Une approche polyédrale pour le problème du double voyageur de commerce sous contraintes de piles et pour les ordres lexicographiques

Michele Barbato

Université Paris 13

École Doctorale Galilée

THÈSE

présentée par

Michele BARBATO

pour obtenir le grade de
DOCTEUR D'UNIVERSITÉ
Spécialité: INFORMATIQUE

A Polyhedral Approach for the Double TSP with Multiple Stacks and Lexicographical Orders

soutenue publiquement le 05 octobre 2016 devant le jury :

R.	Wolfler Calvo	Directeur de Thèse
R.	Grappe	Co-encadrant
M.	Lacroix	Co-encadrant
A.R.	Mahjoub	Président de jury
M.	Iori	Rapporteur
F.	Meunier	Rapporteur
L.E.	Gouveia	Examinateur
F.	Roupin	Examinateur

Université Paris 13

École Doctorale Galilée

THÈSE

présentée par

Michele BARBATO

pour obtenir le grade de
DOCTEUR D'UNIVERSITÉ
Spécialité: INFORMATIQUE

A Polyhedral Approach for the Double TSP with Multiple Stacks and Lexicographical Orders

soutenue publiquement le 05 octobre 2016 devant le jury :

R.	Wolfler Calvo	Directeur de Thèse
R.	Grappe	Co-encadrant
M.	Lacroix	Co-encadrant
A.R.	Mahjoub	Président de jury
M.	Iori	Rapporteur
F.	Meunier	Rapporteur
L.E.	Gouveia	Examinateur
F.	Roupin	Examinateur

Résumé

Dans cette thèse nous considérons deux problèmes d'optimisation combinatoire.

Le premier s'appelle problème du double voyageur de commerce avec contraintes de piles. Dans ce problème, un véhicule doit ramasser un certain nombre d'objets dans une région pour les livrer à des clients situés dans une autre région. Lors du ramassage, les objets sont stockés dans les différentes piles du véhicule et la livraison des objets se fait selon une politique de type last-in-first-out. Le ramassage et la livraison consistent chacune en une tournée Hamiltonienne effectuée par le véhicule dans la région correspondante. Nous donnons une formulation linéaire en nombres entiers pour ce problème. Elle est basée sur des variables de précédence et sur des contraintes de chemins infaisables. Nous donnons par la suite des résultats polyédraux sur l'enveloppe convexe des solutions de notre formulation. En particulier, nous montrons des liens forts avec un polytope associé au problème du voyageur de commerce et des liens avec un polytope de type *set covering*. Cette étude polyédrale nous permet de renforcer la formulation initiale et de développer un algorithme de coupes et branchements efficace.

Le deuxième problème que nous considérons consiste à trouver la description des polytopes lexicographiques. Ces derniers sont les enveloppes convexes des points entiers lexicographiquement compris entre deux points entiers fixés. Nous donnons une description complète de ces polytopes en terme d'inégalités linéaires. Nous démontrons que la famille des polytopes lexicographiques est fermée par intersection.

Mots clés: problème du double voyageur de commerce avec contraintes de piles, étude polyédrale, polytope, inégalité valide, *set covering*, problème de séparation, algorithme de coupes et branchements, polytope lexicographique.

Abstract

In this thesis we consider two problems arising in combinatorial optimization.

The first one is the double traveling salesman problem with multiple stacks. In this problem a vehicle picks up a given set of items in a region and subsequently delivers them to demanding customers in another region. When an item is picked up, it is put in a stack of the vehicle. The items are delivered observing a last-in-first-out policy. The pickup phase and the delivery phase consist in two Hamiltonian circuits, each performed by the vehicle in the corresponding region. We give a new integer linear programming formulation for this problem. Its main features are the presence of precedence variables and new infeasible path constraints. We provide polyhedral results on the convex hull of the solutions to our formulation. In particular, we show strong links with a specific TSP polytope and a specific set covering polytope. We deduce strengthening inequalities for the initial formulation, subsequently embedded in an efficient branch-and-cut algorithm.

The second problem we consider consists in finding the description of the lexicographical polytopes. These are convex hulls of the integer points lexicographically between two given integer points. We give a complete description of these polytopes by means of linear inequalities. We show that the lexicographical polytope family is closed under intersection.

Keywords: double traveling salesman problem with multiple stacks, polyhedral study, polytope, valid inequality, set covering, separation problem, branch-and-cut algorithm, lexicographical polytope.

Remerciements

Je tiens à remercier tout d'abord mes pères scientifiques Roberto Wolfler Calvo, Mathieu Lacroix et Roland Grappe pour m'avoir accepté en thèse et pour m'avoir guidé au cours de ces quatre années. Ce manuscrit est surtout le fruit des enseignements que Roberto, Mathieu et Roland m'ont transmis.

Je me souviendrai à jamais de la profonde humanité de Roberto, sans laquelle je n'aurais pas été capable de faire face aux périodes les plus difficiles de mon doctorat. D'ailleurs l'expérience de Roberto m'a permis d'acquérir un regard scientifiquement solide sur les problèmes traités dans cette thèse.

Je remercie particulièrement Mathieu pour m'avoir guidé jusqu'au dernier jour dans le déroulement du travail de recherche: à plusieurs reprises ses remarques ont été éclairantes et ont ouvert de nouvelles routes. De plus, les compétences et la patience de Mathieu ont été décisives pour surmonter les obstacles techniques rencontrés pendant l'obtention des résultats contenus dans ce manuscrit. Sans doute, si aujourd'hui je ne crains pas les difficultés qu'une démonstration mathématique peut présenter je le dois à Mathieu.

Je suis très reconnaissant à Roland pour m'avoir appris l'importance de l'élégance dans les mathématiques: ses intuitions m'ont souvent aidé à simplifier et formaliser mes idées initialement compliquées ou vagues.

Je souhaite remercier le Prof. Frédéric Meunieur et le Prof. Manuel Iori pour avoir accepté de rapporter ce manuscrit. Je remercie également le Prof. Luís Eduardo N. Gouveia et le Prof. Frédéric Roupin pour avoir accepté d'examiner mes travaux. Je souhaite exprimer ma gratitude au Prof. Ali Ridha Mahjoub pour avoir accepté d'examiner mes travaux et de présider le jury.

Plusieurs personnes m'ont encouragé et aidé à entrer dans le monde de la recherche. Parmi ces personnes j'aimerais citer les Prof. Michele Conforti, Antoine Deza et Andrea Lodi.

Ma gratitude va ensuite à Emiliano, Paolo Gianessi et Fabio pour les échanges scientifiques et pour leur amitié. Je n'aurai pas pu survivre aux démarches administratives et aux heures d'enseignement de l'Université de Paris 13 sans le soutien de ces collègues: Antoine, Joseph, Julien, Lucas, Sébastien, Sophie et Sylvie.

Je tiens à remercier chaleureusement tous les doctorants du LIPN pour leur amitié et leur aide, et en particulier: Domenico, Leila, Moufida, Hanane, Aïcha, Ievgen, Pegah

et Luc. Un grand merci à Marco pour tous les moments passés ensemble à discuter de l'informatique et de la société et pour tous ses conseils.

Il s'est passé beaucoup de temps depuis mes premiers pas comme étudiant en mathématiques à l'Université de Padoue. Dans cette petite ville de l'Italie mon chemin a croisé ceux de personnes merveilleuses: Giovanni, Gigi, Buoso (déjà Davide), Gianluigi, Anna et Valeria. Je les remercie pour être restés à mes côtés pendant toutes ces années et parce que, avec eux, la vie redevient immédiatement sans souci.

Mes jours à Paris auraient été sûrement plus durs sans d'autres amis exceptionnels. Tout d'abord Vito: sa gentillesse et les moments heureux passés ensemble seront parmi mes plus beaux souvenirs de Paris. Paolo et Beatrice avec qui j'ai passé des très belles soirées gourmandes. Irene et Rapha qui ont rempli mes journées de *giuoia*, un mot que l'on pourrait traduire par "joie" mais que l'on peut comprendre seulement en regardant le sourire de leur enfant, Diego. Marcos, qui un jour réussira à m'apprendre le portugais. Et également, Fiorella, Camilla, Giovanni R. (c'est-à-dire Rosso ou Risotto), Paolo, Gianma, Annalaura, Ariane, Arthur, Daniele, Dario, Didier, Francesca, Joel, Margherita, Peppe, Riccardo, Sanela et Valentina.

Infine il ringraziamento più sentito va ai miei due fratelli, Enrico e Lorenzo. Li ringrazio per essere rimasti generosi e comprensivi come sempre, nonostante tante difficoltà si siano messe di traverso. Grazie alle loro generosità e comprensione immutate ho potuto stare lontano da casa, in un anno così complicato, a finire questa tesi. Tesi che, per questo, dedico ai miei due fratelli.

Contents

Introduction

We spend most of our time making decisions. Perhaps we all are familiar with the difficulties that arise when looking for the best decision. Making the best decision often amounts to solve an optimization problem. This can be a hard task for several reasons: in some cases good solutions to a problem are counter-intuitive, in other cases finding them without appropriate methods is too time-consuming. The necessity of designing general paradigms to help the decision process has attracted the interest of many researchers, leading to the development of the Operations Research. This latter is a flourishing discipline at the intersection of mathematics and computer science, which provides several methods to solve optimization problems. Due to this dual nature, Operations Research has given rise to both applied and theoretical investigations. In this thesis we consider both aspects.

The first of our goals is to apply some well-established methods of the Operations Research to a problem arising in logistics. Logistics deals with the organization of transports at several stages. The recent technological advances and social changes have made logistics central in global economy. For instance, the availability of widely differing products at a low cost (made possible by a more and more automated mass production and by the relocation of the production) has generated an increase of good exchanges and commercial traffic in last years. To avoid loss of competitiveness and to limit the harmful effects of the traffic increase (*e.g.,* pollution, congestion,...), a topical research subject is the optimization of the routes of the transport vehicles.

In this thesis we consider the double traveling salesman problem with multiple stacks, in which the construction of optimal routes is severely constrained by the rules governing the disposition of the merchandise in a vehicle. In this problem, first some goods have to be picked up in one region. Subsequently they are delivered to demanding customers in another region, very far from the first one. For this transport, the items are positioned in stacks obeying the last-in-first-out rule for unloading.

The double traveling salesman problem with multiple stacks is a combinatorial optimization problem. Many successful methods to tackle this kind of problems rely on the polyhedral approach introduced by Edmonds [54].

The idea behind the polyhedral approach is to associate the solutions to a combinatorial optimization problem with a set of points in a suitable space. Such a set of points can be associated to a geometrical object called polytope. A complete description of this polytope by means of linear inequalities can be used to solve the starting problem in practice. Unfortunately, complete descriptions of this type are unlikely to be found for hard problems.

This is why we often have to combine partial descriptions with integrality constraints in order to formulate the problem properly. Integer Programming is a research area which develops tools to solve such kind of formulations, called integer linear programs. A well-known approach in Integer Programming is to seek an optimal solution by performing a dichotomic search and by exploiting the partial description in terms of linear inequalities to avoid a complete enumeration of all possible solutions. A method of this type is called branch-and-cut algorithm. The polyhedral approach and the branch-and-cut paradigm have proven to be very efficient in tackling hard combinatorial optimization problems.

In this thesis we study a polytope associated with the double traveling salesman problem with multiple stacks. We show that it is linked to a polytope associated with the traveling salesman problem and to a polytope associated with a specific set covering problem. These are combinatorial optimization problems that have been thoroughly studied from the Integer Programming point of view. By exploiting the links with these problems we derive linear inequalities subsequently embedded in an effective branch-and-cut algorithm for the double traveling salesman problem with multiple stacks.

In the second part of this manuscript we focus on lexicographical polytopes. A lexicographical polytope is the convex hull of the integer points in an n-dimensional box that are lexicographically between two given integer points in the box. Our goal is to provide a full description of the lexicographical polytopes. Our interest is mainly theoretical as we generalize known results on binary lexicographical polytopes. Nevertheless, the latter have proven to be useful in a wide range of practical situations, such as in the design of cryptosystems, in the design of wireless networks and in the resolution of integer linear programs by binary substitution of general integer variables.

The thesis is organized as follows. The first chapter presents the definitions used in subsequent chapters. The remainder of the thesis is divided into two parts. Part I is devoted to the double traveling salesman problem with multiple stacks. Chapter 2 in this part describes the problem and motivates its applications in real-world situations. The literature appeared on similar pickup and delivery problems is surveyed. Finally, also the theoretical and computational results obtained on the double traveling salesman problem with multiple stacks since its appearance are reported. In Chapter 3 a new integer linear programming formulation for the double traveling salesman problem with multiple stacks is presented. Two variations of the problem are considered: the general case, in which the stacks of the vehicle have a finite capacity and the special case in which the capacity is infinite. In this latter case the formulation can be simplified. Chapter 4 is devoted to the study of the polytope obtained as convex hull of the solutions to the formulation in the infinite capacity case. An important result presented in this chapter is that two facet-defining inequalities for this polytope can be retrieved from a facet-defining inequality of a polytope associated with the traveling salesman problem. Subsequently, polyhedral links with a specific set covering polytope are derived. The set covering polytope is shown to be a vertex cover polytope in the two stack case. This observation enables the derivation of new inequalities for the polytope of the double traveling salesman problem with two stacks. In Chapter 5 the polyhedral results previously found are embedded in a branch-and-cut

algorithm for the two stack case. In the case of the double traveling salesman problem with two stacks the experimental results show that this algorithm outperforms the existing ones.

Part II of the thesis is devoted to the study of lexicographical polytopes. It consists of one chapter. The literature on lexicographical polytopes is first described. Then an extended formulation for the componentwise maximal integer points of lexicographical polytopes is presented. The extended formulation is projected and a complete description of the lexicographical polytopes is deduced. Finally, it is proven that the family of lexicographical polytopes is closed under intersection.

Some of the results reported in this thesis have been published or accepted for publication in international scientific journals. The polyhedral links with the traveling salesman problem polytope and their use in a branch-and-cut algorithm are the core of the article *Polyhedral results and a branch-and-cut algorithm for the double traveling Salesman problem with multiple stacks* (M. Barbato, R. Grappe, M. Lacroix, R. Wolfler Calvo, Discrete Optimization 21 (2016): 25–41). The polyhedral links with the set covering and the vertex cover polytopes are reported in the article written in collaboration with R. Grappe, M. Lacroix and R. Wolfler Calvo and titled *A Set Covering Approach for the Double Traveling Salesman Problem with Multiple Stacks*. This article will appear in the Lecture Notes in Computer Science proceedings of the 4th International Symposium on Combinatorial Optimization (Vietri Sul Mare, May 2016).

Finally, the results on lexicographical polytopes are collected in the article *Lexicographical Polytopes* written in collaboration with R. Grappe, M. Lacroix, C. Pira and submitted for publication.

Chapter 1

General Definitions

This chapter provides the notations and basic definitions used throughout the thesis.

1.1 Sets

In this section we fix the notations on numbers and sets that could be nonstandard. For more details on sets we refer the reader to Halmos' book on naive set theory [96].

The symbols $\mathbb{Z}$, $\mathbb{R}$ and $\mathbb{Q}$ denote respectively the sets of integer, real and rational numbers. A real number a is *nonnegative* if $a \geq 0$. We define $\mathbb{R}_+ = \{a \in \mathbb{R} \colon a \geq 0\}$, that is $\mathbb{R}_+$ is the restriction of $\mathbb{R}$ to its nonnegative elements. Similary, $\mathbb{Z}_+$ and $\mathbb{Q}_+$ are respectively the restrictions of $\mathbb{Z}$ and $\mathbb{Q}$ to their nonnegative elements. Given $a \in \mathbb{R}$, we define $\lfloor a \rfloor$ as the lower integer part of a and $\lceil a \rceil$ as the upper integer part of a.

The *cardinality* of a set S is the number of its elements and it is denoted $|S|$. A set S is *finite* if $|S| < +\infty$, otherwise it is *infinite*. The set of all subsets of S is its *power set*, denoted 2^S.

Let S_1 and S_2 be two sets. The *relative complement* of S_1 in S_2 is:

$$S_2 \setminus S_1 = \{s \in S_2 \colon s \notin S_1\}.$$

When $S_1 \subseteq S_2$, the relative complement of S_1 in S_2 is simply called *complement of* S_1 *in* S_2 and it is indicated with $\bar{S}_1$. The *symmetric difference* of two sets S_1 and S_2 is $S_1 \triangle S_2 = (S_1 \cup S_2) \setminus (S_1 \cap S_2)$.

A *k-uple* is an ordered list of k elements. In this writing, k-uples will be written by listing their elements within two parenthesis, *e.g.,* $(2, 1, 7, 4)$ is a 4-uple of integer numbers. A *pair* is a 2-uple. Given $S_1, \ldots, S_m$ sets, their *product set* is

$$S_1 \times \cdots \times S_m = \{(s_1, \ldots, s_m) \colon s_i \in S_i, i = 1, \ldots, m\}.$$

Ordering Relations on Sets

Given a finite set S, a *binary relation R on S* is a subset of $S \times S$. For the purposes of this section, given $s_1, s_2 \in S$ it is convenient to write $s_1 R s_2$ instead of $(s_1, s_2) \in R$. Here, we are interested in binary relations called partial orderings and linear orderings. A *partial ordering* of S is a binary relation $\preccurlyeq$ on S such that:

- $s \preccurlyeq s$ for all $s \in S$ (*i.e.,* $\preccurlyeq$ is *reflexive*),
- if $s_1 \preccurlyeq s_2$ and $s_2 \preccurlyeq s_1$ then $s_1 = s_2$ for all $s_1, s_2 \in S$ (*i.e.,* $\preccurlyeq$ is *antisymmetric*),
- if $s \preccurlyeq t$ and $t \preccurlyeq u$ then $s \preccurlyeq u$ for all $s, t, u \in S$ (*i.e.,* $\preccurlyeq$ is *transitive*).

A partial ordering $\preccurlyeq$ of S is *total* if in addition $s_1 \preccurlyeq s_2$ or $s_2 \preccurlyeq s_1$ for all $s_1, s_2 \in S$. Note that the totality of a partial ordering implies its reflexivity.

To every total partial ordering $\preccurlyeq$ of S it is associated a *linear ordering* $\prec$ of S defined by $a \prec b$ if and only if $a \preccurlyeq b$ and $a \neq b$. Note that a linear ordering is not reflexive.

If S is a finite set and $\prec$ is a linear ordering on S, the pair $(S, \prec)$ is a *finite totally ordered set*. It can be shown that a finite totally ordered set $(S, \prec)$ admits a least element ℓ, that is $\ell \prec s$ for all $s \in S$. From this it follows that $(S, \prec)$ can be represented by a *sequence* $s_1, s_2, \ldots, s_{|S|}$ meaning that $s_i \prec s_j$ and $s_i \neq s_j$ for all $1 \leq i < j \leq |S|$.

1.2 Linear Algebra

In this section we define matrices and vectors as well as the operations on these structures that will be used in this thesis. We focus in particular on real matrices and vectors.

1.2.1 Matrices

A *matrix* is a disposition of elements in a rectangular array, with a finite number of *rows* (indexing the vertical dimension) and *columns* (indexing the horizontal dimension). The elements of a matrix are also called *entries* in this writing. The entry placed in row i and column j of a matrix A is denoted A_{ij}. The *transpose* a matrix A is the matrix $A^\top$ defined element-wise by $A^\top_{ij} = A_{ji}$.

We define $E^{m \times n}$ to be the set of matrices with m rows, n columns and all entries in the ground set E. For instance, the set of *binary matrices* is $\{0, 1\}^{m \times n}$, *i.e.,* this is the set of matrices containing only 0's and 1's. Other well-known examples are real matrices $\mathbb{R}^{m \times n}$, integer matrices $\mathbb{Z}^{m \times n}$ and so on. We will identify $\mathbb{R}^{1 \times 1}$ with $\mathbb{R}$. The elements of $\mathbb{R}$ are called *scalars*.

Binary Operations on Matrices

Here are some common operations on real matrices.

- The *sum* of two matrices $A, B \in \mathbb{R}^{m\times n}$ is $C = A + B$ defined by $C_{ij} = A_{ij} + B_{ij}$, for all $1 \le i \le m$ and $1 \le j \le n$. Hence $C \in \mathbb{R}^{m\times n}$.
- The *multiplication* of $A \in \mathbb{R}^{m\times n}$ with a scalar $\alpha \in \mathbb{R}$ is the matrix $C \in \mathbb{R}^{m\times n}$ defined element-wise by $C_{ij} = \alpha A_{ij}$, for all $1 \le i \le m$ and $1 \le j \le n$. We indicate this product by αA.
- The *product* of two matrices $A \in \mathbb{R}^{m\times k}$ and $B \in \mathbb{R}^{k\times n}$ is the matrix $AB \in \mathbb{R}^{m\times n}$ defined element-wise by $(AB)_{ij} = \sum_{\ell=1}^{k} A_{i\ell}B_{\ell j}$, for all $1 \le i \le m$ and $1 \le j \le n$.
- Given $A, B \in \mathbb{R}^{m\times n}$, we write $A \le B$ whenever $A_{ij} \le B_{ij}$ for all $1 \le i \le m$ and $1 \le j \le n$. Then $A = B$ if and only if $A \le B$ and $B \le A$.

1.2.2 Vectors and Vector Spaces

Given $n \in \mathbb{Z}_+$, a *real vector of dimension* n is $v \in \mathbb{R}^{n\times 1}$. For short we will write $v \in \mathbb{R}^n$ instead of $v \in \mathbb{R}^{n\times 1}$ and v_i instead of v_{i1} for every $i = 1, \ldots, n$. The *scalar product* of two n-dimensional real vectors c and v is $c^\top v = \sum_{i=1}^{n} c_i v_i$. Unless differently stated, we will always omit the transposition symbol and write instead cv for the scalar product of c and v. Two nonzero vectors $u, v \in \mathbb{R}^n$ are called *orthogonal* if $uv = 0$. The symbols $\mathbf{0}$ and $\mathbf{1}$ will denote the vectors having all entries equal to 0 or to 1, respectively. The dimension of these vectors will be clear from the context.

The *real vector (or linear) spaces of dimension* k is the set $\mathbb{R}^k$ (for some $k \in \mathbb{Z}_+$) provided with the operations of sum of vectors and product of vectors with scalars defined above. It is easily verified that $\mathbb{R}^k$ with these two operations coincides with a vector space over $\mathbb{R}$, — for a definition of vector space in general see Lang's book [121].

Let us consider a finite set E. Since $|E| \in \mathbb{Z}_+$, the vector space $\mathbb{R}^{|E|}$ can be defined by following the above notations. However, we will think in this case that the entries of $v \in \mathbb{R}^{|E|}$ are indexed by the elements of E (assuming a fixed order on them). In this notation v_e is the coordinate corresponding to the element e of E. We define $v(F) = \sum_{f\in F} v_f$ for all $F \subseteq E$. Similarly, $\mathbb{R}^{|E|\times|F|}$, with E and F finite sets, is the set of real matrices with row indices in E and column indices in F. In this case, for $A \in \mathbb{R}^{|E|\times|F|}$, the entry A_{ef} is the one corresponding to $e \in E$ and $f \in F$.

Let us consider the vector space $\mathbb{R}^n$ for some $n \in \mathbb{Z}_+$. A *vector subspace*, or just *subspace* when no confusion may arise, is $\mathcal{V} \subseteq \mathbb{R}^n$ such that:

VS1. $u, v \in \mathcal{V}$ implies $u + v \in \mathcal{V}$,

VS2. $\alpha \in \mathbb{R}$ and $v \in \mathcal{V}$ imply $\alpha v \in \mathcal{V}$.

1.2.3 Linear Functions

Given two vector spaces $\mathbb{R}^m$ and $\mathbb{R}^n$, a *linear function* is a function $\phi\colon \mathbb{R}^m \to \mathbb{R}^n$ such that $\phi(u+v) = \phi(u) + \phi(v)$ and $\phi(\alpha v) = \alpha\phi(v)$, for all $u, v \in \mathbb{R}^m$ and $\alpha \in \mathbb{R}$.

The *kernel* of a linear function $\phi\colon \mathbb{R}^m \to \mathbb{R}^n$ is $\ker(\phi) = \{v \in \mathbb{R}^m\colon \phi(v) = 0\}$. The *image* of ϕ is $\mathrm{im}(\phi) = \{w \in \mathbb{R}^n\colon w = \phi(v), \text{ for some } v \in \mathbb{R}^m\}$.

Example 1.2.1 (Orthogonal Projection)**.** Let us consider two integers n and p. The *orthogonal projection* of $S \subset \mathbb{R}^{n+p}$ onto the subspace $\mathbb{R}^n$ is:

$$proj_x(S) = \{x \in \mathbb{R}^n\colon \exists z \in \mathbb{R}^p \text{ such that } (x, z) \in S\}.$$

1.2.4 Linear Independence and Bases

In this section $\mathbb{R}^n$ denotes the real vector space of dimension n, for some $n \in \mathbb{Z}_+$. That is, we will implicitly assume that the sum between vectors and the multiplication with scalars are available.

The vectors $v^1, v^2, \ldots, v^k \in \mathbb{R}^n$ are *linearly independent* if $\lambda_1 v^1 + \lambda_2 v^2 + \cdots + \lambda_k v^k = \mathbf{0}$ (with $\lambda_1, \lambda_2, \ldots, \lambda_k$ being scalars) implies $\lambda_1 = \lambda_2 = \cdots = \lambda_k = 0$. If $v^1, v^2, \ldots, v^k$ are not linearly independent, they are said *linearly dependent*.

A vector $v \in \mathbb{R}^n$ is a *linear combination* of the vectors $v^1, v^2, \ldots, v^k \in \mathbb{R}^n$ if there exist scalars $\lambda_1, \ldots, \lambda_k$ such that $v = \lambda_1 v^1 + \lambda_2 v^2 + \cdots + \lambda_k v^k$.

We will often use implicitly the following important and well-known result:

Proposition 1.2.2. *The vectors $v^1, v^2, \ldots, v^k \in \mathbb{R}^n$ are linearly independent if and only if none of them is a linear combination of the others.*

Let $\mathcal{V}$ be a subspace of $\mathbb{R}^n$. We say that $v^1, \ldots, v^k$ *generate* $\mathcal{V}$ if every element of $\mathcal{V}$ is a linear combination of $v^1, \ldots, v^k$. Conversely, the set of the linear combinations of $v^1, \ldots, v^k \in \mathbb{R}^n$ is a subspace of $\mathbb{R}^n$, the *subspace generated by* $v^1, \ldots, v^k$.

A *basis* $\mathcal{B}$ of a vector subspace $\mathcal{V}$ of $\mathbb{R}^n$ is a minimal set of vectors generating $\mathcal{V}$, *i.e.,* there is no $\mathcal{B}' \subset \mathcal{B}$ whose elements generate $\mathcal{V}$.

Theorem 1.2.3. *A set $\mathcal{B}$ of vectors of $\mathbb{R}^n$ is a basis of $\mathcal{V}$ if and only if*

- *$\mathcal{B} \subseteq \mathcal{V}$;*
- *the elements of $\mathcal{B}$ are linearly independent;*
- *the elements of $\mathcal{B}$ generate $\mathcal{V}$.*

In addition, all bases of a vector subspace of $\mathbb{R}^n$ have the same cardinality.

Let $\mathcal{B}$ be a basis for $\mathcal{V}$ a subspace of $\mathbb{R}^n$. The cardinality of $\mathcal{B}$ is called *dimension* of $\mathcal{V}$, denoted $\dim(\mathcal{V})$.

From the theorem above it follows that if $\mathcal{B} = \{b^1, \ldots, b^k\}$ is a basis for $\mathcal{V}$ subspace of $\mathbb{R}^n$ then each element v of $\mathcal{V}$ is $v = \lambda_1 b^1 + \cdots + \lambda_k b^k$ with λ_i uniquely determined for every $i = 1, \ldots, k$. Assuming that the elements of $\mathcal{B}$ are ordered following the subscripts from 1 to k, we write $v_{\mathcal{B}} = (\lambda_1, \ldots, \lambda_k)^\top$.

We point out that $\mathbb{R}^n$ has dimension n because, in view of previous theorem, the set of its vectors $e^1 = (1, \mathbf{0})^\top$, $e^2 = (0, 1, \mathbf{0})^\top$,...,$e^n = (\mathbf{0}, 1)^\top$ is a basis of $\mathbb{R}^n$ with n elements. The basis of $\mathbb{R}^n$ given by $\mathcal{E} = \{e^1, e^2, \ldots, e^n\}$ is called *standard basis* and e^i is called, for every $i = 1, \ldots, n$, *i-th canonical vector*. Note that $v_{\mathcal{E}} = v$ for every $v \in \mathbb{R}^n$.

Let us consider a linear function $\phi\colon \mathbb{R}^m \to \mathbb{R}^n$. We get that ϕ corresponds exactly to one matrix F defined by $F_{ij} = (\phi(e^j))_i$, for $1 \leq i \leq m$ and $1 \leq j \leq n$. We have $\phi(v) = Fv$. Conversely, for every matrix $F \in \mathbb{R}^{m \times n}$, the function $\phi\colon \mathbb{R}^m \to \mathbb{R}^n$ defined by Fv for all $v \in \mathbb{R}^m$ is linear.

The *row rank* of a matrix is the greatest number of linearly independent rows it has. The *column rank* of a matrix is the greatest number of linearly independent columns it has. A matrix is said to be of *full row rank* (resp. *full column rank*) if all its rows (resp. columns) are linearly independent.

1.2.5 Affine Spaces

In this section we define the affine independence, affine combinations and affine spaces. In this context $v \in \mathbb{R}^n$ is called *point*.

The points $v^1, v^2, \ldots, v^k \in \mathbb{R}^n$ are *affinely independent* if and only if the system

$$\sum_{i=1}^{k} \lambda_i v^i = 0, \sum_{i=1}^{k} \lambda_i = 0$$

implies $\lambda_1 = \lambda_2 = \cdots = \lambda_k = 0$.

A point $v \in \mathbb{R}^n$ is an *affine combination* of $v^1, v^2, \ldots, v^k \in \mathbb{R}^n$ if and only if $v^k = \sum_{i=1}^{k} \lambda_i v^i$ for some $\lambda_1, \lambda_2, \ldots, \lambda_k$ such that $\sum_{i=1}^{k} \lambda_i = 1$.

It is well-known that the points $v^1, v^2, \ldots, v^k$ are affinely independent if and only if none of them can be expressed as an affine combination of the others.

An *affine space* of $\mathbb{R}^n$ is $A \subseteq \mathbb{R}^n$ closed under affine combinations. Given A an affine space of $\mathbb{R}^n$, a set $B \subseteq A$ is an *affine subspace of* A if B is closed under affine combinations.

An *affine basis* of an affine space A is a set of affinely independent points $v^1, \ldots, v^{k+1} \in A$ such that every point in A can be expressed as an affine combination of $v^1, \ldots, v^{k+1}$. In this case we define the dimension of A as $\dim(A) = k$. The previous definition relies on the fact that all bases of an affine space of $\mathbb{R}^n$ have the same cardinality.

The *affine hull* of a set $S \subseteq \mathbb{R}^n$ is the smallest affine space of $\mathbb{R}^n$ containing S. The affine hull of S is denoted $\mathrm{aff}(S)$. We define the dimension of S as $\dim(S) = \dim(\mathrm{aff}(S))$.

1.2.6 Conic and Convex Combinations

A *conic combination* of the vectors $v^1, \ldots, v^k \in \mathbb{R}^n$ is $v = \lambda_1 v^1 + \cdots + \lambda_k v^k$ with $\lambda_i \geq 0$, for all $i = 1, \ldots, k$. It is called *convex combination* if, in addition, $\lambda_1 + \lambda_2 + \cdots + \lambda_k = 1$.

A set $S \subseteq \mathbb{R}^n$ is *convex* if it is closed under convex combinations of its elements. The *convex hull* of $S \subseteq \mathbb{R}^n$, denoted $\text{conv}(S)$, is the smallest convex set containing S. It can be shown that $\text{conv}(S)$ is the set of the convex combinations of finitely many points of S.

A set $C \subseteq \mathbb{R}^n$ is a *convex cone* if it closed under conic combinations. Given $D \subseteq \mathbb{R}^n$, the convex cone *generated by* D is the set $\text{cone}(D)$ of all conic combinations of vectors of D. Let $C \neq \{\mathbf{0}\}$ be a convex cone. A *ray* of C is any set of the type $cone(r)$ with $r \in C$. We will identify r with $cone(r)$. We say that two rays r^1 and r^2 are *distinct* if $r^2 \notin cone(r^1)$. An *extreme ray* of C is a ray r of C such that no distinct rays $r^1, r^2 \in C$ exist with $r = \lambda^1 r^1 + \lambda^2 r^2$ for some $\lambda^1, \lambda^2 > 0$.

1.3 Graphs

In this section we give the notation for undirected and directed graphs.

1.3.1 Undirected Graphs.

A *simple undirected graph* or just *graph* is a pair of sets $G = (V, E)$ with V being finite and $E \subseteq \{\{u, v\} \colon u \neq v \in V\}$. When $E = \{\{u, v\} \colon u \neq v \in V\}$ the graph G is said *complete*.

The set V is called *vertex set* and an element of V is a *vertex* of the graph G; the set E is called *edge set* and an element of E is an *edge* of G. If $e = \{u, v\} \in E$ is an edge of a graph G, the vertices u and v are called *endpoints* of e. In this case, u and v are said *adjacent* and we also say that e is *incident to* u and v. The set of edges incident to a vertex v is denoted $\delta(v)$. The *neighborhood* of $W \subseteq V$ is the set of vertices adjacent to at least one vertex in W. A vertex v of a graph is *isolated* if the neighborhood of $\{v\}$ is empty.

For each vertex $v \in V$, its *degree* is $\deg(v)$, the number of edges incident to v. For every graph $G = (V, E)$ the degree of vertices and the number of edges are linked by the relation: $\sum_{v \in V} \deg(v) = 2|E|$.

A graph $G = (V, E)$ is *k-regular* if $\deg(v) = k$ for all $v \in V$. The *maximum degree of* $G = (V, E)$ is $\Delta(G) = \max_{v \in V}\{\deg(v)\}$.

Graphs can be drawn as follows. For each vertex we draw a point; for each edge $\{u, v\}$ we draw the line linking the points corresponding to u and v.

A graph $G = (V, E)$ admits other representations. The first one is the *adjacency matrix*. It is the binary matrix $A \in \{0, 1\}^{|V| \times |V|}$ such that $A_{uv} = 1$ if and only if u and v are adjacent in G, for every $u, v \in V$. A second representation is the *incidence matrix*. It is the binary matrix $B \in \{0, 1\}^{|E| \times |V|}$ such that $B_{ev} = 1$ whenever v is an endpoint of e. Note that each row of B has exactly two nonzero entries.

Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic*, denoted $G_1 \cong G_2$ when there is a bijection $f\colon V_1 \to V_2$ such that $\{x, y\} \in E_1$ if and only if $\{f(x), f(y)\} \in E_2$.

