Quelques Algorithmes pour des problèmes de plus court chemin et d'opérations aériennes

Axel Parmentier

UNIVERSITÉ PARIS-EST

École doctorale MATHÉMATIQUES ET SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

Spécialité : Mathématiques

Présentée par

Axel PARMENTIER

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ PARIS-EST

ALGORITHMS FOR SHORTEST PATH AND AIRLINE PROBLEMS

QUELQUES ALGORITHMES POUR DES PROBLÈMES DE PLUS COURT CHEMIN ET D'OPÉRATIONS AÉRIENNES

Soutenance le 10 novembre 2016 devant le jury composé de :

M. Frédéric MEUNIER	*École des Ponts ParisTech*	Directeur de thèse
M. Stéphane GAUBERT	*INRIA*	Rapporteur
M. Patrick JAILLET	*MIT*	Rapporteur
Mme Marie-Christine COSTA	*ENSTA*	Examinatrice
M. Dominique FEILLET	*École des Mines de Saint-Etienne*	Examinateur
M. Frédéric GARDI	*Innovation 24*	Examinateur
M. Vincent LECLÈRE	*École des Ponts ParisTech*	Examinateur

À Marie,

Remerciements

Je tiens à remercier en premier lieu Frédéric Meunier, mon directeur de thèse, pour ces trois années passionnantes et pour son investissement quotidien en tant que directeur de thèse. Merci de m'avoir donné la passion de la recherche. Travailler avec toi est un vrai plaisir. Merci pour ton inaltérable enthousiasme, pour ton amitié et pour tous ces déjeuners ou la discussion court de Ravel à Miles Davis, en passant par l'Homme sans qualités, John von Neumann et Anthony Hopkins.

J'adresse mes sincères remerciements à Stéphane Gaubert et Patrick Jaillet pour avoir accepté d'être les rapporteurs de ma thèse. Je souhaite également remercier chaleureusement les examinateurs qui ont accepté de siéger dans mon jury malgré les emplois du temps chargés et la distance géographique : Marie-Christine Costa, Dominique Feillet, Frédéric Gardi et Vincent Leclère.

Je tiens à remercier les membres du département de Recherche Opérationnelle d'Air France, avec qui j'ai eu le plaisir de travailler tout au long de ma thèse. En particulier, je souhaite remercier Alexandre Boissy de m'avoir donné cette opportunité et Christophe Ressel pour avoir suivi ce projet. Je tiens aussi à remercier Mathieu Sanchez pour nos nombreuses discussions de modélisation, et Mohand Ait Alamara pour nos échanges autour de l'implémentation des algorithmes. Je souhaite aussi remercier Isabel Gomez, Thierry Vanhaverbeke, Pierre de Frémainville et Rémi Pacqueau pour tous les échanges que nous avons pu avoir sur la fin de la thèse. Enfin je souhaite aussi remercier tous ceux avec qui j'ai eu le plaisir de manger des chouquettes, et en particulier Solène, Blaise, Marine, Yousra, Alexandre, Waïl, Sybille, Ferran, Elsa, Magdalena, Stanislas, Anne Laure et tous ceux que j'oublie !

Un grand merci à tous les chercheurs du Cermics et de l'ENPC avec qui j'ai pu échanger pendant ces années. Je souhaite en particulier remercier Jean François Delmas pour ses conseils sur l'après thèse, Michel De Lara pour les références sur les ordres stochastiques, Jean-Philippe Chancelier sans qui les solveurs n'auraient jamais fonctionné, Guillaume Obozinski pour ses conseils sur tous les sujets liés à l'apprentissage automatique, Vincent Leclère, Julien Reygner et Aurélien Alfonsi pour leur aide sur les copules et la distance de Wasserstein. Je souhaite aussi remercier Laurent Monasse pour tous les footings passés discuter de maths et d'athlétisme. Un merci tout particulier à Isabelle Simunic, grâce à qui tous les projets aboutissent ! I want also to thank Marco Lübbecke and all his team for these exciting months in Aachen. Thank-you to Markus : working with you has been very pleasant. Je souhaite aussi à remercier les doctorants du Cermics, et en particulier Thomas, Pauline, Laurent, Étienne, Yannick et François.

Je tiens aussi à remercier Alexandre Bayen de m'avoir convaincu de faire une thèse, et Domi-

nique Bazy de m'avoir aidé dans mon orientation professionnelle.
Je souhaite remercier ma famille pour son indéfectible soutien. Merci Manet, pour ces années à Saint-Maur, ces dix ans de déjeuner du dimanche, et ces vingt cinq ans de petites attentions. Mes parents pour tout ce qu'ils m'ont transmis, et en particulier l'indépendance d'esprit, la curiosité, l'exigence envers soi-même et le goût du challenge. Maman, pour une vie à s'assurer de notre bonheur et de notre réussite. Papa, car on n'a jamais fini d'apprendre de son père. Merci Bonne-Maman. Merci Tanguy, c'est confortable d'être le second ! Merci Solène, Astrid et Servane, mes plus fidèles clientes en maths ! Merci Laure-Hélène et Vianney. Merci oncle Guy pour ces soirées derrière l'ordinateur de Castres à m'apprendre les rudiments du web. Merci Diane, tu es une marraine hors pair. Merci Arnauld, Isabelle, oncle Gérard et tante Anne-Marie pour vos tables ouvertes aux étudiants perdus en région parisienne. Merci Loïc, vous qui comprenez ce qu'est une passion pour la recherche, et merci Sylvie pour votre inépuisable bibliothèque. Merci Jacques et Paul, de prendre un matheux à bord, François, Delphine, Yann, Marie, oncle Jérôme et tante Isabelle. Merci Robert de m'avoir transmis les fondamentaux du guidon et de la fourchette, à défaut de l'art du tournevis. Merci à Jean-Léopold, Vianney, Pierre B., Jonathan, Thomas, Pascal et Cédric pour toutes ces heures passées à discuter de maths et d'avenir. Et merci à Zacharie, Cyprien, François-Xavier, Alexis, Bruno, Matthieu, Pierre G., Simon, Luc et Gaëtan.
Et enfin merci à Marie, ma femme, à qui je dédie ce manuscrit. Car, pour reprendre les mots d'Aragon,

« Que serais-je sans toi qui vins à ma rencontre,
Que serais-je sans toi qu'un cœur au bois dormant,
Que cette heure arrêtée au cadran de la montre,
Que serais-je sans toi que ce balbutiement »

Champs sur Marne, le 3 janvier 2017 A. P.

Abstract

This thesis develops algorithms for resource constrained shortest path problems, and uses them to solve the pricing subproblems of column generation approaches to some airline operations problems.

Resource constrained shortest path problems are usually solved using a smart enumeration of the non-dominated paths. Recent improvements of these enumeration algorithms rely on the use of bounds on path resources to discard partial solutions. The quality of the bounds determines the performance of the algorithm. Our main contribution to the topic is to introduce a procedure to generate bounds on paths resources in a general setting which covers most resource constrained shortest path problems, among which stochastic versions. In that purpose, we introduce a generalization of the resource constrained shortest path problem where the resources are taken in a lattice ordered monoid. The resource of a path is the monoid sum of the resources of its arcs. The problem consists in finding a path whose resource minimizes a non-decreasing cost function of the path resource among the paths that satisfy a given constraint. Enumeration algorithms are generalized to this framework. We use lattice theory to provide polynomial procedures to find good quality bounds. The efficiency of the approach is proved through an extensive numerical study on deterministic and stochastic path problems. Interestingly, the bounding procedures can be seen as generalizations to lattice ordered monoids of some algebraic path problem algorithms of the literature which work with path resources in a semiring.

These path algorithms have been successfully applied to the crew pairing problem. Given a set of flight legs operated by an airline, the aircraft routing and the crew pairing problem build respectively the sequences of flight legs operated by airplanes and crews that enable to cover all legs at minimum cost. As some sequences of flight legs can be operated by crews only if they stay in the same aircraft, the two problems are linked. The current practice in the industry is to solve first the aircraft routing, and then the crew pairing problem, leading to a non-optimal solution. During the last decade, solution schemes for the integrated problem have been developed. We propose a solution scheme for the integrated problem based on two new ingredients: a compact integer program approach to the aircraft routing problem, and a new algorithm for the pricing subproblem of the usual column generation approach to the crew pairing problem. This pricing algorithms is based on our resource constrained shortest path framework. We then generalize the algorithm to take into account delay propagation through probabilistic constraints. The algorithms enable to solve to near optimality Air France industrial instances.

Keywords: constrained shortest path problems, aircraft routing, crew pairing, optimization

under uncertainty, lattice ordered monoid, column generation.

Résumé

Cette thèse développe des algorithmes pour les problèmes de plus court chemin sous contraintes de ressources, et les applique à l'optimisation des rotations des avions et des équipages d'une compagnie aérienne dans le cadre d'approches par génération de colonnes.

Les problèmes de plus court chemin sous contraintes de ressources sont généralement résolus grâce à une énumération intelligente de tous les chemins non dominés. Les approches récentes utilisent des bornes sur les ressources des chemins pour éliminer des solutions partielles. L'efficacité de la méthode est conditionnée par la qualité des bornes utilisées. Notre principale contribution au domaine est l'introduction d'une procédure générique pour calculer des bornes qui s'applique à la plupart des problèmes de chemins sous contraintes, et en particulier les problèmes stochastiques. A cette fin, nous introduisons une généralisation du problème de plus court chemin sous contraintes dans laquelle les ressources des chemins appartiennent à un monoïde ordonné comme un treillis. La ressource d'un chemin est la somme des ressources de ses arcs, le terme somme désignant l'opérateur du monoïde. Le problème consiste à trouver parmi les chemins qui satisfont une contrainte donnée celui dont la ressource minimise une fonction de coût croissante de la ressource des chemins. Nous généralisons les algorithmes d'énumération à ce nouveau problème. La théorie des treillis nous permet de construire une procédure polynomiale pour trouver des bornes de qualité. L'efficacité pratique de la méthode est évaluée au travers d'une étude numérique détaillée sur des problèmes de chemins déterministes et stochastiques. Les procédures de calcul des bornes peuvent être interprétées comme des généralisations aux monoïdes ordonnés comme des treillis d'algorithmes de la littérature définis pour résoudre un problème de chemin pour lequel les ressources des chemins prennent leur valeur dans un semi-anneau.

Nos algorithmes de chemins ont été appliqués avec succès au problème de crew pairing. Étant donné un ensemble de vols opérés par une compagnie aérienne, les problèmes d'aircraft routing et de crew pairing construisent respectivement les séquences de vols opérées par les avions et par les équipages de manière à couvrir tous les vols à moindre coût. Comme certaines séquences de vols ne peuvent être réalisées par un équipage que s'il reste dans le même avion, les deux problèmes sont liés. La pratique actuelle dans l'industrie aéronautique est de résoudre tout d'abord le problème d'aircraft routing, puis le problème de crew pairing, ce qui aboutit à une solution non-optimale. Des méthodes de résolution pour le problème intégré ont été développées ces dix dernières années. Nous proposons une méthode de résolution pour le problème intégré reposant sur deux nouveaux ingrédients : un programme linéaire en nombre entier compact pour le problème d'aircraft routing, ainsi que de nouveaux pour le problème esclave de l'approche usuelle par génération de colonnes du problème de crew pairing. Ces algorithmes pour le problème esclave sont une application de nos algorithmes

pour le problème de plus court chemin sous contraintes. Nous généralisons ensuite cette approche de manière à prendre en compte des contraintes de probabilités sur la propagation du retard. Ces algorithmes permettent de résoudre quasiment à l'optimum les instances industrielles d'Air France.

Mots clefs : plus court chemin sous contraintes, aircraft routing, crew pairing, optimisation dans l'incertitude, monoïdes ordonnés comme des treillis, génération de colonnes.

Contents

List of Figures

List of Tables

1 Introduction

Two research lines are developed along this dissertation in Operations Research. An industrial line in partnership with Air France explores the company's air operations processes, and more precisely the design of the sequence of flight legs operated by aircraft and crews. We model mathematically the industrial constraints, and design algorithms to build efficient sequences of flight legs. By efficient sequences of flight legs, we mean sequences of flight legs of minimum cost among those which respect all the industrial constraints and working rules. We also give strategies to enforce the robustness of the sequences of flight legs with respect to delay. All these algorithms and strategies have been implemented in Air France industrial softwares.

The second line of research that structures this dissertation is theoretical, and focuses on path problems in graphs. A graph, as illustrated on Figure 1.1, is a collection of vertices linked by arcs. For instance, a road network can be considered as a graph whose arcs are the roads and whose vertices are the cross-roads. Given an origin vertex o and a destination vertex d, a path problem consists in finding a "good quality" path from o to d, which in addition satisfies some given constraint(s). An example of such problem is to find a minimum cost path between two cities while ensuring arrival before a given time. In this dissertation, we develop a versatile framework for path problems, and design standard algorithms to solve these problems.

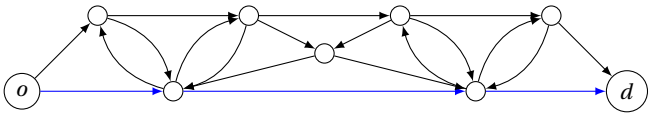


Figure 1.1 – A graph and an origin o to destination d path.

The link between the industrial part and the theoretical part of this dissertation is column generation, which is the mathematical technique we use to design the sequences of flight legs for Air France. Practically speaking, this approach can be divided into two modules. A first module generates sequences of flight legs that respect industrial constraints, and a second module assembles theses sequences of flight legs in such a way that each leg is covered by exactly one airplane and one crew. A sequence of flight legs can be seen as a path in the graph whose vertices are the flight legs and whose arcs are the connections between flight legs. As a consequence, the first module which builds feasible sequences of flight legs solves a

certain path problem. Therefore, the column generation method we have implemented in Air France industrial software relies on the framework and algorithms we have developed for path problems.

We illustrate this process on a graph of eight flight legs between three airports on Figure 1.2. Vertices of the graph correspond to flight legs and are indexed by v. Flight legs of vertices v_1 and v_5 (resp. v_3 and v_7) end in the same airport, and legs v_2 and v_6 (resp. v_4 and v_8) depart from this airport one hour later. We have to decide which connections will be in the sequences of flight legs. Dummy vertices o and d are added to select where crews can begin and end their flight legs sequences. The first module of the column generation builds feasible sequences of flight legs by searching paths from o to d in the graph. The second step selects which sequences of flight legs are operated so that each flight leg is covered by one sequence. On our example, the blue and the red sequences of flight legs are built in the first module, and selected in the second. An alternative choice would have been the sequences (v_1, v_6, v_7, v_4) and (v_5, v_2, v_3, v_8).

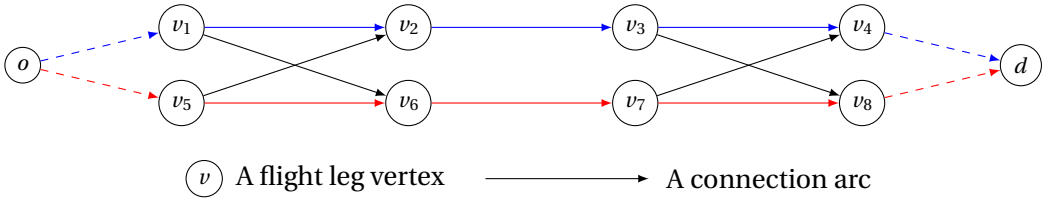


Figure 1.2 – Two pairings on an eight flight legs connection graph.

Section 1.1 details our industrial contributions on Air France processes. Section 1.2 sums up our mathematical contributions to path problems. And finally, Section 1.3 gives an introduction to column generation technique and details the link between the two parts of this dissertation.

As the contributions to path problems are required to design algorithms for Air France processes, this dissertation starts by the path problems. After a technical Chapter 2 which introduces probabilistic tools needed in this dissertation, Part I contains our work on path problems, and Part II our work on Air France industrial processes.

1.1 Mathematics for Air France operations

A sequence of four decisions must be taken to plan the operations of an airline. First, the flight legs operated by the company must be chosen. Then, given a specific flight leg operated by the company, the type of the aircraft that will cover it must be taken into account. For instance, we must decide if a given Paris – New-York will be covered by a Boeing 747 or an Airbus A380. Once the set of flight legs operated by a given fleet are chosen, it must be decided which airplane will cover which leg. For instance, we must choose which A380 will cover our Paris – New-York. Practically, this means building the sequences of flight legs or *rotations* that will be covered by each airplane. Finally, we must chose the rotations that will be operated by crews in such a way that each flight leg is covered by one crew.

In this dissertation, we focus on the two last decisions: the construction of the rotations operated by airplane and crews. Rotations must satisfy several industrial constraints. For instance, the time between two flight legs operated by the same aircraft must be sufficient to discharge and charge the luggages, board the passengers, and fill the kerosene tanks. Maintenance checks must be performed on each aircraft in specific airports every four days. For security reasons, a large number of working rules must be satisfied by crew sequences of flight legs. For instance, the number of flight legs operated by a crew in a day is limited, and there is a minimum rest time between the last flight leg of a day and the first of the following day. All these constraints must be satisfied by the rotations built.

Besides, the rotations that crews can operate depend on the rotations operated by airplanes. For instance, suppose that a leg ℓ_2 departs from CDG airport 45 minutes after the arrival of a leg ℓ_1. If ℓ_1 and ℓ_2 are operated by the same airplane, they can be operated by the same crew. Indeed, the crew can stay in the aircraft during the connection. On the contrary, if ℓ_1 and ℓ_2 are operated by different airplanes, a crew operating ℓ_1 and ℓ_2 would have to cross CDG airport to go from the first plane to the second, which is impossible in 45 minutes.

Presently, airplane rotations are built first, and then crews sequences of flight legs. As a consequence, crew rotations are constrained by the airplane rotations chosen. Building simultaneously airplane and crew rotations enables to reduces these constraints, and thus to build rotations with a better global quality. The main industrial part of my thesis consists in the design of algorithms to build simultaneously airplanes and crews rotations. The problem of constructing aircraft rotations is called the AIRCRAFT ROUTING PROBLEM, and the one of constructing crew rotations the CREW PAIRING PROBLEM. The problem of constructing both simultaneously is referred as the INTEGRATED PROBLEM.

Finally, if legs ℓ_1 and ℓ_2 are operated in a sequence by an airplane or by a crew, then if leg ℓ_1 is delayed, then this delay will propagate to leg ℓ_2. As a consequence, delay propagates along aircraft and crew rotations. We therefore develop a probabilistic model on delay to ensure that the risk of propagation of delay remains small.

1.2 Resource constrained and stochastic path problems

We start our discussion of path problems by introducing the STANDARD SHORTEST PATH PROBLEM. We suppose to have a cost c_a on each arc a of the digraph D. The cost of a path P is the sum of the cost of its arcs

$$c_P = \sum_{a \in P} c_a.$$

Given an origin vertex o and a destination vertex d, we call a path from o to d and o-d path. The objective of the STANDARD SHORTEST PATH PROBLEM is to find an o-d path P of minimum cost c_P. For instance, if the digraph D is a road network and c_a is the time needed to travel along arc a, then the solution P of the STANDARD SHORTEST PATH PROBLEM is the itinerary that enables to join d from o in minimum time c_P. Figure 1.3 illustrates a shortest o-d path of cost $1+3+2+2=10$.

The are two main classes of algorithms for the STANDARD SHORTEST PATH PROBLEM problem. The first ones are polynomial algorithms, such as Ford-Bellman dynamic programming

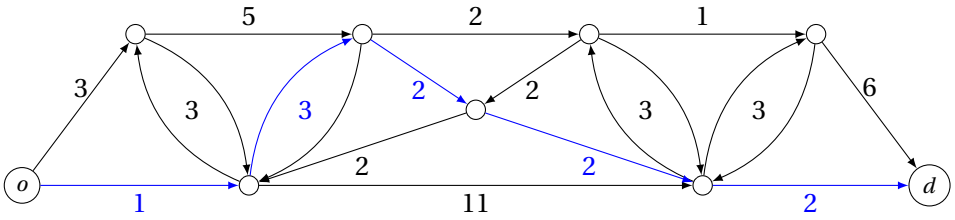


Figure 1.3 – A shortest o-d path of cost 10.

algorithm, or Dijkstra algorithm. The second one are enumeration algorithms, such as A*, which use bounds b_{vd} on v-d paths costs to discard paths in an enumeration of all the o-v paths. Computing the bounds b_{vd} can be long, but once this operation has been done, A* like algorithms are extremely efficient. As a consequence, if only one STANDARD SHORTEST PATH PROBLEM instance has to be solved, polynomial algorithms are generally faster, while if many instances have to be solved, enumeration algorithms become more interesting.

The path problem we have to solve to build crew rotations for Air France is to complicated to be modeled as a STANDARD SHORTEST PATH PROBLEM[1]. Indeed, to be feasible, a rotation must satisfy constraints such as

the number of flight legs in a day is non greater than 4,

and the cost of a rotation is a non-linear function of its properties. The number of legs in a rotation can thus be considered as a *resource* which is consumed along a path P. This resource is available in limited quantity and when it has been exhausted, the rotation becomes infeasible. Air France rotation building problem thus enters in the field of resource constrained shortest path problems.

In this dissertation, we define an algebraic framework for resource constrained shortest path problems, and we generalize both dynamic programming algorithms and A* like algorithms to this framework. This framework is light in the sense that we only require the algebraic properties that are necessary to generalize A* or dynamic programming algorithms. This lightness makes the framework extremely *versatile*, and enabled to use it on Air France complicated problems as well as on some stochastic resource constrained shortest path problem. As our framework's problem contains the $\mathcal{NP}$-complete STANDARD RESOURCE CONSTRAINED SHORTEST PATH PROBLEM, it cannot be solved in polynomial time[2]. Therefore, as dynamic programming algorithms are polynomial, they cannot solve our problem. But they provide a standard procedure to build "good quality" bounds that can be then used in an A* like algorithm. This standard procedure to compute bounds that can be then used in enumeration algorithm makes the strength of our framework. Indeed, if the idea of using bounds in resource constrained shortest path problems is not new, the previous resource constrained shortest path frameworks do not provide standard procedures to compute bounds. Practically, our

[1] To be more precise, using a dynamic programming principle, it could be modeled as a STANDARD SHORTEST PATH PROBLEM problem, but due to the curse of dimensionality, the size of this digraph would not be tractable.

[2] Unless the widely believed conjecture in computer science $\mathcal{P} \neq \mathcal{NP}$ turns out to be false.

bounds lead to order of magnitudes speed-up on the different types of stochastic problems we considered. Besides, our framework behaves especially well in the stochastic setting. To the best of our knowledge, our algorithms are the first that can deal with probabilistic constraints such as

a flight leg must be on time with at least 95% of the days.

which can give problems such as

find a path to the airport of minimum cost such that the probability of arriving on time is greater than 90%,

We now define our resource constrained shortest path framework. To make this definition more concrete, we illustrate it on the following toy problem on truck itineraries on continent wide truck deliveries. A full treatment of this toy problem is given in Section 3.4.2. Suppose that for a delivery, a truck must do a Paris-Beijing itinerary. The distance between the two cities being large, it is impossible to do it on a single day. Due to working rules, a driver cannot drive more than M hours a day. Besides, if a driver stops for a night in a city, he sleeps in a hotel, which has a given cost. Due to fuel consumption and tolls, driving between two cities also has a given cost. The objective of the problem is to find a Paris-Beijing path of minimum total cost and such that the daily driving time is non greater than M.

The first ingredients of our resource constrained shortest path framework are a *digraph* $D = (V, A)$, an *origin vertex* v and a *destination vertex* d.

In the case of our toy problem, the road network is modeled by the digraph D whose vertices $v \in V$ are the continent cities, and whose arcs a are the roads between these cities. Besides, for each city vertex v, there is a loop arc starting and ending in v corresponding to a night in v. Paris vertex is the origin vertex o and Beijing vertex is the destination d.

The second ingredient of our framework is a *resource set* $\mathscr{R}$, and a *resource* $q_a \in \mathscr{R}$ for each arc $a \in A$.

The resource q_a contains the information required on arc a. We therefore define our toy problem resource q_a to be the pair (c_a, d_a), where c_a is the cost of using arc a, and $d_a > 0$ is the driving duration of a. If a is a night arc, c_a is the cost of the hotel, and $d_a = 0$. Arcs a whose driving time is too long have resource $q_a = \infty$. Our resource set is therefore $\mathscr{R} = \mathbb{R}_+^2 \cup \{\infty\}$.

We now need to be able to define the resource of a path P as the sum of its arcs resources. The third ingredient of our framework is thus a *sum operator* $\oplus$ on $\mathscr{R}$. To be able to consider the empty path, we suppose that $\oplus$ admits a neutral element 0. The resource of a path P is defined as the sum of its arc resource

$$q_P = \bigoplus_{a \in P} q_a.$$

Our algorithms require to be able to compute the sum from the origin to the destination or from the destination to the origin. For the resource of a path to be uniquely defined, we therefore suppose that $\oplus$ is *associative*. We therefore suppose that $(\mathscr{R}, \oplus)$ is a monoid. Note that we do not suppose that $\oplus$ is commutative.

Our toy problem illustrates how having a non standard sum operator is useful. Indeed, suppose that we sum the resource $q_P = (c_p, d_p)$ with the resource $q_a = (c_a, d_a)$ of a daily arc a. It then suffices to use the component by component usual sum: $q_P \oplus q_a = (c_P + c_a, d_P + d_a)$. On the contrary, if arc a is a night arc, i.e. $d_a = 0$, then P reaches the end of a day, and there are two possibilities. Either $d_P \leq M$, which means that the driving time constraint is satisfied, and the time resource goes back to zero $q_P \oplus q_a = (c_P + c_a, 0)$, or $d_P > M$, and $q_P \oplus q_a = +\infty$. We have thus defined

$$(c,d) \oplus (\tilde{c},\tilde{d}) = \begin{cases} (c+\tilde{c}, d+\tilde{d}) & \text{if } \tilde{d} > 0, \\ (c+\tilde{c}, 0) & \text{if } \tilde{d} = 0 \text{ and } d \leq M, \\ \infty & \text{if } \tilde{d} = 0 \text{ and } d > M. \end{cases}$$

The definition of an associative operator $\oplus$ is actually a little bit more involved, and is dealt with in full details in Section 3.4.2.

Our algorithms rely on lower bounds on resources and on comparisons between resources. The next ingredient of our framework is therefore an *order* $\preccurlyeq$ on $\mathscr{R}$. We require this order to be *compatible with the sum* $\oplus$, which means that the left and right translations are monotone. This way, bounds on subpaths resources give bounds on paths resources. Besides, we suppose that each pair of resource q and $\tilde{q}$ admits a greatest lower bound denoted $q \wedge \tilde{q}$ and a least upper bound $q \vee \tilde{q}$. From an algebraic point of view, this means that $(\mathscr{R}, \preccurlyeq)$ is a *lattice*, and that $(\mathscr{R}, \oplus, \preccurlyeq)$ is a *lattice ordered monoid*.

For our toy problem, we use the product order: $q = (c,d) \preccurlyeq \tilde{q} = (\tilde{c},\tilde{d})$ if $c \leq \tilde{c}$ and $d \leq \tilde{d}$. Besides, the greatest lower bound of q and $\tilde{q}$ is $q \wedge \tilde{q} = \big(\min(c,\tilde{c}), \min(d,\tilde{d})\big)$.

Finally, the last ingredients of our framework are a *cost function* $c : \mathscr{R} \to \mathbb{R}$ and an *infeasibility function* $\rho : \mathscr{R} \to \{0,1\}$. The cost of a path P is $c(q_P)$, and the path is feasible if and only if $\rho(q_P) = 0$. Finally, for $\preccurlyeq$ to bring information, we suppose that *ρ and c are monotone with respect to $\preccurlyeq$.*

On our toy problem, given $q_P = (c_p, d_P)$, we naturally define $c(q_P) = c_P$ and $\rho(q_P) = 1$ if $d_P > M$. On our toy problem, we check that some constraints are satisfied in the sum operator by defining the result to be equal to $+\infty$. This is a modeling technique that is convenient on some very specific problems, but we generally do not need it.

We can now define our framework as the following MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM.

MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM
Let $(\mathscr{R}, \oplus, \preccurlyeq)$ be a lattice ordered monoid.
Input. A digraph $D = (V, A)$, two vertices $o, d \in V$, a collection $(q_a) \in \mathscr{R}^A$, and two oracles $c : \mathscr{R} \to \mathbb{R}$ and $\rho : \mathscr{R} \to \{0,1\}$.
Output. An o-d path P such that $\rho\left(\bigoplus_{a \in P} q_a\right) = 0$ and with minimum $c\left(\bigoplus_{a \in P} q_a\right)$.

Part I is devoted to the study of the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. It explains how to model a practical problem within this framework, and how to design algorithms for this problem. It also proves numerically the practical efficiency of these algorithms on some deterministic and stochastic path problems.

1.3 Column generation

Column generation is a mathematical programming technique that enables to solve efficiently a wide class of large integer problems. It is easily explained in the context of Air France CREW PAIRING PROBLEM. Let $\mathscr{L}$ be the set of flight legs operated by the company, and $\mathscr{P}$ be the set of feasible pairings. The CREW PAIRING PROBLEM can be expressed by the following integer program

$$\begin{array}{ll} \min & \displaystyle\sum_{p\in\mathscr{P}} c_p y_p \\ s.t. & \left| \begin{array}{ll} \displaystyle\sum_{p\ni\ell} y_p = 1, & \forall \ell \in \mathscr{L}, \\ y_p \in \{0,1\}, & \forall p \in \mathscr{P}, \end{array} \right. \end{array} \tag{1.1}$$

where c_p is the cost of operating pairing, and y_p is a binary variable equal to 1 if pairing p is selected in the solution. The size of $\mathscr{P}$ grows exponentially with the number of legs of the instance considered. Column generation is a technique to solve the linear relaxation of (1.1) on instances whose set of feasible pairings $\mathscr{P}$ is too large to be considered entirely. Column generation replaces the linear relaxation (1.2) of (1.1) by a restricted version (1.3), where only a restricted subset of pairing $\widetilde{\mathscr{P}} \subsetneq \mathscr{P}$ of tractable size $|\widetilde{\mathscr{P}}| \ll |\mathscr{P}|$ is considered.

$$\begin{array}{ll} \min & \displaystyle\sum_{p\in\mathscr{P}} c_p y_p \\ s.t. & \left| \begin{array}{ll} \displaystyle\sum_{p\ni\ell} y_p = 1, & \forall \ell \in \mathscr{L} \\ y_p \geq 0, & \forall p \in \mathscr{P} \end{array} \right. \end{array} \tag{1.2}$$

$$\begin{array}{ll} \min & \displaystyle\sum_{p\in\widetilde{\mathscr{P}}} c_p y_p \\ s.t. & \left| \begin{array}{ll} \displaystyle\sum_{p\ni\ell} y_p = 1, & \forall \ell \in \mathscr{L} \\ y_p \geq 0, & \forall p \in \widetilde{\mathscr{P}} \end{array} \right. \end{array} \tag{1.3}$$

Problem (1.2) is called the master problem. Column generation algorithm iteratively solves (1.3) on a reduced set of columns $\widetilde{\mathscr{P}} \subsetneq \mathscr{P}$ and updates $\widetilde{\mathscr{P}}$ with "interesting" columns in $\mathscr{P}\backslash\widetilde{\mathscr{P}}$. To explain which columns are interesting, we need to introduce the duals (1.4) and (1.5) of problems (1.2) and (1.3).

$$\begin{array}{ll} \max & \displaystyle\sum_{\ell\in\mathscr{L}} z_\ell \\ s.t. & \left| \begin{array}{ll} \displaystyle\sum_{\ell\in p} z_\ell \leq c_p, & \forall p \in \mathscr{P} \\ z_p \geq 0, & \forall \ell \in \mathscr{L} \end{array} \right. \end{array} \tag{1.4}$$

$$\begin{array}{ll} \max & \displaystyle\sum_{\ell\in\mathscr{L}} z_\ell \\ s.t. & \left| \begin{array}{ll} \displaystyle\sum_{\ell\in p} z_\ell \leq c_p, & \forall p \in \widetilde{\mathscr{P}} \\ z_p \geq 0, & \forall \ell \in \mathscr{L} \end{array} \right. \end{array} \tag{1.5}$$

Columns or variables y_p in the primal problem (1.3) correspond to rows or constraints in the dual (1.5). With the dual point of view, the "interesting" columns $p \in \mathscr{P}\backslash\widetilde{\mathscr{P}}$ that must be added appear naturally: those for which the constraint

$$\sum_{\ell\in p} z_\ell \leq c_p$$

is violated by the current solution z_ℓ of the current restricted dual (1.5). Given the current solution z_ℓ of the dual, we solve the following *pricing subproblem* to identify if the most violated constraint or prove that there are no violated constraints.

$$\min_{p\in\mathscr{P}} c_p - \sum_{\ell\in p} z_\ell. \tag{1.6}$$

The column generation procedure can be expressed as follows.

1. Initialize $\widetilde{\mathscr{P}}$ as a subset of $\mathscr{P}$ of tractable size.
2. Solve the reduced master problem (1.3).
3. Solve the pricing subproblem. If there exists a column p such that $c_p - \sum_{\ell \in p} z_\ell < 0$, then add it to $\widetilde{\mathscr{P}}$, and return to Step 2.

The following well-know theorem ensures the rightness of the method.

Theorem 1.1. *The algorithm ends after a finite number of a iterations, and at the end of the algorithm, y_p is an optimal solution of the relaxed master problem* (1.2).

The efficiency of the column generation approach relies on the ability to solve the pricing subproblem (1.6) efficiently. We now explain why the subproblem (1.6) is naturally modeled as a STANDARD RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. Let D be the digraph whose vertices are the legs $\ell \in \mathscr{L}$ and whose arcs are the feasible connections between two legs. As a pairing p is a sequence of flight legs, it can be considered as a path in D. Besides, the reduced costs z_ℓ are summed on the vertices along the path. As a pairing must respect crew working rules, a path P in D must respect several constraints to be a feasible pairing. Finding a pairing of minimum cost can thus be considered as solving a resource constrained shortest path problem in D.

Thanks to their use of new bounds, the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM algorithms we develop in Part I enable to solve the subproblem (1.6) more efficiently than previous algorithms, which enables to solve faster the CREW PAIRING PROBLEM by column generation, and thus to develop algorithm for the INTEGRATED PROBLEM.

1.4 Stochasticity

Due to the development of information systems, the amount of data available on the industrial processes has dramatically increased in the last decades. This amount of data opens the door to new opportunities in modeling risk and uncertainties in operations research. Specific attention has been paid to the modeling of risk and uncertainties in this dissertation. In the context of airline operations, risk is associated to the propagation of delay along sequences of flight legs. When a flight leg is late and connects to another flight leg, part of the delay is absorbed by the extra-time or slack between the arrival of the first flight leg and the departure of the second, and the other part is propagated. The objective is to allocate slack time between flight legs in a way to minimize the propagation of delay along sequences of flight legs.

There are two main approaches in the field of optimization under uncertainty. The robust approach considers a set of possible realizations of uncertainty but does not define a probability distribution on these realizations. The model is then optimized against the worst realization. On the contrary, the stochastic approach defines a probability distribution on the different realizations of uncertainty, and the optimization models typically deal with risk through probability functionals. In this dissertation, we focus on stochastic models. A key step in stochastic modeling is the choice of probability distributions. We need to be able to

compute probability functionals on these distributions. We have therefore two possibilities in the choice of these distributions: either we adopt simplifying assumptions such as random variables independence, and we can compute exactly probability functionals, or we keep a precise stochastic model, and we have to approximate probability functionals through a sampling approach.

Remark 1.1. The distinction between robust and stochastic optimization tends to become artificial, as some recent approaches to robust optimization consider a family of stochastic models, and optimize against the model in the family predicting the worst realizations. Such approaches are not considered in this dissertation.

There are also two ways of modeling the attitude toward risk in stochastic optimization. The first one is to define a cost functional that penalizes risky solutions. The second is to add to the problem probability constraints that forbid risk-prone solutions. For instance, in the context of airline operations problems, the probabilistic constraints correspond to the application of a marketing strategy where the airline guarantees to its customers that 90% of the flight legs are on-time. The second approach consists in minimizing the extra-costs due to delay.

In this dissertation, we develop algorithms for stochastic versions of the AIRCRAFT ROUTING PROBLEM, the CREW PAIRING PROBLEM, and the INTEGRATED PROBLEM, where delay is controlled through probabilistic constraints. To solve the stochastic version of the CREW PAIRING PROBLEM, we adapt the column generation scheme developed for the deterministic version. As delay propagates along sequences of flights, the delay of a flight is determined by the pairing which covers it. As a consequence, delay can be dealt with within the subproblem of the column generation. The solution scheme for the stochastic version of the CREW PAIRING PROBLEM is obtained from the deterministic one only by adapting the pricing subproblem. This new pricing subproblem is a stochastic path problem.

As we already mentioned, many stochastic path problems can be reduced to MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM instances, and are well solved by our MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM algorithms. The key observation behind these reductions is that several families of random variables distributions are lattices when endowed with the suitable order. For instance, the set of independent distributions with discrete and finite support is a lattice when endowed with the usual stochastic order $\leq_{\text{st}}$ defined by

$$\xi \leq_{\text{st}} \tilde{\xi} \quad \text{if} \quad \mathbb{P}(\xi \leq t) \geq \mathbb{P}(\tilde{\xi} \leq t) \quad \text{for all } t.$$

Besides, this order is compatible with the convolution product, and we thus obtain a lattice ordered monoid. Several other families of independent distributions are considered in Chapter 5. For each of these families, we provide a lattice order that it compatible with the convolution product and therefore lead to a lattice ordered monoid structure. Another important example is the one of scenario based random variables, where random variables ξ on a finite probability space Ω are considered. The relevant order is then

$$\xi \leqslant \tilde{\xi} \quad \text{if} \quad \xi(\omega) \leq \tilde{\xi}(\omega) \quad \text{for all scenarios } \omega \text{ in } \Omega.$$

Scenario based distributions are important because, using a sampling approach, any distribu-

tion can be approximated by a sampled scenario distribution.

For each stochastic problem considered in this dissertation, we provide algorithms that, first, can deal with stochasticity in the objective and with probabilistic constraints, and second, can deal with independent random variables and with scenario based random variables. Besides, we give upper bounds on the error committed when sampled distributions are used instead of the original ones. These upper bounds rely on modern probabilistic tools, such as concentrations inequalities that bound the Wasserstein distance between the original and the sampled versions of a random vector.

Chapter 2 introduces the probabilistic tools that are used in the remaining of this dissertation to model stochastic problems and to derive bounds on the error committed when using sampled distributions. Chapter 5 considers stochastic path problems. Finally Chapter 11 develops stochastic models on the propagation of delay in airline operations problems, and solution algorithms for the resulting stochastic versions of the AIRCRAFT ROUTING PROBLEM, the CREW PAIRING PROBLEM, and the INTEGRATED PROBLEM.

Introduction (Français)

Deux thématiques de recherche sont développées dans cette thèse. La première est industrielle et réalisée en partenariat avec Air France. Elle considère l'optimisation des séquences de vols réalisées par les avions et les équipages de la compagnie. Nous modélisons les contraintes industrielles s'exerçant sur ces séquences de vols et développons des algorithmes pour construire des séquences de vols de bonne qualité, c'est à dire des séquences de vols respectant toutes les contraintes industrielles et de coût minimal. Nous proposons aussi des méthodes mathématiques pour assurer la résilience des ces séquences vis -à-vis de la propagation du retard. Tous ces algorithmes et méthodes ont été implémentés au sein des logiciels d'Air France.

La seconde thématique de recherche qui structure cette thèse est plus théorique et considère des problèmes de chemin dans les graphes. Comme nous pouvons le voir sur la figure 1.1, un graphe est une collection de sommets reliés par des arcs. Les réseaux routiers peuvent par exemple être considérés comme des graphes dont les sommets correspondent aux carrefours et les arcs aux routes. Étant donné un sommet origine o et un sommet destination d,un problème de chemin consiste à trouver un chemin de « bonne qualité » entre o et d satisfaisant de plus certaines contraintes. Un exemple de tel problème consiste à trouver un itinéraire entre deux villes de coût minimal parmi les itinéraires permettant d'arriver avant une certaine heure. Dans cette thèse, nous développons un formalisme flexible pour traiter des problèmes de chemin, ainsi que des algorithmes génériques pour résoudre ces problèmes.

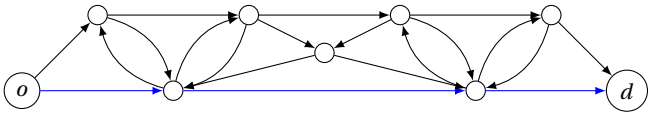


FIGURE 1.1 – Un graphe et un chemin de l'origine o à la destination d.

Le lien entre les deux thématiques de la thèse est la technique que nous utilisons pour construire les séquences de vols pour les équipages : la génération de colonnes. D'un point de vue pratique, cette technique peut être divisée en deux modules. Le premier module génère des séquences de vols respectant les contraintes industrielles, et le second assemble ces séquences de manière à ce que chaque vol soit couvert par exactement un avion et un équipage. Une séquence de vols peut être vue comme un chemin dans le graphe dont les sommets sont les vols et les arcs les correspondances ou connections entre deux vols. Le module qui construit des séquences de vols résout donc un problème de chemin. La génération de colonnes que nous avons implémentée dans les logiciels d'Air France s'appuie sur le formalisme

et les algorithmes que nous avons développés pour les problèmes de chemin.

Nous illustrons maintenant la construction des séquences de vols sur le graphe de la figure 1.2. Ce graphe contient huit vols entre trois aéroports. Les sommets du graphe correspondent à des vols et sont indexés par v. Les vols des sommets v_1 et v_5 (resp. v_3 et v_7) atterrissent dans le même aéroport, et les vols v_2 et v_6 (resp. v_4 et v_8) décollent de cet aéroport une heure plus tard. Nous devons choisir quelles connections seront dans les séquences de vols. Les sommets o et d permettent de choisir où les séquences de vols doivent commencer et finir. Le premier module de la génération de colonnes construit des séquences de vols réalisables en cherchant des chemins de o à d dans le graphe. Le deuxième module sélectionne les chemins qui seront réalisés de manière à ce que chaque vol soit couvert par exactement une séquence de vol. Dans notre exemple, les séquences bleues et rouges sont construites par le premier module et sélectionnées par le second. Un choix alternatif aurait été les séquences (v_1, v_6, v_7, v_4) et (v_5, v_2, v_3, v_8).

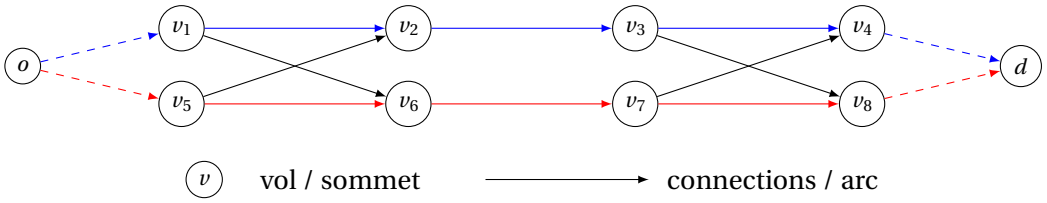


FIGURE 1.2 – Deux séquences de vols dans le graphe des connections.

Le paragraphe 1.1 détaille nos contributions à l'optimisation des processus d'Air France, et le paragraphe 1.2 celles sur les problèmes de chemin. Enfin, le paragraphe 1.3 présente la génération de colonnes.

Comme nos méthodes d'optimisation pour les problèmes d'opérations aériennes reposent sur nos algorithmes de chemin, la thèse commence par ces derniers. Suite au chapitre 2 qui introduit des outils probabilistes utilisés dans le reste de la thèse, la partie I présente nos travaux sur les problèmes de chemin et la partie II ceux sur les opérations aériennes.

1.1 Mathématiques pour les opérations aériennes

Quatre décisions successives doivent êtres prises pour planifier les opérations d'une compagnie aérienne. Tout d'abord, il faut choisir les vols qui seront réalisés par la compagnie. Puis il faut choisir le type d'avion qui réalisera chaque vol en fonction de la demande attendue : par exemple il faudra choisir si un Paris – New-York sera réalisé par un Boeing 747 ou un Airbus A380. Ces deux étapes passées, pour chaque sous-flotte d'avions identiques, il faut choisir quel avion réalisera quel vol. Par exemple, il faut choisir quel A380 réalisera notre Paris – New-York. En pratique, cela demande de construire les séquences de vols ou *rotations* réalisées par chaque avion. Enfin, nous devons construire les rotations réalisées par chaque équipage, de manière à ce que chaque vol soit réalisé par un équipage.

Dans cette thèse, nous nous concentrons sur les deux dernières étapes : la construction des rotations opérées par les avions et les équipages. Les rotations doivent satisfaire un certain

nombre de contraintes industrielles pour être réalisables. Par exemple, la durée entre deux vols réalisés successivement par un même avion doit être suffisante pour permettre de décharger et recharger les bagages, débarquer et rembarquer les passagers, et remplir les réservoirs de kérosène. Des opérations de maintenance doivent être réalisées au minimum tous les quatre jours sur chaque avion, et ces opérations ne peuvent être réalisées que dans certains aéroports possédant l'équipement adéquat. Pour des raisons de sécurité, un nombre important de règles sur les conditions de travail doivent être satisfaites par les rotations des équipages. Par exemple, le nombre de vol que peut réaliser un équipage sur une journée est limité, et une durée minimum de repos doit être assurée entre deux journées de travail. Toutes ces contraintes doivent être satisfaites par les rotations construites.

Par ailleurs, les rotations que les équipages peuvent effectuer dépendent des rotations réalisées par les avions. Par exemple, supposons qu'un vol ℓ_2 décolle de l'aéroport CDG 45 minutes après un vol ℓ_1. Alors ℓ_1 et ℓ_2 ne peuvent être réalisés par le même équipage que s'ils sont réalisés par le même avion. En effet, dans ce cas là, l'équipage peut rester dans l'avion. Si au contraire ils sont opérés par des avions différents, alors l'équipage devrait traverser l'aéroport CDG, ce qui est impossible en 45 minutes.

À l'heure actuelle, les rotations avions sont construites avant les rotations équipages. Les rotations que peuvent réaliser les équipages sont donc contraintes par les rotations avions choisies. Construire simultanément les rotations avions et équipages permet de réduire ces contraintes, et donc de construire des rotations de meilleure qualité. La contribution industrielle principale de cette thèse est un algorithme pour construire simultanément les rotations avions et équipages. Le problème de construction des rotations avions est appelé PROBLÈME D'AIRCRAFT ROUTING, et celui de construction des rotations équipages est appelé PROBLÈME DE CREW PAIRING. Le problème de construction simultanée est appelé PROBLÈME INTÉGRÉ.

Enfin, si deux vols ℓ_1 et ℓ_2 sont opérés successivement par le même avion ou le même équipage, alors si ℓ_1 est en retard, ce retard se propage à ℓ_2. Le retard se propage donc le long des rotations avions et équipages. Nous développons un modèle probabiliste du retard de manière à pouvoir garantir que le risque de propagation du retard reste faible.

1.2 Problèmes de plus court chemin sous contraintes

Pour commencer notre discussion des problèmes de chemin, nous introduisons le PROBLÈME DE PLUS COURT CHEMIN USUEL. Nous supposons avoir un coût c_a pour chaque arc a d'un graphe orienté D. Le coût d'un chemin P est la somme des coûts des ses arcs

$$c_P = \sum_{a \in P} c_a.$$

Étant donné un sommet origine o et un sommet destination d, nous appelons o-d chemin un chemin de o à d. L'objectif du PROBLÈME DE PLUS COURT CHEMIN USUEL est de trouver un o-d chemin P de coût minimum c_P. Par exemple, si le graphe D correspond à un réseau routier et c_a est le temps nécessaire pour parcourir l'arc a, alors la solution P du PROBLÈME DE PLUS COURT CHEMIN USUEL est l'itinéraire permettant de rejoindre d depuis o en un temps

minimum c_P. La figure 1.3 illustre un o-d chemin de coût $1+3+2+2=10$.

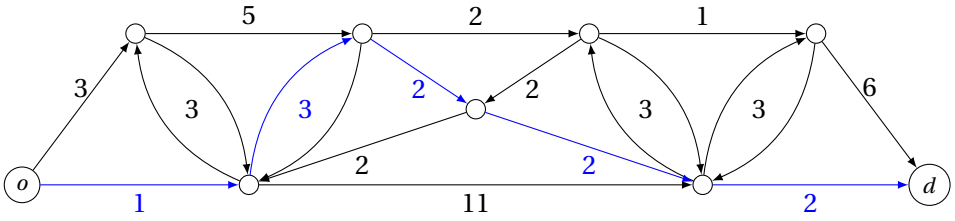


FIGURE 1.3 – Un o-d chemin de coût 10.

Les algorithmes les plus utilisés pour résoudre le PROBLÈME DE PLUS COURT CHEMIN USUEL peuvent être répartis en deux familles. Les premiers sont des algorithmes polynomiaux, comme l'algorithme de programmation dynamique de Ford-Bellman, ou l'algorithme de Dijkstra. Les seconds sont des algorithmes d'énumération, comme l'algorithme A*, et s'appuient sur des bornes b_{vd} sur le coût des v-d chemins pour éliminer des chemins dans une énumération de tous les o-v chemins. Le calcul des bornes b_{vd} peut être long, mais une fois cette étape réalisée, les algorithmes de type A* sont extrêmement efficaces. Par conséquent, si une seule instance du PROBLÈME DE PLUS COURT CHEMIN USUEL doit être résolue, les algorithmes polynomiaux sont généralement plus rapides, tandis que les algorithmes d'énumération deviennent intéressants lorsque de nombreuses instances doivent être résolues.

Le problème de chemin que nous résolvons pour construire les rotations équipages pour le problème d'Air France est trop complexe pour pouvoir être modélisé comme un PROBLÈME DE PLUS COURT CHEMIN USUEL[3]. En effet, pour être réalisables, une rotation doit respecter des contraintes du type

le nombre de vols réalisés dans une journée ne doit pas dépasser 4,

et le coût d'une rotation est une fonction non-linéaire de ses caractéristiques. Le nombre de vols dans une rotation peut donc être considéré comme une *ressource* qui est consommée le long d'un chemin P. Cette ressource est disponible en quantité limitée, et lorsqu'elle a été épuisée, le rotation devient non réalisable. La construction des rotations équipage peut donc être modélisée comme un problème de plus court chemin sous contraintes de ressources.

Dans cette thèse, nous introduisons un formalisme algébrique pour le problème de plus court chemin sous contraintes, et nous généralisons à la fois les algorithmes polynomiaux et les algorithmes d'énumération dans ce cadre. Ce formalisme est économique dans le sens où nous imposons uniquement les propriétés algébriques nécessaires à la généralisation des algorithmes mentionnés précédemment. Cette économie dans les hypothèses rend notre formalisme particulièrement flexible, ce qui a notamment permis de l'utiliser pour modéliser le problème d'Air France. Comme le problème que nous définissons dans notre formalisme

[3] Pour être plus précis, par une approche de type programmation dynamique, il pourrait être modélisé comme un PROBLÈME DE PLUS COURT CHEMIN USUEL, mais du fait de la « malédiction de la dimension », la taille du graphe obtenu serait trop grand pour être tractable.

contient le problème de plus court chemin sous contraintes usuel, et que ce dernier est $\mathcal{NP}$-complet, nous ne pouvons espérer le résoudre en temps polynomial[4]. Nous ne pouvons donc pas utiliser les algorithmes polynomiaux comme les algorithmes de Ford-Bellman et de Dijkstra pour résoudre ce problème. Mais nous verrons que leur généralisation permet de construire une procédure générique pour construire des bornes inférieures de bonne qualité, qui peuvent ensuite être utilisées dans un algorithme d'énumération. Cette procédure générique pour construire des bornes pour les algorithmes d'énumération fait la force de notre formalisme. En effet, si l'idée d'utiliser des bornes dans les algorithmes de plus court chemin sous contraintes n'est pas nouvelle, les formalismes de plus court chemin sous contraintes de la littérature ne disposent pas d'une telle procédure. En pratique, l'accélération des algorithmes obtenue par l'usage de bornes peut atteindre plusieurs ordres de grandeur selon le type de problème considéré. Par ailleurs, notre formalisme permet de résoudre efficacement les problèmes de chemin stochastiques. A notre connaissance, ces algorithmes sont les premiers qui permettent de prendre en compte des contraintes probabilistes dans les problèmes de chemin. Par exemple, nous pouvons prendre en compte des contraintes comme

un vol doit arriver à l'heure au moins 95% des fois,

ou résoudre des problèmes du type

trouver un chemin de l'hôtel a l'aéroport de coût minimum et qui garantisse d'arriver à l'heure avec une probabilité d'au moins 90%.

Nous introduisons maintenant notre formalisme de plus court chemin sous contraintes. Pour rendre cette définition plus concrète, nous l'illustrons sur un problème de livraisons par camion sur très longue distance. Un traitement détaillé de ce problème est disponible dans la section 3.4.2. Supposons que pour une livraison, un camion doive se rendre de Paris à Pékin. La distance entre les deux pays étant longue, il est impossible de la parcourir en une seule journée. Par ailleurs, pour respecter la réglementation routière, un conducteur ne peut pas conduire plus de M heures par jour. De plus, si un conducteur s'arrête pour la nuit dans une ville, il dort à l'hôtel, ce qui entraîne un coût. Enfin, en raison du carburant et des péages, conduire entre deux villes a aussi un coût. L'objectif du problème est de trouver un itinéraire de Paris à Pékin de coût complet minimal et tel que la durée quotidienne de conduite n'excède pas M.

Les premiers ingrédients de notre formalisme de plus court chemin sous contraintes sont un *graphe orienté* $D = (V, A)$, un *sommet origine* o et un *sommet destination* d.

Dans le cas de notre problème de livraison, le réseau routier est modélisé par un graphe orienté D dont les sommets v sont les villes du continent et les arcs a les routes entre ces villes. De plus, pour chaque sommet v, il existe dans A une boucle dont l'origine et la destination sont v. Paris correspond au sommet origine o et Pékin au sommet destination d.

Les seconds ingrédients de notre formalisme sont un *ensemble des ressources* $\mathcal{R}$, et une ressource q_a pour chaque arc a dans A.

[4] À moins que la conjecture $\mathcal{P} \neq \mathcal{NP}$ ne se révèle fausse.

La ressource q_a contient l'information nécessaire sur l'arc a. Nous définissons donc les ressources de notre problème de livraison comme les couples (c_a, d_a) où c_a est le coût encouru lorsque l'arc a est traversé, et d_a est le temps de conduite nécessaire pour traverser a. Si a est une boucle commençant et se terminant en v, alors c_a est le coût lié à une nuit d'hôtel en v, et $d_a = 0$. Les arcs a dont la durée de conduite est strictement supérieure à M ont un coût c_a égal à $+\infty$. Notre ensemble des ressources $\mathscr{R}$ est donc $\mathbb{R}_+^2 \cup \{+\infty\}$.

Nous souhaitons maintenant pouvoir définir la ressource d'un chemin P comme la somme des ressources de ses arcs. Le troisième ingrédient de notre formalisme est donc un *opérateur de somme* $\oplus$ sur $\mathscr{R}$. De manière à pouvoir prendre en compte le chemin vide, nous faisons l'hypothèse que $\oplus$ admet un élément neutre 0. La ressource d'un chemin est définie comme la somme des ressources de ses arcs

$$q_P = \bigoplus_{a \in P} q_a.$$

Par ailleurs, pour que nos algorithmes fonctionnent, nous devons pouvoir sommer les ressources d'un chemin de l'origine à la destination et de la destination à l'origine. Pour que ces quantités correspondent, nous faisons l'hypothèse que l'opérateur $\oplus$ est *associatif*. Nous faisons donc l'hypothèse que $(\mathscr{R}, \oplus)$ est un monoïde. Nous notons que nous ne supposons pas que $\oplus$ est commutatif.

Nous pouvons maintenant illustrer sur notre problème de livraison pourquoi un opérateur de somme $\oplus$ non standard est utile. En effet, supposons que nous sommions la ressource $q_P = (c_P, d_P)$ d'un chemin P avec la ressource $q_a = (c_a, d_a)$ d'un arc a. Si a est un arc de jour entre deux sommets, il suffit de sommer les composantes terme à terme : $q_P \oplus q_a = (c_P + c_a, d_P + d_a)$. Si au contraire a correspond à une boucle de nuit, c'est à dire si $d_a = 0$, alors P atteint la fin d'une journée, et il y a deux possibilités. Soit $q_P \leq M$, et alors la contrainte sur le temps de conduite est satisfaite, et la ressource temporelle retourne à zéro $q_P + q_a = (c_P + c_a, 0)$, soit $q_P + q_a = +\infty$. Nous avons donc défini

$$(c,d) \oplus (\tilde{c}, \tilde{d}) = \begin{cases} (c + \tilde{c}, d + \tilde{d}) & \text{si } \tilde{d} > 0, \\ (c + \tilde{c}, 0) & \text{si } \tilde{d} = 0 \text{ et } d \leq M, \\ +\infty & \text{si } \tilde{d} = 0 \text{ et } d > M. \end{cases}$$

Définir $\oplus$ de manière à ce qu'il soit associatif est en pratique un peu plus technique, et sera fait de manière détaillée dans la section 3.4.2.

Nos algorithmes s'appuient sur des bornes inférieures et sur des comparaisons entre les ressources. L'ingrédient suivant de notre formalisme est donc un *ordre partiel* $\preccurlyeq$ sur $\mathscr{R}$. Par ailleurs, nous imposons que cet ordre soit compatible avec l'opérateur $\oplus$, ce qui veut dire que les translations à droite et à gauche sont monotones. Ainsi, les bornes sur des sous-chemins permettent de construire des bornes sur les chemins. Par ailleurs, nous faisons l'hypothèse que chaque couple $(q, \tilde{q})$ de ressources admet une plus grande borne inférieure notée $q \wedge \tilde{q}$, et une plus petite borne supérieure notée $q \vee \tilde{q}$. D'un point de vue algébrique, cela veut dire que $(\mathscr{R}, \preccurlyeq)$ est un *treillis*, et que $(\mathscr{R}, \oplus, \preccurlyeq)$ est un *monoïde ordonné comme un treillis*.

Pour notre problème de livraison, nous utilisons l'ordre produit : $q = (c,d) \preccurlyeq \tilde{q} = (\tilde{c}, \tilde{d})$ si $c \leq \tilde{c}$ et $d \leq \tilde{d}$. La plus grande borne inférieure est alors égale à $q \wedge \tilde{q} = \big(\min(c, \tilde{c}), \min(d, \tilde{d})\big)$.

Enfin, les derniers ingrédients de notre formalisme sont une *fonction de coût* $c : \mathscr{R} \to \mathbb{R}$ et une *fonction d'infaisabilité* $\rho : \mathscr{R} \to \{0,1\}$. Le coût d'un chemin P est $c(q_P)$, et P est un chemin réalisable si et seulement si $\rho(q_P) = 0$. Enfin, pour que l'information fournie par $\leqslant$ soit exploitable, nous supposons que *c et ρ sont monotones par rapport à $\leqslant$*.

Concernant notre problème de livraison, étant donnée $q_P = (c_P, d_P)$, nous définissons naturellement $c(q_P) = c_P$ et $\rho(q_P) = 1$ si et seulement si $d_P > M$. Nous remarquons que sur notre problème de livraison, nous vérifions si certaines contraintes sont satisfaites au niveau de l'opérateur somme, en définissant le résultat de la somme comme $+\infty$ lorsqu'elles ne le sont pas. Cette technique de modélisation est pratique sur certains problèmes particuliers, mais n'est en général pas nécessaire.

Nous pouvons maintenant définir notre formalisme comme le PROBLÈME DE PLUS COURT CHEMIN AVEC RESSOURCES DANS UN MONOÏDE suivant.

PROBLÈME DE PLUS COURT CHEMIN AVEC RESSOURCES DANS UN MONOÏDE
Entrée. Un graphe orienté $D = (V, A)$, deux sommets $o, d \in V$, une collection $(q_a) \in \mathscr{R}^A$, et deux oracles $c : \mathscr{R} \to \mathbb{R}$ et $\rho : \mathscr{R} \to \{0,1\}$.
Sortie. Un o-d chemin P tel que $\rho\left(\bigoplus_{a \in P} q_a\right) = 0$ et minimisant $c\left(\bigoplus_{a \in P} q_a\right)$.

La partie I se concentre sur l'étude du PROBLÈME DE PLUS COURT CHEMIN AVEC RESSOURCES DANS UN MONOÏDE. Nous expliquons notamment comment modéliser une problème pratique dans ce formalisme, et fournissons des algorithmes de résolution. Nous testons aussi l'efficacité de ces algorithmes sur des problèmes de chemin déterministes et stochastiques.

1.3 Génération de colonnes

La génération de colonnes est une technique de programmation mathématique qui permet de résoudre une large classe de problèmes en nombres entiers de grande taille. Nous pouvons l'expliquer facilement sur l'exemple du problème des rotations équipages. Soit $\mathscr{L}$ l'ensemble des vols opérés par une compagnie aérienne, et $\mathscr{P}$ l'ensemble des rotations réalisables. Le problème des rotations équipages peut être exprimé comme le programme linéaire en nombres entiers suivant :

$$\begin{array}{ll} \min & \displaystyle\sum_{p \in \mathscr{P}} c_p y_p \\ s.c. & \left| \begin{array}{ll} \displaystyle\sum_{p \ni \ell} y_p = 1, & \forall \ell \in \mathscr{L}, \\ y_p \in \{0,1\}, & \forall p \in \mathscr{P}, \end{array} \right. \end{array} \tag{1.7}$$

où c_P est le coût lié à la réalisation de la rotation p et y_P est une variable binaire égale à 1 si la rotation p est sélectionnée dans la solution. La taille de $\mathscr{P}$ croit de manière exponentielle avec le nombre de vols dans l'instance considérée. La génération de colonnes est une méthode pour résoudre le relâché linéaire de (1.7) sur les instances pour lesquelles l'ensemble $\mathscr{P}$ est trop grand pour pouvoir être considéré dans son intégralité. La génération de colonnes remplace le relaché linéaire (1.8) de (1.7) par sa version restreinte (1.9) où seules les rotations p dans un sous ensemble $\widetilde{\mathscr{P}} \subsetneq \mathscr{P}$ de taille tractable $|\widetilde{\mathscr{P}}| \ll |\mathscr{P}|$ sont considérées.

$$\min \sum_{p\in\mathscr{P}} c_p y_p \qquad \text{s.c.} \begin{cases} \sum_{p\ni\ell} y_p = 1, & \forall \ell \in \mathscr{L} \\ y_p \geq 0, & \forall p \in \mathscr{P} \end{cases} \tag{1.8}$$

$$\min \sum_{p\in\widetilde{\mathscr{P}}} c_p y_p \qquad \text{s.c.} \begin{cases} \sum_{p\ni\ell} y_p = 1, & \forall \ell \in \mathscr{L} \\ y_p \geq 0, & \forall p \in \widetilde{\mathscr{P}} \end{cases} \tag{1.9}$$

Le problème (1.8) est appelé problème maître. La génération de colonnes résout de manière récursive (1.9) sur une ensemble réduit de colonnes $\widetilde{\mathscr{P}} \subsetneq \mathscr{P}$, et met à jour $\widetilde{\mathscr{P}}$ à chaque étape avec des colonnes « intéressantes » dans $\mathscr{P}\backslash\widetilde{\mathscr{P}}$. Pour expliquer quelles colonnes sont intéressantes, nous devons introduire les duaux (1.10) et (1.11) de (1.8) et (1.9).

$$\max \sum_{\ell\in\mathscr{L}} z_\ell \qquad \text{s.c.} \begin{cases} \sum_{\ell\in p} z_\ell \leq c_p, & \forall p \in \mathscr{P} \\ z_\ell \geq 0, & \forall \ell \in \mathscr{L} \end{cases} \tag{1.10}$$

$$\max \sum_{\ell\in\mathscr{L}} z_\ell \qquad \text{s.c.} \begin{cases} \sum_{\ell\in p} z_\ell \leq c_p, & \forall p \in \widetilde{\mathscr{P}} \\ z_\ell \geq 0, & \forall \ell \in \mathscr{L} \end{cases} \tag{1.11}$$

Les variables y_p dans le problème primal (1.8) correspondent aux contraintes dans (1.10). Dans le dual, les rotations p intéressantes de $\mathscr{P}\backslash\widetilde{\mathscr{P}}$ qui doivent être ajoutées apparaissent naturellement : ce sont celles pour lesquelles la contrainte

$$\sum_{\ell\in p} z_\ell \leq c_p$$

n'est pas satisfaite par la solution optimale z_ℓ du problème restreint courant (1.11). Étant donné la solution courante du dual z_ℓ, le problème esclave de la génération de colonnes consiste à trouver la contrainte la plus violée ou à prouver que toutes les contraintes sont satisfaites par la solution courante. Le problème esclave peut donc être écrit comme le problème d'optimisation

$$\min_{p\in\mathscr{P}} c_p - \sum_{\ell\in p} z_\ell. \tag{1.12}$$

La procédure de la génération de colonnes peut donc s'exprimer de la manière suivante.

1. Initialiser $\widetilde{\mathscr{P}}$ comme un sous ensemble de $\mathscr{P}$ de taille raisonnable.
2. Résoudre le problème maître restreint (1.9).
3. Résoudre le problème esclave. S'il existe une colonne p telle que $c_P - \sum_{\ell\in p} z_\ell < 0$, alors ajouter cette colonne à $\widetilde{\mathscr{P}}$, et retourner à l'étape 2.

Le théorème bien connu que nous introduisons maintenant assure la validité de la méthode.

Théorème 1.1. *L'algorithme se termine après un nombre fini d'opérations, et à la fin de celui-ci, y_p est une solution optimale du problème maître* (1.8).

L'efficacité de l'approche par génération de colonnes dépend de notre capacité à résoudre efficacement le problème esclave. Nous expliquons maintenant pourquoi le problème esclave (1.12) se modélise naturellement comme un PROBLÈME DE PLUS COURT CHEMIN AVEC RESSOURCES DANS UN MONOÏDE. Soit D le graphe orienté dont les sommets sont les vols $\ell \in \mathscr{L}$ et les arcs sont les connections réalisables entre deux vols. Comme une rotation p est une

séquence de vols, elle peut-être considérée comme un chemin P dans D. De plus, les coûts réduits z_ℓ sont sommés sur les sommets le long du chemin. Comme une rotation équipage doit satisfaire certaines règles sur le travail, un chemin P dans D doit satisfaire plusieurs contraintes pour être une rotation réalisable. Trouver une rotation de coût minimum revient donc à résoudre un problème de plus court chemin sous contraintes dans D.

Grâce à leur capacité à générer et utiliser des bornes, les algorithmes pour le PROBLÈME DE PLUS COURT CHEMIN AVEC RESSOURCES DANS UN MONOÏDE que nous développons dans la partie I permettent de résoudre plus efficacement le problème esclave (1.12), ce qui permet de résoudre plus rapidement le problème de construction des rotations équipages, et donc de proposer des algorithmes pour le problème joint de construction des rotations avions et équipages.

1.4 Prise en compte de l'aléa

Le développement des systèmes d'information dans les dernières décennies a démultiplié la quantité de données accessibles sur les processus industriels. Ces données ouvrent la porte à de nouvelles opportunités en terme de prise en compte du risque et de l'aléa en recherche opérationnelle. Une attention particulière a été accordée aux modélisations du risque et de l'aléa dans cette thèse. Dans le contexte des opérations aériennes, le risque est associé à la propagation du retard le long des rotations avions et équipages. Lorsqu'un vol est en retard, une partie de ce retard est absorbée par la période tampon entre ce vol et le vol suivant, et une partie est propagée au vol suivant. L'objectif est d'allouer les périodes tampons entre les vols de manière à minimiser la propagation du retard.

Deux approches principales se partagent le domaine de l'optimisation avec prise en compte de l'aléa. L'approche robuste considère un ensemble de réalisations possibles de l'aléa, mais ne définit pas de distribution de probabilité sur ces réalisations. Le modèle est ensuite optimisé contre la pire réalisation. L'approche stochastique définit au contraire une distribution de probabilité sur les différentes réalisations de l'incertitude. Les modèles d'optimisation prennent alors en compte le risque par le biais de fonctionnelles de probabilité. Une étape clef dans la modélisation stochastique est le choix des distributions de probabilité. En effet, celles-ci doivent être compatibles avec le calcul des fonctionnelles. Deux options s'offrent alors à nous : soit nous adoptons des hypothèses simplificatrices, comme l'indépendance des variables aléatoires, et nous pouvons calculer ces fonctionnelles de manière exacte, soit nous travaillons avec un modèle stochastique précis, auquel cas nous devons approximer les fonctionnelles de probabilité par échantillonnage. Nous utilisons les deux types d'approches dans cette thèse. Pour chaque approche par échantillonnage, nous fournissons des bornes sur l'erreur que celui-ci entraîne.

Remarque 1.1. La distinction entre approche robuste et approche stochastique tend à devenir artificielle. En effet, certaines approches récentes en optimisation robuste considèrent une famille de modèles stochastiques, et optimise contre le modèle de cette famille qui prédit les pires réalisations. Nous n'utilisons pas de telles approches dans cette thèse.

Il existe aussi deux manières de prendre en compte le risque dans les problèmes d'optimisation. La première utilise comme objectif du problème une fonctionnelle de probabilité qui pénalise

les solutions risquées, quand la seconde ajoute des contraintes de probabilité pour interdire les solutions risquées. Par exemple, dans le contexte des opérations aériennes, l'approche par contraintes de probabilité correspondrait à une stratégie marketing qui garantit au client que 90% des vols sont à l'heure. La seconde approche consiste à minimiser les coûts additionnels dus au retard.

Dans cette thèse, nous introduisons des algorithmes pour des versions stochastiques du PROBLÈME D'AIRCRAFT ROUTING, du PROBLÈME DE CREW PAIRING et du PROBLÈME INTÉGRÉ. Dans ces problèmes stochastiques, le retard est pris en compte par le biais de contraintes de probabilité. Pour résoudre la version stochastique du PROBLÈME DE CREW PAIRING, nous adaptons l'algorithme de génération de colonnes qui traite le cas déterministe. Comme le retard se propage le long des séquences de vols, le retard d'un vol est déterminé par le pairing qui le couvre. Le retard peut donc être considéré au sein du problème esclave de la génération de colonnes. Pour passer du PROBLÈME DE CREW PAIRING déterministe au stochastique, il suffit donc d'adapter le problème esclave. Le nouveau problème esclave est un problème de chemin stochastique.

Nous avons déjà mentionné que de nombreux problèmes de chemin stochastiques peuvent être réduit en des instances du PROBLÈME DE PLUS COURT CHEMIN AVEC RESSOURCES DANS UN MONOÏDE, et sont bien résolus par les algorithmes développés pour le PROBLÈME DE PLUS COURT CHEMIN AVEC RESSOURCES DANS UN MONOÏDE. L'idée clef derrière cette réduction est que de nombreuses familles de lois de variables aléatoires sont des treillis lorsqu'elles sont munies de l'ordre adéquat. Par exemple, l'ensemble des lois de variables aléatoires discrètes à support fini est un treillis lorsqu'il est muni de l'ordre stochastique usuel $\leq_{\text{st}}$ défini par

$$\xi \leq_{\text{st}} \tilde{\xi} \quad \text{si} \quad \mathbb{P}(\xi \leq t) \geq \mathbb{P}(\tilde{\xi} \leq t) \quad \text{pour tout } t.$$

Cet ordre partiel est par ailleurs compatible avec le produit de convolution, et nous obtenons donc un monoïde ordonné comme un treillis. Plusieurs autres familles de variables sont considérées dans le Chapitre 5. Un autre exemple important est celui des variables aléatoires à scénarios, c'est à dire des variables aléatoires sur un espace de probabilité Ω fini. L'ordre pertinent est alors

$$\xi \leqslant \tilde{\xi} \quad \text{si} \quad \xi(\omega) \leq_{\text{st}} \tilde{\xi}(\omega) \quad \text{pour tout scénario } \omega \text{ dans } \Omega.$$

Les mesures à scénarios sont importantes car toute mesure peut-être approximée par une mesure à scénario en utilisant une technique d'échantillonnage.

Pour chaque problème d'optimisation stochastique considéré dans cette thèse, nous développons des algorithmes qui peuvent considérer des fonctionnelles de probabilité à la fois dans les contraintes et dans l'objectif, et peuvent être utilisés à la fois dans le cas de variables aléatoires indépendantes et dans le cas de variables aléatoires à scénarios. De plus, nous fournissons des bornes supérieures sur l'erreur commise lorsqu'une approximation par échantillonnage est utilisée. Ces bornes reposent sur des outils probabilistes récents comme par exemple des inégalités de concentration qui bornent la distance de Wasserstein entre la mesure originale et la mesure échantillonnée.

Le chapitre 2 introduit quelques outils probabilistes que nous utiliserons ensuite pour modéliser des problèmes stochastiques et pour obtenir des bornes sur les erreurs commises lorsque nous utilisons des distributions échantillonnées. La chapitre 5 considère des problèmes de chemin stochastiques, et le chapitre 11 développe des modèles stochastiques pour prendre en compte la propagation du retard lors de la construction des rotations avions et équipages d'une compagnie aérienne.

2 Probabilistic tools for stochastic optimization

The purpose of this chapter is to introduce the tools of stochastic optimization required for the analysis of stochastic shortest path problems in Chapter 5, and for the analysis of delay propagation in airline operations in Chapter 11.

The chapter is organized as follows.

- Section 2.1 introduces risk measures, which are probability functionals designed to handle risk.
- Section 2.2 introduces non-asymptotic exponential bounds on the quality of approximation of a stochastic optimization problem by Monte-Carlo methods. The proofs of the theorems of this section are available in Appendix A.

2.1 Risk measures

2.1.1 Version independent and risk-averse probability functional

Let ξ be a random variable on a probability space $(\Omega, \mathscr{F}, \mu)$. A probability functional ρ is *risk-averse* if $\rho(\xi) > \rho(\mathbb{E}(\xi))$ for non-constant random variables ξ, *risk-prone* if $\rho(\xi) < \rho(\mathbb{E}(\xi))$, and *risk-neutral* if $\rho(\xi) = \rho(\mathbb{E}(\xi))$.

A probability functional ρ is *version independent* if $\rho(\xi)$ only depends on the distribution of ξ: if ρ is version independent, then we can write $\rho(\xi) = \rho\left(\mu^{\xi}\right)$ where μ^{ξ} is the image measure of μ under ξ. In this dissertation, we restrict ourselves to version independent probability functionals.

On some practical problems, computing $\rho(\mu^{\xi})$ happens to be non-tractable due to the complexity of μ. A usual technique in these cases is to replace the initial measure μ by a measure $\tilde{\mu}$ such that computing $\rho(\tilde{\mu}^{\xi})$ is tractable. A wide part of the results in this section gives bounds on the error made on $\rho(\mu^{\xi})$ when μ is replaced by $\tilde{\mu}$. In this context, it is convenient to consider that a version independent probability functional ρ has two arguments: the measure μ on the probability space, and the random variable ξ. We therefore introduce the notation $\rho_{\mu}(\xi) = \rho(\mu^{\xi})$.

2.1.2 Risk measures definition

Given a vector space of $\mathscr{F}$-measurable random variables on a probability space $(\Omega, \mathscr{F}, \mu)$, a *coherent risk measure* ρ is a probability functional if each pair $(\xi, \tilde{\xi})$ of random variables satisfies the four following properties:

- *Monotonicity:* $\rho(\xi) \leq \rho(\tilde{\xi})$ if $\xi \leq \tilde{\xi}$ almost surely,
- *Convexity:* $\rho\left(\lambda\xi + (1-\lambda)\tilde{\xi}\right) \leq \lambda\rho(\xi) + (1-\lambda)\rho(\tilde{\xi})$, for all ξ, $\tilde{\xi}$, and $0 \leq \lambda \leq 1$,
- *Translation invariance:* $\rho(\xi + c) = \rho(\xi) + c$ for all $c \in \mathbb{R}$,
- *Positive homogeneity:* $\rho(\lambda\xi) = \lambda\rho(\xi)$ for all $\lambda \in \mathbb{R}_+$.

As we consider only coherent risk measures, we call them simply *risk measures*. Artzner et al. [15] introduced the notion of risk measures in financial mathematics. Many of these axioms have been considered previously in the context of insurance [49, 50, 169]. The first example of risk measure is the expectation. Expectation is risk-neutral. On the contrary, the variance is not a risk measure, and the probability functional $\mathbb{E}c(\cdot)$ of utility theory is not a risk measure either for most cost functions $c(\cdot)$.

The monotonicity property plays a key role in the design of solution algorithm for stochastic path problems in Chapter 5. A consequence of this property in the context of version independent risk measures is given in Section 2.1.5.

2.1.3 Conditional Value at Risk

The *Condition Value-at-Risk* [149] of level $\beta \in [0,1)$ is

$$\mathrm{CVaR}_\beta(\xi) = \frac{1}{1-\beta}\int_\beta^1 \mathrm{VaR}_\alpha(\xi)\,d\alpha \quad \text{where} \quad \mathrm{VaR}_\beta(\xi) = \inf\left\{t | \mathbb{P}(\xi \leq t) \geq \beta\right\}. \tag{2.1}$$

Intuitively, the Conditional Value at Risk of level β can be interpreted as the expectation in the β worst case. Parameter β enables to choose a level of risk awareness. Indeed, if $\beta = 0$, then $\mathrm{CVaR}_\beta = \mathbb{E}$ and the Conditional Value at Risk is risk-neutral. On the contrary, when $\beta \to 1$, CVaR_β converges to the worst case value of ξ.

The Conditional Value at Risk is version independent. Furthermore, the following theorem, known as Kusuoka's representation theorem [111], states that any version independent risk measure can be represented as a sum of CVaR_β.

Theorem 2.1. *If ρ is a version independent risk measure on a probability space without atoms, then there exists a set $\mathscr{M}$ of probability measures on $[0,1]$ such that*

$$\rho(\cdot) = \sup_{\nu \in \mathscr{M}} \int_0^1 \mathrm{CVaR}_\beta(\cdot)\,\nu(d\beta).$$

We do not use Kusuoka's theorem directly in this dissertation, but we use instead its corollary introduced in the next section.

2.1.4 Distortion Functional

In this section, we introduce a parametrization of version independent risk measures. Given a real random variable ξ and a real number $u \in [0,1]$, let

$$F_\xi^{-1}(u) = \min\{x \in \mathbb{R} | \mathbb{P}(\xi \leq x) \geq u\}$$

denote the *Value-at-Risk* of ξ at level u. A *distortion function* $\sigma : [0,1) \to [0,\infty)$ is a nondecreasing and nonnegative function satisfying $\int_0^1 \sigma(u)du = 1$. The *distortion functional* associated to σ is the function

$$\rho_\sigma(\xi) = \int_0^1 \sigma(u) F_\xi^{-1}(u)du.$$

Distortion functionals have been introduced by Denneberg [49] and studied by Acerbi [3], Acerbi and Simonetti [4]. They are easily shown to be version independent risk measures. Besides, a risk measure is *generated by a set of distortion functions* $\mathscr{S}$ if

$$\rho(\xi) = \sup_{\sigma \in \mathscr{S}} \rho_\sigma(\xi). \tag{2.2}$$

The following corollary [143] of Kusuoka's representation theorem replaces Conditional Value-at-Risk by distortion functionals in Theorem 2.1.

Corollary 2.2. *If ρ is a version independent risk measure on a probability space without atoms, then ρ is generated by a set of distortion functions $\mathscr{S}$.*

Corollary 2.2 is used in the statement of Theorems 2.5 and 2.7 and in Appendix A.

2.1.5 Monotonicity with respect to stochastic orders

Monotonicity assumption in the definition of a risk measure ρ states that if $\xi \leq \tilde{\xi}$ almost surely, then $\rho(\xi) \leqslant \rho(\tilde{\xi})$. For version independent risk measures, this property can be extended to other stochastic orders. Detailed studies of stochastic orders are available in the books by Müller and Stoyan [131] and Shaked and Shanthikumar [160].

A random variable ξ is smaller than a random variable $\tilde{\xi}$ for the *usual stochastic order*, denoted $\xi \leq_{\text{st}} \tilde{\xi}$ if

$$\mathbb{P}(\xi \leq t) \geq \mathbb{P}(\tilde{\xi} \leq t) \quad \text{for all } t. \tag{2.3}$$

A probability functional ρ is monotone with respect to the usual stochastic order

$$\xi \leq_{\text{st}} \tilde{\xi} \quad \text{implies} \quad \rho(\xi) \leq \rho(\tilde{\xi}).$$

If $\xi \leq \tilde{\xi}$ almost surely implies $\xi \leq_{\text{st}} \tilde{\xi}$, the converse is false. As a consequence, a probability functional monotone with respect to the usual stochastic order is monotone with respect to almost sure order, but the converse is false. Nonetheless, the following proposition shows that the converse is true for version independent probability functionals.

Proposition 2.3. *A version independent probability functional ρ monotone with respect to the almost sure order is monotone with respect to the usual stochastic order. This is notably the case of version independent risk measures.*

Proposition 2.3 is the cornerstone of the algorithms for stochastic path problems developed in Chapter 5. Bäuerle and Müller [22] provides analogue of Proposition 2.3 in the general case of risk measures that are not version-independent on finite probability spaces and non-atomic probability spaces. To the best of our knowledge, the version-independent case, which is much simpler, has not been studied.

Proof of Proposition 2.3. Let ξ and $\tilde{\xi}$ be such that $\xi \leq_{\text{st}} \tilde{\xi}$. Let F and G be their respective cumulative distribution functions, and F^{-1} and G^{-1} be their right continuous inverses. Equation (2.3) implies $F^{-1}(t) \leq G^{-1}(t)$ for all t in $[0,1]$. Let U be a uniform $[0,1]$ random variable, $\widehat{\xi} = F^{-1}(U)$, and $\widehat{\tilde{\xi}} = G^{-1}(U)$. For any atom ω, we have $\widehat{\xi}(\omega) = F^{-1}(U(\omega)) \leq G^{-1}(U(\omega)) = \widehat{\tilde{\xi}}(\omega)$ By monotonicity of probability functional ρ, this implies $\rho(\widehat{\xi}) \leq \rho(\widehat{\tilde{\xi}})$.

As ξ and $\widehat{\xi}$ (resp. $\tilde{\xi}$ and $\widehat{\tilde{\xi}}$) have the same cumulative distribution, the hypothesis that ρ is version independent implies that $\rho(\xi) = \rho(\widehat{\xi})$ (resp. $\rho(\tilde{\xi}) = \rho(\widehat{\tilde{\xi}})$). The inequality $\rho(\widehat{\xi}) \leq \rho(\widehat{\tilde{\xi}})$ then gives the proposition. □

2.2 Convergence of the Monte-Carlo approximation of a stochastic problem

Let $(\Omega, \mathscr{F}, \mu)$ be a probability space, and ξ be a $\mathbb{R}^d$ random variable on $(\Omega, \mathscr{F}, \mu)$. Let f be a Lipschitz function on $\mathbb{R}^d$, ρ_μ be a version independent probability functional, and $\mathbb{X}_\mu$ be a set-valued version independent probability functional. Consider the following stochastic optimization problem

$$\min_{x \in \mathbb{X}_\mu(\xi)} \rho_\mu\big(f(x,\xi)\big), \tag{2.4}$$

where x is deterministic, $\mathbb{X}_\mu(\xi)$ is the *set of feasible solutions* given ξ, and f is the *objective function*. A difficulty which often arises in stochastic optimization is that the computation of $\rho_\mu\big(f(x,\xi)\big)$ may be intractable for a fixed x. In those case, a standard methodology is to replace μ by another probability distribution $\tilde{\mu}$ such that the computation of $\rho_{\tilde{\mu}}\big(f(x,\xi)\big)$ is simpler. Given an N-sample $\xi^1, \ldots, \xi^N$ of ξ, the principle of Monte-Carlo methods is to replace Problem (2.4) by the following *Monte-Carlo approximation problem*,

$$\min_{x \in \mathbb{X}_{\widehat{\mu}^N}(\xi)} \rho_{\widehat{\mu}^N}\big(f(x,\xi)\big) \quad \text{where} \quad \widehat{\mu}^N = \frac{1}{N}\sum_{i=1}^{N} \delta_{\xi^i}, \tag{2.5}$$

where δ_x is the Dirac distribution in x. The objective of this section is to introduce bounds on the error made when replacing Problem (2.4) by its Monte-Carlo approximation (2.5).

When $\mathbb{X}$ does not depend on ξ, we provide non-asymptotic exponential bounds on the difference between $\rho_\mu\big(f(x^*,\xi)\big)$ and $\rho_\mu\big(f(\widehat{x}^N,\xi)\big)$, where x^* is an optimal solution of Problem (2.4)

and $\widehat{x}^N$ is an optimal solution of Problem (2.5). The bounds are typically of the form

$$\mathbb{P}\left[\rho_\mu\left(f(\widehat{x}^N,\xi)\right)-\rho_\mu\left(f(x^*,\xi)\right)>\varepsilon\right]\leq C'\left(\exp\left(-c'N\varepsilon^d\right)\mathbf{1}_{\varepsilon\leq 1}+\exp\left(-c'N\varepsilon^\alpha\right)\mathbf{1}_{\varepsilon>1}\right), \tag{2.6}$$

where C', c' and α are constants. The non-deterministic term inside the probability on the left hand side is $\widehat{x}^N$, which depends on the sampling of $\widehat{\mu}^N$ from μ. As a consequence, the probability $\mathbb{P}$ is with respect to the sampling of $\widehat{\mu}^N$ from μ. The analysis, which follows the one of Pflug and Pichler [143], is in two steps. First, Theorem 2.4 by Fournier and Guillin [80] ensures that under the assumption that μ admits an exponential moment of degree α, there exists constants C and c such that the Wasserstein distance $W_1(\widehat{\mu}^N,\mu)$ between μ and $\widehat{\mu}^N$ satisfies

$$\mathbb{P}\left[W_1\left(\widehat{\mu}^N,\mu\right)>\varepsilon\right]\leq C\left(\exp\left(-cN\varepsilon^d\right)\mathbf{1}_{\varepsilon\leq 1}+\exp\left(-cN\varepsilon^\alpha\right)\mathbf{1}_{\varepsilon>1}\right). \tag{2.7}$$

Second, Theorems 2.5 and 2.6 show the robustness of optimization with respect to Wasserstein distance through bounds of the type

$$\rho_\mu\left(f(\widehat{x}^N,\xi)\right)-\rho_\mu\left(f(x^*,\xi)\right)\leq\gamma W_1\left(\widehat{\mu}^N,\mu\right), \tag{2.8}$$

where γ is a constant depending on the probability functional ρ. Different bounds are provided depending on the assumptions on ρ and μ.

When $\mathbb{X}$ depends on ξ and is of the form,

$$\mathbb{X}^\varepsilon_\mu(\xi)=\left\{x\in\mathbb{X}|\mathbb{P}_\mu\left[g(x,\xi)>0\right]\leq\varepsilon\right\}\quad\text{where the map }\xi\mapsto g(x,\xi)\text{ is Lipschitz,}$$

we provide results of the type $\mathbb{X}^\alpha_{\widehat{\mu}^N}\subseteq\mathbb{X}^\varepsilon_\mu\subseteq\mathbb{X}^\beta_{\widehat{\mu}^N}$, where α and β only depend on $\varepsilon, W_1(\mu,\widehat{\mu}^N)$ and some regularity assumptions on μ.

