mécanismes de synchronisations ont été ajoutés pour répondre aux problématiques des véhicules autonomes. Cette étude n'a fait partie d'aucun projet industriel particulier et s'est donc concentré sur les aspects généraux des véhicules autonomes.

Dans le premier chapitre, nous avons étudié un problème de plus court chemin dans un environnement risqué. Ce risque provient des autres véhicules autonomes déjà présent sur place et de leur planning. Il est donc dépendant du temps et nous cherchons à ajouter un nouveau véhicule autonome pour répondre à une tâche non prise en charge par la flotte actuelle. Ce nouveau véhicule cherche un chemin jusqu'à sa destination en tenant compte du risque sur son trajet et en adaptant sa vitesse en conséquence. L'étude s'est décomposée en trois parties :

Tout d'abord, nous avons explicité ce qu'est le risque induit par la flotte et le risque pris par un nouveau véhicule en fonction de sa vitesse. De ce modèle, nous avons proposé une reformulation appelée *Risque versus Distance* de notre problème. Cette reformulation nous a permis de construire trois méthodes de calcul de la fonction vitesse sur un arc donné (*Risque versus Distance*, *Risque versus Temps* et *Distance versus Temps*).

Puis, nous avons proposé deux schémas algorithmiques pour rechercher la solution à notre problème : un chemin et sa fonction vitesse à tout instant qui minimise le temps de trajet en maintenant le risque en dessous d'un seuil. La première de ces deux méthodes cherche itérativement un chemin puis sa fonction vitesse. La recherche de chemin se base sur le calcul de la fonction vitesse pour essayer d'éviter les zones les plus à risque en utilisant un opérateur de voisinage. La deuxième méthode construit le chemin en même temps que la fonction vitesse à la manière d'un algorithme A*.

Cependant, pour répondre aux attentes de réactivités des algorithmes nécessaires dans le contexte des véhicules autonomes, nous avons d'abord proposé une méthode de filtrage des états par une évaluation adaptative en temps réel. Enfin, nous avons construit trois schémas d'amélioration d'une solution : la première est une recherche locale, la deuxième est une méthode d'exploration globale et la troisième est une descente de gradient.

Dans le deuxième chapitre, nous avons exposé une idée que nous proposons pour résoudre les problèmes intrinsèquement dépendant du temps : projeter le problème en supprimant la dimension temporelle, résoudre ce problème projeté et construire une solution du problème temporel dont la projection est exactement la solution du problème projeté. Nous avons appliqué cette idée à un problème de relocalisation

de marchandises sous la forme d'un problème à deux flots (un flot de véhicules et un flot de marchandises) dans lequel la préemption des marchandises est autorisée. La préemption des marchandises est le fait d'autoriser une marchandise à être déchargé par le véhicule qui la transporte afin d'être récupérée par un deuxième véhicule pour continuer son trajet. Cette contrainte de synchronisation est de plus en plus utilisée dans les entrepôts logistiques autonomes et semi-autonomes. L'étude s'est décomposée en trois parties :

Nous avons d'abord proposé un modèle de notre problème grâce à un réseau *time-expanded*. Cette modélisation nous servant de référence pour les méthodes à venir, nous avons choisi de discrétiser l'horizon de temps et d'expliciter le graphe lors de la résolution du modèle. Bien qu'assez simple, la taille du modèle très conséquente ne permet pas une résolution efficace, y compris pour des instances relativement petite.

Alors, nous avons proposé un modèle projeté dans laquelle la dimension temporelle a disparu. Malgré cela, nous avons proposé des contraintes faisant apparaître l'horizon de temps, notamment dans la contrainte de sous-tour que nous avons nommée *Extanded No Sub Tour*. L'idée de cette contrainte est que, pour parcourir des arcs liant un sous-ensemble de nœuds, il faut que suffisamment de véhicules entre dans le sous-ensemble correspondant. "Suffisamment" implique, non seulement, qu'au moins un véhicule doit entrer, mais surtout que si la somme des temps de trajets dépasse un certain nombre de fois l'horizon de temps (par exemple 3.2 fois l'horizon) autant de véhicules, plus un, seront nécessaires (donc 4 véhicules).