The *complement* of a graph $G = (V, E)$ is the graph $\bar{G} = (V, \bar{E})$ in which $\{i, j\} \in \bar{E}$ if and only if $\{i, j\} \notin E$.

Weighted Graphs. A *vertex-weighted graph* is (G, w), where $G = (V, E)$ is a graph and $w \in \mathbb{R}^{|V|}$. For all $v \in V$, we call w_v the *weight of* v. An *edge-weighted graph* is defined in analogous manner, with $w \in \mathbb{R}^{|E|}$. In this case, for every $e \in E$, we call w_e the *weight of* e. For every $E' \subseteq E$ the *weight of* E' is $w(E')$.

Subgraphs. A *subgraph of* a graph $G = (V, E)$ is a graph $H = (U, F)$ with $U \subseteq V$ and $F \subseteq E$. Unless differently stated, given a subgraph S of G, we denote by $V(S)$ the set of vertices of S and by $E(S)$ the set of its edges. Let $G = (V, E)$ be a graph and $U \subseteq V$. The subgraph of G *vertex-induced by* U is $G[U] = (U, F)$ where $F = E \cap \{\{i, j\} \colon i \neq j \in U\}$. If W is a subset of V, we write $G \setminus W$ for $G[V \setminus W]$. Similarly, the subgraph of G *edge-induced by* $E' \subseteq E$ is $G[E'] = (V(E'), E')$, where $V(E')$ is the set of vertices that are endpoints of some edges in E'. We simply say *induced* if it is clear from the context whether a subgraph is vertex induced or edge induced.

Walks, Paths and Circuits. A *walk* of a graph $G = (V, E)$ is a set of vertices and edges $\{v_0, e_1, v_1, \ldots, v_{k-1}, e_k, v_k\}$, where $v_j \in V$ for all $0 \leq j \leq k$ and $e_j = \{v_{j-1}, v_j\}$ for all $1 \leq j \leq k$. We say that the walk *visits* its vertices. The vertices v_0 and v_k are called *extremities* of the walk. The walk is called *closed* if $v_0 = v_k$, *non-closed* otherwise.

A non-closed walk is called *path* when all its edges and vertices are distinct. Given a path P, a *subpath of* P is a path whose vertex set and edge set are contained in P. A v_0, v_k*-path* is a path having v_0 and v_k as extremities.

A closed walk is called *circuit* if all its vertices (except its extremities) are pairwise distinct. A circuit of a graph G is *Hamiltonian* if it contains all vertices of G.

Unless differently stated, the parity of walks, paths and circuits is understood according to the number of their edges. Note that walks, paths and circuits naturally give rise to subgraphs, by considering their vertex and edge sets. Hence, when needed we will identify them with the corresponding subgraphs.

In addition, we will also identify the set of edges of a walk with the walk itself. The *length* of a walk is the number of its edges. Similar considerations hold for paths and circuits.

Let U be the vertex set of a walk and F be its edge set. In a vertex-weighted graph (G, w), the *vertex-weight* of the walk is $w(U)$. In an edge-weighted graph (G, w), the *edge-weight* of the walk is $w(F)$. When the context is clear, we simply write *weight*.

Let us take a walk W in a graph G and consider $\{u, v\}$ and $\{v, t\}$ edges of this walk. If we replace the two edges $\{u, v\}$ and $\{v, t\}$ with the edge $\{u, t\}$ we get a new walk W'.

We write in this case $W' = W/v$. The same definition is given on paths. In particular if v is an extremity of a path P, the new path P/v is the one obtained by removing v and the edge of P incident to v.

Subdivisions. Let G be a graph and $\{u, v\}$ one of its edges. We *subdivide* $\{u, v\}$ if we replace it with a u, v-path. The graph resulting from G after subdividing some of its edges is called a *subdivision of* G.

Connectivity and Cuts. We say that a graph G is *connected* when, for every two vertices u, v of G, there exists a u, v-path in G. Given $W \subseteq V$, the *W-cut* is the set $\delta(W)$ of edges having one endpoint in W and the other in $V \setminus W$.

Given two vertices $s, t \in V$, an *s, t-cut* is a W-cut with $s \in W$ and $t \in V \setminus W$.

A *minimum weight cut* (resp. minimum weight s, t-cut) is a W-cut (resp. s, t-cut) of minimum weight.

Cliques. A *clique* of a graph G is a subset of vertices of G inducing a complete subgraph of G. A clique of G is *maximal* if it is not properly contained in a clique of G. It is a *maximum* clique if it is a clique of G of greatest cardinality. The *clique number of* G, denoted $\omega(G)$, is the cardinality of a maximum clique of G.

Stable Sets. A *stable set* of a graph G is a subset of vertices of G pairwise nonadjacent. A stable set of G is *maximal* if it is not properly contained in a stable set of G. It is a *maximum* stable set if it is a stable set of maximum cardinality. The *stability number of* G, denoted $\alpha(G)$, is the cardinality of a maximum stable set of G.

Colorings. A *proper vertex coloring* of a graph G, or simply *coloring* in this writing, consists in assigning labels to the vertices of G so that any two adjacent vertices have different labels. The labels are called *colors*. Note that every graph $G = (V, E)$ admits a coloring using $|V|$ colors. From now on, the colors in a coloring of $G = (V, E)$ are identified with the natural numbers $1, 2, 3, \ldots$.

Let us consider a graph $G = (V, E)$ and a coloring for this graph. For every $i = 1, \ldots, n$, we define the *color class* of i as the set of vertices of G that are assigned color i in this coloring.

A coloring of a graph is a k-coloring if it has at most k nonempty color classes. Given a graph G, the smallest $k \in \mathbb{Z}_+$ for which G admits a k-coloring is called *chromatic number of* G. The chromatic number of a graph G is denoted $\chi(G)$.

It follows from the definitions that $\chi(G) \geq \omega(G)$ for every graph G.

Bipartite Graphs. A graph G is *bipartite* if $\chi(G) \leq 2$. It follows that the vertex set of a bipartite graph G can be divided into two stable sets U and V of G. Hence, we will indicate a bipartite graph by $G = (U, V; E)$ where U and V are the two vertex sets and E the edge set of the graph.

For a bipartite graph $G = (U, V; E)$, we say that $C \subseteq V$ *covers* $T \subseteq U$ if the neighborhood of C coincides with T. A bipartite graph with at least two vertices is called *complete* if $\chi(G) = 2$ and every vertex in a color class is adjacent to all vertices in the other color class.

Perfect Graphs

A *perfect graph* is a graph $G = (V, E)$ such that $\chi(G) = \omega(G)$ and $\chi(G[W]) = \omega(G[W])$ for all $W \subseteq V$.

Theorem 1.3.1 (Lovász, [130]). *A graph $G = (V, E)$ is perfect if and only if its complement $\bar{G}$ is perfect. This is also equivalent to $\omega(G[W])\alpha(G[W]) \geq |W|$ for all $W \subseteq V$.*

We cite few examples of known perfect graphs.

Comparability Graphs. Given a graph $G = (V, E)$ and an edge $\{u, v\} \in E$, an *orientation of* $\{u, v\}$ is one of the two pairs (u, v) or (v, u). That is, we enforce an order on the endpoints of the edge.

An *orientation* F of $G = (V, E)$ is a set containing exactly one orientation of e, for each $e \in E$. An orientation F of $G = (V, E)$ is *transitive* if $(u, v) \in F$ and $(v, t) \in F$ implies $(u, t) \in F$ for all $u, v, t \in V$.

A graph G is a *comparability graph* if it admits a transitive orientation.

Theorem 1.3.2 (See p.133 Golumbic [82]). *Comparability graphs are perfect.*

Permutation Graphs. Let us be given a permutation π of $\{1, 2 \ldots, n\}$. We associate π with a graph $G(\pi) = (V_\pi, E_\pi)$ defined by

- $V_\pi = \{1, 2, \ldots, n\}$,
- $E_\pi = \{\{i, j\} : i > j \text{ and } \pi(i) < \pi(j) \text{ for all } i, j \in V_\pi\}$.

We say that a graph $G = (V, E)$ is a *permutation graph* when there exists a permutation π of $\{1, \ldots, |V|\}$ such that G is isomorphic to $G(\pi)$.

Every permutation graph is a comparability graph as it can be seen by orienting as (i, j) each edge $\{i, j\} \in E_\pi$ with $i > j$. In fact, using comparability graphs we can completely characterize permutation graphs.

Theorem 1.3.3 ([159]). *A graph G is a permutation graph if and only if G and $\bar{G}$ are comparability graphs.*

It follows that permutation graphs (and their complement) are perfect.

1.3.2 Directed Graphs

A *directed simple graph* or simply *digraph* is a pair of sets $D = (V, A)$ with V being finite and $A \subseteq \{(i, j) \colon i \neq j \in V\}$. The set V is called *vertex set of* D and its elements are called *vertices*. The set A is the *arc set of* D and its elements are called *arcs*. Hence an arc is a pair of vertices. The digraph D is *complete* if its arc set contains all possible pairs of distinct vertices of D. A *subgraph* of a digraph $D = (V, A)$ is any digraph $D' = (V', A')$ with $V' \subseteq V$ an $A' \subseteq A$. Unless differently stated, given a subgraph D' of D, we denote by $V(D')$ the set of vertices of D' and by $A(D')$ the set of its arcs. A *vertex-weighted* digraph is (D, w) with $D = (V, A)$ being a digraph and $w \in \mathbb{R}^{|V|}$. An *arc-weighted* digraph is (D, w) with $D = (V, A)$ being a digraph and $w \in \mathbb{R}^{|A|}$. We simply say *weighted* if the context is clear.

For each arc $a = (u, v)$ of a digraph, u and v are called respectively the *tail* of a and the *head* of a. In addition, we say that a is an arc *entering in* v and *leaving* u. The *out-degree* $\deg^+(v)$ (resp. *in-degree* $\deg^-(v)$) of a vertex v is the number of arcs leaving (resp. entering in) v. Similarly, given $W \subseteq V$ we define its *outcut* $\delta^+(W)$ as the set of arcs having their tail in W and their head in $V \setminus W$. Its *incut* $\delta^-(W)$ is defined analogously as the set of arcs having their head in W and their tail $V \setminus W$. Outcuts and incuts of a graph are referred generically as *cuts* of the graph. The *degree* of a cut is the number of its arcs. We define an (s, t)-*cut* of a digraph $D = (V, A)$ to be an outcut of $W \subseteq V$ with $s \in W$ and $t \notin W$.

Given a weighted digraph (D, w), the *weight* of a cut of D is the sum of the weights associated with the arcs in the cut. A *minimum weight* cut is one whose weight is minimum among all cuts. Analogous definitions are given for (s, t)-cuts.

Digraphs can be drawn as graphs but using arrows instead of lines to represent the arcs. The arrow goes from the tail of the arc to its head. Note that an orientation of a graph yields a digraph.

A *directed walk* of a digraph $D = (V, A)$ is a set of the form $\{v_0, a_1, v_1, \ldots, v_{k-1}, a_k, v_k\}$ with $v_j \in V$ for all $0 \leq j \leq k$ and $a_j = (v_{j-1}, v_j)$ for all $1 \leq j \leq k$. We say that the walk *starts at* v_0 and *ends in* v_k and that v_0 and v_k are respectively the *starting* and *ending points* of the walk. The walk is called *closed* if $v_0 = v_k$, *non-closed* otherwise. A non-closed walk is a *directed path* if all its vertices and arcs are distinct. Given a directed path P, a *subpath of* P is any directed path whose vertex set and arc set is contained in P. A (v_0, v_k)-*path* is a path having v_0 as starting point and v_k as ending point. A closed directed walk is called *directed circuit* if all its vertices (except the starting and ending points) are pairwise distinct. A directed circuit of a digraph D is *Hamiltonian* if it visits all vertices of D. Given a directed path or circuit C the *reverse of* C, denoted $\overleftarrow{C}$ is the path or circuit composed of the opposite arcs of C. We will omit the word "directed" when no confusion may arise. A digraph $D = (V, A)$ is *strongly connected* if it has a (u, v)-path and a (v, u)-path for every $u, v \in V$.

As for walks, paths and circuits of undirected graphs, also in digraphs we will identify them with the corresponding subgraph and with the set of their arcs. Their *length* is then defined as the number of their arcs.

Let us consider a directed walk of a digraph D with vertex set U and arc set B. If (D, w) is a vertex-weighted digraph, the *vertex-weight* of the walk is $w(U)$. If (D, w) is a arc-weighted digraph, the *arc-weight* of the walk is $w(B)$.

Let us take a walk W in a digraph D and consider (u, v) and (v, t) arcs of this walk. If we replace in W the two arcs (u, v) and (v, t) with the arc (u, t) we get a new walk W'. We write $W' = W/v$. We give the same definition on paths. In this case, if v is an the starting or ending point of the directed path P, the new directed path P/v is the one obtained by removing v and the arc incident to v.

Let D be a digraph and (u, v) one of its arcs. We *subdivide* (u, v) if we replace it with a (u, v)-path. The digraph resulting from D after subdividing some of its arcs is called a *subdivision* of D.

1.4 Notions of Computational Complexity Theory

We give an informal presentation of the tools used in computational complexity to analyze the difficulty of problems and the performance of algorithms. A more complete treatment of this topic can be found in [69].

Algorithms. An *algorithm* is a procedure decomposable into a finite sequence of elementary arithmetic operations that transforms an initial *input* into an *output*. The number of elementary operations performed by an algorithm is the *running time* of the algorithm. The *size* of an input is the amount of space needed to write down the input.

Problems. A *problem* is a general question which depends on some parameters (whose values are unknown). A problem also requires to specify the list of the properties of its *solutions*. An *instance* of a problem is obtained by fixing the values of the parameters. The *size* of an instance is the amount of space necessary to write down the instance.

An algorithm *for a problem* $\mathcal{P}$ is an algorithm which takes an instance of $\mathcal{P}$ as initial input and produces a solution to $\mathcal{P}$. We also say that the algorithm *solves* $\mathcal{P}$.

Polynomial Algorithms

Let $\mathcal{A}$ be an algorithm. Given a function $f\colon \mathbb{Z}_+ \to \mathbb{Z}_+$, we say that algorithm $\mathcal{A}$ *runs in* $O(f(n))$ *time in the input size* if there exist a scalar α and a natural n_0 such that, for any input of size $n \geq n_0$, the running time of $\mathcal{A}$ is at most $\alpha f(n)$. If $\mathcal{A}$ is an algorithm for a problem $\mathcal{P}$ that runs in $O(f(n))$ time we say that $\mathcal{P}$ can be *solved in* $O(f(n))$ time. If f is a polynomial function we say that $\mathcal{A}$ runs (resp. $\mathcal{P}$ can be solved) in *polynomial time* in the input (resp. instance) size. For short we also say that $\mathcal{A}$ is a *polynomial time* algorithm.

The classes P and NP

A *decision problem* is a problem having only two possible answers: *yes* or *no*. The *class P* is the set of decision problems that can be solved in polynomial time.

Let $\mathcal{P}$ be a decision problem. The $\mathcal{Y}$-instances of $\mathcal{P}$ is the set of its instances having answer *yes*. A *certificate* for a $\mathcal{Y}$-instance is a proof that *yes* is the correct answer for that instance. We say $\mathcal{P}$ belongs to the *class NP* if all its $\mathcal{Y}$-instances admit a certificate whose correctness can be verified in polynomial time. The class P is contained in the class NP.

NP-Completeness and NP-hardness

We say that the decision problem $\mathcal{P}'$ is *reducible in polynomial time to* a problem $\mathcal{P}$ if there exists a polynomial algorithm that, taking an instance I of $\mathcal{P}'$ as input, produces as output a $\mathcal{Y}$-instance of $\mathcal{P}$ if and only if I is a $\mathcal{Y}$-instance of $\mathcal{P}'$.

A decision problem $\mathcal{P}$ is *NP-complete* if $\mathcal{P} \in$ NP and all decision problems in NP are reducible in polynomial time to $\mathcal{P}$. A problem $\mathcal{P}$ is *NP-hard* if all problems in NP are reducible in polynomial time to $\mathcal{P}$. Hence the NP-complete problems are also NP-hard problems.

1.5 Polyhedral Theory

We recall some basic notions in polyhedral theory. We follow to a large extent the book [40].

Polyhedra and Polytopes. A *polyhedron* of $\mathbb{R}^m$ is $P = \{x \in \mathbb{R}^m \colon Ax \leq b\}$, where $A \in \mathbb{R}^{\ell \times m}$ and $b \in \mathbb{R}^\ell$ for some positive integers ℓ and m. If $b = \mathbf{0}$ then P is called *polyhedral cone*. A bounded polyhedron is called *polytope*. A polyhedron P of $\mathbb{R}^n$ is *full dimensional* if $\dim(P) = n$.

Note that polyhedral cones are convex cones.

Recession Cone and Lineality Space Given a polyhedron $P = \{x \in \mathbb{R}^n \colon Ax \leq b\}$, its *recession cone* is the polyhedral cone $\mathrm{rec}(P) = \{x \in \mathbb{R}^n \colon Ax \leq \mathbf{0}\}$. The *lineality space* of P is the set $\mathrm{lin}(P) = \{x \in \mathbb{R}^n \colon Ax = \mathbf{0}\}$. A polyhedron P is *pointed* if $\mathrm{lin}(P) = \{\mathbf{0}\}$.

Faces. A *valid inequality* for a polyhedron P of $\mathbb{R}^n$ is a linear inequality $ax \leq \delta$ verified by all points of P. A *face* of a polyhedron P is any set of the form $P \cap \{x \in \mathbb{R}^n \colon ax = \delta\}$ with $ax \leq \delta$ a valid inequality of P. In this case, we say that $ax \leq b$ *defines* the face. A face F of a polyhedron P is *proper* if $\emptyset \subset F \subset P$. A *facet* of a polyhedron P is a proper face maximal with respect to the inclusion. A *vertex* or *extreme point* v of a polyhedron P is a point of P that cannot be expressed as a convex combination of two points of P other than v.

Integer Hulls. The *integer hull* P_I of a polyhedron P of $\mathbb{R}^n$ is the convex hull of its integer points, that is $P_I = \text{conv}(P \cap \mathbb{Z}^n)$.

Linear Programs. A *linear program* is the problem of optimizing a linear function over a polyhedron. In its maximization form it is:

$$\begin{aligned} &\max cx \\ &\text{subject to:} \\ &\quad Ax \leq b \\ &\quad x \in \mathbb{R}^n \end{aligned}$$

for some $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. For short we will also write $\max\{cx\colon Ax \leq b\}$ to indicate the linear program above. The dual of the linear program $\max\{cx\colon Ax \leq b\}$ is the linear program $\min\{ub\colon A^\top u = c, u \geq 0\}$.

Known Results

Here is a compendium of well-known results in polyhedral theory. The proofs are omitted. They can be found in *e.g.,* [40].

Proposition 1.5.1 (Farkas' Lemma)**.** *Let us take $A \in \mathbb{R}^{\ell \times m}$ and $b \in \mathbb{R}^\ell$. The system of inequalities $Ax \leq b$ is infeasible if and only there exists $\lambda \in \mathbb{R}^\ell_+$ such that $A^\top \lambda = 0$ and $\lambda b < 0$.*

Proposition 1.5.2 (Strong Duality)**.** *Given $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}$ if $P = \{x \in \mathbb{R}^n\colon Ax \leq b\}$ and $\{u \in \mathbb{R}^m\colon A^\top u = c, u \geq 0\}$ are nonempty, then:*

$$\max\{cx\colon Ax \leq b\} = \min\{ub\colon A^\top u = c, u \geq 0\}.$$

In addition if $P \neq \emptyset$ then $\max\{cx\colon x \in P\} = +\infty$ if and only if $D = \emptyset$.

Given $V, R \subseteq \mathbb{R}^n$ their *Minkowski sum* is:

$$V + R = \{x \in \mathbb{R}^n\colon x = v + r \text{ for some } v \in V \text{ and } r \in R\}.$$

Theorem 1.5.3 (Minkowski-Weyl Theorem)**.** *A subset $P \subseteq \mathbb{R}^n$ is a* polytope *if and only if it is the convex hull of a finite number of points of $\mathbb{R}^n$. More generally, it is a polyhedron if and only if $P = Q + C$ with Q being a polytope and C being a cone generated by a finite set.*

Theorem 1.5.4 (Decomposition Theorem for Polyhedra, see *e.g.,* [40])**.** *Let P be a pointed polyhedron, let $V = \{v^1, \ldots, v^k\}$ be the set of its vertices and $C = \{r^1, \ldots, r^\ell\}$ be the set of extreme rays of* $\text{rec}(P)$*. Then $P = \text{conv}(V) + \text{cone}(C)$.*

Let $P = \{x \in \mathbb{R}^n\colon Ax \leq b\}$ be a polyhedron of $\mathbb{R}^n$. The system $Ax \geq b$ is a *minimal representation of* P if P is strictly contained in $\{x \in \mathbb{R}^n\colon A'x \leq b'\}$ for every subsystem $A'x \leq b'$ of $Ax \leq b$.

Proposition 1.5.5. *Let $P = \{x \in \mathbb{R}^n \colon Ax \leq b\}$ be a full-dimensional polytope, with $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $Ax \leq b$ being a minimal representation of P. Then P has m facets and for every facet F of P we have $F = \{x \in P \colon A^i x = b_i\}$ for exactly one $1 \leq i \leq m$ where A^i is the i-th row of A. Conversely, for every facet-defining inequality $\alpha x \leq \beta$ for P there exists $\lambda > 0$ such that $\alpha = \lambda A^i$ and $\beta = \lambda b_i$. An inequality $cx \leq d$ is valid for P if and only if there exists $u \geq 0$ such that $uA = c$ and $ub \leq d$.*

Proposition 1.5.6. *Let $P = \{x \in \mathbb{R}^n \colon Ax \leq b\}$ be a polyhedron of $\mathbb{R}^n$. If A and b have rational entries, then P_I is a polyhedron. If $P_I \neq \emptyset$, then $rec(P) = rec(P_I)$.*

Proposition 1.5.7. *Let P be a nonempty polyhedron. Then a face F of P is a facet of P if and only if* $\dim(F) = \dim(P) - 1$.

In the following example we summarize the concepts introduced in this section.

Example 1.5.8 (Set Covering Polytope)**.** A *set covering polytope* is the convex hull of the binary solutions to a system $Az \geq \mathbf{1}$, where A is a binary matrix and $\mathbf{1} = (1, 1, \ldots, 1)^\top$. When A is the incidence matrix of a graph, it is called *vertex cover polytope*.

Proposition 1.5.9 (*e.g.,* [143])**.** *Let $SC_A = \text{conv}\{z \in \{0,1\}^m \colon Az \geq \mathbf{1}\}$ be a set covering polytope with $A \in \{0,1\}^{\ell \times m}$. We have:*

1. *SC_A is nonempty if and only if each row of A has at least one positive entry;*
2. *SC_A is full-dimensional if and only if each row of A has at least two positive entries.*

Additionally, if SC_A is full dimensional, then for every $1 \leq j \leq m$ the following hold:

(a) the inequality $z_j \geq 0$ is facet-defining for SC_A if and only if each row of A has two positive entries other than entry j;

(b) the inequality $z_j \leq 1$ is facet-defining for SC_A;

(c) every other facet-defining inequality for SC_A is of the form $\alpha z \geq \beta$ with $\alpha_k \geq 0$ and $\beta \geq 0$.

1.6 Combinatorial Optimization Problems and Integer Linear Programming

A *combinatorial optimization problem* consists in minimizing (or maximizing) a function over a finite (but usually huge) set of elements, called *feasible solutions* to the problem.

As an example, finding a shortest path linking two locations in a network can be seen as a combinatorial optimization problem. In fact, a shortest path problem in a network can be easily modeled using a weighted digraph. In general, the word "combinatorial" refers to the existence of an underlying abstract structure of finite cardinality (as the digraph of our example). Another well-known example is the Traveling Salesman Problem.

Example 1.6.1 (The Traveling Salesman Problem)**.** Let us consider a weighted digraph (D, w). The *Traveling Salesman Problem (TSP)* is to find the least weight Hamiltonian circuit of D, that is to solve $\min\{w(H)\colon H \text{ is a Hamiltonian circuit of } D\}$. A feasible solution to the TSP can be imagined as a route that visits exactly once each location in a network. The weight associated with an arc (i, j) is also called *cost* since it represents the distance covered to reach the location represented by i, starting from j. Hence, often in TSP literature, we assume that the weights are nonnegative. The weight of the circuit will be then its *cost*.

Combinatorial optimization comes in different flavors. In this thesis we will always consider the case in which a linear function (called *objective function*) is to be optimized over a finite subset of $\mathbb{R}^n$.[1]

Example 1.6.2 (The Binary Knapsack Problem)**.** A *binary knapsack set* is

$$K = \{x \in \{0,1\}^n \colon ax \leq b\}$$

for some $a > \mathbf{0}$ and $b > 0$. The corresponding *binary knapsack problem* is $\max\{cx \colon x \in K\}$ for some $c \in \mathbb{R}^n$.

Incidence Vectors. Requiring that the feasible solutions to a combinatorial optimization problem yield a finite subset of $\mathbb{R}^n$ can seem restrictive at first. Actually, this setting lets us model many combinatorial optimization problems. Let us consider a finite set U and a finite set $\mathcal{S} \subseteq 2^U$. We define the *incidence vector* of $S \in \mathcal{S}$ as the vector $\chi^S \in \{0,1\}^{|U|}$ such that $\chi^S_j = 1$ if and only if $j \in S$, for every $j \in U$.[2]

Note that $U \not\subseteq \mathbb{R}^n$ in general. It can be any type of combinatorial structure. For instance we could take $U = \{\text{subgraphs of a digraph } D\}$ and $\mathcal{S} = \{(s,t)\text{-paths of } D\}$, for fixed vertices s and t of D. In this case the set of the feasible solutions to a shortest (s,t)-path problem corresponds to $\{\chi^S \colon S \in \mathcal{S}\}$.

It turns out that the considered setting is powerful enough to capture many problems arising in real life, such as transportation problems, scheduling problems, facility location problems, to cite a few.

Combinatorial optimization problem can be reformulated as the problem of optimizing a linear function over a polytope. Using this latter property, algorithmic frameworks working quite well in practice have been designed to deal with difficult combinatorial optimization problems.

[1] Therefore, from now on combinatorial optimization refers only to this setting.

[2] We prefer the letter χ to indicate incidence vectors. Sometimes we will adopt other letters, by pointing out that they represent incidence vectors.

1.6.1 Polyhedral Approach

Let us consider the problem $\mathcal{P}$ of minimizing a linear function cx over a finite set $\mathcal{S} \subset \mathbb{R}^n$, *i.e.,* $\mathcal{P}$ is:

$$\min\{cx\colon x \in \mathcal{S}\}.$$

The set $\mathcal{S}$ is called *feasibility region*. A $x \in \mathcal{S}$ attaining the minimum in the expression above is called *optimal solution* to $\mathcal{P}$. If $x^\star$ is an optimal solution to $\mathcal{P}$ then $cx^\star$ is the *optimal value* of $\mathcal{P}$. However, it can be shown that $\mathcal{P}$ is equivalent to:

$$\min\{cx\colon x \in \text{conv}(\mathcal{S})\}.$$

Suppose to know the *linear description* of $\text{conv}(\mathcal{S})$, that is the linear inequalities corresponding to the faces of this polytope. In this case, by the equivalence above, the resolution of $\mathcal{P}$ can be performed with the resolution of a linear program. The resolution of linear programs can be done in polynomial time [119, 115]. The idea of converting a combinatorial optimization problem in the resolution of a linear program is due to Edmonds [54] and is known as *polyhedral approach*.

The effectiveness of the polyhedral approach can be undermined by the size of the linear program to which the original combinatorial optimization problem has been reduced. Indeed, the number of inequalities in the linear description of $\text{conv}(\mathcal{S})$ can be exponential in the input size. In this case, we have no guarantee that the resolution of the linear program can be accomplished in polynomial time in the input size.

Fortunately enough, Grötschel, Lovász and Schrijver have proven that the linear program $\min\{cx\colon Ax \geq b\}$ over a polyhedron can be solved in polynomial time independently on the number of inequalities in $Ax \geq b$, provided that the following problem, called *separation problem*, can be solved in polynomial time [89]:

Separation Problem. Given $x^\star \in \mathbb{R}^n$, decide whether $x^\star \in P = \{x \in \mathbb{R}^n\colon Ax \geq b\}$ or find a inequality $\alpha x \geq \beta$ valid for P such that $\alpha x^\star < \beta$.

The above important result is also known as the "equivalence of separation and optimization" and it is based on the ellipsoid method [119].

Integer Linear Programming Formulations. Given a combinatorial optimization problem $\mathcal{P}$ an *integer linear programming formulation* for $\mathcal{P}$ is $\mathcal{F} = \{x \in \mathbb{Z}^n\colon Ax \geq b\}$ such that the elements of $\mathcal{F}$ are in one-to-one correspondence with the feasible solutions to $\mathcal{P}$. The *linear relaxation* of $\mathcal{F}$ is then $\mathcal{L} = \{x \in \mathbb{R}^n\colon Ax \geq b\}$. With abuse of language we call $\mathcal{L}_I$ the *integer hull* of the formulation $\mathcal{F}$. Let $\mathcal{L}$ be the linear relaxation of an integer linear programming formulation $\mathcal{F}$ for a combinatorial optimization problem requiring to optimize cx. *Solving the linear relaxation* of $\mathcal{F}$ means to optimize cx over $\mathcal{L}$. Problems requiring to optimize a linear function cx over sets of the form $\{x \in \mathbb{Z}^n\colon Ax \geq b\}$ are called *integer linear programs*.

1.6.2 Cutting Plane Method

The *cutting plane method* aims at solving the integer linear program over a polyhedron $P = \{x \in \mathbb{Z}^n \colon Ax \geq b\}$ with linear objective function cx. In the discussion, we assume that the integer linear program is $\min\{cx \colon Ax \geq b, x \in \mathbb{Z}^n\}$.

The cutting plane method initially solves the linear program $\min\{cx \colon Ax \geq b, x \in \mathbb{R}^n\}$ getting an optimal solution $x^\star$. At this point it solves the separation problem for $x^\star$ and P_I: if $x^\star \in P_I$ then $x^\star$ is also optimal for the starting integer linear program. Otherwise, the separation problem produces $\alpha x \geq \beta$ valid for P_I and violated by $x^\star$. The inequality $\alpha x \geq \beta$ is added to the system $Ax \geq b$. The process above is repeated on the optimal solution to the linear program $\min\{cx \colon Ax \geq b, \alpha x \geq \beta, x \in \mathbb{R}^n\}$.

The inequalities generated during the separation step are called *cutting planes*. The cutting plane method is a general framework. The generation of cutting plane methods can vary. One of the first examples of cutting planes is due to Gomory [83]. They are generated from the output of the simplex method. Under nonrestrictive conditions Gomory's cutting plane method terminates after a finite number of iterations, see p. 356 in [174].

The same method can be used to solve linear programs with a huge number of constraints. In this case one initially discards a large subset $\mathcal{C}$ of inequalities from the constraint set, so as to obtain a tractable relaxation of the initial problem. Then the cutting plane method is applied to the solution to this relaxation, by solving its separation problem with respect to the constraints in $\mathcal{C}$.

1.6.3 Branch-and-Cut Algorithm

In general, the cutting plane method as described above converges very slowly to an optimal solution to an integer linear program. Nevertheless, it is nowadays successfully used in the so-called *branch-and-cut algorithm*. We explain how this algorithm can solve a combinatorial minimization problem. Suppose to have an integer linear programming formulation for this problem. The branch-and-cut algorithm solves the linear relaxation of the formulation. If its solution is also integer, then it is optimal and the algorithm terminates. Otherwise a *separation step* takes place. It performs several iterations of the cutting plane method. This generates, from a prescribed set of inequalities valid for the integer hull of the formulation, one or more inequalities violated by the optimal solution to the current linear relaxation. The inequalities are added to the linear relaxation and the resulting linear program is solved again. The separation step continues until no violated inequality is found by the last fractional solution. At this point the algorithm performs a *branching step*. In general, the goal of the branching step is to partition the feasibility region in smaller sets, so that the integer linear programs on these sets are easier to solve. This can be done *e.g.,* by choosing a fractional coordinate of the fractional solution and by separately setting its value to an integer value (among the possible ones). Hence a number of new integer linear programs is generated by the branching step and added to a list of unsolved problems. Solutions to these problems are, by construction, upper bounds for the

value of the original problem. The algorithm can discard an unsolved problem from its list whenever the lower bound obtained from its linear relaxation is higher than the best upper bound found so far (the solution associated with this upper bound is called *incumbent*). This latter mechanism is called *bounding step*. It is useful to reduce the number of integer linear programs to be solved during the algorithm. After the bounding step, all unsolved problems still in the list can be solved by applying separately the same method described above. When no unsolved problem is left, an optimal solution to the original problem is the last incumbent.

When the algorithm ends it has solved a number of integer linear programs that are organized in a tree (called *search tree*): indeed each of these linear programs generates several integer linear programs during the branching step. Each integer linear program solved during the algorithm is called *node* of the search tree.

The role of the cutting plane method in the branch-and-cut algorithm is to produce good lower bounds on the optimal value of the problem so that many integer linear programs are discarded during the bounding step. In other words, it is used to avoid a complete exploration of the feasibility region and hence to limit the size of the search tree. In the context of branch-and-cut algorithms the linear inequalities generated dynamically during the separation step are called, among others, *strengthening cuts* or *strengthening inequalities*.

1.6.4 Heuristics

In the branch-and-cut algorithm described in previous section the bounding step is crucial. It relies on two ingredients. Good lower bounds obtained by solving linear programs, possibly reinforced in the separation step, and good upper bounds. Upper bounds are automatically produced by the branch-and-cut algorithm, when it finds feasible solutions to the original problem. However, one can provide the algorithm with upper bounds of good quality by using heuristics for the problem to be solved.

Sometimes, heuristics guarantee a certain quality of the found solution. Consider the problem $\mathcal{P}$ of minimizing a linear function over a finite set $\mathcal{S} \subseteq \mathbb{R}^n$. An *$\alpha$-approximation algorithm* for $\mathcal{P}$ is a polynomial algorithm that, for every c, returns $\tilde{x} \in \mathcal{S}$ such that the optimal value of $\min\{cx\colon x \in \mathcal{S}\}$ is at least $\frac{c\tilde{x}}{\alpha}$. We refer to Williamson and Shmoys' book [188] for more details on approximation algorithms.

Other types of heuristics belong to the category of *local search* methods [127, 128]. These methods use a so-called neighborhood and a descent method. Given a solution to a combinatorial optimization problem, its *neighborhood* is a set of solutions considered close to it and that can be produced in a few steps by a *neighborhood operator*. The operator in general modifies the current solution to produce new feasible solutions. The *descent methods* are used to choose a solution in the neighborhood of the current solution and to decide whether the latter has to be replaced by the new one. The simplest descent method consists in taking always the best solution according to its value. This method is called

steepest descent method. In the basic local search methods the neighborhood has a small size hence the descent can in general test all possibilities. On the other hand, doing so the local search often gets stuck in local optima. Hence a number of strategies have been designed to escape from local optima. The combination of these strategies with the local search framework has given rise to many different heuristic schemes. We list some of them.