2.2.1 Wasserstein distances

Let Ω and $\tilde{\Omega}$ be two probability spaces with probability measures μ and $\tilde{\mu}$, and $\xi:\Omega\to\mathbb{R}^M$ and $\tilde{\xi}:\tilde{\Omega}\to\mathbb{R}^M$ be two random variables on $\mathbb{R}^M$. Then the *inherited distance* $d(\omega,\tilde{\omega})$ between elements of Ω and $\tilde{\Omega}$ is defined as

$$d(\omega,\tilde{\omega})=\left\|\xi(\omega)-\tilde{\xi}(\tilde{\omega})\right\|.$$

Given $p\in[1,+\infty)$, the *Wasserstein distance* of order 1 between μ and $\tilde{\mu}$ inherited from ξ and $\tilde{\xi}$ is

$$W_1(\mu,\tilde{\mu})=W_1(\mu^\xi,\tilde{\mu}^{\tilde{\xi}})=\inf_{\pi\in\Pi(\mu,\tilde{\mu})}\left(\iint d(\omega,\tilde{\omega})d\pi(\omega,\tilde{\omega})\right), \tag{2.9}$$

where π runs over $\Pi\left(\mu,\tilde{\mu}\right)$, the set of all joint probability measures on the product space $\Omega\times\tilde{\Omega}$ with marginals μ and $\tilde{\mu}$.

The following concentration inequality is a special case of Theorem 2 in [80]. It is used in

Section 5.3.4 and Section 11.5.

Theorem 2.4. *Let μ be a probability measure on $\mathbb{R}^d$ with $d > 2$. Suppose that there exist $\alpha > 1$ and $\gamma > 0$ such that*

$$\varepsilon_{\alpha,\gamma}(\mu) = \int_{\mathbb{R}^d} e^{\gamma|x|^\alpha} \mu(dx) < \infty,$$

then there exist c and C such that for all $N \geq 1$ and all $x > 0$,

$$\mathbb{P}\left[W_1\left(\widehat{\mu}^N, \mu\right) > x\right] \leq C\left(\exp\left(-cNx^d\right)\mathbf{1}_{x\leq 1} + \exp\left(-cNx^\alpha\right)\mathbf{1}_{x>1}\right),$$

where constants c and C only depend on d, α, γ, and $\varepsilon_{\alpha,\gamma}(\mu)$.

Fournier and Guillin [80] provide alternative results when the hypothesis only stands for some $\alpha \in (0,1)$.

2.2.2 Robustness of solutions with respect to Wasserstein distance

Theorem 2.5. *Suppose that $\mathbb{X}$ is compact, $\xi \mapsto f(x,\xi)$ is κ-Lipschitz for all x, and that ρ is a version independent risk measure, and $\mathscr{S}$ be a set of distortion function that generates it. The existence of such an $\mathscr{S}$ is ensured by Corollary 2.2. Let μ and $\tilde{\mu}$ be two probability distributions, and let x^* and $\tilde{x}$ denote optimal solutions of $\min_{x\in\mathbb{X}} \rho_\mu\left(f(x,\xi)\right)$ and $\min_{x\in\mathbb{X}} \rho_{\tilde{\mu}}\left(f(x,\xi)\right)$ respectively. Then,*

$$\rho_\mu\left(f(\tilde{x},\xi)\right) - \rho_\mu\left(f(x^*,\xi)\right) \leq 2\kappa \sup_{\sigma\in\mathscr{S}} \|\sigma\|_\infty W_1\left(\mu,\tilde{\mu}\right).$$

Note that the existence of optimal solutions x^* and $\tilde{x}$ is a consequence of the compactness of $\mathbb{X}$ and the fact that f is Lipschitz.

In this dissertation, we consider three types of probability functionals ρ. Risk measures, expectation of cost functions, and probability $\mathbb{P}(\cdot > \tau)$ of being greater than given a threshold τ. Theorem 2.5 enables to deal with risk measures and expectation of Lipschitz cost functions. The next theorem enables to deal with $\mathbb{P}(\cdot > \tau)$, and with expectation of non-decreasing piecewise Lipschitz functions when combined with Theorem 2.5.

Theorem 2.6. *Suppose that $\mathbb{X}$ is compact, $\xi \mapsto f(x,\xi)$ is κ-Lipschitz for all x, and that τ is a given threshold, and let x^* and $\tilde{x}$ be defined as in Theorem 2.5.*

1. *If the support of $f(x,\xi)$ under μ and $\tilde{\mu}$ is in $\mathbb{Z}$ for all x, then*

$$\mathbb{P}_\mu\left(f(\tilde{x},\xi) > \tau\right) - \mathbb{P}_\mu\left(f(x^*,\xi) > \tau\right) \leq 2\kappa W_1\left(\mu,\tilde{\mu}\right).$$

2. *If there exists a B such that, for each $x \in \mathbb{X}$, the distribution of $f(x,\xi)$ is non-atomic and with density bounded by B, then*

$$\mathbb{P}_\mu\left(f(\tilde{x},\xi) > \tau\right) - \mathbb{P}_\mu\left(f(x^*,\xi) > \tau\right) \leq 4\sqrt{B\kappa W_1(\mu,\tilde{\mu})}.$$

Theorems 2.5 and 2.6 are proved in Appendix A. They are used in Section 5.3.4.

Remark 2.1. The fact that μ admits a bounded density on $\mathbb{R}^m$ does not imply that the induced measure $\mu^{f(x,\xi)}$ has a bounded density.

Remark 2.2. The assumption that $\mathbb{X}$ is compact in Theorems 2.5 and 2.6 is required to ensure the existence of x^* and $\tilde{x}$. Without this assumption, analogue results can be obtained on the value of the problems. See for instance Theorem 6.1 in [143].

2.2.3 Probabilistic constraints

In this section, we consider the case of stochastic optimization with non-deterministic feasible set

$$\min_{x\in\mathbb{X}^{\varepsilon}_{\mu}(\xi)} \rho_{\mu}\left[f(x,\xi)\right], \tag{2.10}$$

where $\xi \mapsto f(x,\xi)$ is κ_f-Lipschitz for all x, and the set of feasible solutions satisfies probabilistic constraints of the form

$$\mathbb{X}^{\varepsilon}_{\mu}(\xi) = \left\{x \in \mathbb{X} | \mathbb{P}_{\mu}\left[g(x,\xi) > 0\right] \leq \varepsilon\right\} \quad \text{where the mapping } \xi \mapsto g(x,\xi) \text{ is Lipschitz,}$$

with $\mathbb{X}$ compact, which implies that $\mathbb{X}^{\varepsilon}_{\mu}(\xi)$ is compact. Note that taking one constraint $g(x,\xi) \leq 0$ is non-restrictive in practice. Indeed, suppose that we have m constraints $g^i(x,\xi) \leq 0$, if we define $g(x,\xi) = \max_i g^i(x,\xi)$, we have $\mathbb{P}\left[\exists i : g^{i(x,\xi)} > 0\right] = \mathbb{P}\left[g(x,\xi) > 0\right]$.

Given a probability distributions $\tilde{\mu}$, we approximate Problem (2.10) through the resolution of the two following problems

$$\min_{x\in\mathbb{X}^{\alpha}_{\tilde{\mu}}(\xi)} \rho_{\tilde{\mu}}\left[f(x,\xi)\right] \quad \text{and} \quad \min_{x\in\mathbb{X}^{\beta}_{\tilde{\mu}}(\xi)} \rho_{\tilde{\mu}}\left[f(x,\xi)\right] \quad \text{with} \quad 0 < \alpha < \varepsilon < \beta.$$

The following theorem shows that with α and β carefully chosen as functions of the Wasserstein distance between μ and $\tilde{\mu}$, then $\min_{x\in\mathbb{X}^{\alpha}_{\tilde{\mu}}(\xi)} \rho_{\tilde{\mu}}\left[f(x,\xi)\right]$ provides a feasible solution of the initial problem (2.10) and an upper bound on its value, and $\min_{x\in\mathbb{X}^{\beta}_{\tilde{\mu}}(\xi)} \rho_{\tilde{\mu}}\left[f(x,\xi)\right]$ provides a lower bound on the value of (2.10).

Theorem 2.7. *Suppose that one of the following assumptions is satisfied.*

1. *For all x and ξ in the union of the supports of μ and $\tilde{\mu}$, $g(x,\xi) \in \mathbb{Z}$.*
2. *There exists a B such that, for each $x \in \mathbb{X}$, the distribution of $f(x,\xi)$ is non-atomic and with density bounded by B.*

Then,

$$\mathbb{X}^{\alpha}_{\tilde{\mu}} \subseteq X^{\varepsilon}_{\mu} \subseteq X^{\beta}_{\tilde{\mu}}, \quad \textit{where} \quad \begin{cases} \alpha = \varepsilon - \kappa_G W_1(\mu,\tilde{\mu}), \\ \beta = \varepsilon + \kappa_G W_1(\mu,\tilde{\mu}), \end{cases} \quad \textit{under Assumption 1,}$$

$$\begin{cases} \alpha = \varepsilon - 2\sqrt{B\kappa_G W_1(\mu,\tilde{\mu})}, \\ \beta = \varepsilon + 2\sqrt{B\kappa_G W_1(\mu,\tilde{\mu})}, \end{cases} \quad \textit{under Assumption 2,}$$

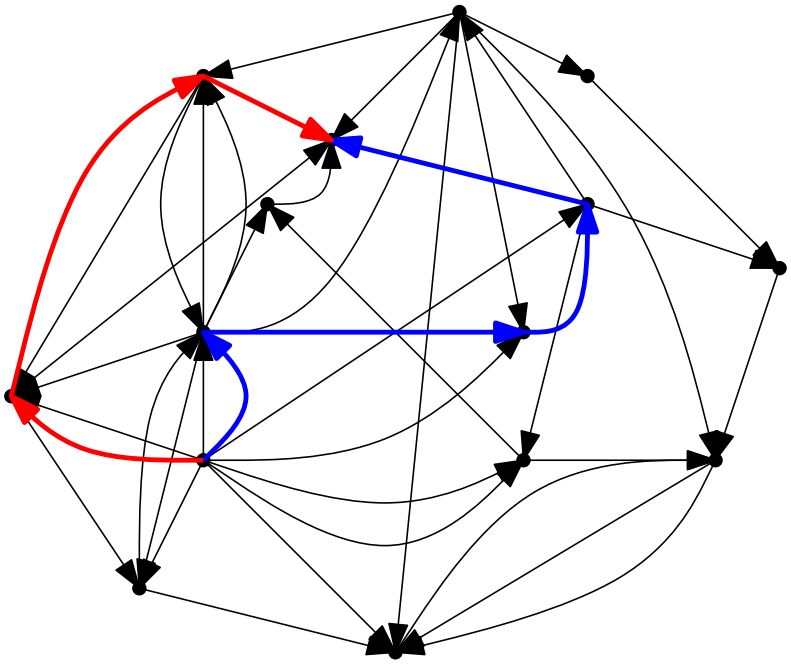
κ_G is the Lipschitz constant of g, and $X^{\alpha}_{\tilde{\mu}} = \emptyset$ if $\alpha < 0$. If in addition ρ_{μ} is a version independent

risk measure generated by a set of functionals $\mathscr{S}$ and $\sigma_\infty = \sup_{\sigma\in\mathscr{S}} \|\sigma\|_\infty$, we have

$$\min_{x\in\mathbb{X}^{\beta}_{\tilde{\mu}}(\xi)} \rho_{\tilde{\mu}}\left[f(x,\xi)\right] - 2\kappa_f\sigma_\infty W_1(\mu,\tilde{\mu}) \le \min_{x\in\mathbb{X}^{e}_{\mu}(\xi)} \rho_{\mu}\left[f(x,\xi)\right] \le \min_{x\in\mathbb{X}^{\alpha}_{\tilde{\mu}}(\xi)} \rho_{\tilde{\mu}}\left[f(x,\xi)\right] + 2\kappa_f\sigma_\infty W_1(\mu,\tilde{\mu}).$$

Theorem 2.7 is used in Section 5.3.4 and Section 11.5, and proved in Appendix A.

Shortest path problems Part I



Introduction to Part I

This part focuses on resource constrained shortest path problems. We recall that a *monoid* $(\mathscr{R}, \oplus)$ is a set $\mathscr{R}$ endowed with an associative law $\oplus$ which admits a neutral element. Given a partial order $\leqslant$ on $\mathscr{R}$, the monoid $(\mathscr{R}, \oplus, \leqslant)$ is a *lattice ordered monoid* if $\oplus$ induces a lattice structure on $\mathscr{R}$ and if translations $\cdot \mapsto \cdot \oplus q$ and $\cdot \mapsto q \oplus \cdot$ are non decreasing for each q in $\mathscr{R}$. Lattice ordered monoids and graph notions are introduced in Section 3.1.1. A general framework for resource constrained shortest path problems is the following.

MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM
Let $(\mathscr{R}, \oplus, \leqslant)$ be a lattice ordered monoid.
Input. A digraph $D = (V, A)$, two vertices $o, d \in V$, a collection $(q_a) \in \mathscr{R}^A$, and two non-decreasing mappings $c : \mathscr{R} \to \mathbb{R}$ and $\rho : \mathscr{R} \to \{0, 1\}$.
Output. An o-d path P such that $\rho\left(\bigoplus_{a \in P} q_a\right) = 0$ and with minimum $c\left(\bigoplus_{a \in P} q_a\right)$.

$\mathscr{R}$ is the *set of resources*. The *resource* of a path P is $\bigoplus_{a \in P} q_a$, and the *cost* of P is $c\left(\bigoplus_{a \in P} q_a\right)$. P is *feasible* if $\rho\left(\bigoplus_{a \in P} q_a\right)$ is a equal to 0. ρ is latter referred as the *infeasibility function*. We denote q_P the resource $\bigoplus_{a \in P} q_a$ of a path P. Both mappings are considered as oracles: apart from their monotonicity and the fact that we can compute their result, we do not make or use assumptions on their properties. The lattice order $\leqslant$ enables to compare the resources. It is a crucial element in the design of algorithms for the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. The MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM contains the standard shortest path and resource constrained shortest path problems with one or several resources or constraints. The MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM is therefore $\mathscr{NP}$-complete as the standard resource constrained shortest path problem is $\mathscr{NP}$-complete.

The main messages of this part are the following ones.

- Practically efficient algorithms can be derived for the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM.
- The lattice ordered monoid algebraic structure is light, and therefore a wide range of non-constrained and resource constrained shortest path problems can be modeled within the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework.

The practical advantage of this approach is that all the algorithms mentioned above depend only on the functions ρ and c and on the operators $\oplus$ and $\leqslant$ of the ordered monoid $(\mathscr{R}, \oplus, \leqslant)$. We have therefore implemented them once and for all in a `C++` library, and we have then used this library to solve a wide range of path problems. The main features of this library

are described in Appendix C. Solving practical instances then only requires to implement the specific resource set $(\mathscr{R}, \oplus, \preccurlyeq)$ and functions ρ and c considered. We fully detail this procedure on three types of problems:

- In Chapter 3, we consider the toy problem of finding a continent-wide minimum cost path for a truck driver, taking into account hotel costs and the fact that a driver should not drive more than a given number of hours per day. This toy problem enables to introduce the key techniques to model within the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework.
- In Chapter 5, we show how stochastic counterparts of problems that can be treated in the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework can also be treated in this framework. We detail the cases of the generic shortest path problem and of the generic resource constrained shortest path problem.
- Finally, we use the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM algorithms to solve the subproblems of the column generation approaches we use to optimize Air France operations in Part II. More precisely we solve the problem of building sequences of flights for Air France crews in Chapter 9, and its stochastic counterpart taking into account flight legs delay in Chapter 11.

The specificity of the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM lies in the lattice ordered structure of the resource set. The other resource constrained shortest path frameworks in the literature [96, 97] endow the resource set $\mathscr{R}$ with an order $\preccurlyeq$, but they do not assume that it implies a lattice structure on $\mathscr{R}$. Besides, they do not define an associative operator sum operator $\oplus$. Instead, they define a resource extension function ζ_a on each arc a of the network. Given the resource q_P of a path P, the resource of path P followed by arc a is $\zeta_a(q_P)$. The main advantage of an associative operator $\oplus$ over resource extension functions is that the associativity of $\oplus$ enables to compute paths resources in both directions, i.e. from the beginning to the end or from the end to the beginning of the path. Joined with the lattice structure, this enables to define standardized algorithms that compute bounds on paths resources. It is well known that using bounds in an A^* like way enables to dramatically increase the performance of the resource constrained shortest path algorithms [147]. There exists some problem-specific technique to build such bounds when the resource extension function is simple, or based on Lagrangean relaxation when the problem can be written as an integer program. But, to the best of our knowledge, there are no standard procedure to build such bounds for generic resource extension functions, and no standard procedure to build bounds for non-linear and stochastic resource constrained shortest path problems. The main advantage of lattice ordered monoid structure is that it enables to standardize the construction and the use of these bounds.

Indeed, the algebraic path problem literature has shown that the usual polynomial algorithms for the standard shortest path problem can be generalized to idempotent semiring. We show in Chapter 4 that the solution of these generalized algorithms can be interpreted as a lower bound on the resource of all the paths between given origin and destination vertices. Unfortunately, the semiring structure is too strong and does not enable to model many practical applications of the resource constrained shortest path problem. Nonetheless, idempotent semirings are special cases of lattice ordered monoids, and we show in this dissertation that these polynomial

algorithms can be generalized to lattice ordered monoids, and can therefore be used as standard procedures to build lower bounds on paths resources. Besides, the lattice ordered monoid structure is versatile enough to model almost all problems that can be handled within the resource extension function framework. The price we have to pay for this algebraic structure is that modeling can be made more technical, especially to obtain the associativity. But the increase in algorithms performance it enables justifies the use of the lattice ordered monoid framework on difficult problems.

Part I is organized as follows:

- Chapter 3 details the graph notions used and introduces the main notions on lattices and ordered monoid that are used in the remaining of the part. It then shows that many resource constrained shortest path problems studied in the literature can be modeled as instances of the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. Finally, it outlines the links with other frameworks in the literature.
- Chapter 4 is the main chapter of this part. It generalizes Ford-Bellman, Dijkstra, and A* algorithms to the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework. These new algorithms are tested numerically on stochastic path problems in Chapters 5 and 6, and are used as subroutine in the column generation algorithm of Chapter 9.
- Chapter 5 focuses on stochastic path problems as specific cases of the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. It also contains a probabilistic analysis of these problems.
- Chapter 6 introduces the notion of state graph to enhance the algorithms of Chapter 4 and to make them able to better solve the difficult MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM instances outlined in Chapter 4 and Chapter 5. Chapter 6 is technical and independent from the remaining of this dissertation. It can be skipped in a first read.

3 Algebraic structure of the resource set in path problems

The goals of this chapter are to introduce the algebraic and graph notions used in this part, to compare the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework to other path problems frameworks in the literature, and to show the modeling power of the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework.

Chapter 3 is organized as follows:

- Section 3.1 introduces the mathematical notions on graphs, lattices and monoids used along Part I.
- Section 3.2 re-states the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. It shows that the standard path problems are instances of the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. The main properties enjoyed by the solutions of the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM are then highlighted. These properties are the building blocks of the generalization of A* algorithm and of dynamic programming in the next chapter.
- Section 3.3 introduces the algebraic path problem semiring framework, and shows its links with our lattice ordered monoid framework.
- Section 3.4 shows the modeling power of the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework. As we said in the introduction, the most common resource constrained shortest path framework is based on resource extension functions. After a review of the different classes of resource constrained shortest path problems considered in resource extension function literature, we show that, through the use of some modeling techniques, most of them can be dealt with within the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework. To illustrate these modeling techniques, we detail how the continent wide truck delivery toy problem of Chapter 1 can be modeled as a MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM.

Bibliographical remarks on non-constrained and resource constrained shortest path problems are provided in Sections 3.3 and 3.4.

3.1 Generalities on graphs and ordered monoids

3.1.1 Digraphs

A *digraph* D is a pair (V, A) where V is the set of *vertices* and A is the set of *arcs* of D. The set of arcs A is a multiset of elements of V^2. An *arc* a links a *tail* vertex to a *head* vertex. An arc a is *incoming* to (resp. *outgoing* from) v if v is the head (resp. the tail) of v. The set of arcs incoming to (resp. outgoing from) v is denoted $\delta^-(v)$ (resp. $\delta^+(v)$). A *path* is a sequence of arcs $a_1, \ldots, a_k$ such that for each $i \in \{1, \ldots, k-1\}$, the head vertex of a_i is the tail vertex of a_{i+1}. Note that with this definition, paths can contain multiple copies of an arc or of a vertex. A path P is *elementary* if it contains at most one copy of each vertex. It is *simple* if it contains at most one copy of each arc. The *origin* of a path is the tail of its first arc and its *destination* is the head of its last arc. Given two vertices o and d in V, an *o-d path* P is a path with origin o and destination d. A *cycle* is a path whose origin is identical to its destination. Given two paths P_1 and P_2 such that the destination of P_1 is equal to the origin of P_2, we denote $P_1 + P_2$ the path whose sequence of arcs is composed of the arcs of P_1 followed by the arcs of P_2

3.1.2 Ordered monoid and lattices

Let $(\mathscr{R}, \leqslant)$ and $(\mathscr{S}, \leq)$ be two partially ordered sets. A map $\rho : \mathscr{R} \to \mathscr{S}$ is *isotone* if $q \leqslant \tilde{q}$ implies $\rho(q) \leq \rho(\tilde{q})$. It is *antitone* if $q \leqslant \tilde{q}$ implies $\rho(q) \geq \rho(\tilde{q})$. A mapping is *monotone* if it is either isotone or antitone. Given two partial orders $\leqslant$ and $\preceq$ on a set $\mathscr{R}$, $\leqslant$ is *coarser* than $\preceq$ if $q \preceq \tilde{q}$ implies $q \leqslant \tilde{q}$

Let $(\mathscr{R}, \oplus)$ be a set endowed with a law of composition. $(\mathscr{R}, \oplus)$ is a *monoid* if $\oplus$ is associative and admits a neutral element in $\mathscr{R}$. A partial order $\leqslant$ is a *compatible order* on $(\mathscr{R}, \oplus)$ if all translations $q \mapsto q \oplus \tilde{q}$ and $q \mapsto \tilde{q} \oplus q$ are isotone. An *ordered monoid* $(\mathscr{R}, \oplus, \leqslant)$ is a monoid endowed with a compatible order. An element q of an ordered monoid $(\mathscr{R}, \oplus, \leqslant)$ is *positive* if $q > 0$ where 0 denotes the neutral element of $\oplus$.

A partially ordered set $(\mathscr{R}, \leqslant)$ is a *lattice* if each pair $(q, \tilde{q})$ of elements of $\mathscr{R}$ admits a greatest lower bound or *meet* and a least upper bound or *join*. The join of two elements q and $\tilde{q}$ is denoted $q \vee \tilde{q}$, and their meet is denoted $q \wedge \tilde{q}$. It is a *lower semi-lattice* if each pair $(q, \tilde{q})$ of elements of $\mathscr{R}$ admits a meet. An ordered monoid $(\mathscr{R}, \oplus, \leqslant)$ is a *lattice ordered monoid* if $\leqslant$ induces a lattice structure on $\mathscr{R}$. It is a *lower semi-lattice ordered monoid* if $\leqslant$ induces a lower semi-lattice structure. The notions of meet and join can be extended to subsets of a lattice. When they exist, the least upper bound of a subset S is the *join* of S and is denoted $\bigvee S$ or $\bigvee_{q \in S} q$, and the greatest lower bound is the *meet* of S and is denoted $\bigwedge S$ or $\bigwedge_{q \in S} q$. A lattice $\mathscr{R}$ is *complete* if each subset $S \subseteq \mathscr{R}$ admits a meet and a join. Any lattice $\mathscr{R}$ can be completed, i.e. transformed into a complete lattice $\mathscr{R}'$ by adding some elements.

A mapping between partially ordered sets $f : S_1 \to S_2$ is an embedding if it is injective and $x \leq y$ if and only if $f(x) \leq f(y)$. Two partially ordered set $\mathscr{R}$ and $\mathscr{R}'$ are isomorph if there exists a mapping ϕ from $\mathscr{R}$ to $\mathscr{R}'$ that admits an inverse ϕ^{-1} and such that ϕ and ϕ^{-1} are isotone. A partially ordered set of $\mathscr{R}'$ is a *completion* of a partially ordered set $\mathscr{R}$ if $\mathscr{R}'$ is a complete lattice and there exists an embedding ϕ of $\mathscr{R}$ into $\mathscr{R}'$. Any lattice can be embedded in a complete

lattice. However, this completion can be much larger than the initial lattice. For instance, the smallest completion of $\mathbb{Q}$ is $\mathbb{R} \cup \{-\infty, +\infty\}$. To explain why it is not the case for the lattices that we consider, we introduce the notion of conditional completeness.

A partially ordered set $\mathscr{R}$ is *conditionally complete* if any subset $S \subseteq \mathscr{R}$ that has an upper bound and is not empty has a join. The following lemma is a well known result on conditionally complete lattices [159].

Lemma 3.1. *Let $\mathscr{R}$ be a conditionally complete lattice. Then*

1. *$\mathscr{R} \cup \{-\infty, +\infty\}$ is a completion of $\mathscr{R}$,*
2. *if the join $\bigvee \mathscr{R}$ (resp. the meet $\bigwedge \mathscr{R}$) exists, then $\mathscr{R} \cup \{-\infty\}$ (resp. $\mathscr{R} \cup \{+\infty\}$) is a completion of $\mathscr{R}$,*
3. *if the join $\bigvee \mathscr{R}$ and the meet $\bigwedge \mathscr{R}$ exist, then $\mathscr{R}$ is a complete lattice,*

where $-\infty$ and $+\infty$ are respectively the meet and the join of the resulting lattice.

All the practical examples of lattices we consider in this dissertation are conditionally complete. The definition of several quantities considered in this dissertation require a complete lattice. Whenever such a quantity is defined, we mention that it may belong to the completion of $\mathscr{R}$. As running algorithms on $\mathscr{R} \cup \{-\infty, +\infty\}$ is not more difficult that on $\mathscr{R}$, such quantities do not affect the practical performance of the algorithms.

A (lattice) ordered monoid $(\mathscr{R}, \oplus, \leqslant)$ is a *(lattice) ordered group* if $(\mathscr{R}, \oplus)$ is a group. A lower semi-lattice ordered group is a lattice ordered group. Indeed, given two elements a and b, their $a \vee b$ exists and is equal to $a \oplus (a \wedge b)^{-1} \oplus b$.

For an introduction to lattices, see e.g. [47, 85, 159]. For an introduction to lattice ordered algebraic structures, see [28].

3.2 Monoid Resource Constrained Shortest Path Problem

We now recall the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM.

MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM
Let $(\mathscr{R}, \oplus, \leqslant)$ be a lattice ordered monoid.
Input. A digraph $D = (V, A)$, two vertices $o, d \in V$, a collection $(q_a) \in \mathscr{R}^A$, and two isotone mappings $c : \mathscr{R} \to \mathbb{R}$ and $\rho : \mathscr{R} \to \{0, 1\}$.
Output. An o-d path P such that $\rho\left(\bigoplus_{a \in P} q_a\right) = 0$ and with minimum $c\left(\bigoplus_{a \in P} q_a\right)$.

$\mathscr{R}$ is the *set of resources*. The *resource* of a path P is $\bigoplus_{a \in P} q_a$, and the *cost* of P is $c\left(\bigoplus_{a \in P} q_a\right)$. P is *feasible* if $\rho\left(\bigoplus_{a \in P} q_a\right)$ is a equal to 0. Mappings c and ρ are respectively called the cost and the infeasibility functions. Section 3.2.1 gives first examples of MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM, and Section 3.2.2 highlights its main properties.

Remark 3.1. Our algorithms for the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM only need the $\oplus$, $\leqslant$ and $\wedge$ operators. As a consequence, we only need $(\mathscr{R}, \oplus, \leqslant)$ to be a lower semi-lattice ordered monoid, and all the results given in this dissertation remain true in that case. Nonetheless, in all practical applications, a join can be defined and $(\mathscr{R}, \oplus, \leqslant)$ is lattice ordered monoid. We stick to the lattice ordered monoid case to simplify terminology.

3.2.1 First examples

A first application of the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM is the STANDARD SHORTEST PATH PROBLEM.

STANDARD SHORTEST PATH PROBLEM
Input. A digraph $D = (V, A)$, two vertices $o, d \in V$, and a collection $(c_a) \in \mathbb{R}^A$.
Output. An o-d path P with minimum $\sum_{a \in P} c_a$

The set of real numbers $\mathbb{R}$ endowed with its usual sum and order is a lattice ordered monoid. It then suffices to define $q_p = c_p$, $\rho = 0$, and $c(q) = q$ to obtain the reduction to a MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM.

A second natural application is the STANDARD RESOURCE CONSTRAINED SHORTEST PATH PROBLEM.

STANDARD RESOURCE CONSTRAINED SHORTEST PATH PROBLEM
Input. A digraph $D = (V, A)$, two vertices $o, d \in V$, collections (c_a) and (w_a) in $\mathbb{R}^A$, and a threshold $\tau \in \mathbb{R}$.
Output. An o-d path P such that $\sum_{a \in P} w_a \leq \tau$ and with minimum $\sum_{a \in P} c_a$.

The set $\mathbb{R}^2$ endowed with its usual sum and its component-wise order is a lattice ordered monoid, and $(q_1, q_2) \wedge (\widetilde{q_1}, \widetilde{q_2}) = \big(\min(q_1, \widetilde{q_1}), \min(q_2, \widetilde{q_2})\big)$. Arc resource q_a are defined as (c_a, w_a), and isotone oracles ρ and c as $c : (q_1, q_2) \to q_1$ and $\rho : (q_1, q_2) \mapsto \begin{cases} 0 & \text{if } q_2 \leq \tau, \\ 1 & \text{otherwise.} \end{cases}$

The $\mathcal{NP}$-completeness of the STANDARD RESOURCE CONSTRAINED SHORTEST PATH PROBLEM implies that the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM is $\mathcal{NP}$-complete. More generally, $\mathbb{R}^n$ endowed with its usual sum and component-wise order is a lattice ordered monoid. We use it several times in this dissertation, and notably in the sample-based stochastic approaches to the shortest path problem considered in Chapter 5.

3.2.2 Main properties

There are two types of algorithms for the STANDARD SHORTEST PATH PROBLEM. The enumeration algorithms, such as A*, which rely on bounds to discard paths, and the polynomial algorithm, such as Ford-Bellman or Dijkstra algorithm. In the next chapter, we generalize them to the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. In this section, we outline the properties on which rely both STANDARD SHORTEST PATH PROBLEM algorithms type, and show how they generalize to the lattice ordered monoïd framework.

We start by the enumeration algorithms. Consider an instance of the STANDARD SHORTEST PATH PROBLEM. Let c_P denote the cost $\sum_{a \in P} c_a$ of a path P in the STANDARD SHORTEST PATH PROBLEM setting. The enumeration algorithms rely on the following property to cut paths.

Property 3.2. *Weak subpath property*
Consider an instance of the STANDARD SHORTEST PATH PROBLEM. Let P_1 and P_2 be o-v paths, and P be a v-d path. Then $c_{P_1} \leq c_{P_2}$ implies $c_{P_1+P} \leq c_{P_2+P}$, where $P_i + P$ denotes the path

composed of P_i followed by P.

The following lemma is a generalization of Property 3.2, enables to generalize enumeration algorithms to the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM.

Property 3.3. Let P_1 and P_2 be o-v paths, and P be a v-d path. If $q_{P_1} \leqslant q_{P_2}$, then

$$q_{P_1+P} \leqslant q_{P_2+P}, \quad \rho(q_{P_1+P}) \leq \rho(q_{P_2+P}), \quad \text{and} \quad c(q_{P_1+P}) \leq c(q_{P_2+P}).$$

The polynomial algorithms for the STANDARD SHORTEST PATH PROBLEM rely on the following dynamic programming property.

Property 3.4. *Dynamic programming property*
Let b_v be cost of a shortest v-d path.

$$b_v = \min_{(v,u)\in\delta^+(v)} c_{(v,u)} + b_u$$

where $\delta^+(v)$ is the set of arcs outgoing from v.

The dynamic programming equation of Property 3.4 cannot be directly generalized to the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. Indeed, the lattice order is not a total ordering, and therefore there is no notion of minimum for this order. Nonetheless, as $(\mathscr{R}, \oplus, \leqslant)$ is a lattice ordered monoid, we can define the following generalization of the dynamic equation, where the min operator is replaced by the meet operator

$$b_v = \bigwedge_{(v,u)\in\delta^+(v)} q_{(v,u)} \oplus b_u. \tag{3.1}$$

As we will see in Chapter 4, Equation (3.1) always admits a solution, and this solution can be computed in polynomial time using generalization of Dijkstra and Ford-Bellman algorithms. Besides, if (b_v) is a solution of Equation (3.1), then it is a lower bound on the resource of any v-d path. This lower-bound can then be used by the A^* algorithm.

3.3 Links with the algebraic path problem

Another algebraic framework, *the algebraic path problem*, has been developed to generalize the STANDARD SHORTEST PATH PROBLEM. There is no cost or feasibility functions in that framework. The objective of the algebraic path problem is to solve Equation (3.1) under slightly different hypotheses on the structure resource set. Therefore, the algebraic path problem does not apply to the resource constrained and stochastic shortest path problems for which the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM has been designed. Nonetheless, as we are interested in solving Equation (3.1) to obtain bounds, the algebraic path problem theory provides insightful results for some specific lattice ordered monoid on the convergence of the algorithms we use. We therefore now introduce the algebraic path problem.

The algebraic path problem is a generalization of the STANDARD SHORTEST PATH PROBLEM where arc resources belong to a semiring. It has been introduced as a common framework

to study algorithms developed for several paths, flows, and algebraic problems [25, 59, 76–78, 170, 175]. Numerous authors studied the algebraic path problem with semiring frameworks of varying generality [8, 17, 34, 74, 86, 115, 130, 152, 178]. Fink [74] surveys the contributions on the algebraic path problem. We introduce the algebraic path problem as is has been defined by Mohri [130]. The algebraic results we use to make the link between the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM and the algebraic path problem frameworks are available in e.g. [86, 178]. Bibliographical remarks on the algorithms for the algebraic path problems are available in Chapter 4.

3.3.1 Problem statement

We use the notations + and × for generic binary operators on a set only in this section, in order to distinguish semiring and lattice ordered monoid notations. Elsewhere in this dissertation, operator + refers to the standard sum on $\mathbb{R}$. Given two binary operators + and × on a set $\mathscr{R}$, operator × is *distributive* with respect to + if $(a+b)\times c = (a\times c)+(b\times c)$ and $c\times(a+b) = (c\times a)+(c\times b)$ for all a, b, and c in $\mathscr{R}$. An element e is an *absorbing element* for a binary operator × if $q\times e = e\times q = e$ for all q in $\mathscr{R}$. Let $(\mathscr{R},+,\times)$ be a set endowed with two binary operators. Then $(\mathscr{R},+,\times)$ is a *semiring* if, first, $(\mathscr{R},+)$ is a commutative monoid whose neutral element for + is denoted e, second, $(\mathscr{R},\times)$ is a monoid whose neutral element for × is denoted ε, third, operator × is distributive with respect to +, and fourth, e is an absorbing element for ×. An element q of a semiring $(\mathscr{R},+,\times)$ is *idempotent* if $q+q=q$. An *idempotent semiring*, or *diod*, is a semiring whose elements are all idempotent[1]. A semiring is *bounded* if ε is an absorbing element for +. A bounded semiring is idempotent, and therefore it is a dioid.

Given a digraph $D=(V,A)$, resources q_a in a bounded dioid $(\mathscr{R},+,\times)$, and two vertices (o,d) the *shortest distance* $\delta(o,d)$ between o and d is defined to be

$$\begin{cases} \delta(o,d) = \varepsilon & \text{if o = d} \\ \delta(o,d) = \displaystyle\sum_{P\in\mathscr{P}_{od}} q_P & \text{otherwise,} \end{cases} \quad \text{where} \quad q_P = \prod_{a\in P} q_a \tag{3.2}$$

and $\mathscr{P}_{od}$ denotes the set of o-d paths. The *algebraic shortest path problem* consists in computing $\delta(o,d)$.

Remark 3.2. The shortest distance can be defined in the case of k-closed semirings [130], which is a little more general than the one of bounded dioids. But for our needs, bounded dioids are sufficient, and later in the section, we give an interpretation of Equation (3.2) in the case on non-bounded dioids.

Given a destination vertex d, the shortest distances $b_v = \delta(v,d)$ for each vertex v are well known to be the solutions of the dynamic programming equation

$$\begin{cases} b_d = \varepsilon & \text{if o = d,} \\ b_v = b_v + \displaystyle\sum_{(v,u)\in\delta^+(v)} q_{(v,u)}\times b_u & \text{otherwise.} \end{cases} \tag{3.3}$$

[1] Several distinct algebraic structures have been called dioid. See Remark 3.4 for more details

3.3.2 Link with our lattice ordered monoid framework

On a dioid $(\mathcal{R},+,\times)$, the relation $\leq_+$ defined by

$$q \leq_+ \tilde{q} \quad \text{if} \quad q+\tilde{q}=q$$

is a partial order called the *canonical order* on $(\mathcal{R},+,\times)$. Besides, $\leq_+$ is compatible with $\times$ [86], and $\leqslant_+$ induces a lower semi-lattice structure on $\mathcal{R}$. Indeed, suppose that $q \leqslant_+ a$ and $q \leqslant_+ b$. Then $q+a+b=(q+a)+b=q+b=q$, and thus $q \leq a+b$. We therefore have that operator $+$ is the meet operator for $\leqslant_+$. This canonical order enables to make the link between lattice ordered monoid and dioids.

Let $(\mathcal{R},+,\times)$ be a dioid. Then $(\mathcal{R},\times,\leq_+)$ is a lower semi-lattice ordered monoid whose meet operator is $+$, and e is the join of $\mathcal{R}$ for $\leq_+$. Indeed, $\leq_+$ is compatible with $\times$, and $(\mathcal{R},\leq_+)$ is a lower semi-lattice. We mentioned in Remark 3.1, the lower semi-lattice ordered monoid structure is sufficient for our framework. Besides, if $(\mathcal{R},+,\times)$ is bounded, all the elements of $\mathcal{R}\backslash\{\varepsilon\}$ are positive.

Conversely, let $(\mathcal{R},\oplus,\leqslant)$ be an upper bounded lattice ordered monoid, and $\wedge$ be its meet operator. If $\oplus$ is distributive with respect to $\wedge$, then $(\mathcal{R},\wedge,\oplus)$ is a dioid. Indeed, the neutral element of $\wedge$ is the join $+\infty$ of $\mathcal{R}$, and $(\mathcal{R},\wedge)$ is a commutative monoid, and $\wedge$ is idempotent. The result then follows from the definition of a dioid. Later in this dissertation, we say that a lattice ordered monoid is a dioid when $\oplus$ is distributive with respect to $\wedge$. Indeed, the compatibility of the order implies that given the resources q, $\tilde{q}$ and b,

$$(q\wedge\tilde{q})\oplus b \leqslant (q\oplus b)\wedge(\tilde{q}\oplus b) \quad \text{and} \quad b\oplus(q\wedge\tilde{q}) \leqslant (b\oplus q)\wedge(b\oplus\tilde{q}), \tag{3.4}$$

but the converse inequality is not satisfied in the general case. Chapter 5 introduces the practically important lattice ordered monoid of independent discrete distributions, which is not a dioid.

Note that our generalized dynamic programming equation (3.1) corresponds to the dynamic programming equation for dioid (3.3) written for $(\mathcal{R},\wedge,\oplus)$. As a consequence, when $(\mathcal{R},\oplus,\leqslant)$ is a dioid, a solution of our generalized dynamic programming equation (3.1) is also a solution of the algebraic shortest path problem (3.2). In that case, the shortest distance $b_v=\delta(v,d)$ is the meet of the resources q_P of all the v-d paths. The algebraic path problem theory enables to obtain stronger convergence results for the algorithms of the next chapter when $(\mathcal{R},\oplus,\leqslant)$ is a dioid. We also underline the fact that, when the lattice ordered monoid is not a dioid, then the solutions b_v of the generalized dynamic equation (3.1) are not necessarily the meet of all the v-d paths.

Furthermore, the lattice ordered monoid point of view enables to give a meaning to the dioid shortest distance (3.2) and to the dioid dynamic programming equation (3.3) when the dioid $(\mathcal{R},+,\times)$ is not bounded. Indeed, it suffices to complete the lower semi-lattice $(\mathcal{R},\leq_+)$ with a lower bound $-\infty$, and the Knaster-Tarski fixed point theorem in lattices ensures the existence of a solution to Equation (3.3), as we will see in Chapter 4.

Remark 3.3. A lattice ordered group $(\mathcal{R},\oplus,\leqslant)$ is a dioid as the commutativity of $\oplus$ implies that it is distributive with respect to the meet operator $\wedge$ of the lattice. Indeed it suffices to

apply (3.4) with $q \oplus b$, $\tilde{q} \oplus b$ and b^{-1}, and to right add b on both sides to obtain the reversed inequality. The vector space $\mathbb{R}^n$ endowed with its product sum is a lattice ordered group, and thus a dioid.

Remark 3.4. Kuntzmann [110] first introduced the term dioid, for double monoid, to design what is now called a semiring. Baccelli et al. [16] introduced its use to mean idempotent semiring. For Gondran and Minoux [86], a dioid is a semiring such that the preorder $q \preceq \tilde{q}$ if there exists b in $\mathscr{R}$ such that $q = \tilde{q} + b$ is an order. According to their definition, operator + is not necessarily idempotent. Nonetheless, if + is idempotent, $\preceq$ and $\leq_+$ coincide. Idempotent semiring are therefore idempotent dioid according to the terminology of [86]. As only idempotent semiring induces a lattice ordered monoid structure, in this dissertation, we stick to the terminolgy of Baccelli et al. [16] and call a dioid an idempotent semiring.

3.4 Modeling with lattice ordered monoid

3.4.1 Bibliographical remarks and path problems classification

Desrochers [53] first defined the terminology resource constrained shortest path problem in his Ph.D. dissertation in order to model the subproblem of a column generation approach to a bus driver scheduling problem. Some resource constraint shortest path problems had been considered before under another terminology [56, 57, 92]. The importance of the resource constrained shortest path problem comes from the fact that it naturally arises as the subproblem of many column generation approaches. It is specifically true in the context vehicle routing problems and crew scheduling problems. Irnich and Desaulniers [97] review the different types of resource constrained shortest path problems and the solution approaches in the literature. In their classification of the different types of problems, they outline five elements that enter in the definition of resource constrained shortest path problems.

The first element is the *type of constraints on the resources*. A constraint is an interval constraint if one component of the resource q is a real number and must belong to a given interval I. If we allow us to use intervals of the types $(-\infty, b]$, $[b, +\infty)$, or $[b, \tilde{b}]$, most constraints on the resource considered in the literature are interval constraints. An important case of interval constraints are the time-windows constraints [57], where a vertex must be visited within a time interval I. Time-windows constraints are frequent in the context of vehicle routing problems.

The second element in the definition of a path problem is the presence or the absence of *path structural constraints*. This type of constraints state that a path must be elementary [23] or simple to be feasible.

The third element is the presence or the absence of *time dependent quantities*. The subproblem of the column generation approach to the CREW PAIRING PROBLEM of Air France we deal with in Chapter 9 belongs to that family of problems. We therefore postpone the discussion on how to handle this type of constraints within the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework to Chapter 9.

The fourth element is the presence or the absence of *cycles* in digraph D. In the definition of the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM, we considered cyclic digraphs, which enables to cover all cases. We always mention when faster versions of the

algorithm exposed can be obtained for acyclic digraphs.

Finally, the last element is the *non-linearity* or the *stochasticity* of the resource extension functions. As no assumption is made on the monoid sum, we can use sum operators $\oplus$ that are highly non linear. Such cases are better explained on a practical example. We therefore refer the reader to the toy problem of the next section, and to Chapter 9, where highly non linear constraints are dealt with in the context of Air France CREW PAIRING PROBLEM. Chapter 5 is dedicated to the study of stochastic path problems.

In Section 3.4.3, we show that all these types of problems can be handled within the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework. As a consequence, the algorithms of the next chapters are valid for most resource constraint shortest path problems considered in the literature. Practically, we expect them to perform well in the absence of negative resource cycles.

Before giving the modeling techniques for the different types of constraints mentioned above, we start by showing how a toy problem can be modeled in the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework.

3.4.2 Example: a continent wide truck delivery

We now illustrate the modeling power of the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM on the example of a continent wide truck delivery defined in Chapter 1. Let o and d be two distant vertices on the digraph of the roads of a continent. Suppose that a truck driver must transport a good from o to d. For safety reasons, the driver has to respect a driving time constraint: he should not drive more than a certain amount of time T every day. The objective is to find an itinerary from o to d of minimum cost among those satisfying the driving time constraint.

We therefore model the problem using a digraph D whose vertices are the cities and with two types of arcs. *Day arcs* $a = (u, v)$, whose origin is different from their destination, correspond to the road of the network. *Night arcs* a_v that are loops at vertices v corresponding to places where the driver can spend a night. A night arc a_v has a *night resource* $q^{\mathrm{n}}_{a_v} = c_{a_v}$, where c_{a_v} is the cost of spending a night in the vertex v of a_v, modeling for instance hotel fares. A day arc has a *day resource* $q^{\mathrm{d}}_a = (c_a, d_a)$, where c_a corresponds to the cost of driving along a, modeling for instance fuel costs and tolls, and d_a is the time needed to drive along a. Finally, consider the path in blue on Figure 3.1. This path spans several days. We therefore introduce *multiple day resources* $q^{\mathrm{m}} = (d^{\mathrm{f}}, c, d^{\mathrm{l}})$, where d^{f} corresponds to the duration of the first day, d^{l} to the one of the last day, and c to the total cost. The *set of resources* $\mathscr{R}$ is the union of the set of day resources, the set of night resources, and the set of multiple day resources.

In order to define the sum operator on this set of resources, it suffices to consider if the resulting path spans multiple day or not, and to test if the driving duration constraint is violated. Therefore, we introduce the penalty function f defined as follows.

$$f(d) = \begin{cases} 0 & \text{if } d \leq T, \\ \infty & \text{otherwise.} \end{cases}$$

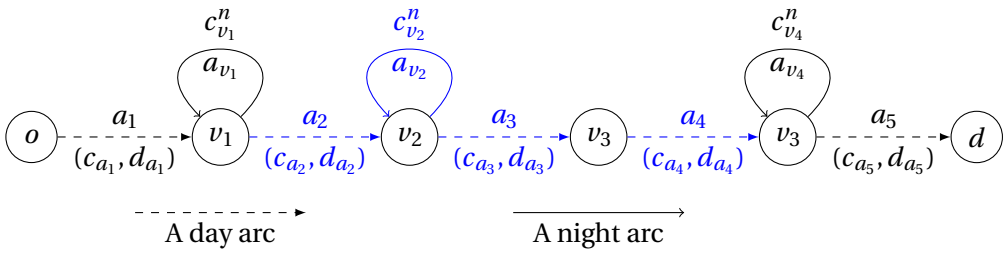


Figure 3.1 – A path P with one night arc is illustrated in blue.

We know that a subpath corresponds to a complete day when it is preceded and followed by a night arc. The following equations define $\oplus$ by disjunction of cases, using f only when a day is complete. Note that a sum of day resources gives a day resource. Any other combination of resource gives a multiple day resource.

$$\begin{array}{rcrcl}
(c) & \oplus & \tilde{c} & = & (0, c+\tilde{c}, 0) \\
(c) & \oplus & (\tilde{c}, \tilde{d}) & = & (0, c+\tilde{c}, \tilde{d}) \\
(c) & \oplus & (\tilde{d}^{\mathrm{f}}, \tilde{c}, \tilde{d}^{\mathrm{l}}) & = & (0, c+f(\tilde{d}^{\mathrm{f}})+\tilde{c}, \tilde{d}^{\mathrm{l}}) \\
(c,d) & \oplus & \tilde{c} & = & (d, c+\tilde{c}, 0) \\
(c,d) & \oplus & (\tilde{c}, \tilde{d}) & = & (c+\tilde{c}, d+\tilde{d}) \\
(c,d) & \oplus & (\tilde{d}^{\mathrm{f}}, \tilde{c}, \tilde{d}^{\mathrm{l}}) & = & (d+\tilde{d}^{\mathrm{f}}, c+\tilde{c}, \tilde{d}^{\mathrm{l}}) \\
(d^{\mathrm{f}}, c, d^{\mathrm{l}}) & \oplus & \tilde{c} & = & (d^{\mathrm{f}}, c+f(\tilde{d}^{\mathrm{l}})+\tilde{c}, 0) \\
(d^{\mathrm{f}}, c, d^{\mathrm{l}}) & \oplus & (\tilde{c}, \tilde{d}) & = & (d^{\mathrm{f}}, c+\tilde{c}, d^{\mathrm{l}}+\tilde{d}) \\
(d^{\mathrm{f}}, c, d^{\mathrm{l}}) & \oplus & (\tilde{d}^{\mathrm{f}}, \tilde{c}, \tilde{d}^{\mathrm{l}}) & = & (d^{\mathrm{f}}, c+f(d^{\mathrm{l}}+\tilde{d}^{\mathrm{f}})+\tilde{c}, \tilde{d}^{\mathrm{l}})
\end{array} \tag{3.5}$$

As defined in (3.5), operator $\oplus$ is associative but not commutative. We define $\leqslant$ as an extension of the component by component order.

$$\begin{array}{rcllllll}
(c) & \leqslant & (\tilde{c}) & \text{if } c \leq \tilde{c} & & & & \\
(c) & \leqslant & (\tilde{c}, \tilde{d}) & \text{if } c \leq \tilde{c} & & & & \\
(c) & \leqslant & (\tilde{d}^{\mathrm{f}}, \tilde{c}, \tilde{d}^{\mathrm{l}}) & \text{if } c \leq \tilde{c} & & & & \\
(c,d) & \leqslant & (\tilde{c}) & \text{if } c \leq \tilde{c} & \text{and} & d & = & 0 \\
(c,d) & \leqslant & (\tilde{c}, \tilde{d}) & \text{if } c \leq \tilde{c} & \text{and} & d & \leq & \tilde{d} \\
(c,d) & \leqslant & (\tilde{d}^{\mathrm{f}}, \tilde{c}, \tilde{d}^{\mathrm{l}}) & \text{if } c \leq \tilde{c} & \text{and} & d & \leq & \min(\tilde{d}^{\mathrm{f}}, \tilde{d}^{\mathrm{l}}) \\
(d^{\mathrm{f}}, c, d^{\mathrm{l}}) & \leqslant & (\tilde{c}) & \text{if } c \leq \tilde{c} & \text{and} & \max(d^{\mathrm{f}}, d^{\mathrm{l}}) & = & 0 \\
(d^{\mathrm{f}}, c, d^{\mathrm{l}}) & \leqslant & (\tilde{c}, \tilde{d}) & \text{if } c \leq \tilde{c} & \text{and} & \max(d^{\mathrm{f}}, d^{\mathrm{l}}) & \leq & \tilde{d} \\
(d^{\mathrm{f}}, c, d^{\mathrm{l}}) & \leqslant & (\tilde{c}) & \text{if } c \leq \tilde{c} & \text{and} & d^{\mathrm{f}} \leq \tilde{d}^{\mathrm{f}} & \text{and} & d^{\mathrm{l}} \leq \tilde{d}^{\mathrm{l}}
\end{array} \tag{3.6}$$

With this definition of $\leqslant$, the monoid $(\mathscr{R}, \oplus, \leqslant)$ is a lattice ordered monoid. Finally, we define the cost function $c_{\mathscr{R}}$ and feasibility function $\rho_{\mathscr{R}}$ as follows.

$$\begin{array}{rcl} c_{\mathscr{R}}((c)) & = & c \\ c_{\mathscr{R}}((c,d)) & = & c + f(d) \\ c_{\mathscr{R}}\left((d^{\mathrm{f}},c,d^{\mathrm{l}})\right) & = & f(d^{\mathrm{f}}) + c + f(d^{\mathrm{l}}) \end{array} \quad \text{and} \quad \begin{array}{rcll} \rho_{\mathscr{R}}((c)) & = & 1 & \text{if } c = \infty \\ \rho_{\mathscr{R}}((c,d)) & = & 1 & \text{if } c + f(d) = \infty \\ \rho_{\mathscr{R}}\left((d^{\mathrm{f}},c,d^{\mathrm{l}})\right) & = & 1 & \text{if } f(d^{\mathrm{f}}) + c + f(d^{\mathrm{l}}) = \infty \end{array}$$

We index the cost and the feasibility function by $.\_{\mathscr{R}}$ to avoid the confusion with the real number c appearing inside resources of $\mathscr{R}$. An optimal solution of the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM on D with cost $c_{\mathscr{R}}$, objective $\rho_{\mathscr{R}}$, and resources in $(\mathscr{R}, \oplus, \leqslant)$ defined as above is an optimal solution of the continent wide delivery problem.

Modeling the continent wide delivery problem in the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework is made technical by the associativity property required on $\oplus$. Indeed, in the absence of such a property, the problem can be modeled using only (c,d), as we have shown in Chapter 1. But associativity is a key element in our framework, as it is required to generalize A* and the dynamic programming algorithms to resource constrained shortest path problems.

3.4.3 Modeling techniques

We now give techniques that enable to model the different resource constrained shortest path problems mentioned in Section 3.4.1

Additional constraint and lattice ordered monoid product

The *product* of two lattices ordered monoid is the product set endowed with the product sum and the product order. The product of two lattices order monoids is a lattice order monoid. This property enables to model easily new constraints. Suppose for instance that in our continent wide delivery problem, the itinerary must in addition not be longer than a given distance L. This new constraint is easily modeled with a positive real resource ℓ in $(\mathbb{R}_+, +, \leq)$, and with the infeasibility function $\rho_\ell(\ell) = 1$ if $\ell > L$. The constrained continent wide delivery problem can then be modeled using the product $(\mathscr{R}, \oplus, \leqslant) \times (\mathbb{R}, +, \leq)$ of the lattice ordered monoid defined in the previous section with $(\mathbb{R}_+, +, \leq)$, the cost function $c_{\mathscr{R}}$, and the infeasibility functions $(q,\ell) \mapsto \max\left(\rho_{\mathscr{R}}(q), \rho_\ell(\ell)\right)$.

This example also shows how to deal with interval constraints of the type $\ell \in (-\infty, b]$. The other types of interval constraints are considered in the next section.

Antitone cost function and interval constraints

Suppose that for any given reason, the itinerary should be non-smaller than a given number of kilometers. Such a constraint can be dealt with using a resource ℓ' in $(\mathbb{R}_+, +, \leqslant')$, where $\leqslant'$ is the reversed order $\ell' \leqslant' \tilde{\ell}'$ if $\ell' \geq \tilde{\ell}'$, and we again obtain a structure of lattice ordered monoid. Indeed, an antitone cost function is isotone for the reversed order. This shows how to deal with interval constraints of the type $\ell \in [b, +\infty)$.

The same idea enables to deal with interval constraints of the type $\ell \in [b, \tilde{b}]$. Such an *interval*

constraint can be dealt with by "splitting" the resource into two components ℓ and $\tilde{\ell}$. We can use the lattice ordered monoid $\mathbb{R}^2$ endowed with the usual sum and the order

$$(\ell, \tilde{\ell}) \leqslant (\ell', \tilde{\ell}') \quad \text{if} \quad \ell \leq \ell' \quad \text{and} \quad \tilde{\ell} \geq \tilde{\ell}'.$$

The infeasibility function $\rho\left((\ell, \tilde{\ell})\right) = \begin{cases} 0 & \text{if } \ell \leq b \text{ and } \tilde{\ell} \geq \tilde{b}, \\ 1 & \text{otherwise,} \end{cases}$ is isotone and models the desired interval constraint.

Subset lattices and elementary paths

The solution of the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM can be enforced to be elementary by using the powerset of the set of vertices V as resource set. The sum of two resources is the union of the two set if the are disjoint, and ∞ otherwise, and $q \leqslant \tilde{q}$ if $q \subseteq \tilde{q}$. Similarly, it can be enforced to be simple by using the powerset of A.

4 Algorithms for path problems with resources in an ordered monoid

This chapter extends the STANDARD SHORTEST PATH PROBLEM algorithms to the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM with resources in an ordered monoid, whose definition has been given in the introduction of Part I. There are two main branches of algorithms for the STANDARD SHORTEST PATH PROBLEM.

First, the *enumeration algorithms*, such as A*, which enumerate all candidate paths using bounds to discard partial paths. The use of bounds to discard partial paths is enabled by Property 3.2. Property 3.3, which extends Property 3.2 to resources in an ordered monoid, enables to generalize A* to the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM.

The second branch is composed of *polynomial algorithms*, such as Ford-Bellman dynamic programming algorithm, or Dijkstra algorithm. As the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM is $\mathcal{NP}$-complete, we cannot expect to solve it using a generalization of Dijkstra or Ford-Bellman algorithm. But in the lattice ordered monoid framework, the dynamic programming equation can be generalized to (3.1). In this chapter, we show that this generalized equation gives lower bounds on paths resources, and that it can be solved by generalizations of Dijkstra and Ford-Bellman algorithms.

From a practical point of view, this chapter can be summed-up as follows: the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM *with resources in a lattice ordered monoid can be solved efficiently* in two steps. First, a polynomial algorithm enables to compute bound on paths resources by solving the generalized dynamic programming equation. Second, these bounds can be used in an enumeration algorithm to find an optimal path.

This solution strategy is tested numerically on standard resource constrained shortest path problems in this chapter, on stochastic path problems in Chapters 5 and 6, and in the context of column generation for an airline operation problem in Chapter 9.

The chapter is organized as follows:

- Section 4.1 introduces the enumeration algorithms of type A* for the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. In resource constrained shortest path literature, these algorithms are called *label algorithms*.
- Section 4.2 introduces the generalization of the dynamic programming equation to the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM, and generalizations of

Ford-Bellman and Dijkstra algorithms to solve it in polynomial time.
- Section 4.3 provides numerical results on resource constrained shortest path problems. It also contains technical tips to accelerate the algorithms introduced in the previous sections.
- Finally, Section 4.4 contains bibliographical remarks.

4.1 Enumeration algorithms

In this section, we detail the enumeration algorithms for the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. In the STANDARD SHORTEST PATH PROBLEM terminology, these algorithms are called A^* algorithms [93]. In the STANDARD RESOURCE CONSTRAINED SHORTEST PATH PROBLEM terminology, these algorithms are called label algorithms.

4.1.1 Generic algorithm

In this section, we give three algorithms for the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM: the generalized A*, the label dominance, and the label correcting algorithms. These three algorithms share the same structure. They enumerate all the paths in the graph using tests to discard partial paths. They differ only by the tests used and the processing order of the paths. We therefore give a generic algorithm, and define the algorithms used in practice as specializations of this generic algorithm. Later in this section, we sometimes call optimal path an optimal solution of the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM.

We now describe the *generic algorithm*. A list L of partial paths P, and an upper bound c_{od}^{UB} on the cost of an optimal solution are maintained. Initially, L contains the empty path at the origin o and $c_{od}^{UB} = +\infty$. While L is not empty, the following operations are repeated.

1. Extract a path P *of minimum key* from L. Let v be the destination of P.
2. If $v = d$ and P is feasible and better than the current solution, i.e. $\rho(q_P) = 0$, and $c(q_P) < c_{od}^{UB}$, then update c_{od}^{UB} to $c(q_P)$.
3. Else *test if P must be extended*, and if yes, extend P: for each arc a outgoing from v, add $P + a$ to L.

We prove later in the section that, under mild assumptions, at the end of the algorithm, c_{od}^{UB} is the cost of an optimal path. The algorithms differ by the choice of the minimum key in Step 1, and by the choice of the test before extension in Step 3.

The usual A* algorithm uses bounds on path resources to discard paths. Besides, as we show in the next section, the lattice ordered monoid structure in the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework enables to design polynomial algorithms to build bounds on paths resources. Therefore, for each vertex v, we suppose to have a lower bound b_v on the resource q_P of any v-d path. The lower-bound test relies on the following lemma.

Lemma 4.1. *If an o-v path P is a subpath of an optimal path, then $\rho(q_P \oplus b_v) = 0$ and $c(q_P \oplus b_v) \leqslant c_{od}^{UB}$.*

Algorithm	Test	Key	Pre-processing structures	Maintained structures
Generalized A*	(Low)	$c(q_P \oplus b_v)$	b_v	L, c^{UB}_{od}
Label dominance	(Dom)	$c(q_P)$	—	L, c^{UB}_{od}, M_v
Label correcting	(Dom), (Low)	$c(q_P \oplus b_v)$	b_v	L, c^{UB}_{od}, M_v

Table 4.1 – MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM algorithms.

Proof. For any v-d path $\tilde{P}$ starting by P, we have $q_P \oplus b_v \leqslant q_{\tilde{P}}$. The result then follows from the monotonicity of ρ and c. □

Given an o-v path P, the *lower bound test* can be expressed as follows.

(Low) If $\rho(q_P \oplus b_v) = 0$ and $c(q_P \oplus b_v) \leqslant c^{UB}_{od}$, then extend P.

The *generalized A** is obtained from the generic algorithm by using the lower bound $c(q_P \oplus b_v)$ as key in Step 1, and extending a path P in Step 3 only if it satisfies the lower bound test (Low).

The usual label algorithms for resource constrained shortest path problem relies on the notion of dominance. An o-v path P *dominates* an o-v path $\tilde{P}$ if $q_P \leqslant q_{\tilde{P}}$. The *dominance test* relies on the following lemma.

Lemma 4.2. *There exists an optimal solution of the* MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM *whose subpaths are all non-dominated.*

As a consequence, a list M_v of non-dominated o-v paths is maintained for each vertex v, and the dominance test can be expressed as follows.

(Dom) If *no path in M_v dominates P*, then if P is feasible, remove from M_v all paths dominated by P, add P to M_v, and extend P.

The *label dominance* algorithm is obtained from the generic algorithm by using the partial path cost $\rho(c_P)$ as key in Step 1, and extending an o-v path P in Step 3 only if it satisfies the dominance test (Dom). The *label correcting* algorithm is obtained from the generic algorithm by using the lower bounds $c(q_P \oplus b_v)$ as key in Step 1, and extending an o-v path P in Step 3 only if it satisfies both the lower bound test (Low) and the dominance test (Dom). We note that the label dominance and the label correcting algorithm require to define the lists M_v of non dominated paths, and to maintain them using the operations mentioned in dominance test (Dom). Table 4.1 sums up the properties and the structures maintained by the different algorithms.

Both our label dominance and our label correcting algorithms are referred as label correcting algorithms in the literature. When bounds are available, the label correcting algorithm outperforms the label dominance algorithm. Nonetheless, to the best of our knowledge, there is no standard procedure to build bounds b_v for non-linear or stochastic problems in the literature. The main contribution of the lattice ordered monoid framework is that it enables to design a standard procedure to build lower bounds. Therefore, when we want to benchmark our approach with existing ones on our instances of non-linear or stochastic problems,

we compare our generalized A* and our label correcting algorithm to the label dominance algorithms. When we consider the linear resource constrained shortest path problem, i.e. the resource set $(\mathscr{R}, \oplus, \leqslant)$ is $\mathbb{R}^n$ endowed with the product sum and order, the bounding procedure of the next section leads to the same bounds as those used in the literature [61]. Nonetheless, when considering difficult instances, we can improve the quality of theses bounds using the techniques exposed in Chapter 6.

4.1.2 Convergence of the algorithms

We now prove the convergence of the enumeration algorithms of the previous section under mild assumptions.

Generalized A* algorithm

The following assumptions will be used to define some settings under which ones the generalized A* algorithm converges:

$$\text{For all } a, b < \bigvee \mathscr{R} \text{ and } q > 0 \text{ in } \mathscr{R}, \text{ there exists an } n \in \mathbb{Z}_+ \text{ such that } nq \oplus a \not\leqslant b. \tag{4.1}$$

$$\text{The set } \rho^{-1}(0) \text{ is upper-bounded by a } q_M < \bigvee \mathscr{R}. \tag{4.2}$$

$$\text{There exists a feasible } o\text{-}d \text{ path } P \text{ such that } c^{-1}\left(\left]-\infty, c(q_P)\right]\right) \cap \rho^{-1}(0) \text{ is upper-bounded by a } q_M < \bigvee \mathscr{R}. \tag{4.3}$$

In Assumptions (4.1) to (4.3), $\bigvee \mathscr{R}$ may be in the completion of $\mathscr{R}$.

Remark 4.1. An ordered monoid $\mathscr{R}$ is Archimedean if, for each resource b and $q > 0$ in $\mathscr{R}$, there exists an integer n such that $nq \geqslant b$. Assumption (4.1) is related but much weaker than the Archimedean property. It is satisfied by all the lattice ordered monoids considered in this dissertation. $\mathbb{R}^2$ endowed with its product sum and order is an example of lattice ordered monoid that is not Archimedean which satisfies assumption (4.1).

Theorem 4.3. *Suppose that at least one of the following conditions is satisfied.*

(a) D is acyclic.
(b) Assumption (4.1) *is satisfied, at least one of the assumptions* (4.2) *and* (4.3) *is satisfied, q_a is positive for each arc a, and $b_v \geqslant 0$ for each vertex v.*
(c) Assumption (4.1) *is satisfied, at least one of the assumptions* (4.2) *and* (4.3) *is satisfied, $\oplus$ is commutative, and $\bigoplus_{a \in C} q_a$ is positive for any cycle C in D.*

Then the generalized A algorithm converges after a finite number of iterations, and at the end, if c_{od}^{UB} is finite, then it is the cost of an optimal solution of the* Monoid Resource Constrained Shortest Path Problem. *Otherwise, the problem admits no feasible solutions.*

Note that $\rho^{-1}(0)$ is the *set of feasible resources*. Case (c) is notably satisfied when $(\mathscr{R}, \oplus, \leqslant)$ is an Archimedean lattice ordered group and cycle resources are positive. Indeed, Theorem 10.19 in [28] ensures that any Archimedean lattice ordered group is commutative, and Assumption (4.1) is a consequence of the Archimedean property in lattice ordered groups. Besides, the generalized A* algorithm can still be used even if none of the conditions (a), (b), and (c) are

satisfied: see Section 4.1.3. We also note that in case (b), the hypothesis that $b_v \geqslant 0$ is not restrictive, as $q_a > 0$ implies that $q_P \geqslant 0$ for all paths P.

Given two paths P and Q such that P ends in the origin of Q, we denote $P+Q$ the union of paths P and Q, and q_{P+Q} its resource. Theorem 4.3 relies on the following lemmas.

Lemma 4.4. *Let P be an o-d path satisfying $\rho(q_P) = 0$. Then at a given step of the generalized A^* algorithm, at least one of the following statements is satisfied:*

- *there is a subpath P' of P in L,*
- $c_{od}^{UB} \leq c(q_P)$.

Note that P' can be equal to P.

Proof. We start with preliminary results. Paths are added to L only due to extension of paths in Step 3. A path Q can therefore be in L only if its subpaths have been considered, removed from L, and extended by the algorithm. Thus, at a given step of the generalized A* algorithm, for each path Q with origin o, exactly one of the following statements is satisfied:

- Q has been considered by the generalized A* algorithm,
- a subpath Q' of Q is in L,
- a strict subpath Q' of Q has not been extended by the algorithm when considered.

Besides, if a feasible o-d path Q has already been considered, Step 2 of the algorithm implies $c_{od}^{UB} \leq c(q_Q)$.

We now prove Lemma 4.4. Suppose that none of the statements of Lemma 4.4 are satisfied. As P is a feasible o-d path, the two results above implies that a subpath P' of P has not been extended by the algorithm when considered. Let P' be this subpath, and v' be its destination. As c_{od}^{UB} decreases along the algorithm, the hypothesis implies that $c_{od}^{UB} > c(q_P)$ when P' is considered. As b'_v is a lower bound on the resource of all v'-d path, we have $q_{P'} \oplus b'_v \leqslant q_P$. By monotonicity of ρ and feasibility of P, we have $\rho(q_{P'} \oplus b_{v'}) = 0$. By monotonicity of c, we have $c(q_{P'} \oplus b_{v'}) \leqslant c(q_P) < c_{od}^{UB}$ when P' is considered. The two last inequalities imply that P' satisfies the lower bound test, which contradicts the fact that P' has not been extended, and we obtain the lemma. □

Lemma 4.5. *Under Assumption* (4.2) *or under Assumption* (4.3)*, if an o-v path Q such that $q_Q \oplus b_v \not\leqslant q_M$ is considered by the algorithm, it does not satisfy the lower bound test (Low).*

Proof. Under Assumption (4.2), we have $\rho(q_Q \oplus b_v) = 1$ by definition of q_M, and Q does not satisfy the lower bound test.

Suppose now that Assumption (4.3) is satisfied, and that Q is considered by the algorithm. If $\rho(q_Q \oplus b_v) = 1$, path Q does not satisfy the lower bound test and we obtain the result. Otherwise, Q is such that $c(q_Q \oplus b_v) > c(q_P)$. We now prove that, under this second hypothesis, we have $c_{od}^{UB} < c(q_Q \oplus b_v)$ when Q is considered by the algorithm, which then implies that Q does not satisfy the lower bound test when it is considered. Suppose that it is not the case. We place ourselves at the step when Q is considered. As a consequence, Q minimizes $c(q_Q \oplus b_v)$

among the paths in L. Let P and q_M be as in Assumption (4.3). By definition of P and q_M, the hypothesis $q_Q \oplus b_v \not\leqslant q_M$ implies $c(q_P) < c(q_Q \oplus b_v) \leq c_{od}^{UB}$ when Q is considered. Lemma 4.4 implies that there is a subpath P' of P in L when Q is considered. By monotonicity of c we have $c(q_{P'} \oplus b_{v'}) \leqslant c(q_P) < c(q_Q \oplus b_v)$. This contradicts the fact that Q minimizes $c(q_Q \oplus b_v)$ among the paths in L, and gives the lemma. □

Lemma 4.6. *Suppose that* (a), (b), *or* (c) *is satisfied, then there is a finite number of paths in D that satisfy the lower-bound test.*

Proof. In case (a), graph D is acyclic and there is a finite number of paths. We now suppose that we are in case (b) or (c): let q_M be as in one of the assumptions satisfied among Assumptions (4.2) and (4.3). Lemma 4.5 ensures that only o-v paths P such that $q_P \oplus b_v \leqslant q_M$ can satisfy the lower bound test. We show the lemma by proving that there is a finite number of such paths.

As we are in case (b) or (c), any elementary cycle C in D satisfies $q_C > 0$. Thus, given an elementary cycle C, an elementary path Q, and a vertex v in P, as the resource of C is positive, Assumption (4.1) implies that there exists an integer $n_{C,Q,v}$ such that $(n_{C,Q,v}q_C) \oplus q_Q \oplus b_v \not\leqslant q_M$. As there is a finite number of elementary paths and a finite number of elementary cycles in D, we can define n to be an integer such that

$$(nq_C) \oplus q_Q \oplus b_v \not\leqslant q_M \tag{4.4}$$

for any elementary cycle C, elementary path Q, and bounds b_v. Let n_c be the number of elementary cycles in D.

The proof of case (b) relies on the following well-known result.

Any path in a directed graph can be decomposed in a sequence of elementary paths and elementary cycles.

Suppose that we are in case (b), let P be a path with at least $2nn_c|V|$ arcs and consider such a decomposition. As an elementary path or an elementary cycle contains at most $|V|$ arcs, this decomposition contains at least nn_c cycles, and thus at least n copies of a given cycle C_0. As the resource of all arcs are positive by hypothesis of case (b), we have $q_P \geqslant nq_{C_0}$. We therefore have $q_P \oplus b_v \geqslant nq_{C_0} \oplus b_v$, and by applying Equation (4.4) with the empty path as Q, we obtain $q_P \oplus b_v \not\leqslant q_M$. As a consequence, only o-v paths P with fewer than $2nn_c|V|$ arcs can satisfy $q_P \oplus b_v \leqslant q_m$, and Lemma 4.5 ensures that there is a finite number of paths that satisfy the lower bound test in case (b).

We now consider case (c). As the monoid is supposed to be commutative, the resource of a path does not depend on the order of the sequence of its arcs, but only on its multiset of arcs. The proof of case (c) relies on the following well-known result.

The multiset of arcs of any path in a directed graph can be decomposed in the union of the sets of arcs of an elementary path and of several elementary cycles.

Suppose that we are in case (c), let P be a path with at least $n|V|(n_c+1)$ arcs and consider such a decomposition where Q denotes the elementary path. As by hypothesis of case (c), the

operator $\oplus$ is commutative, the resource of P is entirely defined by the resource of its arcs, independently of their order. As an elementary path or an elementary cycle contains at most $|V|$ arcs, this decomposition contains at least nn_c cycles, and thus at least n copies of a given cycle C_0. As, by hypothesis of case (c), all cycles are positive, we have $q_P \oplus b_v \geqslant nq_{C_0} \oplus q_Q \oplus b_v$. Equation (4.4) ensures that only paths with less than $n|V|(n_c+1)$ arcs can satisfy $q_P \oplus b_v \leqslant q_m$, and Lemma 4.5 ensures that there is a finite number of paths that satisfy the lower bound test in case (c). □

Proof of Theorem 4.3. As any path inserted in L is the extension of a previously considered path, a given path is considered at most once by the algorithm. Thus, Lemma 4.6 implies the convergence after a finite number of iterations as only paths satisfying the lower bound test can be extended by the algorithm.

At the end of the algorithm, list L is empty and Lemma 4.4 ensures that c_{od}^{UB} is a lower bound on the cost of any o-d path satisfying $\rho(q_P) = 0$. Besides, Step 2 of the algorithm ensures that if c_{od}^{UB} is different from $+\infty$, then there is a path P such that $c(q_P) = c_{od}^{UB}$ and $\rho(q_P) = 0$. This concludes the proof. □

Label correcting and label dominance algorithms

Theorem 4.7. *Suppose that all cycles in D are positive, then the label correcting algorithm converges after a finite number of iterations, and at the end, if c_{od}^{UB} is finite, then c_{od}^{UB} is the cost of a non-dominated optimal solution of the* MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. *Otherwise, the problem admits no feasible solutions.*

Theorem 4.8. *Suppose that all cycles in D are positive, then the label dominance algorithm converges after a finite number of iterations, and at the end, if c_{od}^{UB} is finite, then c_{od}^{UB} is the cost of a non-dominated optimal solution of the* MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. *Otherwise, the problem admits no feasible solutions.*

Remark that the label correcting and the label dominance algorithms converge under weaker conditions than those required for the convergence of the generalized A* algorithm in Theorem 4.3.

Lemma 4.9. *If all cycles in D are positive, then if a path P containing a cycle is considered by the label dominance or the label correcting algorithms, then it does not satisfy the dominance test (dom).*

Proof. As paths in L are added by extension of paths previously in L, we only need to prove the result for paths ending by a cycle. Let P be such a path, let $Q + C$ be its decomposition in a path and a cycle, and let v be the common destination vertex of P and Q. By hypothesis, we have $q_C \geqslant 0$. As a consequence, $q_P = q_Q \oplus q_C \geqslant q_Q$. As P is processed, all its subpaths have been extended by the algorithm, and thus path Q has necessarily been extended. This implies that either Q or a path Q' such that $q_{Q'} < q_Q \leq q_P$ is in M_v, and thus P is dominated by a path in M_v and is therefore not extended. □

Proof of Theorem 4.7. As any path inserted in L is the extension of a previously considered path, a given path is considered at most once by the algorithm. Thus, as there is only a finite number of acyclic paths in a graph, Lemma 4.9 ensures that the algorithm converges after a finite number of iterations.

Step (b) of the algorithm ensures that c_{od}^{UB} is greater or equal to the cost of an optimal solution of the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. We now prove that at the end of the algorithm, c_{od}^{UB} is equal to the cost of an optimal solution. Indeed, suppose that it is not the case. Let P be an optimal solution. Let $\mathscr{L}$ be the set of all paths that have been contained in L along the algorithm, and for each vertex v in P, let P_{ov} be the subpath of P starting v. Let v be the last vertex of P such that there is an o-v path Q in $\mathscr{L}$ with $q_Q \leqslant q_{P_{ov}}$ and is therefore in $\mathscr{L}$. It exists because the empty path P_{oo} is added to L at the beginning of the algorithm. Besides, it is not equal to d, as otherwise we would have $c_{od}^{UB} \leq c(q_P)$. Among the o-v paths dominating P_{ov} in L, let Q be the first one generated by the algorithm. By definition of Q and as any path that has been in M_v along the algorithm is in $\mathscr{L}$, there is no path dominating Q in M_v when Q is processed. As c_{od}^{UB} decreases along the algorithm and by hypothesis, when Q is processed we have $\rho(q_Q + b_v) \leq \rho(q_{P_{ov}} + b_v) \leq \rho(q_P) = 0$ and $c(q_Q + b_v) \leq c(q_{P_{ov}} + b_v) \leq c(q_P) < c_{od}^{UB}$. As a consequence, Q has been extended, and $Q + (v, w)$ is in $\mathscr{L}$, where w be the vertex after v in P. Besides, we have $q_{Q+(v,w)} = q_Q \oplus q_{(v,w)} \leqslant q_{P_{ov}} \oplus q_{(v,w)} = q_{P_{ow}}$, which contradicts the definition of v. □

Proof of Theorem 4.8. As for Theorem 4.7, Lemma 4.9 ensures that the algorithm converges after a finite number of iterations. The fact that the set of paths extended by the label dominance algorithm contains the set of path extended by the label correcting algorithm ensures that if c_{od}^{UB} is finite at the end of the algorithm, then c_{od}^{UB} is the cost of a non-dominated optimal solution. □

4.1.3 Dealing with non positive cycles, non commutativity or absence of the Archimedean property

In this section, we show that, even if the assumptions of Theorems 4.3 and 4.7 are not satisfied, both the generalized A* and the label correcting algorithms converge after a finite number of iterations on an extended resource monoid. It provides a way to deal with situations where the hypotheses of Theorems 4.3 and 4.7 are not satisfied. Let B_n be the set $\{0,1\}^n \cup \{\infty\}$ where ∞ is an "infeasibility" element. The law $\oplus$ on B_n is the addition operator on $\mathbb{Z}_+^n$ such that if $q_1 + q_2 \notin \{0,1\}^n$, then $q_1 \oplus q_2 = \infty$. Consider the extended resource set $\widetilde{\mathscr{R}} = \mathscr{R} \times B_{|V|}$. We associate to each arc $a = (v_1, v_2)$ an extended resource $\widetilde{q}_a = (q_a, e_{v_2})$ where e_v is the vector composed of 0 in all positions except the index of v. The new resource $\widetilde{q}_P \in \widetilde{\mathscr{R}}$ of path P is an element on $\mathscr{R} \times B_{|V|}$: $\tilde{q}_P = (q_P, q_P^B)$. We define the extended infeasibility function $\widetilde{\rho}$ on $\widetilde{\mathscr{R}}$ as the mapping $(q_P, q_P^B) \mapsto \max(\rho(q_P), \mathbf{1}_\infty(q_P^B))$ where $\mathbf{1}_\infty$ is the indicator function of $\{\infty\}$. Finally, we extend the cost function to the mapping $\widetilde{c}\colon (q_P, q_P^B) \mapsto c(q_P)$. The following theorem proves the convergence of the enumeration algorithms with resources in $\widetilde{\mathscr{R}}$ under the general condition that $\mathscr{R}$ is an ordered monoid.

Proposition 4.10. *The generalized A*, label dominance, and label correcting algorithms on* $\widetilde{\mathscr{R}}$

with infeasibility function $\widetilde{\rho}$ and cost function $\widetilde{c}$ converge after a finite number of iterations on $\mathscr{R}^*$, and at the end, if c_{od}^{UB} is finite, then it is the cost of a feasible elementary path of minimum cost. Otherwise the problem admits no feasible solutions.

Proof. The definition of $\widetilde{q}_a$ for each arc A ensures that, given a path P, we have $q_P^B = i$ if and only if P contains a cycle. As a consequence, we have

$$\widetilde{\rho}(\widetilde{q}_P) = \begin{cases} \rho(q_P) & \text{if } P \text{ is elementary,} \\ 1 & \text{otherwise.} \end{cases}$$

As a consequence, any non elementary path is infeasible. The fact that there is a finite number of elementary paths gives the convergence after a finite number of iterations. The infeasibility of non-elementary paths and the equality $\widetilde{c}(\widetilde{q}_P) = c(q_P)$ ensure that at the end, c_{od}^{UB} is the resource of an optimal elementary path. □

4.2 Extended dynamic programming in lattice ordered monoids

The aim of this section is to provide a standard method to compute good bounds b_v on the resource q_P of any v-d path P when $(\mathscr{R}, \oplus, \leqslant)$ is a lattice ordered monoid. These bounds happen to be tight when $(\mathscr{R}, \oplus, \leqslant)$ is a dioid. This method is an extension of the dynamic programming algorithms for the standard shortest path problem. Combined with the algorithms of Section 4.1, it provides a standard methods to solve the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM problem.

In all the convergence theorems of the enumeration algorithms in Section 4.1.2, we suppose that $q_C \geqslant 0$ for all cycles C. Under this hypothesis, a lower-bound b_v on the resources of all the elementary v-d paths is a lower-bound on the resources of all the paths. And without this hypothesis, we have to use the technique of Section 4.1.3 and restrict ourselves to elementary paths. In all these cases, we can restrict ourselves to lower bounds b_v on the resources of elementary v-d paths.

Given a vertex $v \in V$, let b_v^{opt} be defined as $b_v^{\text{opt}} = \bigwedge_{P \in \mathscr{P}_{vd}} q_P$, where $\mathscr{P}_{vd}$ is the set of all the elementary v-d paths. The resource b_v^{opt} is well defined as there is a finite number of elementary paths. Resource b_v^{opt} is the best lower bound on all the elementary v-d paths: indeed, $b \leqslant q_P$ for all elementary v-d paths P implies $b \leqslant b_v^{\text{opt}}$. However, the following proposition shows that computing b_v^{opt} is difficult even on fairly simple lattice ordered monoids.

Proposition 4.11. *Unless $\mathscr{P} = \mathscr{NP}$, there is no polynomial algorithm independent of $\mathscr{R}$ that enables to compute b_v^{opt} even when restricted to a commutative monoid with positive resources.*

Proof. Consider the set $\mathscr{R} = [0,1]^2 \cup \{+\infty\}$. We endow it with the partial order

$$q \leqslant +\infty, \ \forall q \quad \text{and} \quad (q^1, q^2) \leqslant (\tilde{q}^1, \tilde{q}^2) \text{ if } q^i \leq \tilde{q}^i, \quad \text{for } i = 1,2, \tag{4.5}$$

and the sum operator

$$\left| \begin{array}{l} q \oplus +\infty = +\infty \oplus q = +\infty, \ \forall q \quad \text{and} \\ (q^1, q^2) \oplus (\tilde{q}^1, \tilde{q}^2) = \left\{ \begin{array}{ll} (q^1 + \tilde{q}^1, q^2 + \tilde{q}^2), & \text{if } q^i + \tilde{q}^i \leq 1, \ \forall i, \\ +\infty & \text{otherwise.} \end{array} \right. \end{array} \right. \tag{4.6}$$

Operator $\oplus$ is associative and commutative because

$$q_a \oplus q_b \oplus q_c = \left\{ \begin{array}{ll} (q_a^i + q_b^i + q_c^i)_i & \text{if } q_a^i + q_b^i + q_c^i \leq 1, \ \text{for } i \in \{1,2\} \\ +\infty & \text{otherwise.} \end{array} \right.$$

The order $\leqslant$ is compatible with $\oplus$ because, if $q_a \leq q_b$ and $q_a \oplus q = +\infty$, then $q_b \oplus q = +\infty$. Hence, $(\mathscr{R}, \oplus, \leqslant)$ is a lattice ordered monoid.

We now prove that we can reduce the $\mathscr{NP}$-complete STANDARD RESOURCE CONSTRAINED SHORTEST PATH PROBLEM stated in Section 3.2.1 to the problem of computing b_v^{opt} on a digraph with resources in $(\mathscr{R}, \oplus, \leqslant)$. Let D, o, d, w_a^i and R^i be an instance of the STANDARD RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. Let $q_a = (\frac{w_a^i}{R^i})_{i=1,2}$ if $\frac{w_a^i}{R^i} \leq 1$ for $i = 1,2$ and $+\infty$ otherwise. The $q_P = \sum_{a \in P} q_a \neq +\infty$ if and only if $\sum_{a \in P} q_a^i \leq R_i$ for all i. As a consequence, $\bigwedge_{P \in \mathscr{P}_{od}} q_P < +\infty$ if and only if there exists and o-d path P such that $\sum_{a \in P} q_a^i \leq R_i$ for all i. The value of $\bigwedge_{P \in \mathscr{P}_{od}} q_P$ gives the solution of the STANDARD RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. □

Nonetheless, computing b_v^{opt} is polynomial when $(\mathscr{R}, \oplus, \leqslant)$ is a dioid, which is the case of many applications. Indeed, using the algebraic path problem terminology introduced in Section 3.3, bound b_v^{opt} is the shortest distance between v and d. As a consequence, when $(\mathscr{R}, \oplus, \leqslant)$ is a dioid, the bounds b_v^{opt} can be computed by solving the dynamic programming equation (3.3). The solution method we introduce in the remaining of the section is a polynomial algorithm to solve the generalized dynamic programming equation in the lattice ordered monoid case. It therefore enables to compute b_v^{opt} in polynomial time when $(\mathscr{R}, \oplus, \leqslant)$ is a dioid.

For general lattice ordered monoids, the solution b_v of the dynamic programming equation is only a lower bound on b_v^{opt}. The set of discrete distributions with finite support endowed with the usual stochastic order and the convolution product introduced in Section 5.1.1 is an example of lattice ordered monoid in which one the bounds b_v are not tight.

4.2.1 Extended Ford-Bellman algorithm

The remaining of the section introduces a polynomial method to compute a good lower bound on b_v^{opt}. This lower bound turns out to be equal to b_v^{opt} when $\mathscr{R}$ is a dioid. Let $\left(b_v^n\right)_n$ be the sequence of collection of resources defined recursively as follows.

$$\left\{ \begin{array}{l} b_d^n = 0, \\ \\ b_v^0 = \infty \text{ and } b_v^{n+1} = \displaystyle\bigwedge_{(v,u) \in \delta^+(v)} (q_{(v,u)} \oplus b_u^n) \text{ for } v \in V \backslash \{d\}. \end{array} \right. \tag{4.7}$$

As $\mathscr{R}$ is a complete lattice, we can define $b_v^\infty = \bigwedge_{n\in\mathbb{Z}_+} b_v^n$ for each vertex v. Let $\mathbf{z}^n$ denotes the vector $(b_v^n)_{v\in V}$, and define $\mathbf{b}^\dagger = (b_v^\dagger)_{v\in V}$ to be the greatest solution of the following equation.

$$\begin{cases} b_d = 0, \\ b_v = \bigwedge_{(v,u)\in\delta^+(v)} (q_{(v,u)} \oplus b_u) \text{ for all } v \in V\backslash\{d\}. \end{cases} \tag{4.8}$$

The existence of a greatest solution of Equation (4.8) is a direct consequence of the Knaster-Tarski fixed point theorem applied in the complete product lattice $\mathscr{R}^V$. This theorem states that the set of fixed points of a monotone mapping in a complete lattice is a non-empty complete lattice. Details on the Knaster-Tarski fixed point theorem can be found in [47]. We underline that both $b_v^\dagger$ and b_v^∞ may be defined only in the completion of $\mathscr{R}$.

Theorem 4.12. *Let ℓ^* be the length of the longest elementary v-d path. For each vertex and elementary v-d path P, we have*

$$b_v^\dagger \leqslant b_v^\infty \leqslant b_v^{\ell^*} \leqslant b_v^{\text{opt}} \leqslant q_P \tag{4.9}$$

The three first inequalities are equalities when both $\mathscr{R}$ is a dioid and $q_c \geq 0$ for each cycle c in D.