Pour terminer, nous avons construit trois méthodes pour construire une solution temporelle, soit dont la projection est exactement la solution du problème projeté (*Strong Lift*), soit dont le flot de marchandises se projette exactement sur le flot de marchandises de la solution projetée (*Weak Lift*). Nous nous sommes concentrés sur le *Weak Lift* qui possède toujours une solution. Nous avons proposé un modèle linéaire pour la résolution exacte ainsi que deux méthodes de résolution plus rapide. La première de ces deux méthodes se base sur le graphe du modèle linaire et y construit une solution en deux étapes : construction du flot de marchandises respectant la projection, puis construction des tournées connaissant une relation d'ordre et des fenêtres de temps sur les arcs porteurs de flots. La seconde méthode est une simulation dans laquelle les tournées sont construites avec le flot de marchandises.

Dans le troisième chapitre, nous avons travaillé sur un problème de synchronisation de deux acteurs, l'un produisant une ressource nécessaire au bon fonctionnement du deuxième. Périodiquement, un transfert de cette ressource doit avoir lieu pour maintenir l'activité du deuxième acteur. Ce problème peut être vu comme l'échange de batteries entre un véhicule autonome électrique et une station de production d'électricité qui va recharger la batterie inutilisée pendant que le véhicule réalise ses tâches. L'étude de ce problème s'est décomposée en trois parties :

Dans un premier temps, nous avons proposé un modèle MILP pour la résolution exacte du problème de synchronisation. Ce modèle est construit par un premier ensemble de variables entières et leurs contraintes associées que nous avons nommées *structurelles* et par un second ensemble de variables réelles et les contraintes liantes que nous avons nommées *non structurelles*. Nous avons également montré comment séparer les contraintes d'antichaînes qui sont en nombre exponentiel.

Ensuite, nous avons proposé de retirer les variables *non structurelles* du modèle car elles le rendent hétérogène et réduisent les performances de la résolution, notamment en diminuant la qualité de la relaxation linéaire. L'objectif est ici d'accélérer la résolution tout en gardant les garanties d'optimalités. Nous avons donc proposé de séparer les contraintes *non structurelles* à l'aide d'un schéma de décomposition de Benders. Pour accélérer la résolution de cette décomposition, nous avons proposé un ensemble de contraintes supplémentaires sur les variables *structurelles* que nous avons nommées contraintes de *renforcements*. Ces contraintes permettent d'améliorer la valeur de la relaxation linéaire et de réduire le nombre de nœuds du *Branch&Bound* grâce à des coupes valides.

Enfin, nous avons montré que notre problème pouvait être reformulé comme un problème de plus court chemin sous contrainte et nous avons proposé d'adapter un algorithme A* pour le résoudre exactement. Cet algorithme s'est montré être le plus efficace de tous et donne des résultats suffisamment rapides pour lui permettre d'être appliqué dans la gestion d'une flotte de véhicules autonomes.

Bibliographie

[Adulyasak et al., 2015] Adulyasak, Y., Cordeau, J. F., and Jans, R. (2015). The production routing problem : A review of formulations and solution algorithms. *Computers and Operations Research*, 55:141–152.

[Amaliah et al., 2022] Amaliah, B., Fatichah, C., and Suryani, E. (2022). A Supply Selection Method for better Feasible Solution of balanced transportation problem. *Expert Systems with Applications*, 203:117399.

[Amazon, 2022] Amazon (2022). https://www.amazon.com/primeair.

[Balas and Toth, 1983] Balas, E. and Toth, P. (1983). Branch and bound methods for the traveling salesman problem.