- *Tabu Search*: invented by Glover [78, 79, 80] in this method a worsening of the current solution is admitted. In addition, to avoid of revisiting repeatedly the same solution, some solutions are marked as *tabu* according to some rule (*e.g.,* if a solution has been previously visited within a short-term period).

- *Large Neighborhood Search*: introduced by Shaw [177] it consists in a local search in which the neighborhood size grows exponentially with the instance size. The neighborhood of a solution is defined as the set of solutions that can be obtained by partially destroying the current solution and subsequently rebuilding the destroyed solution. Both phases of the neighborhood operator can be accomplished in several ways, *e.g.,* by a random destruction and a greedy repair.

- *Variable Neighborhood Search (VNS)*: invented by Mladenović and Hansen [139] it consists in considering more than one neighborhood for the current solution x. For each neighborhood in a given order, it produces a random solution in that neighborhood. Then a local search is applied starting from this new solution. If the local optima $x^\star$ reached by the local search is better than the current solution x, then the new current solution is set to $x^\star$. Now the process repeats on $x^\star$ by considering the first neighborhood in the order. If $x^\star$ is not better than x then the VNS chooses the next neighborhood of x in the order.

- *Generalized Variable Neighborhood Search (GVNS)*: it is identical to the VNS apart from the local search step. The latter is performed by using many neighborhoods in a given order. That is, given the starting point x' in a neighborhood of the current solution, x' is improved by considering the first neighborhood in the order. When no improvement is obtained, another neighborhood is selected according to the order and so on until having tried all neighborhood. This special type of local search step is called *Variable Neighborhood Descent (VND).*

Part I

The Double Traveling Salesman Problem With Multiple Stacks

Chapter 2

The Double Traveling Salesman Problem with Multiple Stacks

In this chapter we present the problem studied in the first part of this thesis, the double traveling salesman problem with multiple stacks. In Section 2.1 we first give an informal description of the problem, by motivating its applicability in logistics. Then we formally describe the problem in terms of graphs. In Section 2.3 we provide a literature survey of routing and pickup and delivery problems. The material in Section 2.3 is useful to find the similarities between the double traveling salesman problem with multiple stacks and problems already studied in the literature. In Section 2.4 we report the results obtained on the double traveling salesman problem with multiple stacks in previous research works. The content of Section 2.4 is useful to better locate the contributions of this thesis to the problem.

2.1 Introducing the Double Traveling Salesman Problem with Multiple Stacks

The double traveling salesman problem with multiple stacks (abbreviated to *double TSP with multiple stacks*) is a combinatorial optimization problem introduced in 2009 by Petersen and Madsen in their paper entitled "The double travelling salesman problem with multiple stacks – Formulation and heuristic solution approaches" [158]. In this section we will briefly see how the double traveling salesman problem with multiple stacks models a specific situation arising in real-world logistics. To this purpose, our description goes along the same lines of Petersen and Madsen's paper which originally motivated the interest in the problem. This section is meant to be a gentle introduction to the double traveling salesman problem with multiple stacks. For a detailed account of the results obtained by the scientific community on this problem, we refer the reader to Section 2.4 of this chapter.

Let us consider a logistic company that needs to satisfy a set of orders structured as follows. First, a prescribed amount of pallets of identical dimension (henceforth called

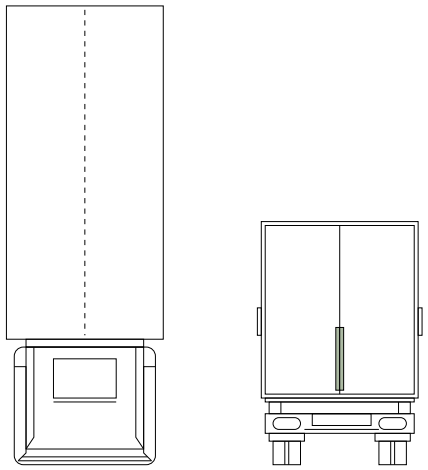


Figure 2.1: This truck has two rows obeying a last-in-first-out principle. The rows can be imagined as vertical stacks.

items) has to be collected by performing a route in a "pickup region". Next, each collected item must be delivered to a corresponding customer in a "delivery region", very far from the first one. The company has at its disposal two truck stations (hereafter called depots), one in each region, and a container in which the items can be packed. The items are all identical from a packing point of view, *i.e.,* they occupy the same volume in a row. The pickup route is performed by a truck onto which the container is loaded. This route has to start from the depot, and it has to visit exactly once each location of the region where an item is present before coming back to the depot. Each time a location other than the depot in the pickup region is visited, the corresponding item is loaded into the container. The container is organized in rows of identical capacity. This implies that each item is loaded in exactly one row without exceeding the given capacity. In addition each row obeys a last-in-first-out principle. This means that before delivering an item packed in a row, all items collected after it and packed in the same row, must be delivered too. Instead, items packed in two distinct rows do not generate any constraint in the unloading operations. Another important constraint that the company must observe is that a repacking of the items in the rows is not permitted, *i.e.,* once an item has been assigned to a row it cannot be reassigned. In Figure 2.1 we schematically illustrate two rows as those described above. Figure 2.1 also clarifies the name of the problem: the rows can be seen as vertical stacks each subject to a last-in-first-out rule.

Once the pickup route has been completed, the container is transported by a long-distance means of transport (*e.g.,* a train or a ship) to the depot of the delivery region. Starting from this depot, the container is transported again by a truck that visits exactly once each customer in the delivery region before coming back to the depot. Clearly, the delivery route will partially depend on the assignment of items to the rows, because of the last-in-first-out principle. The goal of the company is to minimize the sum of the total distances traveled for the pickup and the delivery routes, without violating the constraints.

The long-distance transportation of the container from the pickup region to the delivery region is not taken into account in the distance to be minimized.

The situation described above can occur in real life. In fact, the double TSP with multiple stacks was introduced in the scientific literature during the collaboration between the authors of [158] and a company producing fleet management software for small and medium size transportation companies. The double TSP with multiple stacks was encountered at one of the perspective customers of this software company. Here, we motivate the hypothesis of the problem, again following the discussion in [158]. First, it is not very restrictive to assume that the items to be collected are identical from a packing point of view: indeed, in many situations, they are standard Euro Pallets that are arranged in several rows (three in the original description of the problem). In addition, the forbidden rearrangement of the items in the rows can be motivated by a number of arguments: fragile or heavy content of the Euro Pallets for instance, but also costs deriving from handling the pallets as well as union restriction compliance. Finally, we do not take into account the cost to transfer the container from the pickup region to the delivery region. Indeed, one could imagine that this cost is constant due to the fact that the two depots are the start and end points in both the pickup and delivery routes and that they are very far from each other.

2.2 The Double TSP with Multiple Stacks in Terms of Graphs

We formalize the problem described above by using the graph terminology. To this end, we assume that n, s and q are three positive integers. Informally speaking, n is the number of items to be transported, s the number of stacks[1] of the vehicle and q their capacity. Unless differently stated, we will assume that $n \geq 3$, and $sq \geq n$. The second condition ensures that the vehicle can contain all the items picked up in the pickup city.

In the double TSP with multiple stacks, a vehicle performs a pickup route to collect n items in a city. Subsequently it performs a delivery route to deliver the items to n customers in another city. Items and customers are *paired*, meaning that each item corresponds to exactly one customer. Hence the transportation networks of the two cities are modeled by two arc-weighted digraphs (G_n, c^1) and (G_n, c^2), where $G_n = (V, A)$ is the complete digraph with $V = \{0, \ldots, n\}$ as vertex set and $c^1, c^2 \in \mathbb{R}^{|A|}$. In this representation, (G_n, c^1) is the pickup city and (G_n, c^2) is the delivery city. In both digraphs, vertex 0 represents the depot. For every $i \in \{1, \ldots, n\}$, vertex i of (G_n, c^1) represents the location of the i-th item, whereas vertex i of (G_n, c^2) is the customer to which the i-th item has to be delivered. Finally, c^1_{ij} (resp. c^2_{ij}) represents the cost to visit i immediately before j in the pickup (resp. delivery) city, for all distinct $i, j \in V$. Note that we use a digraph because

[1]From now on in this manuscript, the term "stack" will be preferred to "row", since it is widely adopted in the scientific works on the double TSP with multiple stacks.

we consider the asymmetric version of the problem, where in general $c^T_{ij} \neq c^T_{ji}$ for some $i, j \in V$ and $T = 1, 2$. Unless differently stated, we consider paths and circuits of G_n as sets of arcs.

Vertex 0 corresponds to the depot, hence every Hamiltonian circuit of G_n *induces* a linear ordering on $\{1, \ldots, n\}$. More precisely, if $H = \{(0, v_1), (v_1, v_2), \ldots, (v_{n-1}, v_n), (v_n, 0)\}$ is such a Hamiltonian circuit, the corresponding linear ordering, denoted $\prec_H$, is defined by $v_1 \prec_H v_2 \prec_H \cdots \prec_H v_n$. Conversely, each linear ordering $\prec$ on $\{1, \ldots, n\}$ corresponds to a Hamiltonian circuit H of G_n because the latter is complete. Namely, if $v_1 \prec v_2 \prec \cdots \prec v_n$ the corresponding Hamiltonian circuit is $H = \{(0, v_1), (v_1, v_2), \ldots, (v_n, 0)\}$. We will also write $H = 0, v_1, \ldots, v_n, 0$ for short.

The pickup and the delivery routes are represented by a pair of Hamiltonian circuits of G_n. Conversely, let us consider two Hamiltonian circuits H_1 and H_2 of G_n. They represent a pickup and a delivery routes feasible for the double TSP with multiple stacks if and only if they allow to construct a disposition of the items in s stacks of capacity q that is consistent with the last-in-first-out rule. This means that the following requirements must be fulfilled. First, each item is in exactly one stack after the pickup route has been performed. Additionally, the capacity of the stacks is not exceeded. Finally, in each stack the items are ordered from the bottom to the top exactly as in the order in which they appear in H_1. Of course, the last-in-first-out rule is respected for this stack if this order is the reverse of that of H_2.

Hence, we define an *s, q-loading plan for* the pair (H_1, H_2) as a partition of $\{1, \ldots, n\}$ into s totally ordered sets $\{(Q_1, \prec_{Q_1}), \ldots, (Q_s, \prec_{Q_s})\}$ such that $|Q_i| \leq q$ and $k \prec_{Q_i} \ell$ implies $k \prec_{H_1} \ell$ and $\ell \prec_{H_2} k$ for $i = 1, \ldots, s$ and for all $k, \ell \in Q_i$.

A pair (H_1, H_2) of Hamiltonian circuits of G_n is *s, q-consistent* if there exists an s, q-loading plan for (H_1, H_2).

Note that if a pair of Hamiltonian circuits (H_1, H_2) is s, q-consistent, so is (H_2, H_1) (take the same s, q-loading plan with reversed linear orderings). Hence, we will also say that H_1 and H_2 are s, q-consistent.

When q is infinite, we write *s-loading plan* and *s-consistent*. The property of a pair of Hamiltonian circuits of being s, q-consistent is called *s, q-consistency* (*s-consistency* if q is infinite) or simply *consistency* when no confusion may arise.

Given the number of items n and the cost vectors c^1 and c^2, the double TSP with s stacks of capacity q consists in finding an s, q-consistent pair (H_1, H_2) of Hamiltonian circuits of G_n that minimizes $c^1(H_1) + c^2(H_2)$ and in finding an s, q-loading plan for (H_1, H_2).

Here is an example to illustrate the notions introduced above. It also shows that, in general, several loading plans could be available for an s, q-consistent pair of Hamiltonian circuits.

Example 2.2.1. Let us consider an instance of the double TSP with two stacks of capacity three. We assume that six items are to be delivered to six corresponding customers. A feasible solution to this instance is given by the $2, 3$-consistent pair of Hamiltonian circuits:

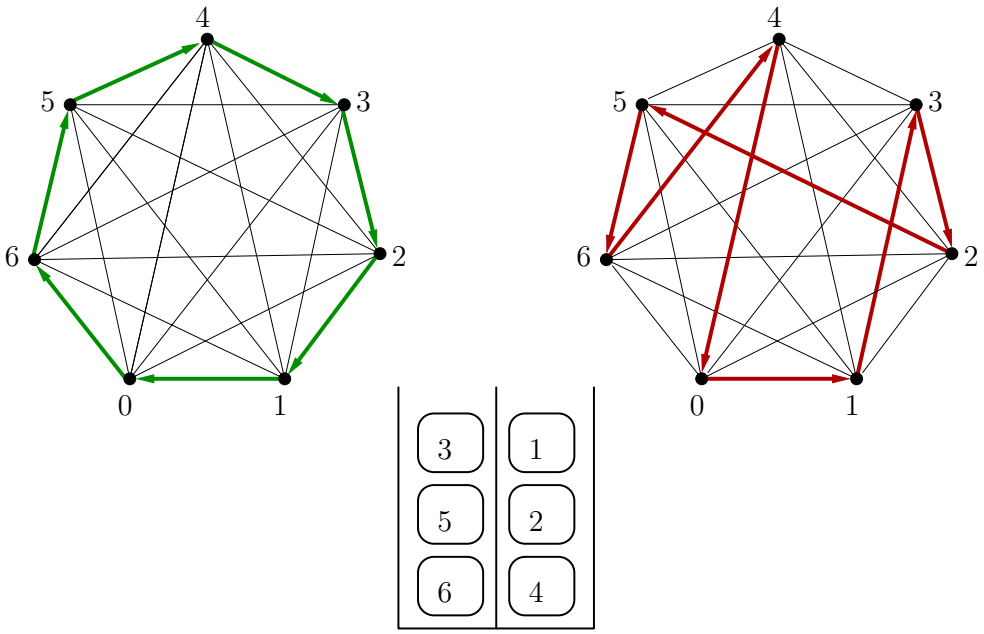


Figure 2.2: On the left a pickup circuit H_1 for the instance defined in Example 2.2.1, on the right a delivery circuit H_2 such that (H_1, H_2) is $2,3$-consistent. In the center of the figure, the disposition of the items corresponding to the $2,3$-loading plan L defined in Example 2.2.1.

- $H_1 = 0, 6, 5, 4, 3, 2, 1, 0$
- $H_2 = 0, 1, 3, 2, 5, 6, 4, 0$

The $2, 3$-loading plan $L = \{(Q_1, \prec_{Q_1}), (Q_2, \prec_{Q_2})\}$ defined by

- $Q_1 = \{3, 5, 6\}$ and $6 \prec_{Q_1} 5 \prec_{Q_1} 3$
- $Q_2 = \{1, 2, 4\}$ and $4 \prec_{Q_2} 2 \prec_{Q_2} 1$

proves the $2, 3$-consistency of (H_1, H_2). In Figure 2.2 we give a representation of L in terms of disposition of items in the stacks. Another $2, 3$-loading plan for the same pair of circuits is $L' = \{(Q'_1, \prec_{Q'_1}), (Q'_2, \prec_{Q'_2})\}$ defined by:

- $Q'_1 = \{2, 5, 6\}$ and $6 \prec_{Q'_1} 5 \prec_{Q'_1} 2$
- $Q'_2 = \{1, 3, 4\}$ and $4 \prec_{Q'_2} 3 \prec_{Q'_2} 1$

It is noteworthy that in two special cases the double TSP with multiple stacks is equivalent to a TSP. The first special case is when the vehicle has only one stack, that is $s = 1$. Then the delivery circuit must be the reverse of the pickup circuit. Hence the problem is equivalent to find the Hamiltonian circuit of G_n that minimizes $c(H)$, where $c_{ij} = c^1_{ij} + c^2_{ji}$ for every distinct $i, j \in V$. The second special case arises when $s \geq n$, *i.e.,* when the vehicle has at least as many stacks as the number of items. In this case, the pickup and the delivery circuits are independent, since each item can be put in a different stack. Hence the problem amounts to find two Hamiltonian circuits H_1 and H_2 of G_n that minimize, respectively $c^1(H_1)$ and $c^2(H_2)$.

2.3 Links with Other Routing and Pickup and Delivery Problems

The double TSP with multiple stacks exhibits similarities with problems previously studied in transportation science. Clearly, it is inherently connected to the TSP, and in fact we have shown that it is a generalization of the latter. Additionally, it involves the transportation of items to customers via pickup and delivery operations. Problems taking into account the latter aspect are generically known as pickup and delivery problems. Pickup and delivery problems are widely studied because of their importance and ubiquity in modern transportation systems.

It is useful to classify such problems, since they attracted the interest of many researchers, consequently generating a large amount of scientific works. In fact, Berbeglia *et al.* [20] propose a classification scheme based on the structure of the pickup and delivery locations, on the type of operations allowed at a given location and on the number of vehicles involved in the transportation process. Following [20] we then distinguish *many-to-many* problems from *one-to-many-to-one* problems and from *one-to-one* problems. In

the first case, any vertex can be the source or the destination of any commodity. In one-to-many-to-one problems an initial set of commodities is provided by a depot and these commodities are destined to some demanding customers (corresponding to the delivery locations); in addition, some customers (corresponding to pickup locations) provide additional commodities destined to the depot. In one-to-many-to-one problems pickup and delivery locations can coincide. Finally, in one-to-one problems each commodity is located at a given origin and destined at a given customer.

On top of this structural classification, Berbeglia *et al.* [20] also introduce the notation *PD* that designates those problems in which at each location a pickup and a delivery operation take place simultaneously exactly once during the route. In other problems (denoted as *P-D*), the pickup and delivery operations associated with a given location can be performed separately. Finally, in problems indicated as *P/D*, at each location either a pickup or a delivery operation take place but not both.

According to Berbeglia *et al.*'s classification described above, we get that the double TSP with multiple stacks is a single vehicle one-to-one P/D problem. However, the double TSP with multiple stacks presents some complicating constraints due to the last-in-first-out rule governing the stacks of the vehicle. Problems of this type have also been studied, generalizing some pickup and delivery problems in which the disposition of the items in the transport vehicle is not considered.

Since we want to locate the double TSP with multiple stacks in the vast literature on routing problems and pickup and delivery problems, we first pay attention to the relevant scientific works in these areas. For a detailed account of the literature concerning the double TSP with multiple stacks the reader is instead referred to Section 2.4.

2.3.1 The Traveling Salesman Problem

The TSP, defined in Example 1.6.1, is to find a Hamiltonian circuit of minimum weight in a complete arc-weighted digraph. Although its initial motivation lies in logistics, the TSP finds applications in several other areas: examples of TSP have been observed in scheduling, frequency assignment problems and cellular manufacturing to cite a few, see Gutin and Punnen's book [95] for more details. "Traveling salesman problem" is a name appeared probably for the first time in 1949 in a technical report by Robinson [168]. It designates however the same problem in which other researchers have been interested earlier (*e.g.,* Menger [134], Mahalanobis [133]).

It is customary to consider two distinct cases of the TSP. In the first case, the weight of arc (i, j) equals the weight of arc (j, i) for every distinct vertices i, j of the digraph. In this case we call the problem *Symmetric TSP (STSP)*; otherwise, the problem is called *Asymmetric TSP (ATSP)*. The STSP is usually restated as the problem of finding a least cost Hamiltonian circuit in a weighted complete undirected graph. In the following, we will say simply TSP if the distinction between the symmetric and asymmetric cases is not relevant to the discussion. We mention that, quite surprisingly, each ATSP on a digraph can be expressed as a STSP on a suitable undirected graph having twice as many nodes [112].

The TSP is an NP-hard problem, as first proved in 1972 by Karp [116]. Nevertheless, one of the greatest advances in combinatorial optimization has been the design of algorithms that made possible to practically solve TSP instances of considerable size. In particular, branch-and-bound and branch-and-cut methods turned out to be very effective for the exact resolution of the TSP. We now review the main works on the TSP, focusing in particular on exact methods. We consider separately the symmetric and the asymmetric case.

Symmetric Traveling Salesman Problem

The solutions to the STSP in a graph $G = (V, E)$ with $|V| \geq 5$ can be described by means of their incidence vectors which are the solutions to the following integer linear programming formulation:

$$x(\delta(v)) = 2 \qquad \forall v \in V, \tag{2.1}$$

$$x(\delta(S)) \geq 2 \qquad \forall S \subseteq V \text{ such that } 2 \leq |S| \leq |V|/2, \tag{2.2}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E. \tag{2.3}$$

It is based on binary variables each corresponding to an edge of G. Constraints (2.1) ensure that each vertex has degree 2. Constraints (2.2) ensure that the solution corresponds to a connected subgraph of G. Constraints (2.2) are called *subtour elimination constraints*.

The integer hull of this formulation is called *STSP polytope* and denoted $STSP_n$ when G has n vertices. It is well-known that $\dim(STSP_n) = \binom{n}{2} - n$, see Queyranne and Wang [162].

In [95] p. 34 it is reported that a simple combination of a branch-and-bound algorithm and a cutting plane method that generates dynamically the subtour elimination constraints (using a minimum cut-based separation routine) is able to solve almost all problems of the TSPLIB set [165] in very few iterations.

So far, probably the best method to tackle large non-trivial instances of the STSP is the branch-and-cut algorithm. According to the discussion of Chapter 2 in Gutin and Punnen's book on the TSP [95], besides the problem-defining subtour elimination constraints, other good families of inequalities to be added during the separation step of a branch-and-cut algorithm for the STSP could be the *comb inequalities* [37, 91] and the *path inequalities* [42].

One of the best solvers for the STSP based on the branch-and-cut paradigm is CONCORDE by Applegate, Bixby, Chvátal and Cook [7]. The largest instance from the TSPLIB set solved using this solver has 85900 vertices. For the implementation details of CONCORDE we refer the reader to Chapter 2 in [95]. Since any instance of the ATSP can be transformed in one instance of the STSP, CONCORDE can be used to tackle the asymmetric case. In addition, it has been observed experimentally, see *e.g.,* [167], that the performance of CONCORDE on large ATSP instances remains comparable to the performances of other state-of-the-art branch-and-cut algorithms for the ATSP, two of which will be briefly described later.

Asymmetric Traveling Salesman Problem

According to Gutin and Punnen's book on the TSP [95], a "systematic study of the ATSP as a combinatorial optimization problem began with the work of Dantzig, Fulkerson, and Johnson [47]".

Dantzig, Fulkerson and Johnson's Formulation. Dantzig, Fulkerson and Johnson propose in [47] the following integer linear programming formulation for the ATSP in the complete digraph $G = (V, E)$:

$$x(\delta^+(i)) = 1 \qquad \forall i \in V, \tag{2.4}$$

$$x(\delta^-(i)) = 1 \qquad \forall i \in V, \tag{2.5}$$

$$x(\delta^+(S)) \geq 1 \qquad \forall\ S \subset V \setminus \{0\} \text{ such that } 2 \leq |S| \leq |V| - 1, \tag{2.6}$$

$$x_{ij} \in \{0, 1\} \qquad \forall \text{ distinct } i, j \in V. \tag{2.7}$$

It involves binary variables, each representing one arc of G. Hence these variables are called *arc variables*. Constraints (2.4) and (2.5) are called *assignment constraints* and they ensure that each vertex has exactly one leaving and exactly one entering arc in a solution; constraints (2.6) are called *subtour elimination constraints* and they ensure that the solution is a connected subgraph of the given digraph. The resulting formulation has exponential size in the number of vertices of the digraph, since the subtour elimination constraints are exponentially-many. Nevertheless, the optimization over its linear relaxation can be performed efficiently, *e.g.,* with the procedure introduced by Padberg and Rinaldi [149]. This is due to the fact that the subtour elimination constraints can be separated in polynomial time by computing minimum weight cuts in G [150].

The integer hull of the formulation is called *ATSP polytope* and denoted $ATSP_n$ when G has n vertices. It is equivalently the convex hull of the incidence vectors of the Hamiltonian circuits of the complete digraph with n vertices. It is well-known that $\dim(ATSP_n) = n^2 - 3n + 1$, see Grötschel and Padberg [92].

Since the STSP is a special case of the ATSP, all inequalities valid for the STSP polytope can be transformed into inequalities valid for the ATSP polytope, see *e.g.,* p. 120 of Gutin and Punnen's book [95]. The inequalities arising in this way are also called *symmetric inequalities*. The symmetric inequalities do not characterize completely the ATSP polytope. In fact, several families of inequalities have been found for the ATSP that cannot be obtained from the STSP, see again Chapter 2 of [95]. To cite two of non-symmetrical inequalities (commonly used in branch-and-cut algorithms for the ATSP), we have the D_k^+ and the D_k^- inequalities found by Grötschel and Padberg [87, 92] and the *Odd Closed Alternating Trail (CAT)* inequalities by Balas [8]. Both these families are facet-defining for the ATSP (except two special cases of the Odd CAT inequalities), see [61] for the D_k^+ and D_k^- inequalities and [8] for the Odd CAT inequalities.

Other Formulations. The literature on the TSP has shown that formulation (2.4)–(2.6) is by far the most useful formulation to tackle the TSP. However, other formulations

for the TSP have been proposed. A well-known formulation is the MTZ formulation, named after Miller Tucker and Zemlin who presented it for the first time in 1960 [136]. The simple idea behind this formulation is to consider, beside the arc variables seen above, also new variables u_j for each vertex j of the digraph different from a fixed vertex k. Then the value of u_j in an integer solution to the MTZ formulation, corresponds to the position of vertex j in the circuit (starting from k). Using the arc variables and these new position variables, a polynomial number of constraints can be used to guarantee that an integer solution to the formulation represents a connected subgraph. In the following, we will call these constraints *subtour breaking inequalities*. The lower bound yielded by the MTZ formulation turns out to be quite weak, hence the formulation is rarely used to tackle the ATSP.

On the other hand, the MTZ formulation inspired a number of subsequent works. Firstly, Desrochers and Laporte [49] reinforced the MTZ formulation by replacing its subtour breaking inequalities with three new families of inequalities involving again the arc and the position variables. In fact, one of the three families is obtained by lifting the subtour breaking inequalities — see *e.g.,* [93] for the notion of lifting.

Reformulating the MTZ subtour breaking inequalities, Gouveia and Pires [85] find new formulations for the ATSP. They imagine to fix a starting vertex for the Hamiltonian circuits in the digraph, say v_0. Then they introduce *precedence variables* y_{ij} for every distinct vertices i, j in the digraph. Each of these variables is 1 in a solution exactly when i is in the path between v_0 and j in the Hamiltonian circuit corresponding to the solution. Using these new variables they propose four new formulation for the ATSP and establish the relations between their linear relaxations in terms of yielded lower bounds.

Building upon the same precedence variables, Sarin *et al.* [171] have introduced a formulation to deal with the ATSP in which some precedence relations between vertices can be imposed. We refer the reader to Section 3.2.1 of this thesis for a formal presentation of Sarin *et al.*'s formulation. We mention that the ATSP in which some precedence constraints between vertices are present can be also modeled only using arc variables, see *e.g.,* [13]. Subsequently, Gouveia and Pesneau [84] have given a number of strengthening inequalities for the formulations of Gouveia and Pires and Sarin *et al.* and tested them in a computational test session, to assess their quality. In Gouveia and Pesneau's work both polynomial-size and exponential-size families of strengthening inequalities are found and the corresponding separation algorithms are discussed.

We refer to Roberti and Toth's survey [167] for other polynomial-size formulations for the ATSP and for a comparative analysis of their linear relaxation as well as their behavior in a computational framework.

Branch-and-Bound Algorithms for the ATSP. According to Roberti and Toth's survey on the ATSP [167], a variety of branch-and-bound methods have been proposed for this problem [18, 70, 179, 28, 10, 137, 156, 155, 64, 27]. Following Roberti and Toth three of the most effective ones are those of Fischetti and Toth [64], of Pekny and Miller [155] and Carpaneto *et al.* [27]. For their description we follow [95].

The methods in [64, 155, 27] are based on Carpaneto and Toth's method [28]. The latter solves at each iteration a so-called *Modified Assignment Problem (MAP) relaxation.* It consists in discarding the subtour elimination constraints from the formulation, in fixing the value of some arc variables to 0 or to 1 and in solving to optimality the resulting integer linear program. The resolution of a MAP relaxation can be done efficiently as it is based on a $O(n^3)$ time algorithm for the *assignment problem (AP)*, see [125]. A solution to a MAP corresponds to a subgraph being either a Hamiltonian circuit or the union of circuits. In the latter case, the algorithm branches on the variables corresponding to the arcs of one of these circuits, by fixing their value to 0 or 1. This step generates new MAP's and it is proven to be correct for partitioning the feasibility region.

Fischetti and Toth [64] use the same branching scheme, and calculate the lower bound by combining other relaxations of $ATSP_n$, *i.e.,* they consider, besides the AP relaxation, also the *Spanning r-Arborescence Problem (r-SAP)* and the *Spanning r-Antiarborescence Problem (r-SAAP)* relaxations, obtained by removing from formulation (2.4)–(2.6) one of the two families of assignment constraints. Both these relaxations can be solved in $O(n^2)$ time, see *e.g.,* [181]. Each relaxation produces a lower bound to the ATSP. The lower bounds are combined using the *additive bounding* technique [63]. It consists in generating a sequence of ATSP instances starting from the original one and in calculating for each of them a lower bound using a different relaxation. It can be proven that the sum of the computed lower bounds is a lower bound to the original ATSP instance.

Pekny and Miller [155] propose an effective procedure obtained by parallelizing the original algorithm of Carpaneto and Toth [28]. It also has some new features. At root node the cost matrix of the instance is made sparse by the removal of costs greater than a prescribed threshold. The correctness of the algorithm is proven using a condition that certificates when an optimal solution for the modified cost matrix is also optimal for the original cost matrix. A *patching algorithm* due to Karp [117] is used to merge two non-Hamiltonian circuits obtained from the resolution of the AP, in a circuit containing all their vertices.

The algorithm of Pekny and Miller [155] exhibits comparable performance with the last effective branch-and-bound algorithm we describe. It is due to Carpaneto *et al.* [27]. With respect to the algorithm of [155], the cost matrix is made sparse only at root node, by removing the arcs that cannot be in an optimal solution. This is accomplished using bounding arguments on the reduced costs obtained by solving AP relaxations. Indeed the reduced costs indicate how much a solution will increase its value under setting variables to 1. If the increase for a variable is too high, then that variable cannot be in an optimal solution. With respect to the algorithm of [28], the number of subtours that constitute an optimal solution to a MAP at a given node is decreased by using a fast "subtour merging" procedure. This choice is made because some experimental tests indicated that the nodes in the search tree that generate few new nodes correspond to MAP's whose solutions do not contain many subtours.

Branch-and-Cut Algorithms for the ATSP. Another exact algorithm that turned out to be very effective in the resolution of large ATSP instances is the branch-and-cut method. This heavily relies on the polyhedral study of the ATSP polytope since inequalities corresponding to the faces of the ATSP polytope (and in particular its facets) are added during the separation step. In successful branch-and-cut algorithms for the ATSP, both symmetrical and non-symmetrical inequalities are added during the separation step. The separation problem for the symmetric inequalities of the ATSP polytope can be performed by applying directly the separation routines used for the STSP.

The algorithm of Fischetti and Toth [65] is one of the most effective for the ATSP. It combines the common branch-and-cut framework with the resolution of the so-called *AP pricing problem* which lets one work on the problem with a small set of variables, dynamically updated, see [65] for more details. The algorithm uses both symmetric and non-symmetric inequalities as strengthening cuts. It uses as strengthening cuts the 2-*matching inequalities* (a special case of comb inequalities), the D_k^+ and the D_k^- inequalities, as well as the Odd CAT inequalities. Also the subtour elimination constraints are added dynamically during the separation step. The separation of all these families is simplified by two procedures called *clique lifting* [11] and *shrinking* [150]. The branch-and-cut algorithm in [65] finally branches on variables that maximize a score function.

Another exact algorithm for the ATSP is the one of [62]. Its main difference with the algorithm of [65], is in the branching scheme: in [62] the variables on which the algorithm branches are those that have been persistently fractional in the last optimal solutions to the linear programs solved during the current node. This more sophisticated criterion is called *Fractionality Persistency*. In a computational comparison made in [167], CONCORDE (after the transformation of the ATSP instances into STSP instances) and the algorithm of [62] turned out to be the best ones to solve large real-world instances of the ATSP.

2.3.2 Pickup and Delivery Problems

We now consider routing problems involving pickup and delivery operations. These belong to a wide area of investigation thus an exhaustive survey of the literature is not provided. We focus our attention on those problems exhibiting similarities with the double TSP with multiple stacks. In particular, we consider only *static* problems, *i.e.,* problems in which the information on the pickup and delivery requests is known in advance. Also, only single-vehicle cases are treated here. Finally, we are interested in problems where a location in a transportation network can be either a pickup or a delivery location, but not both. For more versions of problems with pickup and delivery requests, we refer the interested reader to [17, 52, 153, 154].

Problems with Backhauls and Linehauls

We first report *problems with backhauls and linehauls*. These are similar to the double TSP with multiple stacks for the capacity constraints of the vehicle and for the presence of a depot as starting and ending points of the route. Indeed, in problems of this type,

a single vehicle having a finite capacity transports two heterogeneous kinds of good. One unit of good of the first type is required at each delivery location (by a so-called *linehaul customer*). Similarly, one unit of the second type of good has to be picked up at each pickup location (from a so-called *backhaul customer*). The goods to be delivered have to be loaded by the vehicle at a depot. The goods to be picked up have to be transported by the vehicle to the depot. The vehicle starts its route from a depot with the entire load sufficient to fulfill all the linehaul requests. Without exceeding its capacity, it has to visit all the linehaul and the backhaul customers before coming back to the depot.

According to [153], problems with backhauls and linehauls can be divided into two categories: *TSP with mixed linehauls and backhauls (TSPMB)*,[2] and *TSP with clustered backhauls (TSPCB)*. In the first case, the pickup and the delivery orders can be satisfied in an arbitrary order. In the second case, the linehauls must be served before the backhauls. Note that this special feature of the TSPCB is very close to the requirement of the double TSP with multiple stacks of a pickup phase entirely preceding the delivery phase.

TSP with Mixed linehauls and Backhauls. The study of the TSPMB was initiated by Mosheiov [140] to solve the problem of transportation of under-privileged children to some host families during the vacation period. The transportation is made by a vehicle performing a single route. The children starting their vacation period represent the "good" satisfying the linehaul customers' request. The children ending their vacation period instead satisfy the backhaul customers' demand. In Mosheiov's formulation of the problem, both the total amounts of requested pickup quantities and delivery quantities equal the capacity of the vehicle. A route of the vehicle corresponds to a Hamiltonian circuit in the complete digraph whose vertices represent all the pickup and delivery locations and the depot. It is feasible if the vehicle can perform it without exceeding its capacity.

In [140] the author first gives an integer programming formulation for the TSPMB. Then he observes that an α-approximation algorithm for the TSP yields a $1+\alpha$-approximation algorithm for the TSPMB. Using the Christofides' 1.5-approximation algorithm [36] for the TSP he gets a 2.5-approximation algorithm whose running time is $O(n^3)$ where n is the total number of customers. The algorithm is finally tested against another heuristic algorithm for the same problem based on cheapest insertion, introduced in the same paper.