These *bounds $b_v^{\ell^*}$ are good candidates to be used as a bound b_v on the resource q_P of all v-d paths P in the enumeration algorithms* of Section 4.1. Indeed, they can be computed in $O(|A|\ell^*)$ operations $\oplus$ and $\wedge$ by computing the ℓ^* first terms of sequence $(\mathbf{z}^n)_n$ using its definition in Equation (4.7). Besides, the sequence $(\mathbf{z}^n)_n$ can be interpreted as a *generalization of the Ford-Bellman algorithm*. Indeed, when $(\mathscr{R},\oplus,\leqslant) = (\mathbb{R},+,\leq)$, or more generally when $(\mathscr{R},\oplus,\leqslant)$ is a totally ordered group, the meet $q_1 \wedge q_2$ of two resources q_1 and q_2 is the minimum of q_1 and q_2. In that case, the sequence of Equation (4.7) corresponds to the successive steps of the Ford-Bellman shortest path algorithm, and for each integer k, the bound b_v^k is the value of a shortest o-v path with at most k arcs.

The proof of Theorem 4.12 relies on two lemmas. The mapping F defined as follows is useful in the proof.

$$\begin{array}{rccl} F: & \mathscr{R}^V & \longrightarrow & \mathscr{R}^V \\ & \mathbf{b} & \longmapsto & \mathbf{b}' \text{ such that } \begin{cases} b'_d = 0, \\ b'_v = \bigwedge_{(v,u)\in\delta^+(v)} (q_{(v,u)} \oplus b_u) \text{ for all } v \in V\backslash\{d\}, \end{cases} \end{array} \tag{4.10}$$

where $\mathscr{R}^V$ denotes the Cartesian product. Note that $\mathbf{b}^{n+1} = F(\mathbf{b}^n)$ and that F is isotone by monotonicity of the operators $\oplus$ and $\wedge$.

Lemma 4.13. *For each vertex v and integer n, we have $b_v^\dagger \leqslant b_v^\infty \leqslant b_v^n$.*

Proof. A straightforward induction on n based on the isotony of mapping F defined in Equation (4.10) gives $b_v^\dagger \leqslant b_v^n$ for all n, which implies that $b_v^\dagger \leqslant b_v^\infty$. □

Lemma 4.14. *The resource b_v^k is a lower bound on the resource q_P of the v-d paths P of length at most k. When $\mathscr{R}$ is a dioid, b_v^k is the meet of the resources of the v-d paths of length at most k.*

Proof. The result is proved by induction on k. The result for $k=0$, i.e. b_v^0 is equal to 0 if $v=d$ and ∞ otherwise, follows from the fact that the only path of length 0 is the trivial path. Let $k>0$ be an integer and suppose the result is true up to $k-1$, let v be a vertex, and let P be a v-d path with $\ell(P)\leq k$. If $\ell(P)=0$ then $v=d$ and $q_P \geqslant b_d^k = 0$. Otherwise let (v,u) be the first arc of P and Q be the subpath of P obtained by removing (v,u) from P. Then $\ell(Q)\leq k-1$, thus $b_u^{k-1} \leqslant q_Q$ which implies $q_{(v,u)} \oplus b_u^{k-1} \leqslant q_{(v,u)} \oplus q_Q$ and finally $b_v^k \leqslant q_P$, which gives that b_v^k is a lower bound on the resource q_P of the v-d paths P of length at most k.

In the dioid case, we have $q_1 \oplus (q_2 \wedge q_3) = (q_1 \oplus q_2) \wedge (q_1 \oplus q_3)$. Suppose that b_u^{k-1} is the meet of the resource of all the v-d paths of length at most k for each vertex u. Then $q_{(v,u)} \oplus b_u^{k-1}$ is the meet of the resources q_P of all the v-d paths P starting by (v,u) such that $\ell(P)\leq k$. Thus $b_v^{k-1} \wedge \bigwedge_{(v,u)\in\delta^+(v)} \left(q_{(v,u)} \oplus b_u^{k-1}\right)$ is the meet of all the v-d paths of length at most k. □

Proof of Theorem 4.12. As the length of any elementary v-d path is non greater than ℓ^*, Lemma 4.14 implies that $b_v^{\ell^*} \leqslant q_P$ for all elementary v-d paths P, and thus $b_v^{\ell^*} \leq b_v^{\text{opt}}$. Lemma 4.13 then gives Equation (4.9).

Suppose now that all cycles c in D satisfy $q_c \geq 0$ and that $\mathscr{R}$ is a dioid. A non-elementary v-d path is dominated by any of its elementary v-d subpaths. As the length of an elementary path is non greater than ℓ^*, the resource $b_v^{\ell^*}$ is a lower bound on the resources of all v-d paths. Thus, it is the meet of the resources of all the v-d paths. This implies that $F(\mathbf{b}^{\ell^*}) = \mathbf{b}^{\ell^*}$ and $b_v^{\ell^*}$ is a solution of Equation (4.8). Thus, $b_v^{\ell^*} = b_v^\dagger$, which gives the result. □

Remark 4.2. Note that if the resource of all cycles in D are non smaller than 0, then $b_v^{\ell^*}$ is a lower bound on the resources of all the v-d paths. On the contrary, if there exists an o-d path containing a negative cycle, then Lemma 4.14 implies that $b_v^\dagger = b_v^\infty = -\infty$.

Remark 4.3. The sequence $(\mathbf{b}^n)_n$ is the sequence used in the constructive proof by [45] of the Knaster-Tarski fixed point theorem for mapping F defined in Equation (4.10). Given a topology and some weak assumptions on $\mathscr{R}$, it can be proved that b_v^n converges to b_v^∞ and $b_v^\infty = b_v^\dagger$. The inequality $\bigwedge_{(v,u)\in\delta^+(v)}(q_{(v,u)} \oplus b_u^\infty) \leqslant b_v^\infty$ is easy to prove: indeed, as $q_{(v,u)} \oplus b_u^\infty \leqslant q_{(v,u)} \oplus b_u^n$ for each arc (v,u) in $\delta^+(v)$ and for all n in $\mathbb{Z}_+$, we have $\bigwedge_{(v,u)\in\delta^+(v)}(q_{(v,u)} \oplus b_u^\infty) \leqslant \bigwedge_{(v,u)\in\delta^+(v)}(q_{(v,u)} \oplus b_u^n) = b_v^{n+1}$ for all $n\in\mathbb{Z}_+$, which gives the result. The inequality $\bigwedge_{(v,u)\in\delta^+(v)}(q_{(v,u)} \oplus b_u^\infty) \geqslant b_v^\infty$ requires a transfinite induction.

4.2.2 Generalized Dijkstra algorithm for faster bound computations

In this section, we give a new algorithm to compute $\left(b_v^{\ell^*}\right)$ defined in Section 4.2. It exploits the fact that $b_v^{k+1} = b_v^k$ for most integers k to perform fewer operations.

A resource $\tilde{b}_v$ and an integer $\tilde{n}_v$ are attached to each vertex v and updated during the algorithm. Initially, $\tilde{b}_v = \bigvee \mathscr{R}$, which may be defined only in the completion of $\mathscr{R}$, and $\tilde{n}_v = +\infty$ for each vertex $v \neq d$, and $\tilde{b}_d = 0$. During the algorithm, a queue L of vertices "to be extended" is maintained. Initially, the queue L contains only d. The algorithm ends when L is empty. While

L is not empty and the minimum $\tilde{n}_v$ over all vertices v is non greater than ℓ^*, where ℓ^* is the maximum length of an elementary path ending in d, the following operations are repeated:

- Extract from L a vertex v with minimum $\tilde{n}_v$.
- For each arc (u,v) in $\delta^-(v)$, *extend* v along (u,v): if $\tilde{b}_u \not\leqslant q_{(u,v)} \oplus \tilde{b}_v$, then
  - *Update* $\tilde{n}_u = \min(\tilde{n}_u, 1+\tilde{n}_v)$.
  - *Update* $\tilde{b}_u$ to $\tilde{b}_u \wedge (q_{(u,v)} \oplus \tilde{b}_v)$.
  - Add u to L (if it is not already present).
- Set $\tilde{n}_v = +\infty$.

Proposition 4.15. *This algorithm terminates in less than $\ell^*|V|$ iterations, where ℓ^* is the maximum length of an elementary path ending in d. The value b_v of $\tilde{b}_v$ at the end of the algorithm is equal to $b_v^{\ell^*}$ for each $v \in V$. If L is empty at the end of the algorithm, then $b_v = b_v^\dagger$ for all vertices v.*

When $\mathscr{R}$ is a dioid, at the end of the algorithm, we have $L = \emptyset$ and $b_v = b_v^\dagger$ for all vertices v.

Remark 4.4. Mohri [130] proposes a generic algorithm for the algebraic path problem, whose only difference with our algorithm when both are restricted to dioid is the absence of $\tilde{n}_v$. The vertex picked-up in L at each iteration is arbitrarily chosen. The algorithms thus terminates when L is empty. He shows that the algorithm terminates after a finite but possibly exponential number of iterations.

Remark 4.5. Instead of $\tilde{n}_v$, we can use any key function $\phi(b_v)$. The only difference is that the algorithm ends only when L is empty, and convergence after a finite number of iterations cannot be proved in the general case. Remark 4.4 shows that we have convergence after a finite number of iterations if $(\mathscr{R}, \oplus, \leqslant)$ is a dioid. In practice, the list L is always empty after $\gamma|V|$ iterations for a relatively small γ, and we obtain $b_v^\dagger$. With carefully chosen ϕ, the worst γ encountered in the numerical experiments is 5.5 in a graph with 194675 vertices and 342735 arcs. Using the default $\tilde{n}_v$, the ratio γ can be ten times larger on the same graph.

Remark 4.6. When $(\mathscr{R}, \oplus, \leqslant) = (\mathbb{R}_+, +, \leq)$, the algorithm computing the iterates of F corresponds to the Ford-Bellman algorithm, whereas this algorithm corresponds to Dijkstra algorithm when using $\phi(x) = x$ as key function.

The proof of Proposition 4.15 relies on the following technical lemma.

Lemma 4.16. *The quantity $\min_v \tilde{n}_v$ does not decrease along the algorithm.*

Proof. Let $\hat{n}_v$ be the value of $\tilde{n}_v$ right before being extended if v has been extended, and $\hat{n}_v = 0$ otherwise. Note (u,v) is arbitrarily chosen and not required to be an arc in A. We prove the lemma by showing that, at any time during the algorithm, for each pair of vertices (u,v), we have $\hat{n}_u < \tilde{n}_u$ and $\hat{n}_u \leq \tilde{n}_v$.

The proof is by iteration on the steps of the algorithm. The result is true at the beginning of the algorithm. Let v be the vertex currently extended, and suppose that the result is true before the extension of v. For each vertex u, let n_u^α be the value of $\tilde{n}_u$ before the extension of v and n_u^β the value of $\tilde{n}_u$ after the extension. After the extension of v, we have $\hat{n}_v = n_v^\alpha < \infty = n_v^\beta$. Let u be a vertex distinct from v. The index $\hat{n}_u$ is not modified during the extension of v. If (u,v) is

not an arc, or if $\tilde{b}_v \leqslant q_{(u,v)} \oplus \tilde{b}_u$ before the extension, then $\tilde{n}_u$ is not updated, and $n_u^\beta = n_u^\alpha > \hat{n}_u$ and $\hat{n}_u \leq n_v^\alpha \leq n_u^\alpha = n_u^\beta$. If on the contrary u is updated, $n_u^\beta = \min(n_u^\alpha, n_v^\alpha + 1)$. As $n_v^\alpha \leq n_u^\alpha$, there are two possibilities. In the first case, $n_u^\alpha = n_v^\alpha$, which implies $n_u^\beta = n_u^\alpha$, and by induction hypothesis $\hat{n}_u < n_u^\alpha = n_v^\alpha \leq n_u^\beta$. In the second case $n_u^\alpha > n_v^\alpha$, and we have $n_u^\beta = n_v^\alpha + 1 > n_v^\alpha \geq \hat{n}_u$. Finally, in both cases $\hat{n}_u \leq n_v^\alpha$. We have thus proved that $\hat{n}_u < n_u^\beta$ and $\hat{n}_u \leq n_v^\alpha \leq n_u^\beta$ for each vertex u, which gives the result and the lemma. □

The proof of Proposition 4.15 is now relatively straightforward, as Lemma 4.16 enables to link the values taken by b'_v along the algorithm to the sequence (b_v^i) defined in Equation 4.7.

Proof of Proposition 4.15. Lemma 4.16 ensures that $\min_v \tilde{n}_v$ does not decrease. Based on this results, the update rule ensures that for each vertex u in L, if $\tilde{n}_u \neq +\infty$, we have either $\tilde{n}_u = \min_v \tilde{n}_v$ or $\tilde{n}_u = 1 + \min_v \tilde{n}_v$. As a consequence, $\min_v \tilde{n}_v$ increases by at most one between two iterations. For each vertex u and index i, define b_u^i and n_v^i to be equal to the values of $\tilde{b}_u$ and $\tilde{n}_u$ when $\min_v \tilde{n}_v = i$ for the first time. Due to the update rule, we obtain by induction on i that $b_u^i = b_u^{i-1} \bigwedge_{(u,v)\in\delta^+(u)} \left(q_{(u,v)} \oplus b_v^{i-1}\right)$. Indeed, suppose that the result is true up to $i-1$, and consider a vertex u, and an arc $(u,v) \in \delta^+(u)$. The update rule then implies $b_u^i = b_u^{i-1} \bigwedge_{v\in U_u^i} \left(q_{(u,v)} \oplus b_v^{i-1}\right)$ and $n_v^{i-1} = +\infty$, where U_u^i is the set of all v such that $(u,v) \in \delta^+(u)$ and $n_v^{i-1} = i-1$. Besides, if $n_v^{i-1} \neq i-1$, the $b_v^{i-1} = b_v^{i-2}$ and the induction hypothesis gives the result. As a consequence, the b_v^i correspond to those defined by Equation (4.7), and we obtain the first part of the proposition and the dioid case. Besides the update rule ensures that if there is no vertex u in L such that $(v,u) \in \delta^+(v)$, then $\tilde{b}_v = \bigwedge_{(v,u)\in\delta^+(v)} b_v$. Thus, if L is empty at the end of the algorithm, then $(b_v)_v$ defines a solution of Equation (4.8), which gives the second part of the proposition. □

4.3 Numerical results on a resource constrained shortest path problem

In this section, we test the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM algorithms on instance of the STANDARD RESOURCE CONSTRAINED SHORTEST PATH PROBLEM with one or ten constraints. Given a digraph D, an origin vertex o and a destination vertex d, a cost c_a and weights w_a^i for i in $[k]$, and thresholds τ^i for i in $[k]$, this problem can be expressed as follows

$$\begin{array}{rl} \min\limits_{P\in\mathscr{P}_{od}} & \sum\limits_{a\in P} c_a, \\ \text{s.t.} & \sum\limits_{a\in P} w_a^i \leq \tau^i \quad \text{for } i \text{ in } [k], \end{array} \tag{4.11}$$

where $\mathscr{P}_{od}$ is the set of o-d paths.

The lattice ordered monoid used to solve this problem is $\mathbb{R}^{k+1}$ endowed with its product sum and order. Our algorithms have been primarily designed for non-linear and stochastic resource constrained shortest path problem. The objective of this section is to give general ideas on the relative performances of our different algorithms. We have therefore not exploited the specific structure of $\mathbb{R}^{k+1}$ to accelerate them. Remark 4.7 gives an example of how to accelerate them. If the arc resources are specific to this chapter numerical experiments, the

Name	*Generator*	*Brief description*
road	extraction	Road networks with a given number of vertices.
square	grid	Square grid of size $m \times m$
long	grid	Long grid of size $16m \times m$
wide	grid	Wide grid of size $m \times 16m$
acyc	acyclic	Acyclic graph with n vertices $v_1, \ldots, v_n$ and m arcs (v_i, v_j) with $i < j$ $m = hn$ for h between 2 and 50
rand	random	Hamiltonian cycle with n vertices and $m - n$ chords. $m = hn$ for h between 2 and 50

Table 4.2 – Summary of the families of graphs used

graphs of the instances will be used in the next chapter with different resources to test our algorithms on stochastic path problems.

We solve each instance in two steps. First, we run the generalized Dijkstra algorithm to compute the lower bounds b_v, and second we use an enumeration algorithm to solve the problem. As we use positive resources, Theorems 4.7 and 4.8 ensure respectively that the label correcting and the label dominance algorithm converge. Case (b) of Theorem 4.3 with assumption (4.3) ensures the convergence of the generalized A* algorithm.

Remark 4.7. We use our generalized Dijkstra algorithm to compute the lower bounds. As $\mathbb{R}^{k+1}$ is a dioid, Theorem 4.12 ensures that the bounds computed are the component-wise shortest paths. If each component of the resources is non-negative, a speed-up of order γ of the preprocessing can be obtained by using Dijkstra algorithm or any other shortest path algorithm to compute each component shortest path, where γ has been defined in Remark 4.5.

4.3.1 Graph instances used

We now introduce the graphs instances that are used in this chapter. These instances are also used in Chapter 5 and Chapter 6. We use four families of graphs: road networks, acyclic graphs, grids, and random graphs. The three last families of graphs are used by Cherkassky et al. [38] in their experimental study of algorithms for the STANDARD SHORTEST PATH PROBLEM. We have used adapted versions of their generators `spgrid`, `sprand`, `spacyc` to produce instances of these families. The adaptation consists in the insertion of a destination vertex. Among others, these four family of graphs have been used by Dumitrescu and Boland [61] to test the different STANDARD RESOURCE CONSTRAINED SHORTEST PATH PROBLEM algorithms available in the literature. The remaining of the section provides the definition of the different families of instances, Table 4.2 provides a summary of the main characteristics of these families of graph, and Figure 4.1 gives examples of graphs from these families

The *road* network graphs have been extracted from the Rome and the San Francisco Bay Area instances of the Dimacs challenge [2] as follows. We suppose to have a road network digraph $D = (V, A)$, and we want to extract from it a subgraph with $n < |V|$ vertices. We choose randomly an initial vertex and compute the subgraph of the n nearest vertices to v. We then repeat this operations ten times and keep the subgraph with the maximum number of vertices.

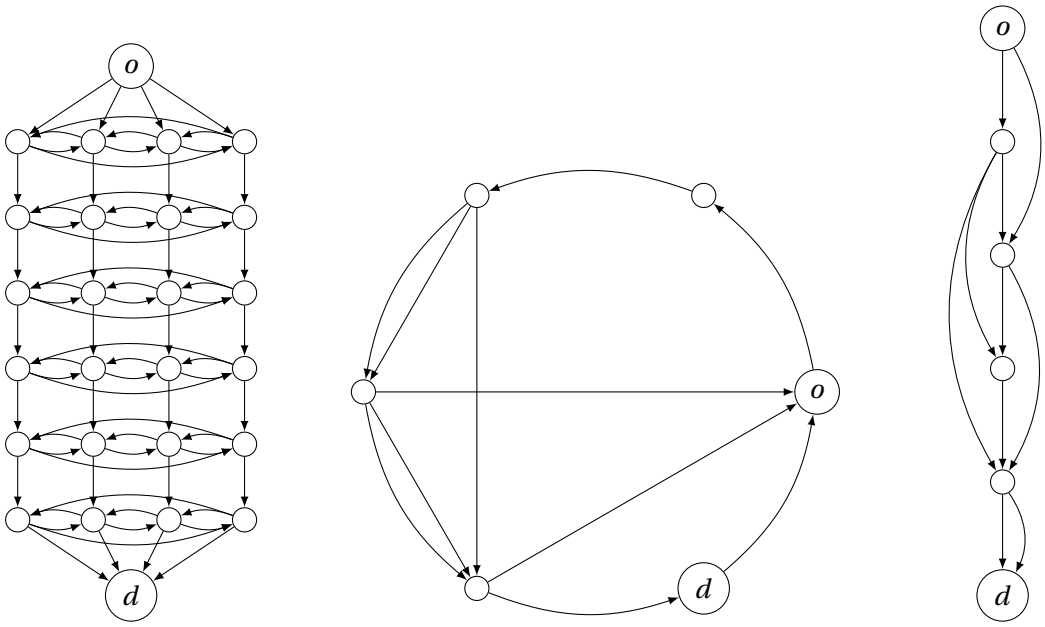


Figure 4.1 – Examples of our families of graphs. From left to right, a grid graph of width 4 and depth 6, a random graph with 6 vertices and 10 arcs, and an acyclic graph with 6 vertices and 10 arcs.

The subgraph being extracted, we use a graph traversal approach [46] to find an origin and a destination vertex far from each other. We choose randomly an origin o and compute a furthest vertex d in terms of number of arcs. The vertex d found is then chosen as the new origin o, and a new vertex d is computed as the furthest vertex from the previous one in terms of number of arcs. This operation is repeated five times. We finally obtain an origin o and a destination d such that paths between o and d have a large number of arcs. In the newly generated subgraph, we keep the length of the arcs of the original graph. In The Dimacs instances used, these lengths correspond to road distances.

We now give the definition of the families of graphs used in [38]. A *grid* graph of length ℓ and width m is composed of ℓ layers of m vertices, as illustrated on the left part of Figure 4.1. Each layer i can be seen as a Hamiltonian cycle $v_{i,1}, \ldots, v_{i,m}$ with arcs in both direction. Each vertex $v_{i,j}$ of layer $i \in [\ell - 1]$ is connected to the same vertex $v_{i+1,j}$ in the next layer. An origin vertex o and a destination vertex d are added. There is an arc between the origin o and each vertex of the first layer, and an arc between each vertex of the last layer and the destination. A *random* graph with n vertices and $m \geq n$ arcs is composed of n vertices on a Hamiltonian cycle and randomly generated chords on that cycle. A random graph is illustrated in the middle of Figure 4.1. Finally *acyclic* graphs with n vertices an $m > n$ arcs are generated as follows. A path with n vertices $v_1, \ldots, v_n$ is generated. The origin and the destination are respectively the first and the last vertex of the path. Additional arcs between v_i and v_j, with i and j randomly generated and satisfying $i < j$. Such and acyclic graph is illustrated on the right part of Figure 4.1. We consider random and acyclic graphs with n vertices and hn arcs with h between 2 and 50. In each of these graphs, the length of each arc is chosen randomly

between 1 and 100.

4.3.2 Resources and constraints

The resource of the STOCHASTIC RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. The cost of an arc is its length. The weights w_a^i of an arc is randomly chosen between 1 and 100. We consider the case with 1 weight and the case with 10 weights. We choose the thresholds in the same way as Dumitrescu and Boland [61]. A o-d path P_c of minimum cost, and an o-d path P_w minimizing $\sum_{a\in P_w}\sum_{i\in k} w_a^i$ are computed. We then choose the *constraint strength* α

$$\tau^i = (1-\alpha)\, w^i_{P_w} + \alpha \max(w^i_P, w^i_{P_w}) \tag{4.12}$$

with $\alpha \in \{0.1, 0.5, 0.9\}$ corresponding respectively to strong, medium and light constraints. This choice of thresholds ensures the existence of a solution, as P_w is feasible, but that the optimal solution of the non-constrained problem is not feasible.

4.3.3 Experimental setting

The numerical experiments are performed on a Macbook Pro of 2012 with four 2.5 Ghz processors and 4 Gb of ram. We use our `latticeRCSP` library described in Appendix C to solve the instances. The algorithms are not parallelized. The limiting parameter for the algorithms is the memory available. Therefore, we stop the algorithms if the list L of candidate paths contains more that 1e+05 elements. We also stop the label correcting and the label dominance algorithms if the number of paths in the union of the sets of non dominated paths M_v is larger than 1e+05. The minimum $c(q_P + b_v)$ on the paths P in L when the algorithm is stopped provides a lower bound on the cost of an optimal solution, and c_{od}^{UB} provides an upper bound.

Figure 4.2 provides the CPU times as a function of the number of vertices in the graph for the instances solved to optimality on each family of instances for constraints of medium strength $\alpha = 0.5$ for the resource constrained shortest path problem with $k = 1$ constraint. For the random and the acyclic instances, we have $h = 5$. We have plotted curves for each family of graph and algorithms because, once the parameter h is fixed and the grid family has been split into long, wide, and square grids, there is only one degree of freedom in the definition of instances of each family of graphs, and thus giving the number of vertices of the graph identifies the instance considered.

Candidate paths

A standard technique to accelerate resource constrained shortest path algorithms [61] is to compute candidate v-d paths Q_v for each vertex v during a preprocessing. If a candidate o-v path is not discarded, before extending it, we test if $P + Q_v$ is a feasible solution of cost smaller than c_{od}^{UB}, and update c_{od}^{UB} if it is. The idea behind these candidate paths is that candidate paths enable to find faster feasible o-d of small cost, and thus to get faster a small c_{od}^{UB}, which enable to reduce the number of paths considered by the algorithm. We compute these candidate paths in a preprocessing by taking those that minimize $\sum_{a\in P} c_a + \sum_i w_a^i$.

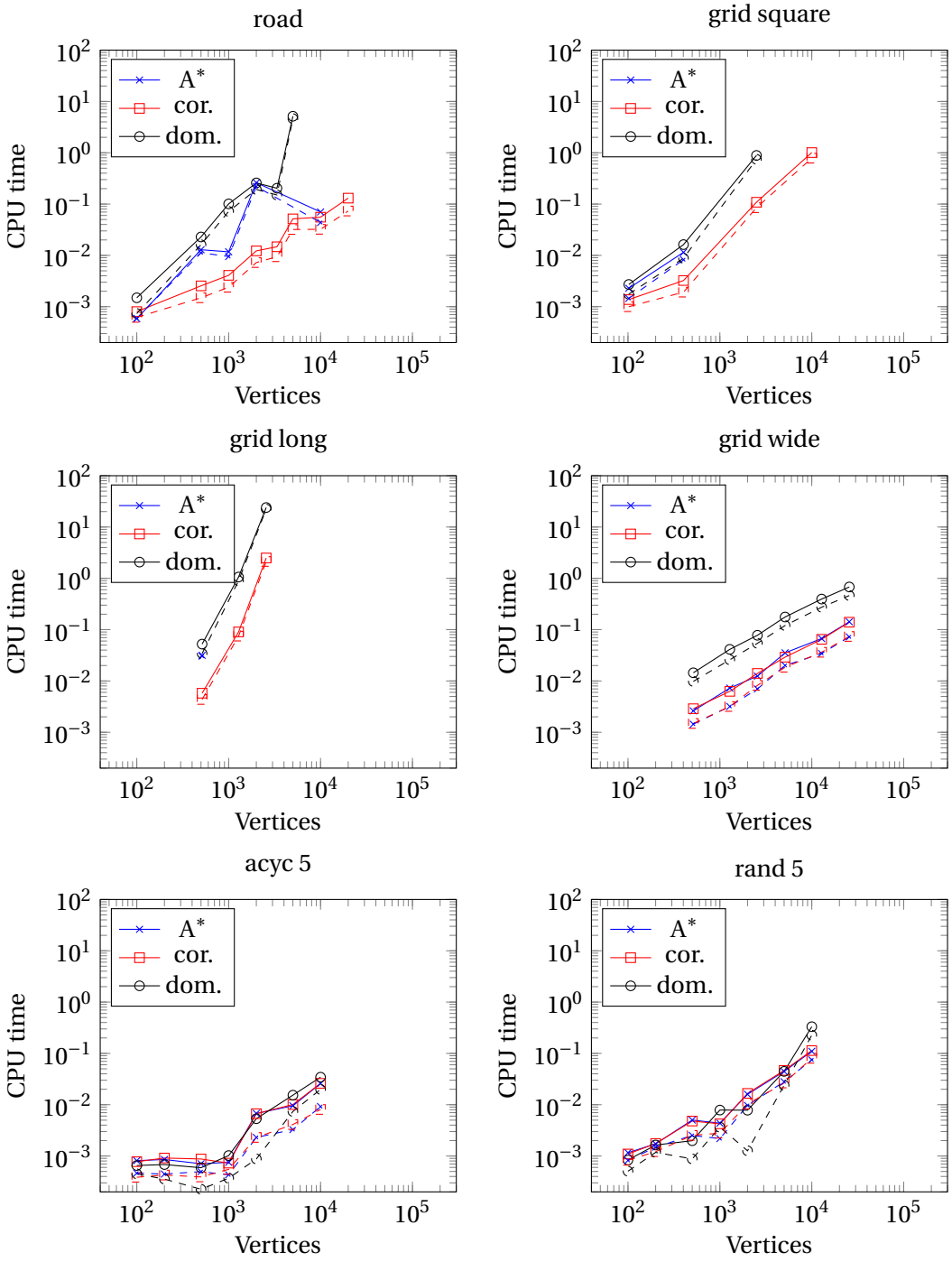


Figure 4.2 – Resource constrained shortest path problem results with $k = 1$ and $\alpha = 0.5$. The dashed lines correspond to algorithms without candidate paths, and the plain lines to algorithms with candidate paths.

On Figure 4.2, we therefore provide two versions of the labeling algorithms: one with candidate paths, and one without. Plain lines correspond to algorithms with candidate paths, and dashed lines to algorithms without.

4.3.4 Diameter, difficulty of instances, and algorithms performances

For each family of instances considered, Table 4.3 provides the results of the algorithms on the largest instance solve to optimality and the smallest non-solved to optimality. The results on the two largest instances are provided for the family where all the instances have been solved to optimality. The first column of Table 4.3 indicated the name of the instance considered. Instances are named after their family and the parameter that identify them in their family. The next columns provide the number of vertices and the number of arcs in the instances. The column Alg. provides the algorithm used to solve the algorithm. The name cor. and dom. corresponds to the label correcting and the label dominances algorithm. When the suffix no is added to the name of an algorithm, it means that no candidate paths have been computed in a preprocessing, while when the suffix no is not present, candidate paths have been computed. The next column corresponds to the quantity γ defined in Remark 4.5: being defined as the ratio of the number of vertices extensions divided by the total number of vertices, it indicates the performance of the generalized Dijkstra algorithm of Section 4.2.2. The column Preproc. provides the percentage of the total CPU time spent in the preprocessing phase computing the bounds. There is no such phase for the label dominance algorithm. The two next columns provide the number of paths extended and the number of of paths cut. The column Dom. provide the percentage of paths cut by the dominance test in the label correcting algorithm, the remaining of the paths being cut by the lower bound test. The column ℓ corresponds to the number of arcs in the solution returned, and the last column provides the total CPU time of the algorithm, including the preprocessing.

We now analyze the results of Table 4.3 and Figure 4.2. The acyclic and the random instances are the easiest to solve. The road network instances are of medium difficulty. The wide grid instances are rather easy to solve, while the square grids are difficult and the long grids are very difficult to solve. Even if the difficulty highly depend on the topology of the instances, we can observe that the difficulty of an instance tend to be correlated with the number of arcs in an optimal solution. The wide grid instances are therefore easier to solve than the square and the long grid instances. As the acyclic and the random instances are easy to solve, we do not consider them in the remaining of the dissertation.

The limit on the label map size leads to small computing times. A natural idea is to increase it, in order to get better results on the instances not solved to optimality. Nonetheless, increasing the maximum label map size leads to the use of a too large amount of memory and makes the program crash.

The parameter γ of our generalized Dijkstra algorithm remains much smaller than its upper bound ℓ. It is therefore more interesting to use this algorithm than the extended Ford-Bellman algorithm. The preprocessing can take a substantial proportion of the total computation times, but when it is the case, it means that the instance can be well solved.

On the STANDARD RESOURCE CONSTRAINED SHORTEST PATH PROBLEM with one constraint, the

Instance	$\|V\|$	$\|A\|$	α	Alg.	γ	Preproc.	Ext.	Cut.	Dom.	ℓ	Gap	CPU (s)
road20	20000	55180	0.5	A*	1.6	9%	55762	17579	–	252	2.5%	4.91e-01
				cor.	1.6	36%	7852	8228	44%	243	opt	1.30e-01
				dom.	–	–	1505043	995349	–	243	21.0%	1.20e+01
				A* no	1.6	15%	54131	14121	–	0	inf%	3.08e-01
				cor. no	1.6	64%	7899	8306	44%	243	opt	7.32e-02
				dom. no	–	–	1505043	995349	–	0	inf%	1.06e+01
road50	50000	138112	0.5	A*	2.7	25%	44613	3710	–	494	13.6%	7.21e-01
				cor.	2.7	12%	153221	80488	96%	478	5.5%	1.60e+00
				dom.	–	–	1557409	1029848	–	462	431.2%	1.44e+01
				A* no	2.7	43%	44613	3710	–	0	inf%	4.42e-01
				cor. no	2.7	17%	153191	80428	96%	0	inf%	1.12e+00
				dom. no	–	–	1557409	1029848	–	0	inf%	1.21e+01
square50	2502	7550	0.5	A*	1.7	2%	70109	40264	–	64	28.1%	4.01e-01
				cor.	1.7	6%	25466	25428	50%	62	opt	1.09e-01
				dom.	–	–	196904	126563	–	61	opt	8.87e-01
				A* no	1.7	2%	70109	40264	–	0	inf%	2.78e-01
				cor. no	1.7	9%	25553	25489	50%	61	opt	8.50e-02
				dom. no	–	–	197010	126645	–	61	opt	7.45e-01
square100	10002	30100	0.5	A*	1.6	7%	58533	17162	–	131	30.4%	4.36e-01
				cor.	1.6	2%	168016	155396	58%	118	opt	1.01e+00
				dom.	–	–	1493529	973040	–	131	25.1%	1.21e+01
				A* no	1.6	9%	58533	17162	–	0	inf%	2.69e-01
				cor. no	1.6	3%	168036	155429	58%	118	opt	7.84e-01
				dom. no	–	–	1493529	973040	–	0	inf%	1.05e+01
long10	2562	7696	0.5	A*	1.6	2%	52564	5139	–	211	43.9%	3.39e-01
				cor.	1.6	0%	392260	291164	85%	193	opt	2.49e+00
				dom.	–	–	1492274	983655	–	193	opt	2.40e+01
				A* no	1.6	2%	52564	5139	–	0	inf%	2.24e-01
				cor. no	1.6	0%	392301	291221	85%	193	opt	2.13e+00
				dom. no	–	–	1492535	983841	–	193	opt	2.24e+01
long20	5122	15376	0.5	A*	1.6	3%	51961	3933	–	438	58.6%	4.17e-01
				cor.	1.6	1%	186590	93798	96%	419	33.9%	1.45e+00
				dom.	–	–	1558884	1024684	–	437	202.3%	2.62e+01
				A* no	1.6	5%	51961	3933	–	0	inf%	2.66e-01
				cor. no	1.6	1%	186590	93798	96%	0	inf%	1.12e+00
				dom. no	–	–	1558884	1024684	–	0	inf%	2.36e+01
wide50	12802	39200	0.5	A*	1.5	52%	118	1034	–	19	opt	6.71e-02
				cor.	1.5	51%	102	986	1%	19	opt	6.48e-02
				dom.	–	–	93248	53879	–	19	opt	3.95e-01
				A* no	1.5	96%	124	1043	–	19	opt	3.42e-02
				cor. no	1.5	96%	108	995	1%	19	opt	3.63e-02
				dom. no	–	–	93511	54052	–	19	opt	2.74e-01
wide100	25602	78400	0.5	A*	1.5	51%	104	1806	–	21	opt	1.42e-01
				cor.	1.5	50%	95	1782	0%	21	opt	1.40e-01
				dom.	–	–	148564	83772	–	21	opt	6.79e-01
				A* no	1.5	97%	107	1809	–	21	opt	7.23e-02
				cor. no	1.5	97%	98	1785	0%	21	opt	7.36e-02
				dom. no	–	–	148903	83968	–	21	opt	4.62e-01

Table 4.3 – Standard resource constrained shortest path with one resource constraint

label correcting algorithm performs better than the generalized A* and the label dominance algorithms. The generalized A* algorithm performs better than the label dominance algorithm on the road network and on the wide grid instances, but worse on the other instances. This can be explained by the fact that most paths cut by the label correcting algorithm are cut by the dominance test on these instances. Using lower bounds is nonetheless a good idea, as label correcting algorithm always performs better than the label dominance algorithm.

The algorithms without candidate paths perform better on the instances they can solve, but when it is not the case, the candidate paths enable to obtain a better solution and a smaller gap at the end of the algorithm. Besides, on the instances solved to optimality, the additional CPU time due to candidate paths is not large. Therefore, when the objective is to solve the problem, using candidate paths is a good idea.

4.3.5 Influence of constraint strength

Figure 4.3 provides A* and label correcting algorithm performances on identical instances but with constraint strength α in $\{0.1, 0.5, 0.9\}$. We expect instances with strong constraints, i.e. with small α, to be more difficult to solve. This is in practice the case for long grids, but on the other family of instances, the results are less stringent. Random and acyclic instances seem to be easier to solve when more constrained. Road and square grid instances seem to be more difficult to solve for constraints of medium strength.

4.3.6 Influence of dimension

Figure 4.4 and Table 4.4 are respectively the analogues of Figure 4.2 and Table 4.3 for the problem with $k = 10$ resources. Only instances solved to optimality are plotted on Figure 4.4. In Table 4.4, we only provide the algorithms with candidate paths, as the behavior with or without candidate paths is analogue to the one constraint case. Instances with ten resource constraints are more difficult than instances with only one. Besides, difficulty of the family of instances are similar: acyclic and random instances are easy while grids are difficult. We now focus on the difficult instances, i.e. the road and the grid instances.

Concerning the performance of the algorithms, the main difference with the one constraint case is the poor performance of the label dominance algorithm – except on the easy acyclic and random instances. We can explain this phenomenon by observing that, when there are ten constraints, dominance is rare, because it suffices of one component $q^i > \tilde{q}^j$ for q not to dominate $\tilde{q}$. This can be observed by the fact that the proportion of resources cut by the dominance test in Table 4.4 is much smaller than the proportion of paths cut by the dominance test for analogue instances in Table 4.3. This phenomenon also explain why the proportion of paths cut by the dominance test in the label correcting algorithm tends to be smaller than in the one constraint case, which itself explains why the generalized A* algorithm performs almost as well as the label correcting algorithm.

Instance	$\|V\|$	$\|A\|$	α	Alg.	γ	Preproc.	Ext.	Cut.	Dom.	ℓ	Gap	CPU (s)
road20	20000	55180	0.5	A*	2.1	6%	253214	323551	–	221	55.3%	1.23e+00
				cor.	2.1	18%	45034	66317	21%	221	opt	3.82e-01
				dom.	–	–	1103713	603707	–	221	373.3%	3.91e+01
road50	50000	138112	0.5	A*	2.5	30%	56159	9481	–	400	18.1%	7.12e-01
				cor.	2.5	10%	64093	17783	53%	400	18.0%	2.10e+00
				dom.	–	–	477467	307503	–	400	1321.4%	2.76e+01
square20	402	1220	0.5	A*	2.2	7%	6140	12298	–	21	opt	2.31e-02
				cor.	2.2	9%	3476	5540	13%	21	opt	1.45e-02
				dom.	–	–	642456	325326	–	21	opt	3.00e+01
square50	2502	7550	0.5	A*	2.2	2%	81247	62539	–	52	57.5%	4.73e-01
				cor.	2.2	0%	141815	103501	39%	52	49.3%	2.64e+00
				dom.	–	–	168274	78864	–	52	376.1%	1.84e+00
long2	514	1552	0.5	A*	2.3	0%	225825	451664	–	33	opt	9.38e-01
				cor.	2.3	1%	34149	49036	20%	33	opt	2.51e-01
				dom.	–	–	196469	97437	–	33	113.9%	3.06e+00
long5	1282	3856	0.5	A*	2.2	1%	64642	29295	–	81	67.9%	4.02e-01
				cor.	2.2	0%	101197	52388	48%	81	62.9%	1.90e+00
				dom.	–	–	195012	96678	–	81	599.7%	3.03e+00
wide50	12802	39200	0.5	A*	2.1	63%	2749	6296	–	17	opt	8.73e-02
				cor.	2.1	59%	2706	6000	2%	17	opt	8.54e-02
				dom.	–	–	125699	48327	–	17	117.1%	8.27e-01
wide100	25602	78400	0.5	A*	2.1	59%	6698	14994	–	17	opt	1.81e-01
				cor.	2.1	60%	6409	13726	3%	17	opt	1.74e-01
				dom.	–	–	108128	38179	–	17	314.6%	6.69e-01

Table 4.4 – Standard resource constrained shortest path with ten resource constraints

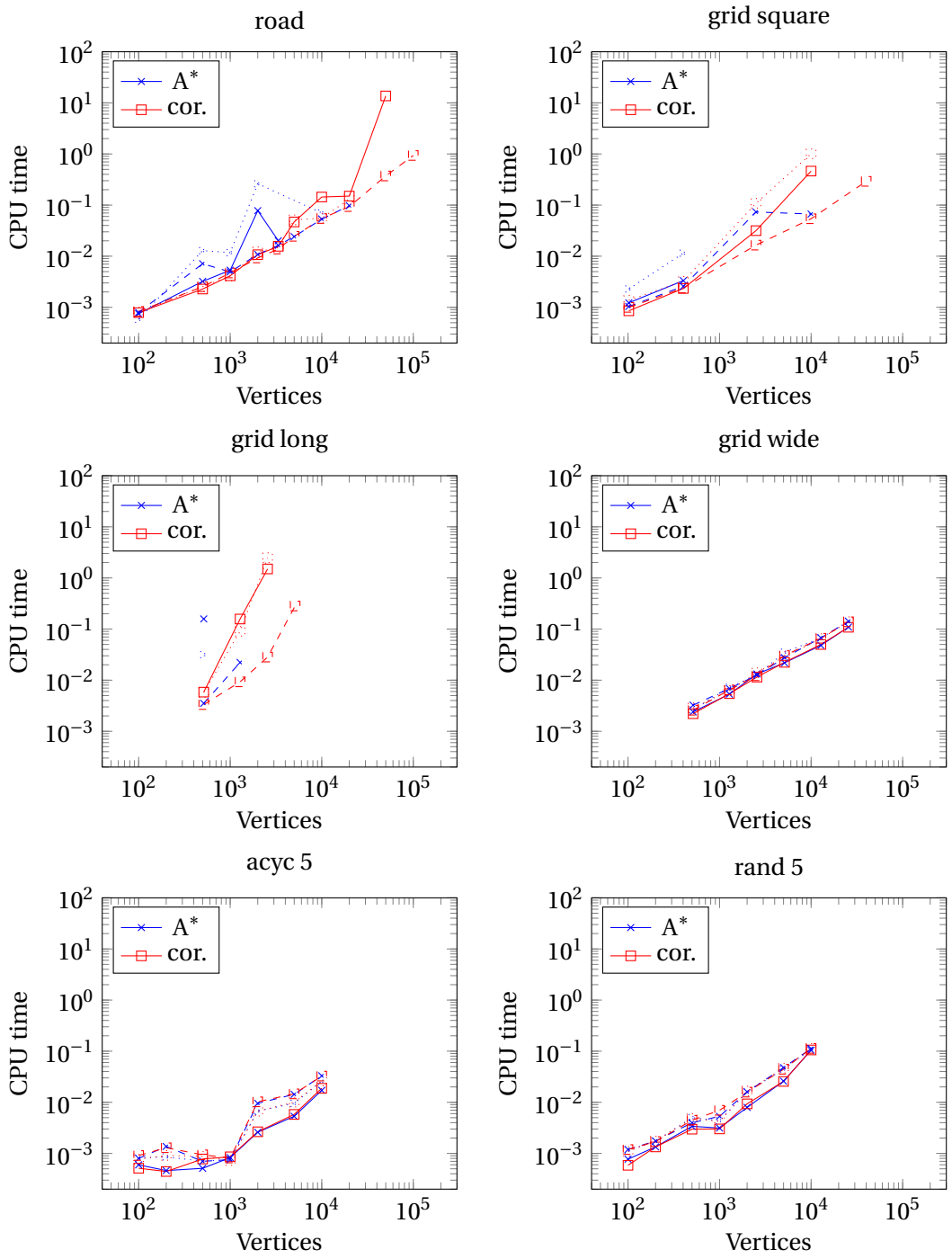


Figure 4.3 – Resource constrained shortest path problem solved to optimality with $k = 1$. Plain lines correspond to constraint strength $\alpha = 0.1$, dotted lines to $\alpha = 0.5$, and dashed lines to $\alpha = 0.9$

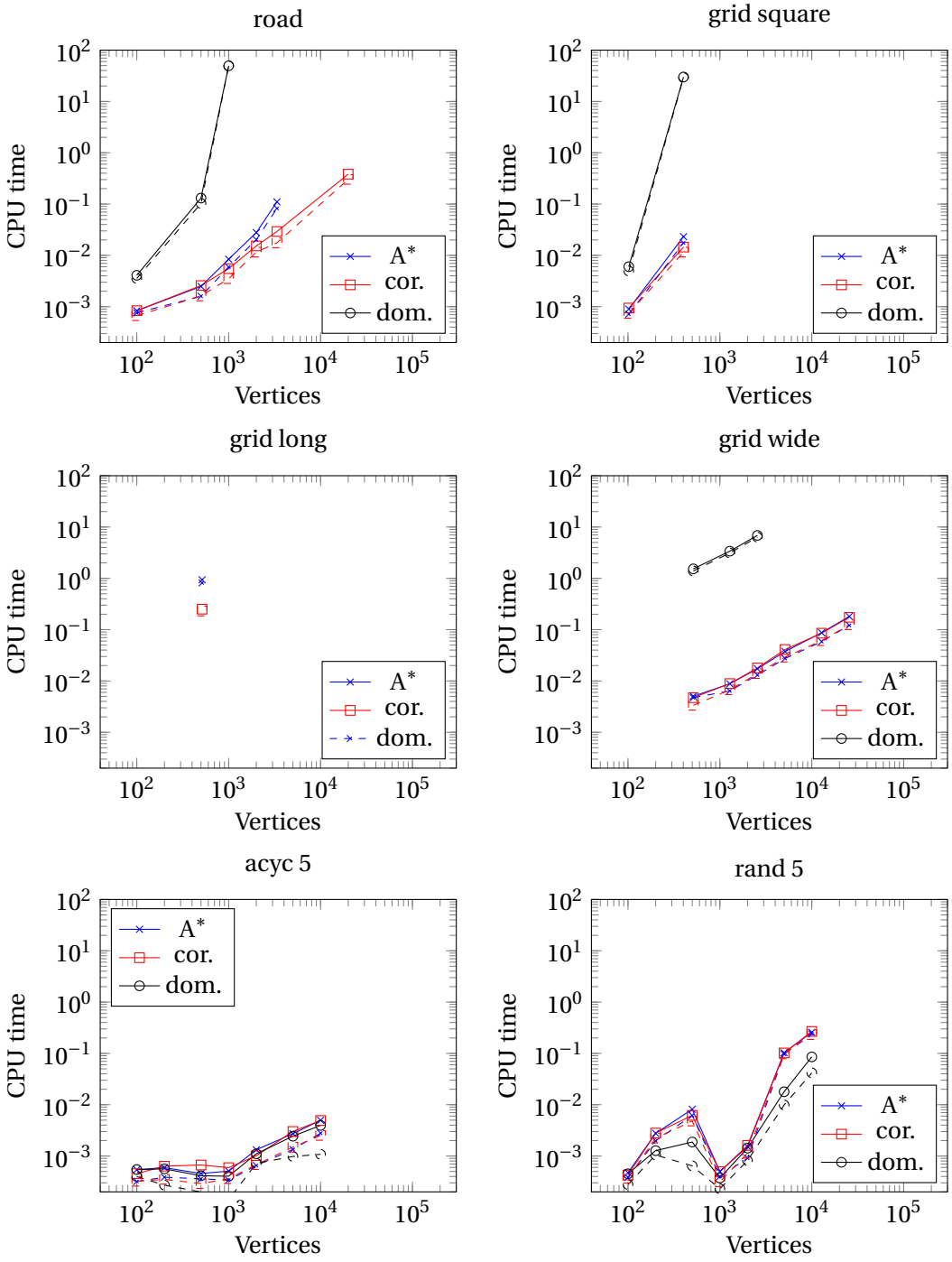


Figure 4.4 – Resource constrained shortest path problem solved to optimality with $k = 10$ and $\alpha = 0.5$. The dashed lines correspond to algorithms without candidate paths, and the plain lines to algorithms with candidate paths.

4.4 Bibliographical remarks

Variants of the STANDARD SHORTEST PATH PROBLEM has been thoroughly studied during the last six decades. As we already mentioned, there are two main types of algorithms for the STANDARD SHORTEST PATH PROBLEM: the polynomial algorithms, such as Dijkstra's algorithm or Ford-Bellman algorithm, and the enumeration algorithms, such as A*.

Dijkstra's algorithm [59] is the standard algorithm to solve it when arcs costs are non-negative, and Ford-Bellman [25, 77] algorithm is the standard way to solve it when arc costs can be negative. Both algorithms computes the shortest path between one vertex and all the other ones. In that sense they can be seen as solution algorithms for the dynamic programming equation. Ahuja et al. [9] call these algorithms label algorithms, in the sense that they assign a label b_v, i.e. the solution of the dynamic programming equation, to each vertices. They distinguish two classes of label algorithms. The label setting algorithm are such that when a label b_v has been set, it is never updated. On the contrary, in label correcting update several times the label b_v before convergence. Label setting algorithms, such as Dijkstra's algorithm, mostly apply to problems with non-negative arc costs, while label correcting algorithm, such as Ford-Bellman algorithm, apply to all problems. We note that our generalized Dijkstra algorithm is a label correcting algorithm according to Ahuja et al. [9]'s definition. We named one of our enumeration algorithms a label correcting algorithm in order to stick to resource constrained shortest path problem terminology, which defines label algorithms are enumeration algorithms. In label correcting algorithms for resource constrained shortest path problem, labels correspond to paths and not to vertices, and there can therefore be several labels on the same vertex. They are therefore not label algorithms according to Ahuja et al. [9]'s definition.

Enumeration algorithms use bounds to discard paths in an enumeration of all the paths. The typical example of enumeration is A* [93]. These algorithms are called goal directed algorithms as they compute the shortest path only between a given pair of origin and destination vertices. When good bounds are used, enumeration algorithms are faster than polynomial algorithm. These algorithms have been thoroughly studied during the last decades as they are the core of online journey planning systems. The difference between enumeration algorithms are the bounds used, and the trade-off they achieve between the memory needed to store the bounds and the computation needed to answer a given request. Cherkassky et al. [38] benchmark the usual shortest path algorithms and Bast et al. [21] survey the recent contributions to this field.

The generalizations of Dijkstra and Ford-Bellman algorithms aim at finding a solution to generalized versions of the dynamic programming equation. They are therefore studied in the algebraic path problem community. Generalizations of the enumeration algorithms are often used to solve resource constrained shortest path problems.

4.4.1 Bounding algorithms and algebraic path problem

The aim of the algebraic path problem, which we introduced in Section 3.3, is to solve the dynamic programming equation (3.3) when resources belong to a semiring. It has been studied by many authors on semiring frameworks of various generality [8, 17, 34, 74, 86, 115, 130, 152, 178]. Their algorithms which are valid on bounded semirings can therefore be used

to solve our generalized dynamic programming equation (4.8) when $(\mathscr{R}, \oplus, \leqslant)$ is a bounded dioid. Fink [74] surveys the sequential and parallel algorithms for the algebraic path problem.

Zimmermann [178] and Gondran and Minoux [86] sum up the literature which considers the algebraic shortest path problem in terms of linear equation systems in dioids. Many algebraic path problem algorithms can therefore be interpreted as dioid versions of the usual algorithms used to solve linear equations. Our generalized Ford-Bellman algorithm is identical to their one when restricted to dioids. They show that it is a generalization of Jacobi algorithm for linear equations systems. Unfortunately, their generalized Gauss-Seidel and Gauss-Jordan algorithms do not extend to lattice ordered monoids as they require $\oplus$ to be distributive with respect to $\wedge$.

As we already mentioned in Remark 4.4, Mohri [130] gives a generic algorithm for the algebraic shortest path problem that is very close to our generalized Dijkstra algorithm. The main difference is that, as they are working on semirings, convergence after a finite number of iterations is obtained even if the only stopping criterion used is the emptiness of L. Besides, the generalized dynamic equation is a fixed-point equation in a lattice. Our generalized Ford-Bellman algorithm can therefore be interpreted as a specific case of the fixed-point algorithms in lattices [44, 45]. Several approaches have been developed to accellerate the convergence of such algorithms. See e.g. [11, 13] for recent contributions. As our generalized Dijkstra algorithm converges quickly in practice, as mentioned in Remark 4.5, we haven't considered these accelerating techniques.

4.4.2 Enumeration algorithm for resource constrained path problems

In this dissertation, we focus on exact approaches to the resource constrained shortest path problem. A review of heuristics for this problem is available in the survey of Irnich and Desaulniers [97]. There are three main types of exact approaches to resource constrained shortest path problems: constraint programming, branch and bound, and enumeration algorithms. Constraint programming algorithms [48, 70, 90, 104, 151] for resource constrained shortest path problem follow the usual principles of constraint programming [14], and combine specifically designed search, domain reduction, and propagation algorithms. Junker et al. [104] and Gualandi and Malucelli [90] provide constraint programming frameworks for resource constrained shortest path problems. Specific branching patterns have been developed by branch and bound algorithms for resource constrained shortest path problems. They branch on cycles, on arcs, and on resources [96, 148]. The bounds for these branch and bounds approaches are obtained either using heuristic enumeration algorithms [148], or using Lagrangean relaxation on an integer programming formulation of the problem [23, 30]. Finally, enumeration algorithms are variants of our enumeration algorithm. They also follow a branch and bound principle when only bounds are used to discard paths in these algorithms. As they only extend paths, enumeration algorithms are nonetheless completely different from the branch and bound algorithms mentioned above, which branch on arcs, cycles, and resources.

We now make a detailed review of enumeration algorithms, and start by some remarks on terminology. Enumeration algorithms are generally referred as dynamic programming algorithms in the resource constrained shortest path community [97, 147], because the first

versions [54] can be seen as generalization of the Ford Bellman algorithm [25, 77]. As these algorithms do not solve the dynamic programing equation, we have chosen not to call them dynamic programming but enumeration algorithms in order to avoid confusion with our bounding algorithms. They are also called labeling algorithm. Using the terminology of Ahuja et al. [9] introduced at the beginning of the section, most approaches to the resource constrained shortest path problem are label correcting algorithms, even if label setting algorithms have been developed under restrictive assumptions on the monotonicity of the resource extension function [55]. Desrosiers et al. [58] provide a review of those approaches. We now focus on label correcting enumeration algorithms. As we already noted, in the context of label correcting enumeration algorithms, label refer to paths, and they are therefore not label algorithms in the sense of Ahuja et al. [9].

In their survey on resource constrained shortest path problems, Irnich and Desaulniers [97] provide a generic enumeration algorithm, and describe the various contributions in the literature as special cases of this generic algorithm. For the reasons mentioned in the previous paragraph, they call it the generic dynamic programming resource constrained shortest path algorithm. This algorithm is the analogue of our generic enumeration algorithm in the resource extension function framework. The differences between the algorithms available in the literature lie in the choice of the key, of the bounds, and of the dominance rule. The key defines the order in which paths are processed. The bounds and the dominance rules are what enable to discard paths. The dominance rule is the order on the set of resources and the algorithm to check if a resource dominates another one. Desrochers and Soumis [55] and [146] provide specific keys for routing problems with time windows, but these strategy apply to most resource constrained shortest path problem. Irnich [96] provide general techniques to define resource extension functions leading to good dominance rules, and techniques to handle path with identical resources [98]. The remaining of the techniques are problem-specific. Finally, we note that variants of the enumeration algorithm based on the k-shortest path problem [68] have been proposed [23, 92, 157].

Several bounding techniques have been proposed when $(\mathscr{R}, \oplus, \leqslant)$ is $\mathbb{R}^n$ endowed with its product order and sum. Some contributions computed them in a preprocessing algorithm using component by component shortest path algorithms [55, 61, 102, 120]. Another branch of the literature uses Lagrangian relaxation on an integer formulation of the problem to obtain lower bounds [32, 61, 92, 157]. These methods require the absence of negative cost cycles, in order to be able to solve the Langrangean relaxation using a shortest path problem. The case with negative cost cycles was considered by Feillet et al. [72]. The difference between these algorithms and the branch and bound algorithms mentioned at the beginning of the section is the branching procedure: here, an enumeration algorithm is used, which only extends paths. We now come to dominance rules on specific problems. Several specific dominance rules have been developed for the elementary path constraint [23, 72]. Kohl et al. [107] and Larsen [113] provide dominance rules for the 2-cycle constraint, and Irnich and Villeneuve [98] for the k-cycle. Finally, Ioachim et al. [95] provide ad-hoc dominance rules for time windows constraints.

5 Applications to stochastic path problems

This chapter is devoted to the study of offline stochastic versions of the STANDARD SHORTEST PATH PROBLEM and of the STANDARD RESOURCE CONSTRAINED SHORTEST PATH PROBLEM within the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework. The notions of probability theory used in this chapter are introduced in Chapter 2.

In many applications of the STANDARD SHORTEST PATH PROBLEM, the resource q_a of an arc is uncertain. Congestion on road networks is a natural example. Mathematically, arc resources become random variables ξ_a on a probability space $(\Omega, \mathscr{F}, \mu)$. In this context, the length of a path $\xi_P = \sum_{a \in P} \xi_a$ is also a random variable. One way to quickly evaluate the quality of a path is to use a version independent probability functional ρ, which maps random variables to reals. We recall from Chapter 2 that a probability functional ρ is *version independent* if $\rho(\xi)$ only depends on the image measure μ^ξ induced by ξ, and that it is *monotone* if it is isotone with respect to the almost sure order: $\rho(\xi) \leq \rho(\tilde{\xi})$ if $\xi \leq \tilde{\xi}$. The following problem is an analogue of the STANDARD SHORTEST PATH PROBLEM where the arc resources are random variables, and the cost function c is a probability functional.

STOCHASTIC SHORTEST PATH PROBLEM
Input. A digraph $D = (V, A)$, two vertices $o, d \in V$, a collection (ξ_a) of real random variables on $(\Omega, \mathscr{F}, \mu)$, and a version independent monotone probability functional c.
Output. An o-d path P with minimum $c\left(\sum_{a \in P} \xi_a\right)$.

On other problems of practical interest, uncertainty lies in the constraints of a STANDARD RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. For instance, probabilistic constraints enable to model the quality of service in public transportation systems. The following problem is a stochastic analogue of the STANDARD RESOURCE CONSTRAINED SHORTEST PATH PROBLEM

STOCHASTIC RESOURCE CONSTRAINED SHORTEST PATH PROBLEM
Input. A digraph $D = (V, A)$, two vertices $o, d \in V$, a collection (ξ_a) of real random variables on $(\Omega, \mathscr{F}, \mu)$, a collection of real costs c_a, and a version independent monotone probability functional ρ.
Output. An elementary o-d path P such that $\rho\left(\sum_{a \in P} \xi_a\right) \leq 0$ with minimum $\sum_{a \in P} c_a$.

We underline the fact that both problems are *offline*, i.e. their solution path is computed

once and for all before the observation of the random variables, and not adapted once some random variables are observed. We also note that ρ is in this chapter a mapping to $\mathbb{R}$, and not a mapping to $\{0,1\}$ as the infeasibility function of the previous chapter. An infeasibility function can be obtained by taking the indicator function of $\rho(\cdot)$ being greater than 0.

The literature on stochastic path problems focuses mainly on three types of probability functionals. The first type is composed of probability functionals $\mathbb{P}(\cdot > \tau)$ for a given threshold τ. It has been widely studied because it corresponds to the probability of arriving on time. The second type is composed of expectations of given cost functions $\mathbb{E}\big(f(\cdot)\big)$. In this chapter, we restrict ourselves to non-decreasing cost functions f. Finally, the last type of probability functional is composed of the risk measures, which have been defined in Chapter 2, as they are standard tools to deal with uncertainty.

We develop the following strategy to tackle with the STOCHASTIC SHORTEST PATH PROBLEM: first, we endow the set of ξ_a with a lattice ordered monoid structure such that the probability functionals c and ρ are isotone with respect to the lattice order, and second we apply the algorithms of Chapter 4 to solve it. This chapter therefore considers lattice ordered structure on sets of random variables ξ_a and probability functionals isotone with respect to these orders. Note that this method naturally extends to the STOCHASTIC RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. Indeed, suppose that $(\mathscr{R}, \oplus, \leqslant)$ is a lattice ordered monoid encoding the set of ξ_a, then a pair (c_a, ξ_a) is an element of the product $(\mathbb{R}, +, \leq) \times (\mathscr{R}, \oplus, \leqslant)$, which is also a lattice ordered monoid. See Section 3.4 for more details on lattice ordered monoid products.

The next lemma shows that the set of random variables is naturally endowed with a lattice ordered monoid structure.

Lemma 5.1. *The space of random variables endowed with its usual sum and with the almost-sure order is a lattice ordered monoid.*

Proof. We only have to prove that the almost sure order $\leq$ is compatible with random variables sum, and that it induces a lattice structure. Let ξ, ξ_1 and ξ_2 be three random variables such that $\xi_1 \leq \xi_2$. We have $\xi_1(\omega) \leq \xi_2(\omega)$ for each element $\omega \in \Omega$ except on a negligible set, which induces $\xi_1(\omega) + \xi(\omega) \leq \xi_2(\omega) + \xi(\omega)$, and thus $\xi_1 + \xi \leq \xi_2 + \xi$, which gives the compatibility of the almost sure order with the sum. The lattice structure follows from the fact that the meet of two random variables ξ and $\tilde{\xi}$ is their minimum $(\xi \wedge \tilde{\xi})(\omega) = \min(\xi(\omega), \tilde{\xi}(\omega))$. □

Nonetheless, the elementary operations performed by the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM algorithms of Chapter 4 are the sum and the meet of ξ_a, whose complexities depend on the definition of $(\Omega, \mathscr{F}, \mu)$.

When Ω is finite and of reasonable size, the computation of sums and meets of random variables for the almost sure order is tractable. Section 5.3 shows that in this case, the algorithms of Chapter 4 enable to solve the STOCHASTIC SHORTEST PATH PROBLEM on instances of medium size, and Chapter 6 introduces enhanced versions of these algorithms which enable to solve the STOCHASTIC SHORTEST PATH PROBLEM with finite Ω on large instances. The importance of this results comes from the fact that, using the Monte-Carlo method, any random variable ξ can be approximated by a sampled version $\hat{\xi}$ on a finite event probability space. To the best of

our knowledge, this is the first practical solution scheme for the STOCHASTIC SHORTEST PATH PROBLEM when arc random variables ξ_a are not independent.

When Ω is infinite, computing sums and meets of random variables for the almost sure order is challenging. Nonetheless, as ρ is assumed to be version independent, we only need the law of the random variable ξ_P to evaluate $\rho(\xi_P)$. Besides, if the ξ_a are assumed to be independent and belong to a family of distribution $\mathbb{F}$ stable by convolution product $*$, then the $\xi_P = \sum_{a\in P} \xi_a$ belongs to $\mathbb{F}$ for all paths P. As a consequence, all the computations of the STOCHASTIC SHORTEST PATH PROBLEM can be done in the set $\mathbb{F}$. If we turn $(\mathbb{F}, *)$ into a lattice ordered monoid thanks to a properly defined order, we can use the algorithms of Chapter 4 to solve the problem.

The chapter is organized as follows:

- Section 5.1 focuses on the case of independent ξ_a. Lattice ordered monoids are introduced for several families of distributions stable by convolution product, including discrete distributions with finite support, normal distributions, Gamma distributions and Poisson distributions. The normal and discrete distributions case are dealt with in details. Their complexity is analyzed. A link is established between the bounds of Section 4.2 and an online stochastic path problem in the discrete distributions case. Finally, the efficiency of the algorithms of Chapter 4 in the STOCHASTIC SHORTEST PATH PROBLEM context is proved through extensive numerical experiments.
- Section 5.3 focuses on non-independent scenario based distributions. We prove that for most probability functionals c of interest, there is no polynomial algorithm for the STOCHASTIC SHORTEST PATH PROBLEM when $|\Omega|$ is finite. The convergence of the Monte-Carlo approximation of any STOCHASTIC SHORTEST PATH PROBLEM problem is studied. Finally, numerical experiments are carried out.
- Section 5.5 contains bibliographical remarks.

The three types of probability functionals mentioned above are shown to be isotone with respect to the order of the lattice ordered monoids introduced.

5.1 Independent distribution

This section focuses on the independent ξ_a case. As explained in the introduction, the hypothesis that probability function c is version independent enables us to work on random variables laws when ξ_a are independents. For several families of distributions $\mathbb{F}$ stable by convolution product $*$, we define an order $\leqslant$ such that $(\mathbb{F}, *, \leqslant)$ is a lattice ordered monoid. Most of the section is devoted to the family of discrete distributions endowed with the usual stochastic order, which is introduced in Section 2.1.5. Several parametric families of continuous distributions are considered in Section 5.1.2.

In this dissertation, we denote F_ξ the *cumulative distribution* of a random variable ξ.

5.1.1 Discrete distributions lattice ordered monoid

Recall from Section 2.1.5 that a random variable ξ is non greater than a random variable $\tilde{\xi}$ for the *usual stochastic order*, denoted $\xi \leq_{\text{st}} \tilde{\xi}$ if

$$\mathbb{P}(\xi \leq t) \geq \mathbb{P}(\tilde{\xi} \leq t) \quad \text{for all } t. \tag{5.1}$$

The proof of the following lemmas can be found in [131, 160].

Lemma 5.2. *The usual stochastic order $\leq_{\text{st}}$ is compatible with the convolution product. Given three random variables ξ, $\tilde{\xi}$ and ξ', with ξ' independent from ξ and $\tilde{\xi}$, we have*

$$\xi \leq_{\text{st}} \tilde{\xi} \Longrightarrow \xi + \xi' \leq_{\text{st}} \xi + \tilde{\xi}.$$

Lemma 5.3. *The set of laws of real random variables endowed with the convolution product and the usual stochastic order is a lattice ordered monoid.*

With a slight abuse of notation, we denote $\xi \wedge_{\text{st}} \tilde{\xi}$ (resp. $\xi \vee_{\text{st}} \tilde{\xi}$) a random variable whose distribution is the meet (resp. the join) of the distributions of ξ and $\tilde{\xi}$. Given two variables ξ and $\tilde{\xi}$, we have

$$F_{\xi \wedge_{\text{st}} \tilde{\xi}} = \max\left(F_\xi, F_{\tilde{\xi}}\right) \quad \textit{and} \quad F_{\xi \vee_{\text{st}} \tilde{\xi}} = \min\left(F_\xi, F_{\tilde{\xi}}\right).$$

Proposition 2.3 ensures that any version independent monotone risk measure is isotone with respect to the usual stochastic order. The isotony of $\mathbb{P}(\cdot > \tau)$ and the isotony $E(f(\cdot))$ for non-decreasing f follow directly from the definition of $\leq_{\text{st}}$.

From a practical point of view, the maximum of two cumulative distributions can be computed efficiently only when these distributions have finite support in $\mathbb{Z}$. Besides, the convolution products can also be computed efficiently on the set of distributions with finite support in $\mathbb{Z}$.

Let $\mathbb{M}$ denote the set of distributions with finite support in $\mathbb{Z}$. When the random variables ξ_a are independent with finite support in $\mathbb{Z}$, the lattice ordered monoid $(\mathbb{M}, *, \leq_{\text{st}})$ can be used in practice to solve the STOCHASTIC SHORTEST PATH PROBLEM with the algorithms of Chapter 4. The numerical experiments in Section 5.2 show that they solve it very efficiently.

The relative easiness of this problem asks the question of its complexity. The following theorem shows that there is no polynomial algorithm for the STOCHASTIC SHORTEST PATH PROBLEM.

Theorem 5.4. *Even if the random variables ξ_a are independent with finite support in $\mathbb{Z}$, there is no polynomial algorithm with a complexity function independent of c solving the* STOCHASTIC SHORTEST PATH PROBLEM, *unless $\mathscr{P} = \mathscr{NP}$.*

In the proof, we use the terminology STOCHASTIC $\overline{c}$-SHORTEST PATH PROBLEM instead of STOCHASTIC SHORTEST PATH PROBLEM to underline the choice of a specific probabilistic functional $\overline{c}$ as the cost function of the STOCHASTIC SHORTEST PATH PROBLEM instance built.

The proof of Theorem 5.4 is a reduction of the $\mathscr{NP}$-complete STANDARD RESOURCE CONSTRAINED SHORTEST PATH PROBLEM to a STOCHASTIC SHORTEST PATH PROBLEM. See Chapter 3 for the definition of the STANDARD RESOURCE CONSTRAINED SHORTEST PATH PROBLEM.

Proof. Let digraph $D = (V, A)$, origin vertex o, destination vertex d, costs c_a for a in A, weights w_a for a in A, and the threshold τ be an instance of the STANDARD RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. We define two risk measures $c_{\min}$ and $c_{\max}$.

$$\begin{aligned} c_{\min}(\xi) &= \min\{t \in \mathbb{Z}_+ : F_\xi(t) > 0)\} \\ c_{\max}(\xi) &= \begin{cases} 0 & \text{if } \max\{t \in \mathbb{Z}_+ : \mathbb{P}(\xi = t) > 0\} \leq M \\ 1 & \text{otherwise,} \end{cases} \end{aligned}$$

where $M = Q(1 + \max_{a \in A} c_a)$. We define

$$\overline{c} = c_{\min} + \left(\sum_{a \in A} c_a \right) c_{\max}.$$

$c_{\min}$ is isotone with respect to the usual stochastic order because $F_\xi(t) \geq F_{\tilde{\xi}}(t)$ for all t implies $\min\{t \in \mathbb{Z}_+ : F_\xi(t) > 0)\} \leq \min\{t \in \mathbb{Z}_+ : F_{\tilde{\xi}}(t) > 0)\}$. Besides, $c_{\max}$ is also isotone with respect to the usual stochastic order because $\max\{t \in \mathbb{Z}_+ : \mathbb{P}(\xi = t) > 0\} = \min\{t \in \mathbb{Z}_+ : F_\xi(t) = 1\}$ and $F_\xi(t) \geq F_{\tilde{\xi}}(t)$ for all t implies $\min\{t \in \mathbb{Z}_+ : F_\xi(t) = 1)\} \leq \min\{t \in \mathbb{Z}_+ : F_{\tilde{\xi}}(t) = 1)\}$. Thus $\overline{c}$ is also isotone with respect to the usual stochastic order.

We describe now the reduction to the STOCHASTIC $\overline{c}$-SHORTEST PATH PROBLEM. Given an arc a, we define ξ_a as follows:

$$\mathbb{P}(\xi_a = c_a) = \frac{1}{2} \quad \text{and} \quad \mathbb{P}\left(\xi_a = q_a \left(1 + \max_{b \in A} c_b\right)\right) = \frac{1}{2}.$$

ξ_a can only take two values.

The deterministic resource constrained shortest path problem has a feasible solution if and only if the STOCHASTIC $\overline{c}$-SHORTEST PATH PROBLEM has a solution of cost less than $\sum_{a \in A} c_a$. In this case, the optimal solutions of both problems coincide.

We assume that the deterministic resource constrained shortest path problem has a feasible solution. Let P be an optimal solution of the STOCHASTIC $\overline{c}$-SHORTEST PATH PROBLEM. We have then $\overline{c}(\sum_{a \in P} X_a) = \sum_{a \in P} c_a$ and $\sum_{a \in P} q_a \leq Q$, which implies that the deterministic resource constraint shortest path problem has a feasible solution of cost $\sum_{a \in P} c_a$. Conversely, an optimal solution P' of the deterministic resource constrained shortest path problem provides a feasible solution of the STOCHASTIC $\overline{c}$-SHORTEST PATH PROBLEM of cost $\sum_{a \in P'} c_a$. □

The lattice ordered monoid $(\mathbb{M}, *, \leq_{\text{st}})$ is an interesting example from the point of view of the complexity of the algebraic path problem. In the case of lattice ordered monoids that are not dioids, Theorem 4.12 and Proposition 4.15 ensure that we can compute the lower bound $b_v^{\ell^*}$ in polynomial time but not the solution $b_v^\dagger$ of the generalized dynamic programming equation Equation (4.8). The set $(\mathbb{M}, *, \leq_{\text{st}})$ is an example of lattice ordered monoid that is not a dioid and for which we can compute $b_v^\dagger$ in polynomial time. Indeed, we prove in [142] that a variant of the algorithm of Proposition 4.15 where a key $\phi(b_v)$ is used instead of $\tilde{n}_v$ converges to $b_v^\dagger$ after a number of iterations that is polynomial in $|V|$ and in the size of the support of ξ_a.

5.1.2 Parametrized families of distribution

We now explore STOCHASTIC SHORTEST PATH PROBLEM where ξ_a are independent with distribution in parametrized families $\mathbb{F}$ of continuous distributions. To be used in the context of STOCHASTIC SHORTEST PATH PROBLEM, a family $\mathbb{F}$ must be stable by convolution product.

Normal distributions

The convolution of two normal distributions $\mathcal{N}(\mu,\sigma^2)$ and $\mathcal{N}(\tilde{\mu},\tilde{\sigma}^2)$ is a normal distribution $\mathcal{N}(\mu+\tilde{\mu},\sigma^2+\tilde{\sigma}^2)$. As a consequence, the case of normal distributions can be dealt with using the monoid $(\mathbb{R}^2,+)$ endowed with properly defined orders.

The two following orders are compatible with + and endow $\mathbb{R}^2$ with a lattice ordered structure.

$$(\mu,\sigma)\leqslant_1(\tilde{\mu},\tilde{\sigma}) \quad \text{if} \quad \begin{cases} \mu\leq\tilde{\mu}, \\ \sigma\leq\tilde{\sigma}, \end{cases} \quad \text{and} \quad (\mu,\sigma)\leqslant_2(\tilde{\mu},\tilde{\sigma}) \quad \text{if} \quad \begin{cases} \mu\leq\tilde{\mu}, \\ \sigma\geq\tilde{\sigma}. \end{cases} \tag{5.2}$$

When evaluating them on random variables $\mathcal{N}(\mu,\sigma)$, risk-averse probability functionals c such as the expectation of convex cost functions, or the value at risk VaR_β for $\beta\in[0.5,1]$, are isotone with respect to $\leqslant_1$. On the contrary risk-prone probability functionals such as the expectation of concave cost function and the value at risk VaR_β with $\beta\in[0,0.5]$ are isotone with respect to order $\leqslant_2$.

An important probability functional that is not isotone with respect to $\leqslant_1$ or $\leqslant_2$ is $\mathbb{P}(\cdot>\tau)$ for a given threshold τ. This probability functional can be dealt with by using $(\mu,\underline{\sigma},\overline{\sigma})\in\mathbb{R}^3$ as parameter. We then define the order and functional

$$(\mu,\underline{\sigma},\overline{\sigma})\leqslant_3(\tilde{\mu},\underline{\tilde{\sigma}},\overline{\tilde{\sigma}}) \quad \text{if} \quad \begin{cases} \mu\leq\tilde{\mu}, \\ \underline{\sigma}\leq\underline{\tilde{\sigma}}, \\ \overline{\sigma}\geq\overline{\tilde{\sigma}}, \end{cases} \quad \text{and} \quad c(\mu,\underline{\sigma},\overline{\sigma})=\begin{cases} \mathbb{P}\left(\mathcal{N}(\mu,\underline{\sigma})>\tau\right) & \text{if } \tau\leq\mu, \\ \mathbb{P}\left(\mathcal{N}(\mu,\overline{\sigma})>\tau\right) & \text{otherwise.} \end{cases}$$

The order $\leqslant_3$ is compatible with +, and the function c is isotone with respect to $\leqslant_3$. If for each arc a, ξ_a has normal distribution $\mathcal{N}(\mu_a,\sigma_a)$, we define $q_a=(\mu,\sigma_a,\sigma_a)$, then we have $c(q_P)=\mathbb{P}(\xi_P>\tau)$, and we have expressed the STOCHASTIC SHORTEST PATH PROBLEM in the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework.

Given a distortion functional ρ_σ associated with a distortion function σ, we define $\tilde{\rho}_\sigma$ on $(\mathbb{R}^3,+,\leqslant_3)$ as follows

$$\tilde{\rho}_\sigma(\mu,\underline{\sigma},\overline{\sigma})=\int_0^{\frac{1}{2}}\sigma(u)F^{-1}_{\mu,\underline{\sigma}}(u)\mathrm{d}u+\int_{\frac{1}{2}}^1\sigma(u)F^{-1}_{\mu,\overline{\sigma}}(u)\mathrm{d}u.$$

Thus defined, $\tilde{\rho}_\sigma$ is compatible with $\leq_3$, and we have $\tilde{\rho}_\sigma(\mu,\sigma^*,\sigma^*)=\rho_\sigma\left(\mathcal{N}(\mu,\sigma^*)\right)$. Corollary 2.2 then ensures that that we can define a generalization $\tilde{\rho}$ of any risk measure ρ on $\mathbb{R}^3$ that is isotone with respect to $\leq_3$ and such that $\tilde{\rho}(\mu,\sigma^*,\sigma^*)=\rho\left(\mathcal{N}(\mu,\sigma^*)\right)$.

Remark 5.1. The proof of Theorem 5.4 can be adapted to the normal distributions framework, the cost of a path being encoded in the mean and the weight in the variance multiplied by the sum of the cost of arcs. The only technical point is the definition of the analogue of $c_{\max}$ as 1 if

and only if the probability of being greater than a threshold is larger than a given value, the choice of the threshold and the value being such that the resource constraint is enforced and the space required by encoding them is polynomial.

Distributions with one parameter

Among other distributions stable by convolution, we have the binomial distribution $\mathscr{B}(n,p)$, the negative binomial distribution $\mathscr{N}(n,p)$ and the Pascal distribution $\mathscr{P}(n,p)$, all of them at fixed p, the Poisson distribution, the Gamma distribution, the Cauchy distribution, and the Lévy distribution, all of them with fixed scale parameter.

With the parameter restriction mentioned above, all of these distributions have only one parameter. Most probability functionals are monotone with respect to this parameter. As a consequence, the resolution of the STOCHASTIC SHORTEST PATH PROBLEM with independent ξ_a with distributions in these families is equivalent to the resolution of a STANDARD SHORTEST PATH PROBLEM where the cost of an arc is the parameter of its distribution.

Among these single parameter families of distributions, positive distributions such as the Poisson distribution and the Gamma distribution can be used to model "delay". We can indeed suppose that $\xi_a = \alpha_a + \tilde{\xi}_a$, where α_a is a constant modeling arc length, and $\tilde{\xi}_a$ a random variable representing delay. Supposing that $\tilde{\xi}_a$ follows a Poisson or a Gamma distribution, we obtain a two parameter family stable by convolution. The techniques shown above to deal with normal distributions in the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework apply to these two parametrized families of continuous distributions.

5.1.3 Link with the online on time arrival problem

When the lattice ordered monoid $(\mathbb{M}, *, \leq_{\text{st}})$ of discrete distributions endowed with the usual stochastic order is used, the bound $\mathbf{b}^\dagger$ defined in Equation (4.8) admits a nice interpretation as the optimal solution of an online problem.

In the offline STOCHASTIC SHORTEST PATH PROBLEM, a traveler chooses his entire path a priori, when the realizations of ξ_a are unknown. Suppose that instead of choosing a priori the path he will use, a traveler can update his choice given the time and his position. A solution of this new problem is a policy $\pi : V \times T \to A$, where $\pi(v,t)$ is the arc that a traveler takes when he is in v at t. Given a graph D, and origin o, a destination vertex d, positive independent random variables ξ_a with discrete support, and a threshold τ, the STOCHASTIC ON TIME ARRIVAL PROBLEM consists in finding a policy maximizing the probability of arrival at d before τ.

Note that if $t > \tau$, then the probability of arriving before τ is equal to 0. As a consequence, an optimal policy must be defined only for $t \leq \tau$ [156].

Proposition 5.5. *Let $(b_v^\dagger)$ be the greatest solution of Equation* (4.8)*. Then the policy which assigns to (v,t) an arc (v,u) in $\delta^+(v)$ maximizing $\sum_{k=0}^{\tau-t} \mathbb{P}(b_u \leq \tau - t - k)\mathbb{P}(\xi_{(v,u)} = k)$ is an optimal policy of the* STOCHASTIC ON TIME ARRIVAL PROBLEM*.*

Proof. Following the dynamic programming principle, if we denote $F_v(t)$ the probability of

arriving before τ in d starting from v at t under an optimal policy, we have

$$F_v(t) = \max_{(v,u)\in\delta^+(v)} \sum_{k=0}^{\tau-t} \mathbb{P}(\xi_{(v,u)} = k) F_u(t+k),$$

If we denotes b_v a random variable with cumulative distribution $F_{\xi_v}(t) = F_v(\tau - t)$, we have

$$\begin{cases} b_d = 0, \\ b_v = \bigwedge_{\text{st}} \xi_{v,u} + b_u, \end{cases}$$

which is exactly Equation (4.8), and gives the proposition. To be perfectly rigorous, the equation above is satisfied only in term of distributions and for $t \leq \tau$, but this does not affect the policy defined in the proposition. □

5.2 Numerical results for independent distributions with discrete support

5.2.1 Instances, resources, and constraints used

In this section, we test the performance of the algorithms of Chapter 4 as solution methods for the STOCHASTIC SHORTEST PATH PROBLEM and the STOCHASTIC RESOURCE CONSTRAINED SHORTEST PATH PROBLEM when random variables distributions belong to $(\mathbb{M}, *, \leq_{\text{st}})$. We build instances by generating resources on the road, square grid, long grid, and wide grid digraphs introduced in Section 4.3.1.

On each digraph, two instances have been generated for the STOCHASTIC SHORTEST PATH PROBLEM: on with generic discrete distributions with finite support, and one with truncated lognormal distributions. The *generic* distributions are generated as follows. First, the length of the arcs are rescaled to be non greater than 200. Second, their support size is chosen as 10 plus a random integer between 0 and the scaled length of the arc. Then $\mathbb{P}(\xi = t)$ is randomly chosen using a uniform distribution on $[0, 1]$ for each t in the support, and then normalized to obtain the support. The measure is then normalized to obtain a probability distribution. The truncated and discretized *lognormal* distributions have been generated as follows. The expectation of an arc is chosen as 10 plus a random integer between 0 and the scaled length of the arc. The probability functionals $\mathbb{P}(\cdot > \tau)$ and CVaR_β have been tested for different values of τ and β. We have chosen τ as follows: a first parameter τ^{-1} is chosen between 0 and 1, then τ is chosen as the smallest t such that $\mathbb{P}(b_o > \tau) \leq \tau^{-1}$ where b_o is the bound provided by the bounding algorithm for the origin vertex o. It enables to obtain a threshold τ such that $\mathbb{P}(\xi_P > \tau) \sim \tau^{-1}$ where ξ_P is the resource of an optimal path.

For the STOCHASTIC RESOURCE CONSTRAINED SHORTEST PATH PROBLEM, the deterministic cost has been randomly chosen between 1 and 100, and the constraint probability distributions and probability functionals have been generated in the same way as the distributions of the STOCHASTIC SHORTEST PATH PROBLEM.

The A*, the label correcting, and the label dominance are tested. In each of these algorithms, we use the candidate paths mentioned in Section 4.3.3. We use our `latticeRCSP` library

described in Appendix C to solve the instances. The algorithms are not parallelized, and the numerical experiments are performed on a Macbook Pro of 2012 with four 2.5 Ghz processors and 4 Gb of ram.

5.2.2 Results on the STOCHASTIC SHORTEST PATH PROBLEM

We now give the numerical results for the STOCHASTIC SHORTEST PATH PROBLEM

$$\min_{P \in \mathscr{P}_{od}} \text{CVaR}_\beta(\sum_{a \in P} \xi_a), \tag{5.3}$$

where $\mathscr{P}_{od}$ is the set of o-d paths. Figure 5.1 provides the algorithms CPU time for the STOCHASTIC SHORTEST PATH PROBLEM using CVaR_β with $\beta = 0.01$ as probability functional c. The plain lines correspond to generic distributions and the dashed lines to lognormal distributions. Only instances solved to optimality are plotted on the graph.

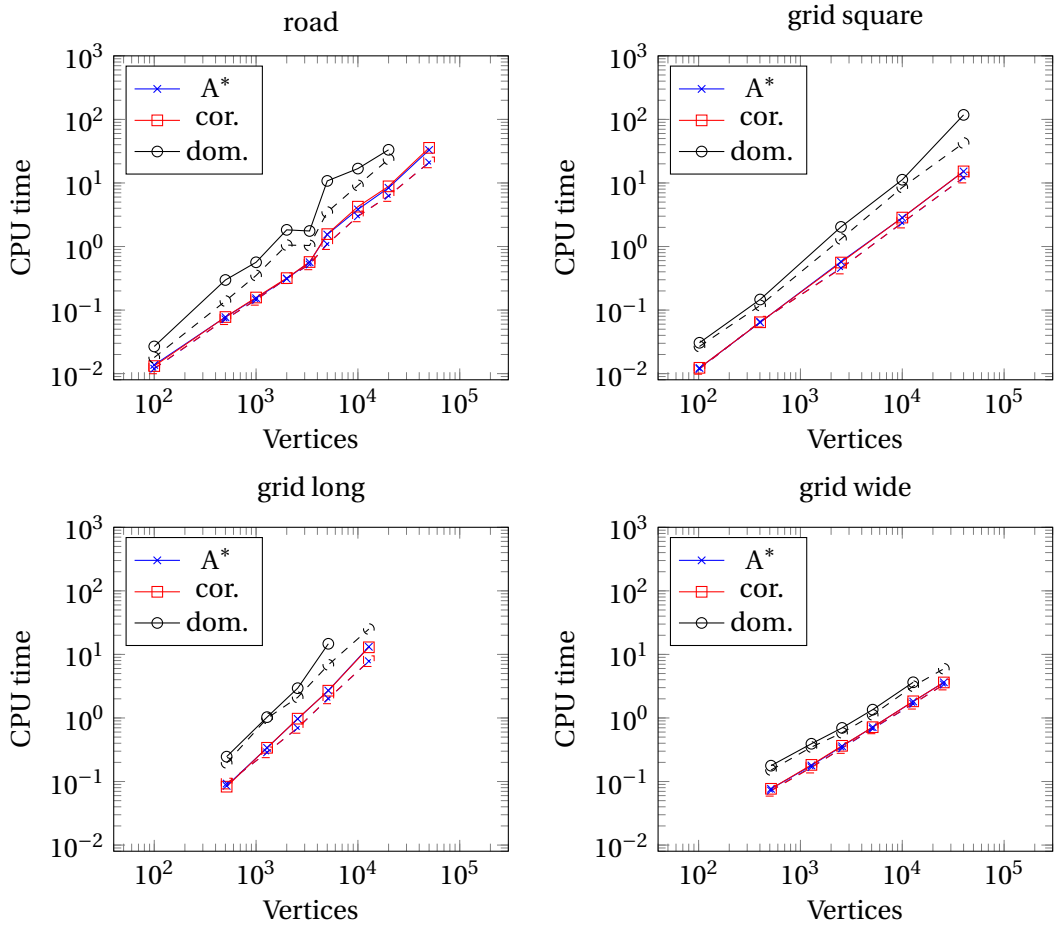


Figure 5.1 – Algorithms performances on stochastic shortest path problem using CVaR_β with $\beta = 0.01$ as probability functional. Plain lines correspond to generic distributions and dashed lines to truncated lognormal distributions

Table 5.1 provides numerical results on rather large road, square grid, long grid, and wide grid instances for each algorithm tested and for parameters β in $\{0.01, 0.1\}$. The first column provides the instance name. The four next columns provide the number of vertices and the number of arcs of the instance, the parameter β of the CVaR_β, the probability distribution of the instance, and the algorithm used. The name of the algorithms are identical to those of Table 4.3. Prefix gen corresponds to the generic distributions, and log to the truncated lognormal distributions. The next two column provide the ratio γ of Remark 4.5 and the percentage of time spent computing bounds in the preprocessing. The three next columns provide the number of paths extended, the number of paths cut, and the proportion of paths cut by the dominance test for the label correcting algorithm. The next columns provide the number of arcs in the solution returned and the optimality gap. We use the prefix opt instead of the optimality gap when the instance is solved to optimality. Finally, the last column provides the total CPU time of the algorithm.