[Belaid, 2011] Belaid, F. (2011). Decision-making process for project portfolio management. *International Journal of Services Operations and Informatics*, 6(1-2):160–181. Publisher : Inderscience Publishers.

[Bellman, 1957] Bellman, R. (1957). Dynamic programming. *Science*, 153:34–37.

[Benders, 1962] Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252.

[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*, 15(1):1–31.

[Biel and Glock, 2016] Biel, K. and Glock, C. H. (2016). Systematic literature review of decision support models for energy-efficient production planning. *Computers and Industrial Engineering*, 101:243–259.

[Biggs et al., 1999] Biggs, N. L., Lloyd, E. K., and Wilson, R. J. (1999). Graph theory 1736-1936. *Oxford University Press*, page 240.

[Chen and Englund, 2016] Chen, L. and Englund, C. (2016). Cooperative Intersection Management : A Survey. *IEEE Transactions on Intelligent Transportation Systems*, 17(2):570–586.

[Clark et al., 2011] Clark, A., Almada-Lobo, B., and Almeder, C. (2011). Lot sizing and scheduling : Industrial extensions and research opportunities. *International Journal of Production Research*, 49(9):2457–2461.

[Colson et al., 2005] Colson, B., Marcotte, P., and Savard, G. (2005). Bilevel programming : A survey. *4or*, 3(2):87–107.

[Cordeau, 2006] Cordeau, J. F. (2006). A branch-and-cut algorithm for the dial-a-ride problem. *https://-doi.org/10.1287/opre.1060.0283*, 54:573–586.

[Courcelle et al., 2022] Courcelle, C., Baril, D., Pomerleau, F., and Laconte, J. (2022). On the importance of quantifying visibility for autonomous vehicles under extreme precipitation. *Towards Human-Vehicle Harmonization*, pages 239–249.

[Dantzig et al., 1954] Dantzig, G., Fulkerson, R., and Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2:393–410.

[Dantzig, 1947] Dantzig, G. B. (1947). Linear programming. *Operations Research*, 50:42–47.

[Desrosiers and Lübbecke, 2005] Desrosiers, J. and Lübbecke, M. E. (2005). A primer in column generation. *Column Generation*, pages 1–32.

[Doran et al., 1966] Doran, J., of the Royal Society of, D. M. P., and undefined 1966 (1966). Experiments with the graph traverser program. *royalsocietypublishing.orgJE Doran, D MichieProceedings of the Royal Society of London. Series A, 1966royalsocietypublishing.org*, 294:235–259.

[Drexl and Kimms, 1997] Drexl, A. and Kimms, A. (1997). Lot sizing and scheduling - Survey and extensions. *European Journal of Operational Research*, 99(2):221–235.

[Dumez, 2021] Dumez, D. (2021). Matheurisc approaches for solving transport optimization problems in urban logistics. *HAL*.

[Erdelic et al., 2019] Erdelic, T., Carić, T., and Lalla-Ruiz, E. (2019). A Survey on the Electric Vehicle Routing Problem : Variants and Solution Approaches. *Journal of Advanced Transportation*, 2019.

[Feo and Resende, 1989] Feo, T. A. and Resende, M. G. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71.

[Fischetti et al., 1994] Fischetti, M., Toth, P., and Vigo, D. (1994). A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graphs. *https://doi.org/10.1287/opre.42.5.846*, 42:846–859.

[Ford and Fulkerson, 1958] Ford, L. R. and Fulkerson, D. R. (1958). Constructing maximal dynamic flows from static flows. *Operations Research*, 6:419–433.

[Franceschetti et al., 2017] Franceschetti, A., Demir, E., Honhon, D., Van Woensel, T., Laporte, G., and Stobbe, M. (2017). A metaheuristic for the time-dependent pollution-routing problem. *European Journal of Operational Research*, 259(3):972–991.