In a subsequent work, Anily and Mosheiov [5] propose a 2–approximation algorithm for the TSPMB. They consider a minimum spanning tree of the graph whose vertices represent all the locations to be visited (including the depot). Their heuristic is based on the fact that two copies of such a minimum spanning tree can be combined to get a feasible solution to the TSPMB. The running time to perform this task is $O(n^2)$ where n is the total number of locations to be visited. Their heuristic is computationally compared to a Mosheiov's heuristic presented in [140]. They observe that the latter is slower than the former but yields a better approximation.

[2]We will use this name from now on. Note, however, that other names have been used in the literature: TSP with Delivery and Backhauls (TSPDB) in [5], TSP with Deliveries and Collections (TSPDC) in [15] and Traveling Salesman Problem with pickup and Delivery (TSPD) in [140].

Another heuristic approach to the TSPMB is given by Gendreau *et al.* [74]. By using a farthest insertion scheme they initially get in $O(n^2)$ time a Hamiltonian circuit of the original graph in which the TSPMB instance is defined. Subsequently they solve the TSPMB using as underlying graph the Hamiltonian circuit obtained in the first step. A solution H to this special TSPMB is finally converted in a solution for the original TSPMB by following the order in which the vertices are visited in H. The whole procedure yields a 3–approximation algorithm for the TSPMB. In the same paper this algorithm is also post-processed by means of a constant-time local search method. Finally in [74], also two tabu search methods for the TSPMB are presented.

An exact algorithm to tackle TSPMB has been presented by Baldacci *et al.* [15]. They consider the case in which the underlying weighted digraph is symmetric. Their approach is based on the resolution of an integer linear programming formulation for the TSPMB via a branch-and-cut algorithm. The formulation they propose is a two-commodity flow formulation in which with each couple of vertices $\{i, j\}$ of the underlying graph it is associated a couple of flow variables f_{ij} and f_{ji} with the following meaning: if the vehicle takes arc (i, j) in its route, then f_{ij} is its total load at the moment of traversing (i, j), while f_{ji} is its available space. Arc variables x_{ij} are then combined with the flow variables to obtain a valid integer linear programming formulation for the TSPMB. The authors also introduce several families of strengthening inequalities for their formulation. A first set of strengthening inequalities only involves the flow variables. The other two families of inequalities only involve the arc variables and are based on a reinforcement of classical subtour elimination constraints for the TSP and on the capacity limit of the vehicle (*capacity constraints*). Efficient separation algorithms of these families are then discussed and implemented to develop a branch-and-cut algorithm. This is finally tested on 3 classes of instances solving some of them with up to 200 customers.

TSP with Clustered Backhauls. In the TSPCB all linehaul customers must be served before the backhaul customers, a strong similarity with the double TSP with multiple stacks in which the pickup phase must be completed before the delivery phase starts. In [72] the authors propose heuristic methods to solve the TSPCB. All heuristics are based on the heuristic GENIUS [71] for the ATSP. This starts from a circuit on three vertices and iteratively inserts a new vertex v between two vertices already in the circuit that are in a neighborhood of v (according to some neighborhood structure). This insertion phase is called GENI. Once all vertices have been inserted, a post-optimization phase, called US, takes place. This consists in removing each vertex from the circuit and reinserting it according to the GENI rule. The best heuristic for the TSPCB presented in [72] consists in applying the GENIUS heuristic to a modified cost matrix that forces a low cost solution to visit all the linehauls before serving the backhauls. The remaining heuristics are minor modifications to the heuristic above, obtained *e.g.,* by applying GENIUS to different subsets of vertices in the underlying graph, or by performing cheapest insertions followed by post-optimization phases. In terms of solution quality the first heuristic turns out to be the best one, although it is more time consuming.

A 1.5-approximation algorithm has been designed in [73]. The idea of such an algorithm goes along the same lines of the Christofides' approximation algorithm for the TSP [36]: given the graph on which one aims at solving the TSPCB, the algorithm of [73] seeks a minimum spanning tree containing a spanning tree for the subgraph of the pickup vertices and a spanning tree for the subgraph of delivery vertices; then it exploits a minimum weight matching and shortcut operations as in the Christofides' algorithm to construct a feasible solution to the TSPCB.

Mladenović and Hansen introduce in [139] the VNS metaheuristic and test its effectiveness on the TSPCB, improving results obtained in [72] for the TSPCB. We finally mention another heuristic for the TSPCB, based on a neural network approach (see [76] for details).

Problems with Pickups and Deliveries

We now report two additional categories of routing problems involving pickup and delivery operations. The first one is the *pickup and delivery TSP (PDTSP)*,[3] concerned with the pickup and the delivery of only one kind of good. Although in the double TSP with multiple stacks several goods are to be transported, in the PDTSP we still have a vehicle of finite capacity to perform the pickup and delivery route.

The second category of problems differ from the PDTSP only in one respect: each pickup location holds a different item that must be delivered to a corresponding customer in the same network. A problem of this type is called *pickup and delivery problem (PDP)*. The PDP is similar to the double TSP with multiple stacks in that paired items and customers are considered. However, the PDP does not present the complication of stacks governed by the last-in-first-out rule, which is instead an inherent aspect of the double TSP with multiple stacks.

Pickup and Delivery TSP. In the PDTSP a vehicle having finite capacity visits, following a Hamiltonian circuit, the pickup and the delivery locations of a network. A depot is the starting and ending point of the circuit. At each pickup location i, the vehicle collects exactly q_i units of the good. At each delivery location j, the vehicle delivers the demanded d_j units of the good. The transportation of the good from pickup locations to delivery locations can be performed in any order. An optimal solution to the PDTSP must fulfill all the delivery requests without violating the capacity constraints and minimize the total traveled distance.

The PDTSP first appeared in [102, 103]. In [102], the complexity of the PDTSP is discussed. It is shown that not only the PDTSP generalizes the TSP, and thus is NP-hard, but also deciding if it has a feasible solution is an NP-complete problem as it generalizes the 3–partition problem [68]. In addition, it is given a method to transform an instance of the

[3] Even in this case several other names appeared in the literature indicate (variants of) the same problem: the PDTSP is also called capacitated traveling salesman problem with pickups and deliveries in [4], One-Commodity Pickup and Delivery Traveling Salesman Problem (1-PDTSP) in [102] and traveling salesman problem with pickup and delivery in [103].

TSPMB into an instance of the PDTSP. In [103], the asymmetric-cost PDTSP is modeled by means of a mixed-integer linear programming formulation involving arc variables and variables recording the load of the vehicle when traversing an arc. In addition, using the Benders' decomposition [19] on a similar formulation for the symmetric-cost PDTSP, the authors get a formulation that only involves binary arc variables. The formulation is essentially characterized by the so-called *Benders' cuts*. An exact separation algorithm for the Benders' cuts is then presented. This algorithm runs in time $O(n^3)$ where n is the number of locations. Two families of strengthening cuts are also presented, the *rounded Benders' cuts* and the *clique inequalities*. For both families, heuristic separation algorithms are described. The integer formulation and the strengthening cuts are finally tested in a branch-and-cut algorithm. Such an algorithm is used to solve instances for the PDTSP but also for the TSPMB. Instances of the PDTSP with up to 50 items are solved to optimality; instances of the TSPMB with up to 75 items are solved to optimality.

The exact method for the PDTSP described above has been subsequently improved in [100]. The core of the paper is the adaptation of valid inequalities arising from the *Capacitated Vehicle Routing Problem* (CVRP) [182] to the PDTSP, and their inclusion in a branch-and-cut algorithm. Instances of the PDTSP with up to 100 locations and instances of the TSPMB with up to 261 locations are solved to optimality.

Exact polynomial resolution methods can be given for the PDTSP when the underlying graph has a special structure (namely, when it is a path or a tree), according to the capacity of the vehicle [187].

The best available heuristic for the PDTSP so far is the one by Mladenović *et al.* [138]. Combining the GVNS method with efficient data structures, they report solutions for instances of the PDTSP with up to 1000 customers, outperforming other heuristics for the PDTSP previously appeared (an "incomplete optimization" heuristic *i.e.,* a branch-and-cut applied to a restricted set of variables [99], the heuristic of [101] combining the GRASP approach [59] and a VNS algorithm, a genetic algorithm [189] and a VNS algorithm [104]).

All the approaches described above are devoted to the resolution of the PDTSP in which the pickup and delivery requests cannot be split. If we assume that the requests can be split, then each location giving or demanding a quantity w of the good can be split in w locations giving or demanding 1 unit of the good [187]. In this special case (*i.e.,* in which each pickup location has 1 unit of the good and each delivery location requires 1 unit of the good) a 9.5-approximation algorithm has been introduced [34]. The approximation factor of this algorithm is independent of the capacity of the vehicle. The algorithm is obtained by adapting the Christofides' approximation algorithm for the TSP [36]. A better approximation factor of 2 can be obtained for the case in which the capacity is infinite or 1 [34] (improving a 2.5-approximation algorithm for the case of capacity 1 given in [4]).

We mention also that, more recently, some research works have addressed variants of the PDTSP that can be applied to the bike sharing systems, see *e.g.,* [55, 97].

Pickup and Delivery Problems. In the PDP the pickup and the delivery operations are made using one vehicle, which can have finite or infinite capacity. The vehicle starts

from a depot, visits all the locations exactly once and comes back to the depot. As in the classical TSP, the resulting route is an Hamiltonian circuit. In this case the underlying complete graph has $2p + 1$ vertices, where p is the number of pickup locations (indeed the number of delivery locations equals the number of pickup locations, since they are paired; the depot is not a pickup nor a delivery location). Such a Hamiltonian circuit must however guarantee that each delivery location is visited after the corresponding pickup location and that the capacity of the vehicle is never exceeded.

According to Berbeglia *et al.*'s classification given in [20], the PDP are single vehicle one-to-one P/D problems.

The seminal work in the category of PDP is the one by Psaraftis [160]. In fact, in [160] the problem was also called *dial-a-ride problem* since it dealt with the transportation of customers in a network. For this reason, the objective is to minimize a linear combination of the total amount of time spent to perform the whole Hamiltonian circuit and the "dissatisfaction" of the customers (computed as a weighted sum of the time the customer waits before being picked up, and the riding time to reach the corresponding delivery location). The proposed solution approach consists in a simple dynamic programming approach running in $O(p^2 3^p)$ time, where p is the number of pickup locations. The state space is given by the $(p + 1)$-uples in which the first entry is the vertex currently visited by the vehicle and each other entry corresponds to a customer; the value of the latter entries can be one of the three available statuses of the customer (*i.e.,* waiting, loaded, delivered).

Early approximation results on the PDP in its dial-a-ride form are due to Stein [180] (providing a method that, with probability 1, yields a 1.06-approximation algorithm for the PDP with infinite capacity to serve origins and destinations in an area of a given region) and to Psaraftis [161] (proposing a 3-approximation algorithm for the PDP without capacity in the plane in which the triangle inequality holds) .

A branch-and-bound procedure has been also proposed to tackle the PDP with finite and infinite capacity by Kalantari *et al.* [114]. In this procedure, the lower bound is calculated by using the reduced matrix introduced by Little *et al.* [129] for the TSP. The branching is made over the arcs that the vehicle would cross. Finally, the cost matrix is modified so that the arcs taken in the subsequent nodes of the exploration tree of the algorithm do not violate any precedence relationship between pickup and delivery customers.

In [63] another branch-and-bound algorithm for the PDP is presented. It uses an additive bounding procedure to get lower bounds. This type of procedure has been briefly described in the presentation of [64] in Section 2.3.1. In [63] the bounds are obtained from shortest spanning trees, assignment problems, decompositions and disjunctions.

The paper [169] proposes a branch-and-cut algorithm for the PDP. The authors first introduce a linear integer programming formulation that only involves arc variables. Then they show the validity of four families of linear inequalities conceived to tighten the starting formulation. Finally, they run a branch-and-cut algorithm that uses the strengthening valid inequalities as cutting planes. They describe simple exact and heuristic routines to separate the proposed inequalities and they also introduce a heuristic to get a good quality upper

bounds.

A subsequent paper by Dumitrescu *et al.* [53] enhanced the branch-and-cut approach described above. In [53] the integer hull of the integer formulation proposed in [169] is thoroughly studied. The authors first determine the dimension of this integer hull. They introduce several families of valid inequalities for their formulation. In addition, they are able in some cases to provide conditions under which known and new valid inequalities are facet-defining for the integer hull of their formulation. Exact and heuristic procedures to separate the valid inequalities are also described. Finally, a branch-and-cut is run on benchmark instances. It solves problem instances with up to 35 pickup locations.

2.3.3 Pickup and Delivery Problems with Last-in-First-Out Constraints

The combination of stacks obeying a last-in-first-out rule with the regular PDP gives rises to the *pickup and delivery TSP with LIFO loading (TSPPDL)* and to *pickup and delivery TSP with multiple stacks (PDTSPMS)*. The first one only involves one stack of infinite capacity, the second one involves multiple stacks.

A very recent generalization of the double TSP with multiple stacks is the *traveling purchaser problem with multiple stacks and deliveries (TPPMSD)* [16]. In this problem, the routing and loading constraints are identical to the double TSP with multiple stacks. The only difference is that a pickup location, here called *market*, can hold more than one item and the same item can be picked up at several markets. Each item, however, is picked up exactly once.

We point out that in all the above mentioned families of problems each item is assigned exactly to one stack and the items occupy the same amount of space in the stack. That is, they are identical from a packing point of view, exactly as in the double TSP with multiple stacks. For problems admitting more complicated stack configurations we refer the reader to *e.g.,* [108].

Pickup and Delivery TSP with LIFO Loading. The TSPPDL is a PDP in which the vehicle has exactly one stack obeying a last-in-first-out principle. Both the finite and infinite capacity versions of the problem are considered in the literature. This type of problem has been first described by Ladany and Mehrez [120]. Heuristics for the TSPPDL are reported by Pacheco [146, 147] (the first reference is in Spanish; the second reference presents a simulated annealing for the problem), by Cassani [33] who gives greedy heuristics along with a VNS and by Carrabs *et al.* [31] who propose a VNS.

Pacheco [148, 145] and Cassani [33] also propose exact methods for the TSPPDL. The first author uses an adaptation of the branch-and-bound algorithm of Kalantari *et al.* for the PDP [114] mentioned above. The second author resorts to bounds obtained from the assignment problem. An additive branch-and-bound for the TSPPDL is introduced by Carrabs *et al.* [29]. In their algorithm, the bounds are again obtained through TSP relaxations but they are improved with respect to their standard form by using *filters*.

These are criteria that, given a solution partially constructed, predict arcs that cannot be taken in the residual part of the solution. The same idea will be applied to the double TSP with multiple stacks and we refer the reader to the next section for more details on additive branch-and-bound combined with filters. The best of the previous three algorithms solves instances with up to 21 items [29].

A branch-and-cut algorithm for the TSPPDL has been proposed by Cordeau *et al.* [41]. The branch-and-cut algorithm is used to solve to optimality instances of the TSPPDL with up to 25 items. The algorithm is run on the best of three models proposed in the same paper [41]. The first model is a modification of the model used by Ruland and Rodin for the PDP [169]. It involves arc variables along with demand variables, and families of constraints enforcing the last-in-first-out rule fulfillment. These families are obtained after linearizing nonlinear constraints expressing the last-in-first-out rule. The second model employs flow variable, keeping track of the load of the vehicle on each arc. The third model (the best one in the preliminary computational results) only involves arc variables and uses an exponential-size family of constraints to model the last-in-first-out rule. Many strengthening cuts for the three models are presented and both exact and heuristic procedure of separation are discussed.

Pickup and Delivery TSP with Multiple Stacks. The PDTSPMS is a generalization of both the double TSP with multiple stacks and the TSPPDL seen above. Indeed, the only difference with the TSPPDL is that the vehicle has more than one stack. All stacks in this case obey the last-in-first-out principle. With respect to the double TSP with multiple stacks in the PDTSPMS the pickup phase has not to be completed necessarily before the delivery phase starts. However, by choosing suitable costs on the arcs of the graph representing the network for the PDTSPMS instance, one can transform such an instance into an instance of the double TSP with multiple stacks. The problem has been addressed by Côté *et al.* [44], Côté *et al.* [45] and Sampaio and Urrutia [170]. The first two references also provide results for the double TSP with multiple stacks. Hence we discuss them in next section, that is dedicated to this problem. Sampaio and Urrutia implement a branch-and-cut algorithm based on a integer linear programming formulation with two sets of variables. The first set contains arc variables. The second set includes variables with three indices and keep track of which stack (first index) has been modified by loading or delivering a given item (second index) immediately after having visited a given location (third index). Some strengthening cuts are also provided to be added dynamically during the execution of the algorithm. A comparison of the computational results with those obtained by the branch-and-cut of Côté *et al.* [44] shows that the algorithm is competitive in average (although slower), solves some new instances and reduces the optimality gap on unsolved instances.

Traveling Purchaser Problem with Multiple Stacks and Deliveries. In the TPPMSD, each customer demands exactly one item present in a pickup region. All the items must be picked up before the delivery phase starts, as in the double TSP with multi-

ple stacks. The pickup (resp. delivery) phase consists in performing a Hamiltonian circuit in the pickup (resp. delivery) region. If customers and items are paired in the TPPMSD, this is no longer the case for the pickup and delivery locations. Indeed, each item can be picked up at several locations (the so-called markets) and each location could provide several items. Each time an item is collected, a cost depending on the market is payed. In addition, each item is collected exactly once and loaded in vehicle with multiple stacks of finite capacity obeying a last-in-first-out rule as in the double TSP with multiple stacks. The goal is to find a pair of pickup and delivery circuits that fulfills all constraints, and yields the minimum total cost, calculated as the sum of the total traveled distance and the total cost payed to collect the items. Hence the TPPMSD is a generalization of the double TSP with multiple stacks. The TPPMSD has been introduced in [16]. The authors propose a formulation for the problem and a branch-and-cut algorithm for its resolution. Since both are adapted to the double TSP with multiple stacks we describe them in Section 2.4.1 and in Section 2.4.4 dedicated to this problem.

2.4 State of the Art on the Double TSP with Multiple Stacks

In this section we review the literature on the double TSP with multiple stacks. As mentioned in Section 2.1, this problem has been introduced by Petersen and Madsen [158] in 2009. It has received since a growing interest, resulting in a number of resolution approaches as well as some theoretical results.

2.4.1 Integer Linear Programming Formulations

The first integer linear programming formulation for the double TSP with multiple stacks has been presented in [158]. The cities are modeled by using two complete digraphs $G^P = (V^P, E^P)$ and $G^D = (V^D, E^D)$. Let d^P and d^D be the vertices representing the depot respectively in the first and the second city. Let also R be the set of stacks. Petersen *et al.* model the problem in which each stack has capacity q and is fully loaded in a solution.

Their formulation involves three sets of binary variables x, y, z subject to constraints:

$$x^T(\delta^+(i)) = 1 \qquad \forall i \in V^T, T = P, D \tag{2.8}$$

$$x^T(\delta^-(i)) = 1 \qquad \forall i \in V^T, T = P, D \tag{2.9}$$

$$y^T_{ij} + y^T_{ji} = 1 \qquad \forall \text{ distinct } i, j \in V^T \setminus \{d^T\}, T = P, D \tag{2.10}$$

$$y^T_{ik} + y^T_{kj} \leq 1 + y^T_{ij} \qquad \forall \text{ distinct } i, j, k \in V^T \setminus \{d^T\}, T = P, D \tag{2.11}$$

$$x^T_{ij} \leq y^T_{ij} \qquad \forall \text{ distinct } i, j \in V^T \setminus \{d^T\}, T = P, D \tag{2.12}$$

$$y^P_{ij} + z_{ir} + z_{jr} \leq 3 - y^D_{ij} \qquad \forall \text{ distinct } i, j \in V^T, r \in R \tag{2.13}$$

$$\sum_{r \in R} z_{ir} = 1 \qquad \forall i \in V^P \setminus \{d^P\} \tag{2.14}$$

$$\sum_{i \in V^P \setminus \{d^P\}} z_{ir} = q \qquad \forall r \in R \tag{2.15}$$

Variables x_{ij} are arc variables as in formulation (2.4)–(2.7). Variables y_{ij} are called *precedence variables* since they capture the precedence relations between pairs of vertices in the same route. Variables z_{ir} record the stack r in which item i is packed. Petersen and Madsen tried to solve exactly the problem using their formulation but they only succeeded for very small numbers of items (10 items to be packed in two stacks or 12 items to be packed in three stacks) within a time limit of one hour with a 1.2GHz processor.

A variation of the model above has appeared in [157]. It is called *row precedence model*. The difference with the formulation (2.8)–(2.15) is that the precedence variables are replaced by binary variables w^r_{ij} which are 1 if and only if item i is picked up before item j and both are in the same stack r. Constraints involving precedence variables are replaced by exponential size families of constraints that guarantee the correctness of the resulting formulation. The latter includes the subtour elimination constraints (2.6) and constraints indexed by all paths linking two pickup locations or two delivery locations.

Other two formulations are introduced in [157]. The first one is referred to as *flow model*. This model considers one copy of each pickup location for each different stack. The pickup and delivery routes correspond then to two circuits in this extended digraph. The circuits visit exactly one copy of each location. A stack corresponds to a directed path that visits the copies of vertices associated with the items in that stack. The last model is called *infeasible path model*. It consists in duplicating the formulation (2.4)–(2.7) for the TSP by using arc variables x^P and x^D. They describe pairs of Hamiltonian circuits. The feasibility of the pair for the problem is then ensured by the *infeasible path constraints*:

$$x(P) \leq |P| - 1 \qquad \forall P \text{ infeasible path.} \tag{2.16}$$

Here an *infeasible path* is a directed path given by the last portion of a pickup circuit and the first portion of a delivery circuit. The path is infeasible if no loading plan exists for the whole sequence of vertices appearing in the path. In the above constraint, the expression $x(P)$ is the sum of variables x^P_{ij} and x^D_{ij} associated with the arcs in P.

The infeasible path model has been considered also in [2]. The only difference is in the definition of constraints (2.16). Indeed in this case they are replaced by

$$x^P(P_1) + x^D(P_2) \leq |P_1| + |P_2| - 1 \qquad \forall (P_1, P_2) \text{ infeasible pair of paths.} \tag{2.17}$$

In constraints (2.17), the pair (P_1, P_2) is *infeasible* if there is no way to accommodate the items picked up by following the directed path P_1 and to unload them while performing P_2 without violating the last-in-first-out rule or the capacity limit of some stack.

Formulations from Generalizations. Some formulations can be obtained also by considering generalizations of the double TSP with multiple stacks. Côté *et al.* [44, 45] introduce several formulations to tackle the PDTSPMS (recall that this problem generalizes the double TSP with multiple stacks). In [45] the authors initially provide an integer linear programming formulation for the PDTSPMS that includes arc variables, variables to keep track of the stack assignment, variables to keep track of the position of each vertex in the route, and variables to keep track of the load of each stack upon leaving a vertex in the circuit. In [44] this formulation is simplified by removing the variables keeping track of the positions of the vertices in a circuit.

Other two formulations are proposed in [44]. The first one is an adaptation of a model for the TSPPDL presented in Cordeau *et al.* [41] and based on flow variables. The second formulation is again an infeasible path model similar to the one of Petersen *et al.* [157] and Alba Martínez *et al.* [2]. In [44], the infeasible path constraints are of the form (2.16), with P being an infeasible path whenever it visits pickup and delivery locations in such a way that the last-in-first-out rule or the capacity constraints are necessarily violated.

Another integer linear programming formulation for the double TSP with multiple stacks arises from the TPPMSD, see Section 2.3. It is due to Batista-Galván [16]. The formulation models the pickup and the delivery circuits with two distinct sets of variables. The pickup circuit is modeled by arc variables as a TSP in a graph in which each vertex coincides with a pair market-item. The delivery circuit is instead treated as a regular TSP in the graph representing the delivery city, and is thus described by variables associated with arcs in this graph. Additional variables record the market in which an item is picked up. With respect to a classical duplication of the TSP formulation in terms of arc variables, the model of [16] includes constraints to ensure that each item is picked up exactly once and all markets are visited exactly once, and infeasible path constraints for the loading rules (last-in-first-out policy and capacity of the stacks).

2.4.2 Theoretical Results

The double TSP with multiple stacks has stimulated a number of theoretical investigations. The NP-hardness of the problem is an easy consequence of the observation that in the single stack case the double TSP with multiple stacks is a TSP. In addition, several authors have focused on the complexity of subproblems of the double TSP with multiple stacks or on some relaxations.

In [32] Casazza *et al.* study theoretical properties of the infinite capacity case. They are first interested in determining a loading plan consistent with a fixed pair of pickup and delivery circuits that uses the minimum number of stacks. They reduce the problem to a coloring problem on a permutation graph having as many vertices as the number of items in the instance of the problem (see Section 1.3.1 and Section 3.3.2 for more details respectively on permutation graphs and on the reduction to a graph coloring problem). From this they get that a solution to the problem of determining such a loading plan can be found in $O(n \log n)$ time, where n is the number of items in the instance. Their result also yields a characterization of s-consistent pairs of Hamiltonian circuits in terms of sequences of vertices, see also Section 3.3.2. The second subproblem the authors consider is to find the pair of Hamiltonian circuits which is consistent with a given loading plan and that has minimum total cost. They use a dynamic programming algorithm to solve this problem. Its running time is in $O(n^s)$ where n is the number of items and s is the number of stacks in the instance. The last subproblem the authors consider is the construction of a so-called *partial loading plan*. This is simply a loading plan in which the items of a subset of customers do not appear. The subproblem is to find a partial loading plan of the items in at most s stacks that is consistent with a given pair of pickup and delivery circuits and that includes the maximum number of items. Casazza *et al.* reduce this task to the computation of a minimum cost flow in an auxiliary weighted graph.

In Toulouse and Wolfler Calvo [183] it is observed that good quality solutions to the TSP may not lead to good solutions to the double TSP with multiple stacks of infinite capacity. Their construction can be resumed as follows. They construct two heuristics for the double TSP with two stacks of infinite capacity, based on the possibility of solving to optimality a single TSP. Then they show that these heuristics can have an unbounded error with respect to the optimal solutions to the double TSP with multiple stacks of infinite capacity, on specific instances.

An important complexity result has appeared in Bonomo *et al.* [23]. The authors focus on the problem of deciding whether a pair of Hamiltonian circuits admits an s, q-loading plan for fixed values s of the stacks and q of their capacity. Bonomo *et al.* show that this subproblem of the double TSP with multiple stacks can be solved in $O(n^{s^2+s+1}s^3)$ time. Their idea is to reduce the problem to a *bounded coloring* problem of a permutation graph. This reduction is explained in greater detail in Section 3.2.2, hence we skip it here. We however precise that a bounded coloring of a graph is a proper coloring of the vertices in which each color class contains at most a prescribed amount of vertices. To prove their result, Bonomo *et al.* resort to the notion of *k-thin graph*, *i.e.,* a graph whose vertices can be partitioned in k classes and ordered with a linear ordering $\prec$ such that if $v_r \prec v_s \prec v_t$ and v_r and v_s are in the same class, then the adjacency of v_t and v_r in the graph implies the adjacency of v_t and v_s. The *thinness* of a graph is the least k such that the graph is k-thin and Bonomo *et al.* show that the thinness of a co-comparability graph is bounded above by its chromatic number. Bonomo *et al.* prove that the bounded s-coloring can be solved in $O(n^{ks+s+1}s^2k)$-time in a k-thin graph, using an approach very similar to a classical dynamic programming. Then the result follows since a permutation graph is a

co-comparability graph and the chromatic number of the latter can be found in $O(n^3)$.

The last results we survey here are those of Borne *et al.* [26]. Their paper proposes a polyhedral study of the infeasible path model based on constraints (2.17) in the infinite capacity case. Borne *et al.* prove that the dimension of the integer hull of their formulation is twice the integer hull of the ATSP polytope. They also show that each facet of the ATSP polytope induces two facets of the integer hull of their formulation. The authors then focus on the double TSP with two stacks of infinite capacity. They give a polynomial time algorithm to separate their infeasible path constraints over fractional points. From this result, they infer that the optimization problem over the linear relaxation of their formulation for the two stack case is polynomially solvable. The last part of their paper introduces some new families of strengthening inequalities for the case of the double TSP with two stacks of infinite capacity.

2.4.3 Heuristic Approaches

Several heuristic methods have been used to tackle the double TSP with multiple stacks. To the best of our knowledge the best of these methods are the LNS approaches and the VNS approaches.

LNS Approaches. The first LNS approach to the double TSP with multiple stacks is due to Petersen and Madsen [158]. To generate a starting solution for their LNS the authors solve heuristically the double TSP with one stack: as seen in Section 2.2, this special case amounts to solve a TSP and in a solution the delivery circuits is the opposite of the pickup circuit. The resulting pair of circuits is then s, q-consistent for every s and q.

For the removal strategies in their LNS method, Petersen and Madsen use the concept of *relatedness* described in [177] and the idea of removing the most expensive orders to cover. Reinsertion is based on nearest, farthest, cheapest and most expensive operators, see Chapter 4 of [113]. The LNS of [158] is then computationally compared to other heuristic approaches presented in the same paper (namely, a tabu search, a simulated annealing and an iterated local search). To this purpose Petersen and Madsen create benchmark instances, widely used also in subsequent works on the double TSP with multiple stacks. These instances are described in Section 5.4.2 of this thesis. The quality of the heuristics is determined by comparing their results on short runs (10 seconds or 3 minutes of execution) with the best known upper bounds for instances with 12, 33 and 66 items. The best upper bounds are obtained by running the LNS for about two hours or, for 12 item instances, by solving exactly an integer linear programming formulation proposed in the same paper. The LNS approach outperforms all other heuristic methods presented in [158]. In particular, it solves to optimality all the instances with 12 items and it is at most 6% above the best known solutions to the instances with 33 items. For the larger instances with 66 items all heuristics fail to find solutions of good quality.

A second LNS approach is proposed by Côté *et al.* in [45]. Their algorithm is designed to tackle the PDTSPMS which generalizes the double TSP with multiple stacks, see Sec-

tion 2.3.2. They introduce four operators for the removal of *requests*, *i.e.,* paired pickup and delivery locations, from a solution represented by a Hamiltonian circuit. The first removal operator consists in randomly choosing the pickup and delivery requests to be removed. The second and third operators remove a random request and then the requests that are close (according to two distinct measures) to the previously removed request. The last removal operator eliminates requests that correspond to consecutive vertices in the route, by starting at a random pickup vertex. Two insertion operators are proposed. The first one reinserts a vertex in a partial solution so as to minimize the cost of the resulting circuit. The second insertion operator minimizes instead a so-called *generalized regret*, see p. 23 in [45] for details. The removal and insertion operators are then used in a LNS which also contains a probabilistic acceptance condition inherited from the simulated annealing. The solution obtained by the LNS is finally postprocessed by means of a dynamic programming algorithm: this constructs the optimal solution consistent with the evolution of the stacks of the LNS solution. We resume the results obtained by the heuristic of Côté *et al.* [45] on the double TSP with three stacks (only this case is considered in [45]). It is tested over the four sets of instances with 12, 33, 66 and 132 requests introduced in [158, 57]. For each instance, the results are of two types that we will call A and B. For the first type, an average value of the solutions returned by three runs of the LNS on a given instance is calculated, and the result of type A is then the gap of this value over the best known value for the same instance. In results of type B, the best value obtained by the LNS in the three runs is used to compute the gap with the best known value. In all instances with up to 66 items results of type A obtained by Côté *et al.* are better or similar to those obtained by the previously known heuristics. On the same instances results of B type improve substantially the results obtained by the previous heuristics. This improvement is also more apparent for the instances with 132 items, where, out of the total of 20 instance, the algorithm of Côté *et al.* finds 12 new best solutions.

VNS Approaches. The paper [57] by Felipe *et al.* presents a VND, a GVNS and a "hybridized" VNS for the double TSP with multiple stacks. To generate an initial solution, the authors solve the single stack case of the problem, as done in [158]. They also randomly generate a pickup circuit and subsequently construct a consistent delivery circuit, according to several schemes. For instance, they can simply reverse the pickup circuit. More complex schemes involve the construction of a loading plan and a subsequent unloading of the items yielding the delivery circuit. The construction of a loading plan can be done *e.g.,* by following the order of the pickup circuit and randomly assigning items to the stacks, see [57] for additional criteria. The unloading phase either considers consecutive items in the same stacks, or observe some distance-based criteria, such as nearest delivery location, farthest one, etc. Besides some neighborhood previously introduced in [158] for their local search algorithms, the authors also propose four new neighborhoods. They essentially consist in swapping two or more items in the same stack, or in changing the positions of one or more requests in the circuits. The neighborhoods are used in all the proposed algorithms, that is a VND, a GVNS and a *hybridized* GVNS. The latter is an application

of the GVNS combined with some tabu lists to a large set of random initial solutions. This application returns a set of improved solutions. The GVNS is subsequently applied to the best solution in this set. In their test session, Felipe *et al.* consider instances with 33, 66 and 132 items. The sets of instances with 33 and 66 items is the same as used in Petersen and Madsen [158]. The set with 132 items is new but generated following the same method. Their results show that the hybridized method outperforms the other two methods and improves the results of [158].

An enhancement of the previous heuristic method is given in [58] where the hybridized VNS of [57] is combined with new neighborhood structures producing neighbors possibly infeasible for the double TSP with multiple stacks. However, the algorithm is organized in such a way that, at its termination, the feasibility of the output is guaranteed. Infeasible neighbors are generated by allowing an extra capacity on the stacks (*Capacity Infeasibility (CI)*) or by allowing violations to the last-in-first-out rule (*Precedence Infeasibility (PI)*). To reconstruct a feasible solution from a neighbor presenting the CI, two similar procedures are used, the *Multi-Step Reduction with Backtracking (MSRB)* and the *Global Reduction with Adjustable Size (GRAS)*. Both are based on the notion of target solution: given a loading plan with overloaded stacks the procedures attempt to generate a new loading plan (the target) in which the stacks are less overloaded. Both procedures try to perform this change without modifying the total cost of the pair of circuits given in the generated neighbor. While the MSRB reduces of one unit the overload (hence it must be applied iteratively to reach the target), the GRAS provides a simultaneous reduction of the overload of the stacks. It may be possible that the procedures fail to produce the target without changing the neighbor cost. In this case, so-called *projection operators* modify both the loading plan and the pair of pickup and delivery circuits, ensuring feasibility but increasing the total cost. To fix the PI, Felipe *et al.* develop operators similar to the ones discussed about the CI, but able to remove and reinsert the items causing the PI. The authors also design two infeasibility measures, one to evaluate the degree of CI in a solution, the other to measure the PI. The choice of the next neighbor is guided by a weighted sum of the cost of the neighbor and its CI and PI measures. The resulting algorithm is called *Exterior Search Algorithm (EXT)*. The EXT is combined with the hybridized VNS approach of Felipe *et al.* [57], in a new algorithm called EXT-HVNS. Here, the EXT is used to escape from local optima, and the hybridized VNS to get local optima of good quality. The computational results show an improvement in terms of gap with the best known solutions when compared to the previous heuristic algorithm of [57]. In particular, the improvement becomes more evident when the size of instances increases to 132 items.