We first remark that the distribution has little impact on the performance of the algorithms. We have also tested them on the same instances with truncated gamma distributions and obtained similar results.

The STOCHASTIC SHORTEST PATH PROBLEM is rather easy to solve even for instances with more than 10^4 vertices. The fastest algorithm are the extended A* and the label correcting algorithms. The limit on the size of the instance we are able to solve using these algorithms is the memory needed to store these instances. Both of these algorithms are between 2 and 10 times faster than the usual label dominance algorithm. Besides, they enable to solve larger instances. There are two reasons for that. First, the bound test cuts path much better than the dominance test on these instances. Indeed all paths are cut by the bound test for the label correcting algorithm. This explains why the label correcting and the A* algorithms have the same performance. As the bound test cuts paths very well, the number of paths extended by the A* and the label correcting algorithms is several order of magnitude smaller than the number of paths extended by the label dominance algorithm. Another consequence of that fact is that most of the CPU time of the A* and the label correcting algorithms is spent computing bounds in the preprocessing. The second reason for the algorithms good performance is that the generalized Dijkstra algorithm computes the bounds very efficiently. Indeed, the ratio γ of the number of extension divided by the number of vertices introduced in Remark 4.5 is close to its minimal value 1. We note that we use the expectation of the current resource of the vertex in the Dijkstra algorithm instead of the path length. Using the path length leads to γ between 10 and 50 on the instances of Table 5.1, which strongly affects the overall performance of the A* and the label correcting algorithms.

Considered the instance not solved to optimality in Table 5.1, we note that the smaller β, the harder the problem is. This result is not surprising, because large values of β correspond to conditional value at risk close to the expectation, while small β correspond to more risk averse measures. Similar results are obtained when using the probability functional $\mathbb{P}(\cdot > \tau)$ instead of the conditional value at risk. This probability function is considered in the next section on the STOCHASTIC RESOURCE CONSTRAINED SHORTEST PATH PROBLEM.

Instance	$\|V\|$	$\|A\|$	β	Dis.	Alg.	γ	Prep.	Ext.	Cut.	Dom.	ℓ	Gap	CPU (s)
road50	50000	138112	0.01	gen	A*	1.003	64%	281	593	–	468	opt	3.30e+01
					cor.	1.003	63%	281	593	0%	468	opt	3.57e+01
					dom.	–	–	147223	97217	–	469	139.9%	7.84e+01
			0.1		A*	1.003	65%	281	614	–	468	opt	3.28e+01
					cor.	1.003	64%	281	614	0%	468	opt	3.48e+01
					dom.	–	–	147202	97201	–	469	140.5%	7.93e+01
			0.01	log	A*	1.001	65%	320	712	–	504	opt	2.10e+01
					cor.	1.001	65%	320	712	0%	504	opt	2.11e+01
					dom.	–	–	148462	98283	–	509	120.7%	4.38e+01
			0.1		A*	1.001	65%	311	699	–	509	opt	2.11e+01
					cor.	1.001	65%	311	699	0%	509	opt	2.11e+01
					dom.	–	–	147995	97987	–	509	121.0%	4.38e+01
square200	40002	120200	0.01	gen	A*	1.003	60%	149	481	–	261	opt	1.52e+01
					cor.	1.003	60%	149	481	0%	261	opt	1.52e+01
					dom.	–	–	420653	278478	–	261	opt	1.18e+02
			0.1		A*	1.003	60%	194	578	–	261	opt	1.53e+01
					cor.	1.003	60%	194	578	0%	261	opt	1.53e+01
					dom.	–	–	423298	280217	–	261	opt	1.19e+02
			0.01	log	A*	1.002	60%	155	500	–	262	opt	1.21e+01
					cor.	1.002	60%	155	500	0%	262	opt	1.22e+01
					dom.	–	–	228833	152005	–	262	opt	4.25e+01
			0.1		A*	1.002	60%	266	723	–	265	opt	1.23e+01
					cor.	1.002	60%	266	723	0%	265	opt	1.23e+01
					dom.	–	–	229133	152199	–	265	opt	4.26e+01
long50	12802	38416	0.01	gen	A*	1.002	51%	695	1291	–	1008	opt	1.32e+01
					cor.	1.002	52%	612	1123	0%	1008	opt	1.29e+01
					dom.	–	–	148528	98519	–	1010	112.1%	1.21e+02
			0.1		A*	1.002	52%	602	1199	–	1014	opt	1.28e+01
					cor.	1.002	52%	602	1199	0%	1014	opt	1.29e+01
					dom.	–	–	148495	98494	–	1018	112.7%	1.21e+02
			0.01	log	A*	1.001	54%	542	1067	–	1030	opt	7.81e+00
					cor.	1.001	54%	542	1067	0%	1030	opt	7.83e+00
					dom.	–	–	45989	30633	–	1030	opt	2.50e+01
			0.1		A*	1.001	53%	625	1256	–	1030	opt	7.95e+00
					cor.	1.001	53%	625	1256	0%	1030	opt	7.97e+00
					dom.	–	–	46470	30947	–	1030	opt	2.52e+01
wide100	25602	78400	0.01	gen	A*	1.004	61%	21	1631	–	21	opt	3.64e+00
					cor.	1.004	60%	21	1631	0%	21	opt	3.62e+00
					dom.	–	–	46326	28082	–	21	32.5%	5.22e+00
			0.1		A*	1.004	61%	1	1591	–	20	opt	3.59e+00
					cor.	1.004	61%	1	1591	0%	20	opt	3.60e+00
					dom.	–	–	45829	27752	–	20	33.7%	5.20e+00
			0.01	log	A*	1.003	61%	1	1589	–	20	opt	3.37e+00
					cor.	1.003	61%	1	1589	0%	20	opt	3.36e+00
					dom.	–	–	64916	40386	–	20	opt	5.94e+00
			0.1		A*	1.003	60%	1	1590	–	20	opt	3.36e+00
					cor.	1.003	60%	1	1590	0%	20	opt	3.38e+00
					dom.	–	–	64454	40078	–	20	opt	5.90e+00

Table 5.1 – STOCHASTIC SHORTEST PATH PROBLEM results

5.2.3 Results on the STOCHASTIC RESOURCE CONSTRAINED SHORTEST PATH PROBLEM

We now provide result for the STOCHASTIC RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. The problem solved is

$$\begin{aligned} \min_{P \in \mathscr{P}_{od}} \quad & \sum_{a \in P} c_a \\ \text{s.t.} \quad & \mathbb{P}\left(\sum_{a \in P} \xi_a > \tau \right) \leq \rho_o \end{aligned}$$

where c_a are deterministic and ξ_a are independent random variables with discrete and finite support, and $\mathscr{P}_{od}$ is the set of o-d paths. Resources (c_a, ξ_a) are generated as mentioned in Section 5.2.1. Parameter τ is chosen in a way that the minimal value of $\mathbb{P}(\sum_{a \in P} \xi_a > \tau)$ on the o-d paths is not too far from 0.5 using the technique mentioned in Section 5.2.1. The threshold parameter ρ_o is chosen as

$$\rho_0 = (1-\alpha)\mathbb{P}(\sum_{a \in P_\rho} \xi_a > \tau) + \alpha \mathbb{P}(\sum_{a \in P_c} \xi_a > \tau) \tag{5.4}$$

where P_c is the o-d path minimizing $\sum_{a \in P} c_a$ and P_ρ the path minimizing $\mathbb{P}\left(\sum_{a \in P_\rho} \xi_a > \tau\right)$. Figure 5.2 provides the CPU times for road network and grid instances for $\alpha = 0.5$ and $\alpha = 0.9$.

Table 5.2 is the analogue of Table 5.1 for the STOCHASTIC RESOURCE CONSTRAINED SHORTEST PATH PROBLEM studied in this section. The columns of Table 5.2 are identical to those of Table 5.1, except the column β, which is replaced by the constraint strength α of the probabilistic constraint (5.4).

Comparing Figure 5.2 and Figure 5.3 with Figure 5.1, we can see that the STOCHASTIC RESOURCE CONSTRAINED SHORTEST PATH PROBLEM is harder than the STOCHASTIC SHORTEST PATH PROBLEM. This is not surprising, as the deterministic resource constrained shortest path problem is much harder than the deterministic shortest path problem. Depending on the instances type, the standard label dominance algorithm enables to solve to optimality instances with hundreds or thousands of vertices. The use of lower bounds in the generalized A* and label correcting algorithms enable to solve to optimality instances with dozens of thousands of vertices. Besides, on the instances the label dominance algorithm, the A* and label correcting algorithms are between five and ten times faster than the label dominance algorithm. We mentioned in Section 4.3.6 that dominance becomes rare when "dimension" of the resource increases. Here, we can see that when the resource is the product of a real with a finite support distribution, dominance is rare, and the label dominance algorithm becomes inefficient. This "dimension effect" also affect the lower bound test, but less strongly. We also note that, on most family of instances, the label correcting algorithm is faster than the generalized A* and enables to solve larger instances. Finally, we remark that medium strength constraint with $\alpha = 0.5$ on Figure 5.2 are easier to solve than weak constraints with $\alpha = 0.9$ on Figure 5.3.

In Table 5.2, we note that γ of Remark 4.5 is larger than in the non-constrained case, even if it remains smaller than 1.6 on all instances studied. As in the deterministic case, the

Instance	$\|V\|$	$\|A\|$	τ	α	Dis.	Alg.	γ	Prep.	Ext.	Cut.	Dom.	ℓ	Gap	CPU (s)
road20	20000	55180	20095	0.5	gen	A*	1.4	9%	79207	166575	–	240	opt	85.4
						cor.	1.4	66%	1502	2576	9%	240	opt	11.5
						dom.	–	–	113537	63535	–	0	inf%	32.8
				0.9		A*	1.4	15%	49262	72656	–	244	17.3%	51.1
						cor.	1.4	50%	7359	9176	31%	237	opt	15.1
						dom.	–	–	113537	63535	–	0	inf%	33.1
				0.5	log	A*	1.5	7%	98184	213563	–	245	opt	93.8
						cor.	1.5	66%	1618	2859	9%	245	opt	9.6
						dom.	–	–	115757	65756	–	0	inf%	23.6
				0.9		A*	1.5	11%	62959	111126	–	244	13.2%	58.2
						cor.	1.5	61%	3714	4323	35%	244	opt	10.4
						dom.	–	–	115757	65756	–	0	inf%	23.7
square200	40002	120200	8963	0.5	gen	A*	1.5	15%	87822	165841	–	255	51.3%	89.1
						cor.	1.5	31%	30301	50234	11%	255	opt	43.1
						dom.	–	–	37691	21859	–	0	inf%	6.2
				0.9		A*	1.5	27%	41607	73411	–	248	63.8%	51.4
						cor.	1.5	21%	74965	97489	26%	244	29.6%	66.5
						dom.	–	–	37691	21859	–	0	inf%	6.3
				0.5	log	A*	1.5	12%	162320	314837	–	257	34.9%	95.1
						cor.	1.5	37%	29704	51436	8%	259	opt	31.0
						dom.	–	–	42443	25028	–	0	inf%	5.0
				0.9		A*	1.6	26%	60014	110224	–	257	61.2%	44.1
						cor.	1.6	26%	71873	98473	22%	255	19.8%	44.8
						dom.	–	–	42443	25028	–	0	inf%	5.0
long20	5122	15376	14701	0.5	gen	A*	1.5	2%	115855	221722	–	395	67.0%	103.8
						cor.	1.5	2%	62412	99413	12%	395	7.4%	90.0
						dom.	–	–	140783	90781	–	0	inf%	17.9
				0.9		A*	1.5	11%	19386	28782	–	388	76.2%	20.1
						cor.	1.5	2%	97634	97996	49%	389	32.8%	88.8
						dom.	–	–	140783	90781	–	0	inf%	17.9
				0.5	log	A*	1.5	2%	136045	262102	–	419	76.8%	103.0
						cor.	1.5	3%	62843	99434	13%	419	13.1%	48.5
						dom.	–	–	141587	91585	–	0	inf%	12.6
				0.9		A*	1.5	8%	25125	40262	–	419	90.6%	21.8
						cor.	1.5	3%	90920	98293	42%	413	47.0%	52.1
						dom.	–	–	141587	91585	–	0	inf%	12.6
wide100	25602	78400	574	0.5	gen	A*	1	65%	47	1692	–	21	opt	2.8
						cor.	1	66%	37	1670	0%	21	opt	2.8
						dom.	–	–	8631	2939	–	0	inf%	1.2
				0.9		A*	1	65%	59	1715	–	19	opt	3.1
						cor.	1	66%	50	1689	0%	19	opt	3.1
						dom.	–	–	8538	2884	–	0	inf%	1.3
				0.5	log	A*	1	66%	67	1732	–	21	opt	2.7
						cor.	1	66%	67	1732	0%	21	opt	2.7
						dom.	–	–	8611	2886	–	0	inf%	1.1
				0.9		A*	1	66%	87	1772	–	19	opt	2.8
						cor.	1	66%	87	1772	0%	19	opt	2.8
						dom.	–	–	8582	2872	–	0	inf%	1.2

Table 5.2 – STOCHASTIC RESOURCE CONSTRAINED SHORTEST PATH PROBLEM results with independent and discrete distributions and constraints $\mathbb{P}\left(\sum_{a\in P}\xi_a > \tau\right) \leq \rho_0$

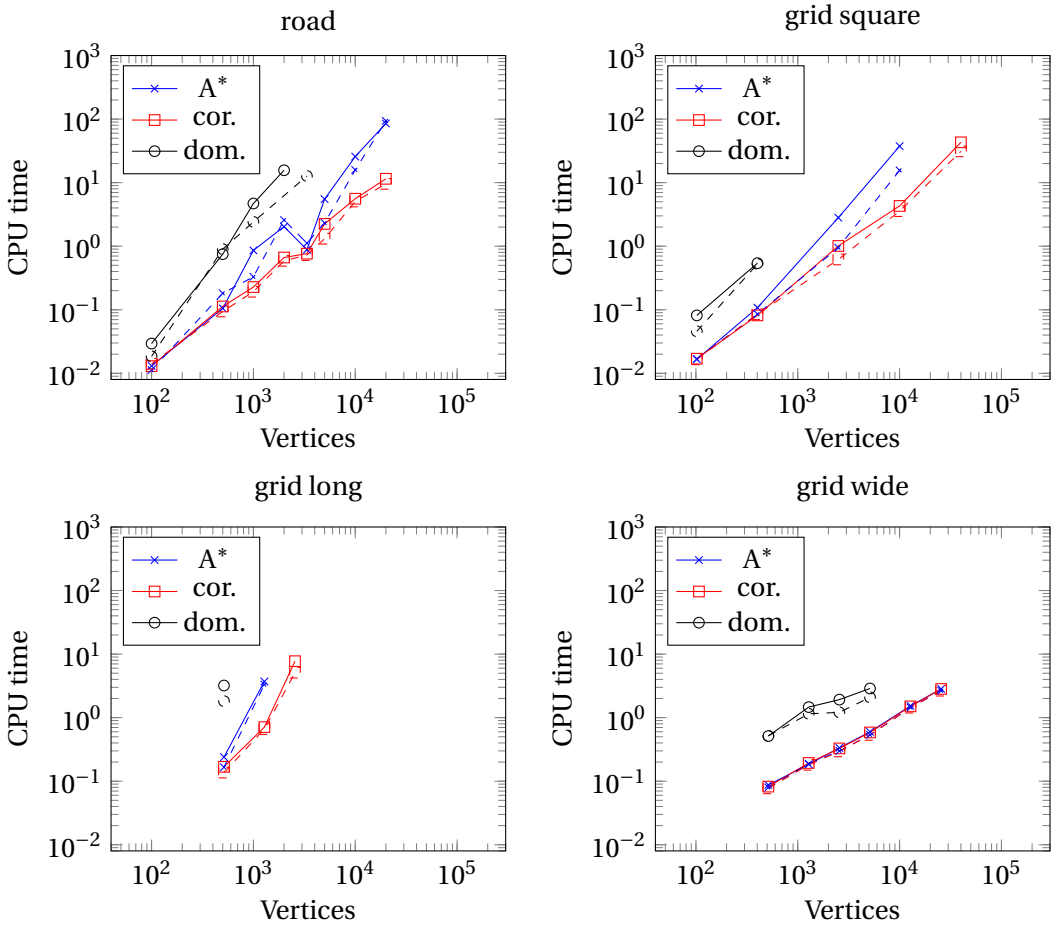


Figure 5.2 – Algorithms performances on the STOCHASTIC RESOURCE CONSTRAINED SHORTEST PATH PROBLEM problem using resource constraints $\mathbb{P}(\cdot > \tau) \leq \rho_0$ and $\alpha = 0.5$. Plain lines correspond to generic distributions and dashed lines to truncated lognormal distributions

preprocessing time can be divide by a factor γ if we compute the lower bound separately for the constraint and the cost. See Section 4.3.4 for more details. The number of paths extended by the algorithm is much larger than in the deterministic case. Therefore, the preprocessing takes a smaller proportion of the total time than in the non constrained case. Nonetheless, the dominance test still enables to cut paths, as we can see in the proportion of paths cut by the dominance test in the label correcting algorithm. This is the reason why the label correcting algorithm performs better than the generalized A* algorithm: it solves faster the instances solved to optimality, and lead to smaller gaps on instances that are not solved to optimality.

5.3 Scenario based distributions

In this section, we study the STOCHASTIC SHORTEST PATH PROBLEM under the additional assumption that the random variables ξ_a belong to a probability space $(\Omega, \mathcal{F}, \mu)$ with finite Ω.

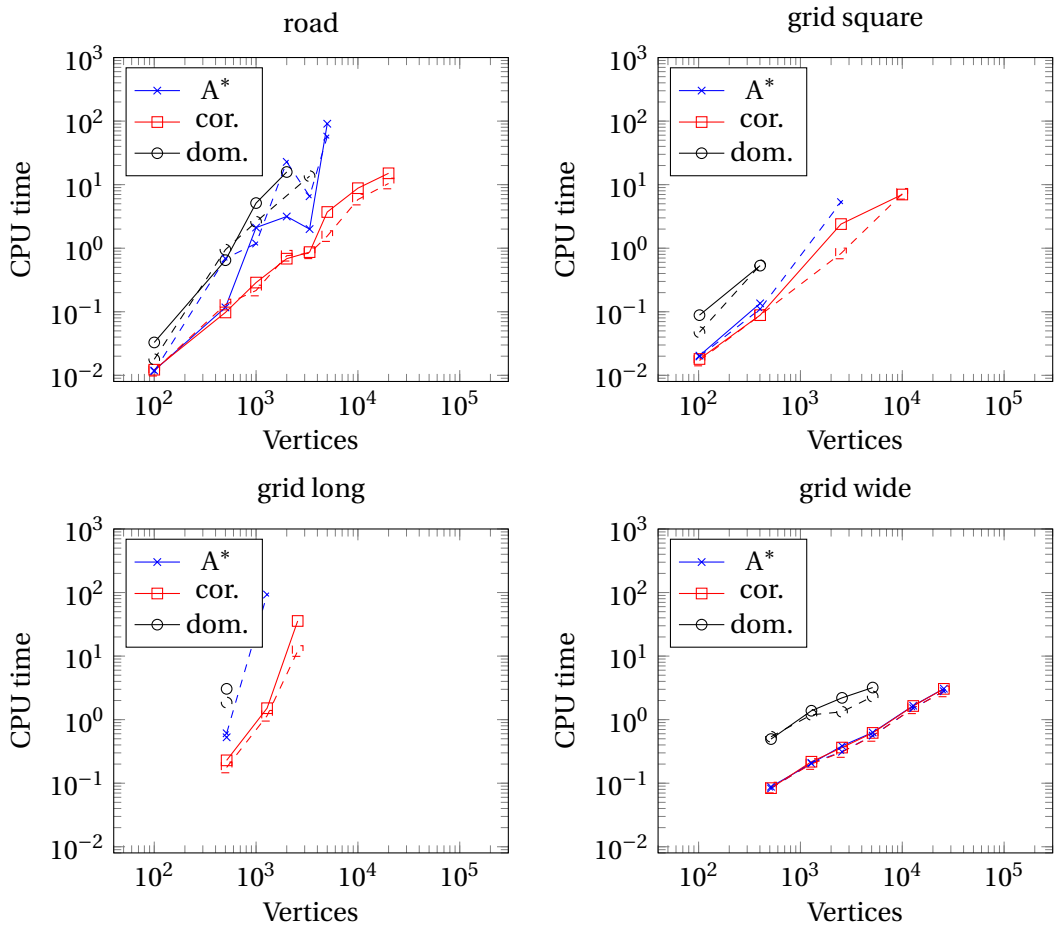


Figure 5.3 – Algorithms performances on the STOCHASTIC RESOURCE CONSTRAINED SHORTEST PATH PROBLEM problem using probability constraints $\mathbb{P}(\cdot > \tau) \leq \rho_0$ with $\alpha = 0.9$. Plain lines correspond to generic distributions and dashed lines to truncated lognormal distributions

5.3.1 Scenario lattice ordered monoid

As mentioned in the introduction of the chapter, when the set Ω of scenarios has finite cardinal N, the set of real random variables on Ω endowed with the usual sum and scenario by scenario order is a lattice ordered monoid whose operations $+$ and $\wedge$ can be computed efficiently. Besides, all the probability functionals of interest we mentioned in the introduction are isotone with respect to the almost sure order. Practically, each random variable ξ on this space can be considered as a vector of $\mathbb{R}^N$ whose i^{th} coordinate is $\xi(i)$ the value of under scenario i.

5.3.2 Complexity of the problem with finite number of scenarios

In this section, we study the complexity of the STOCHASTIC SHORTEST PATH PROBLEM with finite Ω when the probability functional ρ is fixed. We say that a random variable has 2-sample

distributions if Ω has two elements ω_1 and ω_2 and $\mathbb{P}(\omega_1) = \mathbb{P}(\omega_2) = 1/2$.

Theorem 5.6. *Suppose that c is such that, for any real μ, there exists constants α and β such that any random variable with 2-sample distribution and with expectation c_0 satisfies $c(\alpha\xi + \beta) > c(\alpha c_0 + \beta)$ if $\xi \neq c_0$. Then, unless $\mathscr{P} = \mathscr{NP}$, there is no polynomial algorithm that solves the* STOCHASTIC SHORTEST PATH PROBLEM *with finite Ω and cost function c.*

The hypothesis on ρ is Theorem 5.6 is satisfied by most probability functionals of interest different from the expectation of an affine function. For, instance, it is satisfied by $c(\cdot) = \mathbb{P}(\cdot > \tau)$, or by the expectation of any function strictly convex or concave on an interval. The next lemma shows that it is satisfied for any risk measure different from the expectation.

Lemma 5.7. *The hypothesis of Theorem 5.6 is satisfied by any risk measure different from the expectation.*

Proof. We start by proving the result for distortion functionals. Let ρ be a distortion functional associated to distortion function σ. Let ξ be a non constant random variable with 2-sample distributions. There exists c^- and x^+ such that $x^- < x^+$ and $\xi = x^-$ (resp. x^+) with probability $1/2$. We therefore have

$$\rho(X) = x^- \left(\int_0^{1/2} \sigma \right) + x^+ \left(\int_{1/2}^1 \sigma \right).$$

By definition of distortion functionals, σ is non-decreasing and $\int_0^1 \sigma = 1$. The hypothesis that ρ is not the expectation implies that σ is non-constant on $]0,1[$. As a consequence, $\int_0^{1/2} \sigma < 1/2$, which gives the result with $\alpha = 1$ and $\beta = 0$.

We now extend the result to risk measures generated by a set of distortion functions $\mathscr{S}$. Indeed, the hypothesis that ρ is not the expectation implies that $\sup_{\sigma\in\mathscr{S}} \int_0^{1/2} \sigma < 1/2$. Indeed, suppose that $\sup_{\sigma\in\mathscr{S}} \int_0^{1/2} \sigma = 1/2$, and consider a sequence of σ_n such that $\int_0^{1/2} \sigma_n \to_n 1/2$. Then given a random variable ξ such that $|\xi| \leq M$, we obtain that $|\rho_{\sigma_n}(\xi) - \mathbb{E}(\xi)| \leq \left(\frac{1}{2} - \int_0^{1/2} \sigma_n\right) M$, and thus $\rho(\xi) = \mathbb{E}(\xi)$ for bounded random variables ξ. Using a converging sequence of bounded distributions and the dominated convergence theorem, this enables to conclude that ρ is the expectation for any integrable random variable. Theorem 2.1 then enables to conclude. □

Our proof of Theorem 5.6 is an extension of the proof of $\mathscr{NP}$-hardness of the robust shortest path problem by [176].

Proof of Theorem 5.6. We reduce the STOCHASTIC SHORTEST PATH PROBLEM with finite Ω to the $\mathscr{NP}$-complete 2-PARTITION PROBLEM [84]. The 2-PARTITION PROBLEM can be stated as follows: given m integers c_i indexed by $i \in [m]$, is there a subset I of $[m]$ such that $\sum_{i\in I} c_i = \sum_{i\in[m]\setminus I} c_i$?

In the remaining of the proof, we reduce the 2-PARTITION PROBLEM to a STOCHASTIC SHORTEST PATH PROBLEM instance with random variables on a probability space $\Omega = \{\omega_1, \omega_2\}$ with two events. We use the notation (ξ_1, ξ_2) to define the random variable ξ such that $\xi(\omega_i) = \xi_i$ for i in $\{1,2\}$.

Let α and β be the constants of the hypothesis for ρ and $\mu = \frac{1}{2}\sum_{i\in I} c_i$. We build a digraph D with a source vertex o, a sink vertex v, and for each i in I, two vertices $v_{i,1}$ and $v_{i,2}$. Digraph D is illustrated on Figure 5.4. We build two arcs $(o, v_{1,1})$ and $(o, v_{1,2})$ with resource random variable (β, β). For each $i < m$, we build two arcs $(v_{i,1}, v_{i+1,1})$ and $(v_{i,1}, v_{i+1,2})$ with random variables $\alpha(c_i, 0)$, and two arcs $(v_{i,2}, v_{i+1,1})$ and $(v_{i,2}, v_{i+1,2})$ with sample $\alpha(0, c_i)$. Finally, we build two arcs $(v_{m,1}, d)$ and $(v_{m,2}, d)$ with respective random variables $\alpha(c_m, 0)$ and $\alpha(0, c_m)$.

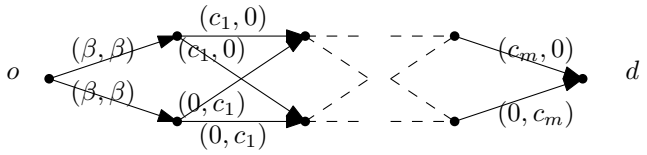


Figure 5.4 – Partition problem reduction to a graph problem.

There is a bijection between subsets I' of I and o-d paths P, the o-d path corresponding to a subset I' being the unique path $P_{I'}$ such that $v_{i,1} \in P_{I'}$ if and only if $i \in I'$, and $v_{i,2} \in P_{I'}$ if and only if $i \in I \backslash I'$. Besides, the sample corresponding to path $P_{I'}$ is $\xi_{P_{I'}} = (\alpha\sum_{i\in I'} c_i + \beta, \alpha\sum_{i\in I\backslash I'} c_i + \beta)$. Using the hypothesis on ρ, we have $\rho\left(\xi_{\mathbb{P}_{I'}}\right) \geq \rho\left(\frac{\alpha}{2}\sum_{i\in I} c_i + \beta\right)$ with equality if and only if $\sum_{i\in I'} c_i = \sum_{i\in I\backslash I'} c_i$. Finally, there exists a solution to the 2-partition of I if and only if the optimal solution of the OFFLINE SAMPLED SHORTEST PATH PROBLEM is $\rho(\frac{\alpha}{2}\sum_{i\in I} c_i + \beta)$, which concludes the reduction and gives the theorem. □

Remark 5.2. Note that Theorem 5.6 remains true even when there are only two scenarios, the treewidth of D is non-greater than 2, and $x_a \geq 0$ for each arc a. Indeed, the STOCHASTIC SHORTEST PATH PROBLEM instance built in the proof of Theorem 5.6 satisfies these assumptions.

Remark 5.3. Note that an analogue of Theorem 5.6 is true for the sampled minimum spanning tree problem defined as follows: the input is identical to the one of the OFFLINE SAMPLED SHORTEST PATH PROBLEM without an origin and a destination, and the output is a spanning tree T with minimum $\rho^N(\sum_{a\in T}\xi_a^N)$. The proof can be easily adapted to this new problem. Indeed if arcs $(v_{i,1}, v_{i,2})$ with sample $(0,0)$ are added to the graph for each i, then the set of arcs composed of the arcs of an optimal path plus $(v_{i,1}, v_{i,2})$ for each i is a minimum spanning tree.

5.3.3 Bounds and online problem

The bounds $b_v^\dagger$ of Equation (4.8) also admit a natural interpretation in the scenario based distributions: $b_v^\dagger(\omega)$ is the length of the shortest v-d path under scenario ω. As a consequence, these bounds can be computed using a STANDARD SHORTEST PATH PROBLEM algorithm scenario by scenario instead of the algorithms of Chapter 4.

Note that bounds b_v can no more be interpreted as solution of the STOCHASTIC ON TIME ARRIVAL PROBLEM, as it was the case for independent distributions in Proposition 5.5. Besides, in the case of non-independent random variables on a finite $|\Omega|$, the STOCHASTIC ON TIME ARRIVAL PROBLEM is difficult to solve, as it belongs to the field of multistage stochastic

programming [33, 143].

5.3.4 Convergence to the optimal solution of the initial problem

Consider the general STOCHASTIC SHORTEST PATH PROBLEM where ξ_a are random variables ξ_a on a probability space $(\Omega, \mathscr{F}, \mathbb{P})$. Given N independent samples $\xi^1, \dots, \xi^N$ of ξ, the Monte-Carlo approximation of this problems consists in replacing μ by $\widehat{\mu^N}$, where

$$\widehat{\mu^N} = \frac{\sum_{i=1}^{N} \delta_{\xi^i}}{N}, \quad \text{and } \delta_{\xi^i} \text{ is the Dirac distribution in } \xi^i.$$

As $\widehat{\mu^N}$ lives on a probability space with N events, the resulting problem can be solved using the lattice ordered monoid of Section 5.3 and the algorithms of Chapter 4. The remaining of the section provides bounds on the error made when replacing μ by $\widehat{\mu^N}$. Such bounds enable to choose by which factor we should increase N if we want to obtain a desired accuracy. *In the remaining of the section, we denote by P^* an optimal solution of the* STOCHASTIC SHORTEST PATH PROBLEM *with ξ having the initial distribution μ, and $\widehat{P^N}$ an optimal solution with ξ having distribution $\widehat{\mu^N}$*. Given a path P, let ξ_P denote the sum $\sum_{a \in P} \xi_a$.

Under the assumption that μ admits a finite exponential moment, the following corollary bounds the probability of large deviations of a solution of the Monte-Carlo approximation from the optimal solution of the initial problem when the cost function is a version independent risk measure or a probability.

Corollary 5.8. *Suppose that there exists $\alpha > 1$ and γ such that*

$$\varepsilon_{\alpha,\gamma}(\mu) = \int_{\mathbb{R}^d} e^{\gamma |x|^\alpha} \mu(dx) < \infty,$$

then there exists C and $\tilde{C}$ in $\mathbb{R}_+$ depending only on $|A|, \alpha, \gamma$, and $\varepsilon_{\alpha,\gamma}(\mu)$ such that for all $N \geq 1$ and all $x > 0$, we have

1. *if c is a version independent risk measure generated by a set of distortion functions $\mathscr{S}$, then*

$$\mathbb{P}\left[c_\mu\left(\xi_{\widehat{P^N}}\right) - c_\mu(\xi_{P^*}) > x\right] \leq C \exp\left(-\tilde{C} N \left(\frac{x}{2\sqrt{|A|} \sup_{\sigma \in \mathscr{S}} \|\sigma\|_\infty}\right)^\beta\right),$$

2. *if $c = \mathbb{P}(\cdot > \tau)$ and μ has a support in $\mathbb{Z}^A$, then*

$$\mathbb{P}\left[c_\mu\left(\xi_{\widehat{P^N}}\right) - c_\mu(\xi_{P^*}) > x\right] \leq C \exp\left(-\tilde{C} N \left(\frac{x}{2\sqrt{|A|}}\right)^\beta\right),$$

3. *and if $c = \mathbb{P}(\cdot > \tau)$ and ξ_P admits a density bounded by B for each path P, then*

$$\mathbb{P}\left[c_\mu\left(\xi_{\widehat{P^N}}\right) - c_\mu(\xi_{P^*}) > x\right] \leq C \exp\left(-\tilde{C} N \left(\frac{x^2}{2B\sqrt{|A|}}\right)^\beta\right),$$

where $\beta = |A|$ if $x < 1$ and α otherwise, and probability $\mathbb{P}$ is with respect to the sampling of $\widehat{\mu^N}$.

Proof. Given a path P with ℓ arcs, the application $\xi \mapsto \sum_{a\in P}\xi_a$ is $\sqrt{\ell}$-Lipschitz and thus $\sqrt{|A|}$-Lipschitz. Corollary 5.8 is then a direct application of Theorems 2.5 and 2.6 given Theorem 2.4. □

We now consider the STOCHASTIC RESOURCE CONSTRAINED SHORTEST PATH PROBLEM.

Corollary 5.9. *Let $\widehat{P^N}$ be a solution of the* STOCHASTIC RESOURCE CONSTRAINED SHORTEST PATH PROBLEM *where $\rho_\mu(x_P) \le 0$ has been replaced by $\rho_{\widehat{\mu^N}}(x_P) \le -t$. Suppose that there exists $\alpha > 1$ and γ such that*

$$\varepsilon_{\alpha,\gamma}(\mu) = \int_{\mathbb{R}^d} e^{\gamma|x|^\alpha}\mu(dx) < \infty,$$

then there exists C and $\tilde{C}$ in $\mathbb{R}_+$ depending only on $|A|, \alpha, \gamma$, and $\varepsilon_{\alpha,\gamma}(\mu)$ such that for all $N \ge 1$ and all $x > 0$, we have

1. *if c is a version independent risk measure generated by a set of distortion functions $\mathscr{S}$, then*
$$\mathbb{P}\left(\rho(\xi_{P^N}) > t\right) \le C\left(\exp\left(-\tilde{C}N(t/K)^{|A|}\right)\mathbf{1}_{t\le 1} + \exp\left(-\tilde{C}N(t/K)^{\alpha}\right)\mathbf{1}_{t>1}\right),$$
where $K = \sqrt{|A|}\sup_{\sigma\in\mathscr{S}}\|\sigma\|_\infty$
2. *if $c = \mathbb{P}(\cdot > \tau)$ and μ has a support in $\mathbb{Z}^A$, then*
$$\mathbb{P}\left(\rho(\xi_{P^N}) > t\right) \le C\left(\exp\left(-\tilde{C}N\left(t/\sqrt{|A|}\right)^{|A|}\right)\mathbf{1}_{t\le 1} + \exp\left(-\tilde{C}N\left(t/\sqrt{|A|}\right)^{\alpha}\right)\mathbf{1}_{t>1}\right),$$
3. *and if $c = \mathbb{P}(\cdot > \tau)$ and ξ_P admits a density bounded by B for each path P, then*
$$\mathbb{P}\left(\rho(\xi_{P^N}) > t\right) \le C\left(\exp\left(-\tilde{C}N\left(t^2/K\right)^{|A|}\right)\mathbf{1}_{t\le 1} + \exp\left(-\tilde{C}N\left(t^2/K\right)^{\alpha}\right)\mathbf{1}_{t>1}\right),$$
where $K = 4\sqrt{|A|}B$.

Proof. Given a path P with ℓ arcs, the application $\xi \mapsto \sum_{a\in P}\xi_a$ is $\sqrt{\ell}$-Lipschitz and thus $\sqrt{|A|}$-Lipschitz. Corollary 5.8 is then a direct application of Theorem 2.7 given Theorem 2.4. □

5.4 Numerical results for scenario based distributions

5.4.1 Instances and problem considered

This section tests the performance of the algorithms of Chapter 4 on the resolution of STOCHASTIC SHORTEST PATH PROBLEM problems with scenario based distributions. We use again the road, square grid, long grid, and wide grid digraphs of Section 4.3.1.

We want the scenario distributions on the different arcs to be samples of non-independent random variables. We therefore suppose that the set of arc A is partitioned in a set $\mathrm{cl}(a)|a \in A$ of clusters, each arc a belonging to a cluster $\mathrm{cl}(a)$. For each cluster $cl(a)$, we suppose to have a random variable ξ_v^{cl}, and for each arc a, we suppose to have an intrinsic random variable ξ_a^{int}.

The random variables ξ_a^{int} and ξ_v^{cl} are supposed to be independent. We define the random variable ξ_a of each arc a to be the sum

$$\xi_a = \xi_{\text{cl}(a)}^{\text{cl}} + \xi_a^{\text{int}}.$$

To build the clusters of arc, we start by building clusters of vertices as follows: we randomly choose $|V|/20$ vertices in V to build a set V_o of vertices. We then affect each vertex v to the nearest vertex $\text{cl}_V(v)$ in V_o, nearest being with respect to the distance in the initial deterministic graph. We build an additional cluster with all the vertices that are not reachable from any of the vertices in V_o. Finally, we build clusters of arc by affecting each arc $a = (u, v)$ to $\text{cl}(a) = \text{cl}_V(v)$. We obtain a partition of A into clusters $\{a|\text{cl}(a) = v\}$ for each v in V_o.

We define each cluster random variable ξ_v^{cl} distribution to be a uniform distributions over $\{0, 1, \ldots, t_v\}$, where t_v is randomly chosen between 3 and 10. For the random variable ξ_a^{int}, we use the lognormal distributions of Section 5.2.1, both in their truncated and discretized version and in their non-discretized and non-truncated versions. We build instances by sampling N realizations of the cluster and instrinsic random variables. We have tested the algorithm on instances with N in $\{10, 100, 1000\}$. We first provide detailed results for $N = 100$, and then study the influence of N on algorithms performance is studied.

The stochastic path problems with scenario based distributions happen to be much more difficult that the one with independent distributions with finite support. We therefore focus only on the STOCHASTIC SHORTEST PATH PROBLEM, as the STOCHASTIC RESOURCE CONSTRAINED SHORTEST PATH PROBLEM is not so well solved. For the STOCHASTIC SHORTEST PATH PROBLEM, we use the conditional value at risk CVaR_β as cost function c, with β in $\{0.01, 0.1\}$.

The generalized A^*, the label correcting, and the label dominance algorithms have been tested. As in Chapter 4 numerical experiments, Theorems 4.7 and 4.8 ensure respectively that the label correcting and the label dominance algorithm converge. Case (b) of Theorem 4.3 with assumption (4.3) ensures the convergence of the generalized A* algorithm. In each of these algorithms, we use the candidate paths mentioned in Section 4.3.3. We use our `latticeRCSP` library described in Appendix C to solve the instances. The algorithms are not parallelized, and the numerical experiments are performed on a Macbook Pro of 2012 with four 2.5 Ghz processors and 4 Gb of ram. The maximum size of the candidate paths list L used is 5e+04 for the label correcting and the label dominance algorithms and 1e+05 for the generalized A* algorithm. Indeed, as label correcting and label dominance algorithms require to store non dominated paths resources, less memory is available for L. Bounds are computed using the generalized Dijkstra algorithm of Section 4.2.

5.4.2 Main results

Figure 5.5 provides the CPU time for instances solved to optimality on the different family of instances for $N = 100$. Plain lines correspond to instances with truncated and discretized lognormal distributions ξ_a^{int}, while dashed lines correspond to non-truncated and non-discretized ξ_a^{int}. Table 5.3 provides detailed numerical results on some truncated and discretized instances, while Table 5.4 provides results on some non-truncated lognormal distributions. The first

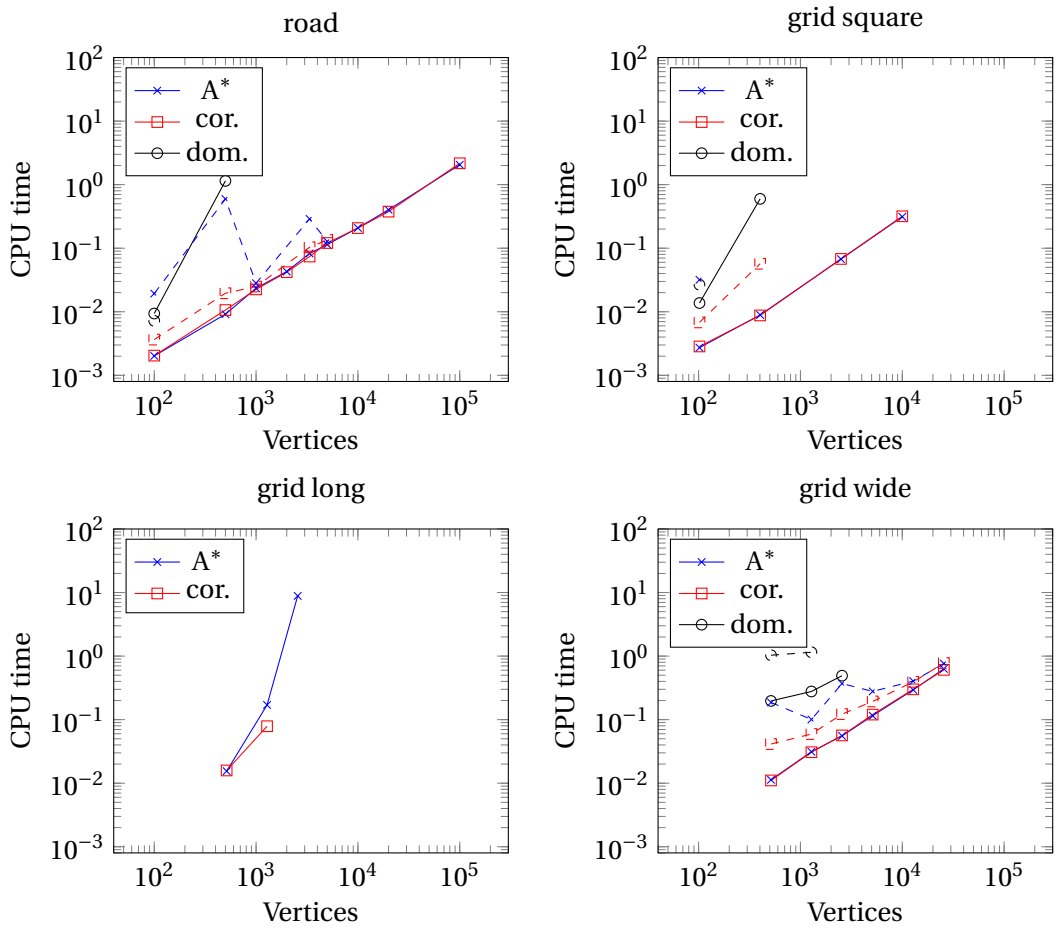


Figure 5.5 – Algorithms performances on stochastic shortest path problem using CVaR_β with $\beta = 0.01$ as probability functional. Plain lines corresponds to discretized truncated lognormal distributions, and dashed lines to non truncated and non discretized lognormal distributions. None of the long grind instances could be solved by the label dominance algorithm.

column provides the instance name, the two next ones the number of vertices and the number of arcs of the instance, the two next columns the parameter β of the probability function CVaR_β used, and the next column the algorithm used. For this column, A* corresponds to the generalized A* algorithm, cor. to the label correcting algorithm and dom. to the label dominance algorithm. The two next columns provide the ratio γ of Remark 4.5 between the number of extension by the generalized Dijkstra algorithm and the number of vertices, and the proportion of time spent in the preprocessing. The three next columns provide the number of paths extended by the enumeration algorithm, the number of paths cut, and the proportion of paths cut by the dominance test when the label correcting algorithm is used. The column ℓ provides the number of arcs in the solution returned. The next column provide the gap between the lower bound proved by the enumeration algorithm and the solution returned. It has the mention opt when the instances is solved to optimality. Finally, the last column

Instance	$\|V\|$	$\|A\|$	β	Alg.	γ	Prep.	Ext.	Cut.	Dom.	ℓ	Gap	CPU (s)
road100	100000	258486	0.01	A*	1.082	59%	367	605	–	611	opt	2.06e+00
				cor.	1.082	53%	367	605	0%	611	opt	2.17e+00
				dom.	–	–	64766	40316	–	611	1729.7%	1.97e+00
			0.1	A*	1.082	15%	310176	442115	–	611	opt	7.47e+00
				cor.	1.082	43%	16453	29458	2%	611	opt	2.66e+00
				dom.	–	–	65089	40560	–	611	1746.0%	1.97e+00
square100	10002	30100	0.01	A*	1.256	44%	4095	7813	–	131	opt	3.12e-01
				cor.	1.256	44%	2934	5432	1%	131	opt	3.18e-01
				dom.	–	–	33217	18844	–	122	497.4%	6.55e-01
			0.1	A*	1.256	37%	7600	15255	–	132	opt	3.86e-01
				cor.	1.256	36%	4912	9591	2%	132	opt	3.80e-01
				dom.	–	–	32545	18396	–	134	509.0%	6.55e-01
long10	2562	7696	0.01	A*	1.189	0%	418031	836049	–	198	opt	8.86e+00
				cor.	1.189	0%	50383	94494	0%	198	0.3%	7.34e+00
				dom.	–	–	32710	18477	–	200	537.0%	7.52e-01
			0.1	A*	1.189	5%	23942	37866	–	198	0.6%	5.96e-01
				cor.	1.189	1%	35549	60327	1%	198	0.5%	3.98e+00
				dom.	–	–	32656	18442	–	200	553.5%	7.62e-01
wide100	25602	78400	0.01	A*	1.125	60%	26	1641	–	21	opt	6.30e-01
				cor.	1.125	59%	26	1641	0%	21	opt	6.08e-01
				dom.	–	–	8157	2636	–	21	282.8%	4.27e-01
			0.1	A*	1.125	60%	24	1635	–	19	opt	6.23e-01
				cor.	1.125	59%	24	1635	0%	19	opt	6.17e-01
				dom.	–	–	7957	2503	–	19	306.8%	4.37e-01

Table 5.3 – STOCHASTIC SHORTEST PATH PROBLEM results with truncated and discretized lognormal distribution

provides the total CPU time.

We start the analysis with instances difficulty. By comparing Figure 5.5 with Figure 5.1, we can see that the STOCHASTIC SHORTEST PATH PROBLEM with scenario based distributions is much harder to solve than with independent discretized distributions. Besides, instances sampled from truncated and discretized distributions are much easier to solve than those sampled from non-truncated and non-discretized distributions. This increased difficulty comes from the upper tail of the non-truncated lognormal distributions. The tail of a distributions has a strong influence on the value of the CVaR_β. Concerning the performance of the algorithms, when we compute the meet $q \wedge \tilde{q}$ of two scenario resources q and $\tilde{q}$, the largest scenarios of each resource disappear as it is larger than the one of the other resource. As a consequence, the lower bounds provided by the approach of Section 4.2 are not tight on the largest scenarios, and $\text{CVaR}_\beta(q \wedge \tilde{q})$ is not a tight lower bounds on $\text{CVaR}_\beta(q)$ and $\text{CVaR}_\beta(\tilde{q})$.

We now come to the performance of the algorithms on the truncated and discretized distributions. We can see on Figure 5.5 that the label correcting and the generalized A* algorithm can solve instances between one and two order of magnitude larger in term of number of vertices than the label dominance algorithm. This performance can be understood by the fact that, as we can see in Table 5.3, most paths are cut by the bound test and not by the label dominance

Instance	$\|V\|$	$\|A\|$	β	Alg.	γ	Prep.	Ext.	Cut.	Dom.	ℓ	Gap	CPU (s)
road5	5000	15508	0.01	A*	1.618	71%	100	80	–	111	opt	1.28e-01
				cor.	1.618	70%	100	80	0%	111	opt	1.35e-01
				dom.	–	–	12553	5506	–	105	891.8%	3.93e-01
			0.1	A*	1.618	35%	4217	412	–	105	13.3%	2.66e-01
				cor.	1.618	24%	5613	1721	82%	107	7.2%	3.72e-01
				dom.	–	–	12057	5378	–	95	399.9%	3.67e-01
square20	402	1220	0.01	A*	1.709	3%	8309	6281	–	26	21.7%	2.23e-01
				cor.	1.709	13%	2164	2331	22%	25	opt	5.69e-02
				dom.	–	–	22130	8936	–	25	19.8%	9.73e-01
			0.1	A*	1.709	4%	6128	2131	–	26	16.0%	1.85e-01
				cor.	1.709	5%	6700	10385	13%	26	opt	1.98e-01
				dom.	–	–	12213	4446	–	26	77.2%	4.36e-01
long5	1282	3856	0.01	A*	1.784	14%	5806	1623	–	100	45.3%	2.05e-01
				cor.	1.784	7%	8592	3017	69%	100	40.2%	3.74e-01
				dom.	–	–	11833	4327	–	100	407.7%	4.41e-01
			0.1	A*	1.784	14%	5209	428	–	102	33.9%	1.92e-01
				cor.	1.784	9%	7195	1562	91%	102	30.7%	3.00e-01
				dom.	–	–	10488	3651	–	102	526.3%	3.70e-01
wide100	25602	78400	0.01	A*	1.605	66%	644	2217	–	20	opt	7.62e-01
				cor.	1.605	67%	583	2033	2%	20	opt	7.77e-01
				dom.	–	–	13991	3934	–	20	91.7%	5.38e-01
			0.1	A*	1.605	66%	370	1663	–	20	opt	7.76e-01
				cor.	1.605	66%	357	1621	0%	20	opt	7.58e-01
				dom.	–	–	7386	1668	–	20	216.4%	4.16e-01

Table 5.4 – STOCHASTIC SHORTEST PATH PROBLEM results with non truncated and non discretized lognormal distribution

test. This phenomenon is the same as the one observed in Table 4.4: dominance becomes rare when the dimension increases. As most discarded paths are discarded by the bound test, the label correcting algorithm has almost the same performance as the generalized A* algorithm.

The same general observations on the relative performance of the algorithms can be done on the non truncated and non discretized lognormal instances of Table 5.4: the label correcting and the generalized A* algorithms are still much more efficient than the label dominance algorithm. On hard instances, the bounds are less tight and the label correcting algorithms exhibits better performance that the generalized A* algorithm.

5.4.3 Influence of the number of samples

We now study the influence of the number of scenarios on algorithms performance. We therefore sample from the truncated and discretized lognormal distributions instances with 10, 100, and 1000 scenarios. Table 5.5 provide the results: it is the analogue of Table 5.3 with an additional column N that indicates the number of scenarios of the instance. The instances considered are slightly smaller than those considered in Table 5.3 as more scenarios require more memory. Concering the bounding algorithms, we can see that the ratio γ or Remark 4.5 increases slowly with the number of scenarios: its performance remains good even on large instances. We can evaluate the relative performance of the enumeration algorithms on the same graphs with different number of scenarios by comparing the gap, the number of paths extend, and the CPU time when the instances are solved to optimality. Concerning the number of paths enumerated and the gap, we can see that the instances with 10 scenarios are easier to solve, but that the instances with 100 and 1000 scenarios have almost the same difficulty. As the complexity of the operators $\oplus$, $\wedge$, and $\leqslant$ is linear in the number of scenarios, if the algorithm keeps the same performance, we expect the CPU time to scale linearly with the number of scenarios. This is almost the case on the instance solved to optimality.

5.5 Bibliographical remarks

We provide here a literature review on the approaches to the different stochastic path problems mentioned in this chapter. As the techniques presented in this chapter are versatile, more complex resource constrained shortest path problems can be solved. If there are relatively few approaches to the STOCHASTIC RESOURCE CONSTRAINED SHORTEST PATH PROBLEM considered in this chapter, many complex stochastic resource constrained shortest path problem have been solved as subproblem of column generation approaches to stochastic problems. For instance, the subproblem of the column generation approach to the STOCHASTIC CREW PAIRING PROBLEM considered in Chapter 11 is an highly non-linear stochastic resource constrained shortest path problem. Approaches to airline operations taking into account delay are indeed a source of stochastic resource constrained shortest path problems. We review them in Chapter 11. But the field that probably gives rise to the widest range of stochastic resource constrained shortest path problems is the one of stochastic traveling salesman and vehicle routing problems. We therefore give a quick review on these problems at the end of the section.

Instance	$\|V\|$	$\|A\|$	β	N	Alg.	γ	Prep.	Ext.	Cut.	Dom.	ℓ	Gap	CPU (s)
road20	20000	55180	0.01	10	A*	1.031	45%	470	892	–	247	opt	1.12e-01
					cor.	1.031	48%	343	641	1%	247	opt	1.16e-01
					dom.	–	–	74719	44215	–	247	67.5%	4.64e-01
				100	A*	1.1	58%	1347	2657	–	244	opt	4.03e-01
					cor.	1.1	61%	535	1087	1%	244	opt	3.76e-01
					dom.	–	–	107536	57532	–	247	86.3%	1.75e+00
				1000	A*	1.195	66%	1176	2288	–	240	opt	5.86e+00
					cor.	1.195	70%	651	1282	1%	240	opt	5.62e+00
					dom.	–	–	109724	59722	–	240	148.3%	1.99e+01
square50	2502	7550	0.01	10	A*	1.065	44%	29	96	–	73	opt	1.44e-02
					cor.	1.065	43%	29	96	0%	73	opt	1.45e-02
					dom.	–	–	80593	50410	–	73	36.6%	4.34e-01
				100	A*	1.2	53%	455	944	–	62	opt	6.73e-02
					cor.	1.2	53%	454	940	0%	62	opt	6.78e-02
					dom.	–	–	39418	22961	–	62	143.2%	7.87e-01
				1000	A*	1.376	55%	470	971	–	62	opt	8.64e-01
					cor.	1.376	55%	451	931	0%	62	opt	8.54e-01
					dom.	–	–	23587	12406	–	62	238.2%	5.25e+00
long10	2562	7696	0.01	10	A*	1.065	22%	2484	4928	–	198	opt	2.64e-02
					cor.	1.065	30%	1271	2478	0%	198	opt	2.56e-02
					dom.	–	–	79687	49795	–	196	185.1%	5.99e-01
				100	A*	1.189	0%	418031	836049	–	198	opt	8.86e+00
					cor.	1.189	0%	50383	94494	0%	198	0.3%	7.34e+00
					dom.	–	–	32710	18477	–	200	537.0%	7.52e-01
				1000	A*	1.364	7%	18603	27204	–	200	0.7%	7.01e+00
					cor.	1.364	3%	20575	30890	0%	200	0.7%	1.48e+01
					dom.	–	–	20455	10308	–	200	783.1%	5.90e+00
wide50	12802	39200	0.01	10	A*	1.041	46%	2	788	–	19	opt	9.70e-02
					cor.	1.041	43%	2	788	0%	19	opt	9.39e-02
					dom.	–	–	28610	16005	–	19	42.2%	2.13e-01
				100	A*	1.131	61%	14	811	–	19	opt	2.94e-01
					cor.	1.131	60%	14	811	0%	19	opt	3.00e-01
					dom.	–	–	14395	6528	–	19	130.3%	3.94e-01
				1000	A*	1.235	63%	14	807	–	19	opt	3.12e+00
					cor.	1.235	64%	14	807	0%	19	opt	3.06e+00
					dom.	–	–	11030	4285	–	19	159.8%	3.42e+00

Table 5.5 – Number of scenarios influence on stochastic shortest path problem with truncated and discretized lognormal distributions

5.5.1 Offline stochastic shortest path

Stochastic shortest path problems have been extensively studied since the seminal work of Frank [81]. Models differ by the probability distributions used to model delay on arcs, and by the probability functional optimized.

A first line of papers considers the probability of on time arrival. The objective is to find a path maximizing the probability of on time arrival, or analogously, a path with minimum quantile of given order. Approaches have been developed for both continuous [36, 81, 138, 139] and discrete [129] distributions. Chen et al. [37] describe an efficient labeling algorithm to deal with normal distributions on the arcs. This algorithm is not so far from our label correcting algorithm applied with the lattice ordered monoid presented in Section 5.1.2 when restricted to $\rho(\cdot) = \mathbb{P}(\cdot > \tau)$.

A second line of papers defines a shortest path as a path minimizing the expectation of a cost function [119]. Dynamic programming can be used when cost functions are affine or exponential [65]. Murthy and Sarkar [132, 133] present an efficient labeling algorithm when arc distributions are normal and cost functions are piecewise-linear and concave. Instead of considering the expectation of a cost functions, other approaches search the path that minimizes a positive linear combination of mean and variance [138, 139, 163].

Finally, Miller-Hooks and Mahmassani [126] suggest to use stochastic dominance to compare paths. Algorithms to generate all non-dominated paths have also been proposed [126–128, 136, 137].

5.5.2 Online stochastic path problems

The online STOCHASTIC ON TIME ARRIVAL PROBLEM searches a policy which maximizes the probability of arrival before a given thresholds [82, 83, 91]. As defined in Section 5.1.3, a policy is an application which, given a vertex reached and the time it took to get to this vertex, indicates the next arc to choose to maximize the probability on on time arrival at destination. Fan and Nie [71] provide an algorithm for continuous distributions, and Nie and Fan [135] provide a pseudo-polynomial algorithm for discrete distributions. Samaranayake et al. [156] develop a faster algorithm for discrete distributions, and Sabran et al. [154] provide pre-processing techniques to improve algorithm speed. Finally, Flajolet et al. [75] provide a distributionally robust approach: a set of possible distributions is provided in input, and the objective is to find an adaptive path thats minimizes the worst expectation of the cost function over the set of possible distributions. They solve it using a dynamic programming approach.

5.5.3 Shortest path under probability constraint

Finally, the problem of finding a minimum cost path for deterministic arc costs under stochastic resource constraints have been introduced in [109], and a solution algorithm based on linear programming is derived.

5.5.4 Stochastic traveling salesman and vehicle routing problems

A wide range of stochastic versions of the traveling salesman and the vehicle routing problem have been studied in the last decades. When such problems are solved by column generation, the pricing subproblem is a stochastic resource constrained shortest path problem. Uncertainty in customer presence [99, 100], in demand [26, 27, 89, 164], and in travel time [5, 35, 101, 103, 117, 123, 153, 166].

Most probability functionals considered in this chapter can be dealt with using the lattice ordered monoid and the probability functionals presented in this chapter. Jaillet et al. [101] propose a specific probability functional in the context of vehicle routing with uncertainty on travel times, namely the Requirements Violation Index. This probability functional enables to model "soft" time windows. Using the modeling technique provided for time windows in Section 3.4.3 for interval constraints and the lattice ordered monoid provided in this section, we can define a lattice ordered monoid such that the Requirements Violation Index is isotone with respect to the lattice order. Nonetheless, the time needed to evaluate it on the resource of one path is probably too large for our enumeration algorithms to be practically efficient in solving resource constrained shortest path problem where the the Requirements Violation Index intervenes in the definition of the feasibility function.

Finally, we underline that, in the context of vehicle routing problems, the graph is often complete. In that case, the bounds provided by the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM algorithms are likely to be of poor quality. Thus, if we can model these problems within the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework, the solution approach of Chapter 4 may not suit to the specific structure of these problems on complete graphs.

6 State graphs to improve algorithm convergence

The enumeration algorithms of Chapter 4 rely on two tests to cut paths: the dominance test, and the lower-bound test, which we now restate. The dominance test uses a collection M_v of resources of non-dominated v-d paths.

(Dom) If *no path in M_v dominates P*, then if P is feasible, remove from M_v all paths dominated by P, add P to M_v, and extend P.

(Low) If $\rho(q_P \oplus b_v) = 0$ and $c(q_P \oplus b_v) \leqslant c^{UB}_{od}$, then extend P.

We mentioned in Chapters 4 and 5 that both tests become less efficient when the dimension or the number of scenarios increases. Indeed, if resources belong to $\mathbb{R}^n$ endowed with its usual order, a path q_P dominates a resource $q_{\tilde{P}}$ only if each each component of q_P is smaller than the corresponding component in $q_{\tilde{P}}$. As illustrated on Figure 6.1.(a) in the case of resources in $\mathbb{R}^2$, only one component of q_P greater than the corresponding component in $q_{\tilde{P}}$ is sufficient for P not to dominate $\tilde{P}$. As a consequence, dominance becomes rare when the number of components increases.

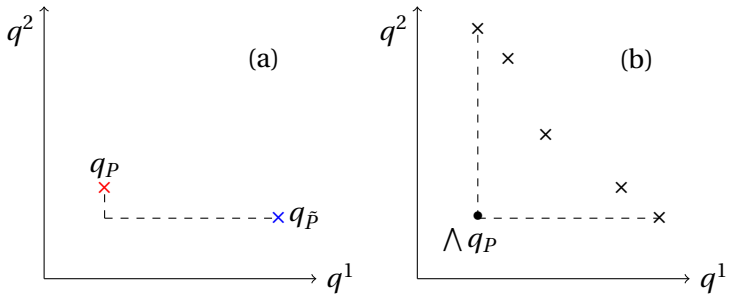


Figure 6.1 – Dominance and lower bound tests when dimension increases. On Figure (b), each symbol × corresponds to the resource of a v-d path.

Figure 6.1.(b) illustrates the problem with lower bounds when dimension increases. Indeed, bound b_v must be non greater than the resource of all the v-d paths P. As a consequence, we have

$$b_v \leqslant \bigwedge_{P \in \mathscr{P}_{vd}} q_P \quad \text{where } \mathscr{P}_{vd} \text{ is the set of } v\text{-}d \text{ paths.}$$

If there are paths with very different resources in $\mathscr{P}_{vd}$, b_v can be much smaller than the resource q_P of each path P in $\mathscr{P}_{vd}$.

The main message of this chapter is that this drawback of the lower bound test in large dimension can be overcome by using several bounds instead of one. Indeed, Figure 6.2.(a) shows that if paths in $\mathscr{P}_{vd}$ are partitioned into clusters of paths with similar resources, then the meet of the resources of paths in the same cluster is a much tighter lower bound than the meet of all the paths in $\mathscr{P}_{vd}$ illustrated on Figure 6.1.(b).

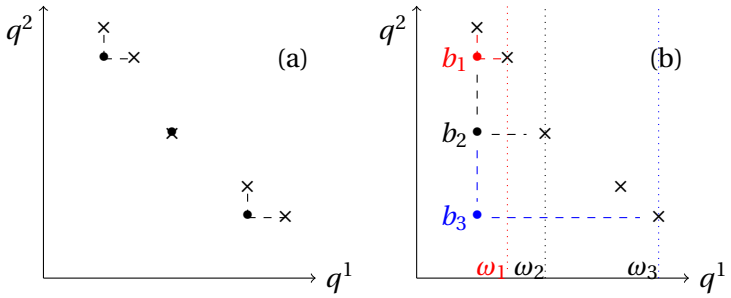


Figure 6.2 – Lower bounds on clusters of paths (a) and conditional lower bounds (b).

We can turn this idea into a test for the enumeration algorithms. Suppose that for each vertex v, we have a set B_v of resources in $\mathscr{R}$ such that, for each v-d path P, there exists a resource $b \in B_v$ satisfying $b \leqslant q_P$. We can formulate the following *clustered lower bounds test*.

(Clu) If there exists b in b_v such that $\rho(q_P \oplus b_v) = 0$ and $c(q_P \oplus b_v) \leq c_{od}^{UB}$, then extend P.

Let b_v be a lower bound that can be used in the single bounds test (Low). Then if the bound set B_v is well chosen, then we have $b > b_v$ for each $b \in B_v$, and the number of paths extended by the enumeration algorithms when using the clustered lower bounds test (Clu) is smaller than the number of paths extended when using the single lower bound test (Low).

Proposition 6.1. *Suppose that for each vertex, we have a set B_v of resources in $\mathscr{R}$ such that, for each v-d path P, there exists a resource $b \in B_v$ satisfying $b \leqslant q_P$. Then Theorem 4.3 and Theorem 4.7 on enumeration algorithms convergence remain true if the clustered lower bounds test (Clu) is used instead of the single lower bound test (Low).*

Proof. We only have to prove that a path P that does not satisfy the test is not the subpath of an optimal o-d path. Let Q be a v-d path. There exists $b \in B_v$ such that $b \leqslant q_Q$, and thus $q_P \oplus b \leqslant q_{P+Q}$. The fact that P does not satisfy the new test and the fact that ρ and c are isotone imply that $\rho(q_{P+Q}) = 0$ and $c(q_{P+Q}) \leq c_{od}^{\text{opt.}}$ cannot be both satisfied, where $c_{od}^{\text{opt.}}$ is the cost of an optimal solution. P can therefore not be the subpath of an optimal path. □

Figure 6.2.(b) shows another technique to compute lower bounds on the resources of paths in subsets of $\mathscr{P}_{vd}$. Here, we consider a STANDARD RESOURCE CONSTRAINED SHORTEST PATH PROBLEM such that the first component q^1 of a resource q corresponds to its cost and the second component is used to test feasibility. We suppose to have in input three costs ω_1, ω_2,

and ω_3, and we consider lower bounds b_1, b_2, and b_3 such that b_i is a lower bound on the resource of all v-d paths P whose cost q_P^1 is non greater than ω_i. Then given an o-v path P and its cost q_P^1, then it is the subpath of an optimal path if and only if there exists a v-d path Q such that $q_P^1 + q_Q^1 < c_{od}^{UB}$ and $\rho(q_P + q_Q) = 0$. As a consequence, given i such that $\omega_{i-1} < c_{od}^{UB} \leq \omega_i$, path P can be the subpath of an optimal path only if $\rho(q_P \oplus b_i) = 0$. This test is *conditional* to the cost.

We now formalize this idea. We assume to have a *weight* morphism ω from $(\mathscr{R}, \oplus, \leqslant)$ to $(\mathbb{R}, +, \leq)$ such that there is no cycle C of negative weight $\sum_{a \in C} \omega(q_a)$. As ω is a morphism, we have $\omega(q_P) = \sum_{a \in P} \omega(q_a)$. Moreover, we assume to have a scalar ω^{UB} such that $\omega(q_P) > \omega^{UB}$ implies that P is not an optimal feasible path. Note that such a scalar always exists if the set of optimal paths is finite, which is always the case if we consider only elementary paths. For each vertex v in V, let n_v be an integer, and $\omega_v^1 < \omega_v^2 < \ldots < \omega_v^{n_v} < \omega^{UB} \leqslant \omega_v^{n_v+1} = +\infty$ be a sequence of real numbers such that ω_v^1 is the minimum weight of a v-d path, which is well defined because there is no cycle of negative weight. Finally, for each vertex v, we suppose to have a set of bounds $b_v^1, \ldots, b_v^{n_v}$ such that b_v^i is a lower bound on the resource of any v-d path P such that $\omega(q_P) < \omega_v^{i+1}$. We can now define the *conditional lower bounds test*.

(Con) If $\rho(q_P \oplus b_v^i) = 0$ and $c(q_P \oplus b_v^k) \leq c_{od}^{UB}$, where i is the minimum index such that $\omega_v^{i+1} \geq \omega^{UB} - \omega(q_P)$, then extend P.

If the ω_v^i and the b_v^i are well chosen, the conditional lower bounds test, the number of paths extended by the conditional lower bounds test is smaller (Con) than the number of paths extended by the single lower bound test (Low). Clustering and conditional lower bounds have both their advantages. On the one hand, as illustrated on Figure 6.2, the clustered lower bounds tend to be tighter than the conditional lower bounds when using the same number of bounds. Besides, as it requires no morphism ω, the clustering approach applies to a wider range of problems. On the other hand, during one clustered lower bounds test (Clu), operator $\oplus$, and functions c and ρ are called $|B_v|$ times, while they are called only once during a conditional test (Con). Another advantage of the conditional lower bounds is that they can be computed faster in a preprocessing, as we will see in this chapter.

Proposition 6.2. *Let ω_v^i and b_v^i be defined as above. Then Theorem 4.3 and Theorem 4.7 on enumeration algorithms convergence remain true if the conditional lower bounds test (Con) is used instead of the single lower bound test (Low).*

Proof. If path P is optimal, then $\omega(q_P) \leq \omega^{UB}$. As a consequence, if a path P is the subpath of an optimal o-d path $P + Q$, then $\omega(Q) \leq \omega^{UB} - \omega(q_P)$. Therefore, if b' is a lower bound on the resources of the v-d paths Q such that $\omega(Q) \leq \omega^{UB} - \omega(q_P)$, then path P needs to be extended only if $\rho(q_P \oplus b') \leq 0$ and $c(q_P \oplus b') \leq c_{od}^{UB}$. □

The objective of this chapter is to introduce standard procedures to compute the sets of bounds used by the clustered lower bounds and conditional lower bounds tests. These procedures use the bounding algorithms of Chapter 4 in a blown up version of the initial graph that we call a state graph. Numerical experiments on the instances of Chapter 4 and Chapter 5 show the increased performance of the enumeration algorithms of Chapter 4 using these new tests.

Chapter 9 shows that the clustered lower bounds test enables to solve faster the subproblem of our column generation approach to the CREW PAIRING PROBLEM.

Chapter 6 is organized as follows.

- Section 6.1 introduces the notion of state graph, and shows that clustered sets of bound can be obtained using the algorithms of Section 4.2 in a state graph.
- Section 6.2 provides a generic method to build state graphs relying on a clustering subroutine.
- Section 6.3 gives a method to build a specific type of state graphs that enable to obtain conditional lower bounds.
- Section 6.4 briefly compares the advantages of using clustered or conditional bound tests.
- Section 6.5 shows the performances of the clustered and conditional lower bound tests on the instances considered in the previous chapters.

6.1 Notion of state graph

A graph $\mathscr{D} = (\mathscr{V}, \mathscr{A})$ is a *state graph on a graph* $D = (V, A)$ if there exists a surjective mapping $\theta : \mathscr{V} \to V$ such that, first, given an arc $(\vartheta_1, \vartheta_2)$, either $(\theta(\vartheta_1), \theta(\vartheta(2))$ is an arc in A or $\theta(\vartheta_1) = \theta(\vartheta_2)$, and second, for each v-d path P in D, there exists at least one path π in $\mathscr{D}$ such that $\theta(\pi) = P$, where $\theta(\pi)$ is the path obtained by taking the images of the successive vertices ϑ in π by θ, and removing two successive copies of the same vertex in the path. Vertices $\vartheta \in \mathscr{V}$ and arcs $\alpha \in \mathscr{A}$ of a state graph are respectively called *state vertices* and *state arcs*. For each vertex $v \in V$, we denote $\mathscr{V}_v$ the set $\theta^{-1}(v)$ of state vertices of v. An example of state graph is plotted on Figure 6.3.

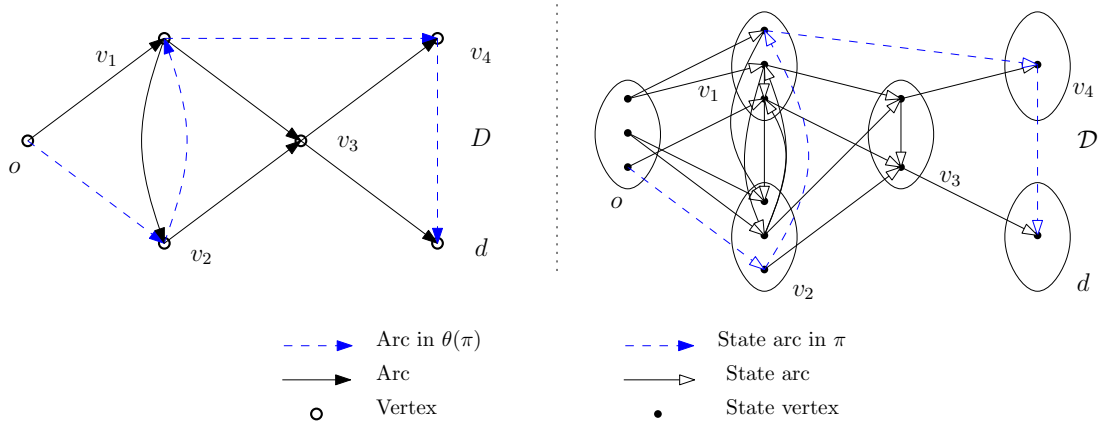


Figure 6.3 – A path π is a state graph $\mathscr{D}$ on graph D and the corresponding path in D.

Note that there are two types of arcs $(\vartheta_1, \vartheta_2)$ in a state graph: those such that $(\theta(\vartheta_1), \theta(\vartheta_2))$ is an arc in A, and those such that $\theta(\vartheta_1) = \theta(\vartheta_2)$. We define the *resource* $q_{(\vartheta_1,\vartheta_2)}$ of an arc $(\vartheta_1, \vartheta_2)$ in the state graph $\mathscr{D}$ to be equal to the resource $q_{(\theta(\vartheta_1),\theta(\vartheta_2))}$ of the corresponding arc $(\theta(\vartheta_1), \theta(\vartheta_2))$ in the initial graph if $\theta(\vartheta_1) \neq \theta(\vartheta_2)$, and to the neutral element 0 otherwise. With

this definition, the resource of a path π in $\mathscr{D}$ is equal to the resource of the corresponding path $\theta(\pi)$ in G.

The definitions of Section 4.2 can be used in $\mathscr{D}$. We define $\left(b_\vartheta^{\ell^*}\right)$ by applying Equation (4.7) in $\mathscr{D}$, where d is replaced by the state vertex ϑ such that $\theta(\vartheta) = d$.

Lemma 6.3. *Let $\mathscr{D}$ be a state graph on a graph D. Then for each v-d path P in D, we have*

$$b_v^{\ell^*} \leqslant b_\vartheta^{\ell^*} \leqslant q_P$$

where $\vartheta \in \mathcal{V}_v$ is the origin of a path π such that $\theta(\pi) = P$.

Lemma 6.3 shows that the set of bounds $B_v = \{b_\vartheta | \vartheta \in \mathcal{V}_v\}$ satisfies the hypotheses of Proposition 6.1, and can thus be used in the the clustered lower bound test (Clu).

Proof. Let $(b_\vartheta)_{\vartheta \in \mathcal{V}}$ be a solution of Equation (4.8) on $\mathscr{D}$. Let P be a v-d path in D and π be a path in D such that $\theta(\pi) = P$. The origin ϑ of π satisfies $\theta(\vartheta) = v$. By Theorem 4.12, we have $b_\vartheta \leqslant q_\pi = q_P$, which gives the result. □

6.2 Clustering state graphs

This section provides a standard procedure to build a state graph on a graph D, and thus to obtain lower bounds for the clustered lower bound test through Lemma 6.3. For clarity, we give a state graph building procedure that applies to *acyclic* digraphs D, and then extend it to graph with cycles.

6.2.1 Clustering state graph for acyclic graphs

The quality of the bounds provided by a state graph are conditioned by the similarity of the resources of the ϑ-ϑ_d paths. In this section we assume to have a *clustering procedure* cl which, given a set S of m resources $q_1, \ldots, q_m$ and an integer $\kappa < m$ returns a partition of S into κ clusters $S_1, \ldots, S_\kappa$ such that two resources q_i and q_j in the same cluster are "similar". Notions of similarity leading to efficient clustering procedures and tight bounds in state graphs are introduced in Section 6.2.3, and clustering algorithms are discussed in Section 6.2.4.

Let D be an acyclic digraph. We start by choosing a topological order $v_1, v_2, \ldots, v_n$ on the vertices V. In order to control the state graph size, we fix a maximum number κ of state vertices on a given vertex. We inductively build a state graph $\mathscr{D} = (\mathcal{V}, \mathscr{A})$ on D as follows. Initially, $\mathcal{V}_v$ is empty for each vertex v except the destination $d = v_n$, and $\mathcal{V}_d$ is the singleton $\{\vartheta_d\}$ with $b_{\vartheta_d} = 0$. For k decreasing from $n-1$ to 1, repeat the following operations:

- Build a resource set S containing $q_{(v_k,v)} \oplus b_\vartheta$ for each (v_k, v) in $\delta^+(v_k)$ and $\vartheta \in \mathcal{V}_v$.
- Extract from S a set S' by removing all resources q dominated by another resource q' in S.
- If S' has less than κ elements, add a state vertex $\vartheta(q)$ to $\mathcal{V}_{v_k}$ with resource $b_\vartheta = q$ for each element q in S'. Otherwise :
  - Partition S' into κ clusters $(S_1, \ldots, S_\kappa)$ using cl.

  - Add a state vertex ϑ to $\mathcal{V}_{v_k}$ for each cluster S_j with resource $b_\vartheta = \bigwedge_{q \in S_j} q$. Denote $\vartheta(q)$ the state vertex of the cluster S_j such that $q \in S_j$
- For each (v_k, v) in $\delta^+(v_k)$ and $\vartheta' \in \mathcal{V}_v$ such that $q = q_{(v_k,v)} \oplus b_{\vartheta'} \in S'$, add to $\mathscr{A}$ a state arc between $\vartheta(q)$ and ϑ'. For each (v_k, v) in $\delta^+(v_k)$ and $\vartheta \in \mathcal{V}_v$ such that $q \in S' \setminus S$, add a state arcs between $\vartheta(q')$ and ϑ' where $q' \in S'$ dominates q.

Proposition 6.4. *At the end of the algorithm, $\mathscr{D}$ is a state graph of D.*

Proof. A straightforward induction shows that at the end of step k, the graph composed of the already built vertices and arcs is a state graph on the subgraph of D induced by $v_1, \dots, v_k$. □

6.2.2 Dealing with cycles

In this section, we now extend the algorithm of the previous section to graph with cycles but no loop. The idea of the algorithm is to split the digraph into two acyclic digraphs, and to apply the algorithm of the previous section in each acyclic digraph several times.

We therefore define three procedures: the *extended clustering procedure* which takes in input a set $\widetilde{S}$ of fixed resources $\tilde{q}_1, \dots, \tilde{q}_{k_1}$, a set of generic resources S and a number k_2 of new clusters, with $|S| \geq k_2$, and returns a partition of $\widetilde{S} \cup S$ in $k_1 + k_2$ clusters S_i such that for each i in $[k_1]$, the fixed resource $\tilde{q}_i$ is in S_i. The *directed state building procedure*, which is a generalization of the algorithm of Section 6.2.1 that takes in input an acyclic partial graph (V, A'), a topological ordering $v_1, \dots, v_n$ of V, a maximum number of new state vertices per vertex κ, and a partially built state graph $(\mathcal{V}, \mathscr{A})$, and returns new state vertices and state arcs for the state graph taken in input. This procedure calls the extended clustering procedure. Finally, the *cyclic state graph building procedure* calls several times the directed state building procedure to build the state graph. It takes in input a digraph $D = (V, A)$, two positive integers κ_1 and κ_2, and a numbering $(i_v)_v$ of the vertices of V by distinct integers i_v such that $d = v_{|V|}$, and returns a state graph with at most $\kappa_1 + \kappa_2$ state vertices per vertex of the initial graph.

The similarity measures and algorithms in Section 6.2.3 and Section 6.2.4 enable to build the *extended clustering procedure* cl′. We start with the cyclic state graph building procedure, and then detail the directed state building procedure.

We now suppose to have the directed state building procedure and expose the *cyclic state graph building procedure*. The successive phases of this procedure are illustrated in different colors on Figure 6.4. Let $D = (V, A)$, κ_1, κ_2, $(i_v)_v$ be the input of the procedure. Let A_1 to be the set of arcs (u, v) such that $i_u < i_v$ and A_2 the set of arcs (u, v) such that $i_v < i_u$. Sets A_1 and A_2 form a partition of A such that both (V, A_1) and (V, A_2) are acyclic graphs, and the numbering $(i_v)_v$ induces a topological ordering on (V, A_1) and a reverse topological ordering on (V, A_2). We can now expose the procedure.

- Initialize $\mathscr{A}_a$ to $\emptyset$ for each arc a in A. Initialize $\mathcal{V}_d$ to the singleton $\{\vartheta\}$ with $b_\vartheta = 0$, and $\mathcal{V}_v$ to the empty set for $v \neq d$.
- *First phase:* run the directed state building procedure on (V, A_1) with the ordering induced by $(i_v)_v$, $\kappa = \kappa_1$, $\mathcal{V}_v$ and $\mathscr{A}_a$ as input. Update $(\mathcal{V}, \mathscr{A})$ with the newly built state vertices and arcs.

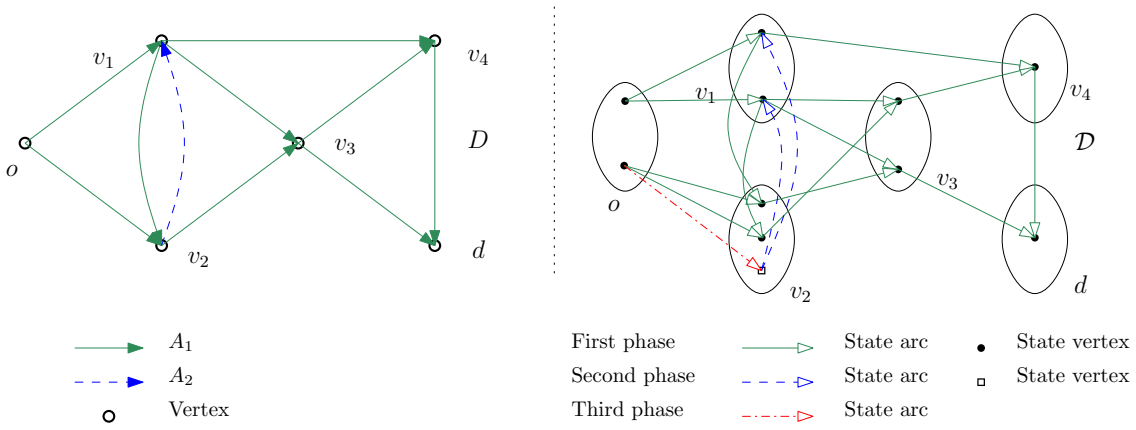


Figure 6.4 – Cluster state graph building on a graph with cycle $\kappa_1 = 2$ and $\kappa_2 = 1$

- *Second phase:* run the directed state building procedure on (V, A_2) with the reversed ordering induced by $(i_v)_v$, $\kappa = \kappa_2$, $\mathcal{V}_v$ and $\mathscr{A}_a$ as input. Update $(\mathcal{V}, \mathscr{A})$ with the newly built state vertices and arcs.
- *Third phase:* run the directed state building procedure on (V, A_1) with the reversed ordering induced by $(i_v)_v$, $\kappa = 0$, $\mathcal{V}_v$ and $\mathscr{A}_a$ as input. Update $(\mathcal{V}, \mathscr{A})$ with the newly built state arcs.

We can now focus on the *directed state building procedure*. This is an analogue of the procedure of Section 6.2.1 where some state vertices already exist at the beginning of the procedure. We also monitor which state vertex has been extended along which arc. Let (V, A'), the ordering $v_1, \dots, v_n$, maximum size κ, and partialbuilt state graph $(\mathcal{V}, \mathscr{A})$ be the input of the procedure. For k decreasing from n to 1, repeat the following operations.

- Build a resource set S containing $q_a \oplus b_\vartheta$ for each $a = (v_k, v)$ in $\delta^+(v_k)$ and $\vartheta \in \mathcal{V}_v$ such that there is no arc ending in ϑ in $\mathscr{A}_a$.
- Extract from S a set S' by removing all resources q dominated by another resource q' in S or by a resource in $\mathcal{V}_v$.
- If S' has less than κ elements, add a state vertex $\vartheta(q)$ to $\mathcal{V}_{v_k}$ with resource $b_\vartheta = q$ for each element q in S'. Otherwise :
  - Call cl$'$ with $\{b_\vartheta | \vartheta \in \mathcal{V}_v\}$ as set of fixed resources, S' as set of generic resources, and κ as maximum number of new vertices. Let $(S_1, \dots, S_\kappa)$ be the new clusters built.
  - For each ϑ in $\mathcal{V}_v$, update b_ϑ to the meet of the resources in their cluster.
  - Add a state vertex ϑ to $\mathcal{V}_{v_k}$ for each cluster S_j with resource $b_\vartheta = \bigwedge_{q \in S_j} q$. Denote $\vartheta(q)$ the state vertex of the cluster S_j such that $q \in S_j$
- For each $A = (v_k, v)$ in $\delta^+(v_k)$ and $\vartheta' \in \mathcal{V}_v$ such that $q = q_{(v_k,v)} \oplus b_{\vartheta'} \in S'$, add to $\mathscr{A}_a$ a state arc between $\vartheta(q)$ and ϑ'. For each (v_k, v) in $\delta^+(v_k)$ and $\vartheta \in \mathcal{V}_v$ such that $q \in S' \setminus S$, add a state arc between $\vartheta(q')$ and ϑ' where $q_{\vartheta'}$ dominates q. Such a ϑ' always exist by definition of S'.

When $\mathcal{V}_v$ and $\mathscr{A}_a$ are initially empty, the directed state building procedure coincides with the algorithm of Section 6.2.1.

Proposition 6.5. *At the end of the cyclic state graph building procedure, $\mathscr{D} = (\mathcal{V}, \mathscr{A})$ is a state graph on D.*

The quality of the bounds obtained depends on the numbering $(i_v)_v$ chosen. On the examples dealt with in this dissertation, we obtained good lower bounds by choosing a numbering as follows: we define a mapping f from $\mathscr{R}$ to $\mathbb{R}_+^*$ that "averages the different components of the resource" and run a depth first search starting from d and with arc weights $f(q_a)$. The processing order of the vertices gives the numbering. Using this technique, arcs in A_1 are more susceptible to be on an optimal path, as they "are in the direction of d". We therefore suggest to use a κ_1 larger than κ_2. In this dissertation, given a maximum number of state per vertices κ, we use $\kappa_1 = 3\kappa/5$ and $\kappa_2 = 2\kappa/5$.

Proof of Proposition 6.5. Using a straightforward induction on the length of paths P, the existence of a path π such that $\theta(\pi) = P$ follows directly from the fact that, for any state vertex ϑ in $\mathcal{V}_v$ and arc a in $\delta^-(v)$, there is a state arc α in $\mathscr{A}_a$ ending in ϑ. □

Remark 6.1. Note that in contrast with the acyclic case, and due to the third phase where no additional states are added, the $(b_v)_v$ obtained do not necessarily satisfy Equation (4.8) at the end. Thus, Equation (4.8) needs to be solved on the state graph to obtain the lower bounds.

6.2.3 Choice of similarity measure for clustering

We use in Section 6.2 a clustering procedure to build a clustering state graph. The clustering procedure takes in input a set S of m resources, returns a partition of S in κ clusters $S_1, \ldots, S_\kappa$ such that two resources q_i and q_j in the same cluster are "similar". There is some latitude in the choice of this clustering procedure, because the validity of the state graph built does not depend on the partition returned. Nonetheless, a good clustering choice leads to better bounds. The design of such a clustering procedure is in two steps: first, we explain how to choose a similarity measure for partitions that leads to good partitions when minimized, and second we explain how to choose an algorithm to minimize such a measure.

Remark 6.2. Note that in the case of graph with cycles, some resources are already assigned to some clusters. Nonetheless, the procedures presented in the remaining of the section are straightforwardly generalized to that case.