[Gambardella et al., 1999] Gambardella, L., Taillard, E., and Agazzi, G. (1999). *MACS-VRPTW : A multiple ant colony system for vehicle routing problems with time windows*, volume 1, pages 63–76. McGraw-Hill Ltd., UKMcGraw -Hill House Shoppenhangers Road Maidenhead, BerkshireUnited Kingdom.

[Garey et al., 1976] Garey, M. R., Johnson, D. S., and Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1:117–129.

[Gaudioso and Paletta, 1992] Gaudioso, M. and Paletta, G. (1992). A heuristic for the periodic vehicle routing problem. *https://doi.org/10.1287/trsc.26.2.86*, 26:86–92.

[Gicquel et al., 2008] Gicquel, C., Minoux, M., Dallery, Y., Gicquel, C., Minoux, M., and Dallery, Y. (2008). Capacitated lot sizing models : a literature review. *HAL*.

[Glover, 1986] Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13:533–549.

[Goisque and Rapine, 2017] Goisque, G. and Rapine, C. (2017). An efficient algorithm for the 2-level capacitated lot-sizing problem with identical capacities at both levels. *European Journal of Operational Research*, 261(3):918–928.

[Goldberg, 1989] Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional.

[Hart et al., 1968] Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

[Hernández-Pérez and Salazar-González, 2009] Hernández-Pérez, H. and Salazar-González, J. J. (2009). The multi-commodity one-to-one pickup-and-delivery traveling salesman problem. *European Journal of Operational Research*, 196(3):987–995.

[Ho et al., 2018] Ho, S. C., Szeto, W. Y., Kuo, Y. H., Leung, J. M., Petering, M., and Tou, T. W. (2018). A survey of dial-a-ride problems : Literature review and recent developments. *Transportation Research Part B : Methodological*, 111:395–421.

[Holland, 1975] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems : An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. Complex Adaptive Systems.* The MIT Press.

[Huang et al., 2022] Huang, H., Li, Y., and Bai, Q. (2022). An improved a star algorithm for wheeled robots path planningwith jump points search and pruning method. *Complex Engineering Systems*, 2.

[Humagain et al., 2020] Humagain, S., Sinha, R., Lai, E., and Ranjitkar, P. (2020). A systematic review of route optimisation and pre-emption methods for emergency vehicles. *Transport Reviews*, 40(1):35–53.

[Hung and Chien, 2000] Hung, Y. F. and Chien, K. L. (2000). A multi-class multi-level capacitated lot sizing model. *Journal of the Operational Research Society*, 51(11):1309–1318.

[Irani and Pruhs, 2005] Irani, S. and Pruhs, K. R. (2005). Algorithmic problems in power management. *ACM SIGACT News*, 36(2):63–76.

[Jain and Bharti, 2023] Jain, S. and Bharti, K. K. (2023). A combinatorial optimization model for post-disaster emergency resource allocation using meta-heuristics. *Soft Computing*, 27:13595–13611.

[Janssens et al., 2015] Janssens, G. K., Soonpracha, K., Manisri, T., and Mungwattana, A. (2015). Robust Vehicle Routing Solutions to Manage Time Windows in the Case of Uncertain Travel Times. In *igi-global.comGK Janssens, K Soonpracha, T Manisri, A MungwattanaHandbook of Research on Artificial Intelligence Techniques and Algorithms, 2015igi-global.com*, pages 655–678. .

[Jarvis and Ratliff, 1982] Jarvis, J. J. and Ratliff, H. D. (1982). Notesome equivalent objectives for dynamic network flow problems. *Management Science*, 28:106–109.

[Jiang et al., 2020] Jiang, S., Song, Z., Weinstein, O., and Zhang, H. (2020). Faster dynamic matrix inverse for faster lps. *arxiv*.

[Kaminsky and Simchi-Levi, 2003] Kaminsky, P. and Simchi-Levi, D. (2003). Production and distribution lot sizing in a two stage supply Chain. *IIE Transactions (Institute of Industrial Engineers)*, 35(11):1065–1075.