Other Heuristics. Here we discuss heuristic methods for the double TSP with multiple stacks that do not fall into the LNS and the VNS categories.

The first one is given by Casazza *et al.* [32]. The authors design an *Alternating Routing-Loading (ARL) heuristic* for the double TSP with multiple stacks of infinite capacity. The ARL heuristic is based on the resolution of several subproblems of the double TSP with multiple stacks of infinite capacity. In the same paper, the resolution of the subproblems is

shown to be tractable, see also Section 2.4.2 of this thesis. The ARL works as follows. First, it finds a pair of pickup and delivery circuits. Then it solves the problem of determining a *partial loading plan* for this pair; the latter is nothing but a loading plan consistent with the circuits but containing a restricted set of items. Next, the ARL creates an optimal pair of circuits consistent with the given partial loading plan and initially containing only the items in the partial loading plan. The removed items/customers are reinserted in the circuits by following some locally optimal criteria. Finally also the stacks are filled with the missing items so that, along with the given pair of Hamiltonian circuits, it represents a solution to the double TSP with multiple stacks of infinite capacity. The procedure is repeated several times starting from the lastly found solution and until a stopping criterion is met. In the test session, the authors show that the ARL heuristic exhibits a good performance in terms of quality of the found solution. The ARL algorithm is also adapted to handle the finite capacity case. To this purpose, the items in stacks exceeding the capacity are moved to stacks with available space and in which no conflict due to the last-in-first-out policy can be generated. If this is not possible, then the items causing the violation of the capacity in some stacks are removed from the routes, and reinserted in other positions of the routes so that the capacity constraint is no longer violated. This *repair procedure* is shown to produce only a small increase of the value of a solution with respect to the best solution of the corresponding uncapacitated double TSP with multiple stacks instances and without increasing the computational time.

The most recent heuristic method for the double TSP with multiple stacks is due to Urrutia *et al.* [184]. Their heuristic is a combination of tabu and local searches in which, instead of considering the construction of the routes, one considers the construction of loading plans. Starting from any loading plan, optimal pickup and delivery circuits consistent with it are reconstructed using a refined version of a dynamic programming algorithm of Casazza *et al.* [32]. Only the three stack case is considered in [184]. The algorithm proposed in [184] needs several initial solutions. These are found by solving the single stack case of the problem by a randomized version of a TSP insertion heuristic and then adapting the obtained solution to the three stack case. A solution obtained in this way is improved as much as possible by a tabu search and by a local search before the next initial solution is generated. In addition, using a technique called *path relinking* [166], the authors also generate other solutions from a set of good feasible solutions found during the previous iterations of the tabu search. A solution for the tabu search is simply a disposition of the items in the stacks. The cost of a solution corresponds to the cost of the optimal pair of circuits consistent with it; nevertheless the authors also introduce some more efficient evaluation operators to speed up the execution. The neighborhood of a solution in the tabu search contains the loading plans obtainable by the solution itself by exchanging the position of three items in the stacks. A neighbor cannot be reached from the current solution (*i.e.,* it is tabu) if it has at least one exchanged items in the same position. The neighborhood of the local search is obtained by considering the exchange of two items in the stacks. The described approach does not improve the results obtained by heuristics of Felipe *et al.* [57] to which it is compared in the test session, although it

remains competitive.

2.4.4 Exact Methods

We describe exact methods to solve the double TSP with multiple stacks that have appeared in previous research works. At the end of the section we report a short discussion on the computational results obtained by the best of these algorithms. These have been tested on the benchmark instances introduced in [158] restricted to small numbers of items. Instance sets are identified by expressions of the type $s \times q$, where s is the value of available stacks in the vehicle and q the value of their capacity. The number n of items equals the product sq.

Ad Hoc Methods. The double TSP with multiple stacks presents some specific structures, hence two similar ad hoc methods have been designed for its resolution in [132, 131]. The algorithms are based on the generation of the *k-best Hamiltonian circuits* in the weighted digraphs representing the pickup and delivery cities. These are k Hamiltonian circuits ordered by increasing total weight, and such that all the Hamiltonian circuits not in the set have a higher total weight. In [132] the k-best Hamiltonian circuits are computed with an algorithm due to Lawler [124]. The algorithm presented in [132] consists of three steps. First, the set of the k-best Hamiltonian circuits in both the pickup and the delivery circuits are computed. Subsequently, all possible pairs of pickup and delivery circuits from this set are produced. The pairs are sorted according to their total cost. Finally, the algorithm checks, in the resulting order, whether a pair is feasible for the double TSP with multiple stacks. The latter check is made by solving an integer linear program whose structure depends on the pair. The algorithm is run multiple times for increasing values of k but some lower bounds (obtained considering also infeasible pairs of circuits) limit the number of repetitions.

The algorithm of [131] is sped up by including two preprocessing techniques before the second step, *i.e.,* before checking the feasibility of a pair. The preprocessing techniques are indeed faster than the resolution of the integer linear program involved in the second step. Preprocessing techniques of the first type deduce the minimal position the pickup circuit (resp. delivery circuit) in which an item is picked up (resp. delivered) from its delivery (resp. pickup) position and in function of the number of stacks. Preprocessing technique of second type find the *largest common sequence* of vertices in the pickup and delivery circuits (see also Section 3.3.2 for more details on this subproblem). For a pair of circuits to be feasible for the double TSP with multiple stacks, the sequence cannot have length greater than the number of stacks.

A Branch-and-Bound Algorithm. A branch-and-bound approach has been proposed by Carrabs *et al.* [30]. The branch-and-bound algorithm constructs a research tree level by level, in a depth-first fashion. Each level corresponds to a newly visited vertex. At level 0 the vertex corresponding to the depot is added. The first n levels correspond to

the construction of the pickup circuit hence also an assignment to an item to one stack is decided. Therefore, at level n a complete pickup circuit and a corresponding loading configuration are available. The construction of the delivery circuit at this point is made using Casazza *et al.*'s dynamic programming algorithm [32] see Section 2.4.2. Lower bounds are computed from TSP relaxations and combined in an additive fashion, see *e.g.,* [63] for additive bounding techniques. The algorithm uses *filters* to ensure that lower bounds calculated from the TSP relaxations are of good quality also for the double TSP with multiple stacks. A filter detects arcs that cannot be used to complete the partial solution of an intermediate level because of the last-in-first-out policy. The proposed algorithm possibly creates *equivalent paths*. The latter are feasible paths that are also associated to the same loading configuration. The authors show that the lower bounds calculated by using the TSP relaxation technique must be the same for all equivalent paths. Computing only once such a lower bound for a given path and then accessing its value quickly whenever an equivalent path is found, reduces the computational effort. The branch-and-bound is tested on instances already used by Petersen *et al.* [157] and only in the case of two stacks. An initial upper bound is calculated by using the heuristic of Côté *et al.* [45].

Branch-and-Cut Algorithms. The first exact approach for solving the double TSP with multiple stacks is the branch-and-cut algorithm of Petersen *et al.* [157]. In the paper four formulations are considered, see also Section 2.4.1. From a computational point of view the best of these formulations is the one based on infeasible path constraints (2.16), that we now describe. Both constraints (2.16) and the subtour elimination constraints are separated dynamically during the branch-and-cut algorithm. Constraints (2.16) are separated only when a solution obtained during the branch-and-cut execution corresponds to a pair of Hamiltonian circuits (hence they are separated after the subtour elimination constraints). In this case, a violation to (2.16) is reduced to test the feasibility of an integer linear program. The authors remark that this test is equivalent to a bounded coloring of a graph and that it is more convenient to separate constraints (2.16) associated with minimal infeasible paths. For their computational tests, the authors set a time limit of one hour and provide the tested algorithms with an initial solution obtained by running the simulated annealing algorithm of [158].

Côté *et al.* [44] also propose branch-and-cut algorithms using the formulations introduced in [44] for the PDTSPMS. The formulation based on infeasible path constraints (2.16) exhibits a better computational performance hence we limit the discussion to it. Constraints (2.16) are added dynamically during the separation step. The approach to detect violation to (2.16) is similar to the one given in [157]. Namely, it is reduced to the feasibility of an integer linear program and only when the solution represents a Hamiltonian circuit (remember that the PDTSPMS admits a single route as a solution). However, the infeasible path model of [44] is reinforced by means of strengthening cuts. Some of them are strengthening cuts inherited from the ATSP with and without precedence constraints between pairs of locations [13, 90]. New cuts are instead based on the last-in-first-out rule and on the capacity limit of the stacks. These cuts are also added dynamically during the

separation step of the branch-and-cut algorithm. Exact separation routines are given for the new cuts, whereas the ATSP cuts are separated following the procedures given in [41]. The algorithm is tested with a time limit of 3 hours of CPU time on the benchmark instances. Initially, the upper bound calculated by the large neighborhood search of Côté *et al.* [45] is provided to the algorithm.

A branch-and-cut approach for the double TSP with multiple stacks is proposed in [2] by Alba Martínez *et al.* The algorithm is based on the infeasible path model with constraints (2.17) incorporated in the classical TSP formulation with arc variables. Several strengthening cuts are introduced. A first family of cuts is obtained by adding to constraints (2.17) variables corresponding to arcs linking non-adjacent vertices in the paths of an infeasible pair. Only "forward arcs" are considered. Constraints of this form are called *tournament constraints*. When the pair of paths is infeasible in the infinite capacity relaxation, the tournament constraints are further strengthened by adding variables corresponding to specific "backward arcs". This gives rise to so-called *lifted tournament constraints*. Other infeasible pairs of paths derive from the observation that if an item is picked up after π items and delivered after δ items then it must hold $\pi + \delta - 2 \leq q$ where q is the capacity of the stacks. Pairs of paths violating this inequality give rise to infeasible pairs of paths. Finally, other infeasible pairs of paths are found by trying to extend a given pair of paths toward the depot. Given a pair of pickup and delivery paths, the separation routine for constraints (2.17) and the tournament constraints first looks for a violation of the last-in-first-out rule in the infinite capacity case. To do this, Alba Martínez *et al.* use the so-called CPM algorithm of Kelley [118] on an acyclic digraph. If the paths are feasible in the infinite capacity case, the routine enumerates all possible disposition of items in the stacks to test whether the pair is infeasible. Some intermediate heuristics try to speed up the whole procedure and are used to separate the remaining families of strengthening cuts. The constraints are separated on both fractional and integer solutions by considering pairs of paths associated with variables whose value can possibly lead to a violation. The algorithm is provided with an initial solution, found by applying the heuristic of Côté *et al.* [45].

A recent branch-and-cut algorithm for the double TSP with multiple stacks is the one presented by Batista-Galván *et al.* [16]. Actually, this algorithm is designed to handle the TPPMSD, a generalization of the double TSP with multiple stacks. The formulation used in the branch-and-cut algorithm is the one for the TPPMSD presented in Section 2.4.1. Some constraints of this formulation are strengthened and three types of the infeasible path constraints provided in [2] are adapted to take into account the specificity of the problem. As a consequence they use the same separation routines of that paper, although some refinements are added to improve the overall time spent to find constraint violations.

Another branch-and-cut method is due to Iori and Riera-Ledesma [109]. It is based on a formulation similar to the one presented in [2]. The formulation has however some specific aspects needed to take into account the characteristics of the *Double VRP with Multiple Stacks (DVRPMS)* for which it is originally designed. The DVRPMS is a generalization of the double TSP with multiple stacks introduced in [109]. Its difference with respect to the

latter lies in the possibility of using more than one vehicle to perform the pickup phase. Each used vehicle then performs the delivery phase, according to the rules of the double TSP with multiple stacks. During the branch-and-cut algorithm of [109] the strengthening inequalities presented in [2] are added dynamically applying the same separation routines of [2].

Column Generation Methods. In [109] also a branch-and-price and a branch-and-price-and-cut methods are presented. They are designed to tackle the DVRPMS and are as a consequence adaptable to the double TSP with multiple stacks. Both algorithms are based on Dantzig-Wolfe reformulations of an infeasible path model presented in the same paper [109] (see p. 330 in [40] for an explanation of Dantzig-Wolfe reformulation). In the first reformulation the columns are associated to pairs of pickup and delivery circuits performed by the vehicles and that are consistent with the loading constraints of the problem (last-in-first-out rule and capacity constraints). In the second reformulation each column represents a pickup or a delivery circuit. A column generation approach [50] is used in the resolution of both reformulations. The pricing problem for the first reformulation is an *elementary shortest path problem with capacity and pickup-and-delivery*, see [178]. The pricing problem for the second reformulation is reduced to a *shortest path with resource constraints* [110]. For the second reformulation a separation step is needed in order to include the infeasible path constraints, thus giving rise to a branch-and-cut-and-price algorithm

Comparison. Among all exact algorithms the best computational performances are reported in [2] and [16] whose results are comparable. On benchmark instances, the algorithm of [2] solves an instance with 28 items (for a 4×7 configuration). However, the results of the exact methods surveyed in this section clearly indicate that the computational difficulty of the problem increases with the value of the capacity. In this sense, the two stacks case seems to be the most difficult one. This is experimentally confirmed by the results obtained by the algorithms described above: the algorithm of [2] solves to optimality instances with two stacks and 14 items, within three hours. Only the algorithm of [44] solves three out of twenty benchmark instances with two stacks and 16 items.

Chapter 3

Models for the Double TSP with Multiple Stacks

Contents

In this chapter we introduce a new formulation for the double TSP with multiple stacks. The formulation we propose is based on an extended integer linear programming formulation for the TSP (possibly with precedence relations between locations) presented in Sarin *et al.* [171]. The latter involves two sets of variables: the arc variables that are used to determine the route of the traveling salesman in the city and to compute the value of solutions, and the precedence variables that determine whether a certain location precedes another location in the route.

To formulate the double TSP with multiple stacks as an integer linear program, we duplicate the model of Sarin *et al.* to describe pairs of Hamiltonian circuits and we introduce a new family of constraints ensuring that both Hamiltonian circuits respect the last-in-first-out policy. These new constraints involve only the precedence variables and are similar to the so-called infeasible path constraints used in Alba Martínez *et al.* [2] to ensure the consistency with the last-in-first-out policy.

In this chapter we will also consider the particular case of the double TSP with multiple stacks of infinite capacity. The infinite capacity case is easier to study than its finite

capacity counter-part, since its combinatorial structure is better understood and characterized. More precisely, one can decide efficiently whether a pair of Hamiltonian circuits represents a feasible solution to the double TSP with stacks of infinite capacity [32, 183]. Using this result, the precedence variables allow us to model the feasible solutions to this version of the problem by using polynomially-many constraints, when the number of stacks is fixed. In fact, this was our initial motivation for modeling the problem using precedence variables.

The remainder of this chapter consists of four sections. In Section 3.1 we introduce the notation used throughout the chapter to formulate the double TSP with multiple stacks as an integer linear program. We next consider two versions of the double TSP with multiple stacks. In Section 3.2 we consider the case in which the stacks have a finite capacity. We first provide a new formulation for the problem. Then we discuss the problem of recognizing whether a pair of Hamiltonian circuits is feasible. In Section 3.3 we consider the double TSP with multiple stacks of infinite capacity. We show that the formulation for the general case can be greatly simplified when the stacks have an unlimited loading capacity. This simplification relies on an elegant characterization found in [32, 183], of the s-consistent pairs of Hamiltonian circuits. In Section 3.4 we draw some conclusions.

3.1 Definitions

In this chapter n represents the number of items to be transported in the double TSP with multiple stacks, s the number of stacks of the vehicle and q their capacity. To avoid some triviality we will always assume that $n \geq 3$, and $sq \geq n$. As already done in Section 2.2, we also consider the complete digraph $G_n = (V, A)$, where $V = \{0, \ldots, n\}$. For every $A' \subseteq A$, the set of vertices entered or left by the arcs in A' is indicated with $V(A')$. Unless differently stated, we consider paths and circuits of G_n as sets of arcs. We recall that a pair of Hamiltonian circuits is s, q-consistent if it admits an s, q-loading plan. The latter is informally a disposition of the items in s stacks of capacity q allowing to perform the circuits in the pair without infringing the last-in-first-out rule.

For later convenience, we extend the notion of s, q-loading plan to pairs of paths. To this end, we point out that a path $R = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)\}$ of $G_n \setminus \{0\}$ *induces* the partial ordering $\prec_R$ on $\{1, \ldots, n\}$ defined by $v_i \prec_R v_j$ whenever $1 \leq i < j \leq k$. An s, q-loading plan for a pair (R_1, R_2) of paths of G_n is a stack disposition of the items that are commonly carried by both paths and such that the items are loaded according to the order of the first path and unloaded according to the reverse order of the second path.

More formally, given two integers s and q and a pair (R_1, R_2) of paths of G_n with $0 \notin V(R_1)$ and $0 \notin V(R_2)$, an s, q-loading plan for (R_1, R_2) is a collection of s finite totally ordered sets $\{(Q_1, \prec_{Q_1}), \ldots, (Q_s, \prec_{Q_s})\}$ such that:

LP1. $Q_1, \ldots, Q_s$ partition $V(R_1) \cap V(R_2)$

LP2. $|Q_i| \leq q$ for every $i = 1, \ldots, s$

LP3. for every $i = 1, \ldots, s$, we have that $k \prec_{Q_i} \ell$ implies $k \prec_{R_1} \ell$ and $\ell \prec_{R_2} k$ for all $k, \ell \in Q_i$

The pair (R_1, R_2) is said s, q-*consistent* if it admits an s, q-loading plan. As for Hamiltonian circuits we will also say that R_1 and R_2 are s, q-consistent whenever (R_1, R_2) is. When q is infinite, we write, as usual, s-*loading plan* and s-*consistent*. This observation is straightforward:

Observation 3.1.1. Let H_1 and H_2 be two Hamiltonian circuits of G_n. Let H_1' and H_2' be the paths obtained respectively from H_1 and H_2 by removing vertex 0. The pair (H_1, H_2) is s, q-consistent if and only if (R_1, R_2) is s, q-consistent for every subpath R_1 of H_1' and every subpath R_2 of H_2'.

3.2 Stacks of Finite Capacity

In this section we focus on the double TSP with multiple stacks of finite capacity. According to the description given in Section 2.2 the problem consists in finding a pair (H_1, H_2) of s, q-consistent Hamiltonian circuits of G_n minimizing $c^1(H_1) + c^2(H_2)$ and a loading plan for (H_1, H_2). We recall that when $s = 1$ or $s = n$, then the double TSP with multiple stacks is equivalent to the TSP. Hence, from now on, we assume that $2 \leq s \leq n - 1$.

We first give a new integer linear programming formulation. Then we focus on the problem of recognizing the s, q-consistency of a pair of Hamiltonian circuits.

3.2.1 Integer Linear Programming Formulation

We formulate the double TSP with multiple stacks of finite capacity as an integer linear program. We start by reviewing an integer linear programming formulation for the TSP on which we base our work. Then we adapt it to formulate our problem.

An Interlude on the Traveling Salesman Problem

Here we consider the ATSP on the digraph G_n. We now review a formulation for this problem introduced by Sarin *et al.* [171]. It is based on two sets of variables that arise from the equivalent description of Hamiltonian circuits as subgraphs of G_n, and as linear orderings of $\{1, \ldots, n\}$. Indeed, recall that, since 0 is the depot, each Hamiltonian circuit of G_n induces a linear ordering $\prec_H$ over $V \setminus \{0\}$, see Section 2.2.

Given a Hamiltonian circuit H of G_n, we call *characteristic point* of H the point

$(\chi^H, \gamma^H) \in \{0,1\}^{n(n+1)} \times \{0,1\}^{n(n-1)}$ defined by:

$$\chi_{ij}^H = \begin{cases} 1 & \text{if } (i,j) \in H, \\ 0 & \text{otherwise,} \end{cases} \quad \text{for all } i \neq j \in V,$$

$$\gamma_{ij}^H = \begin{cases} 1 & \text{if } i \prec_H j, \\ 0 & \text{otherwise,} \end{cases} \quad \text{for all } i \neq j \in V \setminus \{0\}.$$

Hence, χ^H describes the route of the traveling salesman corresponding to H whereas γ^H captures the precedence relations between pairs of locations visited performing this route.

A point $(x, y) \in \mathbb{R}^{n(n+1)} \times \mathbb{R}^{n(n-1)}$ is the characteristic point of a Hamiltonian circuit of G_n if and only if it is a solution to the following system of contraints [171]:

$$\sum_{j=0, j\neq i}^{n} x_{ij} = 1 \qquad \text{for all } i \in V, \tag{3.1}$$

$$\sum_{i=0, i\neq j}^{n} x_{ij} = 1 \qquad \text{for all } j \in V, \tag{3.2}$$

$$y_{ij} + y_{ji} = 1 \qquad \text{for all } i \neq j \in V \setminus \{0\}, \tag{3.3}$$

$$y_{ij} + y_{jk} + y_{ki} \geq 1 \qquad \text{for all } i \neq j \neq k \neq i \in V \setminus \{0\}, \tag{3.4}$$

$$x_{ij} \leq y_{ij} \qquad \text{for all } i \neq j \in V \setminus \{0\}, \tag{3.5}$$

$$y_{ij} \in \{0,1\} \qquad \text{for all } i \neq j \in V \setminus \{0\}, \tag{3.6}$$

$$x_{ij} \in \{0,1\} \qquad \text{for all } i \neq j \in V. \tag{3.7}$$

From now on, variables x will be called *arc variables*, and variables y *precedence variables*. Under the integrality constraints (3.6) and (3.7), constraints (3.1) and (3.2) impose that each vertex of a Hamiltonian circuit H has exactly one entering and one leaving arc. Constraints (3.3) and (3.4) respectively describe the antisymmetry and the transitivity of the linear ordering $\prec_H$. Hence they are respectively called *antisymmetry* and *transitivity constraints*. Finally (3.5) implies that if $(i,j) \in H$ then $i \prec_H j$.

Remark 3.2.1. Given a finite set $\{1, \ldots, n\}$ and a linear ordering $\prec$ on this set, the incidence vector of $\prec$ is the vector $\zeta^{\prec} \in \{0,1\}^{n(n-1)}$ defined by $\zeta_{ij}^{\prec} = 1$ if and only if $i \prec j$. The convex hull of the incidence vectors of the linear orderings of $\{1, \ldots, n\}$ is called *Linear Ordering Polytope*. Grötschel *et al.* [88] show that the Linear Ordering Polytope is $\text{conv}\{y \in \{0,1\}^{n(n-1)} : (3.3)\text{–}(3.4) \text{ are satisfied}\}$. The correspondence between Hamiltonian circuits in a complete digraph and linear orderings implies that for $n \geq 3$, $\text{proj}_y\{(x,y) : (3.1)\text{–}(3.7) \text{ are satisfied}\}$ coincides with the set of the vertices of the Linear Ordering Polytope.

From the TSP to the Double TSP with Multiple Stacks

In order to describe the solutions to the double TSP with multiple stacks of finite capacity, we introduce the variables $(x^1, y^1, x^2, y^2) \in \mathbb{R}^{n(n+1)} \times \mathbb{R}^{n(n-1)} \times \mathbb{R}^{n(n+1)} \times \mathbb{R}^{n(n-1)}$. Here, (x^T, y^T) is a pair of arc and precedence variables as those used in the TSP formulation presented above, for $T = 1, 2$. We are interested in the case in which (x^1, y^1) and (x^2, y^2) represent a feasible pair of pickup and delivery Hamiltonian circuits for the double TSP with multiple stacks. A natural requirement is that (x^1, y^1) and (x^2, y^2) are the characteristic points of two Hamiltonian circuits H_1 and H_2 of G_n. In this case, the point (x^1, y^1, x^2, y^2) is called *characteristic point of the pair* (H_1, H_2). In order to consider exactly the characteristic points of the s, q-consistent pairs of Hamiltonian circuits of G_n, we now introduce a new family of constraints.

The y-infeasible Path Constraints. A pair (R_1, R_2) of paths of $G_n \setminus \{0\}$ is s, q-*infeasible* if (R_1, R_2) is not s, q-consistent. Given an s, q-infeasible pair of paths (R_1, R_2), the corresponding *y-infeasible path constraint* is the inequality:

$$y^1(R_1) + y^2(R_2) \leq |R_1| + |R_2| - 1. \tag{3.8}$$

Since (R_1, R_2) is s, q-infeasible if and only if (R_2, R_1) is s, q-infeasible, we also say that the two paths R_1 and R_2 are s, q-infeasible. The family of s, q-infeasible pairs of paths is closed under taking subdivisions of s, q-infeasible paths, as shown in this lemma.

Lemma 3.2.2. *Let R_1 and R_2 be two s, q-infeasible paths of G_n. Let also $R_1^\star$ be obtained from R_1 by replacing one of its arcs by a path P of G_n not visiting 0 nor the vertices in $V(R_1)$. Then $R_1^\star$ and R_2 are s, q-infeasible.*

Proof. Suppose that $R_1^\star$ and R_2 have an s, q-loading plan $L^\star = \{(Q_1^\star, \prec_{Q_1^\star}), \ldots, (Q_s^\star, \prec_{Q_s^\star})\}$. For $i = 1, \ldots, s$, we define

- $Q_i = Q_i^\star \cap V(R_1) \cap V(R_2)$
- $p \prec_{Q_i} q$ if and only if $p \prec_{Q_i^\star} q$ for all $p, q \in Q_i$.

Since $V(R_1^\star) \cap V(R_2) \supseteq V(R_1) \cap V(R_2)$ and LP1–LP3 are verified by $L^\star$, we get that $\{(Q_1, \prec_{Q_1}), \ldots, (Q_s, \prec_{Q_s})\}$ is an s, q-loading plan for R_1 and R_2, a contradiction. □

Formulation. The integer linear programming formulation we propose for the double TSP with s stacks of finite capacity q is given by the following system of constraints:

$$\sum_{j=0,j\neq i}^{n} x_{ij}^T = 1 \qquad \text{for all } i \in V \text{ and } T = 1,2, \quad (3.9)$$

$$\sum_{i=0,i\neq j}^{n} x_{ij}^T = 1 \qquad \text{for all } j \in V \text{ and } T = 1,2, \quad (3.10)$$

$$y_{ij}^T + y_{ji}^T = 1 \qquad \text{for all } i \neq j \in V \setminus \{0\} \text{ and } T = 1,2, \quad (3.11)$$

$$y_{ij}^T + y_{jk}^T + y_{ki}^T \geq 1 \qquad \text{for all } i \neq j \neq k \neq i \in V \setminus \{0\} \text{ and } T = 1,2, \quad (3.12)$$

$$x_{ij}^T \leq y_{ij}^T \qquad \text{for all } i \neq j \in V \setminus \{0\} \text{ and } T = 1,2, \quad (3.13)$$

$$y^1(R_1) + y^2(R_2) \leq |R_1| + |R_2| - 1 \qquad \text{for all } s,q\text{-infeasible paths } R_1 \text{ and } R_2, \quad (3.14)$$

$$y_{ij}^T \in \{0,1\} \qquad \text{for all } i \neq j \in V \setminus \{0\} \text{ and } T = 1,2, \quad (3.15)$$

$$x_{ij}^T \in \{0,1\} \qquad \text{for all } i \neq j \in V \text{ and } T = 1,2. \quad (3.16)$$

In this proposition we prove the correctness of our formulation for the double TSP with multiple stacks.

Proposition 3.2.3. *A point (x^1, y^1, x^2, y^2) is the characteristic point of an s,q-consistent pair (H_1, H_2) of Hamiltonian circuits of G_n if and only if it satisfies* (3.9)–(3.16).

Proof. We show the sufficiency first. Let (x^1, y^1, x^2, y^2) satisfy (3.9)–(3.16). Then (x^1, y^1) and (x^2, y^2) satisfy (3.1)-(3.7). From the latter we get that (x^1, y^1) and (x^2, y^2) respectively are the characteristic points of two Hamiltonian circuits H_1 and H_2 of G_n.

Suppose that $H_1 = 0, v_1, v_2 \ldots, v_n, 0$ and $H_2 = 0, w_1, w_2 \ldots, w_n, 0$. If H_1 and H_2 are not s,q-consistent, the paths $R_1 = \{(v_1, v_2) \ldots, (v_{n-1}, v_n)\}$ and $R_2 = \{(w_1, w_2), \ldots, (w_{n-1}, w_n)\}$ are s,q-infeasible. In addition $x^1(R_1) = n - 1 = x^2(R_2)$, yielding $x^1(R_1) + x^2(R_2) = |R_1| + |R_2|$. Since $y_{ij}^1 \geq x_{ij}^1$ and $y_{ij}^2 \geq x_{ij}^2$ for all distinct $i, j \in \{1, \ldots, n\}$, also (x^1, y^1, x^2, y^2) violates (3.14) associated with R_1 and R_2, a contradiction.

We now show the necessity. Let us take an s,q-consistent pair (H_1, H_2) of a Hamiltonian circuits of G_n. We have that $(\chi^{H_1}, \gamma^{H_1})$ and $(\chi^{H_2}, \gamma^{H_2})$ satisfy (3.1)-(3.7). This means that, the point $(\chi^{H_1}, \gamma^{H_1}, \chi^{H_2}, \gamma^{H_2})$ satisfies all constraints in our formulation, except maybe (3.14).

Suppose that (3.14) is violated by $(\gamma^{H_1}, \gamma^{H_2})$ for some s,q-infeasible paths R_1 and R_2. Then $\gamma_a^{H_1} = 1$ for all $a \in R_1$ and $\gamma_b^{H_2} = 1$ for all $b \in R_2$. Then there exist two paths $R_1^\star$ and $R_2^\star$ of $G_n \setminus \{0\}$ respectively obtained from R_1 and R_2 by arc substitutions as in Lemma 3.2.2 and such that $\chi_a^{H_1} = 1$ for every $a \in R_1^\star$ and $\chi_b^{H_2} = 1$ for every $b \in R_2^\star$.

By Lemma 3.2.2, $R_1^\star$ and $R_2^\star$ are s,q-infeasible, since so are R_1 and R_2. By Observation 3.1.1 and by $H_1 \supseteq R_1^\star$ and $H_2 \supseteq R_2^\star$, the s,q-loading plan for (H_1, H_2) yields an s,q-loading plan for $R_1^\star$ and $R_2^\star$, a contradiction.

□

The y-infeasible path constraints are similar to the infeasible path constraints introduced in Alba Martínez *et al.* [2] as explained in Remark 3.2.4.

Remark 3.2.4. Let R_1 and R_2 be two paths in G_n such that no pair (H_1, H_2) of s,q-consistent Hamiltonian circuits such that $H_1 \supseteq R_1$ and $H_2 \supseteq R_2$ exists. The corresponding *infeasible path constraint* proposed in Alba Martínez *et al.* [2] is:

$$x^1(R_1) + x^2(R_2) \leq |R_1| + |R_2| - 1. \tag{3.17}$$

A pair of Hamiltonian circuits H_1 and H_2 is s,q-consistent only if (χ^{H_1}, χ^{H_2}) satisfies (3.17) [2]. By an example we show that the family of infeasible path constraints 3.17 is larger than the family of y-infeasible path constraints 3.14. Indeed, let us consider an instance of the double TSP with two stacks of capacity three and six items, and the two paths $R_1 = 1, 2, 5, 3, 4$ and $R_2 = 3, 4, 1, 2$. Given Hamiltonian circuits $H_1 \supseteq R_1$ and $H_2 \supseteq R_2$, then either $5 \prec_{H_2} 3 \prec_{H_2} 4$ or $1 \prec_{H_2} 2 \prec_{H_2} 5$ and the same ordering relations hold in the circuit H_1. Thus, such H_1 and H_2 cannot be 2-consistent. It follows that constraint (3.17) associated with R_1 and R_2 must be satisfied by the $2,3$-consistent pairs of Hamiltonian circuits of G_6. On the other hand, the paths R_1 and R_2 admit the $2,3$-loading plan $\{(Q_1, \prec_{Q_1}), (Q_2, \prec_{Q_2})\}$ with $Q_1 = \{2, 3\}$, $2 \prec_{Q_1} 3$ and $Q_2 = \{4\}$. Hence we cannot define constraint (3.8) for R_1 and R_2. In addition, trying to do so would result in an incorrect constraint for the double TSP with multiple stacks. Indeed, the circuits $H_1 = 0, 1, 2, 5, 3, 4, 6, 0$ and $H_2 = 0, 6, 3, 4, 5, 1, 2, 0$ are $2,3$-consistent and $(\gamma^{H_1}, \gamma^{H_2})$ violates the constraint (3.8) associated with R_1 and R_2.

We conclude with an important property of y-infeasible path constraints. We show that the y-infeasible path constraints are stronger when associated with s,q-infeasible paths of small length. Let us make this idea more precise. We recall that, for any path R, the path R_1/v is the path obtained by "skipping" the vertex v when following R, see Section 1.3.2. Suppose that R_1 and R_2 are two s,q-infeasible paths and consider the two paths $R'_1 = R_1/v$ and $R'_2 = R_2/v$ where v is a vertex visited by at least one path of R_1 and R_2. Under mild conditions, if also R'_1 and R'_2 are s,q-infeasible, the y-infeasible path constraint associated with R_1 and R_2 is implied by the one associated with R'_1 and R'_2.

Proposition 3.2.5. *Let $(x^1, y^1, x^2, y^2) \in [0,1]^{n(n+1)} \times [0,1]^{n(n-1)} \times [0,1]^{n(n+1)} \times [0,1]^{n(n-1)}$ satisfy (3.9)-(3.13). Let R_1 and R_2 be two s,q-infeasible paths of G_n and, for some $v \in V(R_1) \cup V(R_2)$ let $R'_1 = R_1/v$ and $R'_2 = R_2/v$. If R'_1 and R'_2 are s,q-infeasible paths of G_n and (x^1, y^1, x^2, y^2) satisfies*

$$y^1(R'_1) + y^1(R'_2) \leq |R'_1| + |R'_2| - 1 \tag{3.18}$$

then it also satisfies the y-infeasible path constraint associated with R_1 and R_2.