Measures based on similarity

A first option is to build clusters of "similar resources". We suppose to have a non negative function $d(q, \tilde{q})$ that measures the similarity between two resources. The smaller $d(q, \tilde{q})$, the more similar are q and $\tilde{q}$. Function d is defined specifically for the lattice ordered monoid. When $\mathscr{R} = \mathbb{R}^n$, taking the L^1 or L^2 distances are standard choices for d. One first choice is to

minimize the sum of the distances of resources to their clusters meet.

$$\sum_{k=1}^{\kappa} \sum_{q \in S_k} d\left(q, \bigwedge_{\tilde{q} \in S_K} \tilde{q}\right) \tag{6.1}$$

An alternative is to minimize the maximum distance of a resource to its cluster meet.

$$\max_{k=1}^{\kappa} \max_{q \in S_k} d\left(q, \bigwedge_{\tilde{q} \in S_K} \tilde{q}\right) \tag{6.2}$$

The measure of Equation (6.1) gives the same weight to each input resources while the measure of Equation (6.2) produces clusters of similar diameter. Both measures give bounds of similar quality in practice. On the contrary, the following measure must be avoided.

$$\sum_{k=1}^{\kappa} \max_{q \in S_k} d\left(q, \bigwedge_{\tilde{q} \in S_K} \tilde{q}\right)$$

This measure is easier to minimize – it can be minimized in polynomial time for resource in a totally ordered set. But it leads to a partition of bad quality. Indeed, it tends to produce one large cluster and $k-1$ very small clusters.

Measures based on bounds size

As good lower bounds are "as large as possible", an alternative option is maximize the lower bounds resources size. Let μ be an isotone mapping from $\mathscr{R}$ to $\mathbb{R}$. Maximizing the following measures produces good results when resources are random variables.

$$\sum_{k=1}^{\kappa} \mu\left(\bigwedge_{\rho \in S_k} \rho\right) \tag{6.3}$$

6.2.4 Clustering algorithm

We now consider heuristic algorithms to find a partition of S that minimizes one of the measures of Equations (6.1) and (6.2), or to maximize the one of Equation (6.3). We tested a local search metaheuristic, and an adapted version of the k-means clustering algorithm [118, 121], which is popular in the Machine Learning community.

Our *local search* relies on two neighborhoods, explored with equal probability. Two partitions are neighbors in the *swap* neighborhood if the second partition can be obtained from the first partition by exchanging the clusters of two elements. They are neighbors in the *add-remove* neighborhood if the second partition can be obtained from the first by moving one element from one cluster to another. Neighborhoods are searched randomly, and we use a simulated annealing criterion to decide if a move is accepted. The temperature is initially set to one and updated geometrically.

The adapted k-means clustering algorithm starts with an arbitrary partition and repeats the two following steps while the partition is modified during the second step.

- Compute the meet m_k of each cluster S_k.
- Affect each resource q in S to the cluster S_k with minimum $d(q, m_k$.

This difference between this algorithm and the usual k-means clustering algorithm are that resources are elements of a lattice instead of being vectors, and that the barycenter operator is replaced here by the meet operator.

On the different problems on which we used state graphs, we obtained faster convergence and better solutions using the simulated annealing.

Remark 6.3. The cluster of some resources can be fixed in these algorithms, and they can therefore be used as the clustering procedure cl′ of Section 6.2.2.

6.3 Conditional state graphs

In this section, we explain how to build the bounds used by the conditional lower bounds test. We therefore assume that we have a *weight* morphism ω from $(\mathscr{R}, \oplus, \leqslant)$ to $(\mathbb{R}, +, \leq)$ such that there is no cycle C of negative weight $\sum_{a \in C} \omega(q_a)$. In Section 6.3.1, we first suppose to have the thresholds $\omega_v^1 < \omega_v^2 < \ldots < \omega_v^{n_v} < \omega^{UB} \leqslant \omega_v^{n_v+1} = +\infty$ and show how, using these thresholds, we can build a state graphs leading to bounds b_v^i satisfying the desired properties. We then explain in Section 6.3.1 how to choose these thresholds. We denote ω_a the resource $\omega(q_a)$ of an arc a, and ω_P the resource $\omega(q_P)$ of a path P.

6.3.1 Conditional state graph

For each vertex v in V, let n_v be an integer, and $\omega_v^1 < \omega_v^2 < \ldots < \omega_v^{n_v} < \omega^{UB} \leqslant \omega_v^{n_v+1} = +\infty$ be a sequence of real numbers such that ω_v^1 is the minimum weight of a v-d path, which is well defined because there is no cycle of negative weight. We build a state graph $\mathscr{D}$ associated to ω and $\left(\omega_v^k\right)$ as follows.

- For each vertex v and weight ω_v^k, we build a state vertex denoted ϑ_v^k. Thus $\mathscr{V}_v = \{\vartheta_v^k : k = 1, \ldots, n_v\}$
- For each vertex v and index $k \in \{1, \ldots, n_{v-1}\}$, we build arcs $(\vartheta_v^{k+1}, \vartheta_v^k)$
- For each arc $(u, v) \in A$ and index $j \leq n_v$, we build a state arc $(\vartheta_u^i, \vartheta_v^j)$ where i is such that $\omega_u^i \leq \omega_{(u,v)} + \omega_v^j < \omega_u^{i+1}$.

The definitions of ω_u^1 and $\omega_u^{n_u+1} = +\infty$ ensure the existence of such a j. Indeed, as ω_v^1 (resp. ω_u^1) is the minimum weight of a v-d path (resp. a u-d path), we necessarily have $\omega_u^1 \leq \omega_{(u,v)} + \omega_v^1 \leq \omega_{(u,v)} + \omega_v^j$ for all j.

Proposition 6.6. *The digraph $\mathscr{D} = (\mathscr{V}, \mathscr{A})$ built as above is a state graph on D.*

Proof. The mapping θ is defined through sets $\mathscr{V}_v$ for each vertex v. A straightforward induction on the length of paths shows that for each path P in D, there exists a path π such that $\theta(\pi) = P$. □

Lemma 6.7. *Let $\mathscr{D} = (\mathscr{V}, \mathscr{A})$ be built as above. Then $\mathscr{D}$ is a state graph on D, and for each v-d path P such that $\omega(q_P) < \omega_v^{k+1}$, we have*

$$b_{\vartheta_v^k}^{\ell^*} \leqslant q_P,$$

where $z_{\vartheta_v^k}^{\ell^*}$ and ℓ^* are defined as in Section 4.2.

Lemma 6.7 implies that the bounds $b_v^k = b_{\vartheta_v^k}^{\ell^*}$ can be used for the conditional lower bound test.

Proof. An induction on the length of the paths enables to prove that for each v-d path P of weight $\omega_P < \omega_v^{k+1}$, there exists a path π in $\mathscr{D}$ from ϑ_v^k to ϑ_d^1. Theorem 4.12 enables to conclude. □

Remark 6.4. Note that as $\mathscr{D}$ is a state graph, Lemma 6.3 ensures that the $b_{\vartheta_v^k}^{\ell^*}$ can also be used in the clustered lower bound test. Nonetheless, using them for the conditional lower bound test enables to reduce the number operations $\oplus$, c, and ρ performed, and thus to reduce the computation time.

Two heuristic choices of (ω_v^k)

It remains to explain how the *thresholds* ω_v^k can be judiciously chosen. Recall that ω_v^1 denotes the minimum weight of a v-d path. Let ω_v^{LB} denote the minimum weight of an o-v path. The weights ω_v^1 and ω_v^{LB} can be computed for each vertex v using a Dijkstra like algorithm as they are solution of a deterministic STANDARD SHORTEST PATH PROBLEM. Upper bound ω^{UB} is then initialized. A default choice is $\omega(\sum_{a\in A} q_a)$, but a better choice can be done in practice. For instance, when ω is the cost function c, the cost of a feasible path found heuristically can be used. A v-d path Q such that $\omega(Q) > \omega^{UB} - \omega_v^{LB}$ cannot lead to a feasible path of weight smaller than ω^{UB}. As a consequence, the thresholds must be chosen in $[\omega_v^1, \omega^{UB} - \omega_v^{LB}]$. We know present two methods to choose thresholds in this interval.

A first approach consists in arbitrarily choosing n_v and setting $\omega_v^k = \frac{(n_v-k)\omega_v^{LB}+(k-1)\omega_v^{UB}}{n_v-1}$.

The parameter n_k enables to choose a tradeoff between the quality of the bounds on the one hand, and the memory required to store them and the time required to compute a solution of Equation (4.8) on $\mathscr{D}$ on the other hand. The limit of this method is that in the resulting conditional state graph, there may be paths π from ϑ_v^k to the destination such that $\omega(q_\pi)$ is much greater than ω_v^k. In this case, the lower bounds provided by Lemma 6.7 are of poor quality. The next method consists in controlling the weight of paths from ϑ_v^k to the destination via a parameter Δ.

A second approach consists in fixing a gap Δ and choosing thresholds ω_v^k such that for each arc (u,v) and weight ω_v^j on vertex v, there exists a weight ω_u^i on vertex u such that $\omega_u^i \leq \omega_{(u,v)} + \omega_v^j \leq \omega_u^i + \Delta$.

Elementary computations enable to show that in this case, if there exists a path π from ϑ_v^k to destination, then $\omega(q_P) - \omega_v^k \leq \ell(P)\Delta$ where $P = \theta(\pi)$ and $\ell(P)$ is the length of P.

We now provide an algorithm that enables to build such thresholds. For each vertex v, a set F_v of final weights, a current bound b_v equal to the largest weight in F_v, a set U_v of candidate weights, and a current weight ω_v equal to the smallest weight in U_v are updated during the algorithm. A queue L of vertices to be updated is maintained. Current weight ω_v is initially set to $+\infty$ and current bound b_v to $-\infty$, and sets F_v and U_v are initially empty for each vertex $v \neq d$, while $\omega_d = 0$, $F_d = \emptyset$, and $U_d = \{0\}$. Initially, queue L contains only d. The algorithm ends when L is empty. While L is not empty, the following operations are repeated:

- Extract from L a vertex v of minimum ω_v.
- Add ω_v to F_v and set $b_v = \omega_v$.
- *Extend* ω_v: For each arcs $(u,v) \in \delta^-(v)$, if $b_v + \Delta < \omega_{(u,v)} + \omega_v < \omega^{UB} - \pi_u$, then
  - Add $\omega_{(u,v)} + \omega_v$ to U_u.
  - If $\omega_{(u,v)} + \omega_v < \omega_u$, set $\omega_u = \omega_{(u,v)} + \omega_v$.
  - Add u to L.
- Remove from U_v all elements non greater than $b_v + \Delta$.
- Set $\omega_v = \min U_v$ if $U_v \neq \emptyset$. Otherwise, set $\omega_v = +\infty$.

Proposition 6.8. *If $\omega_a \geq 0$ for each arc a, the algorithm terminates after at most $|V| \left\lceil \frac{\omega^{UB}}{\Delta} \right\rceil$ iterations. At the end of the algorithm, for each arc (u,v) and weight ω_v^j in F_v, there exists a weight ω_u^i in F_u such that $\omega_u^i \leq \omega(u,v) + \omega_v^j \leq \omega_u^i + \Delta$.*

Proof. The following invariants remain true along the algorithm: the current weight ω_v of the current vertex increases during the algorithm, and the weights in U_u for each vertex u are greater than ω_v. Besides, the difference between the value of ω_v at successive visits of v is greater than Δ. □

It remains to explain how to choose Δ. Indeed, the choice of Δ determines the accuracy of the conditional state graph. Therefore, if Δ is chosen too large, the bounds built are not precise and the label algorithm will not discard paths efficiently. On the contrary, if Δ is chosen too small, the conditional graph will be too large, which increases the length of the bounding phase and the memory consumed to store the bounds. Paths whose cost diverges by Δ are approximated by the same bound. *We suggest to use as Δ the average weight of an arc*, as it is of the order of magnitude of the difference between two paths.

Remark 6.5. A simple idea to choose Δ more accurately is the following: *use the minimum quality bounds among those that enable the generalized A* algorithm to ends in a reasonable time*. In the conditional graph approach, bounds quality is selected through parameter Δ. Besides, the limiting criterion for the generalized A* algorithm is memory consumed, which can be controlled by fixing the label queue L maximum size. Δ is then chosen by binary search. This approach is useful when several resource constrained shortest path problems on the same graph are solved, as it is the case for instance in the context of column generation.

6.4 Conditional versus clustered lower bounds

We now outline the pros and the cons of the clustered and the conditional lower bound tests.

The main advantage of the clustered lower bound test is its flexibility. Indeed, as we have seen in Section 6.2.3, we only need to define a similarity measure between pairs of resources, which can easily be done in a wide range of contexts. Besides, when the path problem considered is the subproblem of a column generation approach, the only differences between the problems solved at different iterations is the reduced cost. Thus, if we define a similarity measure between resources that do not take into account this reduced cost, a clustering state graph can be computed once and for all before the column generation, and accelerate the resolution of the subproblem at all steps of the column generation. The main disadvantage of a clustering state graph is that it can be long to compute, as the clustering procedure has to be called several times along the construction of the state graph.

Conditional state graphs have three main advantages. First, they can be built faster than clustering state graph as they do not call the time consuming clustering procedure. Second, the bound produced tend to be of better quality. And third, the conditional lower bounds test on a path P call only once the operator $\oplus$ when the clustered lower bound tests, which call it $O(|\mathcal{V}_v|)$ where v is the destination of P. As a consequence, conditional state graphs tend to lead to smaller computations times than clustering state graphs when both are available. Their disadvantage are that they are less versatile than a clustering state graph, as a morphism ω is required. When such a morphism is available and only one instance of a resource constrained shortest path problem is solved, a conditional state graph is probably a better choice than a clustering state graph.

6.5 Numerical results

We use our `latticeRCSP` library described in Appendix C to solve the instances. State graphs enable to improve algorithm performances in two ways. On instances that can be solved using the generalized A* or the label correcting algorithm but after a large CPU time, if most of the CPU time is spent in the enumeration algorithm and not in the bounding algorithm, then a state graph can enable to reduce the number of paths enumerated and thus the CPU time. On difficult instances that cannot be solved using the generalized A* or the label correcting algorithm, a state graph can enable to reduce the gap between the lower bound and the best path found or even to break the instances.

Besides, for a state graph to be efficient, it often has to be much larger than the initial graph. A good order of magnitude is 100 times larger. As a consequence, the initial instance must not be too large, otherwise the state graph bounds cannot be stored in memory. We therefore focus on small but difficult instances. Section 6.5.1 benchmarks the relative performance of the conditional and clustering state graphs on the deterministic resource constrained shortest path problem with ten constraints of Section 4.3.6. Section 6.5.2 gives the performance of clustering state graphs on difficult instances of the STOCHASTIC SHORTEST PATH PROBLEM with scenario distributions considered in Section 5.4.

All instances considered in this section are difficult instances that cannot be solved to optimal-

ity using the standard label correcting or generalized A* algorithms. Besides, all the instances considered in this section have cycles. Due to the "artificial" choice of a topological order in the construction of clustering state graph on graphs with cycles, clustering state graphs tend to perform better on acyclic digraphs. In Chapter 9, we use the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM algorithms to solve the subproblem of a column generation approach to the CREW PAIRING PROBLEM. Section 9.3.3 shows that the use of a clustering state graph enables to divide by three the total computation time of the column generation.

6.5.1 Resource constrained shortest path with ten constraints

In this section, we consider instances of the deterministic resource constrained shortest path problem of Section 4.3. As we already mentioned at the beginning of the section, to use a state graph, an instances must be not to large, as otherwise the lower bounds cannot be stored in memory. We can see in Table 4.4 that the long5 and square50 instances satisfy these requirements: both have fewer than 3000 vertices and cannot be solved to optimality without state graphs. We use a simulated annealing metaheuristic to minimize objective function (6.1) with the L^1 distance between the vector of resources as d.

Table 6.1 provides the results of a clustering state graph approach to these instances. The first column provides the name of the instances, the second one the strength α of the constraint defined in Equation (4.12). The three next columns concern the state graph. Column κ provides the maximum number of state vertices per vertex provides in the cyclic state graph building algorithms of Section 6.2.2: we use $\kappa_1 = 0.6\kappa$ and $\kappa_2 = 0.4\kappa$. The columns $|\mathcal{V}|$ and $|\mathcal{A}|$ provide the number of vertices and the number of arcs of the state graph. When these three line have the symbol –, it means that no state graphs is used. The next column provides the ratio γ of Remark 4.5 which indicates the performance of the generalized Dijkstra algorithm in the state graph. The column Prep. provides the percentage of the total CPU time spent building the state graph and running the generalized Dijkstra algorithm. The three next columns provide the number of paths extended, the number of paths cut, and the percentage of paths cut by the dominance test in the case of the label correcting algorithms. The next column provides the number of arcs in the solution returned. The gap column provides the gap between the lower bound proved and the best solution found, or the mention opt. if the instance is solved to optimality. The last column provides and the total CPU time. Due to the storage of paths as string in the implementation, the maximum size of the list L that can be kept in memory depends on the number of arcs in the paths stored. Besides, label correcting algorithms require to store non-dominated paths. As a consequence, we use a different maximum size for the different instances and algorithms. We remark that this is a pro of the A* algorithm when compared to the label correcting algorithm: more paths can be considered. This maximum size is indicated in the table caption.

We first remark that larger clustering state graphs always lead to better results as they enable to reduce the gap. Nonetheless, this has a cost in term in of CPU time, as the construction and the bounding algorithm takes time in large state graphs, and then each path test is longer. Another less intuitive fact is that, when the instance is not solved to optimality, larger state

Instance	α	Alg.	κ	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{A}\rvert$	γ	Prep.	Ext.	Cut.	Dom.	ℓ	Gap	CPU (s)
long5	0.5	cor.	–	–	–	2.2	0%	230537	130784	50%	81	60.8%	7.61e+00
			20	22070	46467	1.7	17%	190559	88647	52%	81	55.9%	1.17e+01
			100	104918	200221	1.8	22%	187568	83110	55%	81	53.3%	2.54e+01
			500	482804	835597	1.8	35%	190368	85960	55%	81	49.5%	9.85e+01
		A*	–	–	–	2.2	0%	1424776	849563	–	81	64.4%	9.49e+00
			20	22074	46556	1.8	6%	1276845	553700	–	81	57.9%	3.45e+01
			100	104825	199510	1.8	5%	1218043	436097	–	81	54.5%	1.09e+02
			500	483331	836135	1.8	7%	1165819	331651	–	81	51.7%	4.35e+02
square50	0.5	cor.	–	–	–	2.2	0%	350553	273819	42%	52	44.9%	1.15e+01
			20	41169	86311	1.7	23%	299643	207262	46%	52	36.5%	1.64e+01
			100	186078	358864	1.9	30%	285296	203651	41%	52	35.4%	3.32e+01
			500	820474	1436213	1.9	47%	307779	231886	40%	52	31.6%	1.22e+02
		A*	–	–	–	2.2	0%	9954128	11908301	–	52	44.9%	5.93e+01
			20	41131	86819	1.7	2%	8425382	8850810	–	52	34.9%	1.61e+02
			100	185809	359191	1.8	2%	8146176	8292398	–	52	33.8%	4.73e+02
			500	821845	1438573	1.9	3%	9374152	10748349	–	52	27.9%	1.71e+03

Table 6.1 – Results of clustering state graph approach on resource constrained shortest path with ten constraints and $\alpha = 0.5$. List L maximum size is 2e+05 for label correcting algorithm and 2e+06 for A* algorithm on instance long5, and 8e+06 for A* algorithm on instance square50

graphs can increase the number of paths extended. Indeed, they enable to discard more paths, and more paths can be extended before the list L reaches its maximum size.

For the conditional state graphs, we use the cost $c(q)$ as the morphism to $\mathbb{R}$. Table 6.2 is the analogue of Table 6.1 for conditional state graph. The only difference is that the parameter that control the state graph size is no more κ but Δ, the minimum difference between two state vertices costs. The smaller Δ, the larger the state graph.

Again, we observe that the larger the conditional state graphs, the better the result in terms of gap proved. A less intuitive fact is that the ratio of the paths cut divided by the number of paths extended can decrease for instance not solved to optimality. We can explain this phenomenon by the fact that larger state graphs mean better keys, which means that fewer paths that have to be cut are generated at the beginning of the algorithm. When an instance is solved to optimality, this is of course not true at the end of the algorithm, where paths have to be cut to empty list L. We also note that for large conditional state graph, γ is equal to 1. This comes from the fact that is arc costs are positive and Δ is sufficiently small, the conditional state graphs are acyclic.

Table 6.1 and Table 6.2 confirm empirically our statement of Section 6.4 that conditional state graphs give better performance than clustering state graphs when both are available.

Conditional state graphs also enable to improve algorithm performances on instances with only one constraint. The difference with the ten constraints case is that there are few difficult instances of size sufficiently small for a state graph to be computed. We can see in Table 4.3 that the long20 is the only small instance that is not solved to optimality. Table 6.3 provides results with conditional state graphs of different sizes on this instance. The columns of this

Instance	α	Alg.	Δ	$\lvert\mathscr{V}\rvert$	$\lvert\mathscr{A}\rvert$	γ	Prep.	Ext.	Cut.	Dom.	ℓ	Gap	CPU (s)
long5	0.5	cor.	–	–	–	2.2	0%	230537	130784	50%	81	60.8%	7.61e+00
			100	23558	91384	2.2	6%	167283	50956	82%	81	60.0%	4.91e+00
			10	213567	838389	2.6	41%	137348	24950	100%	81	32.2%	8.14e+00
			1	1172386	4608057	1	73%	133873	22655	100%	81	29.2%	2.05e+01
		A*	–	–	–	2.2	0%	1424776	849563	–	81	64.4%	9.49e+00
			100	23558	91384	2.2	2%	1011810	23631	–	81	62.3%	1.41e+01
			10	213567	838389	2.6	16%	1000137	285	–	81	31.0%	2.32e+01
			1	1172386	4608057	1	37%	1000309	629	–	81	27.8%	4.07e+01
square50	0.5	cor.	–	–	–	2.2	0%	350553	273819	42%	52	44.9%	1.15e+01
			100	24370	92087	2.2	6%	217971	100571	67%	52	41.1%	5.61e+00
			10	214156	826929	2.5	41%	158700	57177	53%	52	15.1%	8.85e+00
			1	1180357	4568978	1	73%	165658	71086	42%	52	12.3%	2.24e+01
		A*	–	–	–	2.2	0%	9954128	11908301	–	52	44.9%	5.93e+01
			100	24370	92087	2.2	1%	4627965	1255974	–	52	40.2%	5.96e+01
			10	214156	826929	2.5	3%	6772542	5545130	–	52	7.5%	1.11e+02
			1	1180357	4568978	1	3%	30098520	60197088	–	52	opt	4.73e+02

Table 6.2 – Results of conditional state graph approach on resource constrained shortest path with ten constraints and $\alpha = 0.5$. List L maximum size is 2e+05 for label correcting algorithm and 2e+06 for A* algorithm on instance long5, and 8e+06 for A* algorithm on instance square50

table are identical to those of Table 6.2. The label correcting algorithm without state graph exhibits different performances than those provided in Table 4.3 as we allowed here the list L to contain up to 2e+06 paths while it was limited to 1e+05 paths in Table 4.3. We can see that a larger state graph, obtained for a smaller parameter Δ enables both to solve the instance to optimality and to reduce the number of paths extended without increasing the CPU time. Once again, we observe that in small dimension, the label correcting algorithms performs better than generalized A* algorithm. We also observe that when larger state graph are used on instance not solved by the A* algorithm, fewer paths are cut as a larger state graph gives better bounds, thus better keys, and thus expand more promising paths that cannot be cut.

Instance	α	Alg.	Δ	$\lvert\mathscr{V}\rvert$	$\lvert\mathscr{A}\rvert$	γ	Prep.	Ext.	Cut.	Dom.	ℓ	Gap	CPU (s)
long20	0.5	A*	–	–	–	1.6	0%	1064129	128268	–	435	53.5%	9.72e+00
			500	5122	15376	1.6	0%	1072512	145034	–	430	48.2%	1.66e+01
			100	184405	726113	1.6	11%	999995	0	–	424	38.8%	2.00e+01
			20	878542	3479437	1.5	39%	999995	0	–	391	4.1%	2.88e+01
		cor.	–	–	–	1.6	0%	2118701	1363841	98%	399	2.9%	3.14e+01
			500	5122	15376	1.6	0%	2355602	1700568	89%	393	opt	4.16e+01
			100	184405	726113	1.6	5%	2339942	1690032	88%	393	opt	4.48e+01
			20	878542	3479437	1.5	37%	916185	540546	96%	393	opt	3.12e+01

Table 6.3 – Results of conditional state graph approach on resource constrained shortest path with one constraint and $\alpha = 0.5$. List L maximum size is 2e+06.

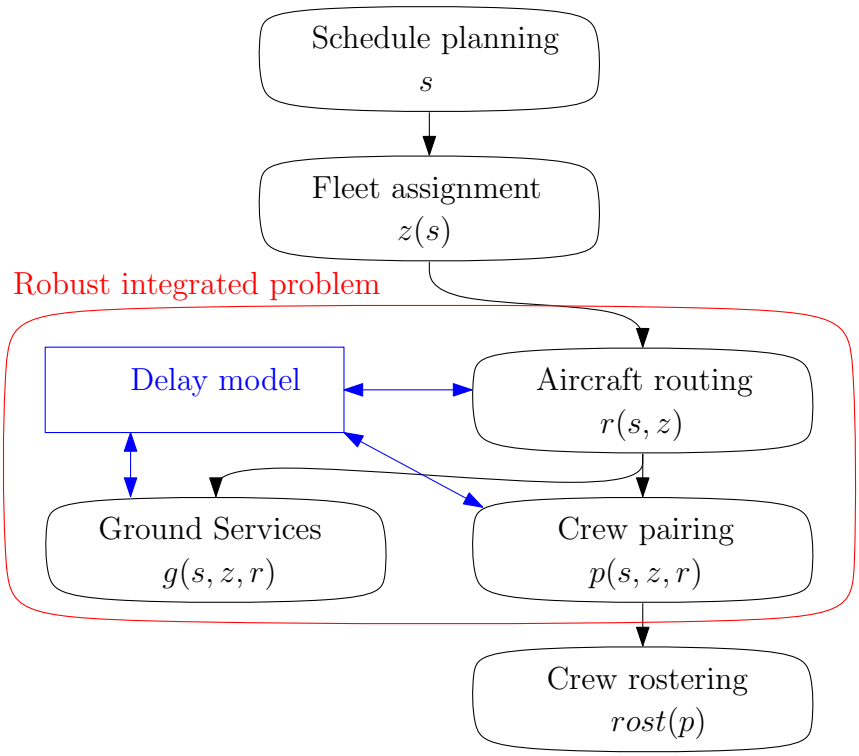
Instance	α	Alg.	κ	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{A}\rvert$	γ	Prep.	Ext.	Cut.	Dom.	ℓ	Gap	CPU (s)
square50	0.01	A*	–	–	–	1.7	3%	59596	19238	–	63	50.2%	2.01e+00
			10	21766	44058	3.6	62%	68251	36547	–	60	16.4%	2.22e+01
			100	194619	342671	6.1	46%	67595	35234	–	60	15.0%	1.10e+02
		cor.	–	–	–	1.7	1%	45783	27704	25%	66	23.2%	4.97e+00
			10	21584	42946	3.7	60%	67671	52593	31%	61	8.2%	2.23e+01
			100	194675	342735	5.5	52%	67059	53520	29%	70	7.1%	9.42e+01
	0.1	A*	–	–	–	1.7	3%	55076	10197	–	61	20.5%	1.89e+00
			10	21724	44025	3.7	68%	56803	13652	–	61	14.2%	2.07e+01
			100	194610	342593	5.2	49%	56324	12692	–	61	13.2%	1.00e+02
		cor.	–	–	–	1.7	2%	39202	14463	48%	61	15.9%	3.07e+00
			10	21717	43244	3.6	67%	43520	20817	39%	61	9.8%	1.98e+01
			100	194329	341602	4.9	60%	44931	22996	37%	61	9.0%	8.04e+01
long5	0.01	A*	–	–	–	1.8	2%	55558	11128	–	100	41.0%	1.80e+00
			10	11336	23235	3.1	49%	59776	19562	–	100	26.0%	1.45e+01
			100	106513	191983	3.3	31%	62406	24822	–	106	24.1%	8.28e+01
		cor.	–	–	–	1.8	0%	46209	17876	69%	100	35.8%	6.39e+00
			10	11282	22151	2.7	50%	52816	30060	43%	106	24.2%	1.33e+01
			100	106078	189078	3.1	42%	53473	32232	38%	100	19.1%	5.89e+01
	0.1	A*	–	–	–	1.8	2%	51382	2774	–	102	30.0%	1.75e+00
			10	11264	22435	2.9	49%	51358	2726	–	106	23.9%	1.37e+01
			100	106630	193825	3.6	34%	51652	3314	–	106	23.1%	7.68e+01
		cor.	–	–	–	1.8	1%	35409	7426	90%	102	27.7%	4.55e+00
			10	11291	22568	2.7	55%	36756	10123	66%	106	20.5%	1.27e+01
			100	106239	189167	3.4	45%	37039	10706	63%	106	19.3%	5.66e+01

Table 6.4 – Clustering state graph approach results on STOCHASTIC SHORTEST PATH PROBLEM with samples of non truncated and non discretized lognormal distribution. List L maximum size is 5e+04 for label correcting algorithm and 1e+05 for generalized A* algorithm.

6.5.2 Clustering state graph and STOCHASTIC SHORTEST PATH PROBLEM

We now consider the STOCHASTIC SHORTEST PATH PROBLEM with scenario distributions sampled from non-truncated and non discretized lognormal distributions. We can see in Table 5.4 that instances long5 and square50 are small instances that are not well solved by the algorithms without state graphs. Table 6.4 provides results using a clustering state graph approach. Again, for the clustering procedure, we use a simulated annealing metaheuristic to minimize the L^1 distance between scenario vectors as d. The columns of Table 6.4 are identical to those of Table 6.1, the constraint strength α being replaced by the parameter β of the minimized probability functional CVaR_β. Again, a larger state graphs enables to reduce the gap.

Airline operations problems Part II



Introduction to Part II

Airline operations are usually split into four successive planning problems. The SCHEDULE PLANNING PROBLEM chooses the set of flight legs operated during a given period based on demand estimates. Then, the FLEET ASSIGNMENT PROBLEM selects which type of aircraft will operate which flight leg. Finally, the AIRCRAFT ROUTING PROBLEM and the CREW PAIRING PROBLEM select the sequences of flight legs operated respectively by airplanes and crews. As the sequences of flight legs that can be operated by crews depend on the flight legs operated by airplanes, the sequential resolution of AIRCRAFT ROUTING PROBLEM and CREW PAIRING PROBLEM is not optimal. The INTEGRATED PROBLEM considered in this dissertation consists in the simultaneous resolution of these two problems.

Given a set of flight legs $\mathscr{L}$ operated by an airline, the AIRCRAFT ROUTING PROBLEM builds the *sequences of flight legs* or *routes* r assigned to the different airplanes. Routes must respect several constraints to be feasible, the most important being maintenance constraints. These constraints are modeled through the set of feasible routes $\mathscr{R}$. A route r *covers* a leg ℓ if ℓ is in r. Each leg ℓ must be covered by exactly one route. At Air France, the cost of operating one leg ℓ is independent from the route containing ℓ. As a consequence, the AIRCRAFT ROUTING PROBLEM is a feasibility problem, which can be stated by the following system.

$$\begin{aligned} &\sum_{r \ni \ell} x_r = 1, \quad \forall \ell \in \mathscr{L} \\ &x_r \in \{0,1\}, \quad \forall r \in \mathscr{R} \end{aligned} \tag{6.4}$$

where variable x_r is equal to 1 if route r is selected and 0 otherwise.

The CREW PAIRING PROBLEM is the analogue of the AIRCRAFT ROUTING PROBLEM where airplanes are replaced by crews. Given a set of legs $\mathscr{L}$ operated by an airline, the CREW PAIRING PROBLEM aims at building the *sequences of flight legs* or *pairings* p realized by crews. The crew working rules enable to define the set of feasible pairings $\mathscr{P}$ given $\mathscr{L}$. Once again, each flight leg ℓ must be covered by exactly one pairing. Besides, each pairing p has a cost c_p which depends on crew wages and hotels costs. The objective of the CREW PAIRING PROBLEM is thus to find a pairing cover of minimum cost. The CREW PAIRING PROBLEM can be stated as

the following integer program.

$$\begin{array}{ll} \min & \displaystyle\sum_{p\in\mathscr{P}} c_p y_p \\ s.t. & \left| \begin{array}{ll} \displaystyle\sum_{p\ni\ell} y_p = 1, & \forall \ell \in \mathscr{L} \\ y_p \in \{0,1\}, & \forall p \in \mathscr{P} \end{array} \right. \end{array} \tag{6.5}$$

where variable y_p is equal to 1 if pairing p is selected and 0 otherwise. Some additional coupling constraints have been omitted for clarity.

When AIRCRAFT ROUTING PROBLEM and CREW PAIRING PROBLEM are solved sequentially, the set of pairings that can be operated depends on the routes selected in the AIRCRAFT ROUTING PROBLEM solution. Indeed, a pair of flight legs (ℓ_1, ℓ_2) must respect several constraints to be chained in a pairing. A pair (ℓ_1, ℓ_2) is a *feasible connection* if the arrival airport of ℓ_1 is identical to the departure airport of flight leg ℓ_2, and the interval of time Δt between the arrival of ℓ_1 and the departure of ℓ_2 is within a minimum turn time $t^{\min}$ and a maximum turn time $t^{\max}$. Besides, if the interval of time Δt of a feasible connection satisfies $t^{\min} \leq \Delta t < t^{\text{short}}$ where t^{short} is a given constant, then the connection is a *short connection*. A short connection can be used by a crew only if it stays in the same aircraft. Let A^{short} denote the short connection set. A pairing p is feasible only if all its short connections are contained in routes selected in the solution of the AIRCRAFT ROUTING PROBLEM. The following short connection constraint links aircraft routing and crew pairing solutions:

$$\sum_{p\ni\alpha} y_p \leq \sum_{r\ni\alpha} x_r, \qquad \forall \alpha \in A^{\text{short}}. \tag{6.6}$$

The INTEGRATED PROBLEM can now be stated as follows.

$$\begin{array}{ll} \min & \displaystyle\sum_{p\in\mathscr{P}} c_p y_p \\ s.t. & \left| \begin{array}{ll} \displaystyle\sum_{r\ni\ell} x_r = 1, & \forall \ell \in \mathscr{L} \\ \displaystyle\sum_{p\ni\ell} y_p = 1, & \forall \ell \in \mathscr{L} \\ \displaystyle\sum_{p\ni\alpha} y_p \leq \sum_{r\ni\alpha} x_r, & \forall \alpha \in A^{\text{short}} \\ x_r \in \{0,1\}, & \forall r \in \mathscr{R} \\ y_p \in \{0,1\}, & \forall p \in \mathscr{P} \end{array} \right. \end{array} \tag{6.7}$$

The purpose of Part II is to introduce a solution scheme for the INTEGRATED PROBLEM. Part II is organized as follows.

- Chapter 7 introduces a solution scheme for the INTEGRATED PROBLEM. This solution scheme uses as black-boxes solution methods for the AIRCRAFT ROUTING PROBLEM and the CREW PAIRING PROBLEM, which are both introduced in the next chapters.
- Chapter 8 introduces a compact integer program to solve the AIRCRAFT ROUTING PROBLEM. Numerical experiments show that this approach, which is simpler than the tradi-

tional column generation approach to the AIRCRAFT ROUTING PROBLEM, is sufficiently efficient to be used in the INTEGRATED PROBLEM solution scheme of Chapter 7 on Air France industrial instances.

- Chapter 9 details a column generation approach to the CREW PAIRING PROBLEM. This approach relies on the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM algorithms of Chapter 4. Numerical experiments prove the efficiency of the approach on Air France instances.
- Chapter 10 tests experimentally the solution method for the INTEGRATED PROBLEM of Chapter 7 when the AIRCRAFT ROUTING PROBLEM solution scheme of Chapter 8 and the CREW PAIRING PROBLEM solution scheme of Chapter 9 are used.
- Chapter 11 focuses on the propagation of delay in airline operations, and extends the methods of the previous chapter to take into account probabilistic constraints on delay propagation.

Literature reviews on the INTEGRATED PROBLEM, the AIRCRAFT ROUTING PROBLEM, and the CREW PAIRING PROBLEM are respectively available in Chapters 7, 8, and 9. Chapter 11 gives a literature review on solution methods for these problems that take into account delay propagation.

7 Integrated aircraft routing and crew pairing problem statement

This chapter introduces a cutting plane approach to the INTEGRATED PROBLEM (6.7). This solution scheme uses solution schemes for the AIRCRAFT ROUTING PROBLEM and the CREW PAIRING PROBLEM as block-boxes. These schemes are introduced respectively in Chapters 7 and 8. The efficiency of the approach is then tested numerically in Chapter 10.

Chapter 7 is organized as follows.

- Section 7.1 introduces the cutting plane approach to the INTEGRATED PROBLEM. This solution scheme is tested numerically in Chapter 10.
- Section 7.2 details the instances that are used to test the algorithms of the next chapters.
- Section 7.3 gives a literature review on the solution approaches to the INTEGRATED PROBLEM.

In this dissertation, we use the bold notations for vectors. Solutions (x_r) and y_p of the AIRCRAFT ROUTING PROBLEM and the CREW PAIRING PROBLEM are therefore respectively denoted $\mathbf{x}$ and $\mathbf{y}$.

7.1 Solution scheme

As the AIRCRAFT ROUTING PROBLEM is a feasibility problem and not an optimization problem, the INTEGRATED PROBLEM can be seen as an extended CREW PAIRING PROBLEM with additional constraints modeling the AIRCRAFT ROUTING PROBLEM feasibility. Indeed, let $\mathbf{y}^\dagger$ be a solution of the CREW PAIRING PROBLEM (6.5): there is a solution to the INTEGRATED PROBLEM with $\mathbf{y}^\dagger$ as crew pairing solution only if the following problem modeling aircraft routing admits a solution.

$$\begin{aligned} &\sum_{r \ni \ell} x_r = 1, && \forall \ell \in \mathscr{L} \\ &x_r \in \{0,1\}, && \forall r \in \mathscr{R} \\ &\sum_{p \ni \alpha} y_p^\dagger \leq \sum_{r \ni \alpha} x_r, && \forall \alpha \in A^{\text{short}} \end{aligned} \tag{7.1}$$

If Problem (7.1) admits a solution $\mathbf{x}^\dagger$, then $(\mathbf{x}^\dagger, \mathbf{y}^\dagger)$ *is an optimal solution of the* INTEGRATED PROBLEM. Indeed, if $(\mathbf{x}, \mathbf{y})$ is a solution of Problem (6.7), then $(\mathbf{y})$ is a solution of Problem

(6.5) with identical cost, and $\sum_{p\in\mathscr{P}} c_p y_p \geq \sum_{p\in\mathscr{P}} c_p y_p^\dagger$ by definition of $\mathbf{y}^\dagger$. The reason why the INTEGRATED PROBLEM is hard to solve is that the AIRCRAFT ROUTING PROBLEM actually constrains the CREW PAIRING PROBLEM on industrial instances, in the sense that Problem (7.1) is not feasible for most industrial instances [42]. The main idea behind INTEGRATED PROBLEM solution scheme presented in this section is to add progressively constraints to the CREW PAIRING PROBLEM (6.5) in order to obtain a solution $\mathbf{y}^\dagger$ such that the AIRCRAFT ROUTING PROBLEM (7.1) becomes feasible. The algorithm uses as subroutines solutions schemes for the AIRCRAFT ROUTING PROBLEM and the CREW PAIRING PROBLEM that are respectively introduced in Sections 8.2, 9.1 and 9.2.

7.1.1 Feedback AIRCRAFT ROUTING PROBLEM loop

In order to tackle with infeasibility in the AIRCRAFT ROUTING PROBLEM when the CREW PAIRING PROBLEM is solved first, we use the solution $\mathbf{y}^\dagger$ that leads to infeasibility in the AIRCRAFT ROUTING PROBLEM (7.1) to constrain CREW PAIRING PROBLEM. Therefore, we introduce *short connections* constraints. A short connections constraint is defined by a set of incompatible short connections S and forbids the simultaneous selection of all the short connections in S.

We can now state the *constrained* CREW PAIRING PROBLEM.

$$\min \sum_{p\in\mathscr{P}} c_p y_p \tag{7.2a}$$

$$s.t. \sum_{p\ni\ell} y_p = 1, \qquad \forall \ell \in \mathscr{L} \tag{7.2b}$$

$$\sum_{p\in\mathscr{P}} |p\cap S| y_p \leq |S|-1, \qquad \forall S \in \mathscr{I}^{SC} \tag{7.2c}$$

$$y_p \in \{0,1\}, \qquad \forall p \in \mathscr{P} \tag{7.2d}$$

The INTEGRATED PROBLEM algorithm can now be presented. The *set of short connections constraints* is initially empty. The following steps are then executed.

1. Solve the constrained CREW PAIRING PROBLEM (7.2) to obtain $\mathbf{y}$.
2. Solve the constrained AIRCRAFT ROUTING PROBLEM (7.1):
   - if Problem (7.1) is not feasible, then add S to $\mathscr{I}^{SC}$, where S is the set of short connections $\alpha \in A^{\text{short}}$ contained in a pairing p such that $y_p = 1$. Return to Step (1).
   - else store the solution of the Problem (7.1) in $\mathbf{x}$.

Theorem 7.1. *The above algorithm converges after a finite number of steps, and at the end of the algorithm* $(\mathbf{x},\mathbf{y})$ *is an optimal solution of the* INTEGRATED PROBLEM (6.7).

Proof. Thanks to short connections set S, a pairing solution $\mathbf{y}$ cannot be generated twice by the algorithm. As there is a finite number of solutions of the crew pairing problem, only a finite

number of short connections constraints set can be generated and the algorithm converges after a finite number of iterations.

Let $(\mathbf{x}, \mathbf{y})$ be a feasible solution of the INTEGRATED PROBLEM. We prove iteratively on the iterations of the algorithm that $\mathbf{y}$ is a solution of Problem (7.2) at all steps of the algorithm. Indeed, suppose it is not the case and let S be the first constraint added by the algorithm among the short connections constraints not-satisfied by $\mathbf{y}$. Note that if $\mathbf{y}$ does not satisfy short connections constraint S, then the solution encoded by $\mathbf{y}$ contains all the short connections in S. As $(\mathbf{x}, \mathbf{y})$ is a feasible solution of the integrated problem, this is in contradiction with the addition of short connections constraint S by the algorithm. As a consequence, any feasible solution of the integrated problem $(\mathbf{x}, \mathbf{y})$ is such that $\mathbf{y}$ is a feasible solution of Problem (7.2) at the end of the algorithm, which implies that the solution returned by the algorithm has a cost smaller to the one of any solution of the INTEGRATED PROBLEM, and thus is an optimal solution of the INTEGRATED PROBLEM. □

Remark 7.1. This algorithm can be interpreted as a Branch-and-Check [167] algorithm where the CREW PAIRING PROBLEM is the basic problem and the AIRCRAFT ROUTING PROBLEM is the delayed problem. It is well known [167] that the performance of a Branch-and-Check algorithm is improved when a linear relaxation of the delayed problem is added to the basic problem in addition to the constraints of Equation (7.2c). We introduce in Chapter 8 a compact integer program for the AIRCRAFT ROUTING PROBLEM which has a tight relaxation. A natural Branch-and-Check implementation in our setting is therefore to add this relaxation to the constraints of Equation (7.2). We do not implement it for industrial reasons: as the maintenance and working rules evolve, a modular solution scheme where the AIRCRAFT ROUTING PROBLEM and the CREW PAIRING PROBLEM solvers are totally separated is easier to maintain. From an industrial point of view, this modularity is one of the main strengths of our solution scheme.

7.1.2 From exact algorithm to matheuristic

The convergence of the algorithm on industrial instances may be long. A heuristic algorithm which converges faster is obtained by strengthening short connections constraints. This can be done by replacing "$\leq |S|-1$" by "$\leq m_S$" with $m_S \ll |S|-1$ in short connection constraint (7.2c). Note that this can lead to infeasibility issues if m_S is to small. One way to overcome this problem and to ensure that the solution returned is at least as good as the one computed by the sequential approach is to compute as a preprocessing a non-constrained solution $\mathbf{x}^*$ of the AIRCRAFT ROUTING PROBLEM, and to choose m_S non smaller than the number of short connection of S in $\mathbf{x}^*$. Besides, convergence can be accelerated by using heuristic CREW PAIRING PROBLEM methods, such as those presented in Sections 9.1 and 9.2.

7.2 Instances

Airlines generally split the planning process in two steps: in a first step, the AIRCRAFT ROUTING PROBLEM and the CREW PAIRING PROBLEM are solved on a one week horizon. Then, the weekly solutions are combined to build solutions on a whole month. In this dissertation, we focus on instances on one week. As the schedule on two successive weeks is very similar, solutions on a

Instance	Legs	Airplane connect.	Airplanes	Crew connect.	Crew pairings (≃)
A318	669	39564	18	3742	130
A319	957	45901	41	3738	240
A320	918	49647	45	3813	280
A321	778	29841	25	3918	165
A318-9	1766	–	(59)	8070	350
A320-fam	3398	–	(129)	21563	690

Table 7.1 – Air France industrial instances

weekly horizon are required to "cycle": a route or a pairing starting at the end of the horizon can end at the beginning. Practically, this means that the pairing is split on two weeks. The cycling horizon technique enables to avoid border conditions at the beginning and at the end of the horizon. As border conditions are easily taken into account in our AIRCRAFT ROUTING PROBLEM solution scheme, we use this technique only for the CREW PAIRING PROBLEM.

The algorithms presented in this dissertation have been tested on a set of industrial instances corresponding to one week of flight legs of the A320 family at Air France. The A320 family is the largest fleet at Air France, and is composed of the A318, A319, A320, and A321 subfleets. Table 7.1 gives the main characteristics of the instances considered: the first column provides the name of the instance, and the second column the number of flight legs in the instance. The two next column provide the number of connections feasible for airplanes, and the number of airplanes in a solution. The two last columns provide the number of connections feasible for crews, and the number of crew pairings in a solution. Instances A318, A319, A320, and A321 correspond to the different subfleets of the A320 family instance. Instance A318-9 is the union of instances A318 and A319. Finally, instance A320-fam is the union of all theses instances and of a smaller subfleet.

As the subfleet of the airplane that operates a flight leg is given in input, solving the AIRCRAFT ROUTING PROBLEM problem on the A320 family instance is done by solving separately the AIRCRAFT ROUTING PROBLEM problem on the A318, the A319, the A320 and the A321 instances. This is not true of the CREW PAIRING PROBLEM, as crews can operate, and actually do operate on airplanes of different subfleets during the same pairing. This is notably the reason why the number of crew connections in the full A320 family is larger than the sum of the number of connections inside subfleets. This is also the reason why the number of pairings in a solution of the A320 family is smaller than the sum of the number of pairings required for the subfleets. We note that the number of airplane connections is much larger than the number of crew connections. This is due to the fact that there can be no night in Paris hub in the middle of a pairing due to crew working rules. On the contrary, an airplane can stay a night in Paris hub. As the number of possible night connections at the hub is large, this explains the difference between the number of airplane and the number of crew connections.

The instances in Table 7.1 are large. For instance, considering the INTEGRATED PROBLEM, the largest instances considered by Mercier and Soumis [124] on the INTEGRATED PROBLEM has

Instance	Legs	Airplane connect.	Airplanes	Crew connect.	Crew pairings (≃)
AR4	152	2107	4	389	60
AR8	313	8723	8	1112	100
AR12	470	19536	12	2055	125
CP50	290	–	–	1006	50
CP70	408	–	–	1705	70
CP90	516	–	–	2490	90

Table 7.2 – Aircraft Routing and Crew Pairing extracted instances

523 legs, and the largest considered by Shao et al. [161] has 676 legs.

To test the performance of the algorithms on instances of different sizes, smaller instances are derived from these large ones. Given the complexity of the working rules, designing instances is not an easy task. Therefore, to build feasible instances, we extract part of the solution on industrial instances. Aircraft routing and crew pairing are relatively different. Therefore, instances extracted from AIRCRAFT ROUTING PROBLEM solutions lead to costly solutions in the CREW PAIRING PROBLEM, and instances extracted from the CREW PAIRING PROBLEM require too many airplanes in the AIRCRAFT ROUTING PROBLEM and are often not feasible for the AIRCRAFT ROUTING PROBLEM. As the link between AIRCRAFT ROUTING PROBLEM and CREW PAIRING PROBLEM in the INTEGRATED PROBLEM lies in the feasibility of the AIRCRAFT ROUTING PROBLEM, we test the INTEGRATED PROBLEM solution scheme only on instances in Table 7.1 and on instances extracted from the AIRCRAFT ROUTING PROBLEM solutions. Table 7.2 provides the characteristics of the extracted instances: its columns are identical to those of Table 7.1. AIRCRAFT ROUTING PROBLEM instances are prefixed by AR, and CREW PAIRING PROBLEM instances are prefixed by CP. The CREW PAIRING PROBLEM instances being mostly not feasible for AIRCRAFT ROUTING PROBLEM, we do not provide the AIRCRAFT ROUTING PROBLEM statistics for these instances.

7.3 Bibliographical remarks

As the AIRCRAFT ROUTING PROBLEM is a feasibility problem, the natural idea to improve the joint cost of the AIRCRAFT ROUTING PROBLEM and the CREW PAIRING PROBLEM is to solve first the CREW PAIRING PROBLEM and the AIRCRAFT ROUTING PROBLEM [105]. Nonetheless, this reverse sequential leads to non-feasibility issues in the AIRCRAFT ROUTING PROBLEM [43], which opened the question of integrated optimization.

Two lines of models have been developed for the integrated models. In the first line, AIRCRAFT ROUTING PROBLEM solutions are generated first and included in an extended CREW PAIRING PROBLEM. Cohn and Barnhart [42] show that only routing solutions including unique and maximal short connection sets needed to be generated to obtain optimal solutions. A two-step solution pattern can therefore be proposed: first, the aircraft routing solutions including maximal short-connection sets are generated, and then the extended crew pairing is solved

using matheuristics[1] relying on branch and price. This method enables to produce quickly solutions with an optimality gap of 2% on instances containing about 125 flight legs. The time consuming phase of the algorithm is the generation of unique and maximal connection sets, and the time necessary to its resolutions increases quickly with the size of the instance. Thus, if it is competitive on small and medium instances, it quickly becomes intractable on large instances.

The other line of models uses a double column generation technique: column are generated for airplanes and for crews. The problem is solved faster if split into two problems thanks to Benders decomposition [43]. Crew pairing is "hidden" in a subproblem whose result is taken into account in the aircraft routing master problem. Besides, on large instances, an exact resolution leads to too large computation time. A three phase matheuristic relying on Benders decomposition has therefore been developed [43]: integrity constraints initially relaxed are progressively re-established to produce a good quality feasible solution. The efficiency of the algorithm is improved by a factor 10 by exchanging the master problem and the subproblem in the Benders decomposition [125]. The model is then generalized to integrate fleet assignment and restricted connect concept [124]. Besides, aggregating some of the short connections linking constraints enables to reduce by a factor 12 the time needed to reach equivalent quality solution [124]. Finally, this Benders decomposition approach has been adapted to integrate FLEET ASSIGNMENT PROBLEM [140].

Extended crew pairing solution method is faster on small size instances, but it is less versatile, and slower on larger instances. The Benders decomposition matheuristic enables to reach solution with an optimality gap smaller than one percent in less than 12 hours on large instances [124]. Nonetheless, the double column generation makes it hard to solve.

The contribution of our cutting planes approach is thus three-fold. First, it enables to solve efficiently large instances. Second, the modularity of the approach makes it very versatile from an industrial point of view, as it enable to separate the resolution of the AIRCRAFT ROUTING PROBLEM and the CREW PAIRING PROBLEM in two distinct black-boxes. Third, all the algorithms used for the INTEGRATED PROBLEM and in the CREW PAIRING PROBLEM can be turned into efficient heuristics if exact algorithms are too slow on large industrial instances.

[1] A matheuristic is a heuristic based on mathematical programming

8 Compact integer program for aircraft routing

This chapter studies the complexity and details a solution approach to the AIRCRAFT ROUTING PROBLEM (6.4). The solution approach applies to the constrained version (7.1) used by the INTEGRATED PROBLEM solution scheme of Chapter 7. Given a set of flight legs $\mathscr{L}$ operated by a fleet of identical airplanes of an airline, the AIRCRAFT ROUTING PROBLEM (6.4) builds the *sequences of flight legs* or *routes* r assigned to the different airplanes. Routes must respect several constraints to be feasible. These constraints are modeled through the set of feasible routes $\mathscr{R}$. A route r *covers* a leg ℓ if ℓ is in r. Each leg ℓ must be covered by exactly one route. We can now restate the AIRCRAFT ROUTING PROBLEM (6.4).

$$\begin{aligned} &\sum_{r \ni \ell} x_r = 1, \quad \forall \ell \in \mathscr{L} \\ &x_r \in \{0,1\}, \quad \forall r \in \mathscr{R} \end{aligned} \tag{6.4}$$

where variable x_r is equal to 1 if route r is selected and 0 otherwise. The constrained version (6.4) has the following additional constraint,

$$\sum_{p \ni \alpha} y_p^\dagger \leq \sum_{r \ni \alpha} x_r, \quad \forall \alpha \in A^{\text{short}}, \tag{8.1}$$

where $y_p^\dagger$ is a fixed solution of the CREW PAIRING PROBLEM.

In the literature, the AIRCRAFT ROUTING PROBLEM is generally a minimization problem with costs defined on each routes. It is therefore solved by column generation [20]. The subproblem which arises in these approach is a STANDARD RESOURCE CONSTRAINED SHORTEST PATH PROBLEM which can be solved efficiently in practice. In this dissertation, we introduce a compact integer program that encodes the AIRCRAFT ROUTING PROBLEM. It is solved in reasonable time by recent MIP solvers, and we therefore do not have to use a column generation approach. We also use this compact integer program approach to solve the constrained AIRCRAFT ROUTING PROBLEM (7.1) used in the INTEGRATED PROBLEM solution scheme of Chapter 7.

Both the study of the AIRCRAFT ROUTING PROBLEM complexity and the definition of the compact integer program to solve it require a precise definition of the AIRCRAFT ROUTING PROBLEM. We enter into the details of the definition of the set of feasible routes $\mathscr{R}$. To make it easier to treat, the set of legs $\mathscr{L}$ is usually turned into a digraph D, because it then enables to

consider routes as paths in this graph. There are two classical ways to build such a graph. In an *airport-time* graph, a vertex corresponds to an airport-time pair, and an arc to a flight leg. Indeed, a flight leg leads from an airport-time pair to another. In a *connection graph*, flight legs are vertices while arcs are feasible connections between legs. Connection graphs are larger than airport-time graphs, but their modeling power is stronger as they consider connections individually, which enables to model connection specific properties. As short connections make the link between the AIRCRAFT ROUTING PROBLEM and the CREW PAIRING PROBLEM, it is very important to be able to consider them individually. We therefore use connection graphs in this dissertation. Airport-time graphs suit better to the study of AIRCRAFT ROUTING PROBLEM complexity. To avoid redundant definitions, in this chapter, we make this study in the context of connection graphs. The elements that cannot be proved in the context of connection graphs are postponed to Appendix B.

Chapter 8 is organized as follows:

- Section 8.1 gives a practical definition of the AIRCRAFT ROUTING PROBLEM based on connection graphs.
- Section 8.2 focuses on the resolution of AIRCRAFT ROUTING PROBLEM. It defines the notion of maintenance state graph and introduces the compact integer program for the AIRCRAFT ROUTING PROBLEM problem. Numerical results on Air France instances show the practical efficiency of the approach.
- Section 8.3 provides a detailed study of the complexity of the AIRCRAFT ROUTING PROBLEM. AIRCRAFT ROUTING PROBLEM is proved to be $\mathcal{NP}$-complete in the general case and polynomial when the number of airplanes is fixed.
- Section 8.4 gives a literature review on the solution method of the AIRCRAFT ROUTING PROBLEM.

8.1 Aircraft routing problem definition

The AIRCRAFT ROUTING PROBLEM problem takes in input the set of legs $\mathscr{L}$ operated by an airline. Several notions being analogous between the AIRCRAFT ROUTING PROBLEM and the CREW PAIRING PROBLEM, we use the subscript $\mathscr{R}$ to identify the routing notions. A pair of flight legs (ℓ_1, ℓ_2) must respect several constraints to be chained in a route r. A pair (ℓ_1, ℓ_2) is a feasible routing connection if the arrival airport of ℓ_1 is identical to the departure airport of flight leg ℓ_2, and the interval of time ΔT between the arrival of ℓ_1 and the departure of ℓ_2 is within a minimum turn time $t^{\min,\mathscr{R}}$ and a maximum turn time $t^{\max,\mathscr{R}}$.

An easy way to model the AIRCRAFT ROUTING PROBLEM is to use a *routing connection digraph* $D^{\mathscr{R}} = (V^{\mathscr{R}}, A^{\mathscr{R}})$, where $V^{\mathscr{R}}$ is the set of vertices and $A^{\mathscr{R}}$ is the set of arcs. In a connection graph, there is one vertex v_ℓ for each flight leg $\ell \in \mathscr{L}$, and one arc $a \in A^{\mathscr{R}}$ for each feasible connection. Note that due to the role of time in the definition of arcs, a routing connection graph is necessarily acyclic. Additional source and sink vertices are added to model the desired location airport of airplanes at the beginning and at the end of the horizon.

A *route* r is a source to sink path in the connection graph. Not all source to sink paths of the routing connection graph $D^{\mathscr{R}}$ correspond to feasible routes $r \in \mathscr{R}$. To be feasible, an aircraft

route must satisfy an additional property. Indeed, maintenance operations must be regularly performed on airplanes. As the equipment required to perform the maintenance operations is expensive, maintenances can only be performed in a few airports called *maintenance bases*. A route r satisfies the *maintenance constraint* if the number of days between two visits of an aircraft in a maintenance base is non-greater than a given upper-bound δ_{maint}. An arc $a = (v_{\ell_1}, v_{\ell_2})$ is a *night arc* if the departure of leg ℓ_1 is not on the same day as the arrival of leg ℓ_2, and it is a *maintenance night arc* if the arrival airport of ℓ_1 is a maintenance base, and the interval of time between the arrival of ℓ_1 and the departure of ℓ_2 is non-smaller than a minimum maintenance time $t^{\text{maint},\mathscr{R}}$. Let $N \subseteq A^{\mathscr{R}}$ be the set of night arcs, and $M \subseteq N$ be the set of maintenance night arcs. A route r is *feasible* if any subpath with δ_{maint} night arcs has at least one maintenance night arc.

Let k be the number of airplanes in the fleet. Each aircraft of the fleet follows one feasible route. A *routing* is a partition of $D^{\mathscr{R}}$ into at most k vertex disjoint feasible routes.

AIRCRAFT ROUTING PROBLEM
Input. A graph $D^{\mathscr{R}} = (V^{\mathscr{R}}, A^{\mathscr{R}})$, a set of night arcs $N \subseteq A^{\mathscr{R}}$, a set of maintenance night arcs $M \subseteq N$, and a fleet size $k \in \mathbb{Z}_+$.
Output. A routing

When a solution $\mathbf{y}$ of the CREW PAIRING PROBLEM is fixed and constrains the AIRCRAFT ROUTING PROBLEM, the short connection constraint (6.6) can be dealt with in this framework. Indeed, it suffices to remove from $A^{\mathscr{R}}$ any connection (ℓ_1, ℓ_2) such that there is a connection in A^{short} different from (ℓ_1, ℓ_2) but having ℓ_1 as origin or ℓ_2 as the destination. We therefore omit them in the remaining of the chapter.

To enhance readability, we omitted the *border constraints*, which give the number of days since the last visit in the maintenance base of the airplanes at the beginning and at the end of the horizon.

Note that an AIRCRAFT ROUTING PROBLEM instance is more naturally defined as a list of flight legs that have to be covered by airplanes. This natural definition is introduced in Appendix B. Figure 8.1 illustrates the link between a list of flight legs or schedule and a connection graph, and the fact that a routing is a path partition. The solution scheme detailed in this chapter enables to solve both problems. But working with the list of flight legs may have an impact on the theoretical complexity of the problem. This is the reason why the complexity results that are proved in this chapter in the connection graph setting are done in Appendix B in the list of flight legs setting.

8.2 Maintenance state graph and compact integer program

Solutions of the AIRCRAFT ROUTING PROBLEM are partitions of the AIRCRAFT ROUTING PROBLEM in vertex-disjoint source into sink paths. Paths partition are easily encoded in an integer program using flow and cover constraints. In this section, we extend this integer program to the AIRCRAFT ROUTING PROBLEM. This extension relies on the notion of maintenance state graph, which enables to take into account the maintenance rule.

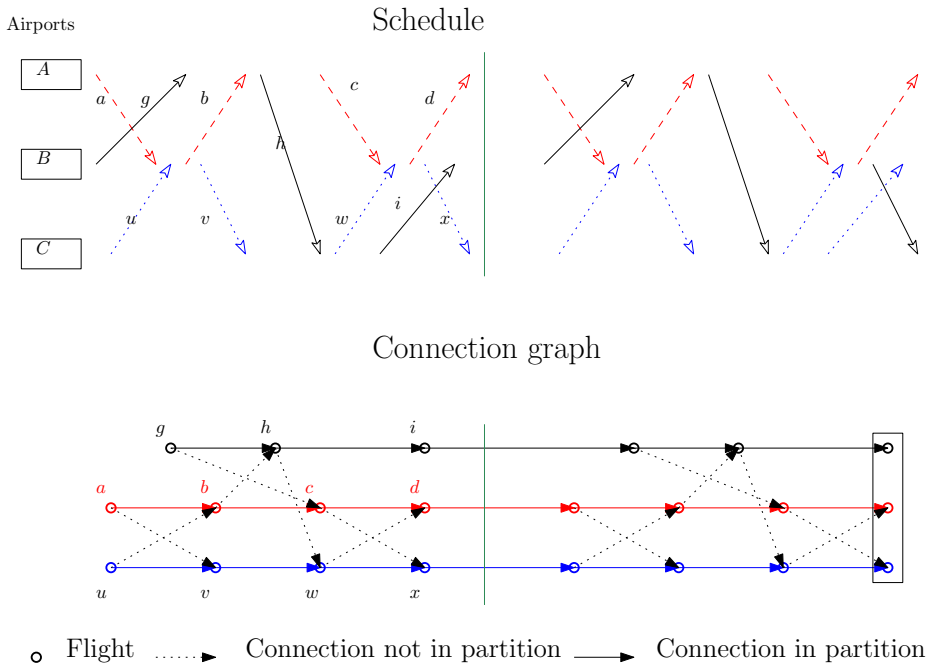


Figure 8.1 – AIRCRAFT ROUTING PROBLEM solution as a path partition in the routing connection graph

8.2.1 Maintenance state graph and integer program

In order to handle the maintenance rule, we want to count the number of days since the last visit in a maintenance base in our linear program. We therefore build a *maintenance state graph* $\mathscr{D} = (\mathscr{V}, \mathscr{A})$, where $\mathscr{V}$ is the set of *state vertices* ϑ and $\mathscr{A}$ is the set of *state arcs*. State vertices are built as follows: for each connection graph vertex v, we build δ_{maint} state vertices $\vartheta_v^1, \ldots, \vartheta_v^{\delta_{\text{maint}}} \in \mathscr{V}_v$. Like in dynamic programming, index i on state vertex ϑ_v^i carries information: it means that the last maintenance of a plane which crosses state vertex ϑ_v^i happened i nights ago.

We therefore build the state arcs in order to satisfy the following property: *the last maintenance of an airplane going through ϑ_v^k happened k days ago.* Let $a = (v_1, v_2)$ be an arc in the connection graph. We build its state arc set in order to count the number of days since the last maintenance check. If v_1 and v_2 are on the same day: the number of days since the last maintenance does not change when crossing a. Thus for each day $d \in \{1, \ldots, \delta_{\text{maint}}\}$, we add a state arc $\alpha = (\vartheta_{v_1}^d, \vartheta_{v_2}^d)$. If v_1 and v_2 are not on the same day, then it corresponds to a night. If this night is a maintenance night, then the number of days since the last maintenance night after crossing a is 1. Therefore, for each day $d \in \{1, \ldots, \delta_{\text{maint}}\}$, we add a state arc $\alpha = (\vartheta_{v_1}^d, \vartheta_{v_2}^1)$. Finally, if it is not a maintenance night, the number of days since the last maintenance increases by one: for each day $d \in \{1, \ldots, \delta_{\text{maint}} - 1\}$, we add an arc $\alpha = (\vartheta_{v_1}^d, \vartheta_{v_2}^{d+1})$. A plane arriving in $\vartheta_{v_1}^{\delta_{\text{maint}}}$ needs to perform a maintenance on the next night: it cannot take an arc a which is not a maintenance night.

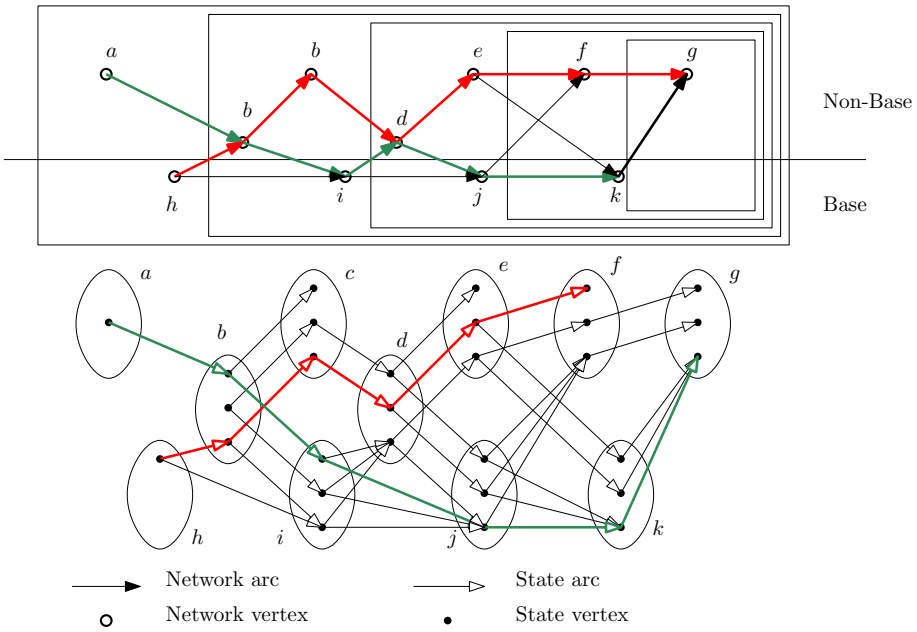


Figure 8.2 – Feasible and infeasible paths in a state graph.

Let $\mathcal{V}_v$ denote the set of state vertices of connection graph vertex v, and $\mathscr{A}_a$ denote the set of state arcs of connection graph arc a. Let $\mathcal{V} = \bigcup_{v \in V} \mathcal{V}_v$ be the set of state vertices and $\mathscr{A} = \bigcup_{a \in A} \mathscr{A}_a$ be the set of state arcs. We have thus defined the state graph $\mathscr{D} = (\mathcal{V}, \mathscr{A})$.

We recall that a feasible route is a source to sink path in the connection graph that satisfies the maintenance constraint. Let P be a path in the connection graph. If P is feasible, like the green path on Figure 8.2, it induces a unique path $\pi(P)$ in the state graph. On the contrary, if P is not feasible, it does not induce a feasible path in the state graph, because the first arc a that does not satisfy the maintenance constraint does not exist in the state graph, as illustrated by the red path on Figure 8.2.

Claim 8.1. *There is a bijection between the feasible paths in the connection graph and the paths in the state graph.*

We can now state the compact integer program. Let x_α be a binary variable equal to one if and only if state arc α is in the partition. A solution of the following equations is a feasible routing.

$$\sum_{\substack{\vartheta \in \mathcal{V}_v \\ \alpha \in \delta^-(\vartheta)}} x_\alpha = 1 \qquad \forall v \in V \setminus S \tag{8.2a}$$

$$\sum_{\substack{\vartheta \in \mathcal{V}_v \\ \alpha \in \delta^+(\vartheta)}} x_\alpha = 1 \qquad \forall v \in S \tag{8.2b}$$

$$\sum_{\alpha \in \delta^-(\vartheta)} x_\alpha = \sum_{\alpha \in \delta^+(\vartheta)} x_\alpha \qquad \forall \vartheta \in \mathcal{V} \tag{8.2c}$$

$$x_\alpha \in \{0,1\} \qquad \forall \alpha \in \mathscr{A} \tag{8.2d}$$

This linear program is in an alternative to the usual column generation formulation (6.4) to model Air France AIRCRAFT ROUTING PROBLEM. Combined with the cover equations (8.2a) and (8.2b), the flow equation (8.2c) ensures that the solution is a path partition of the connection graph in source to sink paths of the state graph, and thus a feasible solution to the AIRCRAFT ROUTING PROBLEM.

State sources and state sinks enable to enforce border conditions, i.e. the number of days since the last maintenance of each airplane at the beginning of the horizon, and upper bounds on that number at the end of the horizon. The conditions at the end of the horizon can be achieved by adding extra state arcs to the sinks state vertices.

Remark 8.1. Note that the maintenance state graphs introduced in this section are similar but not equivalent to the state graphs defined in Chapter 6. Indeed, the property that for each path P in $D^{\mathscr{R}}$ there is a path π in $\mathscr{D}$ such that $\theta(\pi) = P$ is not satisfied in a maintenance state graph.

Short connection constraints

When used in the context of the INTEGRATED PROBLEM solution scheme of Chapter 7, an additional constraint must be taken into account. Indeed, each short connection in a pairing solution must be in a routing solution. Let A^{short} be the set of short connections that must be in the AIRCRAFT ROUTING PROBLEM solution. We therefore add the constraints

$$\sum_{\alpha \in \mathscr{A}} x_\alpha = 1, \quad \forall \alpha \in A^{\text{short}},$$

to (8.2). With these new constraints, a short connection $a \in A^{\text{short}}$ belongs to any feasible solution of (8.2).

8.2.2 Costs

Even if the AIRCRAFT ROUTING PROBLEM is considered as a feasibility problem in this dissertation, the solution of AIRCRAFT ROUTING PROBLEM has an impact on the ground operations. In this section, we briefly explain how to take into account such costs. In Chapter 11, we show how to extend this program in order to take into account constraints on delay.

Airplanes are not identified in integer program (8.2). As a consequence, only costs that are independent of the airplane can be handled in this program. Ground operations costs are aircraft-independent and can therefore be handled, whereas fuel consumption costs are aircraft-dependent and cannot be handled.

Let c_a denote the ground cost of a connection a between two flight legs ℓ_1 and ℓ_2. It is the sum of several terms. The first term is the terminal cost c_a^{terminal}. It represents the cost of moving the aircraft from the arrival terminal of ℓ_1 to the departure terminal of ℓ_2, and is equal to zero if these terminals are identical. A second cost c_a^{Schengen} is incurred when a non-Schengen flight leg is connected with a Schengen flight leg. This cost is a function of the time at which it is performed. A third source of cost c_a^{duration} is linked to the turn time of the connection. Indeed, if this turn time is long, the airplane must be moved to a parking, which increases the cost of

Instance	Legs	Airplanes	Airplane connect.
AR4	152	2107	4
AR8	313	8723	8
AR12	470	19536	12
A318	669	39564	18
A319	957	45901	41
A320	918	49647	45
A321	778	29841	25

Table 8.1 – Aircraft Routing instances

the connection.

As all these costs only depend on the connection, they can easily be added into program (8.2) using $c_\alpha = c_a = c_a^{\text{terminal}} + c_a^{\text{Schengen}} + c_a^{\text{duration}}$ where a is the unique connection arc in $A^{\mathscr{R}}$ corresponding to state arc α. This leads to the following linear program.

$$\min \sum_{\alpha \in \mathscr{A}} c_\alpha x_\alpha \tag{8.3a}$$

$$\sum_{\substack{\vartheta \in \mathscr{V}_v \\ \alpha \in \delta^-(\vartheta)}} x_\alpha = 1 \qquad \forall v \in V^{\mathscr{R}} \setminus S \tag{8.3b}$$

$$\sum_{\substack{\vartheta \in \mathscr{V}_v \\ \alpha \in \delta^+(\vartheta)}} x_\alpha = 1 \qquad \forall v \in S \tag{8.3c}$$

$$\sum_{\alpha \in \delta^-(\vartheta)} x_\alpha = \sum_{\alpha \in \delta^+(\vartheta)} x_\alpha \qquad \forall \vartheta \in \mathscr{V} \tag{8.3d}$$

$$x_\alpha \in \{0,1\} \qquad \forall \alpha \in \mathscr{A} \tag{8.3e}$$

Remark 8.2. The strength of this linear program is that it admits the same linear relaxation as the column generation linear program of Equations (6.4). Indeed, any solution of the relaxation of one of these problems can be expressed as a convex combination of solutions of the other one.

8.2.3 Numerical results

In this section, we test the efficiency of the algorithms for the AIRCRAFT ROUTING PROBLEM on the instances in Table 8.1. These instances have been introduced in the context of the INTEGRATED PROBLEM in Chapter 7.

The numerical experiments are performed on a server with 128 Gb of ram and 12 cores at 2.4 GHz. `CPLEX 12.1.0` is used to solve (8.2). In the INTEGRATED PROBLEM scheme of Chapter 7, the AIRCRAFT ROUTING PROBLEM is solved several times with different short connection constraints. The time needed to prove infeasibility on infeasible instances is therefore at least as important as the time needed to find a feasible solution when the AIRCRAFT ROUTING

Instance	Feasibility	Computation time (mm:ss)
AR4	True	00:00.34
	False	00:00.17
AR8	True	00:03.27
	False	00:01.99
AR12	True	00:06.24
	False	00:04.03
A318	True	01:35.20
	False	00:14.03
A319	True	00:18.85
	False	00:21.57
A320	True	13:27.96
	False	00:34.76
A321	True	00:18.75
	False	00:22.95

Table 8.2 – Feasibility and infeasibility of Aircraft Routing problem

Instance	Computation time (mm:ss)
AR4	00:01.01
AR8	00:05.44
AR12	00:21.59
A318	00:58.01
A319	01:04.84
A320	03:54.55
A321	01:02.82

Table 8.3 – Computation time for Aircraft Routing optimization problem

PROBLEM is feasible. Therefore, in Table 8.2, we consider two version of each instances: one that is feasible, and one that has short connections constraints that make it infeasible.

These solution times are much smaller than those required to solve the CREW PAIRING PROBLEM. This proves that our compact integer programming approach to AIRCRAFT ROUTING PROBLEM is relevant in the context of the INTEGRATED PROBLEM.

Finally, in Table 8.3, we test the ability of compact integer program (8.3) to solve the optimization version of AIRCRAFT ROUTING PROBLEM. The objective here is to minimize the number of airplanes used. We observe that the solution times needed are not much higher than the ones needed for feasibility problems.

8.3 Aircraft routing problem complexity

As the content of this section is not used in the context INTEGRATED PROBLEM, we denote the AIRCRAFT ROUTING PROBLEM connection graph $D = (V, A)$ instead of $D^{\mathcal{R}} = (V^{\mathcal{R}}, A^{\mathcal{R}})$ in this section. The AIRCRAFT ROUTING PROBLEM is often referred to be $\mathcal{NP}$-complete, but its mathematical formulation is not fixed in the literature, and we are not aware of a paper stating a precise definition of the problem completed by the proof of its $\mathcal{NP}$-completeness in the general case. Talluri [165] considers a restricted version of the AIRCRAFT ROUTING PROBLEM, gives a polynomial algorithm for $\delta_{\text{maint}} \leq 3$, and proves its $\mathcal{NP}$-completeness for $\delta_{\text{maint}} \geq 4$. In this dissertation, we prove the two following theorems.

Theorem 8.2. *The* AIRCRAFT ROUTING PROBLEM *is $\mathcal{NP}$-complete for $\delta_{\text{maint}} \geq 3$.*

Theorem 8.3. *The* AIRCRAFT ROUTING PROBLEM *is polynomial when the number of airplanes k is fixed. It can be solved in time bounded from above by $2|V|\delta_{\text{maint}}^{k}$, where k is the number of airplanes and n the number of flight legs.*

Both theorems rely on an analogy with vertex-disjoint paths problems, which are special cases of multicommodity flow problems, and are proved to be $\mathcal{NP}$-complete in the general case [69] and polynomial [79] when the number of commodity is fixed.

The input of the AIRCRAFT ROUTING PROBLEM defined in Section 8.1 is a connection graph. As we already noticed, a more natural input for this input would be the list of the legs to be covered or schedule. Appendix B defines the SCHEDULE AIRCRAFT ROUTING PROBLEM, whose input is a schedule, and proves Theorems 8.2 and 8.3 in the context of SCHEDULE AIRCRAFT ROUTING PROBLEM. As the general case approach does not bring additional ideas but requires the introduction of many technical details, we give in this section a proof of Theorems 8.2 and 8.3 based on the connection-graph version of the AIRCRAFT ROUTING PROBLEM introduced in Section 8.1.

8.3.1 $\mathcal{NP}$-completeness

We now prove that AIRCRAFT ROUTING PROBLEM as it has been stated in 8.1 is $\mathcal{NP}$-complete. Nonetheless, if we take a generic graph D as a connection-graph, it is not always possible to construct a schedule, that is to say a set of flight legs between airports at given time, with D as connection graph. Indeed, as we will see in Appendix B, a schedule can easily be reconstructed from an airport-time graph. But as the connection graph shares most of its structure with the line-graph of the airport-time graph, an airport-time graph and thus a schedule cannot be easily reconstructed from the connection graph.

The TWO-COMMODITIES ARC-DISJOINT PATHS PROBLEM ON ACYCLIC DIGRAPH is $\mathcal{NP}$-complete [69].

TWO-COMMODITIES ARC-DISJOINT PATHS PROBLEM ON ACYCLIC DIGRAPH
Input. An acyclic digraph $D = (V, A)$, vertices s_1, s_2, t_1, and t_2, two non negative integers R_1 and R_2.

Output. R_i arc-disjoint paths from s_i to t_i for $i = 1,2$.

A *source* in a digraph is a vertex with no incoming arc, and a *sink* is a vertex with no outgoing arc. Adding R_i new sources (resp. sinks) vertices u_i (resp. v_i), and R_i arcs (u_i, s_i) (resp. (t_i, v_i)), and going to the line graph, we reduce the TWO-COMMODITIES ARC-DISJOINT PATHS PROBLEM ON ACYCLIC DIGRAPH to the following $\mathcal{NP}$-complete problem.

TWO-COMMODITIES VERTEX-DISJOINT PATHS PROBLEM ON ACYCLIC DIGRAPH
Input. A directed acyclic graph $D = (V, A)$, sets of sources S_1 and S_2, sets of sinks T_1 and T_2 such that $|S_i| = |T_i| = R_i$.
Output. R_i vertex-disjoint paths from S_i to T_i.

Proof of Theorem 8.2. The proof is a reduction of the TWO-COMMODITIES VERTEX-DISJOINT PATHS PROBLEM ON ACYCLIC DIGRAPH. Without loss of generality, we suppose that $\delta_{\text{maint}} = 3$. Larger δ_{maint} can be dealt with analogously by adding $(\delta_{\text{maint}} - 3)$ non-maintenance night arcs before each source of the instance constructed.

Let D, S, T be an instance of the TWO-COMMODITIES VERTEX-DISJOINT PATHS PROBLEM ON ACYCLIC DIGRAPH. We start by building from D a digraph D' as follows. Let I be the set of vertices $V \setminus (S_1 \cup S_2 \cup T_1 \cup T_2)$. For each vertex $v \in I$, we add two vertices s_v and t_v and three arcs (s_v, v), (v, t_v), and (s_v, t_v) to obtain $D' = (V', A')$. The set S' of sources of D' is equal to $S_1 \cup S_2 \cup \bigcup_{v \in I} \{s_v\}$, and the set T' of sinks is equal to $T_1 \cup T_2 \cup \bigcup_{v \in I} \{t_v\}$.

We now show that the existence of $R_1 + R_2$ vertex-disjoint paths from $S_1 \cup S_2$ to $T_1 \cup T_2$ in D implies the existence of a partition of the vertices of D' into source to sink paths. Indeed, suppose that $\mathscr{P}$ contains $R_1 + R_2$ vertex-disjoint paths from $S_1 \cup S_2$ to $T_1 \cup T_2$ in D. For each vertex $v \in I$, we define path P_v to be equal to (s_v, t_v) if v belongs to some path $P \in \mathscr{P}$, and to (s_v, v, t_v) otherwise. Then $\mathscr{P} \cup [\bigcup_{v \in I} \{P_v\}]$ is a partition of the V' into source to sink paths. This construction is illustrated on Figure 8.3.

We now build a connection graph $D'' = (V'', A'')$ on D', a night set $N \subseteq A''$, and a maintenance night set $M \subseteq N$ as follows. For each vertex $s \in S'$, we build two vertices u_s, v_s and the two arcs (u_s, v_s) and (v_s, s), and we add these arcs to N. If $s \notin S_1 \cup S_2$, then we add both (u_s, v_s) and (v_s, s) to M. If $s \in S_1$, then we add only (u_s, v_s) to M, and if $s \in S_2$, we do not add (u_s, v_s) and (v_s, s) to M. Symmetrically, for each vertex $t \in T'$, we build two vertices u_t, v_t and the arcs (t, u_t) and (u_t, v_t) and we add these arcs to N. If $t \in\notin T_1 \cup T_2$, we do not add (t, u_t) and (u_t, v_t) to M. If $t \in T_1$, then we add (u_t, v_t) to M, and if $t \in T_2$, we add both (t, u_t) and (u_t, v_t) to M. The triple (D, N, M) defines an instance of the AIRCRAFT ROUTING PROBLEM with no boundary conditions. This construction is also illustrated on Figure 8.3.

We now prove that the existence of a solution to the TWO-COMMODITIES VERTEX-DISJOINT PATHS PROBLEM ON ACYCLIC DIGRAPH instance (D, S_i, T_i) is equivalent to the existence of a solution to the AIRCRAFT ROUTING PROBLEM problem instance (D'', N, M). The implication directly follows from the fact that any solution to the initial problem can be extended to a path partition of the vertices of D', which is naturally extended to a path partition of D'' by adding to any u_s, v_s, u_t, and v_t to any s-t path. Besides, by disjunction of case on the S_1-T_1, the S_2-T_2, and the remaining paths, we can check that each path satisfies the maintenance constraint,

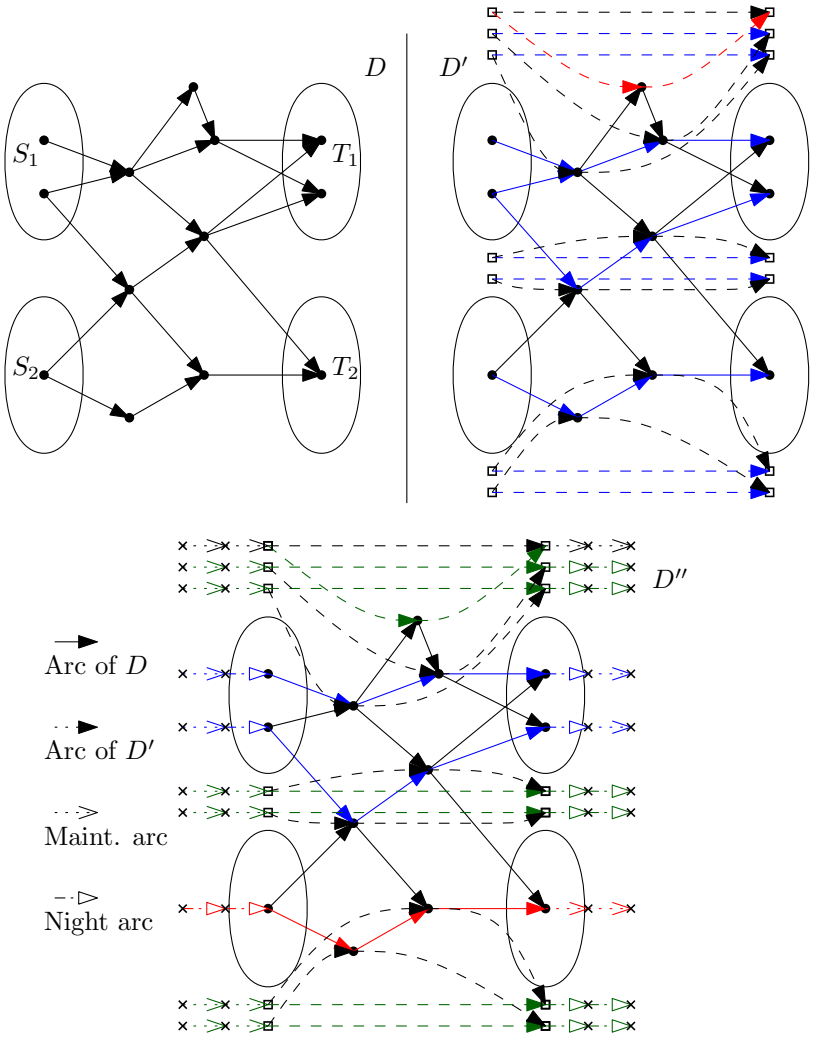


Figure 8.3 – Reduction of the disjoint path problem instance D to the aircraft routing instance D''. Arcs in color on D' show a vertex disjoint path partition. Arcs in red show how the new arcs enable to obtain a path partition from a solution of the initial problem. Arcs in color on D'' indicate the solution of both problems.

and thus is a feasible route, which gives the result.

Conversely, let $\mathscr{P}$ be a partition of D'' into feasible routes. If there is a vertex of S_2 on a path $P \in \mathscr{P}$, then there is necessarily a vertex of T_2 on this path, as otherwise there would be $3 = \delta_{\text{maint}}$ consecutive night arcs of P that are not maintenance night arcs, which contradicts the hypothesis of P being a feasible route. The same reasoning and the fact that all vertices of T_2 are on paths going through S_2 enable to conclude that any path of $\mathscr{P}$ going through a vertex of S_1 goes through a vertex of T_1. As a consequence, the restriction of $\mathscr{P}$ to paths in D gives a solution to the TWO-COMMODITIES VERTEX-DISJOINT PATHS PROBLEM ON ACYCLIC DIGRAPH, which gives the equivalence and the theorem. □

Remark 8.3. The definition of the connection graph from an airline schedule induces some structure on the graph. The most important point is that the night set N sets is the union of several N_i corresponding to each night of the schedule. If S denotes the set of sources and T the set of sinks of the connection graph, each night is an S-T cut. Note that all the nights defined the the proof of Theorem 8.2 are S-T cuts. Additional details are given in Appendix B.

Remark 8.4. The reduction of the TWO-COMMODITIES ARC-DISJOINT PATHS PROBLEM ON ACYCLIC DIGRAPH to the TWO-COMMODITIES VERTEX-DISJOINT PATHS PROBLEM ON ACYCLIC DIGRAPH is done by going into the line-graph. The proof of $\mathcal{NP}$-completeness in the airport-time graph given in Appendix B is analogue of the proof given in this section, but the reduction is directly from the TWO-COMMODITIES ARC-DISJOINT PATHS PROBLEM ON ACYCLIC DIGRAPH, which makes sense as the connection graph shares most of its structure with the line graph of the airport-time graph.

8.3.2 Fixed parameter tractability

We now focus on the proof of Theorem 8.2. As polynomial algorithms for the AIRCRAFT ROUTING PROBLEM only have a theoretical interest, we give in this section a simple polynomial algorithm in $O\left(|V|^{k+1}\delta_{\text{maint}}^k\right)$. The algorithm giving the bound of Theorem 8.3 has the same flavor, but as it requires technical developments, we postpone it to Appendix B. Both algorithms are inspired by pebbling game algorithms for integer multi-commodity flows problems [79]. We underline the fact that, in these algorithms, the k airplaines are not identifiable.