[Karp, 1972] Karp, R. M. (1972). *Reducibility among Combinatorial Problems*, pages 85–103. Springer US.

[Kellerer et al., 2004] Kellerer, H., Pferschy, U., and Pisinger, D. (2004). Multidimensional Knapsack Problems. *Knapsack Problems*, pages 235–283.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220:671–680.

[Kleinert et al., 2021] Kleinert, T., Labbé, M., Ljubić, I., and Schmidt, M. (2021). A Survey on Mixed-Integer Programming Techniques in Bilevel Optimization. *EURO Journal on Computational Optimization*, 9.

[Koes et al., 2005] Koes, M., Nourbakhsh, I., and Sycara, K. (2005). Heterogeneous multirobot coordination with spatial and temporal constraints. *AAAI Workshop - Technical Report*, WS-05-06:9–16.

[Koopmans and Beckmann, 1957] Koopmans, T. C. and Beckmann, M. (1957). Assignment problems and the location of economic activities. *Econometrica*, 25:53.

[Krumke et al., 2014] Krumke, S. O., Quilliot, A., Wagler, A. K., and Wegener, J. T. (2014). Relocation in carsharing systems using flows in time-expanded networks. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8504 LNCS, pages 87–98.

[Labbé et al., 2020] Labbé, M., Plein, F., and Schmidt, M. (2020). Bookings in the European gas market : characterisation of feasibility and computational complexity results. *Optimization and Engineering*, 21(1):305–334.

[Land and Doig, 1960] Land, A. H. and Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica*, 28:497.

[Laporte and Nobert, 1987] Laporte, G. and Nobert, Y. (1987). Exact Algorithms for the Vehicle Routing Problem. *North-Holland Mathematics Studies*, 132(C):147–184.

[Le-Anh and De Koster, 2006] Le-Anh, T. and De Koster, M. B. (2006). A review of design and control of automated guided vehicle systems. *European Journal of Operational Research*, 171(1):1–23.

[Lera-Romero and Miranda-Bront, 2018] Lera-Romero, G. and Miranda-Bront, J. J. (2018). Integer programming formulations for the time-dependent elementary shortest path problem with resource constraints. *Electronic Notes in Discrete Mathematics*, 69:53–60.

[Li et al., 2013] Li, B., Springer, J., Bebis, G., and Hadi Gunes, M. (2013). A survey of network flow applications. *Journal of Network and Computer Applications*, 36(2):567–581.

[Lozano and Medaglia, 2013] Lozano, L. and Medaglia, A. L. (2013). On an exact method for the constrained shortest path problem. *Computers and Operations Research*, 40(1):378–384.

[Lu and Dessouky, 2006] Lu, Q. and Dessouky, M. M. (2006). A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows. *European Journal of Operational Research*, 175:672–687.

[Macrina et al., 2020] Macrina, G., Pugliese, L. D. P., and Guerriero, F. (2020). The Green-Vehicle Routing Problem : A Survey. *Modeling and Optimization in Green Logistics*, pages 1–26.

[Madduri et al., 2007] Madduri, K., Bader, D. A., Jonathan, W. B., and Crobak, J. R. (2007). An experimental study of a parallel shortest path algorithm for solving large-scale graph instances. *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments and the 4th Workshop on Analytic Algorithms and Combinatorics*, pages 23–35.

[Madsen et al., 1995] Madsen, O. B., Ravn, H. F., and Rygaard, J. M. (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.

[Malikopoulos et al., 2019] Malikopoulos, A. A., Hong, S., Park, B. B., Lee, J., and Ryu, S. (2019). Optimal Control for Speed Harmonization of Automated Vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 20(7):2405–2417.

[Martínez-Barberá and Herrero-Pérez, 2010] Martínez-Barberá, H. and Herrero-Pérez, D. (2010). Autonomous navigation of an automated guided vehicle in industrial environments. *Robotics and Computer-Integrated Manufacturing*, 26(4):296–311.