Proof. Let us prove that (x^1, y^1, x^2, y^2) satisfies $y^1(R_1) + y^2(R_2) \leq |R_1| + |R_2| - 1$ under the hypothesis of the proposition. For $T = 1, 2$, we define $b_T \in \{0, 1\}$ such that $b_T = 1$ if and only $v \in R_T$. For $T = 1, 2$, if $v \in R_T$ then R'_T is obtained by removing two arcs of R_T and adding one new arc, then we have:

$$|R'_T| = |R_T| - b_T \qquad \text{for } T = 1, 2. \tag{3.19}$$

In addition, $y^1(R_1) \leq y^1(R'_1) + b_1$. The only nontrivial case to verify is when $b_1 = 1$, that is when $v \in R_1$. If v is the starting or the ending point of R_1, the equality above follows from the observation that R'_1 is a subpath of R_1 and from $y^1_{vj} \leq 1$ and $y^1_{jv} \leq 1$ for all $j \in V \setminus \{0, v\}$. Thus, let us consider the case in which v is not the starting nor the ending point of R_1. Let us call P the subpath of R_1 going from its starting point to the vertex p preceding v in R_1. Similarly, let us call S the subpath of R_1 going from the vertex s following v in R_1 to its ending point. We obtain $y^1(R_1) = y^1(P) + y^1(S) + y^1_{pv} + y^1_{vs} \leq y^1(P) + y^1(S) + y^1_{ps} + 1$. To get the last inequality in the previous expression we rewrite the transitivity constraint $y^1_{ps} + y^1_{sv} + y^1_{vp} \geq 1$ from (3.12) as $y^1_{pv} + y^1_{vs} \leq y^1_{ps} + 1$ by using the antisymmetry constraints (3.11). Since $R'_1 = P \cup \{(p, s)\} \cup S$ we have $y^1(R_1) \leq y^1(R'_1) + 1$. In a similar manner one can prove that $y^2(R_2) \leq y^2(R'_2) + b_2$. Hence summing up these two expressions we finally have:

$$y^1(R_1) + y^2(R_2) \leq y^1(R'_1) + y^2(R'_2) + b_1 + b_2 \leq |R'_1| + |R'_2| - 1 + b_1 + b_2 = |R_1| + |R_2| - 1.$$

In the last equality we used (3.19). This concludes the proof. □

We say that two paths R_1 and R_2 are *minimally* s, q-infeasible if R_1/v and R_2/v are s, q-consistent for every $v \in V(R_1) \cup V(R_2)$. The result of Proposition 3.2.5 implies that the y-infeasible path constraints are more interesting when associated with minimally s, q-infeasible paths.

3.2.2 Recognizing the s, q-consistency by Graph Coloring

In the previous section we have introduced a new integer linear programming formulation for the double TSP with s stacks of capacity q. In this section we discuss the construction of an s, q-loading plan starting from an s, q-consistent pair of Hamiltonian circuits. Indeed, our formulation only involves variables describing the pickup and the delivery circuits. In particular, a feasible solution to formulation (3.9)–(3.16) does not carry any information about the construction of an s, q-loading plan. It only ensures that the corresponding pair of Hamiltonian circuit is s, q-consistent. This is without loss of generality because the following question admits a positive answer when s is fixed (as first proved by Bonomo *et al.* [23]).

Question 3.2.6. Given a pair of Hamiltonian circuits (H_1, H_2) of G_n, can we decide, in polynomial time in n, whether it is s, q-consistent and, in positive case, construct a corresponding s, q-loading plan?

Bonomo *et al.* give an algorithm answering Question 3.2.6 that runs in $O(n^{s^2+s+1}s^3)$ time. The underlying idea is to reduce Question 3.2.6 to an equivalent problem of graph coloring. We now recall their construction since we want to show that when $s = 2$ there is a $O(n^2)$ algorithm that solves Question 3.2.6. As a side remark, we mention that several authors [44, 2, 157] have observed that the difficulty of the double TSP with multiple stacks of finite capacity increases, from a computational point of view, with the value of the capacity. Hence, the case of the double TSP with two stacks is challenging on its own.

Let us be given a pair (H_1, H_2) of Hamiltonian circuits of G_n. We may assume, up to a relabeling of the indices, that $H_1 = 0, n, n-1, n-2, \ldots, 1, 0$. We describe H_2 by a permutation $\pi\colon \{1, \ldots, n\} \to \{1, \ldots, n\}$ where $\pi(i)$ is the position of vertex i in the circuit H_2. Clearly, π can be constructed in linear time when H_2 is known. We point out that, for all distinct $i, j \in \{1, \ldots, n\}$, π is defined equivalently by:

$$\pi(i) < \pi(j) \text{ if and only if } i \prec_{H_2} j.$$

If H_1 and H_2 are respectively a pickup and a delivery circuits for the double TSP with multiple stacks fulfilling the last-in-first-out rule, items i and j cannot be in the same stack when $i \prec_{H_1} j$ and $i \prec_{H_2} j$. The first precedence relation is equivalent to $i > j$, whereas the second is equivalent to $\pi(i) < \pi(j)$. We can describe these "conflicts" by the permutation graph $G(\pi) = (V_\pi, E_\pi)$ defined as follows. The vertex set is $V_\pi = V \setminus \{0\} = \{1, \ldots, n\}$ and there is an edge between i and j if and only if both $i > j$ and $\pi(i) < \pi(j)$ hold, for every $i, j \in V \setminus \{0\}$.

We now show that Question 3.2.6 is equivalent to a specific coloring problem on $G(\pi)$. A coloring of a graph is a labeling of the vertices of the graph, in which the labels are called colors and such that the endpoints of each edge are not assigned the same color. A k-coloring is a coloring using at most k distinct colors. See Section 1.3.1 for more details.

Unless differently stated, given a k-coloring of $G(\pi)$, C_i denotes the color class of i, *i.e.,* the set of vertices colored with color i, for every $i = 1, \ldots, k$. As observed by Bonomo *et al.* [23], Question 3.2.6 is then equivalent to the so-called *Bounded Coloring Problem* [98] of $G(\pi)$:

Problem 3.2.7 (Bounded Coloring Problem of $G(\pi)$)**.** Given the integers $s, q \geq 1$, find an s-coloring of $G(\pi)$ such that $|C_i| \leq q$ for each $i = 1, \ldots, s$, or prove that none exists.

We show the equivalence. Let us take a pair (H_1, H_2) of Hamiltonian circuits of G_n. Without loss of generality, let $H_1 = 0, n, n-1, \ldots, 1, 0$ and π be the permutation describing H_2 under this assumption. Given a coloring solving Problem 3.2.7, it can be seen that $L = \{(C_1, \prec_1), \ldots, (C_s, \prec_s)\}$ where $i \prec_k j$ if and only if $i > j$ for every distinct $i, j \in C_k$ and $1 \leq k \leq s$, is an s, q-loading plan for (H_1, H_2). Conversely, if $\{(Q_1, \prec_{Q_1}), \ldots, (Q_s, \prec_{Q_s})\}$ is an s, q-loading plan for (H_1, H_2), then assigning color i to vertex j for all $j \in Q_i$ and $i = 1, \ldots, s$ yields an s-coloring solving Problem 3.2.7.

Example 3.2.8 illustrates the construction of $G(\pi)$ for a pair of Hamiltonian circuits, as well as the relation between colorings and loading plans.

Example 3.2.8. Let us consider the pair of Hamiltonian circuits (H_1, H_2) of G_6 where:

- $H_1 = 0, 6, 5, 4, 3, 2, 1, 0,$
- $H_2 = 0, 1, 3, 2, 5, 6, 4, 0.$

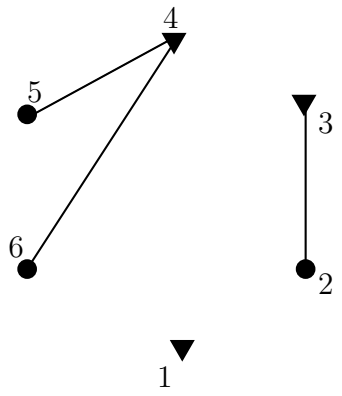


Figure 3.1: The permutation graph $G(\pi)$ corresponding to H_1 and H_2 of Example 3.2.8. The shape of the vertices indicates a 2-coloring that solves Problem 3.2.7 for $s = 2$ and $q = 3$. Circles indicate vertices in C_1, triangles those in C_2.

Since π represents the position of the vertices in the delivery circuit, we get $\pi(1) = 1, \pi(2) = 3, \pi(3) = 2, \pi(4) = 6, \pi(5) = 4, \pi(6) = 5$. The graph $G(\pi)$ is represented in Figure 3.1. The pair (H_1, H_2) is $2, 3$-consistent. Indeed the following 2-coloring of $G(\pi)$ solves Problem 3.2.7 for $q = 3$: $C_1 = \{3, 5, 6\}$ and $C_2 = \{1, 2, 4\}$. Hence a $2, 3$-loading plan is given by $\{(C_1, \prec_1), (C_2, \prec_2)\}$ with $6 \prec_1 5 \prec_1 3$ and $4 \prec_2 2 \prec_2 1$. A different 2-coloring of $G(\pi)$ that solves the same problem is described in Figure 3.1. It yields the $2, 3$-loading for (H_1, H_2) where items 6, 5 and 2 are in this order in the first stack, and items 4, 3 and 1 are in this order in the second stack.

Bonomo *et al.* [23] solve in polynomial time, for fixed s, the following generalization of Problem 3.2.7:

Problem 3.2.9 (Capacitated Coloring Problem of $G(\pi)$ [23])**.** Given nonnegative integers s and $\kappa_1, \ldots, \kappa_s$, find an s-coloring of $G(\pi)$, such that $|C_i| \leq \kappa_i$ for $i = 1, \ldots, s$, or prove that none exists.

Clearly, choosing $\kappa_1 = \kappa_2 = \cdots = \kappa_s = q$, a polynomial time algorithm to solve Problem 3.2.9 is also a polynomial time algorithm to solve Problem 3.2.7. We mention that Problem 3.2.7 is NP-hard even for fixed $q \geq 6$, a result found by Jansen [111]. The main result of Bonomo *et al.* [23] is that Problem 3.2.9 can be solved in $O(n^{s^2+s+1}s^3)$ time. Since we may always assume that the number of stacks is fixed, checking whether a pair of Hamiltonian circuits is s, q-consistent can be done in polynomial time. In positive case Bonomo *et al.*'s algorithm also yields an s, q-loading plan. However, this algorithm is quite expensive even when the number of stacks is low. We provide a more efficient algorithm when there are two stacks, as shown in the following proposition.

Proposition 3.2.10. *The Capacitated Coloring Problem of $G(\pi)$ can be solved in $O(n^2)$ time when $s = 2$.*

Proof. Without loss of generality let us assume that the instance of the Capacitated Coloring Problem for $s = 2$ is such that $\kappa_1 \geq \kappa_2$. Let $G_1, \ldots, G_k$ be the connected components of $G(\pi)$, with vertex set respectively $V(G_1), \ldots, V(G_k)$. We check that $G(\pi)$ admits a 2-coloring. This can be done by exploring the edge set of $G(\pi)$ and thus the check is performed in $O(n^2)$ time. From now on, we assume that $G(\pi)$ admits a 2-coloring, as otherwise Problem 3.2.9 for $s = 2$ has no solution. A 2-coloring can be explicitly found by performing the same exploration used to certificate its existence.

Let C_1 and C_2 be respectively the color classes of 1 and 2 in this 2-coloring of $G(\pi)$. For $j = 1, \ldots, k$ we define the pair (m^j, M^j) where m^j (resp. M^j) is the minimum (resp. maximum) between $|C_1 \cap G_j|$ and $|C_2 \cap G_j|$. Note that the pair does not depend on the considered 2-coloring of $G(\pi)$, because, when it exists, a 2-coloring of a connected graph is unique up to switching the two colors. The construction of the pairs (m^j, M^j) for all $j = 1, \ldots, k$ takes $O(n)$ time. At this point, if $\kappa_2 < \sum_{j=1}^{k} m^j$ no solution to the Capacitated Coloring Problem exists. We have the following Claim.

Claim 3.2.11. *The Capacitated Coloring Problem for $G(\pi)$ and $s = 2$ admits a solution if and only if the optimal value of the following problem is at least $\sum_{j=1}^{k} M^j - \kappa_2$:*

$$\begin{array}{ll} \max & \sum_{j=1}^{k} z_j (M^j - m^j) \\ & \sum_{j=1}^{k} z_j (M^j - m^j) \leq \kappa_1 - \sum_{j=1}^{k} m^j \quad (K) \\ & z_j \in \{0, 1\} \end{array}$$

Proof. We first prove the "only if" part. Let us consider a 2-coloring solving the Capacitated Coloring Problem for $G(\pi)$ and $s = 2$. We assume that K_1 and K_2 are respectively the color classes of 1 and 2 in this 2-coloring. We identify the latter with the vector $z^\star \in \{0, 1\}^k$ defined for $j = 1, \ldots, k$ by

$$z_j^\star = \begin{cases} 1 & \text{if } |K_1 \cap V(G_j)| = M^j, \\ 0 & \text{otherwise.} \end{cases}$$

In other words, $z_j^\star = 1$ whenever the 2-coloring uses M^j times color 1 in G_j.

Hence, $z_\ell^\star = 0$ whenever it uses m^ℓ times color 1 in component G_ℓ. Thus, defining $\mathcal{J} = \{j \in \{1, \ldots, k\} \colon z_j^\star = 1\}$ and using $|K_1| \leq \kappa_1$, we have that:

$$\sum_{j=1}^{k} m^j + \sum_{j=1}^{k} z_j^\star (M^j - m^j) = \sum_{j \in \mathcal{J}} M^j + \sum_{\ell \in \bar{\mathcal{J}}} m^j \leq \kappa_1.$$

Hence $z^\star$ is a feasible solution to problem (K). Note also that the color 2 is used m^j times in the components indexed by $\mathcal{J}$ and M^j times in those indexed by $\bar{\mathcal{J}}$. This means that:

$$\sum_{j \in \mathcal{J}} m^j + \sum_{\ell \in \bar{\mathcal{J}}} M^j \leq \kappa_2. \tag{3.20}$$

The value of $z^\star$ in problem (K) is $\sum_{j=1}^{k} z_j^\star(M^j - m^j) = \sum_{j\in\mathcal{J}}(M^j - m^j)$. Using (3.20) the latter is at least $\sum_{j\in\mathcal{J}} M_j + \sum_{j\in\bar{\mathcal{J}}} M_j - \kappa_2 = \sum_{j=1}^{k} M^j - \kappa_2$. Hence problem (K) admits an optimal solution of value at least $\sum_{j=1}^{k} M^j - \kappa_2$.

We now prove the "if" part. Let $z^\star$ be the optimal solution to problem (K). Each pair (m^j, M^j) corresponds to a 2-coloring of the component G_j, that is explicitly known. For every $j = 1, \ldots, k$, let V_j be the set of vertices of G_j of cardinality M^j and whose elements have the same color in this 2-coloring. For every $j = 1, \ldots, k$, if $z_j^\star = 1$ we assign color 1 to the vertices in V_j and color 2 to the vertices in $V(G_j) \setminus V_j$; if $z_j^\star = 0$ we assign color 1 to the vertices in $V(G_j) \setminus V_j$ and color 2 to the vertices in V_j. This results in a 2-coloring of $G(\pi)$ solving the Capacitated Coloring Problem for $G(\pi)$ and $s = 2$. ■

Problem (K) in the previous claim is a knapsack problem, see Section 1.6. It can be solved in $O(k(\kappa_1 - \sum_{j=1}^{k} m^j))$ time using a dynamic programming algorithm since $M^j - m^j \geq 0$ for all $j = 1, \ldots, k$. In the case of the Capacitated Coloring Problem above it is not restrictive to assume $\kappa_1 - \sum_{j=1}^{k} m^j \leq n$. This, together with $k \leq n$, yields a $O(n^2)$ running time of our algorithm. □

Remark 3.2.12. Please, note that in the previous proof we do not use the fact that $G(\pi)$ is a permutation graph. Then the Capacitated Coloring Problem can be solved in $O(n^2)$ time for $s = 2$ on every graph. (In the general case, the Capacitated Coloring Problem for $s = 2$ is obtained from Problem 3.2.9 by replacing $G(\pi)$ by a generic graph G.)

Remark 3.2.13. We point out that the resolution of the Capacitated Coloring Problem via the knapsack problem (K) of Claim 3.2.11 can be used to decide whether a pair of paths is s, q-consistent or not. In this case a permutation graph is constructed as done for a pair of Hamiltonian circuits. The difference is that the vertex set of the permutation graph contains precisely the vertices visited by both paths, hence in general has cardinality smaller than n.

3.3 Stacks of Infinite Capacity

In this section we are interested in the double TSP with multiple stacks of infinite capacity. Since this is a special case of the general problem studied in Section 3.2, it can be stated as the problem of finding an s-consistent pair (H_1, H_2) of Hamiltonian circuits of two weighted digraphs (G_n, c^1) and (G_n, c^2) such that $c^1(H_1) + c^2(H_2)$ is minimum. That is, we omit the construction of an s-loading plan from our description. It turns out that an s-loading plan for an s-consistent pair of Hamiltonian circuits can be constructed in polynomial time, independently on the number of stacks. In Section 3.3.1 we present a simplification of formulation (3.9)–(3.16) for the infinite capacity case. In Section 3.3.2 we report the construction of an s-loading plan for an s-consistent pair of Hamiltonian circuits.

3.3.1 Integer Linear Programming Formulation

The formulation of Section 3.2.1 can be greatly simplified to describe the solutions to the double TSP with multiple stacks of infinite capacity. We obtain a polynomial-size formulation when s, the number of stacks, is fixed. Note that this is not the case in general for the formulation of Section 3.2.1, since constraints (3.14) are exponentially-many.

The simplification of our formulation relies on the following characterization of s-consistent Hamiltonian circuits, whose proof is postponed to Section 3.3.2:

Proposition 3.3.1 ([32, 183])**.** *Two Hamiltonian circuits of G_n are s-consistent if and only if no $s+1$ vertices of $V \setminus \{0\}$ appear in the same order in both circuits.*

The integer linear programming formulation (3.9)–(3.16) can then be rewritten as follows in the case of the double TSP with s stacks of infinite capacity:

$$\sum_{j=0, j\neq i}^{n} x_{ij}^T = 1 \qquad \text{for all } i \in V \text{ and } T = 1, 2, \tag{3.21}$$

$$\sum_{i=0, i\neq j}^{n} x_{ij}^T = 1 \qquad \text{for all } j \in V \text{ and } T = 1, 2, \tag{3.22}$$

$$y_{ij}^T + y_{ji}^T = 1 \qquad \text{for all } i \neq j \in V \setminus \{0\} \text{ and } T = 1, 2, \tag{3.23}$$

$$y_{ij}^T + y_{jk}^T + y_{ki}^T \geq 1 \qquad \text{for all } i \neq j \neq k \neq i \in V \setminus \{0\} \text{ and } T = 1, 2, \tag{3.24}$$

$$x_{ij}^T \leq y_{ij}^T \qquad \text{for all } i \neq j \in V \setminus \{0\} \text{ and } T = 1, 2, \tag{3.25}$$

$$\sum_{i=1}^{s} (y_{j_i j_{i+1}}^1 + y_{j_i j_{i+1}}^2) \geq 1 \qquad \text{for all distinct } j_1, \ldots, j_{s+1} \in V \setminus \{0\}, \tag{3.26}$$

$$y_{ij}^T \in \{0, 1\} \qquad \text{for all } i \neq j \in V \setminus \{0\} \text{ and } T = 1, 2, \tag{3.27}$$

$$x_{ij}^T \in \{0, 1\} \qquad \text{for all } i \neq j \in V \text{ and } T = 1, 2. \tag{3.28}$$

The only difference of formulation (3.21)–(3.28) with the formulation of Section 3.2.1 is that we replace (3.14) by constraints (3.26). We call constraints (3.26) *s-consistency constraints*. The s-consistency constraints can be rewritten as

$$\sum_{i=1}^{s} (y_{j_i j_{i+1}}^1 + y_{j_i j_{i+1}}^2) \leq 2s - 1,$$

for all distinct $j_1, \ldots, j_{s+1} \in \{1, \ldots, n\}$, by using constraint (3.23). Thus, from Proposition 3.3.1 and the fact that $\gamma_{ij}^{H_1} = 1$ (resp. $\gamma_{ij}^{H_2} = 1$) means that i precedes j in the Hamiltonian circuit H_1 (resp. H_2), constraints (3.26) are nothing but y-infeasible path constraints associated with all minimally s-infeasible paths of G_n. By Proposition 3.2.5 they imply all other y-infeasible constraints in the infinite capacity case. Hence (3.21)–(3.28) is a correct formulation for the double TSP with multiple stacks of infinite capacity.

3.3.2 Recognizing the s-consistency by Graph Coloring

Here we report a proof of Proposition 3.3.1 characterizing the s-consistent Hamiltonian circuits of G_n. The characterization was independently found in Casazza *et al.* [32] and Toulouse and Wolfler Calvo [183]. The proof of Proposition 3.3.1 can be also seen from an algorithmic perspective, thus yielding an s-loading plan for an s-consistent pair of Hamiltonian circuits.

We remember that the consistency requirement can be represented by a suitable permutation graph $G(\pi)$, as done in Section 3.2. For the sake of clarity, we repeat the construction of $G(\pi)$.

Let us be given two Hamiltonian circuits H_1 and H_2 and without loss of generality let $H_1 = 0, n, n-1, \ldots, 1, 0$. Hence H_2 corresponds to a permutation π of $\{1, \ldots, n\}$ where $\pi(i)$ is the position of i in H_2. We construct the permutation graph $G(\pi) = (V_\pi, E_\pi)$ by setting $V_\pi = \{1, \ldots, n\}$ and linking i and j by an edge whenever $i > j$ and $\pi(i) < \pi(j)$.

Proposition 3.3.1 states that the two Hamiltonian circuits H_1 and H_2 are s-consistent if and only if no $s+1$ vertices of $V \setminus \{0\}$ appear in the same order in both circuits. The starting point is the equivalence of an s-coloring of $G(\pi)$ with the existence of an s-loading plan for H_1 and H_2. More precisely, the color classes of an s-coloring correspond to a partition of the items in s sets $C_1, \ldots, C_s$. Then C_i can be ordered by restricting $\prec_{H_1}$ to its elements, for every $i = 1, \ldots, s$. The resulting collection of finite totally ordered sets is an s-loading plan for (H_1, H_2). Conversely, each set in the partition corresponding to an s-loading plan for (H_1, H_2) can be seen as a color class in an s-coloring of $G(\pi)$. We can now prove Proposition 3.3.1.

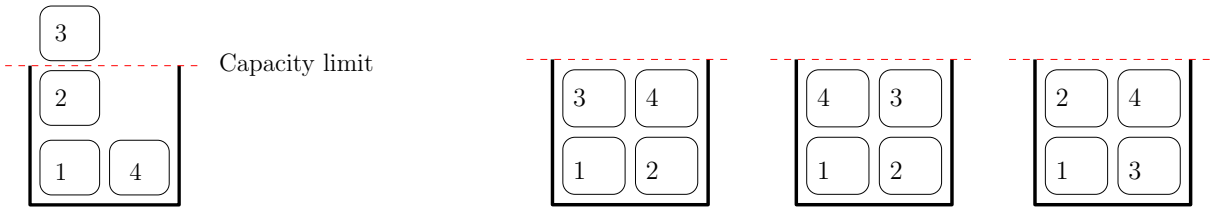
Proof of Proposition 3.3.1 (adapted from [26]). The necessity follows from the pigeonhole principle applied to the number of stacks (remember that $n > s$). For the sufficiency, consider (H_1, H_2) a pair of Hamiltonian circuits with no $s+1$ vertices of $V \setminus \{0\}$ in the same order. It corresponds to a permutation graph $G(\pi) = (V_\pi, E_\pi)$ thus having the following properties:

GP1. Every clique of $G(\pi)$ is given by a set of vertices $C \subseteq V_\pi$ appearing in the same order in both H_1 and H_2;

GP2. $G(\pi)$ being perfect — see Section 1.3.1 on perfect graphs — the chromatic number of $G(\pi)$ is equal to its clique number, that is $\chi(G(\pi)) = \omega(G(\pi))$.

By GP1 and our assumption, $\omega(G(\pi)) \leq s$. By GP2, $G(\pi)$ admits an s-coloring and we conclude. □

Using the result of Proposition 3.3.1 and its proof, we describe a polynomial algorithm given in [32] to decide the s-consistency of two Hamiltonian circuits H_1 and H_2 of G_n. This algorithm runs in $O(n \log n)$ time and, in case (H_1, H_2) is s-consistent, it constructs a corresponding s-loading plan. To see this, first observe that in linear time we can relabel the vertices of G_n so that $H_1 = 0, n, n-1, \ldots, 1, 0$ and also obtain the permutation π corresponding to the circuit H_2 under this relabeling. Now, an s-loading plan for (H_1, H_2)



(a) 2-consistency for $\tilde{P} = 0, 1, 2, 3, 4, 0$ and $\tilde{D} = 0, 3, 2, 1, 4, 0$.

(b) Loading plans for $\tilde{P}$ not violating capacity constraints. But $\tilde{D}$ cannot be performed.

Figure 3.2: The s-consistency does not imply the s, q-consistency, in general.

can be constructed from an s-coloring of $G(\pi)$. We solve then the minimum coloring problem on $G(\pi)$. This is done in $O(n \log n)$, *e.g.,* with the algorithm described at p. 167–168 in Golumbic's book [82]. It essentially consists in a greedy algorithm in which the choice of the first available color is made in $O(\log n)$ time by a dichotomic search. The coloring algorithm outputs the chromatic number $\chi(G(\pi))$ along with a $\chi(G(\pi))$-coloring.

We can improve the previous running time if we are only interested in proving the s-consistency of H_1 and H_2, without exhibiting an s-loading plan in positive case. In the following discussion Hamiltonian circuits are seen as linear orderings of $V \setminus \{0\}$. Let H_1, H_2 and π be as above. Deciding the s-consistency of (H_1, H_2) amounts to exhibit a largest cardinality set $S = \{i_1, \ldots, i_k\}$ of vertices in $V \setminus \{0\}$ appearing in the same order in both H_1 and H_2. Finding such a S is the same as finding the greatest k such that $i_1 > i_2 > \cdots > i_k$ and $i_1 \prec_{H_2} i_2 \prec_{H_2} \cdots \prec_{H_2} i_k$ for $i_j \in V \setminus \{0\}$ and $j = 1, \ldots, k$. This is known as the *Longest Decreasing Subsequence Problem* [173]. It can be solved in $O(n \log \log n)$ time, see Hunt and Szymanski [105]. This time complexity holds because H_2 corresponds to a permutation of $V \setminus \{0\}$ and it is achieved by using van Emde Boas trees [185] as underlying data structures.

Remark 3.3.2. The previous complexity results do not depend on the number of stacks s defining the instance of the problem.

We conclude with an example showing that, in general, this infinite capacity version of the problem is a relaxation of its finite capacity counterpart.

Example 3.3.3. Let us consider a small instance of the double TSP with two stacks of capacity two in which 4 items are to be delivered to the corresponding customers. The circuits $\tilde{P} = 0, 1, 2, 3, 4, 0$ and $\tilde{D} = 0, 3, 2, 1, 4, 0$ are 2-consistent as it can be seen by putting item 4 in one stack and all other items in the other stack (see Figure 3.2a). However, the loading configuration that guarantees the 2-consistency violates the capacity of one stack. In Figure 3.2b we depicted the three possible loading configurations that fulfill the prescribed capacity and that can be constructed by performing $\tilde{P}$ (up to switching the two stacks). None of them corresponds to a $2, 2$-loading plan for $(\tilde{P}, \tilde{D})$.

3.4 Conclusions

In this chapter we have modeled the double TSP with multiple stacks. We have considered the problem as presented in Petersen and Madsen [158] as well as the particular case in which the stacks have unlimited loading capacity.

We have first introduced a new formulation for the double TSP with multiple stacks. Its main feature is the presence of precedence variables. These are inherited from a formulation for the TSP introduced by Sarin *et al.* [171]. The precedence variables let us model the s, q-consistency by introducing the family of y-infeasible path constraints. These constraints along with those used to model the routing part, are sufficient to formulate the double TSP with multiple stacks as an integer linear program.

Our formulation does not take into account the construction of a loading plan but, as observed in Section 3.2.2, this is not restrictive. In particular, in the case of two stacks of capacity q, we have provided a $O(n^2)$ time algorithm to solve the subproblem of deciding whether a pair of Hamiltonian circuits is $2, q$-consistent. Our algorithm also generates a $2, q$-loading plan and improves, in the two stack case, the worst-case running time of an algorithm introduced by Bonomo *et al.* [23].

When adapted to the double TSP with multiple stacks of infinite capacity our formulation can be greatly simplified. Indeed, in this case, only a polynomial number of y-infeasible path constraints is needed, when the number of stacks is fixed. This property stems from the fact that, for both the finite capacity and infinite capacity cases, the strongest y-infeasible path constraints are those associated with minimally infeasible paths. Using a complete characterization of s-consistent pairs of Hamiltonian circuits given in terms of vertex sequences (see Proposition 3.3.1) we can provide all "minimal" y-infeasible path constraints in the infinite capacity case. Of course, the same constraints can be used in the formulation for the double TSP with multiple stacks with finite capacity.

Chapter 4

Polyhedral Results

Contents

In this chapter we investigate the structure of the convex hull of the integer points satisfying the constraints of the formulation introduced in Section 3.3.1 for the double TSP with multiple stacks of infinite capacity. In the following, we refer to this polytope as the DTSPMS polytope. Our formulation exhibits two components. The first one is given by constraints (3.21)–(3.25),(3.27),(3.28) used to describe pairs of Hamiltonian circuits in a complete digraph; the second component is given by constraints (3.26) enforcing the s-consistency. Accordingly, the polyhedral study conducted in this chapter is divided into two parts.

The first part provides links between the DTSPMS polytope and the PATSP polytope. The latter is the convex hull of the points satisfying constraints (3.1)–(3.7) in the ATSP formulation of Section 3.2.1. Our main contribution in this part is to show that the DTSPMS polytope inherits all the facets of the PATSP polytope, see Theorem 4.1.7. This is a desirable property since the latter has $2^{\Omega(\sqrt{n})}$ facets [60]. Hence, our result characterizes a super-polynomial number of facets of the DTSPMS polytope. In order to prove this result we also give closed-form expressions for the dimensions of the DTSPMS and PATSP polytopes. On the other hand, due to the s-consistency requirement, the facets inherited

from the PATSP polytope are not sufficient for a complete description of the DTSPMS polytope.

Therefore, in the second part of this chapter we focus on the convex hull of the binary points satisfying the s-consistency constraints (3.26). This convex hull is a set covering polytope and its valid inequalities induce inequalities valid for the DTSPMS polytope. The facial structure of set covering polytopes has been widely investigated since the pioneering work of Fulkerson [67] and Padberg [152]. Then the existing knowledge on set covering polytopes can be exploited to derive valid inequalities for the DTSPMS polytope.

This "set covering approach" provides a general framework allowing to strengthen the s-consistency in our formulation for the double TSP with multiple stacks. Additionally, we show that the study of the set covering polytope arising from the s-consistency constraints can be reduced to the study of a simpler set covering polytope. The latter simplification turns out to be advantageous in the case of the double TSP with two stacks. Indeed, it reveals a link between the corresponding DTSPMS polytope and a specific vertex cover polytope. Using this link, valid inequalities for the DTSPMS polytope in the case with two stacks can be derived by detecting specific structures in a suitable graph. In particular, we introduce two new families of inequalities valid for the DTSPMS polytope in the case with two stacks: the odd-hole inequalities and the wheel inequalities. We point out that these inequalities are not easily interpreted in terms of structures of the digraph G_n on which we have modeled the problem in Section 2.2. From this point of view, the abstraction provided by this set covering approach is a useful tool to get valid inequalities for the DTSPMS polytope that convey the s-consistency.

We summarize the organization of this chapter. In Section 4.1 we give some general results linking the DTSPMS polytope and the PATSP polytope. In Section 4.1.1 we report some known valid inequalities for the PATSP polytope. These are also valid for the DTSPMS polytope hence they define faces of the latter. In Section 4.1.2 we find the closed-form expression of the dimensions of the two polytopes and we show that every facet of the PATSP polytope gives rise to two facets of the DTSPMS polytope. In Section 4.2 we show that the DTSPMS polytope is linked to a set covering polytope obtained from the s-consistency constraints. Using known results on set covering polytopes we derive a large family of valid inequalities for the DTSPMS polytope. In the special case of the double TSP with two stacks, we show that our approach amounts to study a vertex cover polytope. Exploiting this additional property, we derive inequalities valid for the DTSPMS polytope that are associated with structures of a suitable undirected graph. Section 4.3 is a conclusion to the chapter and presents some directions for future works on the polyhedral properties of the double TSP with multiple stacks.

4.1 Focus on Routing

In this section we focus on the routing aspects of the double TSP with multiple stacks. We investigate these aspects from a polyhedral point of view. The objects of interest in

our discussion are two polytopes.

The first one is the *PATSP polytope* $PATSP_n = \text{conv}\{(x, y) \text{ satisfying (3.1)–(3.7)}\}$. Each Hamiltonian circuit H of G_n corresponds to exactly one vertex (χ^H, γ^H) of $PATSP_n$, and conversely. The second polytope is the *DTSPMS polytope* defined by:

$$DTSPMS_{n,s} = \text{conv}\{(x^1, y^1, x^2, y^2) \text{ satisfying (3.21)–(3.28)}\}.$$

The vertices of $DTSPMS_{n,s}$ are in one-to-one correspondence with the s-consistent pairs of Hamiltonian circuits of G_n. If (H_1, H_2) is such a pair and $(\chi^{H_1}, \gamma^{H_1}, \chi^{H_2}, \gamma^{H_2})$ is the corresponding vertex of $DTSPMS_{n,s}$ then we call it the *characteristic point* of the pair (H_1, H_2).

The facial description of the PATSP polytope yields a partial facial description of the DTSPMS polytope. Indeed, if $(\chi^{H_1}, \gamma^{H_1}, \chi^{H_2}, \gamma^{H_2})$ is the characteristic point of an s-consistent pair of Hamiltonian circuits, then $(\chi^{H_1}, \gamma^{H_1})$ and $(\chi^{H_2}, \gamma^{H_2})$ satisfy every inequality $ax + by \geq c$ valid for the PATSP polytope. This implies that if $ax + by \geq c$ defines a face of the PATSP polytope then $ax^1 + by^1 \geq c$ and $ax^2 + by^2 \geq c$ are satisfied by all vertices of the DTSPMS polytope, hence, by convexity, they define faces of this latter. In next section we report some inequalities valid for the DTSPMS polytope arising from the PATSP polytope.

4.1.1 Faces from the PATSP Polytope

In this section we describe classes of inequalities valid for the PATSP polytope. As pointed out in the previous section, they induce faces of the DTSPMS polytope and hence they can be used to strengthen our formulation for the double TSP with multiple stacks of infinite capacity. Here we focus on inequalities mixing arc and precedence variables, *i.e.,* of the form $ax + by \geq c$ with $a, b \neq 0$.

Two families of inequalities valid for $PATSP_n$ are the following:

$$x_{0i} \leq y_{ij} \qquad \forall \text{ distinct } i, j \in V \setminus \{0\}, \tag{4.1}$$

$$x_{i0} \leq y_{ji} \qquad \forall \text{ distinct } i, j \in V \setminus \{0\}. \tag{4.2}$$

The first (resp. second) inequality states that i is the first (resp. last) vertex visited by a Hamiltonian circuit of G_n if and only if it precedes (resp. follows) all other vertices.

Valid inequalities for the PATSP polytope can be used to strengthen formulation (3.1)–(3.7) for the ATSP. Since the same formulation can be adapted to model the ATSP with precedence constraints between locations, it is not surprising that several classes of valid inequalities have been introduced for the PATSP polytope [84, 86]. The remainder of the section is devoted to the description of inequalities for the PATSP polytope appeared in [84]. We mention that Gouveia and Pesneau [84] also test them in computational experiments to assess their quality. Here we report the inequalities presented in [84] that turned out to be more effective from a computational point of view.