The idea of the algorithm is to move planes on the maintenance state graph $\mathscr{D} = (\mathscr{V}, \mathscr{A})$. Let V_{dist} be the set of distributions δ of k airplanes on the state vertices $\vartheta \in \mathscr{V}$ such that, for each vertex of the initial graph v, there is at most one airplane on the state vertices in $\mathscr{V}_v$.

We now consider these distributions as the vertices of a digraph $D_{\text{dist}} = (V_{\text{dist}}, A_{\text{dist}})$, where A_{dist} is built as follows. Let $\preceq$ be a topological ordering of the vertices of D, let $i(v)$ be the index of v for $\preceq$, let v_i be the vertex with index i, and let $i(\delta)$ be the largest index of the vertices v such that there is a plane on a state vertex of $\mathscr{V}_v$ in δ. Let δ_1 and δ_2 be two distributions such that $i(\delta_2) = i(\delta_1) + 1$. Then there is an arc between δ_1 and δ_2 only if δ_2 can be obtained from δ_1 by moving only one plane along an arc of $\mathscr{A}$. Note that this plane necessarily ends on the state vertex of $\mathscr{V}_{v_i}$ in δ_2, where $i = i(\delta_1) + 1$.

Lemma 8.4. *The cardinal of A_{dist} is non greater than $|V|^{k+1}\delta_{\text{maint}}^k$*

Proof. There are $\begin{pmatrix} V \\ k \end{pmatrix}$ ways of choosing the k vertices v such that there is a plane in $\mathscr{V}_v$, and then for each of these vertices, δ_{maint} ways of choosing the state vertices in $\mathscr{V}_v$ on which the plane is. As a consequence, $|V_{\text{dist}}| \leq |V|^k \delta_{\text{maint}}^k$. The result is given by the fact that each state vertex has at most $|V|$ predecessors. □

Lemma 8.5. *There is a bijection between the solutions of the* AIRCRAFT ROUTING PROBLEM *and the paths in D_{dist} from the origin distributions, where all the planes are on sources vertices, to the destination distributions, where all the planes are on sinks.*

Proof. Let $\mathscr{P}$ be the set of maintenance state graph paths $\pi_1, \ldots, \pi_k$ of a solution of the AIRCRAFT ROUTING PROBLEM. As $\mathscr{P}$ induces a partition of V, for each vertex v in V, there is a unique path π in $\mathscr{P}$ that intersects $\mathcal{V}_v$. For each $j \geq k$ in $[|V|]$, let ϑ_j denote the unique state vertex in $\mathcal{V}_{v_j}$ that is contained in a path of $\mathscr{P}$. And for $j > k$, let ϑ'_j be the unique state vertex such that the arc $(\vartheta'_j, \vartheta_j)$ is in a path of $\mathscr{P}$. We define δ_k to be the distribution $\{\vartheta_1, \ldots, \vartheta_k\}$. And for j from $k+1$ to $|V|$, we define δ_j to be $\delta_{j-1} \cup \{\vartheta_j\} \setminus \{\vartheta'_j\}$. Distribution δ_k is an origin distribution and $\delta_{|V|}$ is a destination distribution. Besides, as $i(\delta_j) = j$, by definition of A_{dist}, there is an arc between δ_j and δ_{j+1} for each j in $k, \ldots, |V|-1$. As a consequence, $(\delta_k, \delta_{k+1}, \ldots, \delta_{|V|})$ is an origin to destination path in the distribution graph.

Conversely, let $(\delta_k, \ldots, \delta_{|V|})$ be an origin to destination path in D_{dist}. For each j in $k+1, \ldots, |V|$, we define ϑ_j (resp. ϑ'_j) to be the unique state vertex in $\delta_j \setminus \delta_{j-1}$ (resp. $\delta_{j-1} \setminus \delta_j$). By definition of A_{dist}, $\alpha_j = (\vartheta'_j, \vartheta_j)$ is an arc in $\mathscr{A}$, and there exists a unique index $h < j$ such that $\vartheta_h = \vartheta'_j$. As a consequence, the union of the α_j for j in $k+1, \ldots, |V|$ is a set of maintenance state graph paths that induces a partition of V, and thus defines an AIRCRAFT ROUTING PROBLEM solution. □

Proof of Theorem 8.3 with the weaker upper bound $|V|^{k+1}\delta_{\text{maint}}^k$. Lemma 8.5 ensures that a path algorithm in distribution graph D_{dist} finds an solution to the AIRCRAFT ROUTING PROBLEM if this solution exists. As D_{dist} is acyclic, the running time of a path finding algorithm in D_{dist} is $|A_{\text{dist}}|$, and Lemma 8.4 gives the theorem. □

8.4 Bibliographical remarks

Traditional approaches split the AIRCRAFT ROUTING PROBLEM into two subproblems. The first problem produces routes for one day, and the second problem combines these one day routes into a week long (or month long) routes which satisfy the maintenance requirement. A polynomial algorithm for the second subproblem exists for $\delta_{\text{maint}} = 3$ [88], but for $\delta_{\text{maint}} \geq 4$, the problem is $\mathscr{NP}$-complete [165]. Heuristics [73] and a Lagrangian relaxation [41] have been proposed to solve the global problem.

During the last two decades, AIRCRAFT ROUTING PROBLEM has often been turned into an optimization for two different reasons. The first reason is its integration with the fleet assignment problem. In low frequency point to point networks, aircraft routing in often infeasible given the fleet assignment solution. Researchers have therefore integrated fleet assignment and aircraft routing models [20, 52]. The second reason is delay. Indeed, delay on flight legs is a significant source of costs for airline companies. A statistical treatment shows that delay is more likely to arise on some specific flight legs. Thus aircraft routing can be optimized while minimizing the expected value of total delay [112] – or any other risk measure of delay. A more details literature review on delay propagation in AIRCRAFT ROUTING PROBLEM is provided in Chapter 11. In both the fleet assignment and the delay versions, the instances of the AIRCRAFT ROUTING PROBLEM considered are one day-long, which corresponds to the first subproblem in the previous paragraph. And in both cases, the solution strategy derived is a column generation approach, where the subproblem generates routes, and the master problem combines these routes in an integer program whose constraints are analogue to those of system (6.4).

These column generation approaches can of course be used to solve the ground cost version

of the AIRCRAFT ROUTING PROBLEM introduced in Section 8.2.2, but they are more difficult to implement than our compact integer program. Besides, Chapter 11 shows that this compact integer program can be generalized to take into account delay propagation.

9 Column generation for crew pairing problem

This chapter focuses on the resolution of the constrained CREW PAIRING PROBLEM (7.2). Given a set of flight legs $\mathscr{L}$ to be operated, the CREW PAIRING PROBLEM builds the *sequences of flight legs* or *pairings* p assigned to the different crews. Pairings must respect several constraints to be feasible. These constraints are modeled though the set of feasible pairings $\mathscr{P}$. Each leg ℓ must be covered by exactly one route. Finally, in the integrated scheme, additional constraints relying on short connections sets S are added to obtain the CREW PAIRING PROBLEM (7.2), which we restate here.

$$\begin{array}{lll} \min & \displaystyle\sum_{p\in\mathscr{P}} c_p y_p & \\ s.t. & \displaystyle\sum_{p\ni\ell} y_p = 1, & \forall \ell \in \mathscr{L} \\ & \displaystyle\sum_{p\in\mathscr{P}} |p\cap S| y_p \leq |S|-1 & \forall S \in \mathscr{I}^{SC} \\ & y_p \in \{0,1\}, & \forall p \in \mathscr{P} \end{array} \tag{7.2}$$

When $\mathscr{I}^{SC}$ is empty, we obtain the usual CREW PAIRING PROBLEM.

The complexity of the crew working rules make the CREW PAIRING PROBLEM hard to express as a compact integer program analogue to the one derived for the AIRCRAFT ROUTING PROBLEM in Chapter 8. We therefore solve the formulation (7.2) of the CREW PAIRING PROBLEM. As the number of feasible pairings in $\mathscr{P}$ is exponential in the number of legs, we cannot consider the full pairing set $\mathscr{P}$ when solving this problem, which we therefore solve by column generation.

Chapter 9 is organized as follows:

- Section 9.1 details the column generation approach to the CREW PAIRING PROBLEM. Additional linking constraints specific to Air France problem are considered.
- Section 9.2 considers the pricing subproblem of the column generation, which happens to be a resource constrained shortest path problem. This subproblem is dealt with in the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework of Part I.
- Section 9.3 gives numerical results which prove the efficiency of the approach on Air France instances.
- Section 9.4 gives a literature review on the solution method of the CREW PAIRING PROB-

LEM.

9.1 Column generation approach to CREW PAIRING PROBLEM

9.1.1 Master problem

The CREW PAIRING PROBLEM solved at Air France is slightly more complicated than Problem (7.2) due to additional linking constraints. A *duty* contains all the legs operated by a crew on one day. Air France crew working rules contains the two following rules.

Air France working rule 1. A duty is *long* if it contains more than three legs. The proportion of long duties must be non-greater than β.

Air France working rule 2. A pairing is *long* if it contains more than three duties. The proportion of long pairings must be non-greater than α.

Let $\mathscr{P}^L \in \mathscr{P}$ be the set of long pairings, and $\mathbf{1}_{\mathscr{P}^L}$ be the indicator function of $\mathscr{P}^L$ on $\mathscr{P}$. Let $\mathscr{D}^S$ be the set of long duties that are not long duties, and $\mathscr{D}^L$ be the set of long duties. As a pairing p is a sequence of duties, we denote $p \cap \mathscr{D}^L$ the intersection of pairing p with $\mathscr{D}^L$ and $|p \cap \mathscr{D}^L|$ the number of long duties in $\mathscr{D}^L$. We can now state the CREW PAIRING MASTER PROBLEM.

$$
\begin{array}{l|lll}
 & \min & \displaystyle\sum_{p\in\mathscr{P}} c_p y_p & \\
(z_v) & \text{st} & \displaystyle\sum_{p\ni v} y_p - 1 = 0 & \forall v \in V \\
(t_s) & & \displaystyle\sum_{p\in\mathscr{P}} |p\cap S| y_p - (|S|-1) \le 0 & \forall S \in \mathscr{I}^{SC} \\
(\lambda) & & \displaystyle\sum_{p\in\mathscr{P}} (\mathbf{1}_{\mathscr{P}^L}(p) - \alpha) y_p \le 0 & \\
(\mu) & & \displaystyle\sum_{p\in\mathscr{P}} ((1-\beta)|p\cap\mathscr{D}^L| - \beta|p\cap\mathscr{D}^S|) y_p \le 0 & \\
 & & y_p \ge 0 & \forall p \in \mathscr{P}
\end{array}
\tag{9.1}
$$

where variables z_v, t_s, λ and μ are the dual variables associated respectively to the cover, the short connections constraints, the long duties, and the long pairings constraints. The dual of Problem (9.1) is Problem (9.2).

$$
\begin{array}{lll}
\max & \displaystyle\sum_{s\in\mathscr{I}^{SC}} -t_s m_s + \sum_{v\in V} -z_v & \\
\text{st} & \displaystyle c_p + \sum_{v\in p} z_v + \sum_{s\in\mathscr{I}^{SC}} t_s |p\cap S_s| + \lambda\left(\mathbf{1}_{\mathscr{P}^L}(p) - \alpha\right) & \\
 & \qquad + \mu\left((1-\beta)|p\cap\mathscr{D}^L| - \beta|p\cap\mathscr{D}^S|\right) \ge 0 & \forall p \in \mathscr{P} \\
 & z_v \in \mathbb{R}, & \forall v \in V \\
 & t_s, \lambda, \mu \ge 0, & \forall s \in \mathscr{I}^{SC}
\end{array}
\tag{9.2}
$$

Let $\mathscr{P}'$ be a subset of $\mathscr{P}$. The following lemma is a well-known result of the column generation theory.

Lemma 9.1. *Let* $\mathbf{y}^*$ *be an optimal solution of the relaxation of* (9.1) *restricted to* $\mathscr{P}'$*, and* $(\mathbf{z}^*, \mathbf{t}^*, \lambda^*, \mu^*)$ *be its dual. Then* $\mathbf{y}^*$ *is the optimal solution of the relaxation of* (9.1) *(on the complete*

$\mathscr{P}$) if the constraint of the dual problem (9.2) *is satisfied by* $(\mathbf{z}^*, \mathbf{t}^*, \lambda^*, \mu^*)$ *for all pairing* $p \in \mathscr{P}$.

9.1.2 Pricing subproblem

Given a solution $(\mathbf{z}, \mathbf{t}, \lambda, \mu)$ of the dual problem restricted to $\mathscr{P}'$, the PRICING SUBPROBLEM aims at finding the pairing $p \in \mathscr{P}$ whose constraint

$$c_p + \sum_{v \in p} z_v + \sum_{s \in \mathscr{I}^{SC}} t_s |p \cap S_s| + \lambda \left(\mathbf{1}_{\mathscr{P}^L}(p) - \alpha \right) + \mu \left((1-\beta)|p \cap \mathscr{D}^L| - \beta |p \cap \mathscr{D}^S| \right) \geq 0$$

is most violated. The PRICING SUBPROBLEM can thus be expressed as follows.

$$\min_{p \in \mathscr{P}} c_p + \sum_{v \in p} z_v + \sum_{s \in \mathscr{I}^{SC}} t_s |p \cap S_s| + \lambda \left(\mathbf{1}_{\mathscr{P}^L}(p) - \alpha \right) + \mu \left((1-\beta)|p \cap \mathscr{D}^L| - \beta |p \cap \mathscr{D}^S| \right) \tag{9.3}$$

Given a solution $(\mathbf{z}, \mathbf{t}, \lambda, \mu)$ of the dual problem, the objective of the pricing subproblem is called the reduced cost of pairing p and denoted $\tilde{c}_p$.

$$\tilde{c}_p = c_p + \sum_{v \in p} z_v + \sum_{s \in \mathscr{I}^{SC}} t_s |p \cap S_s| + \lambda \left(\mathbf{1}_{\mathscr{P}^L}(p) - \alpha \right) + \mu \left((1-\beta)|p \cap \mathscr{D}^L| - \beta |p \cap \mathscr{D}^S| \right) \tag{9.4}$$

Section 9.2 develops an algorithm to solve the PRICING SUBPROBLEM. Lemma 9.1 implies that if the solution of the PRICING SUBPROBLEM has positive reduced cost, then the optimal solution of the relaxed CREW PAIRING MASTER PROBLEM restricted to $\mathscr{P}'$ is the solution of the initial relaxed CREW PAIRING MASTER PROBLEM.

9.1.3 Column generation algorithm for the linear relaxation

The column generation algorithm to solve the relaxation of master problem (9.1) starts with a set of pairings $\mathscr{P}'$ such that (9.1) admits a feasible solution. To ensure feasibility, we initialize $\mathscr{P}'$ with, for each flight leg ℓ, a pairing p_ℓ containing only ℓ. Any other legs needed in p_ℓ to obtain its feasibility are deadhead legs, which means that the members of the crew that operates p_ℓ are only passengers on these legs. The following operations are then executed.

1. Initialize $\mathscr{P}'$.
2. Solve the relaxed CREW PAIRING MASTER PROBLEM (9.1) restricted to $\mathscr{P}'$ and store the solution in (y_p^*) and the dual $(z_v^*, t_s^*, \lambda^*, \mu^*)$.
3. Solve the PRICING SUBPROBLEM with reduced cost associated to $(z_v^*, t_s^*, \lambda^*, \mu^*)$ and denote p its solution.
4. If $\tilde{c}_p < 0$, then add p to $\mathscr{P}'$, and go to (2). Otherwise the algorithm is terminated.

Column generation theory ensures that this algorithm converges after a finite number of iterations, and at the end of the algorithm, (y_p^*) is an optimal solution of the relaxation of CREW PAIRING MASTER PROBLEM (9.1).

Lower bounds on the linear relaxation

The following proposition enables to obtain a lower bound on the value of the linear relaxation when the column generation scheme is still not finished. The number of legs $|\mathscr{L}|$ can be replaced by any other upper bound on the number of pairings in an optimal solution.

Proposition 9.2. *Let $\mathscr{P}'$ be a subset of $\mathscr{P}$, y_p^* be the optimal solution of the master problem* (9.1) *restricted to $\mathscr{P}'$, $c^* = \sum_p c_p y_p^*$, and $(z_v^*, t_s^*, \lambda^*, \mu^*)$ be its dual solution. Suppose that the corresponding* PRICING SUBPROBLEM *has been encoded as a* MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM *where vertices correspond to legs. Let $P_1, \dots, P_{|\mathscr{L}|}$ be $|\mathscr{L}|$ solutions of the pricing subproblem which satisfy one of the following conditions*

1. *$P_1, \dots, P_{|\mathscr{L}|}$ are the $|\mathscr{L}|$ feasible paths of minimum reduced cost $\tilde{c}_{P_1}, \dots, \tilde{c}_{P_{|\mathscr{L}|}}$.*
2. *$P_1, \dots, P_{|\mathscr{L}|}$ are the $|\mathscr{L}|$ feasible paths of minimum reduced cost $\tilde{c}_{P_1}, \dots, \tilde{c}_{P_{|\mathscr{L}|}}$ among those whose subpaths are non-dominated for the lattice ordered monoid order.*

Then $c^* + \sum_{i=1}^{|\mathscr{L}|} \left[\tilde{c}_{P_i}\right]^-$ is a lower bound on the optimal solution of the master problem (9.1) *on $\mathscr{P}$, where $[\cdot]^-$ denotes the negative part.*

Case 1 is a standard column generation result which enables to obtain a lower bounds when the k shortest path of the pricing subproblem are computed. In this dissertation, we use this result to obtain bounds when the generalized A* algorithm of Chapter 4 is used to solve the pricing subproblem. Case 2 is less standard stronger result that enables to obtain lower bounds when the label correcting algorithm of Chapter 4 is used.

Remark 9.1. In the reduction of the pricing subproblem to a MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM, the only vertices that do not correspond to legs are the origin and the destination vertex. As no paths are cut because of domination at these vertex, Case 2 still applies.

Proof. We start by introducing some notations and a standard result on linear program bases. For the purpose of the proof, let $\mathbf{A}$, $\mathbf{b}$, and $\mathbf{c}$ be one matrix and two vectors such that the master problem (9.1) can be rewritten

$$\begin{aligned} \min \quad & \mathbf{c}^t\mathbf{y}, \\ \text{s.t.} \quad & \mathbf{A}\mathbf{y} = \mathbf{b}, \\ & \mathbf{y} \geq 0, \end{aligned}$$

where slack variables have been introduced. These slack variables do not play a role in the objective. Let B be the basis corresponding to the solution $\mathbf{y}^*$ considered in the proposition, $\mathbf{A}_B$ the corresponding matrix, and $\mathbf{A}_N$ be the matrix of non-basic variables. Denoting $\mathbf{c}_B$ (resp. $\mathbf{c}_N$) the cost vector corresponding to the basic (resp. non-basic) variables, we can rewrite the master problem as follows.

$$\begin{aligned} \min \quad & \mathbf{c}_B^t\mathbf{A}_B + (\mathbf{c}_N^t - \mathbf{c}_B^t\mathbf{A}_B^{-1}\mathbf{A}_N)\mathbf{y}_N \\ \text{s.t.} \quad & \mathbf{y}_B = \mathbf{A}_B^{-1}\mathbf{b} - \mathbf{A}_B^{-1}\mathbf{A}_N\mathbf{y}_N \\ & \mathbf{y}_B \geq 0, \mathbf{y}_N \geq 0 \end{aligned} \tag{9.5}$$

Denoting $\mathbf{z}$ the vector of dual variables we can write the dual problem on $\mathscr{P}'$ as follows.

$$\begin{aligned} \max \quad & \mathbf{z}^t\mathbf{b} \\ \text{s.t.} \quad & \mathbf{z}^t\mathbf{A} \leq \mathbf{c} \\ & \mathbf{z} \geq 0 \end{aligned}$$

We now prove that, for each pairing p, the cost $\tilde{c}_p$ defined in Equation (9.4) is the reduced cost $(\mathbf{c}_N - \mathbf{c}_B^t\mathbf{A}_B^{-1}\mathbf{A}_N)_p$ associated to the pairing variable y_p. Let $\mathbf{z}_*^t = \mathbf{c}^t\mathbf{A}_B^{-1}$. We have $\mathbf{z}_*^t\mathbf{A} = \mathbf{c}^t\mathbf{A}_B^{-1}(\mathbf{A}_B, \mathbf{A}_N) = (\mathbf{c}_B^t, \mathbf{c}_B^t A_B^{-1} A_N) \geq 0$. As B is an optimal basis of the master problem reduced to $\mathscr{P}'$, we have $\mathbf{c}_N^t - \mathbf{c}_B^t\mathbf{A}_B^{-1}\mathbf{A}_N$. Thus, $\mathbf{z}_*^t\mathbf{A} \leq \mathbf{c}$, and $\mathbf{z}_*$ is a feasible solution of the dual reduced to $\mathscr{P}'$. Besides, the value of the dual at $\mathbf{z}_*$ is equal to the value of the primal $\mathbf{c}^t A_B^{-1} b$. Therefore, $\mathbf{z}_*$ is an optimal solution of the dual by weak duality. As a consequence, if $\mathbf{z}$ is an optimal solution of the dual, then $c_i - \sum_j z_j a_{ji}$ that appears in the i'th constraint of the dual is the reduced cost associated to variable y_i because the optimal solution of the dual is $\mathbf{c}^t\mathbf{A}_B^{-1}$ where $\mathbf{y}_B$ is the optimal solution of the primal. Given the definition of $\tilde{c}_p$, this gives the result mentioned at the beginning of the paragraph.

Case 1 follows directly from (9.5). Indeed, let y be a solution of the master problem, and $\mathbf{y}_N$ be the corresponding solution to (9.5). Then we have $\mathbf{c}^t\mathbf{y} = \mathbf{c}^t\mathbf{y}^* + (\mathbf{c}_N^t - \mathbf{c}_B^t\mathbf{A}_B^{-1}\mathbf{A}_N)\mathbf{y}_N$. Given the fact that $\sum_{p\in N} y_p \leq |\mathscr{L}|$ due to the family of constraints $\sum_{p\ni\ell} y_p = 1$, $(\mathbf{c}_N^t - \mathbf{c}_B^t\mathbf{A}_B^{-1}\mathbf{A}_N)\mathbf{y}_N$ is non-greater than the sum of the $|\mathscr{L}|$ smallest reduced costs of pairings in N, which gives the result.

We now consider Case 2. Let $(\mathbf{y}'_B, \mathbf{y}'_N)$ be an arbitrary solution master problem (9.5) on the full pairing set $\mathscr{P}$. Let ℓ be a leg, and let $\mathscr{P}(\ell)$ be the set of all pairings such that $y_{p'} > 0$ and whose last leg is ℓ. If $\mathscr{P}(\ell)$ is not empty, among all the pairing of minimum reduced cost ending in ℓ, there exists a non-dominated pairing. Let $p(\ell)$ be such a pairing. We have $\tilde{c}_{p(\ell)} \leq \tilde{c}_p \leq 0$ for each $p \in \mathscr{P}(\ell)$. Besides, as $\sum_{p\ni\ell} y_p = 1$, and as ℓ belongs to a pairing ending in ℓ, we have $\sum_{p\in\mathscr{P}(\ell)} y'_p \leq 1$. As each pairing has a last leg, we have

$$\sum_{p\in\mathscr{P}} \tilde{c}_p y'_p = \sum_{\substack{\ell\in\mathscr{L}:\\ \mathscr{P}(\ell)\neq\emptyset}} \sum_{p\in\mathscr{P}(\ell)} \tilde{c}_p y'_p \geq \sum_{\substack{\ell\in\mathscr{L}:\\ \mathscr{P}(\ell)\neq\emptyset}} \tilde{c}_{p(\ell)} \left(\sum_{p\in\mathscr{P}(\ell)} y'_p \right) \geq \sum_{\ell\in\mathscr{L}} \tilde{c}_{p(\ell)}.$$

The pairings $p(\ell)$ are non-dominated, and their last leg is ℓ, they are distinct for distinct ℓ. As a consequence, the right hand side of the last equation is a sum of at most $|\mathscr{L}|$ distinct non-dominated pairings, which gives the result. □

9.1.4 Integer solutions

In practice, we want to solve the integer program obtained by replacing $y_p \geq 0$ by $y_p \in \{0,1\}$ in (9.1). The standard procedure is to integrate column generation in a branch and bound procedure. As the set of pairings used in the column generation evolves, variables y_p cannot be used for branching. A practical solution consists in branching on the variables ν_a that are equal to $\sum_{p\ni a} y_p$. Nonetheless, as the solution of the relaxed master problem(9.1) is near optimal, we can avoid this branching scheme in our case.

We suppose to be at the end of the column generation scheme. We therefore have an optimal solution y^* of Problem (9.1), and we denote by $\tilde{c}_p$ the associated reduced cost. Let c^{UB} be an upper bound on the cost of the optimal solution.

Proposition 9.3. *If $\tilde{c}_p > c^{UB} - cy^*$, then $y_p = 0$ in any optimal solution of the integer solution of Problem* (9.1).

This result is a direct application of the following well-known and easy lemma on column generation to Problem (9.1).

Lemma 9.4. *Consider the following linear integer program program,*

$$\begin{array}{rl} \min & \mathbf{c}^T\mathbf{x} \\ s.t. & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \in \mathbb{Z}^n_+ \end{array} \tag{IP}$$

with $\mathbf{A} \in \mathbb{R}^{m\times n}$ and its linear relaxation,

$$\begin{array}{rl} \min & \mathbf{c}^T\mathbf{x} \\ s.t. & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq 0 \end{array} \tag{LP}$$

Let $\mathbf{y}'$ be a feasible solution of the dual of (LP) *restricted to the generated pairings, and c^{UB} be an upper bound on the solution of* (IP). *Let $\tilde{c}_i = c_i - \sum_j y'_j a_{ji}$ be the reduced cost of variables x_i. Then if i is such that its $\tilde{c}_i > c^{UB} - y'^T b$, then $\overline{x}_i = 0$ in any optimal solution $\overline{x}$ of* (IP).

Based on Proposition 9.3, the following scheme [18] enables to obtain an optimal solution based on a good quality solution provided by a heuristic.

1. Solve the linear relaxation of Problem (9.1) by column generation and denote y^* the optimal solution.
2. Solve the integer version of Problem (9.1) using a heuristic and denote c^{UB} the cost of the best solution found.
3. Generate all the pairings p whose reduced cost satisfies $\tilde{c}_p \leq c^{UB} - cy^*$.
4. Solve the integer version of Problem (9.1) using an integer programming solver.

Practically, an efficient heuristic is obtained by avoiding steps 2 and 3. Indeed, on all our instances, the same solution is obtained after adding the new columns, the integrality gap of the master problem (9.1) is non greater than 0.005%.

9.2 Resource constrained shortest path subproblem

In this section, we introduce an algorithm to solve the PRICING SUBPROBLEM, which aims at finding a feasible pairing with minimum reduced cost. Let us recall the main notions required by the definition of the PRICING SUBPROBLEM in Section 9.1.2. To be feasible, duties and pairings must respect several rules. The most constraining rules in Air France case are detailed in Section 9.2.1. The PRICING SUBPROBLEM is traditionally solved as a STANDARD

RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. A first approach consists in generating all the feasible duties to build the *duty graph*, and then solve the PRICING SUBPROBLEM as a STANDARD RESOURCE CONSTRAINED SHORTEST PATH PROBLEM in the duty graph. We later refer to this approach as the two steps approach. The second approach consists in solving directly the PRICING SUBPROBLEM as a STANDARD RESOURCE CONSTRAINED SHORTEST PATH PROBLEM in the pairing graph. Due to the inherent complexity of Air France working rules, the two steps approach was traditionally used at Air France [51]. The recent version of the algorithm used the `Boost C++` library [1]. Nonetheless, due to the large number of feasible connections at the hub, the number of feasible duties becomes too large to be tackled practically.

We therefore solve Air France PRICING SUBPROBLEM directly on the pairing connection graph. The complexity of Air France working rules makes the use of the `Boost C++` library impossible in practice. In this section, we show that Air France subproblem can be dealt within the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM formalism of Part I. Section 9.2.1 gives the crew working rules that define Air France specific PRICING SUBPROBLEM. Sections 9.2.2 and 9.2.3 show how to implement Air France PRICING SUBPROBLEM within the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework. Due to the nature of crew working rules, Sections 9.2.1 to 9.2.3 are fairly technical, and they can be skipped by a reader not interested in the practical implementation of an airline regulation.

In the next sections, we formulate the PRICING SUBPROBLEM as a MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. The first step is the definition of the digraph D in Section 9.2.2, and the second step is the definition of the lattice ordered monoid $(M, \oplus, \leqslant)$. Due to the technical complexity of Air France working rules, we do not give the exhaustive definition of the lattice ordered monoid which enables to treat all working rules. A list of tips to handle all working rules are given in Section 9.2.3.

9.2.1 Air France working rules

We now introduce some of the main working rules at Air France.

Air France working rule 3. The number of duties in a pairing must be non greater than $n_d^{\max}$.

Air France working rule 4. The time between two successive legs in a duty must belong to $[t_\alpha^{\min}, t_\alpha^{\max}]$

Air France working rule 5. A night rest must be longer than $t_r^{\min}$

where a night rest is the time between the end of a duty and the beginning of the next duty in a pairing.

Air France working rule 6. A crew is assigned to a special airport called its *base*. A pairing starts and ends in the same base.

In Air France instances considered, ORY and CDG airports are considered to be the same "Paris" base. Besides Paris base is the unique base.

Air France working rule 7. The number of legs in a duty must be non greater than 4.

Air France working rule 8. The flying time in a duty d with one or two legs starting at t must be non greater than $t_\ell^{\max;1,2}(t)$. The flying time in a duty d with three or four legs starting at t must be non greater than $t_\ell^{\max;3,4}(t)$.

Air France working rule 9. The duty time in a duty d with 1 or 2 legs starting at t must be non greater than $t_d^{\max;1,2}(t)$. The duty time in a duty d with 3 or 4 legs starting at t must be non greater than $t_d^{\max;3,4}(t)$.

The next rules define the notions of *normal night rest* and *reduced rest*.

Air France working rule 10. An inter-duty connection is a *normal night rest* if its length is greater than the maximum of 10 h 30 and the duty time of the preceding duty. Otherwise, it is a *reduced rest*.

Air France working rule 11. There is at most one reduced rest in a pairing.

Air France working rule 12. The number of legs in a duty following a reduced rest is 3.

Technically speaking, these rules can be split into two groups. Rules 3 to 6 are used in Section 9.2.2 to define the *pairing connection graph*, while Rules 7 to 12 are used in Section 9.2.3 to define the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM resource set.

9.2.2 Pairing connection graph

The *pairing connection graph* $D^{\mathscr{P}} = (V^{\mathscr{P}}, A^{\mathscr{P}})$ is an analogue for the CREW PAIRING PROBLEM of the routing connection graph introduced in the context of AIRCRAFT ROUTING PROBLEM in Section 8.1. As in the case of the *routing connection graph*, there is one vertex $v_\ell \in V^{\mathscr{P}}$ per flight leg ℓ in $\mathscr{L}$. Vertex set $V^{\mathscr{P}}$ is the union of the leg vertices $\{v_\ell | \ell \in \mathscr{L}\}$ and a source vertex o and a sink vertex d. Arcs in $A^{\mathscr{P}}$ correspond to *feasible connections*.

A pairing corresponds to a unique source to sink path in the connection graph. Therefore, the connection graph enables to enforce three types of working rules. The first type of working rules that can be implemented in the connection graph is composed of working rules which limit the scope of the connection graph. Take for instance Rule 3, which limits the number duties in a pairing. As the length of a duty is one day, this rule can easily be implemented by working on $n_d^{\max}$ days subgraphs of the connection graph.

Arcs in $A^{\mathscr{P}}$ correspond to *feasible connections*. As pairings are sequences of duties, there are two types of connections: *intra-duty connections* $\alpha \in A^{\text{intra}}$ are connections between legs inside a duty, whereas inter-duty connections $\alpha \in A^{\text{inter}}$ are connections between legs belonging to two successive duties. The common property of intra and inter-duty connections (ℓ_1, ℓ_2) is that ℓ_2 must depart from the arrival airport of ℓ_1 after the arrival of ℓ_1. Rule 4 defines the intra-duty feasible connection set A^{intra}, and Rule 5 defines the inter-duty feasible connection set A^{inter}.

Third, the arcs from the source and to the sink enable to define where a pairing can start and where a pairing can end, and thus to enforce Rule 6.

9.2.3 Working rules lattice ordered monoid

We continue the reduction of the PRICING SUBPROBLEM by defining the lattice ordered monoid $(\mathscr{R}_{\text{det.}}, \oplus_{\text{det.}}, \preccurlyeq_{\text{det.}})$, the non-decreasing cost function $c_{\text{det.}}$, the non-decreasing feasibility function $\rho_{\text{det.}}$, and the resources q_a for each arc a of the pairing connection graph. We use the index $\cdot_{\text{det.}}$, where det. stands for deterministic, to contrast with the stochastic monoid introduced in Chapter 11. The requirements on the instance of the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM defined by $D^{\mathscr{P}}$, $(\mathscr{R}_{\text{det.}}, \oplus_{\text{det.}}, \preccurlyeq_{\text{det.}})$, $c_{\text{det.}}$, $\rho_{\text{det.}}$, and (q_a) are thus the following ones: first *it must properly encode the* PRICING SUBPROBLEM, and second *it should be solved efficiently by the* MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM *algorithms* of Chapter 4.

MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM instance $D^{\mathscr{P}}$, $(\mathscr{R}_{\text{det.}}, \oplus_{\text{det.}}, \preccurlyeq_{\text{det.}})$, $c_{\text{det.}}$, $\rho_{\text{det.}}$, and (q_a) properly encodes the PRICING SUBPROBLEM if, given a source to sink path P and the corresponding sequence of flight legs p, then $\rho_{\text{det.}}(\bigoplus_{a\in P} q_a)$ is equal to 0 if p is a feasible pairing and to 1 otherwise, and $c_{\text{det.}}(\bigoplus_{a\in P} q_a)$ is the reduced cost of p. Therefore, *to encode properly the* PRICING SUBPROBLEM, *the resource x in the lattice ordered monoid $(\mathscr{R}_{\text{det.}}, \oplus_{\text{det.}}, \preccurlyeq_{\text{det.}})$ must contain all the information required to check Rules 7 to 12.*

The lattice ordered monoid $(\mathscr{R}_{\text{det.}}, \oplus_{\text{det.}}, \preccurlyeq_{\text{det.}})$ we use is such that $\mathscr{R}_{\text{det.}}$ is a triple $\mathscr{R}_d \times \mathscr{R}_i \times \mathscr{R}_d$. A resource $(q_{d_1}, q_i, q_{d_2}) \in \mathscr{R}_d \times \mathscr{R}_i \times \mathscr{R}_d$ contains the information required to check the working rules. Component q_{d_1} contains the information on the first duty, component q_i contains the information on the rests and on the middle duty(ies), and finally component q_{d_2} contains the information on the last duty. Resources (q_{d_1}, q_i, q_{d_2}) and $(q'_{d_1}, q'_i, q'_{d_2})$ can be added to obtain a resource $(\tilde{q}_{d_1}, \tilde{q}_i, \tilde{q}_{d_2}) = (q_{d_1}, q_i, q_{d_2}) \oplus_{\text{det.}} (q'_{d_1}, q'_i, q'_{d_2})$. If a pairing p has resource (q_{d_1}, q_i, q_{d_2}), and a pairing p' has resource $(q'_{d_1}, q'_i, q'_{d_2})$, then the pairing corresponding to p followed by p' is a pairing with resource $(\tilde{q}_{d_1}, \tilde{q}_i, \tilde{q}_{d_2})$. This implies that $\oplus_{\text{det.}}$ is *non-commutative*, as pairing p followed by pairing p' is not the same pairing as p' followed by p. Note that a resource contains information on a sequence of flight legs which is only a *partial pairing*, in the sense that this sequence may not start and end in a base. In the remaining of the section, we first define $\mathscr{R}_d$ and $\mathscr{R}_i$, and then $\oplus_{\text{det.}}$ and $\preccurlyeq_{\text{det.}}$. The information on a partial pairing described by a resource element of $\mathscr{R}_d \times \mathscr{R}_i \times \mathscr{R}_d$ is illustrated on Figure 9.1.

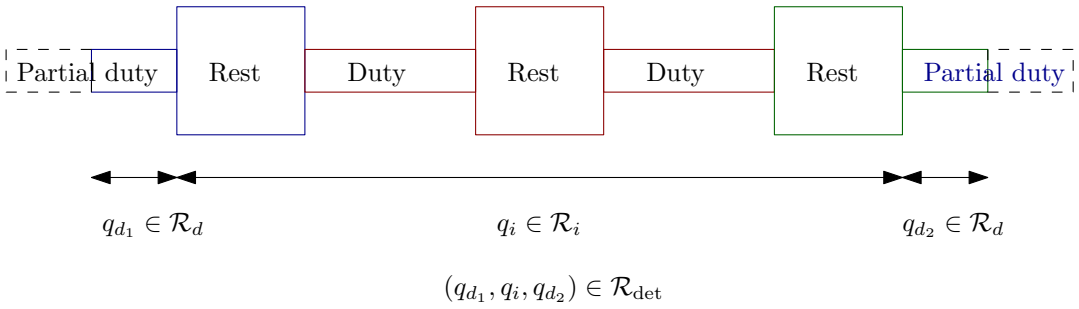


Figure 9.1 – Resource in $\mathscr{R}_{\text{det.}} = \mathscr{R}_d \times \mathscr{R}_i \times \mathscr{R}_d$ containing the information on a partial duty.

Resource components in $\mathscr{R}_d$ thus contain the information on duties. In order to be able to check Rules 7 to 9, elements of $\mathscr{R}_d$ have three components: one for the number of legs, one

for the flying time, and one for the duty time. This leads us to define $\mathscr{R}_d$ as $\mathbb{Z}_+ \times \mathbb{R}_+^2$.

Resource components in $\mathscr{R}_i$ contain the information on the rests and the internal duties necessary to check Rules 10 to 12 on rests, and the information necessary to check Rules 7 to 9 on the internal duties. The information on duties and rests in red on Figure 9.1 is summed up by a binary variable $b \in \{0,1\}$, with $b = 0$ meaning that Rules 7 to 9 are satisfied by all duties in red, and Rules 10 and 12 are satisfied by all rests in red. Besides, the number of reduced rest $n_{rr} \in \mathbb{Z}_+$ among the rests in red is stored in order to be able to check Rule 12. The detailed information required to check Rules 10 and 12 is stored for the first rest, in blue on Figure 9.1, that is to say the length $q_r \in \mathbb{R}$ of the first rest and the number of flight legs in the duty following the rest in blue $n_{fd} \in \mathbb{Z}_+$. For the last rest, in green on Figure 9.1, the duty time of the duty immediately preceding the rest, and the length of the rest must be stored, which means two components in $\mathbb{R}$. To sum things up, set $\mathscr{R}_i$ is defined to be equal to $\{0,1\} \times \mathbb{Z}_+^2 \times \mathbb{R}_+^3$ endowed with its canonical product order.

The set $\mathscr{R}_{\text{det.}} = \mathscr{R}_d \times \mathscr{R}_i \times \mathscr{R}_d$ is endowed with the canonical product order on $(\mathbb{Z}_+ \times \mathbb{R}_+^2) \times (\{0,1\} \times \mathbb{Z}_+^2 \times \mathbb{R}_+^3) \times (\mathbb{Z}_+ \times \mathbb{R}_+^2)$. The operator $\oplus_{\text{det.}}$ is defined by an algorithm which cannot be easily written as an equation. The sequence of operations realized by this algorithm is illustrated on Figure 9.2. It can be shown the canonical order $\leqslant$ is compatible with operator $\oplus_{\text{det.}}$ by showing that each function of the algorithm, described by arrows on Figure 9.2, is non-decreasing.

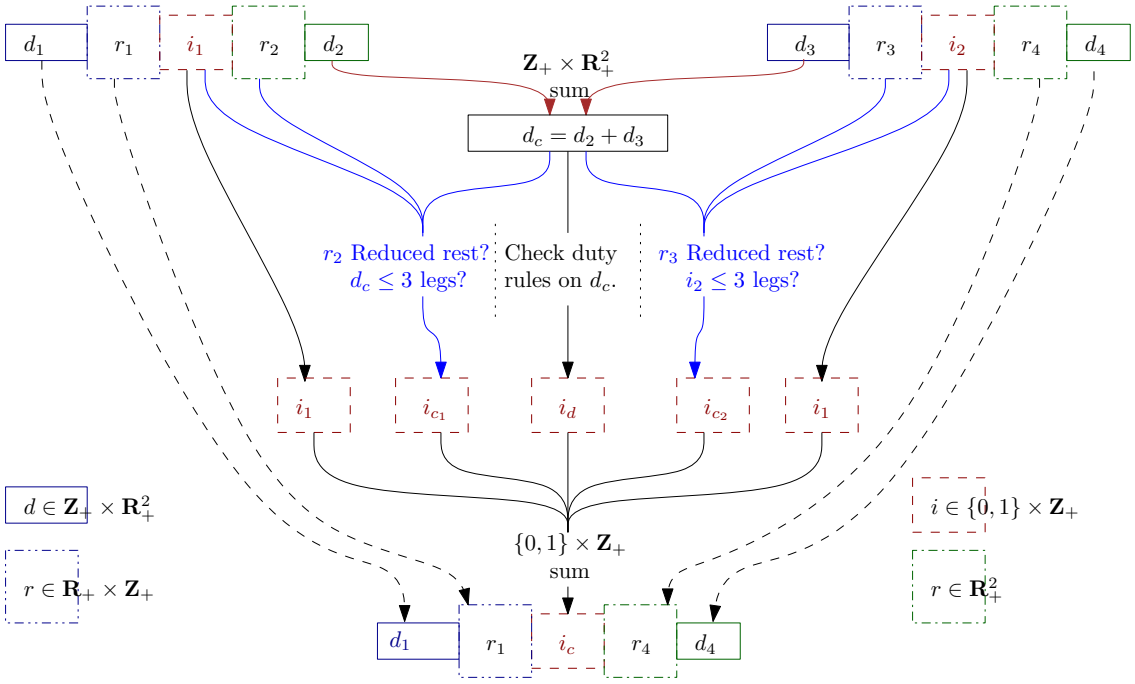


Figure 9.2 – Algorithm defining $\oplus_{\text{det.}}$ on $\mathscr{R}_{\text{det.}} = D \times I \times D$.

Note that the case of the resources for sequences of flight legs corresponding to a single duty must be dealt with. The real lattice ordered monoid used is therefore $\mathscr{R}_{\text{det.}} \cup \mathscr{R}_d$ with properly

adapted operators $\oplus_{\text{det.}}$ and $\preccurlyeq_{\text{det.}}$. Besides, we don't take into account all the working rules in this section. We also have to take into account the cost and the reduced cost components. Therefore additional components must be added to $\mathscr{R}_{\text{det.}}$ and additional cases must be dealt with in the algorithm. As these technical details present no major difficulty, we chose to omit them for clarity.

The cost function $c_{\text{det.}}$ and the feasibility function $\rho_{\text{det.}}$ are defined as algorithms on $\mathscr{R}_{\text{det.}}$. For instance, feasibility function tests that Rules 7 to 9 are satisfied on the components in $\mathscr{R}_d$, that Rules 10 and 12 are satisfied by the $\mathscr{R}_i$ component, and Rule 11 is satisfied globally by the three components. As the cost is non-decreasing in the different components of the resource, and the working rules are mainly limitations on the size of these components, functions $c_{\text{det.}}$ and $\rho_{\text{det.}}$ are easily shown to be non-decreasing.

Intra-duty arc resources are chosen to be elements of $\mathscr{R}_d$, and their value corresponds to the flying time and the duty time of the first leg of the connection. Inter-duty arc resources are chose as elements of $\mathscr{R}_{\text{det.}}$ with both $\mathscr{R}_d$ components equal to 0.

9.2.4 Algorithm

All the enumeration algorithms of Chapter 4 can be used. As the connexion graph is acyclic, their convergence is ensured by Theorems 4.3, 4.7 and 4.8. The numerical results of the next section show that the label correcting algorithm with bounds provides by a clustering state graph gives the best performances.

9.3 Numerical results on CREW PAIRING PROBLEM

9.3.1 Results on the main instance

We test the performance of the approach of this chapter on the CREW PAIRING PROBLEM instances introduced in Tables 7.1 and 7.2. We use the label correcting algorithm with a clustering state graph of our `latticeRCSP` library described in Appendix C to solve the pricing subproblem. As the connection graphs is acyclic, we use the algorithm of Section 6.2.1 to build the state graph. The exact scheme presented in Section 9.1.4 is used to obtain an integer solution. We use `CPLEX 12.1.0` to solve the linear and the integer programs. The numerical experiments are performed on a server with 128 Gb of ram and 12 cores at 2.4 GHz. The algorithms are not parallelized. All instances are solved to optimality.

Table 9.1 provides the performance of the algorithms. The first columns provide the instance solved and the number of flight legs in the instance. The next column provides the maximum number κ of state vertices per vertex in the connection graph. This parameter is given in input to the state graph building algorithm. We give advices on how to choose κ in Section 9.3.3. The two next columns provide the number of iterations in the column generation, and the percentage of time spent in the pricing subproblem. This pricing time includes the time needed by the computation of the clustering state graph during the preprocessing, and the time needed by the bounding algorithm and the enumeration algorithm at each step of the column generation. The two next columns indicate the percentage of the total CPU time spent solving linear program during the column generation, and the percentage of time spent

Instance	Legs	κ	Col. Gen. Iter	Pricing time	LP time	MIP time	Integ. gap %	Total time (hh:mm:ss)
CP50	290	10	89	59.01%	39.56%	1.14%	0.000%	00:00:17
CP70	408	10	152	57.97%	41.16%	0.69%	0.000%	00:01:12
CP90	516	50	268	73.19%	26.49%	0.26%	0.000%	00:09:41
A318	669	150	394	86.60%	13.34%	0.05%	0.000%	01:21:22
A319	957	150	264	60.66%	39.14%	0.15%	0.000%	00:10:47
A320	918	150	226	74.54%	25.20%	0.20%	0.000%	00:08:35
A321	778	150	382	65.82%	32.60%	1.25%	0.012%	00:33:51
A318-9	1626	150	867	69.71%	30.21%	0.07%	0.001%	05:43:00
A320fam	3398	250	2166	43.28%	56.62%	0.10%	0.003%	104:05:59

Table 9.1 – Crew Pairing Results.

solving the final mixed integer program. The next column provides the gap between the linear relaxation at the end of the algorithm and the integer solution returned, and the last column gives the total time needed by the algorithm.

We note that the gap provided is not an optimality gap: Proposition 9.3 ensures that all the instances are solved to optimality. We provide this gap to outline the fact that, at the end of the column generation, the master problem has *near integrality property*. Indeed, the gap between the master problem integer value and its linear relaxation is extremely small. This is the reason why the resolution of the final MIP is fast and takes less than 1% of the total computation time on industrial instances. We also remark that, if the pricing subproblem is the most time consuming phase on small and medium instances, its part in the computation time decreases with the size of the instances. Indeed, thanks to the use of a clustering state graph, the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM algorithms scale well, and the simplex algorithm for the resolution of the relaxed master problem becomes the most time consuming part on the largest instance.

Table 9.1 shows that four days of computing time are required to solve the largest industrial instance. In the next section, we analyze the convergence of the column generation algorithm in order to provide directions to reduce this computing time.

9.3.2 Convergence, branching scheme and stabilization

Figure 9.3 shows the evolution of the value of the master problem along the column generation algorithm. The value has been scaled by the value of the optimal solution of the linear relaxation. The instance solved is the A320 fam, and the results on Figure 9.3 and in Table 9.1 correspond to the same numerical experiment. The lower bound is provided by Proposition 9.2. The bound plotted on Figure 9.3 is the best bound obtained up to the current iteration, as the bound provided by Proposition 9.2 is not monotone.

The first direction to reduced the computation time is to initialize the algorithm with a solution of good quality. Indeed, we have initialized the column generation with a solution of poor quality: initial pairings contain only one flight leg and one deadhead leg. A good quality

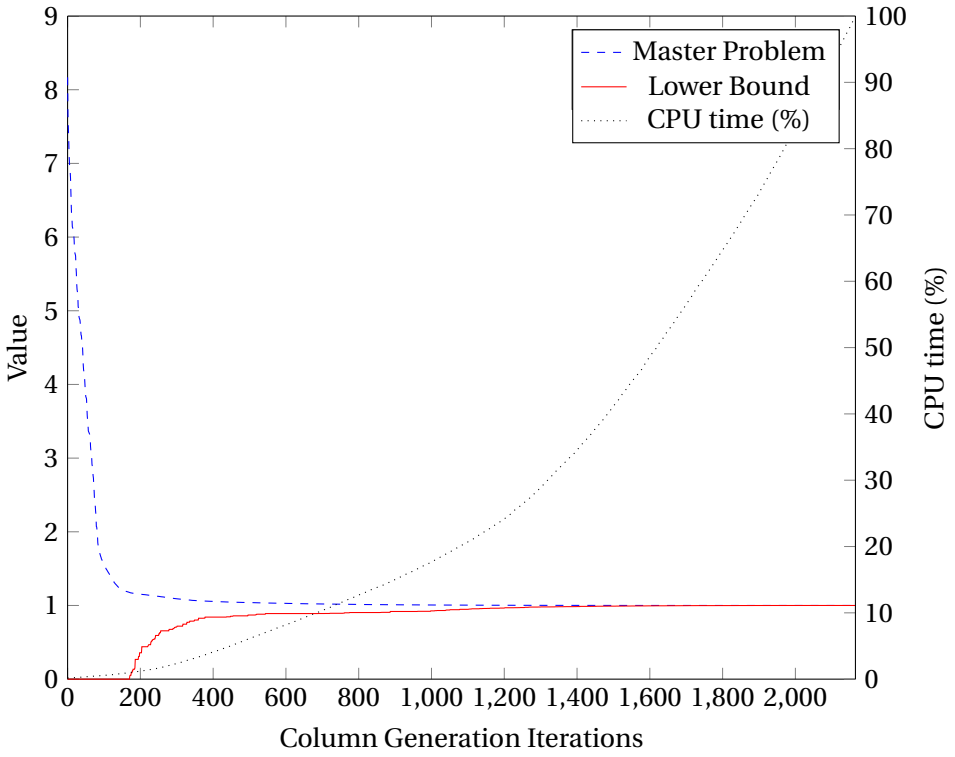


Figure 9.3 – Column Generation tail effect on instance A320 fam.

initial solution can be produced either by a heuristic or by adapting the solution obtained on previous horizons. Nonetheless, the first iterations are not the most time consuming ones. The master problem solution is at 1% from the optimum after 897 iterations and less that 15% of the CPU time, and it is at 0.1% after 1435 iterations and 37% of the CPU time. Indeed, column generation is known to suffer from the *tail effect*. Convergence is slow at the end of the algorithm, and the last iterations are the most time consuming, both due to the increased difficulty of the subproblem and to the increased size of the master problem.

To avoid the large computing time required by the last 0.1%, a simple idea is to stop the column generation algorithm before its convergence. Nonetheless, the reason why the MIP solver finds quickly the optimal integer solution at the end of the column generation is that the master problem has "near integrality" property at the end of the column generation. Unfortunately, when the column generation is interrupted before its end, the master problem obtained does not have the "near integrality" property, and the gap between the solution found by the MIP solver and the optimal solution after two hours of computing time is around 10%. Another direction is to implement an approximate branching scheme which alternates several steps of column generation with heuristic branching. Once again, the quality of the solution we obtain using this heuristic is poor, with an optimality gap of around 3%.

Therefore, implementing a method that reduces the tail effect looks like a more promising direction. The tail effect of Column Generation algorithms is often correlated to the facts

that, first, the primal is degenerated, and second, the solution of the dual oscillates. Dual penalization methods [31, 60] tackle with dual oscillations by solving a modified dual problem with a penalization term that increases with the distance to the current dual solution. Among these dual penalization methods, *stabilized column generation* has shown good performances when tested by Du Merle et al. [60] on the CREW PAIRING PROBLEM. Elhallaoui et al. [66, 67] shows that dynamic constraint aggregation enables to accelerate convergence of the column generation on a CREW PAIRING PROBLEM. The principle of this method is to aggregate constraints in a first phase in order to reduced primal degeneration.

9.3.3 Pricing subproblem algorithms

Table 9.2 provides numerical results on the column generation algorithm performance to solve the relaxed master problem using different algorithms of Chapters 4 and 6 to solve the pricing subproblem. Both the generalized A* and label correcting algorithms are tested with simple bounds and with sets of bounds provided by clustering state graphs of different sizes. The three first columns of Table 9.2 provide the instance name, the algorithm used to solve it, and the maximum number κ of state vertices per vertex of the connection graph. We use the symbol – instead of the maximum number of state vertices per vertex to indicate that no state graph has been used. The two next columns provide the number of vertices and the number of arcs in the connection graph. The next columns provide the proportion of time spent in the pricing algorithm, and the average number of paths enumerated. The average number of paths enumerated is an average on each MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM solved. At each step, a MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM is solved for each 4 days graph. As the instances considered correspond to cyclic week horizons , 7 MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM are solved at each step. The "Cut Dom." column gives the proportion of paths cut by the dominance test when the label correcting algorithm. The remaining of the paths have been cut by the bound test. We indicate – in the column "Cut Dom." when the generalized A* algorithm is used, as this algorithm does not use the dominance test to discard path. The last column provides the total CPU time.

We note that the use of a state graph enables to divide the number of paths enumerated by the algorithm by a factor 3 to 20. Besides, the number of paths enumerated decreases with the size of the state graph. On all these instances, using a state graph enables to reduce the computation time when compared to using no state graph. But the size of the state graph has two effects on the computation time. On the one hand, a larger state graph enables to reduce the number of paths enumerated. But on the other hand, a larger state graph requires larger preprocessing times, and gives larger sets of bounds, which means a larger computing time for each bound test (Clu) done by the enumeration algorithm. Therefore, a balance must be found between a too small clustering state graph, which leads to large computation times due to the number of paths enumerated, and a too large, which leads to avoidable preprocessing times. Practically, we suggest to use a state graph such that the computation time needed by the preprocessing algorithm that computes the bounds is of the same order of magnitude as the time needed by the enumeration algorithm. Finally, we note that, as the difficulty of the problem increases with the instance size, larger instances require larger clustering state

Instance	Alg.	κ	Vert.	Arcs	Pricing time	av. nb paths enum.	Cut Dom.	Total time (hh:mm:ss)
CP50	cor.	–	293	1006	75.28%	6.016e+03	6.89%	00:00:23.0
		10			59.87%	1.601e+03	4.01%	00:00:17.2
		50			69.45%	7.766e+02	4.69%	00:00:24.7
		100			77.06%	6.467e+02	5.03%	00:00:33.3
	A*	–			65.51%	1.512e+04	–	00:00:22.3
		10			61.54%	2.512e+03	–	00:00:19.6
		50			68.30%	1.299e+03	–	00:00:24.4
		100			75.68%	9.982e+02	–	00:00:33.2
CP70	cor.	–	411	1705	90.61%	2.752e+04	7.69%	00:04:40.7
		10			58.48%	7.613e+03	4.28%	00:01:12.1
		50			68.89%	3.917e+03	5.24%	00:01:23.0
		100			77.43%	3.085e+03	5.77%	00:01:42.6
	A*	–			83.97%	1.084e+05	–	00:02:29.0
		10			60.96%	1.493e+04	–	00:01:09.4
		50			73.16%	8.126e+03	–	00:01:37.2
		100			78.48%	6.659e+03	–	00:02:04.9
CP90	cor.	–	519	2490	98.86%	1.488e+05	9.81%	02:56:33.1
		10			81.86%	4.119e+04	5.88%	00:12:36.3
		50			73.42%	2.534e+04	4.87%	00:09:39.7
		100			77.98%	1.879e+04	5.60%	00:10:27.5
	A*	–			93.93%	3.403e+05	–	00:45:43.6
		10			77.47%	1.401e+05	–	00:13:13.2
		50			82.32%	1.000e+05	–	00:16:54.2
		100			85.03%	7.382e+04	–	00:19:38.9
A318	cor.	–	672	3741	97.87%	2.746e+05	8.99%	05:35:41.8
		10			96.02%	2.420e+05	6.62%	05:06:46.6
		50			88.78%	1.489e+05	3.73%	02:06:43.4
		100			86.97%	1.270e+05	3.72%	01:32:49.6
		150			86.94%	1.138e+05	3.75%	01:40:45.4
	A*	–			97.02%	1.148e+06	–	05:17:08.4
		10			96.85%	1.070e+06	–	04:54:52.4
		50			86.94%	5.735e+05	–	01:45:27.4
		100			88.52%	4.783e+05	–	01:45:18.2

Table 9.2 – Influence of subproblem on computations time of the column generation scheme for the linear relaxationof the CREW PAIRING PROBLEM.

graphs.

Concerning the choice between the generalized A* and the label correcting algorithms, we underline the fact that, in all cases, less than 10% of the paths are cut by the dominance test. This is a numerical confirmation of our statement in Chapter 4 that bound tests cut better than dominance tests when there are many constraints. The number of paths enumerated by the label correcting algorithm is smaller than the number of paths enumerated by the generalized A* algorithm, but as the dominance test takes time, the effect on the total computation time is not clear. On the large instances, when no clustering state graph is used, as the number of non-dominated paths enumerated is large, dominance test becomes long, and using the generalized A* algorithm becomes more interesting. Nonetheless, when used with a state graph of proper side, the label correcting algorithm is slightly faster than the generalized A* algorithm. We therefore suggest to use the label correcting algorithm with a state graph whose size is chosen using the criterion mentioned in the previous paragraph.

9.4 Bibliographical remarks

Crews and fuel are the two main sources of variable costs for airlines. The CREW PAIRING PROBLEM has therefore been intensively studied. Due to the complexity of the problem and to the size of industrial instances, heuristics have long been used to generate feasible solutions [12, 94]. During the two last decades, column generation has become the state of the art technique to solve the CREW PAIRING PROBLEM [12, 19, 24, 39, 51, 94, 105, 106, 114, 116, 168]. This technique has indeed two main advantages. First, it enables to hide working rules in the subproblem, which can be efficiently solved within the resource constrained shortest path framework [51]. Second, crew working rules are implemented in the pricing subproblem independently of the master problem. This is a strong industrial advantage, as only the subproblem solver needs to be updated when these rules change, and thus column generation codes are easy to maintain. Second, column generation are easily turned into matheuristics, enabling to cope with instances of large size. A detailed state of the art on CREW PAIRING PROBLEM is available in [87]. Recent approaches to the CREW PAIRING PROBLEM have been developed in the context of integrated approaches to the CREW PAIRING PROBLEM and the CREW ASSIGNMENT PROBLEM [155, 177]. The CREW ASSIGNMENT PROBLEM affects pairing to crews according to their preferences. We do not consider it in this dissertation because given the specifities of Air France working rules, it cannot be integrated with the CREW PAIRING PROBLEM at Air France. Saddoune et al. [155] reports that Dynamic Constraint Aggregation [66, 66] enables to improve the convergence of the column generation algorithm.

The specificity of our approach lies in the fact that we use the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM algorithms of Part I instead of the usual resource constrained shortest path algorithms [97]. We can therefore use bounds to better cut partial paths. The use of bounds on paths is non-standard on the crew pairing subproblem as it is highly non-linear. Besides, by computing a state graph once and for all before the column generation, we collect information on the structure shared by all the instances of the pricing subproblem instances solved along the column generation. This information enables to speed-up the resolution of each pricing subproblem, and thus strongly reduce the total time spent in the

column generation. Finally, the time spent in the preprocessing is easily controlled through the number of state vertices allowed per connection graph vertex, and can thus be adapted to the difficulty of the instance solved.

10 Numerical experiments on integrated problem

This chapter details numerical experiments on the INTEGRATED PROBLEM solution scheme introduced in Chapter 7, using as black boxes for the AIRCRAFT ROUTING PROBLEM and the CREW PAIRING PROBLEM the solution schemes introduced respectively in Chapter 8 and Chapter 9. We use as test set the instances introduced in Chapter 7.

In Section 7.1, we have introduced an exact version and an approximate version of the solution scheme. The difference between these two schemes is done through the short connection constraint (7.2c). In the exact version, the short connection constraint reads

$$\sum_{p \in \mathscr{P}} |p \cap S| y_p \leq |S| - 1, \quad \forall S \in \mathscr{I}^{SC}$$

while in the approximate version, it is replaced by

$$\sum_{p \in \mathscr{P}} |p \cap S| y_p \leq m_S, \quad \forall S \in \mathscr{I}^{SC},$$

with $m_S \ll |S| - 1$. We tested the exact algorithm on the small instances. For the approximate algorithm, we used $m_S = \gamma |S|$ for different values of $\gamma < 1$.

The INTEGRATED PROBLEM is solved using the solution scheme of Section 7.1. AIRCRAFT ROUTING PROBLEM feasibility is solved using compact integer program (8.2). The CREW PAIRING PROBLEM is solved by column generation: following the approach of Section 9.1.4, we solve the linear relaxation of the master problem to optimality, and then solve the integer master problem with the columns generated. The pricing subproblem is solved using the label correcting algorithm, and a clustering state graph is used to improve bounds quality. The numerical experiments are performed on a server with 128 Gb of ram and 12 cores at 2.4 GHz. The algorithms are not parallelized. `CPLEX 12.1.0` is used to solve the linear and integer program.

Table 10.1 provides numerical results on the instances of Section 7.2. The first column provides the instance solved, and the next column the short connection constraints strength parameter γ. The "Cuts added" column indicates the number of short connection constraints (7.2c) that have been added to obtain an optimal solution. As a short connection constraint is added if the last CREW PAIRING PROBLEM solution leads to an infeasible AIRCRAFT ROUTING

Instance	γ	Cuts added	κ	CG it. total	CP CG time	CP MIP time	AR time	Sho. Con.	Gap	Total time (dd:hh:mm:ss)
AR4	0.9	24	20	36	9.58%	54.85%	35.57%	63	0.03280%	00:00:18.4
	0.8	6	20	23	24.31%	43.80%	31.89%	57	0.09370%	00:00:05.4
	0.6	3	20	17	26.37%	44.43%	29.20%	49	0.81856%	00:00:04.5
AR8	0.9	61	20	172	20.55%	50.54%	28.90%	148	0.0070%	00:08:46.7
	0.8	4	20	87	48.55%	29.95%	21.50%	136	0.0073%	00:00:48.9
	0.6	5	20	144	55.66%	29.07%	15.28%	114	0.9426%	00:01:13.3
AR12	0.9	55	20	305	51.97%	31.90%	16.13%	213	0.0051%	00:27:48.3
	0.8	35	20	381	19.21%	76.86%	3.03%	204	0.0403%	01:21:07.1
	0.6	30	20	633	0.76%	99.07%	0.17%	159	1.4285%	01:11:28:58.2
A318	0.9	6	150	460	95.53%	2.56%	1.91%	323	0.0002%	01:53:47.4
	0.8	5	150	461	95.62%	2.71%	1.67%	312	0.0029%	01:48:02.3
	0.6	14	150	1030	26.27%	73.20%	0.53%	269	0.6857%	16:02:03.1
A318-9	0.9	2	150	915	97.66%	1.71%	0.63%	790	0.0008%	06:34:30.9

Table 10.1 – Numeric results on INTEGRATED PROBLEM

PROBLEM, the column"Cuts added" indicates the number of AIRCRAFT ROUTING PROBLEM and CREW PAIRING PROBLEM instances solved along the solution scheme. For each cut added, a column generation is launched to solve the subproblem. The two next columns provide the maximum number κ of state vertices per connection graph vertex in the clustering state graph, and the total number of column generation iterations realized. The column "CP CG time" provides the proportion of the total CPU time spent in the column generation, i.e. solving the pricing subproblem and the linear relaxation of the master problem. The next column provides the proportion of the total CPU time spent solving the integer version of the crew pairing master problem. The column "AR time" provides the percentage of the total CPU time spent solving AIRCRAFT ROUTING PROBLEM compact integer program (8.2). The column "Sho. Con." gives the number of short connections in the final solution is provided. The linear relaxation of the crew pairing master problem (7.2) with no short connection constraint is used as the lower bound on the cost of an optimal solution. The gap provided is between the cost of the solution returned and this lower bound. Finally, the last column provides the total CPU time needed by the INTEGRATED PROBLEM algorithm.

Exact scheme on small instances

We have tested the exact scheme where $m_S = |S| - 1$ on instances AR4 and AR8. On both instances, the algorithm does not converge after 2000 iterations of the INTEGRATED PROBLEM algorithms. The exact scheme does not enable to solve the INTEGRATED PROBLEM in practice.

Approximate scheme

We emphasize the fact that the solution returned by the approximate algorithm is near optimal. Practically speaking, the gap obtained is non-greater than 10 euros. Besides, the computation

time needed to obtain a near optimal solution of the INTEGRATED PROBLEM is of the same order of magnitude as the time needed to obtain a solution of the CREW PAIRING PROBLEM in Table 9.1. The additional cost due to AIRCRAFT ROUTING PROBLEM feasibility is negligible. Besides, the number of short connections in the solution returned is large: around half of the arc in the solution are short connections.

On industrial instances, choosing a constraint strength parameter γ equal to 0.9 enables to obtain solutions of excellent quality in reasonable time. On some instances, using stronger constraints with $\gamma = 0.8$ enables to accelerate convergence and thus reduce the computation time. Nonetheless, too strong constraints with $\gamma = 0.6$ lead to both extra computation time and poorer quality solutions.

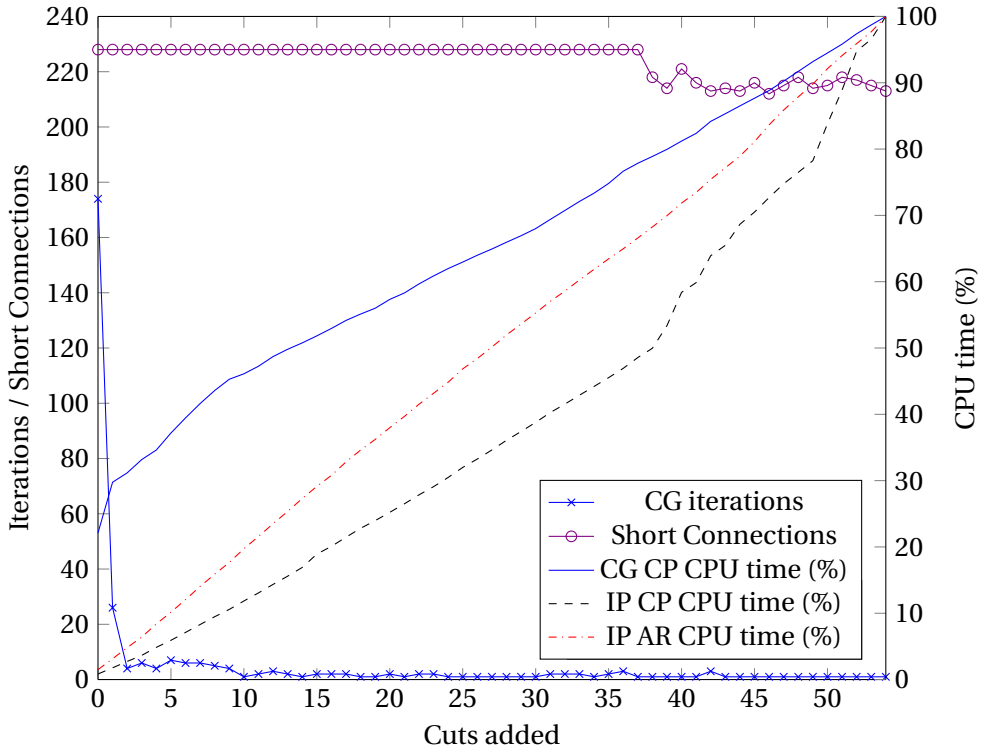


Figure 10.1 – Computation time of INTEGRATED PROBLEM scheme on AR12 with $\gamma = 0.9$

Figure 10.1 shows the evolution of the cumulative computation time spent in the column generation, the crew pairing integer master problem, and the aircraft routing compact integer program along the INTEGRATED PROBLEM solution scheme, for instance AR12 with cut strength parameter $\gamma = 0.9$. It reveals several interesting aspects on the computation time. A column generation is launched at each step of the INTEGRATED PROBLEM scheme, but only the first one is really time consuming. Indeed, during the first iteration, interesting columns must be generated. These interesting columns can then be reused at the next iterations. As we can see on Figure 10.1, very few columns need to be added in the following steps: the short connection constraints do not affect too much the columns in the optimal solution. Besides, the time needed to solve the CREW PAIRING PROBLEM integer program doesn't evolve much along the

algorithm. Finally, proving the non-feasibility of the AIRCRAFT ROUTING PROBLEM becomes more and more difficult along the steps, and its contribution to the CPU time increases along the algorithm. Nonetheless, solving the AIRCRAFT ROUTING PROBLEM remains easier than solving the CREW PAIRING PROBLEM.

Figure 10.2 illustrates the same quantities on the same instance but with $\gamma = 0.6$. This choice leads to a very large total CPU time. When we compare Figure 10.2 and Figure 10.1, the effect of too strong cuts appears clearly. As cuts are strong, the CREW PAIRING PROBLEM is perturbed by these short connection cuts, and a larger number of new columns must be generated by the column generation at each iteration of the INTEGRATED PROBLEM scheme. The number of short connections in the solution is strongly reduced. Besides, the integer program for the CREW PAIRING PROBLEM becomes harder and requires higher computations time. We note that the time spent solving the integer version of the crew pairing master problem remains constant during the last twenty iterations. This is due to the fact that we have set a time limit of two hours on the resolution of this integer problem, and the fact that this time limit is reached by each of the twenty last iterations.

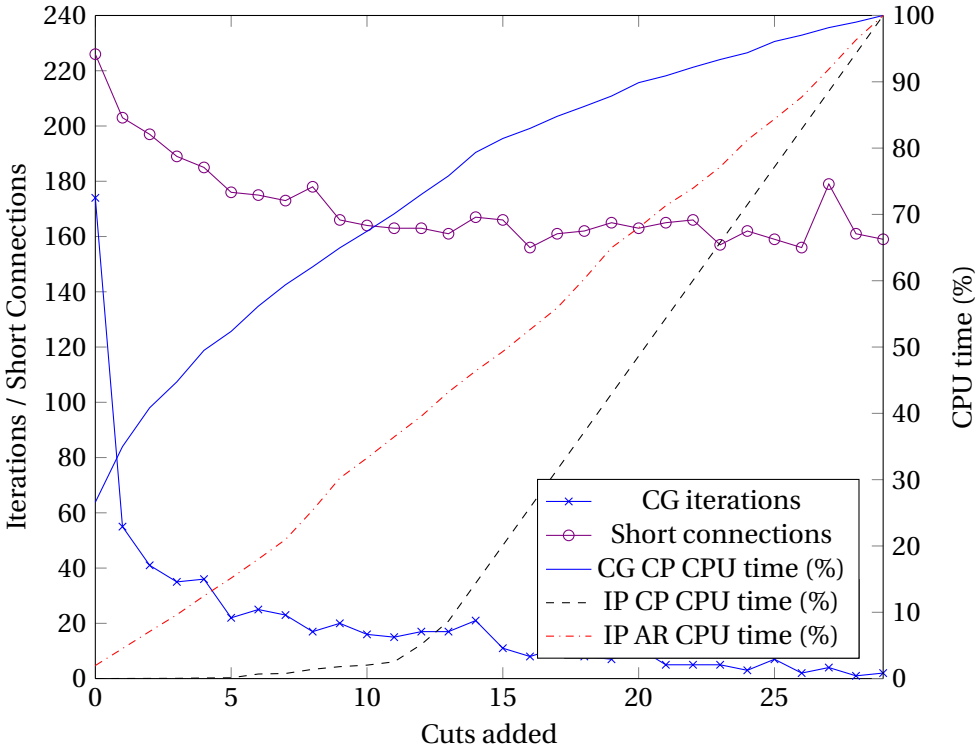


Figure 10.2 – Computation time of INTEGRATED PROBLEM scheme on AR12 with $\gamma = 0.6$

11 Managing delay in airline operations

Due to many external factors such as weather, airport congestion, or mechanical failures, airline operations are subject to delay. As both an airplane and its crew are needed to operate a flight leg, leg delay propagates along airplane routes and crew pairings. As a consequence, the resilience of operations with respect to delay depends on the solution of both the AIRCRAFT ROUTING PROBLEM and the CREW PAIRING PROBLEM. The STOCHASTIC INTEGRATED PROBLEM considered in this chapter consists in the simultaneous resolution of the AIRCRAFT ROUTING PROBLEM and the CREW PAIRING PROBLEM with probabilistic constraints on the resulting delay on each flight leg.

Delay appears on flight legs. On connections, it is partially absorbed due to the slack between flight legs, and partially propagated to the next flight leg. Let $(\tilde{\ell}, \ell)$ be a feasible connection for the AIRCRAFT ROUTING PROBLEM, $t^{\text{arr.}}_{\tilde{\ell}}$ be the arrival time of $\tilde{\ell}$, $t^{\text{dep.}}_{\ell}$ be the departure time of ℓ, and $\Delta t^{\mathscr{R}}$ be the minimum time needed by an airplane to perform a connection. The *routing slack* $s^{\mathscr{R}}_{(\tilde{\ell},\ell)} = t^{\text{dep.}}_{\ell} - t^{\text{arr.}}_{\tilde{\ell}} - \Delta t^{\mathscr{R}}$ corresponds to the maximum airplane delay that can be absorbed by routing connection $(\tilde{\ell}, \ell)$. Similarly, the *pairing slack* $s^{\mathscr{P}}_{(\tilde{\ell},\ell)} = t^{\text{dep.}}_{\ell} - t^{\text{arr.}}_{\tilde{\ell}} - \Delta t^{\mathscr{P}}$ corresponds to the maximum crew delay that can be absorbed by pairing connection $(\tilde{\ell}, \ell)$. The *total delay* random variable ξ_ℓ on a flight leg ℓ can be modeled as the sum of the *intrinsic delay* random variable ξ^{intr}_ℓ of ℓ, which appears when ℓ is operated, and the *propagated delay* random variable ξ^{prop}_l, which comes from the delay on the leg before ℓ on the route r and on the pairing p that contain ℓ,

$$\xi_\ell(\mathbf{x}, \mathbf{y}) = \xi^{\text{prop}}_\ell(\mathbf{x}, \mathbf{y}) + \xi^{\text{intr}}_\ell, \tag{11.1}$$

where $\mathbf{x}$ and $\mathbf{y}$ are respectively solutions of the AIRCRAFT ROUTING PROBLEM (6.4) and of the CREW PAIRING PROBLEM (6.5). The intrinsic delay ξ^{intr}_ℓ and the slacks are inputs of the model. Given these inputs, the propagated delay $\xi^{\text{prop}}_\ell(\mathbf{x}, \mathbf{y})$ and the total delay $\xi_\ell(\mathbf{x}, \mathbf{y})$ are functions of the solution AIRCRAFT ROUTING PROBLEM and CREW PAIRING PROBLEM solutions $\mathbf{x}$ and $\mathbf{y}$. We denote $\pi_\ell(\mathbf{x})$ the *predecessor* of ℓ in the route r that covers ℓ in routing solution $\mathbf{x}$, and $\pi_\ell(\mathbf{y})$

the predecessor of ℓ in the pairing p that covers ℓ in pairing solution $\mathbf{y}$. We therefore have

$$\xi_\ell^{\text{prop}}(\mathbf{x},\mathbf{y}) = \max\left(0, \xi_\ell^{\text{rout.}}, \xi_\ell^{\text{pair.}}\right), \quad \text{where} \quad \left| \begin{array}{l} \xi_\ell^{\text{rout.}} = \xi_{\pi_\ell(\mathbf{x})}(\mathbf{x},\mathbf{y}) - s^{\mathscr{R}}_{(\pi_\ell(\mathbf{x}),\ell)} \\ \xi_\ell^{\text{pair.}} = \xi_{\pi_\ell(\mathbf{y})}(\mathbf{x},\mathbf{y}) - s^{\mathscr{P}}_{(\pi_\ell(\mathbf{y}),\ell)} \end{array} \right.$$

where $\xi_\ell^{\text{rout.}}$ corresponds to the delay propagated by airplanes, and $\xi_\ell^{\text{pair.}}$ corresponds to the delay propagated by crews. Both $\xi_\ell^{\text{rout.}}$ and $\xi_\ell^{\text{pair.}}$ depend on $\mathbf{x}$ and $\mathbf{y}$. To make it easier to compare to other delay models introduced later, we rewrite delay model (11.1) in a single equation without propagated delay random variables ξ^{prop}.

$$\xi_\ell(\mathbf{x},\mathbf{y}) = \max\left(0, \xi_\ell^{\text{rout.}}, \xi_\ell^{\text{pair.}}\right) + \xi_\ell^{\text{intr}}, \quad \text{where} \quad \left| \begin{array}{l} \xi_\ell^{\text{rout.}} = \xi_{\pi_\ell(\mathbf{x})}(\mathbf{x},\mathbf{y}) - s^{\mathscr{R}}_{(\pi_\ell(\mathbf{x}),\ell)} \\ \xi_\ell^{\text{pair.}} = \xi_{\pi_\ell(\mathbf{y})}(\mathbf{x},\mathbf{y}) - s^{\mathscr{P}}_{(\pi_\ell(\mathbf{y}),\ell)} \end{array} \right. \tag{11.2}$$

To make probability computations feasible, assumptions must be made on the distributions of ξ_ℓ^{intr}. In this dissertation, we consider the case of independent ξ_ℓ^{intr} and the case of non-independent but scenario based distributions for ξ_ℓ^{intr}. Generic distributions can be approximated by scenario based distributions through the use of sampling. An analysis of the quality of the solutions obtained when sampling is used is provided a the end of the chapter.

In this chapter, we consider a stochastic version of INTEGRATED PROBLEM (6.7) with the additional constraint that the probability of having a delay ξ_ℓ on leg ℓ greater than a threshold α is non greater than β.

$$\mathbb{P}\left[\xi_\ell(\mathbf{x},\mathbf{y}) \geq \alpha\right] \leq \beta, \quad \forall \ell \in \mathscr{L}. \tag{11.3}$$

We therefore obtain the following STOCHASTIC INTEGRATED PROBLEM.

$$\begin{array}{ll} \min & \displaystyle\sum_{p\in\mathscr{P}} c_p y_p \\ s.t. & \left| \begin{array}{ll} \displaystyle\sum_{r\ni\ell} x_r = 1, & \forall \ell \in \mathscr{L} \\ \displaystyle\sum_{p\ni\ell} y_p = 1, & \forall \ell \in \mathscr{L} \\ \displaystyle\sum_{p\ni\alpha} y_p \leq \sum_{r\ni\alpha} x_r, & \forall \alpha \in A^{\text{short}} \\ \mathbb{P}\left[\xi_\ell(\mathbf{x},\mathbf{y}) \geq \alpha\right] \leq \beta, & \forall \ell \in \mathscr{L} \\ x_r \in \{0,1\}, & \forall r \in \mathscr{R} \\ y_p \in \{0,1\}, & \forall p \in \mathscr{P} \end{array} \right. \end{array} \tag{11.4}$$

This chapter extends the INTEGRATED PROBLEM algorithms in Chapters 7 to 9 to the STOCHASTIC INTEGRATED PROBLEM.

- After an analysis of the complexity of probability computations in delay model (11.2), Section 11.1 details the STOCHASTIC INTEGRATED PROBLEM algorithm. The algorithm of Section 11.1 uses as black boxes solution schemes for stochastic versions of the AIRCRAFT ROUTING PROBLEM and of the CREW PAIRING PROBLEM.

- Section 11.2 extends the column generation scheme for the CREW PAIRING PROBLEM in Chapter 9 to a stochastic version of the CREW PAIRING PROBLEM. This algorithm relies on the work on stochastic resource constrained shortest path problems in Part I. Its efficiency is tested through numerical experiments on industrial instances. To the best of our knowledge, it is the first approach to the stochastic CREW PAIRING PROBLEM where delay is managed through probabilistic constraints. Besides, this approach applies to both independent ξ_ℓ^{intr} and scenario based ξ_ℓ^{intr}. Its efficiency is tested on Air France CREW PAIRING PROBLEM instances.
- Section 11.3 extends the compact integer program approach to AIRCRAFT ROUTING PROBLEM in Chapter 8 to stochastic versions of the problem in the case of scenario based ξ_ℓ^{intr} distributions. Its efficiency is tested through numerical experiments on industrial instances. Its efficiency is tested on Air France AIRCRAFT ROUTING PROBLEM instances.
- Section 11.4 tests on Air France instances the efficiency of the STOCHASTIC INTEGRATED PROBLEM approach of Section 11.2 when the algorithms of Sections 11.2 and 11.3 are used as black boxes.
- Finally, Section 11.5 gives an exponential bound on the probability of obtaining an infeasible solution when sampled distributions are used in problem (11.4).

11.1 Solution approach to robust integrated problem

11.1.1 Intrinsic delay distribution and difficulty of the inference problem

One key element for the tractability of the STOCHASTIC INTEGRATED PROBLEM is the choice of the probability distribution of intrinsic delay random variables ξ_ℓ^{intr}. The two natural choices for these distributions are discrete distributions with finite support under the assumption that ξ_ℓ^{intr} are independent, or scenario based distributions with a finite number of scenarios. Even under the assumption that intrinsic delay random variables are independent, computing probability functionals on total delay random variables is not tractable. The core of the problem is illustrated on Figure 11.1. Suppose that intrinsic delay random variables ξ_ℓ^{intr} are independent and that delay propagates according to model (11.2). Then delay ξ_{ℓ_2} of leg ℓ_2 and delay ξ_{ℓ_3} of leg ℓ_3 are not independent as they both depend on the delay ξ_{ℓ_1} of ℓ_1. Due to this non-independence, delay ξ_{ℓ_4} of leg ℓ_4 cannot be computed directly from the distributions of ξ_{ℓ_2} and ξ_{ℓ_3} using model (11.2), but requires the joint distribution on $(\xi_{\ell_3}, \xi_{\ell_4})$. In the next paragraph, we show that, for theoretical reasons, the computation of the distribution of ξ_ℓ requires to compute the joint distribution on a number of leg delay variables approximately equal to the number of planes in the fleet, and is therefore exponential in the number of planes in the fleet. From a practical point of view this means that exact delay distributions cannot be computed in the case of independent intrinsic delay random variables. As a consequence, we only use scenario based distributions when considering the delay model (11.2).

From a theoretical point of view, if variables ξ_ℓ^{intr} are supposed to be independent, model (11.2) implies that the probability distribution of the random vector $\left(\xi_\ell(\mathbf{x},\mathbf{y})\right)_{\ell\in\mathscr{L}}$ is a graphical model [108] on the digraph $D(\mathbf{x},\mathbf{y}) = (V, A(\mathbf{x},\mathbf{y}))$ whose vertex set V is the leg set $\mathscr{L}$, and whose arc set $A(\mathbf{x},\mathbf{y})$ contains the connections in the routing solution $\mathbf{x}$ and the connections in the pairing solution $\mathbf{y}$. As a consequence, the exact computation of the distribution or any

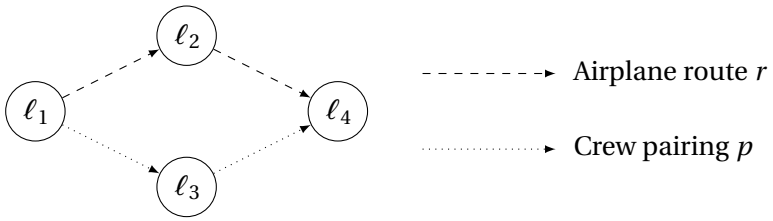


Figure 11.1 – Non independence of legs delay

probability functional on ξ_ℓ is an *exact inference problem* in $D(\mathbf{x},\mathbf{y})$. Graphical model theory tells that solving the inference problem requires to compute joint distributions on a number of random variables equal to the treewidth of $D(\mathbf{x},\mathbf{y})$. The complexity of the inference problem is thus exponential in the treewidth of $D(\mathbf{x},\mathbf{y})$. The treewidth of a graph is an abstract graph notion, but for our purpose, it suffices to know that the treewidth of $D(\mathbf{x},\mathbf{y})$ is approximately equal to the number of airplanes in the fleet to conclude that the exact computation of the distribution of $\xi_\ell(\mathbf{x},\mathbf{y})$ is not tractable. Nonetheless, this analysis suggests that approximate inference algorithms in graphical models [108] can be used as an alternative to scenario based distributions.

11.1.2 Exact solution scheme

We now extend the solution scheme for the deterministic INTEGRATED PROBLEM in Chapter 7 to the STOCHASTIC INTEGRATED PROBLEM. The general structure of the solution scheme remains unchanged. We first solve a CREW PAIRING PROBLEM to obtain a pairing solution $\mathbf{y}$, and we then test if the AIRCRAFT ROUTING PROBLEM remains feasible given solution $\mathbf{y}$. The difference with Chapter 7 is that both the CREW PAIRING PROBLEM solved in the first step and the AIRCRAFT ROUTING PROBLEM solved in the second step must satisfy additional probabilistic constraints.

Let $\mathscr{I}$ be a set of infeasible solutions. We define the *constrained* STOCHASTIC CREW PAIRING PROBLEM as follows.

$$\begin{aligned} \min_{y_p} \quad & \sum_{p\in\mathscr{P}} c_p y_p \\ s.t. \quad & \left|\begin{array}{ll} \sum_{p\ni\ell} y_p = 1, & \forall \ell \in \mathscr{L} \\ \widetilde{\mathbf{y}}\cdot\mathbf{y} \le \widetilde{\mathbf{y}}\cdot\mathbf{1} - 1, & \forall \widetilde{\mathbf{y}} \in \mathscr{I} \\ \mathbb{P}\left[\xi_\ell(\mathbf{0},\mathbf{y}) \ge \alpha\right] \le \beta, & \forall \ell \in \mathscr{L} \\ y_p \in \{0,1\}, & \forall p \in \mathscr{P} \end{array}\right. \end{aligned} \tag{11.5}$$

Note that as the CREW PAIRING PROBLEM is solved before the AIRCRAFT ROUTING PROBLEM, delay propagation is considered only along pairings in $\xi_\ell(\mathbf{0},\mathbf{y})$, as the routing solution $\mathbf{x}$ is not fixed when CREW PAIRING PROBLEM is solved. This is equivalent to replacing model (11.2) by the following pairing only delay model.

$$\xi_\ell(\mathbf{0},\mathbf{y}) = \max\left(0, \xi_{\pi_\ell(\mathbf{y})} - s^{\mathscr{P}}_{(\pi_\ell(\mathbf{y}),\ell)}\right) + \xi^{\text{intr}}_\ell = \xi_\ell(p_\ell(\mathbf{y})) \tag{11.6}$$

where ξ_ℓ only depends on the pairing $p_\ell(\mathbf{y})$ of leg ℓ in pairing solution $\mathbf{y}$. On the contrary, when AIRCRAFT ROUTING PROBLEM is solved, pairing solution $\mathbf{y}$ is known, and the full delay model can be considered in the constrained STOCHASTIC AIRCRAFT ROUTING PROBLEM.

$$\begin{array}{ll} \sum_{r \ni \ell} x_r = 1, & \forall \ell \in \mathscr{L} \\ \sum_{p \ni \alpha} y_p \leq \sum_{r \ni \alpha} x_r, & \forall \alpha \in A^{\text{short}} \\ \mathbb{P}\left[\xi_\ell(\mathbf{x},\mathbf{y}) \geq \alpha\right] \leq \beta, & \forall \ell \in \mathscr{L} \\ x_r \in \{0,1\}, & \forall r \in \mathscr{R} \end{array} \tag{11.7}$$

The exact STOCHASTIC INTEGRATED PROBLEM solution scheme can now be presented. The set of infeasible solutions $\mathscr{I}$ is initially empty. The following steps are executed

1. Solve the constrained STOCHASTIC CREW PAIRING PROBLEM (11.5) to obtain $\mathbf{y}$. Stop if (11.5) does not admit solution.
2. Solve the constrained STOCHASTIC AIRCRAFT ROUTING PROBLEM (11.7) with $\mathbf{y}$ as input:
   - if Problem (11.7) is not feasible, then add $\mathbf{y}$ to $\mathscr{I}$. Return to Step (1).
   - else store the solution of the Problem (11.7) in $\mathbf{x}$ and return $(\mathbf{x},\mathbf{y})$.