[McKinnon, 2011] McKinnon, A. C. (2011). Reducing energy consumption and emissions in the logistics sector. *Energy, Transport, & the Environment : Addressing the Sustainable Mobility Paradigm*, 9781447127178:521–537.

[Mombelli and Quilliot, 2023] Mombelli, A. and Quilliot, A. (2023). Synchronising lot sizing and job scheduling. In ., pages 647–654. ACM.

[Mombelli et al., 2022] Mombelli, A., Quilliot, A., and Baiou, M. (2022). Searching for a safe shortest path in a warehouse. In ., pages 115–122. SCITEPRESS - Science and Technology Publications.

[Mombelli et al., 2023a] Mombelli, A., Quilliot, A., and Baiou, M. (2023a). A comparison of several speed computation methods for the safe shortest path problem. In ., pages 15–26. SCITEPRESS - Science and Technology Publications.

[Mombelli et al., 2023b] Mombelli, A., Quilliot, A., and Baiou, M. (2023b). Managing flow problems defined on time-expanded networks through a project/lift decomposition. .

[Mombelli et al., 2024] Mombelli, A., Quilliot, A., and Baiou, M. (2024). *Multiple Heuristics with Reinforcement Learning to Solve the Safe Shortest Path Problem in a Warehouse*, volume 1985 CCIS, pages 3–25. Springer Science and Business Media Deutschland GmbH.

[Moon and Park, 2014] Moon, J. Y. and Park, J. (2014). Smart production scheduling with time-dependent and machine-dependent electricity cost by considering distributed energy resources and energy storage. *International Journal of Production Research*, 52(13):3922–3939.

[Park et al., 2009] Park, I., Jang, G. U., Park, S., and Lee, J. (2009). Time-dependent optimal routing in micro-scale emergency situation. In *Proceedings - IEEE International Conference on Mobile Data Management*, pages 714–719. IEEE.

[Pechmann and Schöler, 2011] Pechmann, A. and Schöler, I. (2011). Optimizing energy costs by intelligent production scheduling. *Glocalized Solutions for Sustainability in Manufacturing - Proceedings of the 18th CIRP International Conference on Life Cycle Engineering*, pages 293–298.

[Peiron, 1972] Peiron, B. (1972). Dimensionnement d'un parc de véhicules de transport opérant entre deux points. *Revue française d'automatique, informatique, recherche opérationnelle. Recherche opérationnelle*, 6(V2):21–32.

[Philippe et al., 2019] Philippe, C., Adouane, L., Tsourdos, A., Shin, H. S., and Thuilot, B. (2019). Probability collectives algorithm applied to decentralized intersection coordination for connected autonomous vehicles. In *IEEE Intelligent Vehicles Symposium, Proceedings*, volume 2019-June, pages 1928–1934. IEEE.

[Pimenta et al., 2017] Pimenta, V., Quilliot, A., Toussaint, H., and Vigo, D. (2017). Models and algorithms for reliability-oriented Dial-a-Ride with autonomous electric vehicles. *European Journal of Operational Research*, 257(2):601–613.

[Pourrahmani et al., 2015] Pourrahmani, E., Delavar, M. R., Pahlavani, P., and Mostafavi, M. A. (2015). Dynamic evacuation routing plan after an earthquake. *Natural Hazards Review*, 16:04015006.

[Ren et al., 2018] Ren, Q., Man, K. L., Lim, E. G., Lee, J., and Kim, K. K. (2018). Cooperation of multi robots for disaster rescue. In *Proceedings - International SoC Design Conference 2017, ISOCC 2017*, pages 133–134. IEEE.

[Renaud et al., 1996] Renaud, J., Laporte, G., and Boctor, F. F. (1996). A tabu search heuristic for the multi-depot vehicle routing problem. *Computers and Operations Research*, 3:229–235.