Lifted Transitivity Constraints. A first polynomial size family of valid inequalities is obtained by strengthening the transitivity constraints (3.4), see *e.g.,* [84]. Namely, the inequality:

$$y_{ij} + y_{jk} + y_{ki} - x_{ij} \geq 1 \tag{4.3}$$

is valid for $PATSP_n$ for every 3-uple (i, j, k) of distinct vertices in $V \setminus \{0\}$. Indeed, if H is a Hamiltonian circuit, when $\chi^H_{ij} = 0$ for $i, j \in V \setminus \{0\}$ we have that the characteristic point of H satisfies (4.3) because it satisfies (3.4). If $\chi^H_{ij} = 1$ then $\gamma^H_{ij} = 1$ therefore $\gamma^H_{jk} = 0 = \gamma^H_{ki}$ yields a contradiction since then $i \prec_H k \prec_H j$. Hence also in this case the characteristic point of H satisfies (4.3). Inequality (4.3) is stronger than (3.4) because the right-hand-side is the same in both inequalities whereas the left-hand-side of (4.3) has a positive variable with a negative coefficient more. We call inequalities (4.3) *lifted transitivity constraints.*

The Generalized Disaggregated Desrochers and Laporte Inequalities. The next inequalities are the so-called Generalized Disaggregated Desrochers and Laporte inequalities, reported in Proposition 4.1.1. Remember that, given S a vertex set of a digraph, $A(S)$ denotes the set of arcs with their tail and their head in S.

Proposition 4.1.1 ([84])**.** *The* Generalized Disaggregated Desrochers and Laporte (GDDL) inequality

$$y_{ki} + x(A(S)) \leq y_{kj} + |S| - 1 \tag{4.4}$$

is valid for $PATSP_n$, for all distinct $i, j, k \in \{1, \ldots, n\}$ and for all $S \subseteq V \setminus \{0, k\}$ and $i, j \in S$.

Inequality (4.4) is satisfied by all vertices (x, y) of $PATSP_n$. To see this, the only non-trivial case to consider is when $x(A(S)) = |S| - 1$. Then (x, y) describes a Hamiltonian circuit of G_n containing a path visiting all vertices in S. As a consequence, $k \notin S$ precedes both i and j in this Hamiltonian circuit, whenever it precedes one of them.

When $y_{ki} = 0 = y_{kj}$, inequality (4.4) boils down to a classical subtour elimination constraint (2.6) associated with S. As explained by Gouveia and Pesneau [84] and previously noted by Gouveia and Pires [86], if we sum the inequality (4.4) corresponding to given vertices i, j, k with the inequality (4.4) in which the role of i and j is reversed, we get an equivalent form of (2.6) associated with S. Therefore constraints (2.6) are redundant with respect the GDDL inequalities (4.4).

The Simple Cut Inequalities. The last families of inequalities we describe are the Simple Cut Inequalities introduced in [84]. We illustrate the idea behind these inequalities with an example. Let us consider a Hamiltonian circuit of G_n and let $i \neq j \in V \setminus \{0\}$. If i precedes j in the Hamiltonian circuit, the latter contains a path from 0 to i not visiting j. Therefore the Hamiltonian circuit has nonempty intersection with any $(0, i)$-cut in the digraph $G_n \setminus \{j\}$. Similarly, if i precedes j, there exists a path from i to j in $G_n \setminus \{0\}$ and a path from j to 0 in $G \setminus \{i\}$. This implies Proposition 4.1.2 below.

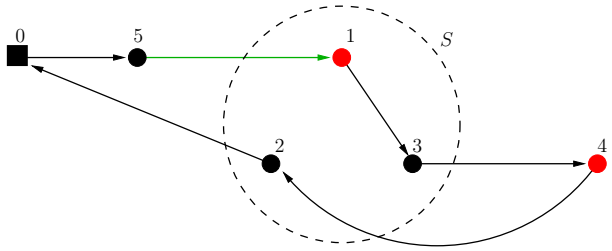


Figure 4.1: Graphical representation of inequality (4.5), with $n = 5$, $S = \{1, 2, 3\}$, $i = 1$ and $j = 4$.

Proposition 4.1.2 ([84]). *The following* Simple Cut Inequalities *are valid for* $PATSP_n$ *for all distinct* $i, j \in V \setminus \{0\}$*:*

$$y_{ij} \leq \sum_{u \in V \setminus (S \cup \{j\})} \sum_{t \in S} x_{ut} \qquad S \subseteq V \text{ such that } 0, j \notin S, i \in S, \tag{4.5}$$

$$y_{ij} \leq \sum_{u \in V \setminus (S \cup \{0\})} \sum_{t \in S} x_{ut} \qquad S \subseteq V \text{ such that } 0, i \notin S, j \in S, \tag{4.6}$$

$$y_{ij} \leq \sum_{u \in V \setminus (S \cup \{i\})} \sum_{t \in S} x_{ut} \qquad S \subseteq V \text{ such that } i, j \notin S, 0 \in S. \tag{4.7}$$

4.1.2 Links with the PATSP Polytope

In this section we derive some general polyhedral links between the DTSPMS polytope and the PATSP polytope. The main result of this section is in Proposition 4.1.7: there we prove that every facet of the PATSP polytope induces two facets of the DTSPMS polytope. Proposition 4.1.7 relies on a number of intermediate results relating the dimension of the PATSP polytope to the dimension of the DTSPMS polytope.

Remember that, in general, the dimension of the linear relaxation of a formulation only provides an upper bound for the dimension of its integer hull. In some cases, however, this upper bound is tight. For instance, it is known [88] that constraints (3.3) yield a minimal set of equalities for $\text{proj}_y(PATSP_n)$ of rank $\binom{n}{2}$. Similarly, it is known [92] that, for $n \geq 4$, the rank of the matrix of constraints (3.1)–(3.2) is $2n + 1$ and this is a minimal set of equalities for $\text{proj}_x(PATSP_n)$.

In the first contribution of this section we parallel the results cited above: namely, we show that equalities (3.1)–(3.3) yield a minimal system of inequalities for $PATSP_n$. Note that from the above-mentioned results, it follows that the rank of the matrix of equations (3.1)–(3.3) is $\binom{n}{2} + 2n + 1 = \frac{n^2+3n+2}{2}$. Let d_n indicate the dimension of $PATSP_n$. The number of variables involved in the formulation (3.1)–(3.7) being $2n^2$, we get that $d_n \leq \frac{3n^2-3n-2}{2}$. We now prove that equality actually holds. The proof is based on an exhaustive enumeration of affinely independent points belonging to the PATSP polytope, combined with an inductive argument. From now on, we say that two Hamiltonian circuits

are affinely (resp. linearly) independent whenever their characteristic points are.

Proposition 4.1.3. *If $n \geq 4$, then* $\dim(PATSP_n) = \frac{3n^2-3n-2}{2}$.

Proof. To prove the result, we define $k_n = \frac{3n^2-3n-2}{2}$ and we find k_n+1 affinely independent points of $PATSP_n$, for every $n \geq 4$. We proceed by induction. For the base case $n = 4$ we observe that $k_4 = 17$ and that the following 18 Hamiltonian circuits are affinely independent:[1]

$0,1,2,3,4,0$	$0,1,2,4,3,0$	$0,1,3,2,4,0$	$0,1,3,4,2,0$	$0,1,4,2,3,0$	$0,1,4,3,2,0$
$0,2,1,3,4,0$	$0,2,1,4,3,0$	$0,2,3,1,4,0$	$0,2,3,4,1,0$	$0,2,4,1,3,0$	$0,2,4,3,1,0$
$0,3,1,2,4,0$	$0,3,1,4,2,0$	$0,3,2,1,4,0$	$0,3,2,4,1,0$	$0,3,4,1,2,0$	$0,4,1,2,3,0$

Now, assuming that the proposition holds for $PATSP_n$ we prove that it also does for $PATSP_{n+1}$. By the inductive hypothesis, there exist $C_1, \ldots, C_{k_n+1}$ affinely independent Hamiltonian circuits of G_n. By inserting $n+1$ at the end of each C_i, we get a set of k_n+1 affinely independent Hamiltonian circuits of G_{n+1}. Since $k_{n+1} - k_n = 3n$, it suffices to complete this set with $3n$ new Hamiltonian circuits of G_n, maintaining the affine independence. The circuits are added in an iterative fashion. We indicate by $C_{(i,j)}$ a circuit that contains the arc (i,j) not belonging to any of the circuits added in the previous iterations; similarly, $C^{\star}_{(i,j)}$ indicates a circuit where i precedes j for the first time until the given iteration. Then, by construction, adding the $2n+1$ circuits below, in the order they are presented, preserves the affine independence:[2]

$C_{(n,0)} = 0,2,3,\ldots,n-1,n+1,1,n,0$

$C^{\star}_{(n+1,2)} = 0,3\ldots,n-1,n+1,1,2,n,0$

$C^{\star}_{(n+1,i)} = 0,2\ldots,i-1,i+1,\ldots,n-1,n+1,1,i,n,0$ for $i = 3,\ldots,n-2$

$C^{\star}_{(n+1,n-1)} = 0,2\ldots,n-2,n+1,1,n-1,n,0$

$C_{(n+1,i)} = 0,1,\ldots,i-1,n+1,i,\ldots,n,0$ for $i = 2,\ldots,n$

$C_{(0,n+1)} = 0,n+1,1,2,\ldots,n,0$

$C_{(2,0)} = 0,1,3,\ldots,n-1,n+1,n,2,0$

$\tilde{C}_1 = 0,1,3,\ldots,n,n+1,2,0$

Adding $\tilde{C}_1$ maintains the affine independence since every previous circuit C such that $\gamma^C_{n(n+1)} = 1$ also verifies $\chi^C_{(n+1)0} = 1$. Hence, $\tilde{C}_1$ cannot be obtained as an affine combination of the previous circuits. Finally, we add the following $n-1$ circuits:

$C_{(1,0)} = 0,2,3,\ldots,n+1,1,0$

$C_{(i,0)} = 0,i+1,\ldots,n+1,1,\ldots,i,0$ for $i = 3,\ldots,n-1$

$\tilde{C}_2 = 0,2,3,\ldots,n-1,n+1,n,1,0$

[1] These Hamiltonian circuits have been found by computing, in a computer-assisted proof, a non-singular submatrix of the matrix containing all the characteristic points of the vertices of $PATSP_4$.

[2] The set of circuits $C^{\star}_{n+1,i}$ is empty if $n = 4$.

Adding $\tilde{C}_2$ preserves the affine independence since every previous circuit C such that $\gamma^C_{(n+1)1} = 1$ also verifies $\chi^C_{(n+1)1} = 1$. The whole family of circuits above forms an affinely independent set, and this concludes the proof. □

Having a formula for the dimension of $PATSP_n$ at hand, we want to find a formula for the dimension of $DTSPMS_{n,s}$. For this purpose, we need the intermediate Lemma 4.1.4. We fix a Hamiltonian circuit, and we consider the convex hull of the Hamiltonian circuits that are s-consistent with the former. The lemma provides a formula for the dimension of this convex hull. Note that in the case of 2-consistency such a formula is not the same as in the cases of s-consistency where $s \geq 3$.

Lemma 4.1.4. *The set of Hamiltonian circuits that are s-consistent with a fixed Hamiltonian circuit of G_n has dimension d_n for $n \geq 4$ and $s \geq 3$ and dimension $d_n - 3$ for $n \geq 5$ and $s = 2$.*

Proof. Let us fix a Hamiltonian circuit of G_n. We can assume that this circuit is $C^n = 0, n, n-1, \ldots, 1, 0$, as otherwise we can relabel the vertices. Let us call $\mathcal{C}^{n,s}$ the set of Hamiltonian circuits that are s-consistent with C^n. We split the proof into two cases.

Case $s \geq 3$ The case $n = 4$ holds because the 18 Hamiltonian circuits given in the beginning of the proof of Proposition 4.1.3 are s-consistent with C^4 and affinely independent. This shows that $\dim(\mathcal{C}^{4,3}) = 17$. Therefore, $\dim(\mathcal{C}^{4,s}) = 17$ for every $s \geq 3$. Then we can apply induction, observing that the circuits constructed in the proof of Proposition 4.1.3 are s-consistent with C^{n+1} when $s \geq 3$.

Case $s = 2$ Note that in the case with two stacks and $n \geq 5$, the following three equalities are valid for the set $\mathcal{C}^{n,2}$ by Proposition 3.3.1:

$$x_{02} = y_{21} \tag{4.8}$$
$$x_{(n-1)0} = y_{n(n-1)} \tag{4.9}$$
$$x_{n1} = y_{n1} \tag{4.10}$$

It is easy to see that the equations (4.8)–(4.10) are linearly independent. In addition, they cannot be expressed as linear combination of equations (3.1)–(3.3) because y_{ij} and y_{ji} appear with the same coefficient in all these equalities, for every distinct $i, j \in \{1, \ldots, n\}$. Therefore, adding (4.8)–(4.10) to equations (3.1)–(3.3) we get $\dim(\mathcal{C}^{n,2}) \leq d_n - 3$.

Let $n \geq 5$ and let us prove by induction that $\dim(\mathcal{C}^{n,2}) = d_n - 3$. Since $d_5 - 3 = 26$, the base case $n = 5$ is proved by the following 27 Hamiltonian circuits that are 2-consistent with C^5 and affinely independent.

$0,1,2,3,5,4,0$ $0,1,2,4,3,5,0$ $0,1,3,2,4,5,0$ $0,1,3,2,5,4,0$ $0,1,3,4,2,5,0$ $0,1,3,4,5,2,0$
$0,1,3,5,2,4,0$ $0,1,4,2,3,5,0$ $0,1,4,2,5,3,0$ $0,1,4,5,2,3,0$ $0,1,5,2,3,4,0$ $0,2,1,3,4,5,0$
$0,2,1,3,5,4,0$ $0,2,1,4,3,5,0$ $0,2,1,4,5,3,0$ $0,2,1,5,3,4,0$ $0,2,3,1,4,5,0$ $0,2,3,1,5,4,0$
$0,2,3,4,1,5,0$ $0,2,3,4,5,1,0$ $0,2,3,5,1,4,0$ $0,2,5,1,3,4,0$ $0,3,4,5,1,2,0$ $0,3,5,1,2,4,0$
$0,4,1,5,2,3,0$ $0,4,5,1,2,3,0$ $0,5,1,2,3,4,0$

Assuming that the result holds for $\mathcal{C}^{n,2}$, let us prove it also for $\mathcal{C}^{n+1,2}$. By the inductive hypothesis, there exist $C_1, \ldots, C_{d_n-2}$ affinely independent Hamiltonian circuits of G_n that are 2-consistent with C^n. By inserting $n+1$ at the end of each C_i we get a set $\mathcal{C}$ of $d_n - 2$ independent Hamiltonian circuits of G_{n+1} that are 2-consistent with C^{n+1}. For later convenience, let us partition $\mathcal{C}$ in the two sets $\mathcal{C}_1 = \{C \in \mathcal{C} : \chi^C_{(n-1)(n+1)} = 1\}$ and $\mathcal{C}_2 = \mathcal{C} \setminus \mathcal{C}_1$.

Observe that if $C \in \mathcal{C}_2$, by (3.3) and (4.9) it follows that $\gamma^C_{(n-1)n} = 1$. As in the proof of Proposition 4.1.3, it suffices to complete $\mathcal{C}$ with $3n$ affinely independent circuits that are 2-consistent with C^{n+1}. For these new circuits, we use the notation from the proof of Proposition 4.1.3. By construction, the affine independence is ensured when the two circuits below are added in the following order:

$C_{(n,0)} := 0, 2, \ldots, n-1, n+1, 1, n, 0$
$C_{(n+1,n)} = 0, 1, \ldots, n-1, n+1, n, 0$

Next, we add the circuit:

$\tilde{C}_1 = 0, 2, \ldots, n-1, 1, n+1, n, 0$

Claim 4.1.5. *The set $\mathcal{C} \cup \{C_{(n,0)}, C_{(n+1,n)}, \tilde{C}_1\}$ is a set of affinely independent circuits.*

Proof. By contradiction, we may assume that $\tilde{C}_1$ is a combination of the circuits in $\mathcal{C} \cup \{C_{(n,0)}, C_{(n+1,n)}\}$. We indicate with λ^C the coefficient of the circuit C in such a combination. Since there is a direct arc from n to 0 only in $C_{(n,0)}$, $C_{(n+1,n)}$ and $\tilde{C}_1$, and a direct arc from $n+1$ to n only in $C_{(n+1,n)}$ and $\tilde{C}_1$, we get that $\lambda^{C_{(n,0)}} = 0$ and $\lambda^{C_{(n+1,n)}} = 1$. In addition $\chi^{\tilde{C}_1}_{(n-1)(n+1)} = 0$, then $\sum_{C \in \mathcal{C}_1} \lambda^C = -1$. Similarly, as $\chi^{\tilde{C}_1}_{(n+1)0} = 0$, we have also that $\sum_{C \in \mathcal{C}_1} \lambda^C + \sum_{C \in \mathcal{C}_2} \lambda^C = 0$, *i.e.,* $\sum_{C \in \mathcal{C}_2} \lambda^C = 1$. But this would imply $\gamma^{\tilde{C}_1}_{(n-1)n} = 2$ — a contradiction. □

Subsequently, we iteratively add the following $2n-2$ circuits:

$C^\star_{(n+1,2)} = 0, 3 \ldots, n-1, n+1, 1, 2, n, 0$
$C^\star_{(n+1,i)} = 0, 2 \ldots, i-1, i+1, \ldots, n-1, n+1, 1, i, n, 0$ for $i = 3, \ldots, n-2$
$C^\star_{(n+1,n-1)} = 0, 2 \ldots, n-2, n+1, 1, n-1, n, 0$
$C_{(n+1,i)} = 0, 1, \ldots, i-1, n+1, i, i+1, \ldots, n, 0$ for $i = 2, \ldots, n-1$
$C_{(0,n+1)} = 0, n+1, 1, 2, \ldots, n, 0$
$C_{(2,0)} = 0, 1, 3, \ldots, n, n+1, 2, 0$

By construction, the resulting set contains only affinely independent circuits. The next circuit we add is:

$\tilde{C}_2 = 0, 3, \ldots, n, n+1, 1, 2, 0$

The circuit $\tilde{C}_2$ is independent of the previous ones because $\gamma_{n1}^{\tilde{C}_2} = 1$ and $\chi_{n1}^{\tilde{C}_2} = 0$, whereas each circuit C added in the previous steps such that $\gamma_{n1}^{C} = 1$ belongs to $\mathcal{C}$, and also verifies $\chi_{n1}^{C} = 1$, because of (4.10). Finally, the last $n-2$ circuits we add are the following ones:

$$C_{(1,0)} = 0, 2, 3, \dots, n+1, 1, 0$$
$$C_{(i,0)} = 0, i+1, \dots, n+1, 1, 2, \dots, i, 0 \qquad \text{for } i = 3, 4, \dots, n-1$$

whose independence again follows by construction. The $3n$ circuits above are 2-consistent with C^{n+1}, and this concludes the proof. □

Now we are ready to prove that the dimension of the DTSPMS polytope is twice the dimension of the PATSP polytope.

Proposition 4.1.6. *For $n \geq 5$ and $s \geq 2$, we have* $\dim(DTSPMS_{n,s}) = 2d_n$.

Proof. Given the inclusions $DTSPMS_{n,2} \subseteq DTSPMS_{n,s} \subseteq PATSP_n \times PATSP_n$, it is enough to prove the result for $s = 2$.

For $n \geq 5$, let $P_1, \dots, P_{d_n+1}$ be affinely independent Hamiltonian circuits of G_n. From Lemma 4.1.4 there exist $D_1, \dots, D_{d_n-2}$ affinely independent Hamiltonian circuits such that $V_i = (P_1, D_i)$ are pairs of 2-consistent Hamiltonian circuits. By relabeling the vertices, we can assume that $P_1 = 0, n, \dots, 1, 0$. Under this assumption, every D_i satisfies (4.8)–(4.10). Given P_j, for some $1 < j \leq d_n + 1$, we will now construct $\tilde{D}_j$ satisfying (4.8)–(4.10) such that $V_{d_n-3+j} = (P_j, \tilde{D}_j)$ is a pair of 2-consistent Hamiltonian circuits. From the affine independence of the P_j for $j = 1, \dots, n$, it will follow that the characteristic points of the pairs $V_1, \dots, V_{2d_n-2}$, taken in the order given by the subscripts, will be affinely independent.

Roughly speaking, $\tilde{D}_j$ is obtained by perturbing $\overleftarrow{P}_j$ in such a way that equations (4.8)–(4.10) are satisfied; we exploit the observation that $(P_j, \overleftarrow{P}_j)$ is a pair of 2-consistent Hamiltonian circuits to find a perturbation such that also the new pair $(P_j, \tilde{D}_j)$ of circuits is 2-consistent.

In detail, if $2 \prec^{\leftarrow}_{P_j} n$, then $\tilde{D}_j$ is obtained from $\overleftarrow{P}_j$ by putting 2 at its beginning and n at its end. Note that $(P_j, \tilde{D}_j)$ is a solution to the double TSP with two stacks and $\tilde{D}_j$ verifies (4.8)–(4.10). Therefore, assume $n \prec^{\leftarrow}_{P_j} 2$. If also $1 \prec^{\leftarrow}_{P_j} 2$, then $\tilde{D}_j$ is obtained from $\overleftarrow{P}_j$ by moving n in its last position. Even in this case, $\tilde{D}_j$ is 2-consistent with P_j and verifies (4.8)–(4.10).

From now on, let $n \prec^{\leftarrow}_{P_j} 2 \prec^{\leftarrow}_{P_j} 1$. Suppose $\overleftarrow{P}_j = 0, X_1, n, X_2, n-1, X_3, 2, X_4, 1, X_5, 0$ where the X_i's represent sequences of vertices. Then, a circuit 2-consistent with P_j and verifying (4.8)–(4.10), is $\tilde{D}_j = 0, X_1, X_2, n-1, n, 1, X_3, 2, X_4, X_5, 0$.
Lastly, if $n-1 \prec^{\leftarrow}_{P_j} n$ or $2 \prec^{\leftarrow}_{P_j} n-1$, let $\overleftarrow{P}_j = 0, X_1, X_2, 0$ where X_2 is the part of the circuit starting at node 2. In both cases, $\tilde{D}_j = 0, X_2, X_1, 0$ is 2-consistent with P_j and satisfies (4.8)–(4.10).

To conclude the proof, consider the following three Hamiltonian circuits:[3]

$$D_1^\star = 0, 3, 2, 1, 4, \ldots, n-2, n, n-1, 0$$
$$D_2^\star = 0, 2, 1, n, n-1, 3, \ldots, n-2, 0$$
$$D_3^\star = 0, 2, n, 3, 1, 4, \ldots, n-1, 0$$

Note that the $D_k^\star$ above are well defined because $n \geq 5$. Setting $P_k^\star = \overleftarrow{D}_k^\star$, we get that the characteristic points of the pairs $V_{2d_n-2+k} = (P_k^\star, D_k^\star)$ are all independent with each other and with the previous points, because $D_k^\star$ only violates the k-th equation of (4.8)–(4.10). The characteristic points of the pairs $V_1, \ldots, V_{2d_n+1}$ form a family of $2d_n+1$ affinely independent points of $DTSPMS_{n,2}$. □

In Theorem 4.1.7 we finally prove that every facet-defining inequality of the PATSP polytope induces two facet-defining inequalities for the DTSPMS polytope. A result of this flavor can be found in [26]. More precisely, it is proved there that every facet of $\text{proj}_x(PATSP_n)$ induces two facets of $\text{proj}_{(x^1,x^2)}(DTSPMS_{n,s})$. Nevertheless, their result cannot be applied to derive facet-defining inequalities involving precedence variables for our formulation.

Theorem 4.1.7. *For $n \geq 5$ and $s \geq 2$, if $ax + by \geq c$ defines a facet of $PATSP_n$, then $ax^1 + by^1 \geq c$ and $ax^2 + by^2 \geq c$ define two facets of $DTSPMS_{n,s}$.*

Proof. Let us consider an inequality $ax + by \geq c$ valid for the PATSP polytope, such that the set $\mathcal{F}' = \{(x, y) \in PATSP_n \colon ax + by = c\}$ is a facet of $PATSP_n$. Without loss of generality, let us show that $\mathcal{F} = \{(x^1, y^1, x^2, y^2) \in DTSPMS_{n,s} \colon ax^1 + by^1 = c\}$ is a facet of $DTSPMS_{n,s}$.

Note that $\mathcal{F}$ is a face of $DTSPMS_{n,s}$: indeed, $ax^1 + by^1 \geq c$ is valid for $DTSPMS_{n,s}$ because $DTSPMS_{n,s} \subseteq PATSP_n \times PATSP_n$ and $ax + by \geq c$ is valid for $PATSP_n$. To prove that $\mathcal{F}$ is a facet of $DTSPMS_{n,s}$ we will exhibit $2d_n$ affinely independent points in $\mathcal{F}$. Since $DTSPMS_{n,2} \subseteq DTSPMS_{n,s}$ for every $s \geq 2$, it is enough to prove the existence of these affinely independent points for $s = 2$.

By hypothesis there exist $P_1, \ldots, P_{d_n}$ affinely independent Hamiltonian circuits belonging to $\mathcal{F}'$. In addition, it is not restrictive to assume $P_1 = 0, n, \ldots, 1, 0$. Under this assumption, and repeating the reasoning used in the proof of Proposition 4.1.6, we can construct Hamiltonian circuits $D_1, \ldots, D_{d_n-2}$ and $\tilde{D}_2, \ldots, \tilde{D}_{d_n}$ such that the $2d_n - 3$ pairs of circuits

$$(P_1, D_1), \ldots, (P_1, D_{d_n-2}), (P_2, \tilde{D}_2), \ldots, (P_{d_n}, \tilde{D}_{d_n}) \tag{4.11}$$

are 2-consistent, and their characteristic points form an affine independent set of $\mathcal{F}$ and verify:

$$x_{02}^2 = y_{21}^2 \tag{4.12}$$
$$x_{(n-1)0}^2 = y_{n(n-1)}^2 \tag{4.13}$$
$$x_{n1}^2 = y_{n1}^2 \tag{4.14}$$

[3] In the case $n = 5$, we set $D_1^\star = 0, 3, 2, 1, 5, 4, 0$

Now, assume that $ax^1 + by^1 \geq c$ coincides with the inequality $x^1_{01} \geq 0$. In this case, the Theorem holds, since we can complete the set in (4.11) to an affine basis of $\mathcal{F}$ by adding the pairs of Hamiltonian circuits $(P^\star_k, D^\star_k)$ as constructed in the proof of Proposition 4.1.6. Indeed, their characteristic points satisfy $x^P_{01} = 0$ and the affine independence of the resulting set is seen as in Proposition 4.1.6. The same also applies if $ax^1+by^1 \geq c$ coincides with $x^1_{(n-1)0} \geq 0$ or with $x^1_{n0} \geq 0$.

Therefore, assume that $ax^1 + by^1 \geq c$ does not coincide with any of the inequalities $x^1_{01} \geq 0$, $x^1_{(n-1)0} \geq 0$, $x^1_{n0} \geq 0$.

By hypothesis, there exists a circuit $P^\star_1 \in \mathcal{F}'$ such that $\chi^{P^\star_1}_{01} = 1$. In other words, $P^\star_1 = 0, 1, X_1, 2, X_2, 0$ for some sequences of vertices X_1 and X_2. We first assume that $X_2 \neq \{n\}$. Then if $X_2 \neq \emptyset$, we define $D^\star_1 = 0, \overleftarrow{X'_2}, 2, \overleftarrow{X'_1}, 1, n, 0$, where X'_1 and X'_2 are respectively obtained from X_1 and X_2 by removing n. If $X_2 = \emptyset$, then $P^\star_1 = 0, 1, X_3, n, X_4, 2, 0$ for some sequences X_3 and X_4. In this case, when $X_4 = \emptyset$ we define $D^\star_1 = 0, r, 2, \overleftarrow{X'_3}, n, 1, 0$, where X'_3 is obtained from X_3 by removing its last vertex r. Instead, if $X_4 \neq \emptyset$, we set $D^\star_1 = 0, t, 2, \overleftarrow{X'_4}, \overleftarrow{X_3}, n, 1, 0$, where X'_4 is obtained from X_4 by removing its first vertex t. In each case, $(P^\star_1, D^\star_1) \in \mathcal{F}$ (because it is a pair of 2-consistent Hamiltonian circuits) and its characteristic point violates (4.12) (while satisfying (4.13) and (4.14)). As a consequence, it cannot be in the affine subspace generated by the points in (4.11).

Similarly, there exists a Hamiltonian circuit $P^\star_2 \in \mathcal{F}'$ such that $\chi^{P^\star_2}_{(n-1)0} = 1$ (possibly $P^\star_2 = P^\star_1$). We distinguish the following two cases, where X_5, X_6, X_7 denote suitable sequences of vertices:

(1) $P^\star_2 = 0, X_5, n, X_6, 1, X_7, n-1, 0$

(2) $P^\star_2 = 0, X_5, 1, X_6, n, X_7, n-1, 0$

We define $D^\star_2 = 0, 1, n, n-1, \overleftarrow{X_7}, \overleftarrow{X_6}, \overleftarrow{X_5}, 0$ in case (1), whereas in case (2), $D^\star_2 = 0, n, 1, n-1, \overleftarrow{X_7}, \overleftarrow{X_6}, \overleftarrow{X_5}, 0$. In both cases, $(P^\star_2, D^\star_2) \in \mathcal{F}$ and the affine independence of its characteristic point with the previous ones is given by the fact that it violates only (4.13) while satisfying (4.12) and (4.14).

Finally, let $P^\star_3 \in \mathcal{F}'$ be a Hamiltonian circuit such that $\chi^{P^\star_3}_{n0} = 1$ ($P^\star_3$ may coincide with $P^\star_1$). Since $P^\star_3 = 0, X_8, 1, X_9, n, 0$ for some sequence of vertices X_8 and X_9, defining $D^\star_3 = 0, n, \overleftarrow{X_9}, \overleftarrow{X_8}, 1, 0$ gives that $(P^\star_3, D^\star_3) \in \mathcal{F}$. In addition, its characteristic point cannot be obtained as an affine combination of the previous ones, as it violates equations (4.12)–(4.14).

Completing the set of circuits in (4.11) with the pairs $(P^\star_i, D^\star_i)$ yields a set of $2d_n$ affinely independent pairs of Hamiltonian circuits belonging to $\mathcal{F}$ and this concludes the proof for the case in which $X_2 \neq \{n\}$.

Finally, let us assume that $X_2 = \{n\}$. We define $D^\star_1 = 0, 2, n-1, n, \overleftarrow{X'_1}, 1, 0$ where X'_1 is obtained from X_1 by removing $n-1$. Note that, since $n \geq 5$, X'_1 is not empty. Now, $(P^\star_1, D^\star_1) \in \mathcal{F}$ and its characteristic vector violates (4.14), while satisfying (4.12) and (4.13). We construct the points $(P^\star_2, D^\star_2)$ and $(P^\star_3, D^\star_3)$ as done above. The same argument

used for the case $X_2 \neq \{n\}$ shows that, also in the case $X_2 = \{n\}$, completing the set of circuits in (4.11) with the pairs $(P_i^\star, D_i^\star)$ yields a set of $2d_n$ affinely independent pairs of Hamiltonian circuits belonging to $\mathcal{F}$. This concludes the proof. □

Remark 4.1.8. Fiorini *et al.* [60] have recently shown that any extended formulation for the ATSP has $2^{\Omega(\sqrt{n})}$ facets. Hence Theorem 4.1.7 characterizes implicitly a super-exponential number of facets of the DTSPMS polytope.

The problem of determining whether the inequalities presented in Section 4.1.1 induce facets for the PATSP polytope remains open. Using Theorem 4.1.7, a result of this kind would immediately provide facets also for the DTSPMS polytope.

4.2 Focus on Consistency

This section addresses the study of the s-consistency from a polyhedral perspective. To this purpose we consider the relaxation of our formulation given by the s-consistency constraints (3.26). Exploiting the structure of these constraints we observe that the considered relaxation relates to a specific set covering polytope. The latter result is then used to introduce new faces of the DTSPMS polytope, borrowed from the literature on set covering polytopes.

Our approach is motivated by the following arguments. In our formulation for the double TSP with multiple stacks of infinite capacity, only constraints (3.26) enforce the s-consistency property. The new faces of the DTSPMS polytope obtained in this section correspond to strengthening inequalities for our formulation that convey additional informations on the s-consistency. Note that this is not the case for the inequalities presented in Section 4.1.1, since they are valid also for a formulation for the ATSP, in which constraints (3.26) are not present. Actually, Theorem 4.1.7 yields a super-polynomial number of facets of the DTSPMS polytope arising from the PATSP polytope yet they are not sufficient to fully characterize the convex hull, as we now explain.