Theorem 11.1. *The above algorithm converges after a finite number of steps. If a solution* $(\mathbf{x},\mathbf{y})$ *is returned, then it is an optimal solution of the* STOCHASTIC INTEGRATED PROBLEM (11.4). *Otherwise Problem* (11.4) *does not admit solutions.*

This solution scheme uses as black boxes solution schemes for the constrained STOCHASTIC CREW PAIRING PROBLEM (11.5) and of the constrained STOCHASTIC AIRCRAFT ROUTING PROBLEM (11.7). Its efficiency relies on our ability to solve efficiently both of these problems. Efficient methods to solve them are provided respectively in Sections 11.2 and 11.3. Nonetheless, constraints $\widetilde{\mathbf{y}}\cdot\mathbf{y} \leq \widetilde{\mathbf{y}}\cdot\mathbf{1} - 1$ are not strong enough on large industrial instances. We therefore introduce in Section 7.1.2 a heuristic version of the algorithm to be able to deal with industrial instances.

Sketch of the proof of Theorem 11.1. The proof of Theorem 11.1 is almost identical to the proof of Theorem 7.1. The only difference is that it is the constraint

$$\widetilde{\mathbf{y}}\cdot\mathbf{y} \leq \widetilde{\mathbf{y}}\cdot\mathbf{1}, \quad \forall \widetilde{\mathbf{y}} \in \mathscr{I}$$

instead of the constraint

$$\sum_{p \in \mathscr{P}} |p \cap S| y_p \leq |S| - 1, \quad \forall S \in \mathscr{I}^{SC}$$

that ensures that a solution considered at one step is not considered anymore in the next iterations. □

11.1.3 Delay minimization as objective of the Aircraft Routing Problem

In this section, we provide a solution scheme motivated by practical applications. From an industrial point of view, short connection constraint (6.6) and delay constraint (11.3) do not play the same role in the STOCHASTIC INTEGRATED PROBLEM. Indeed, the short connection constraint is a strong feasibility constraint, and a solution that violates it cannot be executed in practice, whereas the delay constraint is only there to enforce robustness, and can be violated on some industrial instances. To stick to this industrial policy, we adopt a differentiated approach to these constraints: instead of enforcing delay constraint (11.3) in the STOCHASTIC AIRCRAFT ROUTING PROBLEM, we minimize its violation. We emphasize the fact that the scheme obtained does not solve anymore problem (11.4). Indeed, if the solution returned is a feasible solution of the deterministic problem (6.7), it may not satisfy the delay constraint (11.3). The resolution of (11.4) on large instances remains an open academic problem.

Following the approach mentioned above, we replace (11.7) by the following STOCHASTIC AIRCRAFT ROUTING PROBLEM,

$$\begin{array}{ll} \min\limits_{\mathbf{x}} & \sum\limits_{\ell\in\mathscr{L}} \mathbb{E}\left(\xi_\ell(\mathbf{x},\mathbf{y})\right)^+ \\ s.t. & \left|\begin{array}{ll} \sum\limits_{r\ni\ell} x_r = 1, & \forall \ell\in\mathscr{L} \\ \sum\limits_{p\ni\alpha} y_p \le \sum\limits_{r\ni\alpha} x_r, & \forall \alpha\in A^{\text{short}} \\ x_r\in\{0,1\}, & \forall r\in\mathscr{R} \end{array}\right. \end{array} \tag{11.8}$$

where $(\cdot)^+$ denotes the positive part. Besides, as delay does not lead to infeasibility anymore in the STOCHASTIC AIRCRAFT ROUTING PROBLEM, the only reason why a solution $\mathbf{y}$ can lead to infeasibility in the STOCHASTIC AIRCRAFT ROUTING PROBLEM is the short connection constraint (6.6). As a consequence, instead of forbidding solution $\mathbf{y}$, we forbid the use of all the short connections in $\mathbf{y}$ simultaneously. A short connection constraint set is a set of short connections S. We denote $\mathscr{I}^{SC}$ the set of all short connections set. We obtain the following modified version of the STOCHASTIC CREW PAIRING PROBLEM (11.5).

$$\begin{array}{ll} \min & \sum\limits_{p\in\mathscr{P}} c_p y_p \\ s.t. & \left|\begin{array}{ll} \sum\limits_{p\ni\ell} y_p = 1, & \forall \ell\in\mathscr{L} \\ \sum\limits_{p\in\mathscr{P}} |p\cap S| y_p \le |S|-1 & \forall S\in\mathscr{I}^{SC} \\ \mathbb{P}\left[\xi_\ell(\mathbf{0},\mathbf{y})\ge\alpha\right]\le\beta, & \forall \ell\in\mathscr{L} \\ y_p\in\{0,1\}, & \forall p\in\mathscr{P} \end{array}\right. \end{array} \tag{11.9}$$

Note that this new version of the CREW PAIRING PROBLEM is equivalent to the deterministic one (11.5), with the additional constraint $\mathbb{P}\left[\xi_\ell(\mathbf{0},\mathbf{y})\ge\alpha\right]\le\beta$. We finally obtain the following exact optimization scheme for our new problem. The set of short connections constraints $\mathscr{I}^{SC}$ is initially empty. The following steps are executed.

1. Solve the constrained STOCHASTIC CREW PAIRING PROBLEM (11.9) to obtain $\mathbf{y}$.
2. Solve the constrained STOCHASTIC AIRCRAFT ROUTING PROBLEM (11.8):
   - if Problem (11.8) is not feasible, then add S to $\mathscr{I}^{SC}$, where S is the set of short connections $a \in A^{\text{short}}$ contained in a pairing p such that $y_p = 1$ in $\mathbf{y}$. Return to Step (1).
   - else store the solution of the Problem (11.8) in (x_r) and return $(\mathbf{x}, \mathbf{y})$.

We have mentioned in Section 7.1.2 that the algorithm for the deterministic CREW PAIRING PROBLEM can be turned into a matheuristic by replacing $\leq |S|-1$ by $\leq m_S$ with $m_S \ll |S|-1$ in Problem (11.9). We also apply this technique to its stochastic counterpart to be able to deal with industrial instances.

11.1.4 Probabilistic constraints in STOCHASTIC CREW PAIRING PROBLEM column generation

The large number and the non linearity of crew working rules make them difficult to express inside an integer program. Mathematical programming approach to Crew Pairing problem therefore "hide" these working rules in the sub-problem of a column generation. The strength of Problems (11.5) and (11.9) comes from the fact that, as aircraft routing solution is not active in these problems, the delay propagates only along pairings. As a consequence, the delay of a flight leg $\xi_\ell(\mathbf{0}, \mathbf{y})$ only depends on the pairing p containing ℓ, and delay model (11.2) simplifies to (11.6). The delay constraint can therefore also be "hidden in the subproblem" by restricting the set of feasible pairings $\mathscr{P}$ to the set of pairings which satisfy the delay constraint. From a column generation point of view, it means that we only have to adapt the subproblem when extending the column generation scheme of the CREW PAIRING PROBLEM to the STOCHASTIC CREW PAIRING PROBLEM. Indeed, if we define

$$\mathscr{P}_{\text{del}} = \left\{ p \in \mathscr{P} | \mathbb{P}\left[\xi_\ell(p) \geq \alpha\right] \leq \beta, \forall \ell \in p \right\},$$

the STOCHASTIC CREW PAIRING PROBLEM (11.9) can be rewritten as the following master problem

$$\begin{array}{l|lll} & \min & \displaystyle\sum_{p \in \mathscr{P}_{\text{del}}} c_p y_p, & \\ (z_v) & \text{s.t.} & \displaystyle\sum_{p \ni v} y_p - 1 = 0 & \forall v \in V, \\ (t_S) & & \displaystyle\sum_{p \in \mathscr{P}_{\text{del}}} |p \cap S| y_p - (|S|-1) \leq 0 & \forall S \in \mathscr{I}^{SC}, \\ & & y_p \geq 0, \ \forall p \in \mathscr{P}_{\text{del}}, & \end{array} \tag{11.10}$$

which is identical to the CREW PAIRING PROBLEM master problem (9.1), the only difference being that $\mathscr{P}$ has been replaced by $\mathscr{P}_{\text{del}}$. Note that the long duty and long pairing linking constraints have been omitted for clarity in this chapter. We therefore obtain the following

pricing subproblem

$$\min_{p\in\mathscr{P}_{\text{del}}} c_p + \sum_{v\in p} z_v + \sum_{S\in\mathscr{S}^{SC}} t_S|p\cap S|. \tag{11.11}$$

Again, the only difference with the deterministic PRICING SUBPROBLEM of Section 9.1.2 is that $\mathscr{P}$ has been replaced by $\mathscr{P}_{\text{del}}$. Therefore, extending the column generation approach of Chapter 9 to the STOCHASTIC CREW PAIRING PROBLEM only requires to modify the subproblem solution scheme. This is the goal of Section 11.2.

Remark 11.1. The full delay model (11.2) is used in the STOCHASTIC AIRCRAFT ROUTING PROBLEM (11.8). As delay propagates both along pairings and routes, the analysis of Section 11.1.1 applies. Therefore, the delay on a leg ℓ cannot be expressed as a function of the route r that contains it. If a column generation was to be applied to this problem, it would have to consider the probabilistic constraints as linking constraints instead of hiding them in the subproblem. Therefore, column generation does not suit well to this problem. This is the reason why we use a compact integer programming approach to the STOCHASTIC AIRCRAFT ROUTING PROBLEM in Section 11.3.

11.2 Column generation approach to stochastic Crew Pairing

11.2.1 From deterministic to stochastic subproblem reduction

Deterministic pricing subproblem reduction

As for the deterministic PRICING SUBPROBLEM in Chapter 9, we reduce the stochastic pricing subproblem (11.11) to a MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. We therefore reuse the *pairing connection digraph* $D = (V^{\mathscr{P}}, A^{\mathscr{P}})$, the lattice ordered monoid $(\mathscr{R}_{\text{det.}}, \oplus_{\text{det.}}, \leqslant_{\text{det.}})$, the two monotone oracles $c_{\text{det.}}$, $\rho_{\text{det.}}$, and the arc resources $q_a \in M$ defined in Chapter 9. We recall that the vertices v in $V^{\mathscr{P}}$ are the legs ℓ in $\mathscr{L}$, and the arcs in $A^{\mathscr{P}}$ are the feasible connections. Digraph D also contains an origin vertex o, a destination vertex d, and arcs starting in o (resp. ending in d) which enable to control where and when pairings can start (resp. end). Pairings p are o-d paths P in D, and we have the property that *the sequence of flight legs p of an o-d path P in the pairing connection graph D is a feasible pairing $p \in \mathscr{P}$ if and only if $\rho_{\text{det.}}(q_P^{\text{det.}}) = 0$. Besides, if $\rho_{\text{det.}}(q_P^{\text{det.}}) = 0$, we have $\rho_{\text{det.}}(q_P^{\text{det.}}) = c_p$, where c_p is the cost of p.*

To enhance readability and underline the fact that they correspond to the deterministic problem, the resources q_a of Chapter 9 are denoted $q_a^{\text{det.}}$ in this chapter.

Stochastic pricing subproblem reduction

As we already mentioned, the only difference between the stochastic subproblem (11.11) and its deterministic counterpart (9.3) is that the pairing set $\mathscr{P}$ is replaced by

$$\mathscr{P}_{\text{del}} = \left\{p \in \mathscr{P} | \mathbb{P}\left[\xi_\ell(p) \geq \alpha\right] \leq \beta, \forall \ell \in p\right\}.$$

We therefore build a delay lattice ordered monoid $\mathscr{R}_{\text{del.}}$, arc delay resources $q_a^{\text{del.}}$ and a delay feasibility oracle $\rho_{\text{del.}}$ in such a way that, given a path P in D and the corresponding pairing p,

$$\rho_{\text{del.}}\left(\bigoplus_{a\in P} q_a^{\text{del.}}\right) = 0 \quad \text{if and only if} \quad \mathbb{P}\left[\xi_\ell(p) \geq \alpha\right] \leq \beta \quad \text{for all} \quad \ell \in p. \tag{11.12}$$

The practical construction of such a lattice ordered monoid in the case of independent or scenario based delay distributions is detailed in Section 11.2.2.

As the product of two lattices ordered monoid endowed with the product order is a lattice ordered monoid, we use the product $\mathscr{R} = \mathscr{R}_{\text{det.}} \times \mathscr{R}_{\text{del.}}$ as the lattice ordered monoid of our MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM instances. Finally, we define the feasibility and cost oracles by

$$\begin{aligned}
\rho\left(\left(q^{\text{det.}}, q^{\text{del.}}\right)\right) &= \max\left(\rho_{\text{det.}}\left(q^{\text{det.}}\right), \rho_{\text{del.}}\left(q^{\text{del.}}\right)\right), \\
c\left(\left(q^{\text{det.}}, q^{\text{del.}}\right)\right) &= c_{\text{det.}}\left(q^{\text{det.}}\right),
\end{aligned}$$

to obtain the following lemma.

Lemma 11.2. *The sequence of flight legs p of an o-d path P in the pairing connection graph D is a feasible pairing $p \in \mathscr{P}_{\text{del}}$ if and only if $\rho\left(\left(q_P^{\text{det.}}, q_P^{\text{del.}}\right)\right) = 0$. Besides, if $c\left(\left(q_P^{\text{det.}}, q_P^{\text{del.}}\right)\right) = 0$, we have $c\left(\left(q_P^{\text{det.}}, q_P^{\text{del.}}\right)\right) = c_p$, where c_p is the cost of p.*

Proof. Let P be an o-d path in D, $q_P = (q_P^{\text{det.}}, q_P^{\text{del.}})$ be its resource, and p be the corresponding sequence of flight legs. Then p is a feasible pairing $p \in \mathscr{P}$ if and only if $\rho_{\text{det.}}\left(q_P^{\text{det.}}\right) = 0$, and satisfies the delay constraints if and only if $\rho_{\text{del.}}(q_P^{\text{del.}}) = 0$. Besides, $c(q_P) = c_{\text{det.}}(q_P^{\text{del.}}) = c_p$. □

Given Lemma 11.2, and as $(\mathscr{R}_{\text{det.}}, \oplus_{\text{det.}}, \leqslant_{\text{det.}})$ has already been defined in Chapter 9, it only remains to define $(\mathscr{R}_{\text{del.}}, \oplus_{\text{del.}}, \leqslant_{\text{del.}})$ to finish the reduction of the stochastic pricing subproblem to the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM.

11.2.2 Delay lattice ordered monoid

To model delay random variables in the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework, we need a family of random variables stable by sum and endowed with a lattice order. Both the discrete distributions with finite order endowed with the usual stochastic order or the scenario distributions can be used. See Chapter 5 for more details. In the remaining of this chapter, we suppose that random variable distributions belong to one of these set, and we denote $\leq_{\text{rv}}$ the corresponding order. The specificity of delay model (11.6) comes from the fact that slack makes the sum of two pairing resources non-standard, and it is a little bit technical to design an operator modeling this sum and which is also associative. Therefore, when defining our sum operator, we enhance the specificities of delay computation in the *chronological* direction, as in Equation (11.13), and in the *reverse chronological* direction, as in Equation (11.14).

$$q_P = \big((q_1 \oplus q_2) \oplus q_3\big) \oplus q_4, \tag{11.13}$$
$$= q_1 \oplus \big(q_2 \oplus \big(q_3 \oplus q_4\big)\big). \tag{11.14}$$

We recall for Part I that it is the associativity of $\oplus$ that enables to use bounds in an A* way in resource constrained shortest path algorithms, and that these bounds enable significant speed-ups of the algorithms in the case of stochastic resources. Practically, the algorithms compute paths resources in the chronological direction and bounds resources in the reverse chronological direction.

We now come to the definition of the delay lattice ordered monoid $(\mathscr{R}_{\text{del.}}, \oplus_{\text{del.}}, \leqslant_{\text{del.}})$. Computing the resource of a pairing in the reverse chronological direction requires to sum the resources of the legs at the end of the pairing without knowing those at the beginning of the pairing. As according to delay model (11.6), the delay on a flight leg depends on the previous legs, this leads to a technical difficulty. We therefore have two define three types of resources: *connection resources*, $q^{\text{con.}}$ which contain all the information we need on arcs, *forward resources* $q^{\text{forw.}}$ for pairing resources computed in the chronological direction, and which represent the total delay ξ since the beginning of the duty, and finally *backward resources* $q^{\text{back.}}$ for pairing resources computed in the reverse chronological direction, which are list of connection resources. We then define operators between these resources: a *forward sum operator* $\oplus^{\text{forw.}}$, a *backward sum operator* $\oplus^{\text{back.}}$ for backward sum of resources, and finally a *duty operator* $\oplus_{\text{duty}}$ for the sum of a forward and a backward resource. Similarly, a *connection resource order* $\leqslant^{\text{con.}}$, a *forward resource order* $\leqslant^{\text{forw.}}$ and a *backward resource order* $\leqslant^{\text{back.}}$ are defined. All these elements are then assembled to make the delay lattice ordered monoid $(\mathscr{R}_{\text{del.}}, \oplus_{\text{del.}}, \leqslant_{\text{del.}})$. The remaining of the section is devoted to the definition of these operators and can be skipped by a reader not interested in implementations details.

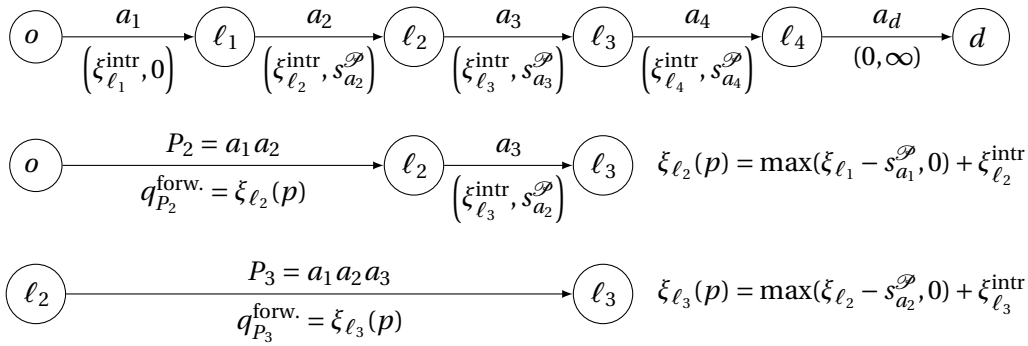


Figure 11.2 – Chronological direction computation of duty delay

Connection resources

Consider the one duty pairing p of path P on Figure 11.2. Each arc corresponds to a feasible connection. Therefore, to compute delay according to model (11.6), we need to store on

each arc a_i the intrinsic delay random variable $\xi^{\text{intr}}_{\ell_{i-1}}$ of its origin arc and the slack $s^{\mathscr{P}}_{a_i}$ of connection a_i. We therefore define the set of *connection resources* $q^{\text{con.}}_a$ to be the set of pairs of a random variable and a real number.

Forward resources for chronological computation of delay

We now explain how to compute delay ξ_ℓ in the chronological direction. Using model (11.6), the total delay ξ_{ℓ_i} on leg i depends only on the delay on $\xi_{\ell_{i-1}}$ and on the resource $q^{\text{con.}}_{a_{i-1}}$ of arc a_{i-1}. As a consequence, to compute delay in the chronological direction, we only have to propagate the total delay. The set of *forward resources* $q^{\text{forw.}}$ is thus defined to be the set of random variables, and we define its order $\preccurlyeq^{\text{forw.}}$ to be the random variable order $\leq_{\text{rv}}$. We finally define the operator $\oplus^{\text{forw.}}$, which to a forward resource $q^{\text{forw.}}_{\ell_i} = \xi_{\ell_i}$ and a connection resource $q^{\text{con.}}_{a_i} = (\xi^{\text{intr}}_{\ell_{i+1}}, s^{\mathscr{P}}_{a_i})$ associates the forward resource $q^{\text{forw.}}_{\ell_i} \oplus^{\text{forw.}} q^{\text{con.}}_{a_i} = q^{\text{forw.}}_{\ell_{i+1}} = \xi_{\ell_{i+1}}$ according to model (11.6).

Backward resources for reverse-chronological computation of delay

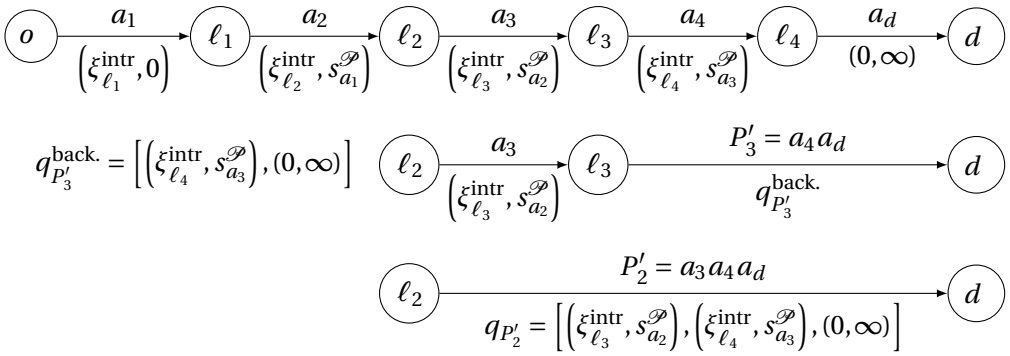


Figure 11.3 – Reverse chronological direction computation of duty delay

Consider now the reverse chronological computation of the resource q_P of the duty pairing p whose path P is illustrated on Figure 11.3. To be able to check the probabilistic delay constraint (11.3), we need again to compute at least once the delay ξ_{ℓ_i} for each leg ℓ_i. According to delay model (11.6), delay ξ_{ℓ_i} on leg ℓ_i depends on the delay on leg ℓ_{i-1}. As a consequence, computing the sum of the resources of arc a_3, a_4, and a_d does not enable to compute ξ_{ℓ_3}. Delay ξ_{ℓ_3} on leg ℓ_3 will be accessible only once the beginning of the pairing has been reached. Therefore the information on the intrinsic delay of each flight leg and on the slack of each connection must be kept until the beginning of the duty has been reached. We therefore define a backward resource $q^{\text{back.}}$ to be a list of connection resources $q^{\text{con.}}$. The *backward sum operator* $\oplus^{\text{back.}}$ operator between two backward resources is simply the concatenation of their respective connection resources lists.

Connection and backward resources orders

As backward resources are lists of connection resources, to compare backward resources, we need to be able to compare connection resources. We therefore define the *connection resource order* $\leqslant^{\text{con.}}$ by

$$q_1^{\text{con.}} = (\xi_1, s_1^{\mathscr{P}}) \leqslant^{\text{con.}} q_2^{\text{con.}} = (\xi_2, s_2^{\mathscr{P}}) \quad \text{if} \quad \xi_1 \leq_{\text{rv}} \xi_2 \quad \text{and} \quad s_1^{\mathscr{P}} \geq s_2^{\mathscr{P}}.$$

Note that a longer slack induces a reduced delay, and thus $s_1^{\mathscr{P}}$ needs to be non smaller than $s_2^{\mathscr{P}}$ for $q_1^{\text{con.}}$ to be non greater than $q_2^{\text{con.}}$. We can now define the *backward resource order* as a term by term connection order.

$$q^{\text{back.}} \preceq \widetilde{q^{\text{back.}}} \quad \text{if} \quad \begin{cases} i \leq \tilde{\imath}, \\ q_j^{\text{con.}} \preceq \widetilde{q_j^{\text{con.}}}, \ \forall j \leq i, \end{cases} \quad \text{where} \quad \begin{cases} q^{\text{back.}} &= \left[q_1^{\text{con.}}, \dots q_i^{\text{con.}}\right], \\ \widetilde{q^{\text{back.}}} &= \left[\widetilde{q_1^{\text{con.}}}, \dots \widetilde{q_{\tilde{\imath}}^{\text{con.}}}\right]. \end{cases}$$

Duty operator for sum of a forward resource with a backward resource

It remains to explain how the list of a backward duty resource $q^{\text{back.}}$ is dealt with when reaching the beginning of a duty. We therefore define the *duty sum operator* $\oplus_{\text{duty}}$ between a forward resource $q^{\text{forw.}}$ and a backward resource. Given a forward resource $q^{\text{forw.}} = \xi$ and a backward resource $q^{\text{back.}} = [q_1^{\text{con.}}, \dots, q_k^{\text{con.}}]$, operator $\oplus_{\text{duty}}$ computes iteratively the delay on the leg after each connection in the connection list of $q^{\text{back.}}$, and returns the delay on the leg after the last connection in $q^{\text{back.}}$ if constraint (11.3) is satisfied after each of the connections in $q^{\text{back.}}$, and $+\infty$ otherwise. Operator $\oplus_{\text{duty}}$ corresponds to recursive applications of the forward duty sum operator $\oplus^{\text{forw.}}$.

$$q^{\text{forw.}} \oplus_{\text{duty}} q^{\text{back.}} = \left(\left(\left(q^{\text{forw.}} \oplus^{\text{forw.}} q_1^{\text{con.}}\right) \oplus^{\text{forw.}} \dots\right) \oplus^{\text{forw.}} q_k^{\text{con.}}\right).$$

Delay lattice ordered monoid

We now explain how to combine the connection, forward, and backward resources to obtain the delay lattice ordered monoid. We start with the case of *inter-duty connections*. As delay does not propagate on nights, these inter-duty connections can be dealt with connection resources $q^{\text{con.}}$ whose slacks are infinite. We now come to the definition of *pairing delay resources* $q^{\text{del.}}$, which are resources for partial pairings. There are two types of partial pairings. Partial pairings of the first type or *middle duty partial pairings* are composed of a single partial duty, as illustrated on Figure 11.4. As such partial pairings are simply lists of connections, they can be represented by a backward resource $q^{\text{back.}}$.

Partial pairings of the second type or *multi-duty partial pairings* are composed of several duties. As illustrated on Figure 11.5, a multi-duty partial pairing starts by the end of a partial duty and ends by the beginning of a partial duty. The resource q_P of such a partial pairing path P is thus encoded by a pair $(q_P^{\text{back.}}, q_P^{\text{forw.}})$, where $q_P^{\text{back.}}$ is the resource of the end of the first duty of the pairing and $q_P^{\text{forw.}}$ is the resource of the beginning of the last duty of the pairing.

As middle duty partial pairings and multi-duty partial pairings use respectively backward resources and multiple duty resources, we can define the delay resource set $\mathscr{R}_{\text{del.}}$ as the union

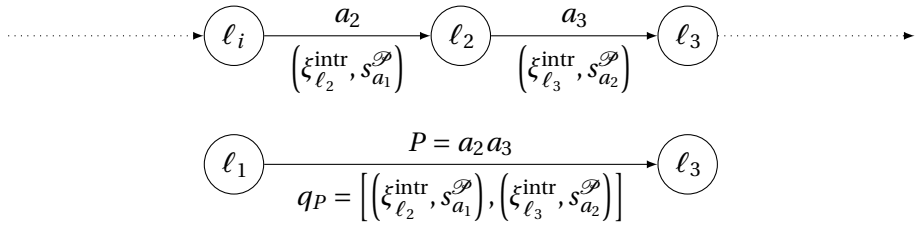


Figure 11.4 – A "middle duty" partial pairing.

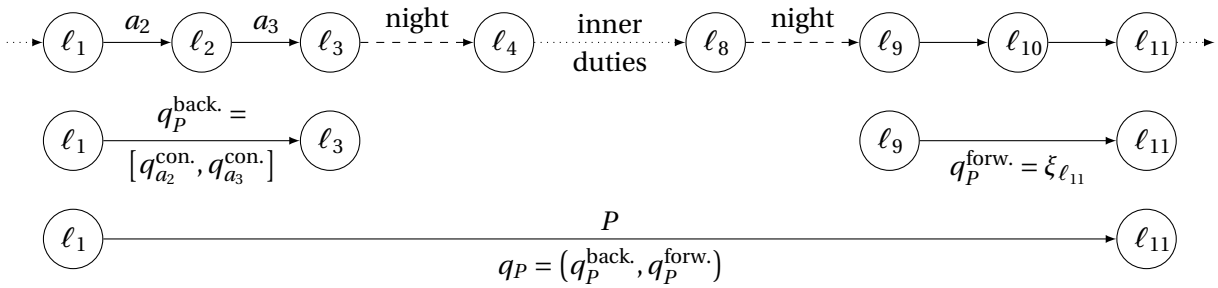


Figure 11.5 – Multiple duties partial pairing

of the backward resources set and the multiple duty resource set. The *pairing resource delay sum operator* $\oplus_{\text{del.}}$ and the *pairing resource delay order* $\leqslant_{\text{del.}}$ are defined by disjunction of cases using the sum operators and orders of the previous sections: Table 11.1 defines $\oplus_{\text{del.}}$, and Table 11.2 defines $\leqslant_{\text{del.}}$.

Proposition 11.3. $(\mathscr{R}_{\text{del.}}, \oplus_{\text{del.}}, \leqslant_{\text{del.}})$ *is a lattice ordered monoid. Besides, given the definitions of* $(\mathscr{R}_{\text{del.}}, \oplus_{\text{del.}}, \leqslant_{\text{del.}})$ *and* $\rho_{\text{del.}}$*, Property* (11.12) *is satisfied for all o-d paths* P *in* D*.*

Proof. Operator $\oplus^{\text{back.}}$ is associative by associativity of list concatenation. The associativity of sequence $q^{\text{forw.}} \oplus^{\text{forw.}} q^{\text{back.}} \oplus^{\text{back.}} \widetilde{q^{\text{back.}}}$ follows from the associativity of the random variable sum, the associativity of $\oplus^{\text{back.}}$, and the fact that forward resources are always summed in the chronological direction.

q	$\tilde{q}$	$q \oplus_{\text{del.}} \tilde{q}$
$q^{\text{back.}}$	$\widetilde{q^{\text{back.}}}$	$q^{\text{back.}} \oplus^{\text{back.}} \widetilde{q^{\text{back.}}}$
$q^{\text{back.}}$	$\left(\widetilde{q^{\text{back.}}}, \widetilde{q^{\text{forw.}}}\right)$	$\left(q^{\text{back.}} \oplus^{\text{back.}} \widetilde{q^{\text{back.}}}, \widetilde{q^{\text{forw.}}}\right)$
$\left(q^{\text{back.}}, q^{\text{forw.}}\right)$	$\widetilde{q^{\text{back.}}}$	$\left(q^{\text{back.}}, q^{\text{forw.}} \oplus_{\text{duty}} \widetilde{q^{\text{back.}}}\right)$
$\left(q^{\text{back.}}, q^{\text{forw.}}\right)$	$\left(\widetilde{q^{\text{back.}}}, \widetilde{q^{\text{forw.}}}\right)$	$\begin{cases} \infty & \text{if } q^{\text{forw.}} \oplus_{\text{duty}} \widetilde{q^{\text{back.}}} = \infty \\ \left(q^{\text{back.}}, \widetilde{q^{\text{forw.}}}\right) & \text{otherwise.} \end{cases}$

Table 11.1 – Definition of $\oplus_{\text{del.}}$

q	$\tilde{q}$	$q \leqslant_{\text{del.}} \tilde{q}$ if
$q^{\text{back.}}$	$\widetilde{q^{\text{back.}}}$	$q^{\text{back.}} \leqslant^{\text{back.}} \widetilde{q^{\text{back.}}}$
$q^{\text{back.}}$	$\left(\widetilde{q^{\text{back.}}}, \widetilde{q^{\text{forw.}}}\right)$	$q^{\text{back.}} \leqslant^{\text{back.}} \widetilde{q^{\text{back.}}}$
$\left(q^{\text{back.}}, q^{\text{forw.}}\right)$	$\widetilde{q^{\text{back.}}}$	$q^{\text{back.}} \leqslant^{\text{back.}} \widetilde{q^{\text{back.}}}$ and $q^{\text{forw.}} = 0$
$\left(q^{\text{back.}}, q^{\text{forw.}}\right)$	$\left(\widetilde{q^{\text{back.}}}, \widetilde{q^{\text{forw.}}}\right)$	$q^{\text{back.}} \leqslant^{\text{back.}} \widetilde{q^{\text{back.}}}$ and $q^{\text{forw.}} \leqslant^{\text{forw.}} \widetilde{q^{\text{forw.}}}$

Table 11.2 – Definition of $\leqslant_{\text{del.}}$

The monotonicity of the $\oplus_{\text{del.}}$ translations comes from the monotonicity of $\oplus^{\text{forw.}}$ left and right translations with respect to $\leqslant^{\text{forw.}}$ and $\leqslant^{\text{back.}}$, which themselves comes from the monotonicity of the sum of a real or of a random variable, and the monotonicity of the positive part with respect to stochastic orders. □

11.2.3 Algorithms

As it has been modeled in the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework, any algorithm of Section 4.1 can solve it. Nonetheless, as we explained in Chapter 5, the dominance is rare when considering random variable resources, and thus dominance tests only slow the algorithm without allowing to cut paths. Thus, the generalized A* algorithm is more appropriate than the label correcting algorithm to solve the stochastic subproblem. Besides, it is interesting to use a state graph to enhance the quality of the bounds used in the algorithms. As the problem is highly non linear, there is no natural linear weight function which would enable to use the conditional state graphs of Section 6.3. We therefore use the clustering state graphs of Section 6.2.

11.2.4 Numerical results

We test our column generation approach to the STOCHASTIC CREW PAIRING PROBLEM on the instances of Chapter 7. Following Lan et al. [112], we use truncated lognormal distributions to model intrinsic delay random variables ξ_ℓ^{intr}. These distribution are fitted on historical data. We increase delay on these historical data to enhance the delay propagation effect. We use the generalized A* and the label correcting algorithm with bounds provided by the clustering state graph. We choose the maximum number of states κ per vertex of the connection graph based on the performance on the deterministic problem in Table 9.2. In delay constraint (11.3), we use 30 minutes as threshold α, and thresholds β equal to 5%, 10%, and 15%. We use the same implementations of discrete distributions with finite support as in Chapter 5. The numerical experiments are performed on a server with 128 Gb of ram and 12 cores at 2.4 GHz. The algorithms are not parallelized.

Table 11.3 provides the results on the different instances. The first columns provide the instance considered, the threshold β, the algorithm used, and the maximum number κ of state vertices per connection graph vertex. The next four columns provide the number of iterations of the column generation, the percentage of time spent solving the pricing subproblem, the

Instance	β (min)	Alg.	κ	CG iter.	Pricing time	Avg. paths	Cut Dom.	MIP time	Add. Cost	Total time (hh:mm:ss)
CP50	5%	A*	10	67	93.23%	1.730e+03	–	0.253%	138.01%	00:00:55.5
CP50	10%	A*	10	78	92.34%	1.741e+03	–	0.161%	72.65%	00:01:02.0
CP50	15%	A*	10	94	93.34%	3.029e+03	–	0.243%	0.00%	00:02:34.1
CP50	5%	cor.	10	54	94.53%	1.903e+03	0.33%	0.232%	138.01%	00:01:02.2
CP50	10%	cor.	10	62	94.75%	1.846e+03	0.30%	0.146%	72.65%	00:01:07.9
CP50	15%	cor.	10	97	95.53%	2.976e+03	0.27%	0.083%	0.00%	00:03:32.8
CP70	5%	A*	10	125	95.62%	1.172e+04	–	0.118%	57.64%	00:06:10.2
CP70	10%	A*	10	150	95.11%	1.059e+04	–	0.756%	53.04%	00:06:28.1
CP70	15%	A*	10	150	95.56%	1.822e+04	–	0.066%	0.00%	00:10:33.0
CP70	5%	cor.	10	121	97.20%	1.150e+04	0.49%	0.069%	57.64%	00:09:37.6
CP70	10%	cor.	10	140	97.46%	1.123e+04	0.45%	0.686%	53.07%	00:12:12.3
CP70	15%	cor.	10	145	98.30%	1.562e+04	0.31%	0.026%	0.00%	00:23:01.4
CP90	5%	A*	30	218	98.66%	7.928e+04	–	0.016%	45.57%	01:53:20.0
CP90	10%	A*	30	236	98.93%	8.701e+04	–	0.053%	41.22%	02:23:22.6
CP90	15%	A*	30	295	98.95%	1.324e+05	–	0.009%	0.00%	04:16:44.0
A318	5%	A*	150	341	99.76%	3.888e+05	–	0.002%	8.30%	57:08:59.0
A318	10%	A*	150	381	99.74%	4.342e+05	–	0.001%	7.32%	70:00:06.8
A318	15%	A*	150	395	99.77%	4.783e+05	–	0.001%	0.00%	94:32:26.5

Table 11.3 – Numerical results of our column generation approach to the STOCHASTIC CREW PAIRING PROBLEM.

average number of paths enumerated by the subproblem algorithm, and the proportion of paths cut by the dominance test when the label correcting algorithm is used. We then provide the percentage of the total CPU time spent solving the mixed integer program at the end of the column generation. The column "Add. Cost" provides the additional cost when compared to the deterministic CREW PAIRING PROBLEM, and the last column gives the total CPU time.

We remark that when $\beta = 15\%$, the constraints are not active and we obtain the non-constrained solution. On small instances, these extra-costs are very large as many additional pairings are required. These extra-cost decrease with the size of the instance, as more alternative pairings become available. The number of iterations of the column generation decreases with the strength of the constraint. This seems logical as the number of feasible pairings also decreases with the constraint strength. Concerning the choice of the algorithm, Table 11.3 confirms our statement of Part I that the dominance test is not useful in stochastic context. The number of paths enumerated by the generalized A* algorithm is larger but of the same order of magnitude as the one obtained in the deterministic case. Nonetheless, as convolutions and stochastic order dominance tests are time consuming, the time needed to solve the problem is much larger than in the deterministic case, and the time needed to solve the column generation linear programs and the final integer program become small compared to the pricing time.

11.3 Compact integer program for Stochastic Aircraft Routing

We introduced in Chapter 8 the compact integer program (8.2) to model the deterministic AIRCRAFT ROUTING PROBLEM (7.1). The objective of this section is to extend this approach to the STOCHASTIC AIRCRAFT ROUTING PROBLEM (11.8), where delay is propagated by crews and airplanes according to delay model (11.2). We were able to extend it only in the case of scenario based distributions.

11.3.1 Deterministic problem compact integer program

We use the *routing connection digraph* $D^R = (V^{\mathscr{R}}, A^R)$ and its *maintenance state graph* $\mathscr{D} = (\mathscr{V}, \mathscr{A})$, which have both been introduced in Chapter 8. In a routing connection digraph, there is one vertex v in $V^{\mathscr{R}}$ for each leg ℓ in $\mathscr{L}$, and one arc a in A^R for each feasible routing connection. $\mathscr{V}$ is the set of *state vertices* ϑ and $\mathscr{A}$ is the set of *state arcs* α. Each vertex v in the routing connection graph has a set of state vertices $\mathscr{V}_v$, and each arc a has a set of state arcs $\mathscr{A}_a$. Chapter 8 introduces the integer program (8.2), that we now restate, ans shows that it encodes the deterministic AIRCRAFT ROUTING PROBLEM (7.1).

$$\begin{aligned}
\sum_{\substack{\vartheta \in \mathscr{V}_v \\ \alpha \in \delta^-(\vartheta)}} x_\alpha &= 1 && \forall v \in V \setminus S \\
\sum_{\substack{\vartheta \in \mathscr{V}_v \\ \alpha \in \delta^+(\vartheta)}} x_\alpha &= 1 && \forall v \in S \\
\sum_{\alpha \in \delta^-(\vartheta)} x_\alpha &= \sum_{\alpha \in \delta^+(\vartheta)} x_\alpha && \forall \vartheta \in \mathscr{V} \\
x_\alpha &\in \{0,1\} && \forall \alpha \in \mathscr{A}
\end{aligned} \tag{8.2}$$

For the use of this chapter, we only need to know that, given a connection and its corresponding arc a, this connection belongs to one pairing of the solution of (8.2) if and only if there is one state arc α in $\mathscr{A}_a$ such that $x_\alpha = 1$.

11.3.2 Scenario approach compact integer program

We now suppose to have a set of scenarios $\omega \in \Omega$. We denote respectively $\xi^{\text{intr}}_{v\omega}$ and $\xi_{v\omega}$ the intrinsic and the total delay of the leg ℓ of vertex v under scenario ω.

The following equations enable to take into account delay propagated by crews according to model (11.2).

$$\begin{aligned}
\xi^{\text{prop}}_{v\omega} &\geq 0 \\
\xi^{\text{prop}}_{v\omega} &\geq \xi_{u\omega} - s^{\mathscr{P}}_{uv}, && \forall (u,v) \in \mathbf{y} \\
\xi_{v\omega} &= \xi^{\text{prop}}_{v\omega} + \xi^{\text{intr}}_{v\omega}
\end{aligned} \tag{11.15}$$

where $(u,v) \in \mathbf{y}$ means that (u,v) belongs to a pairing p such that $y_p = 1$ in $\mathbf{y}$. Let M be a large constant. The following equation enables to take into account delay propagated by airplanes

according to model (11.2).

$$\xi^{\text{prop}}_{v\omega} \geq \xi_{u\omega} - s^{\mathscr{R}}_{uv} - (1 - x_{uv})M, \quad \forall (u,v) \in A \tag{11.16}$$

In order to take into account maintenance constraints, we have go to the state graph formulation to obtain the following program.

$$\min \sum_{\omega} \sum_{v \in V} \xi_{v\omega} \tag{11.17a}$$

$$\text{s.t.} \sum_{\substack{\vartheta \in \mathscr{V}_v \\ \alpha \in \delta^-(\vartheta)}} x_\alpha = 1 \qquad \forall v \in V \backslash S \tag{11.17b}$$

$$\sum_{\substack{\vartheta \in \mathscr{V}_v \\ \alpha \in \delta^+(\vartheta)}} x_\alpha = 1 \qquad \forall v \in S \tag{11.17c}$$

$$\sum_{\vartheta \in \delta^-(\vartheta)} x_\alpha = \sum_{\vartheta \in \delta^+(\vartheta)} x_\alpha \qquad \forall \vartheta \in \mathscr{V} \tag{11.17d}$$

$$\xi^{\text{prop}}_{v\omega} \geq \xi_{u\omega} - s^{\mathscr{P}}_{uv}, \qquad , \forall (u,v) \in \mathbf{y}, \forall \omega \tag{11.17e}$$

$$\xi^{\text{prop}}_{v\omega} \geq \xi_{u\omega} - s^{\mathscr{R}}_{uv} - (1 - \sum_{\alpha \in \mathscr{A}_{(u,v)}} x_\alpha)M, \qquad \forall (u,v) \in A, \forall \omega \tag{11.17f}$$

$$\xi_{v\omega} = \xi^{\text{prop}}_{v\omega} + \xi^{\text{intr}}_{v\omega} \qquad \forall v \in V, \forall \omega \tag{11.17g}$$

$$\xi^{\text{prop}}_{v\omega} \geq 0, \xi_{v\omega} \in \mathbb{R} \qquad \forall v \in V, \forall \omega \tag{11.17h}$$

$$x_\alpha \in \{0,1\} \qquad \forall \alpha \in \mathscr{A} \tag{11.17i}$$

Equations (11.17b) to (11.17d) ensure that the vector of x_α defines a solution of the deterministic AIRCRAFT ROUTING PROBLEM. Equations (11.17e) to (11.17h) ensure that $\xi_{v\omega}$ corresponds to the delay of the leg of vertex v under delay model (11.2).

Theorem 11.4. *An optimal solution of* (11.17) *defines an optimal solution of the* STOCHASTIC AIRCRAFT ROUTING PROBLEM (11.8) *when random variables* ξ^{intr}_v *are supposed to have scenario distributions.*

This compact integer program for robust AIRCRAFT ROUTING PROBLEM in inspired from the scenario approach to robust CREW PAIRING PROBLEM developed by Yen and Birge [174]. But contrary to theirs, our approach leads to a compact integer program that can be solved exactly by a standard MILP solver.

11.3.3 Numerical experiments

We have tested the performance of the STOCHASTIC AIRCRAFT ROUTING PROBLEM compact integer program on the instances of Chapter 7. The intrinsic delay random variables ξ^{intr}_ℓ scenarios are sampled from the truncated lognormal distributions of Section 11.2.4. In delay constraint (11.3), we use 30 minutes as threshold α, and 10% as threshold β. The numerical experiments are performed on a server with 128 Gb of ram and 12 cores at 2.4 GHz. We use

CPLEX 12.1.0 to solve the integer program.

Table 11.4 shows the compact integer program size and the computation time we obtain on the aircraft routing instances of Chapter 7 using between 10 and 200 scenarios. For each instance and number of scenarios, it provides the number of variables and the number of constraints in the integer program, and the CPU time needed to solve it. We note that, even for the largest instances with 200 scenarios, the compact integer program can be solved in less than 10 minutes.

11.4 Numerical results on STOCHASTIC INTEGRATED PROBLEM

We test the solution approach to the STOCHASTIC INTEGRATED PROBLEM in Section 11.1.3 on the small and medium size instances of Chapter 7. The solution schemes of Section 11.3.3 and Section 11.2.4 are respectively used to solve the STOCHASTIC AIRCRAFT ROUTING PROBLEM and STOCHASTIC CREW PAIRING PROBLEM instances. To model delay in the crew pairing part of the integrated problem solution scheme, we use the discrete independent distributions of Section 11.2.4. And we use the scenario based distributions of Section 11.3.3 in the aircraft routing part of the solution scheme. The numerical experiments are performed on a server with 128 Gb of ram and 12 cores at 2.4 GHz. The algorithms are not parallelized. In delay constraint (11.3), we use 30 minutes as threshold α, and 10% as threshold β. Following the heuristic approach at the end of Section 11.1.3, we use $\gamma = 0.9$ in the short connect constraint.

Table 11.5 provides numerical results on the STOCHASTIC INTEGRATED PROBLEM for different number of scenarios. The two first columns provide the instance and the number of scenarios. The next column gives the number of cuts added, i.e. the number of iterations of the INTEGRATED PROBLEM. The four next columns correspond to the crew pairing instances solved: they provide the maximum number κ of state vertices per connection graph vertex used in the clustering state graph, the total number of iterations in all the column generation schemes launched along the algorithm, and the percentage of the total algorithm time spent in the column generation, and the percentage of time spent solving the integer version the crew pairing problem. The next column gives the percentage of time spent solving the AIRCRAFT ROUTING PROBLEM compact integer program. The next columns provide the number of short connections in the solution returned, and the gap between the optimal solution of the linear program without short connect constraints and the solution returned. Finally, the last column provides the total CPU time.

We note that the CPU time needed to solve these STOCHASTIC INTEGRATED PROBLEM instances is between five and ten times larger than for their deterministic counterpart in Table 10.1. Besides, the gap being very small, we can see that using the heuristic with $\gamma = 0.9$ does not penalize the quality of the solution returned.

11.5 Sampling approach

The solution approach to the STOCHASTIC AIRCRAFT ROUTING PROBLEM in Section 11.3, and thus the solution approach to the STOCHASTIC INTEGRATED PROBLEM in Section 11.1, work only for scenario-based distributions. Suppose now that the intrinsic delay (ξ^{intr}) is a family

Instance	Scenarios	Variables	Constraints	Total time (mm:ss)
AR4	10	13377	23791	00:01.33
	20	17297	46821	00:01.68
	50	29057	115911	00:04.05
	100	48657	231061	00:06.69
	200	87857	461361	00:15.55
AR8	10	44001	92466	00:07.42
	20	51341	183366	00:09.17
	50	73361	456066	00:13.44
	100	110061	910566	00:21.79
	200	183461	1819566	00:37.01
AR12	10	90873	202991	00:28.70
	20	101433	403631	00:31.78
	50	133113	1005551	00:41.81
	100	185913	2008751	01:01.59
	200	291513	4015151	01:37.37
A318	10	174967	406256	01:35.95
	20	189507	809166	01:49.89
	50	233127	2017896	02:24.79
	100	305827	4032446	02:41.03
	200	451227	8061546	04:27.61
A319	10	208311	474246	02:25.99
	20	229211	943706	02:33.03
	50	291911	2352086	02:51.56
	100	396411	4699386	03:50.07
	200	605411	9393986	06:16.80
A320	10	222714	511181	04:26.96
	20	242954	1017771	06:07.85
	50	303674	2537541	06:17.05
	100	404874	5070491	05:42.66
	200	607274	10136391	08:04.82
A321	10	135678	307041	01:37.21
	20	152358	610191	01:28.60
	50	202398	1519641	01:46.61
	100	285798	3035391	02:11.53
	200	452598	6066891	03:01.77

Table 11.4 – Numerical results for the compact integer program approach to the STOCHASTIC AIRCRAFT ROUTING PROBLEM

Instance	Scen.	Cuts	κ	CG it. total	CP CG time	CP MIP time	AR time	Sho. Con.	Gap	Total time (hh:mm:ss)
AR4	10	23	20	40	9.98%	49.38%	40.65%	63	0.031%	00:00:48.9
	20	32	20	47	5.27%	51.47%	43.26%	58	0.034%	00:01:47.9
	50	33	20	52	4.98%	51.07%	43.95%	60	0.034%	00:01:56.7
AR8	10	30	20	130	42.88%	30.73%	26.39%	141	0.003%	00:14:50.7
	20	70	20	162	28.06%	39.98%	31.96%	138	0.005%	00:28:12.2
	50	34	20	135	22.91%	40.13%	36.96%	141	0.003%	00:23:04.7
AR12	10	127	20	349	71.62%	18.39%	9.99%	216	0.006%	04:04:51.1
	20	108	20	341	68.03%	18.12%	13.85%	217	0.007%	03:59:10.7
	50	134	20	357	70.94%	18.98%	10.08%	215	0.005%	04:15:35.2

Table 11.5 – Numeric results on STOCHASTIC INTEGRATED PROBLEM

of generic random variables on a probability space $(\Omega, \mathscr{F}, \mu)$. As we mentioned in Chapter 2, given an N-sample $(\xi^{\text{intr}})^1, \dots, (\xi^{\text{intr}})^N$ of these random variables, we can obtain scenario based distributions by replacing μ by its Monte-Carlo approximation $\widehat{\mu}^N = \frac{\sum_{i=1}^N \delta_{(\xi^{\text{intr}})^i}}{N}$, where $\delta_{(\xi^{\text{intr}})^i}$ is the Dirac in $(\xi^{\text{intr}})^i$. In this section, we suppose that under μ, intrinsic delay (ξ^{intr}) has support in $\mathbb{Z}_+$ and admits a square exponential moment

$$\varepsilon(\mu) = \int_{\mathbb{R}^{\mathscr{L}}} e^{|(\xi^{\text{intr}})|^2} \mu(d\xi^{\text{intr}}).$$

We show that, under these hypotheses, if we replace the intractable constraint

$$\mathbb{P}_\mu \left[\xi_\ell(\mathbf{x}, \mathbf{y}) \geq \alpha \right] \leq \beta, \quad \forall \ell \in \mathscr{L}, \tag{11.3}$$

by the tractable scenario constraint

$$\mathbb{P}_{\widehat{\mu}^N} \left[\xi_\ell(\mathbf{x}, \mathbf{y}) \geq \alpha \right] \leq \beta', \quad \forall \ell \in \mathscr{L}, \tag{11.18}$$

where $\beta' < \beta$, then we have an upper bound exponential in $|\mathscr{L}|$ and $|N|$ on the probability that a solution $(\mathbf{x}, \mathbf{y})$ satisfies the intractable constraint (11.3).

Proposition 11.5. *Let* $(\mathbf{x}, \mathbf{y})$ *be a solution of* (11.4) *where the delay constraint* (11.3) *has been replaced by the sampled one* (11.18) *with* $t = \beta - \beta'$. *The probability that* $(\mathbf{x}, \mathbf{y})$ *is not a solution of* (11.4) *is non-greater than*

$$C\left(\exp\left(-cNt^{|\mathscr{L}|}\right)\mathbf{1}_{t\leq 1} + \exp\left(-cNt^2\right)\mathbf{1}_{t>1}\right),$$

where C *and* c *are constants depending only on* $|\mathscr{L}|$ *and* $\varepsilon(\mu)$, *and* $\mathbf{1}_{\cdot\leq 1}$ *and* $\mathbf{1}_{\cdot>1}$ *are the indicator functions of* $(-\infty, 1]$ *and* $(1, +\infty)$.

The probability in (11.5) is with respect to the sampling of (ξ^{intr}). An alternative bound can be derived when μ has continuous support but bounded density.

Proof. Proposition 11.5 is a direct application of Theorems 2.4 and 2.7 as the function that maps intrinsic delay ξ^{intr} to total delay ξ in delay model (11.2) is Lipschitz. □

Using again Theorems 2.4 and 2.7, a similar result can be obtained for problem (11.9).

11.6 Bibliographical remarks

11.6.1 Delay models

There are three types of delay models in the approaches to the STOCHASTIC AIRCRAFT ROUTING PROBLEM, the STOCHASTIC CREW PAIRING PROBLEM, and the STOCHASTIC INTEGRATED PROBLEM. The first type of delay model was developed for the STOCHASTIC AIRCRAFT ROUTING PROBLEM [112]. It relies on the notion of intrinsic ξ^{intr}_ℓ, and considers *only delay propagated by airplanes*. Total delay is thus modeled as follows.

$$\xi_\ell(\mathbf{0},\mathbf{x}) = \max\left(0, \xi_{\pi_\ell(\mathbf{x})} - s^{\mathscr{P}}_{(\pi_\ell(\mathbf{x}),\ell)}\right) = \xi_\ell(r_\ell(\mathbf{x})) \tag{11.19}$$

where $r_\ell(\mathbf{x})$ is the route of leg ℓ in routing solution $\mathbf{x}$. Our pairing delay model (11.6) is an analogue of model (11.19) where delay is propagated only by crews. As we mentioned in Section 11.1.4, the main strength of such models is their simplicity that enables to tackle with delay in the subproblem when a column generation approach is used.

Our delay model (11.2) belongs to the second type of models, where intrinsic delay ξ^{intr}_ℓ appears on leg and is propagated by airplanes and crews. If such models are more realistic than the previous ones, they suffer from the intractability of the exact computation of probabilities, as we already explained in Section 11.1.1. Approaches using this model therefore rely on averaged model [62], approximate heuristic inference [171], or simulation [7, 63].

Finally, richer models takes into account more accurately the origin of delay, modeling precise events in the airports such as runway queue, passengers management system failure, and precise events during the flying time such as events linked to air control. As a consequence flight legs delay variables are heavily dependent, and exact inference is definitely intractable. Therefore, these models rely on simulation. Several discrete event simulators have been developed [40, 150], the most well known being SIM-AIR [150].

In the context of robust models, only the support of the random variable ξ^{intr}_ℓ modeling delay on flight leg ℓ needs to be provided. Nonetheless, this simple model does not take into account dependence between random variables, and may lead to poor results. Therefore, Yan and Kung [173] suppose that the vector of delays on all legs belongs to a given oriented ellipsoid, which is the robust equivalent to a normal distribution assumption in stochastic models. The orientation matrix of the ellipsoid corresponds to the covariance matrix in the normal distribution model and captures the dependencies between flight legs delays. Besides, it can be estimated as the covariance matrix of the observed delay. The strength of this model is its ability to capture dependencies between random variables while avoiding complicated inference problems. Besides, even if it was developed for non-integrated aircraft routing, it can be easily used in the integrated aircraft routing and crew pairing context. The limit of the

model lies in the choice of normal distributions, which is questionable when modeling delays.

11.6.2 Optimization approaches

STOCHASTIC AIRCRAFT ROUTING PROBLEM approaches

In their seminal paper, Lan et al. [112] state delay model (11.19) and consider the problem of finding the aircraft routing solution that minimizes the sum of the expected delay on all flight legs. They also develop an approximate column generation procedure to solve this problem. This procedure is approximate because of an approximate solution of the pricing subproblem. It could be made exact by using the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM approach and the delay monoid developed in Section 11.2. They tested their approach on a 278 flight legs network.

Yan and Kung [173] develop a robust optimization approach based on the robust model mentioned in the previous section. The robust model is solved using column and row generation. Column generation is used to generate routes, while row generation is used to generate constraints. Indeed, Yan and Kung [173] show that only a finite number of the infinite set of constraints required to model an ellipsoidal feasibility set in linear programming is required, and that these constraints can be identified efficiently. Yan and Kung [173]'s approach is the most convincing one already implemented that takes into account dependencies between flight leg delays.

In one of the first attempt to take into account uncertainty in aircraft routing, Ageeva and Clarke [6] use a robust approach. They define an interval of time for flight leg departure depending of a parameter δT and study the influence of this robustness parameter on the cost of the solution.

Other approaches to robust aircraft routing focus on schedule tuning: the idea is to have some flexibility on flight legs departure time in order to assign optimally the slack between flight legs in order to absorb delay. Wu [172] defines a robustness criterion based on scenario generations by a simulator. His solution scheme is a sequential algorithm that alternates aircraft routing optimization and schedule tuning in order to maximize robustness. He mentions that improved result can be obtained by using a richer model such as MEANS [40]. Aloulou et al. [10] develop a similar approach based on a simple delay model. Finally, Maher et al. [122] define a robustness criterion that takes into account maintenance disruption costs due to schedule disruptions and delay. Maintenance disruption costs come from the maintenance process: each aircraft must undergo a maintenance check every 6 nights. Due to schedule adaptation caused by delay, some airplanes may not reach the airport where their maintenances were scheduled, which may lead to a violation of the maintenance rule. The idea of this robust optimization is to minimize the risk of violating the maintenance rule. They implement a two stage model based on a heuristic and a simple stochastic model.

STOCHASTIC CREW PAIRING PROBLEM approaches

In the context of the STOCHASTIC CREW PAIRING PROBLEM, the objective considered by most contributions is to minimize the extra crew pairing costs due to disruptions caused by delay.

These sources of cost are the same as the ones of the deterministic problem: hotel nights, credited hours, and so on. Ehrgott and Ryan [64] develop a heuristic robustness criterion and incorporate it in a bicriteria optimization problem.

Recent approaches minimize the expected costs in a two stage optimization problem, the first stage being crew pairing and the second stage being a disruption management system. Yen and Birge [174] develop an integer programming approach based on sampled scenarios. They treat instances up to 11 planes and 79 legs. The approach of Schaefer et al. [158] is one of the most promising. They use Sim-Air [150] to model the costs of delay. They develop matheuristics to solve the problem. These heuristics rely on Monte Carlo simulation to approximate the average costs. They treat instances with up to 120 legs.

Shebalov and Klabjan [162] deal with uncertainty in another way. Instead of using a stochastic model, they define a robustness criterion in order to maximize the possibility to switch pairings between crews in case of disruptions. They deal with instances with up to 228 legs.

Finally, Muter et al. [134] seek a different objective: for operational reason, the company they work for needs to incorporate extra-legs after crew pairing optimization. The stochasticity is thus in the distribution of these extra-legs. They develop a two-stage model to solve this problem.

STOCHASTIC INTEGRATED PROBLEM approaches

The approaches to the robust integrated aircraft routing and crew pairing problem available in the literature are extensions of the seminal robust aircraft routing approach of [112]. They use similar column generation solution schemes. Weide et al. [171] introduce (11.2), and use an approximate inference scheme based on convolution product of distributions to derive delay distributions. This enables to define approximate expected costs of delay, which are then introduced in an optimization matheuristic. They test their solutions on schedules with up to 750 flight legs. Using the same stochastic model, Dunbar et al. [62] define a bicriteria optimization problem. The results of these two heuristic approaches must be taken with care because of the approximate nature of inference performed. Dunbar et al. [63] enhance these by replacing the approximate inference variational scheme by scenario sampling. Dunbar et al. [62] and Dunbar et al. [63] use a network with 54 flight legs.

12 Conclusion

As a conclusion, we sum up the main contributions of this dissertation, and outline some research directions suggested by our results.

12.1 Main contributions

In this dissertation, we have developed algorithms for some shortest path and airline operations problems. The first part is mainly theoretical and introduces a methodology to solve constrained shortest path problems through the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework. The second part applies this framework to solve some airline operations problems in their full industrial complexity.

Part I introduces an algebraic framework for resource constrained shortest path problem, the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. This framework is at the crossroad of two distinct branches of the literature on path problem: the one that develops solution methods for resource constrained shortest path problems, and the one that studies algebraic modeling of shortest path problems.

Most contributions in the resource constrained shortest path community rely on the resource extension functions framework [55, 96, 97]. In this non-algebraic framework, each path P has a resource q_P, each arc a a resource extension function ζ_a, and the resource of P followed by arc a is $\zeta_a(q_P)$. Enumeration algorithms are used to solve these problems: a dominance relation is used to discard partial solutions in a smart enumeration of all the paths. It is well known that lower bounds on paths resources enable to speed-up enumeration algorithms [61] in two ways. They are used to guide the search, which enables to find faster good solutions, and to discard partial solutions, which reduces the number of paths enumerated. However, if the literature contains ad-hoc procedures to compute lower bounds on some specific problems, there is no standard algorithm to build such lower bounds, and the resource extension function framework does not provide standard way of using them in enumeration algorithms. Our MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework provides standard procedures to build such bounds, and new versions of the enumeration algorithms which use these bounds. The numerical experiments show that the use of these bounds practically speeds-up the resolution of a wide range of constrained shortest path problems, among which stochastic, non-linear, and industrial problems for which no bounding techniques existed.

Our procedures to build lower bounds are in the line of a long tradition of algebraic frameworks for path problems [8, 17, 34, 74, 115, 130, 152, 178]. In this tradition, arcs a do not have resource extension functions ζ_a but resources q_a that belong to the same set $\mathscr{R}$ as the paths resources q_P. The resource set $\mathscr{R}$ is endowed with an algebraic structure, that is generally the one of idempotent semiring. The canonical ordering $\leqslant$ associated to the idempotent binary operator of the semiring plays the role of the dominance relation of the resource extension function framework. We denote $\oplus$ the other binary operator of the semiring. The translation $q \mapsto q \oplus q_a$ plays the role of the resource extension function ζ_a. In that algebraic setting, a dynamic-programming equation can be written. Its solution is the greatest lower bound on all the paths resources, and algorithms such as Ford-Bellman algorithm can compute its solution in polynomial time. These algorithms therefore provide the desired standard procedure to compute lower bounds. However, the idempotent semiring structure is too strong to model the practically interesting constrained shortest path problem for which we need the bounding procedure. We therefore endow $(\mathscr{R}, \oplus, \leqslant)$ with the weaker structure of lattice ordered monoid, which is versatile enough to model a wide range of path problems. The difference between the two structures lies in the fact that $\oplus$ is no more distributive with respect to the meet operator $\wedge$ of $\leqslant$. We show that, in this more general case, the polynomial algorithms mentioned above still provide a lower bound on all the paths resources, even if it is not necessarily the largest one. These algorithms are therefore standard procedures to build lower bounds in the versatile framework of lattice ordered monoid. To sum things up, the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM provides a versatile framework for resource constrained shortest path problems that leverages on ideas and algorithms from the algebraic path problem community to standardize the use of bounds in enumeration algorithms, and thus improve their practical practical performance.

The MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework shows its full power on stochastic path problems. Indeed, we show in Chapter 5 that it can model a wide range of path problems with stochasticity in the objective, in the constraints, or in both, and that our algorithms are practically efficient in solving them. In that case, the monoid is the space of random variables endowed with the addition. Depending on the hypotheses on the random variables, we endow it with the almost sure order, or with usual stochastic order, the second being coarser that the first. The coarser the order is, the tighter the lower bounds are, and the more efficient the enumeration algorithms are. Most probability functionals of interest are monotone with respect to these orders, which make the approach versatile. To the best of our knowledge, this approach provides the first practically efficient algorithms for path problems with probabilistic constraints. Chapter 11 shows the relevance of the approach on industrial instances of the airline STOCHASTIC CREW PAIRING PROBLEM where delay propagation is controlled. This idea of using an algebraic approach to solve stochastic path problems is quite original in stochastic optimization, and it is one of the main contribution of this dissertation.

When it comes to implementation, the advantage of a standardized framework is that its algorithms can be implemented once and for all in a library, and then applied to a wide range of problems. During this thesis, we have developed the `C++` library `LatticeRCSP` that implements all the algorithms in Part I. `C++` templates enable to use this library on any path problem

that can be modeled as a MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM. The coding work to deal with a new path problem is minimal: we only need to implement the elements and the operators of the lattice ordered monoid $(\mathscr{R}, \oplus, \leqslant)$, the cost function c and the feasibility function ρ. All the numerical experiments in this dissertation have been realized using this library. The main features of the library are described in Appendix C.

Part II focuses on airline operations problems. The AIRCRAFT ROUTING PROBLEM builds the rotations operated by the airplanes, and the CREW PAIRING PROBLEM those operated by the crews. As the solutions of these two problems are linked, they should ideally be solved in a single INTEGRATED PROBLEM. However, due to computational complexity, the current industrial practice is to solve them sequentially. Part II introduces a solution scheme for the INTEGRATED PROBLEM that can solve to near optimality large industrial instances. This scheme relies on efficient solution schemes for the AIRCRAFT ROUTING PROBLEM and the CREW PAIRING PROBLEM. We introduce a compact integer program for the AIRCRAFT ROUTING PROBLEM. On large industrial instances, this integer program is solved in at most a few minutes by modern MIP solvers. Besides, it is easier to implement than the column generation approach available in the literature. Following the literature, we solve the CREW PAIRING PROBLEM using a column generation approach. However, due to the complexity of the crew working rules, the usual resource constrained shortest path algorithms solve the pricing subproblem too slowly to be usable in a column generation scheme on large industrial instances. Our main contribution lies in the use of the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM algorithms of Part I to solve the pricing subproblem. Thanks to the use of bounds to discard paths, the use of these paths algorithms dramatically speeds up the resolution of the pricing subproblem and enable to solve to optimality large industrial instances. The value of these contributions comes from the importance of the problems considered for the airline industry.

Chapter 6 introduces techniques to improve the performance of the enumeration algorithms on difficult instances of the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM thanks to a preprocessing. In a column generation approach, this preprocessing needs to be done only once, and speeds-up the resolution of the pricing subproblem at all iterations of the column generation. Numerical experiments show that this technique enables to speed-up by a factor three the resolution of our industrial instances of the CREW PAIRING PROBLEM. This preprocessing technique can be applied to any problem that can be modeled within the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework. It can therefore be applied to speed up column generation approaches to a wide range of routing and scheduling problems whose subproblems can be modeled as instances of the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM.

Finally, Chapter 11 extends the AIRCRAFT ROUTING PROBLEM, the CREW PAIRING PROBLEM, and the INTEGRATED PROBLEM solution schemes to stochastic versions of these problems where the propagation of delay along rotations is controlled. Each of these stochastic problems can be solved to near optimality instances with hundreds of flight legs. The difficult part is the CREW PAIRING PROBLEM. Thanks to the ability of the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework to deal with stochasticity, the column generation approach can solve to optimality industrial instances of the stochastic CREW PAIRING PROBLEM.

Besides, to our knowledge, it is the first approach to these problems where delay is controlled thanks to probabilistic constraints – instead of a penalization in the objective.

Along this thesis, we developed first the ideas for the stochastic path problems, and detailed them in the following preprint.

Axel Parmentier and Frédéric Meunier. Stochastic shortest paths and risk measures. *arXiv preprint arXiv:1408.0272*, 2014

Working in parallel on the column generation approach to the airline CREW PAIRING PROBLEM, we had the problem that the available resource constrained shortest path library were not powerful enough to solve Air France industrial instances. We then figured out that the algebraic ideas we used for stochastic path problems could be generalized to a much wider setting, and we developed the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework in the following preprint.

Axel Parmentier. Algorithms for non-linear and stochastic resource constrained shortest paths. *arXiv preprint arXiv:1504.07880*, 2015

The link of our work to the algebraic path problem community came later, and Part I rereads and extends the content of those two preprints in that context. These contributions to path problems have also been presented in many conferences, and enabled me to win the first prize at the 2016 young researcher award of the French operations research society, the "prix jeune chercheur de la ROADEF". Articles based on Part II are currently being finalized.

12.2 Research directions

We now outline research directions to continue the research opened in this dissertation, from mature techniques to open questions.

The algorithms of Part II can be improved using techniques of the literature. As we detail in Section 9.3.2, the column generation approach to the CREW PAIRING PROBLEM can be improved by defining a heuristic to initialize the column generation with a good quality solution, by using a stabilization approach [60], and by using constraint aggregation [66, 67]. We also have mention in Remark 7.1 that the solution scheme for the INTEGRATED PROBLEM can be improved using a Branch and Check approach [167].

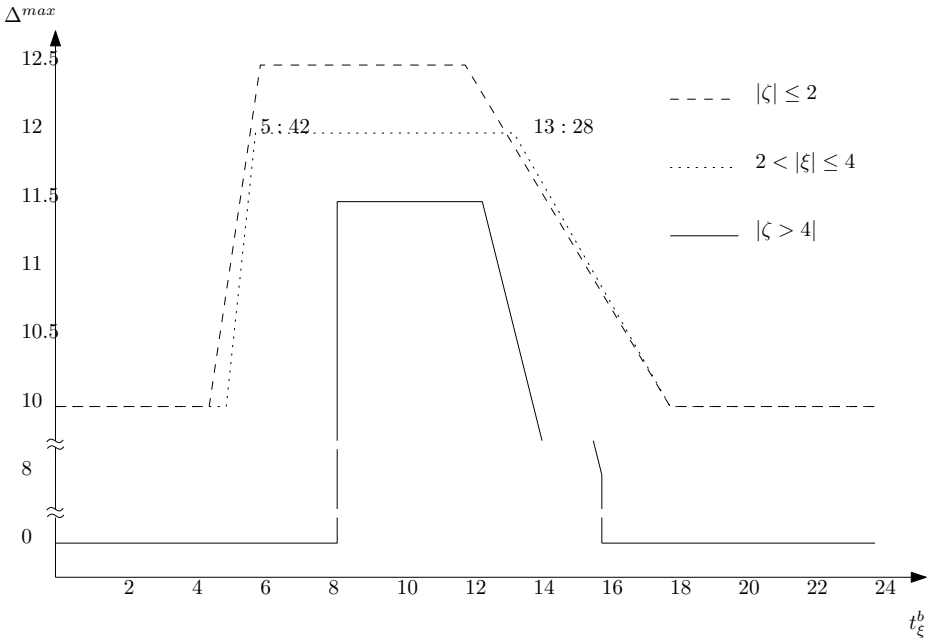
The MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework provides a practically efficient solution methodology for stochastic resource constrained shortest path problems. As resource constrained shortest path problems naturally arise as subproblems of decomposition approach to routing and scheduling problems, it provides a methodology to deal with stochastic versions of these problems. We are currently working on its applications to stochastic versions of the unit commitment problem.

We are also working on more data oriented versions of the stochastic path problems. On some practical applications, there is no easy model of the stochasticity in the resource constraints. From a machine learning point of view, instead of defining a random variable ξ_a, we would consider a vector ϕ_a of sufficient statistics. The infeasibility function ρ can then be interpreted as a binary classifier, that can be learned from the data. However, this raises the following

question: given a machine learning classifier ρ that is adequate from a statistical point of view, is there a lattice ordered monoid $(\mathscr{R}, \oplus, \leqslant)$ modeling the problem such that ρ is isotone with respect to $\leqslant$. The answer is yes in the case of generalized linear problems, and in the one of neural networks with monotone activation function. We are considering the application of this methodology to an airline operation problem not considered in this dissertation.

It is not standard in stochastic optimization to use algebraic approaches such as the one we develop for stochastic path problems. A natural question is therefore: can such an approach be generalized to other stochastic problems? Practically, these algorithms comes from a new look on an old literature of algebraic generalizations of the usual shortest path problem [86, 178]. We can therefore ask ourselves if other algebraic generalizations of traditional problems can be turned into algorithms for stochastic versions of the problem.

Appendix Part



A Wasserstein distance and stochastic optimization

The aim of this appendix is to prove Theorems 2.5 to 2.7 of Chapter 2. The appendix is organized as follows.

- Section A.1 introduces general results on the robusteness of probability functionals with respect to Wasserstein distance.
- Section A.2 considers stochastic optimization problems with stochastic objective function and deterministic set of feasible solutions. Theorems 2.5 and 2.6 are proved in this chapter.
- Section A.3 considers stochastic optimization problems whose set of feasible solutions is defined through probabilistic constraints, and proves Theorem 2.7.

A.1 Robustness of probability functionals with respect to Wasserstein distance

A.1.1 Risk measures

To control the error made in risk measures evaluation, we need the following lemma [144, 145].

Lemma A.1. *Let $\rho(\cdot)$ be a version independent risk measure and $\sup_{\sigma\in\mathscr{S}}\rho_\sigma$ be its distortion functional representation. If f is κ-Lipschitz, then*

$$\left|\rho_\mu f(X)-\rho_{\tilde{\mu}} f(X)\right| \leq \kappa W_1\left(\mu,\tilde{\mu}\right)\sup_{\sigma\in\mathscr{S}}\|\sigma\|_\infty.$$

A.1.2 Distributions with bounded density

To model on time arrival in the context of stochastic shortest path problem, or to model probabilistic constraints, we use probability functionals of the form $\rho(\cdot)=\mathbb{P}(\cdot>\tau)$ for a given threshold τ. Unfortunately, Lemma A.1 does not apply to these functionals. Indeed, given $\varepsilon>0$, let μ be the Dirac in $\tau-\varepsilon/2$ and $\tilde{\mu}$ be the Dirac in in $\tau+\varepsilon/2$, we have

$$\mathbb{P}_\mu(\xi>\tau)=0, \quad \mathbb{P}_{\tilde{\mu}}(\xi>\tau)=1, \quad \text{and } W_1(\mu,\tilde{\mu})=\varepsilon.$$

Nonetheless, under an additional assumption on the regularity of μ, we have the following

analogue of Lemma A.1.

Lemma A.2. *Let $\alpha > 0$. If f is κ-Lipschitz and the distribution $\mu^{f(\xi)}$ induced by $f(\xi)$ on $\mathbb{R}$ is bounded by B, then*

$$\left|\mathbb{P}_\mu\left(f(\xi) > \alpha\right) - \mathbb{P}_{\tilde{\mu}}\left(f(\xi) > \alpha\right)\right| \leq 2\sqrt{B\kappa W_1(\mu, \tilde{\mu})}.$$

Proof. It follows directly from the definition of the Wasserstein distance from the inherited distance that $W_1\left(\mu^{f(\xi)}, \tilde{\mu}^{f(\xi)}\right) \leq \kappa W_1\left(\mu^{\xi}, \tilde{\mu}^{\xi}\right) = \kappa W_1(\mu, \tilde{\mu})$ for κ-Lipschitz functions f. As a consequence, we only need to prove that

$$\left|\mathbb{P}_\mu\left(\xi > \alpha\right) - \mathbb{P}_{\tilde{\mu}}\left(\xi > \alpha\right)\right| \leq 2\sqrt{BW_1(\mu, \tilde{\mu})}.$$

Let $\Delta > 0$, and π be an optimal coupling between $\mathbb{P}_\mu$ and $\mathbb{P}_{\tilde{\mu}}$.

$$\begin{aligned}
&\mathbb{P}_\mu(f(\xi) > \alpha) - \mathbb{P}_{\tilde{\mu}}(f(\xi) > \alpha) \\
&\quad = \mathbb{P}_\pi\left\{(x,y) | x > \alpha \text{ and } y \leq \alpha\right\} - \mathbb{P}_\pi\left\{(x,y) | x \leq \alpha \text{ and } y > \alpha\right\} \\
&\quad \leq \mathbb{P}_\pi\left\{(x,y) | x > \alpha \text{ and } y \leq \alpha\right\} \\
&\quad = \mathbb{P}_\pi\left\{(x,y) | x > \alpha + \Delta \text{ and } y \leq \alpha\right\} + \mathbb{P}_\pi\left\{(x,y) | \alpha + \Delta \geq x > \alpha \text{ and } y \leq \alpha\right\} \\
&\quad \leq \mathbb{P}_\pi\left\{(x,y) | |x - y| \geq \Delta\right\} + \mathbb{P}_\pi\left\{(x,y) | \alpha + \Delta \geq x > \alpha\right\}.
\end{aligned} \tag{A.1}$$

Besides, we have,

$$W_1(\mu, \tilde{\mu}) = \int_{\mathbb{R}^2} |x - y| d\pi \geq \int_{\{|x-y| \geq \Delta\}} \Delta d\pi = \Delta \mathbb{P}_\pi\left\{(x,y) | |x - y| \geq \Delta\right\},$$

which implies $\mathbb{P}_\pi\left\{(x,y) | |x - y| \geq \Delta\right\} \leq \frac{W_1(\mu, \tilde{\mu})}{\Delta}$. In addition, the hypothesis on $\mu^{f(\xi)}$ implies $\mathbb{P}_\pi\left\{(x,y) | \alpha + \Delta \geq x > \alpha\right\} \leq B\Delta$. Therefore, Equation (A.1) gives

$$\mathbb{P}_\mu(f(\xi) > \alpha) - \mathbb{P}_{\tilde{\mu}}(f(\xi) > \alpha) \leq \frac{W_1(\mu, \tilde{\mu})}{\Delta} + B\Delta. \tag{A.2}$$

Minimizing on Δ, and reiterating the proof in the other direction gives the result. □

A.2 Stochastic objective functions

We can now prove the following Theorems 2.5 and 2.6.

Theorem 2.5. *Suppose that $\mathbb{X}$ is compact, $\xi \mapsto f(x, \xi)$ is κ-Lipschitz for all x, and that ρ is a version independent risk measure, and $\mathscr{S}$ be a set of distortion function that generates it. The existence of such an $\mathscr{S}$ is ensured by Corollary 2.2. Let μ and $\tilde{\mu}$ be two probability distributions, and let x^* and $\tilde{x}$ denote optimal solutions of $\min_{x \in \mathbb{X}} \rho_\mu\left(f(x, \xi)\right)$ and $\min_{x \in \mathbb{X}} \rho_{\tilde{\mu}}\left(f(x, \xi)\right)$ respectively. Then,*

$$\rho_\mu\left(f(\tilde{x}, \xi)\right) - \rho_\mu\left(f(x^*, \xi)\right) \leq 2\kappa \sup_{\sigma \in \mathscr{S}} \|\sigma\|_\infty W_1\left(\mu, \tilde{\mu}\right).$$

Note that the existence of optimal solutions x^* and $\tilde{x}$ is a consequence of the compactness of $\mathbb{X}$ and the fact that f is Lipschitz.

Proof of Theorem 2.5. Let $g(x) = \rho_\mu\big(f(x,\xi)\big)$ and $\tilde{g}(x) = \rho_{\tilde{\mu}}\big(f(x,\xi)\big)$, and $M = \kappa W_1\left(\mu,\tilde{\mu}\right)\sup_{\sigma\in\mathscr{S}}\|\sigma\|_\infty$. Lemma A.1 gives $|g(x) - \tilde{g}(x)| \leq M$ for all x in $\mathbb{X}$. As $\tilde{x}$ minimizes $\tilde{g}$, we have

$$\begin{aligned}
\rho_\mu\big(f(x,\xi)\big) &= g(\tilde{x}) \\
&\leq M + \tilde{g}(\tilde{x}) \\
&\leq M + \tilde{g}(x^*) \\
&\leq 2M + g(x^*) \\
&= \rho_\mu\big(f(x^*,\xi)\big) + 2\kappa \sup_{\sigma\in\mathscr{S}} \|\sigma\|_\infty W_1\left(\mu,\tilde{\mu}\right),
\end{aligned}$$

which gives the result. □

Theorem 2.6. *Suppose that $\mathbb{X}$ is compact, $\xi \mapsto f(x,\xi)$ is κ-Lipschitz for all x, and that τ is a given threshold, and let x^* and $\tilde{x}$ be defined as in Theorem 2.5.*

1. *If the support of $f(x,\xi)$ under μ and $\tilde{\mu}$ is in $\mathbb{Z}$ for all x, then*

$$\mathbb{P}_\mu\big(f(\tilde{x},\xi) > \tau\big) - \mathbb{P}_\mu\big(f(x^*,\xi) > \tau\big) \leq 2\kappa W_1\left(\mu,\tilde{\mu}\right).$$

2. *If there exists a B such that, for each $x \in \mathbb{X}$, the distribution of $f(x,\xi)$ is non-atomic and with density bounded by B, then*

$$\mathbb{P}_\mu\big(f(\tilde{x},\xi) > \tau\big) - \mathbb{P}_\mu\big(f(x^*,\xi) > \tau\big) \leq 4\sqrt{B\kappa W_1(\mu,\tilde{\mu})}.$$

Again, the existence of optimal solutions x^* and $\tilde{x}$ is a consequence of the compactness of $\mathbb{X}$ and the fact that f is Lipschitz.

Proof of Theorem 2.6. Point 1 of the theorem is a corollary of Theorem 2.5. Indeed, let m be the floor of τ, and let ϕ be the function $t \mapsto \max(0, \min(1, t-m))$. Then ϕ and $\mathbf{1}_{\cdot>\tau}$ coincide on $\mathbb{Z}$, which implies $\mathbb{P}_\mu\big(f(x,\xi) > \tau\big) = \mathbb{E}_\mu\big(\phi(f(x,\tau)\big)$ and $\mathbb{P}_{\tilde{\mu}}\big(f(x,\xi) > \tau\big) = \mathbb{E}_{\tilde{\mu}}\big(\phi(f(x,\tau)\big)$ for all x. As ϕ is 1-Lipschitz, $\xi \mapsto \phi(f(x,\xi))$ is κ-Lipschitz, and Theorem 2.5 gives the result.

Let $g(x) = \mathbb{P}_\mu\big(f(x,\xi) > \tau\big)$ and $\tilde{g}(x) = \mathbb{P}_{\tilde{\mu}}\big(f(x,\xi) > \tau\big)$, and $M = 2\sqrt{B\kappa W_1(\mu,\tilde{\mu})}$. Lemma A.2 gives $|g(x) - \tilde{g}(x)| \leq M$. As $\tilde{x}$ minimizes $\tilde{g}$, we have

$$\begin{aligned}
\mathbb{P}_\mu\big(f(\tilde{x},\xi) > \tau\big) &= g(\tilde{x}) \\
&\leq M + \tilde{g}(\tilde{x}) \\
&\leq M + \tilde{g}(x^*) \\
&\leq 2M + g(x^*) \\
&= \mathbb{P}_\mu\big(f(x^*,\xi) > \tau\big) + 2\sqrt{B\kappa W_1(\mu,\tilde{\mu})},
\end{aligned}$$

which gives Point 2. □

A.3 Probabilistic constraints

Given a random variable ξ with probability distribution μ, Theorem 2.7 considers feasible sets of the form

$$\mathbb{X}_\mu^\varepsilon(\xi) = \left\{x \in \mathbb{X} | \mathbb{P}_\mu\left[g(x,\xi) > 0\right] \leq \varepsilon\right\} \quad \text{with } \xi \mapsto g(x,\xi)\ \kappa_g\text{-Lipschitz.}$$

Theorem 2.7. *Suppose that one of the following assumptions is satisfied.*

1. *For all x and ξ in the union of the supports of μ and $\tilde{\mu}$, $g(x,\xi) \in \mathbb{Z}$.*
2. *There exists a B such that, for each $x \in \mathbb{X}$, the distribution of $f(x,\xi)$ is non-atomic and with density bounded by B.*

Then,

$$\mathbb{X}_{\tilde{\mu}}^\alpha \subseteq X_\mu^\varepsilon \subseteq X_{\tilde{\mu}}^\beta, \quad \textit{where} \quad \begin{cases} \alpha = \varepsilon - \kappa_G W_1(\mu,\tilde{\mu}), \\ \beta = \varepsilon + \kappa_G W_1(\mu,\tilde{\mu}), \end{cases} \quad \textit{under Assumption 1,}$$

$$\begin{cases} \alpha = \varepsilon - 2\sqrt{B\kappa_G W_1(\mu,\tilde{\mu})}, \\ \beta = \varepsilon + 2\sqrt{B\kappa_G W_1(\mu,\tilde{\mu})}, \end{cases} \quad \textit{under Assumption 2,}$$

κ_G is the Lipschitz constant of g, and $X_{\tilde{\mu}}^\alpha = \emptyset$ if $\alpha < 0$. If in addition ρ_μ is a version independent risk measure generated by a set of functionals $\mathscr{S}$ and $\sigma_\infty = \sup_{\sigma\in\mathscr{S}} \|\sigma\|_\infty$, we have

$$\min_{x\in\mathbb{X}_{\tilde{\mu}}^\beta(\xi)} \rho_{\tilde{\mu}}\left[f(x,\xi)\right] - 2\kappa_f\sigma_\infty W_1(\mu,\tilde{\mu}) \leq \min_{x\in\mathbb{X}_\mu^\varepsilon(\xi)} \rho_\mu\left[f(x,\xi)\right] \leq \min_{x\in\mathbb{X}_{\tilde{\mu}}^\alpha(\xi)} \rho_{\tilde{\mu}}\left[f(x,\xi)\right] + 2\kappa_f\sigma_\infty W_1(\mu,\tilde{\mu}).$$

Proof. Suppose that Assumption 1 is satisfied. As in the proof of Theorem 2.6, remark that $\mathbb{P}_\mu\left(g(x,\xi) > 0\right) = \mathbb{E}_\mu\left(\phi(g(x,\tau)\right)$ and $\mathbb{P}_{\tilde{\mu}}\left(f(x,\xi) > \tau\right) = \mathbb{E}_{\tilde{\mu}}\left(\phi(f(x,\tau)\right)$ for all x where ϕ is the Lipschiz function $t \mapsto \max(0, \min(1,t))$. Let x be in $\mathbb{X}_{\tilde{\mu}}^\alpha$. Lemma A.1 applied to function ϕ then implies that

$$\mathbb{P}_\mu\left[g(x,\xi) > 0\right] \leq \mathbb{P}_{\tilde{\mu}}\left[g(x,\xi) > 0\right] + \kappa_g W_1(\mu,\tilde{\mu}) \leq \alpha + \kappa_g W_1(\mu,\tilde{\mu}) = \varepsilon,$$

and thus x belongs to $\mathbb{X}_\mu^\varepsilon$. The same method gives $X_\mu^\varepsilon \subseteq X_{\tilde{\mu}}^\beta$.

Suppose now that Assumption 2 is satisfied. Let x be in $\mathbb{X}_{\tilde{\mu}}^\alpha$. Lemma A.2 implies

$$\mathbb{P}_\mu\left[g(x,\xi) > 0\right] \leq \mathbb{P}_{\tilde{\mu}}\left[g(x,\xi) > 0\right] + 2\sqrt{B\kappa_G W_1(\mu,\tilde{\mu})} \leq \alpha + 2\sqrt{B\kappa_G W_1(\mu,\tilde{\mu})} = \varepsilon,$$

which again gives the result.