[Ryan et al., 2020] Ryan, C., Murphy, F., and Mullins, M. (2020). Spatial risk modelling of behavioural hotspots : Risk-aware path planning for autonomous vehicles. *Transportation Research Part A : Policy and Practice*, 134:152–163.

[Sargut and Romeijn, 2007] Sargut, F. Z. and Romeijn, H. E. (2007). Capacitated production and subcontracting in a serial supply chain. *IIE Transactions (Institute of Industrial Engineers)*, 39(11):1031–1043.

[Schrijver, 2002] Schrijver, A. (2002). On the history of the transportation and maximum flow problems. *Mathematical Programming*, 91(3):437–445.

[Schrijver, 2005] Schrijver, A. (2005). On the history of combinatorial optimization (till 1960). *Handbooks in Operations Research and Management Science*, 12:1–68.

[Sint and de Champeaux, 1977] Sint, L. and de Champeaux, D. (1977). An improved bidirectional heuristic search algorithm. *Journal of the ACM (JACM)*, 24:177–191.

[Song et al., 2008] Song, Y., Zhang, C., and Fang, Y. (2008). Multiple multidimensional knapsack problem and its applications in cognitive radio networks. *Proceedings - IEEE Military Communications Conference MILCOM*.

[Taillandier et al., 2017] Taillandier, F., Fernandez, C., and Ndiaye, A. (2017). Real Estate Property Maintenance Optimization Based on Multiobjective Multidimensional Knapsack Problem. *Computer-Aided Civil and Infrastructure Engineering*, 32(3):227–251.

[Taillard, 1993] Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285.

[Thomson and Moulin, 1990] Thomson, W. and Moulin, H. (1990). Axioms of Cooperative Decision Making. *Economica*, 57(228):543.

[Toussaint, 2010] Toussaint, H. (2010). *Algorithmique rapide pour les problèmes de tournées et d'ordonnancement*. PhD thesis, UCA.

[Turing, 1950] Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, LIX:433–460.

[Vignaux and Michalewicz, 1991] Vignaux, G. A. and Michalewicz, Z. (1991). A Genetic Algorithm for the Linear Transportation Problem. *IEEE Transactions on Systems, Man and Cybernetics*, 21(2):445–452.

[Vis, 2006] Vis, I. F. (2006). Survey of research in the design and control of automated guided vehicle systems. *European Journal of Operational Research*, 170(3):677–709.

[Vivaldini et al., 2013] Vivaldini, K. C. T., Tamashiro, G., Junior, J. M., and Becker, M. (2013). Communication infrastructure in the centralized management system for intelligent warehouses. In *Communications in Computer and Information Science*, volume 371, pages 127–136.

[Watkins, 1989] Watkins, C. (1989). *Learning from Delayed Rewards*. PhD thesis, Cambridge University.

[Woeginger, 2000] Woeginger, G. J. (2000). When Does a Dynamic Programming Formulation Guarantee the Existence of a Fully Polynomial Time Approximation Scheme (FPTAS) ? *INFORMS Journal on Computing*, 12(1):57–74.

[Wurman et al., 2008] Wurman, P. R., D'Andrea, R., and Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in warehouses. In *AI Magazine*, volume 29, pages 9–19.

[Zachariadis et al., 2009] Zachariadis, E. E., Tarantilis, C. D., and Kiranoudis, C. T. (2009). A guided tabu search for the vehicle routing problem with two-dimensional loading constraints. *European Journal of Operational Research*, 195:729–743.

[Zhang et al., 2009] Zhang, M., Batta, R., and Nagi, R. (2009). Modeling of workflow congestion and optimization of flow routing in a manufacturing/warehouse facility. *Management Science*, 55(2):267–280.

[Ziliaskopoulos and Mahmassani, 1993] Ziliaskopoulos, A. and Mahmassani, H. (1993). Time-dependent, shortest-path algorithm for real-time intelligent vehicle highway system applications. *Transportation Research Record*, 1408:94–100.