Assuming that ρ is generated by the set of distortion function $\mathscr{S}$, we have

$$\min_{x\in\mathbb{X}_\mu^\varepsilon(\xi)} \rho_\mu\left[f(x,\xi)\right] \leq \min_{x\in\mathbb{X}_{\tilde{\mu}}^\alpha(\xi)} \rho_\mu\left[f(x,\xi)\right] \leq \min_{x\in\mathbb{X}_{\tilde{\mu}}^\alpha(\xi)} \rho_{\tilde{\mu}}\left[f(x,\xi)\right] + 2\kappa_f\sigma_\infty W_1(\mu,\tilde{\mu}),$$

the second inequality following from Theorem 2.5. The same method gives

$$\min_{x\in\mathbb{X}^{\beta}_{\tilde{\mu}}(\xi)} \rho_{\tilde{\mu}}\left[f(x,\xi)\right] \leq \min_{x\in\mathbb{X}^{\varepsilon}_{\mu}(\xi)} \rho_{\tilde{\mu}}\left[f(x,\xi)\right] \leq \min_{x\in\mathbb{X}^{\varepsilon}_{\mu}(\xi)} \rho_{\mu}\left[f(x,\xi)\right] + 2\kappa_f \sigma_\infty W_1(\mu,\tilde{\mu}).$$

□

B AIRCRAFT ROUTING PROBLEM $\mathcal{NP}$-completeness

The input of the AIRCRAFT ROUTING PROBLEM introduced in Part II is a connection graph. This is not the standard input for the AIRCRAFT ROUTING PROBLEM. Indeed, a more natural input is the set of flight legs that should be covered. The objective of this appendix is to prove the complexity theorems sketched in Chapter 8 for the "schedule version" of the AIRCRAFT ROUTING PROBLEM.

Appendix B is organized as follows.

- Appendix B.1 gives a definition of AIRCRAFT ROUTING PROBLEM based on flight legs schedule.
- Appendix B.2 gives a polynomial algorithm for the AIRCRAFT ROUTING PROBLEM which proves Theorem 8.3. The bound on AIRCRAFT ROUTING PROBLEM complexity given in this Appendix is stronger than the one in Chapter 8.
- Appendix B.3 proves Theorem 8.2 which states AIRCRAFT ROUTING PROBLEM $\mathcal{NP}$-completeness. The proof in this is valid for the natural definition of AIRCRAFT ROUTING PROBLEM, which takes in input the set of flight legs to be covered. This complexity result is stronger than the one proved in Chapter 8, which is valid only for the definition of AIRCRAFT ROUTING PROBLEM in terms of connection graph.

B.1 AIRCRAFT ROUTING PROBLEM definition

B.1.1 Problem definition

We consider the flight legs operated by a company on a period of time. Let *horizon* H be the number of days this period lasts. Time is discretized in τ steps per day. The network is composed of a set of *airports* $a \in \mathscr{A}$. Maintenance operations can be performed on airplanes in only a few airports called *maintenance bases*. Let $\mathscr{B} \subseteq \mathscr{A}$ be the set of maintenance bases. Each *flight leg* $\ell \in \mathscr{L} \subseteq (\mathscr{A} \times [\tau] \times [H])^2$ is identified by two airport time day triples, one corresponding to its departure $(a_\ell^{\text{dep}}, t_\ell^{\text{dep}}, d_\ell^{\text{dep}})$ and one corresponding to its arrival $(a_\ell^{\text{arr}}, t_\ell^{\text{arr}}, d_\ell^{\text{arr}})$. The scheduled arrival time t_ℓ^{arr} is equal to the scheduled landing time plus the minimum turn time between two flight legs – this way the airplane operating ℓ can be chained to any flight leg leaving a_ℓ^{arr} after t_ℓ^{arr}. To each day $d \in [H-1]$ corresponds a night set $\Lambda_d \subseteq \mathscr{L} \cup \mathscr{L}^2$. A flight leg ℓ belongs to night set Λ_d if $d_{\ell_1}^{\text{dep}} \leq d$ and $d_{\ell_2}^{\text{arr}} > d$. A couple of flight legs (ℓ_1, ℓ_2) belongs

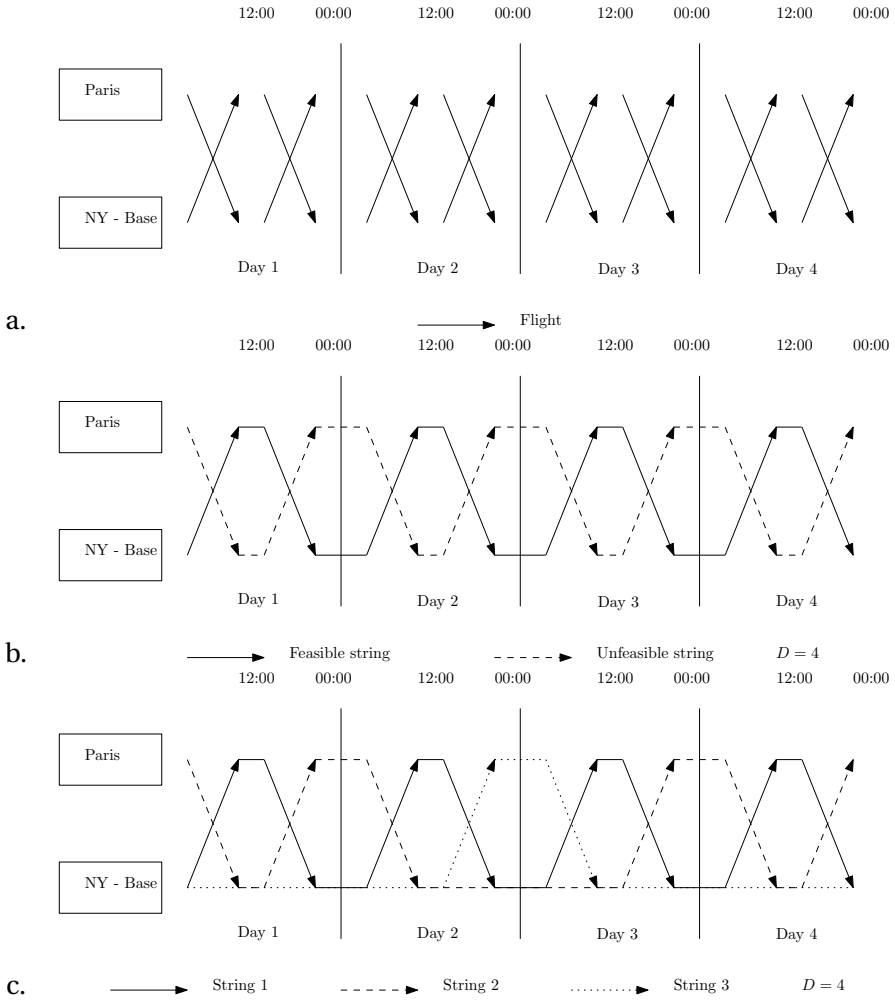


Figure B.1 – a. An aircraft routing instance – b. flight leg strings – c. A feasible routing

to night set Λ_d if the departure airport of ℓ_2 is equal to the arrival airport of ℓ_1 and $d^{\text{arr}}_{\ell_1} \leq d$ and $d^{\text{dep}}_{\ell_2} > d$. A couple of flight legs is a *maintenance couple* $(\ell_1, \ell_2) \in \Pi_d \subseteq \Lambda_d$ if it is a night couple $(\ell_1, \ell_2) \in \Lambda_d$ in a base $\alpha^{\text{arr}}_{\ell_1} \in \mathscr{B}$ such that the time interval between the two flight legs is sufficient to perform a maintenance check $(t^{\text{dep}}_{\ell_2} + \tau) - t^{\text{arr}}_{\ell_1} \geq \tau_M$, where τ_M is the time needed to perform a maintenance check. An example of aircraft routing instance is on Figure B.1.

A *flight leg string* σ is a list of flight legs $\sigma = (\ell_1, \ell_2, \ldots, \ell_p)$ such that $\alpha^{\text{arr}}_{\ell_i} = \alpha^{\text{dep}}_{\ell_{i+1}}$ and $t^{\text{arr}}_{\ell_i} \leq t^{\text{dep}}_{\ell_{i+1}}$ for $i = 1, \ldots, p-1$. A string operates a maintenance on night d if there exist two successive flight legs ℓ_i and ℓ_{i+1} of σ such that $(\ell_i, \ell_{i+1}) \in \Pi_d$ is a maintenance couple for night d. A flight leg string is *feasible* if it visits a maintenance night $(\ell_i, \ell_{i+1}) \in \pi_d$ at least once in each sequence of δ_{maint} successive nights. A flight leg ℓ is *covered* by a flight leg string σ if it is one of the flight legs of σ. Considering a flight leg ℓ, the number of *days of operations* $o_\ell = d^{\text{dep}}_\ell - d$ of ℓ with respect to σ is equal to the number of days between the departure of the flight leg d^{dep}_ℓ and the last maintenance night set Π_d covered by σ. Example of flight leg strings are given on Figure B.1.

The goal of the AIRCRAFT ROUTING PROBLEM is to cover all flight legs with feasible flight leg strings. But considering one specific flight leg string σ, the last flight leg ℓ of σ does not necessarily arrive in a maintenance base, and thus its number of days of operations o_ℓ must be taken into account as an initial condition in the AIRCRAFT ROUTING PROBLEM for the next period. Thus, in the AIRCRAFT ROUTING PROBLEM, the initial positions of airplanes are given: for each day of operation $d \in [\delta_{\text{maint}}]$ and airport α, there are initially κ^d_α airplanes that must visit a maintenance base after at most $\delta_{\text{maint}} - d + 1$ days. For each airport α and day of operation δ_{maint}, the final conditions are given by the γ^o_α: there are at least $\sum_{o=1}^{d} \gamma^o_\alpha$ airplanes that have had a maintenance in the last δ_{maint} nights. The *fleet size* is $k = \sum_{\alpha \in \mathscr{A}} \sum_{d \in [\delta_{\text{maint}}]} \kappa^d_\alpha$.

A *routing* $\mathscr{S}$ is a collection of flight leg strings $\sigma \in \mathscr{S}$. A routing is *feasible* if each flight leg string $\sigma \in \mathscr{S}$ is feasible, each flight leg $\ell \in \mathscr{L}$ is covered by one unique flight leg string $\sigma \in \mathscr{S}$, at least $\sum_{o=d}^{\delta_{\text{maint}}} \kappa^o_\alpha$ flight leg strings starting in airport a visit a maintenance night after at most $\delta_{\text{maint}} - d + 1$ days for all $d \in [\delta_{\text{maint}}]$, and at least $\sum_{o=1}^{d} \gamma^o_\alpha$ flight leg string ending in airport a have visited a maintenance night in the last δ_{maint} days for all $d \in [\delta_{\text{maint}}]$. A feasible routing is on Figure B.1.

AIRCRAFT ROUTING PROBLEM
Input. An horizon H, a time discretization T, a set of airports $a \in \mathscr{A}$, a set of bases $\mathscr{B} \subseteq \mathscr{A}$, a set of flight legs $\mathscr{L} \subseteq (\mathscr{A} \times T \times D)^2$, a maximum number of days between two maintenance nights δ_{maint}, and for each airport $a \in A$ and day $o \in [\delta_{\text{maint}}]$ the initial and final number of airplanes κ^o_α and γ^o_α.
Output. A feasible routing $\mathscr{S}$.

B.1.2 Equivalence with the Airport-Time graph routing problem

In Chapter 8, we use a connection graph whose vertices are the flight legs and whose arcs are the feasible connection to model the AIRCRAFT ROUTING PROBLEM. In this appendix, we use the notion of *airport time* graph. The vertices of an airport time graph are the pairs (a, t) of an airport a and a time t such that there is leg ℓ departing from a at t. The arcs of an airport

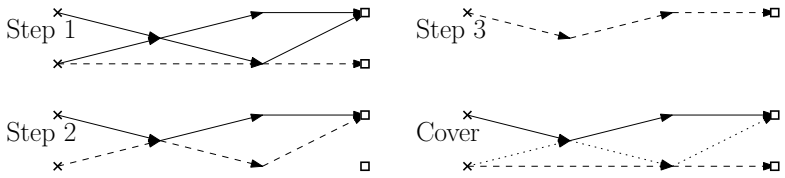


Figure B.2 – Greedy algorithm to find a path partition on an equigraph

time graph correspond to the flight legs or to some ground arcs between successive departure times in the same airport. If the grounds arcs are defined with the right multiplicity, an aircraft route is a source to sink path in the airport time graph, and an aircraft routing problem is a partition of the arcs of the airport time graph into source to sink paths. We therefore start by introducing the properties of graphs that can be partitioned in source to sink paths. These properties are the key ingredients of the algorithms and proofs of this appendix. We then introduce the AIRPORT-TIME GRAPH ROUTING PROBLEM and prove its equivalence with the AIRCRAFT ROUTING PROBLEM.

$S-T$ paths partitions of graphs

Let $D=(V,A)$ be an acyclic directed graph. Let $\delta^-(v)$ (resp. $\delta^+(v)$) be the set of incoming (resp. outgoing) arcs from v. Let $d^-(v)=|\delta^-(v)|$ (resp. $d^+(v)=|\delta^+(v)|$) be the incoming (resp. outgoing) degree of v. A vertex $v\in V$ is a *source* if $\delta^-(v)=\emptyset$, and it is a *sink* if $\delta^+(v)=\emptyset$. We denote by S the set of sources, and by T the set of sinks. Let $I=V\setminus(S\cup T)$ be the set of *internal vertices*. A path $P\in G$ is a *source to sink path*, or *$S-T$ path* if it starts in a source and ends in a sink. An arc $a\in A$ is *covered* by a path P if $a\in P$. A collection of paths $\mathcal{P}$ is a *path partition* of D if for each arc $a\in A$, there exists one and only one path $P\in\mathcal{P}$ such that $a\in P$. In a directed graph, a *directed cut* is an arc set C such that $C=\delta^-(U)$ for a vertex set U satisfying $\delta^+(U)=\emptyset$. Besides, C is an *S-T directed cut* if $T\subseteq U$ and $S\subseteq(V\setminus U)$. Note that $\delta^+(S)$ and $\delta^-(T)$ are S-T directed cuts.

Proposition B.1. *Let $D=(V,A)$ be an acyclic directed graph. The three following properties are equivalent.*

1. *There exists a partition of the arcs of D in S-T paths.*
2. *$d^-(v)=d^+(v)$ for each internal vertex $v\in I$.*
3. *All the S-T directed cuts of D have the same cardinal.*

Besides, all the S-T paths partitions of D have the same number of paths, which is equal to the cardinal of the S-T directed cuts.

An acyclic directed graph satisfying the properties of Proposition B.1 is called an *equigraph*. The *flow* of an equigraph is the cardinal of the S-T directed cuts.

Proof of Proposition B.1. The fact that Property 1 implies Property 2 comes from the fact that, if a digraph admits an S-T paths partition, the incoming degree and the outgoing degree of each internal vertex is equal to the number of paths in the partition that cross v. The converse

comes from the greedy algorithm applied in a graph satisfying $d^-(v) = d^+(v)$. Indeed, the property $d^-(v) = d^+(v)$ is not affected by the removal of an S-T path, as the incoming and the outgoing degree of any internal vertex are decreased by the same quantity.

We now prove that Property 2 is equivalent to Property 3. Let D be a digraph satisfying $d^-(v) = d^+(v)$ for each internal vertex, C be an S-T directed cut, and $\mathscr{P}$ be a path partition of G, then each path $P \in \mathscr{P}$ intersects C exactly once: there is a one to one mapping from $\mathscr{P}$ to C, which gives the "only if" direction. Conversely, let D be a graph that does not satisfy Property 2. There exists a vertex v such that $d^+(v), d^-(v) > 0$ and $d^+(v) \neq d^-(v)$. Suppose first that $|\delta^-(v)| < |\delta^+(v)|$. Let U be the union of the set of vertices u such that there is a path from v to u with the set of sinks. Then $\delta^-(U)$ and $\delta^-(U \cup \{v\})$ are two S-T directed cuts, and $|\delta^-(U \cup \{v\})| \neq |\delta^-(U)|$. The case with $|\delta^-(v)| > |\delta^+(v)|$ is analogous.

The final remark comes from the fact that $\delta^+(S)$ is a directed cut, and that, as D is acyclic, each path of a partition intersects $\delta^+(S)$ exactly once. □

Lemma B.2. *If P is an S-T path in an equigraph D, then $D \backslash P$, the graph obtained by removing the arcs of P and the newly isolated vertices is an equigraph.*

Proof. No incoming (resp. outgoing) arcs are added to sources (resp. sinks), and if v is an internal vertex on P, both its incoming and outgoing degree are decreased by the same quantity. □

This lemma can be seen as a linear time *greedy algorithm to find a path partition on an equigraph* in $O(|A|)$. Indeed, as for each internal vertex, $|\delta^-(v)| = |\delta^+(v)|$, a path from a source to the sinks set is always found by choosing arbitrarily the successive arcs in the set of outgoing arcs from the path current destination. Thus, a path partition is found by iteratively picking a path from sources to sink and remove it from the equigraph, as shown on Figure B.2. At each step, the graph obtained by removing a path is still an equigraph whose number of sources or sinks has been decreased by one. Because it is acyclic, an equigraph without sources and sinks is empty, and the algorithm finally gives a path partition of the equigraph.

Airport-time graph problem

Given an acyclic digraph $D = (V, A)$, a collection of directed cuts $N_d = \delta^-(U_d)$ for $d \in [H]$ is a collection of *nights* if $U_{d+1} \subseteq U_d$ for all $d \in [H-1]$ and $U_H = T$ is the set of sinks. Such a collection is illustrated on Figure B.3. The *horizon* is the number of nights H. The day of a vertex $\text{day}(v)$ is the index of the first night after v: $\text{day}(v) = d$ if $v \in U_{d-1} \backslash U_d$. Besides, the maintenance checks operated in bases during nights are modeled thanks to a given collection M_d of *maintenance arc sets* satisfying $M_d \subseteq N_d$. Set M_d identifies the maintenance night arcs among the night arcs of N_d. A path P intersects a set N if $P \cap N \neq \emptyset$.

Let δ_{maint} be an integer called the *maintenance requirement*. A path P from sources S to sinks T necessarily intersects all nights N_d. It is a *feasible path* if it intersects a maintenance night arc at least every δ_{maint} nights: $P \cap \left(\cup_{o=d}^{d+\delta_{\text{maint}}-1} M^o \right) \neq \emptyset$ for $d \in [H - \delta_{\text{maint}} + 1]$. A path P *covers* an arc a if $a \in P$. For an arc a covered by a path P, the number of days of operations $o_P(a)$

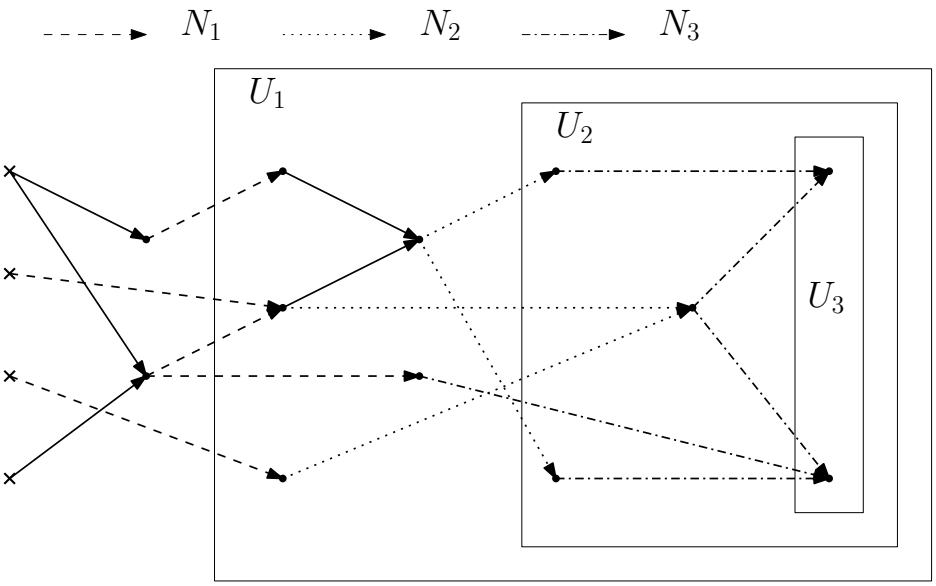


Figure B.3 – A set of nights on an equigraph

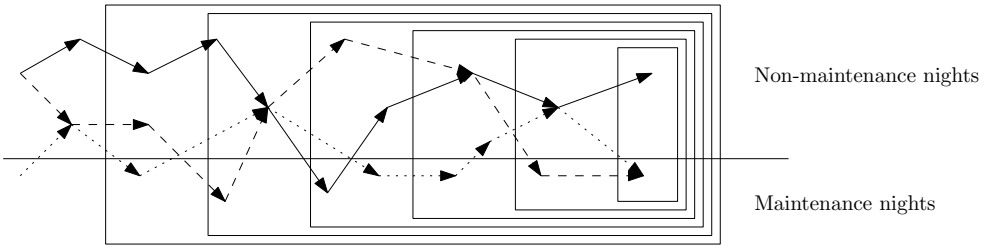


Figure B.4 – A feasible routing

is equal to the number of days between a and the last maintenance night M_d intersected. Thus, a path is feasible if the number of days of operations of all its arcs is smaller than the maintenance constraint: $o_P(a) \leq \delta_{\text{maint}}$ for all $a \in P$.

Initial and final constraints are given on the number of days of operations at the beginning (source) and at the end (sink) of each feasible path. For each source (resp. sink) s and integer $o \in [\delta_{\text{maint}}]$, let κ_s^o (γ_s^o) be these constraints. A collection of *feasible* paths $\mathscr{P}$ is a *feasible routing* if the following conditions are satisfied: first, each vertex is covered exactly once. Second, there are at least $\sum_{o=d}^{\delta_{\text{maint}}} \kappa_s^o$ paths starting in source s that visits one of the first $\delta_{\text{maint}} - d + 1$ maintenance night arc sets $\bigcup_{o=1}^{\delta_{\text{maint}}-d+1} M_o$ for all $d \in [\delta_{\text{maint}}]$. Finally there are at least $\sum_{o=1}^{d} \gamma_t^o$ paths ending in sink t that visits one of the last δ_{maint} maintenance night arc sets $\bigcup_{o=H-d+1}^{H} M_o$ for all $d \in [\delta_{\text{maint}}]$. Integer κ_s^o are the initial requirements, and γ_t^o are the final requirements. A feasible routing is plotted on Figure B.4.

The AIRPORT-TIME GRAPH ROUTING PROBLEM can be stated as follows:

AIRPORT-TIME GRAPH ROUTING PROBLEM
Input. An acyclic digraph $D = (V, A)$, a collection of nights $N_d \subseteq A$, a collections of mainte-

nance nights $M_d \subseteq N_d$, a maintenance requirement δ_{maint}, a collection of initial requirements κ_s^o for each source s and $o \in [\delta_{\text{maint}}]$, and a collection of final requirements γ_t^o for each sink t and $o \in [\delta_{\text{maint}}]$.
Output. A feasible routing.

Note that, as a feasible routing is an S-T path partition of D, if the AIRPORT-TIME GRAPH ROUTING PROBLEM admits a solution, then D is an equigraph.

Theorem B.3. *The* AIRCRAFT ROUTING PROBLEM *and the* AIRPORT-TIME GRAPH ROUTING PROBLEM *are equivalent: each problem can be reduced to the other one in linear time.*

Thanks to this theorem, the AIRPORT-TIME GRAPH ROUTING PROBLEM and the AIRCRAFT ROUTING PROBLEM have the same complexity and can be solved by the same algorithms. The remaining of this appendix focuses on AIRPORT-TIME GRAPH ROUTING PROBLEM. In the following long proof, only the methods used to build the equivalent instance are of practical interest.

Remark B.1. The reason why an analogue of Theorem B.3 cannot be proved for the connection graph statement of the AIRCRAFT ROUTING PROBLEM in Chapter 8 is that the connection graph is the line-graph of the airport time graph. As a consequence, if a connection graph can be rebuilt from the airport time graph, the converse is false.

Proof. We first prove that AIRCRAFT ROUTING PROBLEM can be reduced in linear time to an equivalent AIRPORT-TIME GRAPH ROUTING PROBLEM. Let $T, \mathscr{A}, \mathscr{B}, \mathscr{L}, \delta_{\text{maint}}, \kappa_\alpha^d, \gamma_\alpha^d$ be an instance of the AIRCRAFT ROUTING PROBLEM.

First, we build the AIRPORT-TIME GRAPH ROUTING PROBLEM instance $D', N'_d, M'_d, \delta'_{\text{maint}}, \kappa_s^{d'}, \gamma_t^{d'}$. The total number of airplanes initially in α is $\theta_\alpha^{t_0} = \sum_{o=1}^{\delta_{\text{maint}}} \kappa_\alpha^o$. As each flight leg is covered exactly once, the number of airplanes θ_α^t at t in α is obtained by counting the arrivals and the departures.

$$\theta_\alpha^t = \theta_\alpha^{t_0} + |\{\text{arrivals before t}\}| - |\{\text{departures before t}\}| \tag{B.1}$$

$$= \theta_\alpha^{t_0} + \sum_{\tau=0}^{t} \left[\sum_{\ell | \alpha_\ell^a = \alpha, t_\ell^a = \tau} 1 - \sum_{\ell | \alpha_\ell^d = \alpha, t_\ell^d = \tau} 1 \right] \tag{B.2}$$

The aircraft routing directed graph $D' = (V', A')$ is defined as follows: for each airport $a \in A$ and each time (t, d) such that there is at least one departure from α at (t, d), there is a corresponding internal vertex $v'(\alpha, t) \in V'$. For each airport α there is a corresponding source $s'_\alpha \in S$ and a sink $t'_\alpha \in T$. For each flight leg $\ell = ((\alpha_\ell^d, t_\ell^d, d_\ell^d), (\alpha_\ell^a, t_\ell^a, d_\ell^a))$, there is a corresponding flight leg arc between $v'(\alpha_\ell^d, t_\ell^d, d_\ell^d)$ and $v'(\alpha_\ell^a, t_1, d_1)$ where (t_1, d_1) is the first time instant after (t_ℓ^a, d_ℓ^a) such that there is a flight leg departure from α in (t_1, d_1). If such a time instant does not exist, then the destination vertex of the flight leg arc is t'_α. Each vertex v' corresponds to an airport, a time and a day day(v'). Let $t_1, t_2, \ldots, t_n$ be the successive departure times at an airport α, for $i \in [n]$, there are $\theta_\alpha^{t_i}$ ground arcs between $v'(\alpha, t_i)$ and $v'(\alpha, t_{i+1})$ or t'_a if $i = n$. Night $N'_d = \delta^-(U'_d)$ is the directed cut of events on day greater than δ_{maint}, i.e.

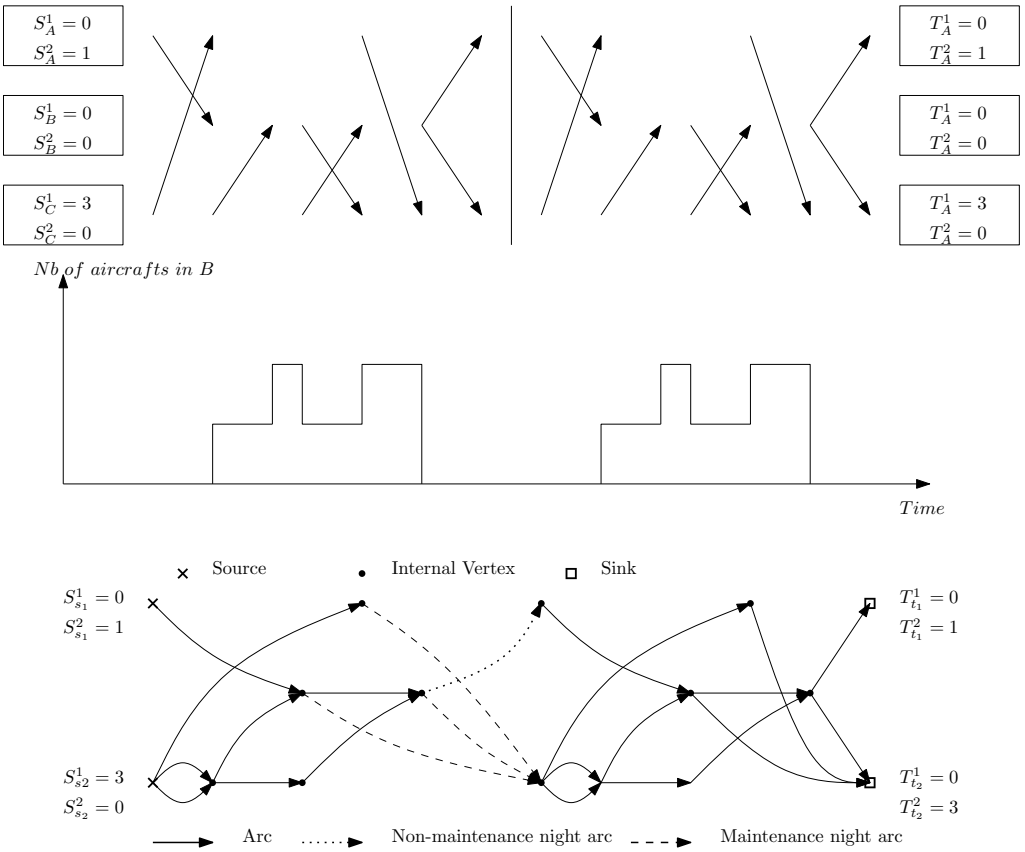


Figure B.5 – Aircraft routing instance and corresponding airport time graph. Rows on the first figure correspond to airports, and arrows on the first figure correspond to flight legs. The second figure gives the number of aircrafts in A airport B, and the last one illustrates the corresponding airport time graph

$U'_d = \{v' | \text{day}(v') > d\}$. Thus, night characterization $U'_{d+1} \subseteq U'_d$ is satisfied. Maintenance arcs in M_d are night arcs in N_d that are ground arcs in a base b. For all s, t and δ_{maint}, initial and final constraints are equal $\kappa_\alpha^{d'} = \kappa_s^d$ and $\gamma_t^{d'} = \gamma_t^d$. This process for a small instance of aircraft routing in on Figure B.5.

The resulting graph is an equigraph. Indeed, sources s' and sinks t' satisfy $d^-(s') = 0$ and $d^+(t') = 0$. Let $v = v'(\alpha, t)$ be an internal vertex, and t_- the former departure time from α, then

$$d^+(v) - d^-(v) = \theta_v^t - \theta_v^{t_-} - \left[\sum_{\ell | \alpha_\ell^a = \alpha, t_\ell^a = t} 1 - \sum_{\ell | \alpha_\ell^d = \alpha, t_\ell^d = t} 1 \right] = 0 \tag{B.3}$$

which gives the result.

AIRCRAFT ROUTING PROBLEM *feasibility implies* AIRPORT-TIME GRAPH ROUTING PROBLEM *feasibility*. Suppose the aircraft routing admits a solution $\mathcal{S}$. A flight leg string σ covers a ground arc a if a is between two successive flight leg arcs in σ. The number of parallel ground arcs is equal to the number or airplanes at the corresponding airport and time, thus each

copy can be assigned to one unique flight leg string σ to form a corresponding path $P'(\sigma)$. $P'(\sigma)$ is feasible because σ is feasible. $\mathscr{P} = \{P(\sigma)|\sigma \in \mathscr{S}\}$ satisfies equigraph cover constraint, initial constraints, and final constraints because R satisfies equigraph cover constraint and sink constraints.

Conversely, AIRPORT-TIME GRAPH ROUTING PROBLEM *feasibility implies* AIRCRAFT ROUTING PROBLEM *feasibility*. Let $\mathscr{P}$ be a solution of the equigraph problem. Each feasible path P induces (without ambiguity) a feasible string $S(P)$ and $\sigma = \{S(P), P \in \mathscr{P}\}$ is a feasible routing.

Now, we prove that the AIRPORT-TIME GRAPH ROUTING PROBLEM can be reduced to the AIRCRAFT ROUTING PROBLEM. Let $D = (V, A), B, \delta_{\text{maint}}, \kappa_s^d, \gamma_t^d$ be an instance of AIRPORT-TIME GRAPH ROUTING PROBLEM. Then an instance $T', \mathscr{A}', \mathscr{B}', \mathscr{L}, \delta'_{\text{maint}}, \kappa_\alpha^{d,}, \gamma_\alpha^{d,}$ of aircraft routing is built as follows. Day day(v) has already been defined for an equigraph with night. A time index respecting the natural ordering of each vertex of a day is obtained thanks to a depth first exploration of the reduced graphs $U_d \backslash U_{d+1}$ corresponding to one day, assigning to each vertex the maximum index of its predecessors plus one. To each arc in $(v_1, v_2) \in A \backslash B$ corresponds a flight leg $f = ((\alpha(v_1), t(v_1), d(v_1)), (\alpha(v_2), t(v_2), d(v_2))$. T' is the maximum of the time indices.

AIRCRAFT ROUTING PROBLEM *feasibility implies* AIRPORT-TIME GRAPH ROUTING PROBLEM *feasibility:* feasible paths give feasible strings. AIRPORT-TIME GRAPH ROUTING PROBLEM *feasibility implies* AIRCRAFT ROUTING PROBLEM *feasibility:* feasible strings give feasible paths. □

B.2 Polynomial algorithm

B.2.1 Maintenance state graph

This section introduces for airport time graphs the notion of state graph that has been introduced in Section 8.2 for connection graphs.

Given an instance $(D, N_d, M_d, \delta_{\text{maint}}, \kappa_s^o, \gamma_s^o)$ of the AIRPORT-TIME GRAPH ROUTING PROBLEM, we define its *maintenance state graph* $\mathscr{D} = (\mathscr{V}, \mathscr{A})$ on the airport time graph D as follows. For each vertex v, we built δ_{maint} *state vertices* $\vartheta_v^1, \ldots, \vartheta_v^{\delta_{\text{maint}}}$. The index o on ϑ_v^o indicates that the last maintenance of an airplane which crosses ϑ_v^o happened o days ago. For each vertex, we denote $\mathscr{V}_v$ the set of state vertices of v, and for each state vertex ϑ, we denote $\theta(\vartheta)$ the vertex v such that $\vartheta \in \mathscr{V}_v$.

We build the state arcs in order to ensure that *for each source to sink path π in $\mathscr{D}$ covering ϑ_v^o, the number of days since the last visit of P in a maintenance arc is equal to o*, where P is the path corresponding to Π in D. We denote $\mathscr{A}_a$ the set of state arcs of a, and $\theta(\alpha)$ the arc a such that $\alpha \in \mathscr{A}_a$. For each arc $a = (v_1, v_2) \in A \setminus (\cup_d N_d)$ and $o \in [\delta_{\text{maint}}]$, we add to $\mathscr{A}_a$ the state arc $(\vartheta_{v_1}^o, \vartheta_{v_2}^o)$. State does not change on a non-night arc. For each arc $a = (v_1, v_2) \in \cup_d M_d$ and $o \in [\delta_{\text{maint}}]$, add the arc $(\vartheta_{v_1}^o, \vartheta_{v_2}^1)$ to $\mathscr{A}_a$. After a maintenance arc, the number of days since the last maintenance night is equal to 1. Finally, for each arc $a = (v_1, v_2) \in (\cup_d N_d \setminus \cup_d M_d)$ and $o \in [\delta_{\text{maint}} - 1]$, add the arc $(\vartheta_{v_1}^o, \vartheta_{v_2}^{o+1})$ to $\mathscr{A}_a$. After a non-maintenance night, the number of days since the last maintenance night has increased by one. If δ_{maint} days have elapsed

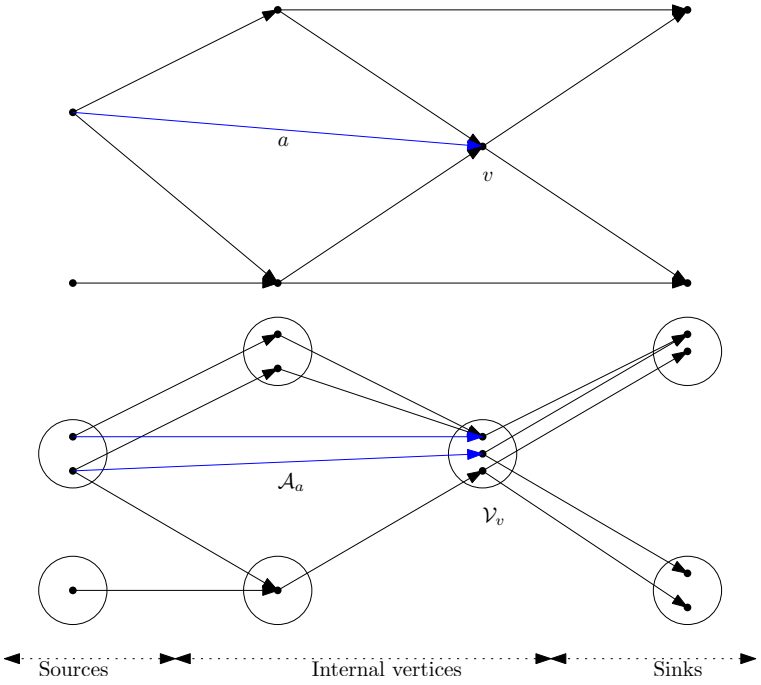


Figure B.6 – A network graph and its state graph

since the last visit of an airplane in v_1 to a maintenance night arc, then it cannot take a non-maintenance night arc.

For each path $\pi = (\alpha_1, \alpha_2, \ldots, \alpha_k) \in \mathscr{D}$, we denote $\theta(\pi)$ the path $(\theta(\alpha_1), \theta(\alpha_2), \ldots, \theta(\alpha_k))$. We call a *maintenance paths partition* of $\mathscr{D}$ a collection Π of arc-disjoint paths in $\mathscr{D}$ such that the set of paths $\theta(\pi)$ for $\pi \in \Pi$ is an S-T path partition of D. We denote $\theta(\Pi)$ this S-T partition. For each state ϑ, we denote $\#^{\Pi}(\vartheta)$ the number of paths in Π that intersects ϑ. We have the following lemma.

Lemma B.4. *Let $\mathscr{D}$ be a routing state graph on an instance $D, N_d, M_d, \kappa_s^o, \gamma_s^o$ of the* AIRPORT-TIME GRAPH ROUTING PROBLEM.

1. *An S-T path $P \in D$ is feasible if and only there exists an $S-T$ path $\pi \in \mathscr{D}$ such that $\theta(\pi) = P$.*
2. *An S-T path partition $\mathscr{P}$ is a feasible routing if and only if there exists a maintenance path partition Π such that for each source s, sink t, and positive integer $o \leq \delta_{\text{maint}}$, we have $\#^{\Pi}(\vartheta_s^o) = \kappa_s^o$, and $\sum_{\tilde{o} \geq o} \gamma_t^{\tilde{o}} \leq \sum_{\tilde{o} \geq o} \#^{\Pi}(\vartheta_t^{\tilde{o}})$.*

Proof. The first point follows from the definition of the maintenance state graph. Indeed, if P is not feasible, any path π such that $\theta(\pi) = P$ would have more than δ_{maint} non-maintenance night arcs without maintenance arcs in-between.

Given a source s (resp. a sink t), an S-T path Π starting in ϑ_s^o (resp. ending in ϑ_t^o) goes through a maintenance night arc at most $\delta_{\text{maint}} - o$ nights after s (resp. has gone through a maintenance night o nights before t). Thus, the hypotheses of the second point implies that $\mathscr{P}$ satisfies the

initial and final conditions, and the first point gives the result. □

B.2.2 Polynomial algorithm for network paths partition problem

This section is devoted to the proof of Theorem 8.3.

Theorem 8.3. *The* AIRCRAFT ROUTING PROBLEM *is polynomial when the number of airplanes k is fixed. It can be solved in time bounded from above by $2|V|\delta^k_{\text{maint}}$, where k is the number of airplanes and n the number of flight legs.*

The proof of Theorem 8.3 consists in a polynomial algorithm inspired of the pebbling game algorithm to solve the integer multicommodity flow problem [79]. In the remaining of the section, three lemmas are introduced to be able to prove Theorem 8.3 at the end of the section.

Given a topological ordering $v_1, v_2, \ldots, v_n$ of the internal vertices I of an equigraph with source set S and sink set T, let U_i be the set of vertices strictly after v_i in the ordering, $U_i = \{v_j \in V | j > i\} \cup T$, and $U_0 = V \setminus S$. As $v_1, v_2, \ldots, v_n$ is a topological ordering, $C_i = \delta^-(U_i)$ is an S-T directed cut. This way, to each topological ordering $v_1, v_2, \ldots, v_n$ corresponds a collection of directed cuts $C_k, C_{k+1}, \ldots, C_{n-k}$ with $C_i = \delta^-(U_i)$ and $V \setminus S = U_k \supsetneq U_{k+1} \supsetneq \cdots \supsetneq U_{n-k} = T$. A topological ordering and its directed cuts collection are illustrated on Figure B.7. Besides, we have:

$$\begin{aligned} U_{i-1} &= \{v_i\} \cup U_i \\ C_{i-1} \setminus C_i &= \delta^-(v_i) \\ C_i \setminus C_{i-1} &= \delta^+(v_i) \end{aligned}$$

Finally, we define the partial graph D_i as the union $\bigcup_{j=0}^{i} C_i$ of the directed cuts C_j whose index j is smaller than i. As all the S-T directed cuts of D_i are S-T directed cuts of D, they have all the same cardinality and Proposition B.1 ensures that D_i is an equigraph. Let $\mathscr{D}_i$ be the partial graph of $\mathscr{D}$ whose arcs are in $\mathscr{A}^i = \cup_{a \in G_i} A_a$. Let T_i be the set of sinks of D_i, and $\mathscr{T}_i$ be the set of sinks of $\mathscr{D}_i$.

A *distribution of pebble* on $\mathscr{T}_i$ is an application $X_i : \mathscr{T}_i \to \mathbb{Z}^+$; it is *reachable* if there exists a maintenance path partition Π_i on $\mathscr{D}_i$ such that $\#^{\Pi_i}(\vartheta) = X_i(\vartheta)$ for each state vertex $\vartheta \in \mathscr{T}_i$. Partition Π_i is *ending in* X_i. Let χ_i be the *set of reachable distributions on* $\mathscr{T}_i$. Let X_i be a reachable distribution in χ_i, and Π_i a corresponding path partition of $\mathscr{D}_i$. A distribution X_{i+1} on $\mathscr{T}_{i+1}$ can be *reached from* X_i if any maintenance path partition Π_i of $\mathscr{D}_i$ ending in X_i can be completed in a maintenance path partition Π_{i+1} on $\mathscr{D}_{i+1}$ ending in X_{i+1}. The idea of the algorithm is to deduce χ_{i+1} from χ_i as the union of the distributions X_{i+1} that can be reached from a distribution $x_i \in \chi_i$ thanks to a legal move of pebbles.

Lemma B.5. *A pebble distribution X_{i+1} on $\mathscr{T}_{i+1}$ is reachable from distribution X_i on $\mathscr{T}_i$ if and only if the following conditions are satisfied:*

1. *Pebbles that are not in $\mathscr{V}_{v_i}$ are unmoved.*

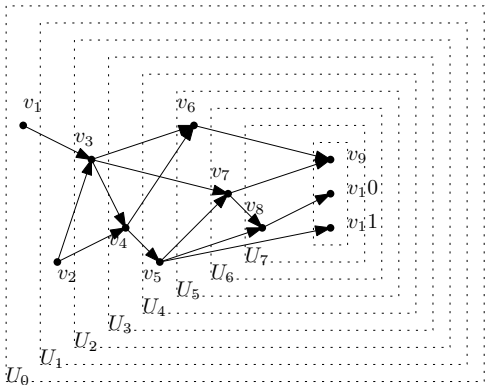


Figure B.7 – A topological ordering and its ordering cuts collection

2. *A pebble initially on vertex ϑ_1 can be moved to vertex ϑ_2 if and only if $(\vartheta_1, \vartheta_2)$ is an arc in $\mathscr{D}$.*
3. *Only one pebble goes through each arc set $\mathscr{A}_a$ with $a \in \delta^+(v)$.*

A pebble move satisfying these conditions is called a legal move. *Let $m(\vartheta)$ be the number of pebbles moved to ϑ, then $X_{i+1}(\vartheta) = X_i(\vartheta) + m(\vartheta)$ if $\vartheta \in \mathscr{T}_i \cap \mathscr{T}_{i+1}$ and $X_{i+1}(\vartheta) = m(\vartheta)$ otherwise.*

Proof. Suppose that a legal move leads from X_i to X_{i+1}. Given a maintenance path partition Π_i ending in X_i, we affect each pebble moved from $\vartheta \in \mathscr{T}_i \setminus \mathscr{T}_{i+1}$ to a path $\pi \in \Pi_i$ ending in ϑ, and complete π by the arc α crossed by its pebble. The completed paths form a maintenance path partition on $\mathscr{D}_{i+1}$ ending in $\mathscr{T}_{i+1}$.

Conversely, if X_{i+1} is reached from X_i in the maintenance path partition Π_{i+1}, then we move one pebble along each arc of $\Pi_{i+1} \cap \left(\cup_{a \in C_i} \mathscr{A}_a\right)$ to obtain a legal move. □

We define $\phi_i : \chi_i \to 2^{\chi_{i+1}}$ as the operator that associates to a reachable distribution X_i the set of distributions X_{i+1} reachable from X_i. Lemma B.5 ensures that $\chi_{i+1} = \cup_{X_i \in \chi_i} \phi_i(X_i)$. Let $\chi = \cup_i \chi_i$ be the set of reachable distributions. Let $\mathscr{I} \subseteq \chi_0$ be the set of distributions of pebbles on the sources such that $X(\vartheta_s^o) = \kappa_s^o$ for each source s and positive integer $o \leq \delta_{\text{maint}}$, and let $\mathscr{J} \subseteq \chi_n$ be the set of distributions on the sinks such that for each sink t, $\sum_{\tilde{o} \geq o} X(\vartheta_t^o) \leq \gamma_t^o$. The acyclic directed *distribution graph* associated to the network state graph is defined as follows: one vertex for each reachable pebble distribution $X \in \chi$, and for each pebble distribution X in χ, one arc between X and each distribution X' in $\phi(X)$.

Lemma B.6. *The* AIRCRAFT ROUTING PROBLEM *admits a solution if and only if there exists an $\mathscr{I}$-$\mathscr{J}$ path in the distribution graph.*

Proof. We first prove that there exists a maintenance path partition of $\mathscr{D}$ if and only if there exists a χ_0-χ_n path in the distributions graph. A path from a source distribution to a sink distribution is obtained from a maintenance path partition by a applying recursively Lemma B.5. Conversely, given a path $\mathscr{D}$ from a source distribution to a sink distribution, let $\mathscr{D}'$ be the partial graph of $\mathscr{D}$ obtained by taking the arcs crossed by a pebble during one of the legal

move of $\mathscr{D}$. This graph is a state equigraph, and thus a path partition can be deduced from it in linear time by applying the greedy path partition algorithm in equigraphs.

The definition of $\mathscr{I}$ and $\mathscr{J}$ ensures that a χ_0-χ_n path in the distribution graph is an $\mathscr{I}$-$\mathscr{J}$ path if and only if its maintenance path partition is such that, for each source s, sink t, and positive integer $o \leq \delta_{\text{maint}}$, we have $\#^{\Pi}(\vartheta_s^o) = \kappa_s^o$, and $\sum_{\bar{o} \geq o} \gamma_t^{\bar{o}} \leq \sum_{\bar{o} \geq o} \#^{\Pi}(\vartheta_t^{\bar{o}})$.

Lemma B.4 then gives the equivalence of the existence of an $\mathscr{I}$-$\mathscr{J}$ path in the distributions graph with the feasibility of the AIRPORT-TIME GRAPH ROUTING PROBLEM. Theorem B.3 enables to conclude. □

Finally, a last lemma on the cardinality of the arc set of the distribution graph is needed to be able to prove Theorem 8.3.

Lemma B.7. *The number of arcs in the distribution graph of a state graph $\mathscr{D}$ is bounded from above by $2|V|\delta_{\text{maint}}^k$, where k is the flow of equigraph D.*

Proof. Let $B_0 = 2|V|\delta_{\text{maint}}^k$. We start by proving that

$$B_1 = \sum_{v \in S \cup I} \left(\prod_{u \in S_v \cap S_{v+1}} \begin{pmatrix} |\mathcal{V}_u| - 1 \\ p_v(u) + |\mathcal{V}_u| - 1 \end{pmatrix} \right) \cdot |\mathcal{V}_v|^{d(v)},$$

and then prove that $B_1 \leq B_0$.

As arcs in the distribution graphs are between χ_i and χ_{i+1}, it suffices to bound the number of legal moves between χ_i and χ_{i+1}. As each network arc $a \in A$ is crossed exactly once, the number of pebbles on vertices in $\mathcal{V}_u$ is identical for each distribution in χ_v. Define $p_v(u)$ as the number of pebbles on state vertices in $\mathcal{V}_u$ in distribution of χ_v. Thus, the number of legal moves is equal to the number of distributions of pebbles on $S_v \cap S_{v+1}$, which correspond to the pebbles which do not move, multiplied by the number of legal moves of the moving pebbles.

For each vertex u in $S_v \cap S_{v+1}$, the number of way to distribute $p_v(u)$ pebbles on $|\mathscr{S}_u|$ states is equal to $\begin{pmatrix} |\mathcal{V}_u| - 1 \\ p_v(u) + |\mathcal{V}_u| - 1 \end{pmatrix}$.

Let $\vartheta_1, \vartheta_2, \ldots, \vartheta_\ell$ be the state vertices in $\mathcal{V}_v$ and x_i be the number of pebbles on ϑ_i in distribution X_i. Then $x_1 + x_2 + \cdots + x_\ell = d(v)$. As, first, in the path partition, each arc of D is covered exactly once and, second, for each network arc, there is at most one state arc outgoing from each state vertex of $\mathcal{V}_v$, the number of legal moves is bounded from above by the number of partitions of the $d(v)$ arcs in $\delta^+(v)$ into sets of cardinal $x_1, x_2, \ldots, x_i$ (which correspond to the origin of the state arcs): $\begin{pmatrix} x_1, x_2, \ldots, x_m \\ d(v) \end{pmatrix}$. Thus, the total number of legal moves is bounded from above by

$$\begin{aligned} & \left(\prod_{u \in S_v \cap S_{v+1}} \begin{pmatrix} |\mathcal{V}_u| - 1 \\ p_v(u) + |\mathcal{V}_u| - 1 \end{pmatrix} \right) \cdot \sum_{x_1 + x_2 + \ldots + x_\ell = d(v)} \begin{pmatrix} x_1, x_2, \ldots, x_\ell \\ d(v) \end{pmatrix} \\ = & \left(\prod_{u \in S_v \cap S_{v+1}} \begin{pmatrix} |\mathcal{V}_u| - 1 \\ p_v(u) + |\mathcal{V}_u| - 1 \end{pmatrix} \right) \cdot l^{d(v)} \end{aligned} \tag{B.4}$$

Summing the upper bounds $|\phi(\chi_i)|$ gives the upper bound B_1 on $|\phi(\chi)|$. $B_1 \leq B_0$ is obtained from $\begin{pmatrix} |\mathcal{V}_u|-1 \\ p_v(u)+|\mathcal{V}_u|-1 \end{pmatrix} \leq |\mathcal{V}_u|^{p_v(u)}$, $\sum_{u\in S_v} p_v(u) = k$, and $\max_{v\in V} |\mathcal{V}_v| \leq \delta_{\text{maint}}$. □

Proof of Theorem 8.3. Lemma B.6 ensures that a path finding algorithm in the distributions graph enables to decide if the AIRCRAFT ROUTING PROBLEM admits a solution. As the distribution graph is acyclic, the complexity of a path finding algorithm in this graph is linear in the number of arcs in the distribution graph. Lemma B.7 then gives the result. □

Remark B.2. A shortest path algorithm in distribution graph solves the optimization version of the AIRCRAFT ROUTING PROBLEM mentioned in Section 8.2.2. As the distribution graph is acyclic, its complexity admits the same bounds B_0 and B_1 as the path finding algorithm.

B.3 Aircraft routing NP-completeness

We now prove Theorem 8.2.

Theorem 8.2. *The* AIRCRAFT ROUTING PROBLEM *is $\mathcal{NP}$-complete for $\delta_{\text{maint}} \geq 3$.*

The TWO-COMMODITIES ARC-DISJOINT PATHS PROBLEM ON ACYCLIC DIGRAPH is $\mathcal{NP}$-complete [69].

TWO-COMMODITIES ARC-DISJOINT PATHS PROBLEM ON ACYCLIC DIGRAPH
Input. A directed acyclic graph $D = (V, A)$, vertices s_1 and s_2 called sources, t_1 and t_2 called terminals, two non negative R_1 and R_2
Output. R_i arc paths forms s_i to t_i for $i = 1, 2$

Proof. The TWO-COMMODITIES ARC-DISJOINT PATHS PROBLEM ON ACYCLIC DIGRAPH can be reduced to the AIRPORT-TIME GRAPH ROUTING PROBLEM. Without restriction, we suppose that $\delta_{\text{maint}} = 3$. Larger δ_{maint} can be dealt with analogously by adding $(\delta_{\text{maint}} - 3)$ non-maintenance night arcs before each source of the instance constructed.

Let $D, s_1, s_2, t_1, t_2, R_1, R_2$ be an instance of the TWO-COMMODITIES ARC-DISJOINT PATHS PROBLEM ON ACYCLIC DIGRAPH. We build the graph D' by extending D in the following way: $2R_1 + 2R_2$ sources vertices are added to form S_1, T_1 and S_2, T_2, an arc is added from each vertex of S_i to s_i and from t_i to each vertex of T_i . Other vertices are internal vertices. If v is such that $\delta^-(v) > \delta^+(v)$, then $\delta^-(v) - \delta^+(v)$ sources are added with one arc between each of these and v, and $\delta^+(v) - \delta^-(v)$ terminal linked to v are added if $\delta^+(v) > \delta^-(v)$. Thus, D is an equigraph. This extension is illustrated on Figure B.8.

We now build night sets $N_d \subseteq A''$, maintenance night sets $M_d \subseteq N_d$ and a digraph $D'' = (V'', A'')$ that extends D' as follows. For each vertex $s \in S'$, we build two vertices u_s, v_s and the two arcs (u_s, v_s) and (v_s, s), and we add these arcs respectively to $_1$ and N_2. If $s \notin S_1 \cup S_2$, then we add both (u_s, v_s) to M_1 and (v_s, s) to M_2. If $s \in S_1$, then we add only (u_s, v_s) to M_1, and if $s \in S_2$, we do not add (u_s, v_s) and (v_s, s) to M_1 or M_2. Symmetrically, for each vertex $t \in T'$, we build two vertices u_t, v_t and the arcs (t, u_t) and (u_t, v_t) and we add these arcs respectively to N_3 and

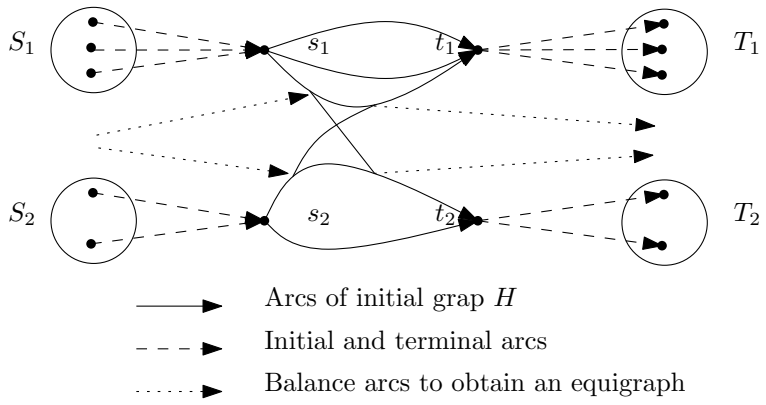


Figure B.8 – Equigraph D as an extension of initial graph H

N_4. If $t \in\notin T_1 \cup T_2$, we do not add (t,u_t) and (u_t,v_t) to M_3 nor to M_4. If $t \in T_1$, then we add (u_t,v_t) to M_4, and if $t \in T_2$, we add both (t,u_t) to M_3 and (u_t,v_t) to M_4. The triple (D'',N,M) defines an instance of the AIRCRAFT ROUTING PROBLEM with no boundary conditions. This construction is also illustrated on Figure 8.3.

Suppose that two-commodities arc-disjoint paths problem on acyclic directed graph admits a solution $\mathscr{P}$. Then, applying Lemma B.2 for each path P of $\mathscr{P}$, $D\backslash\mathscr{P}$ is still an equigraph, and can be covered by arc disjoint paths using the greedy algorithm relying on Lemma B.2. Thus, $\mathscr{P}$ can be extended to an equigraph cover of D with R_i paths from S_i to T_i. Besides, the definition of $M_1,\ldots,M_4$ ensures that each path of the partition satisfies the maintenance constraint, and thus the partition is a solution to AIRPORT-TIME GRAPH ROUTING PROBLEM.

Conversely, suppose that AIRPORT-TIME GRAPH ROUTING PROBLEM admits a solution $\mathscr{P}$. Then $\mathscr{P} \cap H$ is a solution to the two-commodities arc-disjoint paths problem. Indeed, any path going through S_2 has to go through T_2 to be satisfy the maintenance constraint, which implies than any path going through S_1 has to end in T_1 to satisfy the maintenance constraint. □

C LatticeRCSP library

This appendix describes the main features of the `latticeRCSP` library we have developed during this thesis. This templated `C++` library implements the MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM framework of Part I. The numerical experiments in Chapters 4 to 6, 9 and 10 are performed using this library.

The library relies on several families of classes.

- Resource classes enable to implement lattice ordered monoids $(\mathscr{R}, \oplus, \leqslant)$. The attributes of an object of a resource class contain the information that identifies a resource q of the monoid $\mathscr{R}$. The methods of the class describe the operators $\oplus$, $\leqslant$, $\wedge$, and $\vee$ of the lattice ordered monoid $(\mathscr{R}, \oplus, \leqslant)$, as well as its smallest, largest and neutral element.

  These resource classes can be pretty elaborated. For instance, the resource class which implements the lattice ordered monoid of discrete distributions with finite support of Section 5.1 performs a convolution product for each sum. To speed-up the computation, this convolution product is done using a fast fourier transform using the `kissFFT` library [29].
- Classes implementing cost functions c and infeasibility functions ρ. The methods of these classes take in input a resource q and return respectively $c(q)$ and $\rho(q)$.
- Template classes implementing digraph $D = (V, A)$ with origin o, destination d and resource q_a for each arc a. The attributes of this class contain the bounds b_v on v-d paths resources in addition to the elements already mentioned. The methods of the class contain the algorithms computing bounds of Section 4.2.
- Template classes that enable to build and store the state graphs of Chapter 6.
- Template classes that implement MONOID RESOURCE CONSTRAINED SHORTEST PATH PROBLEM problems. The template arguments of a problem are: the resource class of the lattice ordered monoid, the cost function, and the infeasibility function. The attributes of the class are a digraph class and a set of solution to the problem.
- Template classes that implement the enumeration algorithms. These classes take in input a problem, and their method implement the algorithm of Section 4.1.

To tackle with a new problem, the only classes that must be implemented are the resource, the cost function and the infeasibility function. The procedure to solve the problem is then standardized: the following step are executed.

1. Build the digraph.
2. Build the lower bounds b_v on v-d paths or build a state graph and compute its bounds.
3. Build the problem.
4. Select an enumeration algorithm and solve the problem using this enumeration algorithm.

In addition to the main blocks described, several heuristics are implemented. These include notably the use of the mapping ϕ mentionned in Remark 4.5 to speed-up the generalized Dijkstra algorithm, the heuristics to find good quality solutions that can be used as candidate paths, as mentioned in Section 4.3.3. To build clustering state graphs, we have implemented the heuristics mentioned in Section 6.2.4.

Bibliography

[1] Boost resource constrained shortest path library. URL http://www.boost.org/doc/libs/1_58_0/libs/graph/doc/r_c_shortest_paths.html.

[2] 2006. URL http://www.dis.uniroma1.it/challenge9/.

[3] Carlo Acerbi. Coherent representations of subjective risk aversion. *Risk Measures for the 21st Century*, page 147, 2003.

[4] Carlo Acerbi and Prospero Simonetti. Portfolio optimization with spectral measures of risk. *arXiv preprint cond-mat/0203607*, 2002.

[5] Yossiri Adulyasak and Patrick Jaillet. Models and algorithms for stochastic and robust vehicle routing with deadlines. *Transportation Science*, 50(2):608–626, 2015.

[6] Yana Ageeva and John-Paul Clarke. Approaches to incorporating robustness into airline scheduling. *ICAT - Reports and Paper*, 2000.

[7] Shervin Ahmadbeygi, Amy Cohn, and Marcial Lapp. Decreasing airline delay propagation by re-allocating scheduled slack. *IIE Transactions*, 42(7):478–489, 2010.

[8] Alfred V Aho and John E Hopcroft. *The design and analysis of computer algorithms*. Pearson Education India, 1974.

[9] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. Network flows: theory, algorithms, and applications. 1993.

[10] Mohamed Ali Aloulou, Mohamed Haouari, and Farah Zeghal Mansour. Robust aircraft routing and flight retiming. *Electronic Notes in Discrete Mathematics*, 36:367–374, 2010.

[11] Gianluca Amato, Francesca Scozzari, Helmut Seidl, Kalmer Apinis, and Vesal Vojdani. Efficiently intertwining widening and narrowing. *Science of Computer Programming*, 120:1–24, 2016.

[12] Ranga Anbil, Rajan Tanga, and Ellis L. Johnson. A global approach to crew-pairing optimization. *IBM Systems Journal*, 31(1):71–78, 1992.

[13] Kalmer Apinis, Helmut Seidl, and Vesal Vojdani. How to combine widening and narrowing for non-monotonic systems of equations. *ACM SIGPLAN Notices*, 48(6):377–386, 2013.

[14] Krzysztof Apt. *Principles of constraint programming*. Cambridge University Press, 2003.

[15] Philippe Artzner, Freddy Delbaen, Jean-Marc Eber, and David Heath. Coherent measures of risk. *Mathematical finance*, 9(3):203–228, 1999.

[16] François Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat. *Synchronization and linearity: an algebra for discrete event systems*. John Wiley & Sons Ltd, 1992.

[17] Roland C Backhouse and Bernard A Carré. Regular algebra applied to path-finding problems. *IMA Journal of Applied Mathematics*, 15(2):161–186, 1975.

[18] Roberto Baldacci, Nicos Christofides, and Aristide Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115(2):351–385, 2008.

[19] Michael Ball and Anito Roberts. A graph partitioning approach to airline crew scheduling. *Transportation Science*, 19(2):107–126, 1985.

[20] Cynthia Barnhart, Natashia L Boland, Lloyd W Clarke, Ellis L Johnson, George L Nemhauser, and Rajesh G Shenoi. Flight string models for aircraft fleeting and routing. *Transportation Science*, 32(3):208–220, 1998.

[21] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato Werneck. *Route Planning in Transportation Networks*, 2014.

[22] Nicole Bäuerle and Alfred Müller. Stochastic orders and risk measures: consistency and bounds. *Insurance: Mathematics and Economics*, 38(1):132–148, 2006.

[23] JE Beasley and Nicos Christofides. An algorithm for the resource constrained shortest path problem. *Networks*, 19(4):379–394, 1989.

[24] John E Beasley and B Cao. A tree search algorithm for the crew scheduling problem. *European Journal of Operational Research*, 94(3):517–526, 1996.

[25] Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, pages 87–90, 1958.

[26] Dimitris J Bertsimas. A vehicle routing problem with stochastic demand. *Operations Research*, 40(3):574–585, 1992.

[27] Dimitris J Bertsimas and David Simchi-Levi. A new generation of vehicle routing research: robust algorithms, addressing uncertainty. *Operations Research*, 44(2):286–304, 1996.

[28] Thomas Scott Blyth. *Lattices and ordered algebraic structures*, volume 1. Springer, 2005.

[29] Mark Bogerding. Kissfft library, 2013. URL http://sourceforge.net/projects/kissfft/.

[30] Ralf Borndörfer, Martin Grötschel, and Andreas Löbel. Scheduling duties by adaptive column generation. 2001.

[31] Olivier Briant, Claude Lemaréchal, Ph Meurdesoif, Sophie Michel, Nancy Perrot, and François Vanderbeck. Comparison of bundle and classical column generation. *Mathematical programming*, 113(2):299–344, 2008.

[32] W Matthew Carlyle, Johannes O Royset, and R Kevin Wood. Lagrangian relaxation and enumeration for solving constrained shortest-path problems. *Networks*, 52(4):256–270, 2008.

[33] Pierre Carpentier, Jean-Philippe Chancelier, Guy Cohen, and DE Michel. *Stochastic Multi-Stage Optimization: At the Crossroads between Discrete Time Stochastic Control and Stochastic Programming*, volume 75. Springer, 2015.

[34] Bernard A Carré. An algebra for network routing problems. *IMA Journal of Applied Mathematics*, 7(3):273–294, 1971.

[35] Tsung-Sheng Chang, Yat-wah Wan, and Wei Tsang Ooi. A stochastic dynamic traveling salesman problem with hard time windows. *European Journal of Operational Research*, 198(3):748–759, 2009.

[36] Anthony Chen and Zhaowang Ji. Path finding under uncertainty. *Journal of advanced transportation*, 39(1):19–37, 2005.

[37] Bi Yu Chen, William HK Lam, Agachai Sumalee, Qingquan Li, Hu Shao, and Zhixiang Fang. Finding reliable shortest paths in road networks under uncertainty. *Networks and spatial economics*, 13(2):123–148, 2013.

[38] Boris V Cherkassky, Andrew V Goldberg, and Tomasz Radzik. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical programming*, 73(2):129–174, 1996.

[39] Hai D Chu, Eric Gelman, and Ellis L Johnson. Solving large scale crew scheduling problems. *European Journal of Operational Research*, 97(2):260–268, 1997.

[40] John-Paul Clarke, Terran Melconian, Elizabeth Bly, and Fabio Rabbani. Means — MIT extensible air network simulation. *Simulation*, 83(5):385–399, 2007.

[41] Lloyd Clarke, Ellis Johnson, George Nemhauser, and Zhongxi Zhu. The aircraft rotation problem. *Annals of Operations Research*, 69:33–46, 1997.

[42] Amy Mainville Cohn and Cynthia Barnhart. Improving crew scheduling by incorporating key maintenance routing decisions. *Operations Research*, 51(3):387–396, 2003.

[43] Jean-François Cordeau, Goran Stojković, François Soumis, and Jacques Desrosiers. Benders decomposition for simultaneous aircraft routing and crew scheduling. *Transportation science*, 35(4):375–388, 2001.

[44] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252. ACM, 1977.

[45] Patrick Cousot and Radhia Cousot. Constructive versions of tarski's fixed point theorems. *Pacific Journal of Mathematics*, 82(1):43–57, 1979.

[46] Pilu Crescenzi, Roberto Grossi, Michel Habib, Leonardo Lanzi, and Andrea Marino. On computing the diameter of real-world undirected graphs. *Theoretical Computer Science*, 514:84–95, 2013.

[47] Brian A Davey and Hilary A Priestley. *Introduction to lattices and order*. Cambridge university press, 2002.

[48] Amal De Silva. Combining constraint programming and linear programming on an example of bus driver scheduling. *Annals of Operations Research*, 108(1-4):277–291, 2001.

[49] Dieter Denneberg. Distorted probabilities and insurance premiums. *Methods of Operations Research*, 63(3), 1990.

[50] Olivier Deprez and Hans U Gerber. On convex principles of premium calculation. *Insurance: Mathematics and Economics*, 4(3):179–189, 1985.

[51] Guy Desaulniers, J Desrosiers, Y Dumas, S Marc, B Rioux, Marius M Solomon, and Francios Soumis. Crew pairing at air france. *European Journal of Operational Research*, 97(2):245–259, 1997.

[52] Guy Desaulniers, Jacques Desrosiers, Yvan Dumas, Marius M Solomon, and François Soumis. Daily aircraft routing and scheduling. *Management Science*, 43(6):841–855, 1997.

[53] Martin Desrochers. *La fabrication d'horaires de travail pour les conducteurs d'autobus par une méthode de génération de colonnes*. Université de Montréal, Centre de recherche sur les transports, 1987.

[54] Martin Desrochers and Québec). Groupe d'études et de recherche en analyse des décisions École des hautes études commerciales (Montréal. *An algorithm for the shortest path problem with resource constraints*. Montréal: École des hautes études commerciales, 1988.

[55] Martin Desrochers and François Soumis. A generalized permanent labeling algorithm for the shortest path problem with time windows. *INFOR Information Systems and Operational Research*, 1988.

[56] Jacques Desrosiers, Paul Pelletier, and François Soumis. Plus court chemin avec contraintes d'horaires. *Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle*, 17(4):357–377, 1983.

[57] Jacques Desrosiers, François Soumis, and Martin Desrochers. Routing with time windows by column generation. *Networks*, 14(4):545–565, 1984.

[58] Jacques Desrosiers, Yvan Dumas, Marius M Solomon, and François Soumis. Time constrained routing and scheduling. *Handbooks in operations research and management science*, 8:35–139, 1995.

[59] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[60] Olivier Du Merle, Daniel Villeneuve, Jacques Desrosiers, and Pierre Hansen. Stabilized column generation. *Discrete Mathematics*, 194(1):229–237, 1999.

[61] Irina Dumitrescu and Natashia Boland. Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem. *Networks*, 42(3):135–153, 2003.

[62] Michelle Dunbar, Gary Froyland, and Cheng-Lung Wu. Robust airline schedule planning: minimizing propagated delay in an integrated routing and crewing framework. *Transportation Science*, 46(2):204–216, 2012.

[63] Michelle Dunbar, Gary Froyland, and Cheng-Lung Wu. An integrated scenario-based approach for robust aircraft routing, crew pairing and re-timing. *Computers & Operations Research*, 45:68–86, 2014.

[64] Matthias Ehrgott and David Ryan. Constructing robust crew schedules with bicriteria optimization. *Journal of Multi-Criteria Decision Analysis*, 11(3):139–150, 2002.

[65] Amir Eiger, Pitu B Mirchandani, and Hossein Soroush. Path preferences and optimal paths in probabilistic networks. *Transportation Science*, 19(1):75–84, 1985.

[66] Issmail Elhallaoui, Daniel Villeneuve, François Soumis, and Guy Desaulniers. Dynamic aggregation of set-partitioning constraints in column generation. *Operations Research*, 53(4):632–645, 2005.

[67] Issmail Elhallaoui, Abdelmoutalib Metrane, François Soumis, and Guy Desaulniers. Multi-phase dynamic constraint aggregation for set partitioning type problems. *Mathematical Programming*, 123(2):345–370, 2010.

[68] David Eppstein. Finding the k shortest paths. *SIAM Journal on computing*, 28(2): 652–673, 1998.

[69] Shimon Even, Alon Itai, and Adi Shamir. On the complexity of time table and multicommodity flow problems. In *Foundations of Computer Science, 1975., 16th Annual Symposium on*, pages 184–193. IEEE, 1975.

[70] Torsten Fahle, Ulrich Junker, Stefan E Karisch, Niklas Kohl, Meinolf Sellmann, and Bo Vaaben. Constraint programming based column generation for crew assignment. *Journal of Heuristics*, 8(1):59–81, 2002.

[71] Yueyue Fan and Yu Nie. Optimal routing for maximizing the travel time reliability. *Networks and Spatial Economics*, 6(3-4):333–344, 2006.

[72] Dominique Feillet, Pierre Dejax, Michel Gendreau, and Cyrille Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229, 2004.

[73] Thomas A Feo and Jonathan F Bard. Flight scheduling and maintenance base planning. *Management Science*, 35(12):1415–1432, 1989.

[74] Eugene Fink. A survey of sequential and systolic algorithms for the algebraic path problem. 1992.

[75] Arthur Flajolet, Sébastien Blandin, and Patrick Jaillet. Robust adaptive routing under uncertainty. *arXiv preprint arXiv:1408.3374*, 2014.

[76] Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

[77] Lester R Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8(3):399–404, 1956.

[78] Lester R Ford Jr and Delbert Ray Fulkerson. Constructing maximal dynamic flows from static flows. *Operations research*, 6(3):419–433, 1958.

[79] Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980.

[80] Nicolas Fournier and Arnaud Guillin. On the rate of convergence in wasserstein distance of the empirical measure. *Probability Theory and Related Fields*, pages 1–32, 2014.

[81] H Frank. Shortest paths in probabilistic graphs. *Operations Research*, 17(4):583–599, 1969.

[82] Liping Fu. An adaptive routing algorithm for in-vehicle route guidance systems with real-time information. *Transportation Research Part B: Methodological*, 35(8):749–765, 2001.

[83] Liping Fu and Larry R Rilett. Expected shortest paths in dynamic and stochastic traffic networks. *Transportation Research Part B: Methodological*, 32(7):499–516, 1998.

[84] Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman, 2002.

[85] Gerhard Gierz, Karl Heinrich Hofmann, Klaus Keimel, Jimmie D Lawson, Michael Mislove, and Dana S Scott. *Continuous lattices and domains*, volume 93. Cambridge University Press, 2003.

[86] Michel Gondran and Michel Minoux. *Graphs, dioids and semirings: new models and algorithms*, volume 41. Springer Science & Business Media, 2008.

[87] Balaji Gopalakrishnan and Ellis Johnson. Airline crew scheduling: State-of-the-art. *Annals of Operations Research*, 140:305–337, 2005.

[88] Ram Gopalan and Kalyan T Talluri. The aircraft maintenance routing problem. *Operations Research*, 46(2):260–271, 1998.

[89] Chrysanthos E Gounaris, Wolfram Wiesemann, and Christodoulos A Floudas. The robust capacitated vehicle routing problem under demand uncertainty. *Operations Research*, 61(3):677–693, 2013.

[90] Stefano Gualandi and Federico Malucelli. Constraint programming-based column generation. *4OR*, 7(2):113–137, 2009.

[91] Randolph W Hall. The fastest path through a network with random time-dependent travel times. *Transportation science*, 20(3):182–188, 1986.

[92] Gabriel Y Handler and Israel Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10(4):293–309, 1980.

[93] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[94] Karla L Hoffman and Manfred Padberg. Solving airline crew scheduling problems by branch-and-cut. *Management Science*, 39(6):657–682, 1993.

[95] Irina Ioachim, Sylvie Gelinas, Francois Soumis, and Jacques Desrosiers. A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. *Networks*, 31(3):193–204, 1998.

[96] Stefan Irnich. Resource extension functions: Properties, inversion, and generalization to segments. *OR Spectrum*, 30(1):113–148, 2008.

[97] Stefan Irnich and Guy Desaulniers. *Shortest path problems with resource constraints*. Springer, 2005.

[98] Stefan Irnich and Daniel Villeneuve. The shortest-path problem with resource constraints and k-cycle elimination for k ≥ 3. *INFORMS Journal on Computing*, 18(3): 391–406, 2006.

[99] Patrick Jaillet. A priori solution of a traveling salesman problem in which a random subset of the customers are visited. *Operations research*, 36(6):929–936, 1988.

[100] Patrick Jaillet and A Odoni. The probabilistic vehicle routing problem. *Vehicle routing: methods and studies. North Holland, Amsterdam*, 1988.

[101] Patrick Jaillet, Jin Qi, and Melvyn Sim. Routing optimization under uncertainty. *Operations Research*, 2016.

[102] Hans C Joksch. The shortest route problem with constraints. *Journal of Mathematical analysis and applications*, 14(2):191–197, 1966.

[103] Hossein Jula, Maged Dessouky, and Petros A Ioannou. Truck route planning in nonstationary stochastic networks with time windows at customer locations. *IEEE Transactions on Intelligent Transportation Systems*, 7(1):51–62, 2006.

[104] Ulrich Junker, Stefan E Karisch, Niklas Kohl, Bo Vaaben, Torsten Fahle, and Meinolf Sellmann. A framework for constraint programming based column generation. In *International Conference on Principles and Practice of Constraint Programming*, pages 261–274. Springer, 1999.

[105] Diego Klabjan and Karsten Schwan. Airline crew pairing generation in parallel. Technical report, Technical Report TLI/LEC-99-09, Georgia Institute of Technology, Atlanta, GA, 1999.

[106] Diego Klabjan, Ellis L Johnson, George L Nemhauser, Eric Gelman, and Srini Ramaswamy. Airline crew scheduling with time windows and plane-count constraints. *Transportation Science*, 36(3):337–348, 2002.

[107] Niklas Kohl, Jacques Desrosiers, Oli BG Madsen, Marius M Solomon, and Francois Soumis. 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, 33(1):101–116, 1999.

[108] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[109] Stefanie Kosuch and Abdel Lisser. Stochastic shortest path problem with delay excess penalty. *Electronic Notes in Discrete Mathematics*, 36:511–518, 2010.

[110] Jean Kuntzmann. Théorie des réseaux. 1972.

[111] Shigeo Kusuoka. On law invariant coherent risk measures. In *Advances in mathematical economics*, pages 83–95. Springer, 2001.

[112] Shan Lan, John-Paul Clarke, and Cynthia Barnhart. Planning for robust airline operations: Optimizing aircraft routings and flight departure times to minimize passenger disruptions. *Transportation Science*, 40(1):15–28, 2006.

[113] Jesper Larsen. *Parallelization of the vehicle routing problem with time windows*. PhD thesis, Technical University of DenmarkDanmarks Tekniske Universitet, Department of Informatics and Mathematical ModelingInstitut for Informatik og Matematisk Modellering, 1999.

[114] Sylvie Lavoie, Michel Minoux, and Edouard Odier. A new approach for crew pairing problems by column generation with an application to air transportation. *European Journal of Operational Research*, 35(1):45–58, 1988.

[115] Daniel J Lehmann. Algebraic structures for transitive closure. *Theoretical Computer Science*, 4(1):59–76, 1977.

[116] David Levine. Application of a hybrid genetic algorithm to airline crew scheduling. *Computers & Operations Research*, 23(6):547–558, 1996.

[117] Xiangyong Li, Peng Tian, and Stephen CH Leung. Vehicle routing problems with time windows and stochastic travel and service times: Models and algorithm. *International Journal of Production Economics*, 125(1):137–145, 2010.

[118] Stuart P Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982.

[119] Ronald Prescott Loui. Optimal paths in graphs with stochastic or multidimensional weights. *Communications of the ACM*, 26(9):670–676, 1983.

[120] Leonardo Lozano and Andrés L Medaglia. On an exact method for the constrained shortest path problem. *Computers & Operations Research*, 40(1):378–384, 2013.

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

[122] Stephen Maher, Guy Desaulniers, and François Soumis. Recoverable robust single day aircraft maintenance routing problem. *Computers & Operations Research*, 51:130–145, 2014.

[123] Lilit Mazmanyan, Dan Trietsch, and KR Baker. Stochastic traveling salesperson models with safety time. Technical report, Working paper, 2009.

[124] Anne Mercier and François Soumis. An integrated aircraft routing, crew scheduling and flight retiming model. *Computers & Operations Research*, 34(8):2251–2265, 2007.

[125] Anne Mercier, Jean-François Cordeau, and François Soumis. A computational study of benders decomposition for the integrated aircraft routing and crew scheduling problem. *Computers & Operations Research*, 32(6):1451–1476, 2005.

[126] Elise Miller-Hooks and Hani Mahmassani. Path comparisons for a priori and time-adaptive decisions in stochastic, time-varying networks. *European Journal of Operational Research*, 146(1):67–82, 2003.

[127] Elise D Miller-Hooks and Hani S Mahmassani. Least possible time paths in stochastic, time-varying networks. *Computers & operations research*, 25(12):1107–1125, 1998.

[128] Elise Deborah Miller-Hooks. *Optimal routing in time-varying, stochastic networks: algorithms and implementations*. The University of Texas at Austin, 1997.

[129] Pitu B Mirchandani. Shortest distance and reliability of probabilistic networks. *Computers & Operations Research*, 3(4):347–355, 1976.

[130] Mehryar Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002.

[131] Alfred Müller and Dietrich Stoyan. *Comparison methods for stochastic models and risks*, volume 389. Wiley, 2002.

[132] Ishwar Murthy and Sumit Sarkar. A relaxation-based pruning technique for a class of stochastic shortest path problems. *Transportation Science*, 30(3):220–236, 1996.

[133] Ishwar Murthy and Sumit Sarkar. Stochastic shortest path problems with piecewise-linear concave utility functions. *Management Science*, 44(11-part-2):S125–S136, 1998.

[134] İbrahim Muter, Şİlker Birbil, Kerem Bülbül, Güvenç Şahin, Hüsnü Yenigün, Duygu Taş, and Dilek Tüzün. Solving a robust airline crew pairing problem with column generation. *Computers & Operations Research*, 40(3):815–830, 2013.

[135] Yu Nie and Yueyue Fan. Arriving-on-time problem: discrete algorithm that ensures convergence. *Transportation Research Record: Journal of the Transportation Research Board*, 1964(1):193–200, 2006.

[136] Yu Marco Nie and Xing Wu. Shortest path problem considering on-time arrival probability. *Transportation Research Part B: Methodological*, 43(6):597–613, 2009.

[137] Yu Marco Nie, Xing Wu, and Tito Homem-de Mello. Optimal path problems with second-order stochastic dominance constraints. *Networks and Spatial Economics*, 12 (4):561–587, 2012.

[138] Evdokia Nikolova. High-performance heuristics for optimization in stochastic traffic engineering problems. In *Large-Scale Scientific Computing*, pages 352–360. Springer, 2010.

[139] Evdokia Nikolova, Jonathan A Kelner, Matthew Brand, and Michael Mitzenmacher. Stochastic shortest paths via quasi-convex maximization. In *Algorithms–ESA 2006*, pages 552–563. Springer, 2006.

[140] Nikolaos Papadakos. Integrated airline scheduling. *Computers & Operations Research*, 36(1):176–195, 2009.

[141] Axel Parmentier. Algorithms for non-linear and stochastic resource constrained shortest paths. *arXiv preprint arXiv:1504.07880*, 2015.

[142] Axel Parmentier and Frédéric Meunier. Stochastic shortest paths and risk measures. *arXiv preprint arXiv:1408.0272*, 2014.

[143] Georg Ch Pflug and Alois Pichler. *Multistage Stochastic Optimization*. Springer, 2014.

[144] Alois Pichler. *Distance of probability measures and respective continuity properties of acceptability functionals*. PhD thesis, uniwien, 2010.

[145] Alois Pichler. Evaluations of risk measures for different probability measures. *SIAM Journal on Optimization*, 23(1):530–551, 2013.

[146] Warren B Powell and Zhi-Long Chen. A generalized threshold algorithm for the shortest path problem with time windows. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 40:303–318, 1998.

[147] Giovanni Righini and Matteo Salani. Symmetry helps: bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, 3(3):255–273, 2006.

[148] Giovanni Righini and Matteo Salani. Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. *Computers & Operations Research*, 36(4):1191–1203, 2009.

[149] R Tyrrell Rockafellar and Stanislav Uryasev. Optimization of conditional value-at-risk. *Journal of risk*, 2:21–42, 2000.

[150] Jay M Rosenberger, Ellis L Johnson, and George L Nemhauser. A robust fleet-assignment model with hub isolation and short cycles. *Transportation Science*, 38(3):357–368, 2004.

[151] Louis-Martin Rousseau, Michel Gendreau, Gilles Pesant, and Filippo Focacci. Solving vrptws with constraint programming based column generation. *Annals of Operations Research*, 130(1-4):199–216, 2004.

[152] Bernard Roy. Transitivité et connexité. *Comptes Rendus Hebdomadaires Des Seances De L Academie Des Sciences*, 249(2):216–218, 1959.

[153] RA Russell and TL Urban. Vehicle routing with soft time windows and erlang travel times. *Journal of the Operational Research Society*, 59(9):1220–1228, 2008.

[154] Guillaume Sabran, Samitha Samaranayake, and Alexandre M Bayen. Precomputation techniques for the stochastic on-time arrival problem. In *ALENEX*, pages 138–146. SIAM, 2014.

[155] Mohammed Saddoune, Guy Desaulniers, Issmail Elhallaoui, and François Soumis. Integrated airline crew pairing and crew assignment by dynamic constraint aggregation. *Transportation Science*, 46(1):39–55, 2012.

[156] Samitha Samaranayake, Sebastien Blandin, and A Bayen. A tractable class of algorithms for reliable routing in stochastic networks. *Transportation Research Part C: Emerging Technologies*, 20(1):199–217, 2012.

[157] Luis Santos, João Coutinho-Rodrigues, and John R Current. An improved solution algorithm for the constrained shortest path problem. *Transportation Research Part B: Methodological*, 41(7):756–771, 2007.

[158] Andrew Schaefer, Ellis Johnson, Anton Kleywegt, and George Nemhauser. Airline crew scheduling under uncertainty. *Transportation Science*, 39(3):340–348, 2005.

[159] Bernd Schroeder. *Ordered Sets: An Introduction*. Springer Science & Business Media, 2003.

[160] Moshe Shaked and J George Shanthikumar. *Stochastic orders*. Springer, 2007.

[161] Shengzhi Shao, Hanif D Sherali, and Mohamed Haouari. A novel model and decomposition approach for the integrated airline fleet assignment, aircraft routing, and crew pairing problem. *Transportation Science*, 2015.

[162] Sergey Shebalov and Diego Klabjan. Robust airline crew pairing: Move-up crews. *Transportation Science*, 40(3):300–312, 2006.

[163] Raj A Sivakumar and Rajan Batta. The variance-constrained shortest path problem. *Transportation Science*, 28(4):309–316, 1994.

[164] Ilgaz Sungur, Fernando Ordónez, and Maged Dessouky. A robust optimization approach for the capacitated vehicle routing problem with demand uncertainty. *IIE Transactions*, 40(5):509–523, 2008.

[165] Kalyan T Talluri. The four-day aircraft maintenance routing problem. *Transportation Science*, 32(1):43–53, 1998.

[166] D Taş, M Gendreau, N Dellaert, Tom Van Woensel, and AG De Kok. Vehicle routing with soft time windows and stochastic travel times: A column generation and branch-and-price solution approach. *European Journal of Operational Research*, 236(3):789–799, 2014.

[167] Erlendur S Thorsteinsson. Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. In *International Conference on Principles and Practice of Constraint Programming*, pages 16–30. Springer, 2001.

[168] Pamela H Vance, Cynthia Barnhart, Ellis L Johnson, and George L Nemhauser. Airline crew scheduling: A new formulation and decomposition algorithm. *Operations Research*, 45(2):188–200, 1997.

[169] Shaun S Wang, Virginia R Young, and Harry H Panjer. Axiomatic characterization of insurance prices. *Insurance: Mathematics and economics*, 21(2):173–183, 1997.

[170] Stephen Warshall. A theorem on boolean matrices. *Journal of the ACM (JACM)*, 9(1): 11–12, 1962.

[171] Oliver Weide, David Ryan, and Matthias Ehrgott. An iterative approach to robust and integrated aircraft routing and crew scheduling. *Computers & Operations Research*, 37 (5):833–844, 2010.

[172] Cheng-Lung Wu. Improving airline network robustness and operational reliability by sequential optimisation algorithms. *Networks and Spatial Economics*, 6(3-4):235–251, 2006.

[173] Chiwei Yan and Jerry Kung. Robust aircraft routing. *Available at SSRN 2518028*, 2014.

[174] Joyce Yen and John Birge. A stochastic programming approach to the airline crew scheduling problem. *Transportation Science*, 40(1):3–14, 2006.

[175] Michael Yoeli. A note on a generalization of boolean matrix theory. *The American Mathematical Monthly*, 68(6):552–557, 1961.

[176] Gang Yu and Jian Yang. On the robust shortest path problem. *Computers & Operations Research*, 25(6):457–468, 1998.

[177] FM Zeghal and Michel Minoux. Modeling and solving a crew assignment problem in air transportation. *European Journal of Operational Research*, 175(1):187–209, 2006.

[178] Uwe Zimmermann. *Linear and Combinatorial Optimization in Ordered Algebraic Structures*. Elsevier, 1981.

Index