Example 4.2.1. Consider the following instance of the double TSP with five items and two stacks of infinite capacity. We consider two *symmetrical* cost vectors c^1 and c^2, that is $c^1_{ij} = c^1_{ji}$ and $c^2_{ij} = c^2_{ji}$ for all $i \neq j \in V$. We choose vector c^1 such that the TSP on the arc-weighted digraph (G_5, c^1) admits $P^\star = 0, 1, 2, 3, 4, 5, 0$ as optimal solution. We also choose c^2 such that $D^\star = 0, 1, 2, 5, 4, 3, 0$ is a least cost Hamiltonian circuit of the arc-weighted digraph (G_5, c^2). In the following, let $\mathcal{L}$ be the linear relaxation of our formulation for $DTSPMS_{5,2}$. Neither $(P^\star, D^\star)$ nor $(P^\star, \overleftarrow{D^\star})$ is 2-consistent, because in both pairs three vertices appear in the same order in both Hamiltonian circuits yielding the pair. Additionally, the point $S^\star = (\chi^{P^\star}, \gamma^{P^\star}, \frac{1}{2}\chi^{D^\star} + \frac{1}{2}\chi^{\overleftarrow{D^\star}}, \frac{1}{2}\gamma^{D^\star} + \frac{1}{2}\gamma^{\overleftarrow{D^\star}})$ does not belong to $DTSPMS_{5,2}$. Indeed, the inequality $y^1_{12} + y^1_{25} + y^2_{12} + x^2_{21} + x^2_{25} \leq 3$ is valid for $DTSPMS_{5,2}$: a pair of Hamiltonian circuits could violate it only by verifying $y^1_{12} + y^1_{25} = 2$ and $y^2_{12} + x^2_{25} = 2$ because $x^2_{21} + x^2_{25} \leq 1$; but in this case the pair is not 2-consistent since 1, 2 and 5 appear in the same order in both Hamiltonian circuits. The point $S^\star$

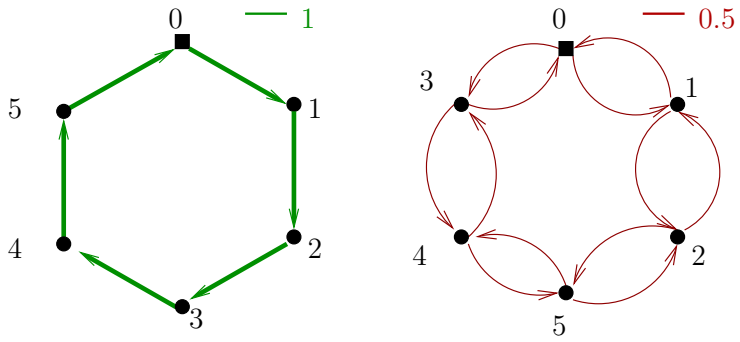


Figure 4.2: The point $S^\star$. The left-hand circuit corresponds to $(\chi^{P^\star}, \gamma^{P^\star})$. The right-hand circuit corresponds to $(\frac{1}{2}\chi^{D^\star} + \frac{1}{2}\chi^{\overleftarrow{D^\star}}, \frac{1}{2}\gamma^{D^\star} + \frac{1}{2}\gamma^{\overleftarrow{D^\star}})$.

violates the previous inequality, hence it cannot be in $DTSPMS_{5,2}$. However, $S^\star$ belongs to $\mathcal{L}$. To prove it, let us first consider constraints (3.26). They are satisfied by $S^\star$ since $\frac{1}{2}\gamma_{ij}^{D^\star} + \frac{1}{2}\gamma_{ij}^{\overleftarrow{D^\star}} = \frac{1}{2}$. All other constraints of our formulation must be satisfied by $S^\star$, since both $(\chi^{P^\star}, \gamma^{P^\star})$ and $(\frac{1}{2}\chi^{D^\star} + \frac{1}{2}\chi^{\overleftarrow{D^\star}}, \frac{1}{2}\gamma^{D^\star} + \frac{1}{2}\gamma^{\overleftarrow{D^\star}})$ belong to $PATSP_5$. The same argument proves that no inequality from Section 4.1.1 cuts off $S^\star$. We mention that $S^\star$ is actually an extreme point of $\mathcal{L} \cap (PATSP_5 \times PATSP_5)$ and also an optimal solution to the problem of minimizing the function $c^1x^1 + c^2x^2$ over this polytope.

The same argument as above holds, as long as $P^\star$ and $D^\star$ are optimal solutions for the symmetric TSP's on G_n and symmetrical cost vectors c^1 and c^2, respectively. Thus, for symmetrical cost vectors, the lower bound yielded by the linear relaxation of our formulation cannot be better than the value obtained by independently optimizing the pickup and delivery circuits, *i.e.,* by independently solving two traveling salesman problems.

In view of tackling the double TSP with multiple stacks by means of a polyhedral approach, the situation illustrated in Example 4.2.1 is undesirable: not only the linear relaxation of our formulation strictly contains the DTSPMS polytope but, in general, it also yields weak lower bounds. Moreover, this situation is not solved by the inequalities presented in Chapter 4.1.1. We need to strengthen the consistency requirement in our formulation. To this end we define the following polytope, based on the s-consistency constraints of our formulation for the double TSP with multiple stacks of infinite capacity:

$$SC_{n,s} = \text{conv}\{(y^1, y^2) \in \{0,1\}^{n(n-1)} \times \{0,1\}^{n(n-1)} \colon (3.26) \text{ are satisfied}\}.$$

The polytopes $SC_{n,s}$ and $DTSPMS_{n,s}$ are related by the following:

$$proj_{(y^1,y^2)}(DTSPMS_{n,s}) \subseteq SC_{n,s}.$$

Thus every inequality valid for $SC_{n,s}$ is also valid for $DTSPMS_{n,s}$.

We recall that, given a matrix $A \in \{0,1\}^{\ell \times m}$, the associated set covering polytope is $SC_A = \text{conv}\{z \in \{0,1\}^m \colon Az \geq \mathbf{1}\}$. Thus, $SC_{n,s}$ is the set covering polytope associated

with the binary matrix corresponding to the left-hand-side coefficients of constraints (3.26), restricted to the precedence variables. Set covering polytopes have been intensively studied — see for instance [24]. The study of set covering polytopes can be transferred to a graph theoretical setting, as proposed by Sassano [172]. We report his construction since it will be useful later.

Let $SC_A = \text{conv}\{z \in \{0,1\}^m \colon Az \geq \mathbf{1}\}$ be a set covering polytope, where $A \in \{0,1\}^{\ell \times m}$. We define the bipartite graph $\mathcal{B} = (\mathcal{U}, \mathcal{V}; \mathcal{E})$ as follows:

- $\mathcal{U} = \{u_1, \ldots, u_\ell\}$ (one vertex for each row of A),
- $\mathcal{V} = \{v_1, \ldots, v_m\}$ (one vertex for each column of A),
- $\{u_i, v_j\} \in \mathcal{E}$ if and only if $A_{ij} = 1$.

We recall that $C \subseteq \mathcal{V}$ covers $\mathcal{U}$ if the neighborhood of C coincides with $\mathcal{U}$. We call such a C a *cover of* $\mathcal{B}$. For C a cover of $\mathcal{B}$, its incidence vector $z^C \in \{0,1\}^{|\mathcal{V}|}$ is defined by $z^C_j = 1$ if and only if $j \in C$, for all $j \in \mathcal{V}$. Then SC_A coincides with the polytope $Q(\mathcal{B}) = \text{conv}\{z^C \in \{0,1\}^{|\mathcal{V}|} \colon C \text{ covers } \mathcal{U}\}$, see [172]. We now give an example of set covering polytope as well as the corresponding bipartite graph.

Example 4.2.2. Let us consider the set covering polytope associated with the following constraints, expressed in the variables y^T_{ij} for $i \neq j \in \{1, \ldots, 5\}$ and $T = 1, 2$:

$$y^1_{12} + y^1_{23} + y^2_{12} + y^2_{23} \geq 1 \tag{4.15}$$
$$y^1_{23} + y^1_{34} + y^2_{23} + y^2_{34} \geq 1 \tag{4.16}$$
$$y^1_{34} + y^1_{45} + y^2_{34} + y^2_{45} \geq 1 \tag{4.17}$$
$$y^1_{45} + y^1_{51} + y^2_{45} + y^2_{51} \geq 1 \tag{4.18}$$
$$y^1_{51} + y^1_{12} + y^2_{51} + y^2_{12} \geq 1 \tag{4.19}$$

This set covering polytope is obtained by considering a subset of s-consistency constraints (3.26) for $n = 5$. The associated bipartite graph $\mathcal{B}$ is depicted in Figure 4.3, where we omitted the isolated vertices corresponding to the variables with a zero coefficient. A solution to system (4.15)–(4.19) corresponds, in this graph, to a subset of white vertices covering all blue vertices. From Figure 4.3 it is apparent that at most one vertex between y^1_{12} and y^2_{12} is needed in a cover of $\mathcal{B}$, because they have the same neighborhood. In fact, this remains true for every pair of variables of the type y^1_{ij} and y^2_{ij} appearing in the previous system. This property of s-consistency constraints will be used in next section.

4.2.1 The Restricted Set Covering Polytope

In this section we analyze the specific structure of $SC_{n,s}$. From basic properties of set covering polytope — see Proposition 1.5.9 — we get that $SC_{n,s}$ is a full dimensional, nonempty polytope for every $n \geq 3$ and $s \geq 2$. In this case, the bound inequalities $y^T_{ij} \geq 0$ and $y^T_{ij} \leq 1$ define facets of $SC_{n,s}$ for all distinct $i, j \in \{1, \ldots, n\}$ and $T = 1, 2$. Facets

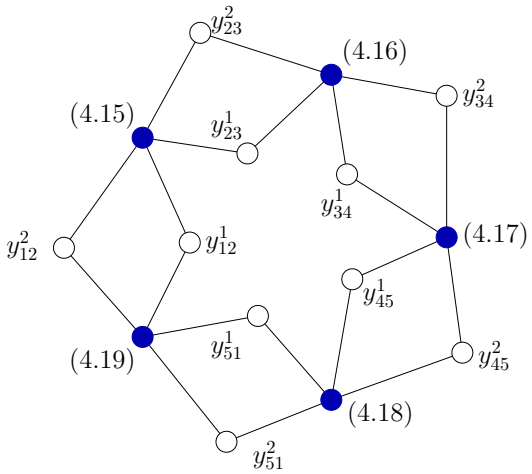


Figure 4.3: The bipartite graph corresponding to the set covering polytope described in Example 4.2.2.

of this type are called *trivial*. Conversely, every facet-defining inequality other than the bound inequalities is called *non-trivial*.

We observe that for $s \geq 2$, the variables y^1_{ij} and y^2_{ij} have the same coefficient in constraints (3.26), for all $i \neq j \in V \setminus \{0\}$. Using this property, we prove that all facets of $SC_{n,s}$ can be obtained by studying the following polytope, hereafter called *restricted set covering polytope*:

$$RSC_{n,s} = \text{conv}\{\ y \in \{0,1\}^{n(n-1)}: \sum_{i=1}^{s} y_{v_i v_{i+1}} \geq 1 \text{ for all distinct } v_1, \ldots, v_{s+1} \in V \setminus \{0\}\}.$$

Indeed, we can prove the following result.

Proposition 4.2.3. *Every non-trivial facet-defining inequality of $SC_{n,s}$ is of the form $ay^1 + ay^2 \geq b$, where $ay \geq b$ is a non-trivial facet-defining inequality of $RSC_{n,s}$.*

Before proving Proposition 4.2.3, we point out that the polyhedral study of $RSC_{n,s}$ can lead to new faces of $DTSPMS_{n,s}$. Indeed, Proposition 4.2.3 asserts that the linear description of the set covering polytope $RSC_{n,s}$ immediately gives the description of $SC_{n,s}$ and, as we already observed, $\text{proj}_{(y^1,y^2)}(DTSPMS_{n,s}) \subseteq SC_{n,s}$. The process described above is represented schematically in Figure 4.4

The remainder of the section is devoted to the proof of Proposition 4.2.3. It uses some intermediate results. In the first one we show how the vertices of $RSC_{n,s}$ are connected to the ones of $SC_{n,s}$. In our proofs we often implicitly use the fact that a binary point of a binary polytope is one of its vertices.

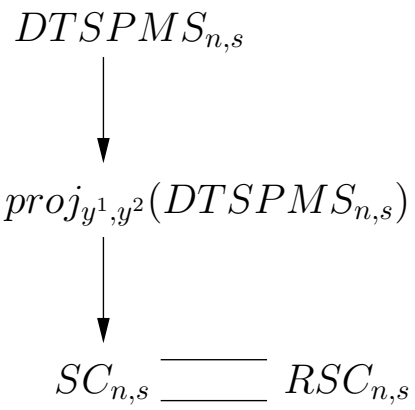


Figure 4.4: Passing from $RSC_{n,s}$ to $DTSPMS_{n,s}$. An arrow indicates that the tail is contained in the head. The double line at the bottom indicates a polyhedral equivalence.

Lemma 4.2.4. *For $y = (y^1, y^2) \in \mathbb{R}^{n(n-1)} \times \mathbb{R}^{n(n-1)}$, define $f(y) \in \mathbb{R}^{n(n-1)}$ by $f(y)_{ij} = \max\{y^1_{ij}, y^2_{ij}\}$, for all distinct $i, j \in \{1, \dots, n\}$. Then,*

$$RSC_{n,s} = \text{conv}\{f(y) \in \mathbb{R}^{n(n-1)} \colon y \text{ is a vertex of } SC_{n,s}\}.$$

Proof. Let $P = \text{conv}\{f(y) \in \mathbb{R}^{n(n-1)} \colon y \text{ is a vertex of } SC_{n,s}\}$. To show $P \subseteq RSC_{n,s}$, let $\bar{v}$ be a vertex of P. By construction, $\bar{v} = f(\bar{y})$ for some vertex $\bar{y}$ of $SC_{n,s}$. Since $\bar{y}$ is binary, so is $\bar{v}$. In addition, $\bar{v}_{j_i j_{i+1}} = 0$ if and only if $\bar{y}^1_{j_i j_{i+1}} = \bar{y}^2_{j_i j_{i+1}} = 0$. The vector $\bar{v}$ being binary, $\sum_{i=1}^{s} \bar{v}_{j_i j_{i+1}} < 1$ if and only if $\bar{v}_{j_i j_{i+1}} = 0$ for all $i = 1, \dots, s$. But this can happen only if $\sum_{i=1}^{s} (\bar{y}^1_{j_i j_{i+1}} + \bar{y}^2_{j_i j_{i+1}}) = 0$, which is impossible by $\bar{y} \in SC_{n,s}$ and (3.26). Hence, $\bar{v} \in RSC_{n,s}$. As this holds for every vertex $\bar{v}$ of P, convexity implies $P \subseteq RSC_{n,s}$.

We prove now that $RSC_{n,s} \subseteq P$. Given a vertex $\bar{v}$ of $RSC_{n,s}$, we define $\bar{y} = (\bar{y}^1, \bar{y}^2) \in \{0,1\}^{n(n-1)} \times \{0,1\}^{n(n-1)}$ as follows:

$$\begin{aligned} \bar{y}^1_{j_i j_{i+1}} = \bar{y}^2_{j_i j_{i+1}} &= 1 \quad \text{if } \bar{v}_{j_i j_{i+1}} = 1, \\ \bar{y}^1_{j_i j_{i+1}} = \bar{y}^2_{j_i j_{i+1}} &= 0 \quad \text{otherwise.} \end{aligned}$$

For distinct $j_1, \dots, j_{s+1}$, we have $\sum_{i=1}^{s} (\bar{y}^1_{j_i j_{i+1}} + \bar{y}^2_{j_i j_{i+1}}) = 0$ if and only if $\bar{v}_{j_1 j_2} = \bar{v}_{j_2 j_3} = \cdots = \bar{v}_{j_s j_{s+1}} = 0$. The latter is impossible since $\bar{v} \in RSC_{n,s}$. Hence, since $\bar{y}$ is binary, it satisfies (3.26). Thus $\bar{y}$ is a vertex of $SC_{n,s}$. By construction, we have $\bar{v} = f(\bar{y})$, therefore $\bar{v}$ is a vertex of P. This holds for every vertex $\bar{v}$ of $RSC_{n,s}$, hence $RSC_{n,s} \subseteq P$ by convexity. □

We can now show that the linear description of $SC_{n,s}$ can be deduced from the one of $RSC_{n,s}$.

Proof (of Proposition 4.2.3). Recall that from Proposition 1.5.9 we have the following properties:

(i) $SC_{n,s}$ is full dimensional.

(ii) Inequalities $y_{ij}^T \leq 1$ define facets of $SC_{n,s}$ for all distinct $1 \leq i, j \leq n$ and $T = 1, 2$.

(iii) If $a^1y^1 + a^2y^2 \geq b$ is non-trivial and defines a facet of $SC_{n,s}$, then $b > 0$ and $a_{ij}^T \geq 0$ for all distinct $1 \leq i, j \leq n$ and $T = 1, 2$.

We first prove that all facets of $RSC_{n,s}$ define facets of $SC_{n,s}$.

Claim 4.2.5. *If $ay \geq b$ is a non-trivial facet-defining inequality of $RSC_{n,s}$, then $ay^1 + ay^2 \geq b$ is a facet-defining inequality of $SC_{n,s}$.*

Proof. We first prove that $ay^1 + ay^2 \geq b$ is valid for $SC_{n,s}$. Let $\gamma = (\gamma^1, \gamma^2)$ be a vertex of $SC_{n,s}$ and suppose that $a\gamma^1 + a\gamma^2 < b$. By Lemma 4.2.4, $f(\gamma)$ is a vertex of $RSC_{n,s}$. From $\gamma \geq 0$, we get $f(\gamma)_{ij} \leq \gamma_{ij}^1 + \gamma_{ij}^2$ for all distinct $1 \leq i, j \leq n$. Since, by (iii), $a_{ij} \geq 0$, we get $af(\gamma) \leq a\gamma^1 + a\gamma^2 < b$, contradicting the validity of $ay \geq b$ for $RSC_{n,s}$.

We now prove that $ay^1 + ay^2 \geq b$ defines a facet of $SC_{n,s}$. Let F' denote the facet of $RSC_{n,s}$ defined by $ay \geq b$ and $\{\xi^1, \ldots, \xi^{n(n-1)}\}$ be an affine basis of F'. Since $b > 0$ these vectors are linearly independent. Thus the $2n(n-1)$ vectors $\{(\xi^\ell, \mathbf{0}), (\mathbf{0}, \xi^\ell)\}_{\ell=1,\ldots,n(n-1)}$ are linearly independent points of $SC_{n,s}$, satisfying $ay^1 + ay^2 \geq b$ with equality. ■

We now show that non-trivial facet-defining inequalities of $SC_{n,s}$ have a symmetric structure:

Claim 4.2.6. *Let $a^1y^1 + a^2y^2 \geq b$ be a non-trivial facet-defining inequality of $SC_{n,s}$. Then $a^1 = a^2$.*

Proof. Let us fix $i, j \in \{1, \ldots, n\}$ with $i \neq j$ and let us write for convenience the vectors $\gamma \in \mathbb{R}^{2n(n-1)}$ as $(\bar{\gamma}, \gamma_{ij}^1, \gamma_{ij}^2)$. By contradiction, we suppose that $a_{ij}^1 > a_{ij}^2$. By (iii), we get $a_{ij}^1 > 0$. If $(\bar{\gamma}, 1, 1)$ is a vertex of $SC_{n,s}$, then so are $(\bar{\gamma}, 1, 0)$ and $(\bar{\gamma}, 0, 1)$, since, in each of constraints (3.26), the coefficients of y_{ij}^1 and y_{ij}^2 are the same.

Let $F = SC_{n,s} \cap \{a^1y^1 + a^2y^2 = b\}$ be the facet defined by the given inequality and B a basis of F. It is not restrictive to assume B is composed of vertices of $SC_{n,s}$. Then, no element of B has the form $(\bar{\gamma}, 1, 1)$, as otherwise, by $\bar{a}\bar{\gamma} + a_{ij}^1 + a_{ij}^2 = b$ and $a_{ij}^1 > 0$, we would get that $(\bar{\gamma}, 0, 1)$ violates the given inequality. Given that F arises from a non-trivial facet-defining inequality of $SC_{n,s}$, there exists $(\bar{\gamma}, 1, 0) \in B$ as otherwise, $F \subseteq SC_{n,s} \cap \{y_{ij}^1 = 0\}$. This implies that $(\bar{\gamma}, 0, 1)$ violates the facet-defining inequality. We deduce that $a_{ij}^1 \leq a_{ij}^2$. Symmetrically, $a_{ij}^2 \leq a_{ij}^1$ and the desired equality follows. ■

We finally prove that all the facets of $RSC_{n,s}$ can be obtained from those of $SC_{n,s}$.

Claim 4.2.7. *If $ay^1 + ay^2 \geq b$ is a non-trivial facet-defining inequality of $SC_{n,s}$, then $ay \geq b$ is a non-trivial facet-defining inequality of $RSC_{n,s}$.*

Proof. The point $(\gamma, \mathbf{0})$ is a vertex of $SC_{n,s}$ whenever γ is a vertex of $RSC_{n,s}$. Thus the validity of $ay \geq b$ for $RSC_{n,s}$ follows from the validity of $ay^1 + ay^2 \geq b$ for $SC_{n,s}$.

Now, let us suppose, by contradiction, that $ay \geq b$ does not define a facet of $RSC_{n,s}$. Then there exists an integer $f \geq 2$ such that $a = \sum_{i=1}^{f} \lambda_i a^i$ and $b \leq \sum_{i=1}^{f} \lambda_i b^i$, where $\lambda_i > 0$ and $a^i y \geq b^i$ is a facet of $RSC_{n,s}$ for every $0 \leq i \leq f$. Thus, the inequalities $a^i y^1 + a^i y^2 \geq b^i$ are valid for $SC_{n,s}$. However, $(a, a) = \sum_{i=1}^{f} \lambda_i (a^i, a^i)$, contradicting the fact that $ay^1 + ay^2 \geq b$ defines a facet of $SC_{n,s}$. ■

□

We stress that a polyhedral study of $RSC_{n,s}$ can be useful to find faces of the DTSPMS polytope. Such a study is greatly simplified by the observation that $RSC_{n,s}$ is a set covering polytope. Indeed, set covering polytopes are pervasive in combinatorial optimization, see p. 54 of [40] and the references therein. Consequently, the facial structure of set covering polytopes has been investigated by several authors *e.g.,* [67, 152, 172, 43, 143, 39, 56, 122, 175, 142]. Moreover, much of this previous literature focused on classes of inequalities valid for set covering polytopes defined by well-structured matrices. In our case, this latter approach is very useful, because the matrix defining $RSC_{n,s}$ exhibits structures associated with known inequalities. This is the subject of next section.

4.2.2 Faces from the Restricted Set Covering Polytope

In this section, we introduce a general class of faces of the DTSPMS polytope. In fact, it arises directly from a family of inequalities valid for specific set covering polytopes introduced by Letchford [126]. We need the following definition. Given $A \in \{0,1\}^{\ell \times m}$ and $1 \leq i \leq \ell$, the *support* of the i-th row of A is $supp(i) = \{j \in \{1, \ldots, m\} \colon A_{ij} = 1\}$. Then we have the following theorem.

Theorem 4.2.8 ([126])**.** *Let $A \in \{0,1\}^{\ell \times m}$ and let p and q be two integers such that $2 \leq q < p \leq \ell$ and q does not divide p. Let $N_0, \ldots, N_{p-1} \subseteq \{1, \ldots, m\}$ such that there exist row indices i_k for $k = 0, \ldots, p-1$ with $supp(i_k) \subseteq N_k \cup N_{k+1} \cup \cdots \cup N_{k+q-1}$ (subscript indices are modulo p). Then the inequality*

$$\sum_{k=0}^{p-1} z(N_k) \geq \left\lceil \frac{p}{q} \right\rceil \tag{4.20}$$

is valid for $SC_A = \text{conv}\{z \in \{0,1\}^m \colon Az \geq \mathbf{1}\}$.

Inequality (4.20) is called *generalized cycle inequality*. We point out that Letchford also proves that the generalized cycle inequalities belong to a larger family of inequalities valid for SC_A which can be separated in polynomial time [126]. However, this result is based on the equivalence of separation and optimization [89] and it seems to be not easily adaptable to obtain fast separation routines. Theorem 4.2.8 provides a generalization of families of inequalities independently found in previous works on set covering polytopes. Namely, for

$q = 2$ inequality (4.20) was already known as *aggregated cycle inequality* [25]. When the N_k's are pairwise disjoint and all of cardinality 1, inequality (4.20) is a *q-rose inequality of order p* [172].

Remark 4.2.9. The q-rose inequalities of order p can be seen also in the graph theoretical setting presented in Section 4.2. Let p and q be two integers with $2 \leq q < p$. A *q-rose of order p* is a bipartite graph $\mathcal{B} = (\mathcal{U}, \mathcal{V}; \mathcal{E})$ with the following properties:

- $\mathcal{V} = \{v_1, \ldots, v_p\}$;
- $\mathcal{U} = \{u_1, \ldots, u_p\}$ such that $N(u_i) = \mathcal{V}_i$, where $\mathcal{V}_i = \{v_i, \ldots, v_{i+q-1}\}$ (subscripts taken modulo p);

When $\mathcal{B}$ is a q-rose of order p, a cover of $\mathcal{B}$ has cardinality at least $\left\lceil \frac{p}{q} \right\rceil$, see [172]. As a consequence, the inequality (which is a q-rose inequality of order p)

$$z(\mathcal{V}) \geq \left\lceil \frac{p}{q} \right\rceil$$

is valid for $Q(\mathcal{B})$. Actually, it defines a facet of $Q(\mathcal{B})$ if and only if q does not divide p, as shown by Sassano [172].

Inequalities Associated with Circuits of G_n. Using Theorem 4.2.8, we can easily construct s-rose inequalities of order at least $s+1$ valid for $RSC_{n,s}$ from the matrix defining this set covering polytope.

Proposition 4.2.10. *Let C be a circuit of G_n with $|C| \geq s + 1$. Then*

$$y(C) \geq \left\lceil \frac{|C|}{s} \right\rceil \tag{4.21}$$

is a s-rose inequality valid for $RSC_{n,s}$.

Proof. Let $C = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{d-1}, v_d), (v_d, v_1)\}$ be a circuit of G_n. By our assumption, $d \geq s + 1$. Consider constraints $y_{v_i v_{i+1}} + y_{v_{i+1} v_{i+2}} + \cdots + y_{v_{i+s-1} v_{i+s}} \geq 1$, for every $i = 1, \ldots, d$ (indices taken modulo d). These constraints yield a subsystem of the system defining $RSC_{n,s}$. Moreover, (4.21) is exactly (4.20) by taking N_k as the singleton containing the column index corresponding to variable $y_{v_k v_{k+1}}$, for $1 \leq k \leq d$ (indices taken modulo $d+1$). Hence (4.21) is valid for $RSC_{n,s}$. By the definition of s-rose inequality, we have the desired result. □

The proof of Proposition 4.2.10 also shows that inequalities (4.21) are interesting only when s does not divide $|C|$. This corollary follows from Claim 4.2.5 and Proposition 4.2.10.

Corollary 4.2.11. *Let C be a circuit of G_n with $|C| \geq s+1$. Then*

$$y^1(C) + y^2(C) \geq \left\lceil \frac{|C|}{s} \right\rceil \tag{4.22}$$

is valid for $DTSPMS_{n,s}$.

Remark that the point $S^\star$ described in Example 4.2.1 is cut off by the inequality $\sum_{T=1,2}(y^T_{54} + y^T_{43} + y^T_{32} + y^T_{21} + y^T_{15}) \geq 3$. The latter is inequality (4.22) associated with the circuit C of G_5 defined as $C = \{(5,4),(4,3),(3,2),(2,1),(1,5)\}$.

The graph representation of set covering polytopes is very useful to visualize schematically the valid inequalities of $RSC_{n,s}$. In the examples that follow, the reader can reconstruct such inequalities by observing the following guidelines. We represent the inequalities with blue vertices and the variables with white vertices. We will label the vertices with the name of the variables, *e.g.,* $y_{12}, y_{13}, \ldots$ To avoid unnecessary notational burden, the white vertices will be identified with the corresponding variables. In this way, when W is a set of white vertices, the notation $y(W)$ indicates the sum of the variables corresponding to the vertices in W.

We conclude this section with two examples in which we describe valid inequalities for some DTSPMS polytopes arising from generalized cycle inequalities. In the first example we provide an inequality obtained from a 2-rose inequality not associated with a circuit of G_n, as was the case in Corollary 4.2.11; in the second example we give an inequality valid for $DTSPMS_{n,3}$ which does not arises from a 3-rose inequality. Thus, on the one hand Proposition 4.2.10 lets us characterize faces of the DTSPMS polytope; on the other hand the facial structure of the latter seems somewhat richer.

Example 4.2.12. The bipartite graph in Figure 4.5 is a 2-rose of order 7. It corresponds to a subset of constraints defining $RSC_{n,2}$ for $n \geq 7$. Hence, calling $\mathcal{V}$ the set of white vertices in Figure 4.5 we have that $y^1(\mathcal{V}) + y^2(\mathcal{V}) \geq 4$ is valid for $DTSPMS_{n,2}$ for every $n \geq 7$. Note that this inequality cannot be expressed as in Proposition 4.2.10 since, for example, the subscript 1 is always in first position. In other words, the 2-rose of Figure 4.5 does not correspond to a circuit of G_n.

Example 4.2.13. Consider the bipartite graph in Figure 4.6. It corresponds to 10 constraints defining $RSC_{n,3}$, with $n \geq 9$. Let A be the left-hand matrix of this subset of constraints. We define $N_{i-1} = \{y_{i,i+1}\}$ for $i = 1, \ldots, 6$, $N_6 = \{y_{79}, y_{95}\}$, $N_7 = \{y_{86}\}$, $N_8 = \{y_{69}\}$, $N_9 = \{y_{91}\}$. Note that the N_k's are pairwise disjoint. Hence, calling $\mathcal{W}$ the set of white vertices in Figure 4.6, we get from Theorem 4.2.8 that $y^1(\mathcal{W}) + y^2(\mathcal{W}) \geq 4$ is valid for $DTSPMS_{n,3}$. This generalized cycle inequality is not associated with a circuit of G_n as in Proposition 4.2.10, since the subscript 8 appears only in first position.

In next section we apply the set covering approach to the double TSP with two stacks. In this case, the set covering polytope reduces to a specific vertex cover polytope.

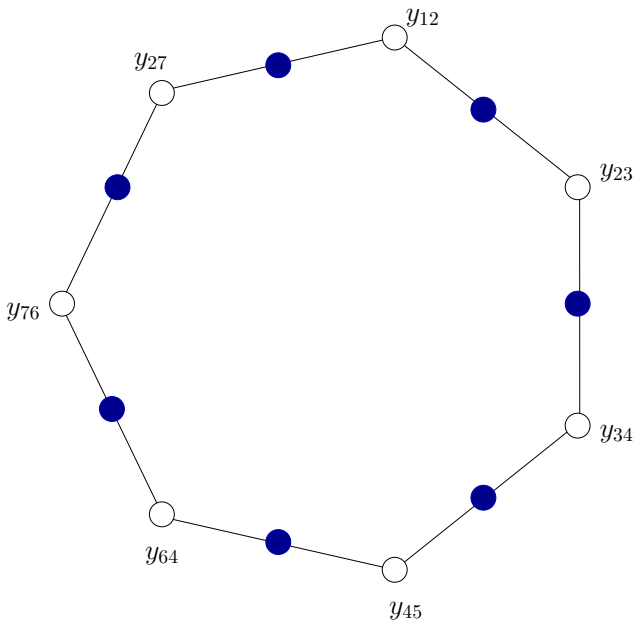


Figure 4.5: A 2-rose of order 7 arising from $RSC_{n,2}$, where $n \geq 7$. It does not arise from a circuit of G_n.

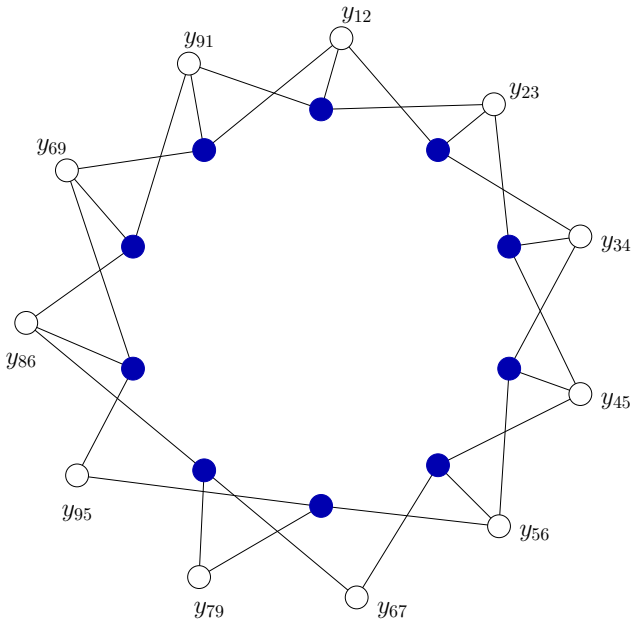


Figure 4.6: The graph corresponding to the generalized cycle inequality of Example 4.2.13. Blue vertices are constraints, white vertices correspond to variables.

4.2.3 Focus on Two Stacks: A Vertex Cover Approach

The starting point of this section is the observation that, in the special case of the double TSP with two stacks, the restricted set covering polytope $RSC_{n,2}$ is the vertex cover polytope of a suitable graph. Thus, valid inequalities for the vertex cover polytope can be converted, by Claim 4.2.5, into inequalities valid for the DTSPMS polytope.

An Interlude on Vertex Cover Polytopes

A *vertex cover* of a graph G is a set S of vertices such that each edge of G is incident to an element of S. The *vertex cover polytope* of a graph G is the convex hull of the incidence vectors of its vertex covers and it is denoted VC_G. It is immediate that for every graph $G = (W, E)$:

$$VC_G = \text{conv}\{y \in \{0,1\}^{|W|} \colon y_u + y_v \geq 1, \forall \{u, v\} \in E\}. \tag{4.23}$$

Vertex cover polytopes can be seen equivalently as stable set polytopes. Indeed, a subset of vertices of a graph is a vertex cover if and only if its complement is a stable set. The *stable set polytope* of G is defined as the convex hull of the incidence vectors of its stable sets. Given a graph $G = (W, E)$, let SS_G be the stable set polytope of G. Then we have:

$$SS_G = \{z \in \mathbb{R}^{|W|} \colon z = \mathbf{1} - y, y \in VC_G\}. \tag{4.24}$$

In fact, stable set polytopes arise frequently from combinatorial optimization problems [21], hence they have been intensively studied. By (4.24), all inequalities valid for the stable set polytope can be transferred to the vertex cover polytope (and vice-versa) by complementing the variables. Our goal is to exploit the existing literature on stable set and vertex cover polytopes to provide new inequalities valid for $RSC_{n,2}$.

The Vertex Cover in $DTSPMS_{n,2}$

Let us consider the double TSP with two stacks. The polytope $RSC_{n,2}$ is:

$$\text{conv}\{y \in \{0,1\}^{n(n-1)} \colon y_{ij} + y_{jk} \geq 1 \text{ for all distinct } i, j, k \in V \setminus \{0\}\}.$$

As it turns out, $RSC_{n,2}$ can be expressed as a vertex cover polytope. Let $S_n = (U, F)$ be the graph whose vertices are u_{ij} for all distinct $i, j \in V \setminus \{0\}$ and whose edges are $\{u_{ij}, u_{jk}\}$ for all distinct $i, j, k \in V \setminus \{0\}$. To describe the vertex cover polytope of S_n we associate the variable y_{ij} with the vertex u_{ij}, for all distinct $i, j \in V \setminus \{0\}$. Note that $RSC_{n,2}$ and the vertex cover polytope of S_n have the same variables. Moreover, each constraint in the $RSC_{n,2}$ definition contains two variables which correspond to the extremities of an edge of S_n. Therefore $RSC_{n,2}$ is nothing but the vertex cover polytope of S_n. We deduce that every inequality valid for the vertex cover polytope of S_n can be transformed into an inequality valid for $DTSPMS_{n,2}$.

The graph S_n can be interpreted as a simplification of the graph theoretical setting for set covering polytopes described at the end of Section 4.2. Let $\mathcal{B}$ be the bipartite graph associated with the set covering polytope defined by the 2-consistency constraints. The vertices of $\mathcal{B}$ associated to the 2-constraints have all degree 2. Hence one can think that $\mathcal{B}$ is obtained from S_n by subdividing the edges of S_n.

Several inequalities valid for the vertex cover polytope of a graph G are associated with subgraphs of G. We now derive inequalities valid for $DTSPMS_{n,2}$ by studying subgraphs of S_n.

Odd Hole Inequalities. Using the vertex cover approach we describe the 2-rose inequalities of $RSC_{n,2}$ in relation to specific vertex subsets of S_n. We say that $H \subseteq U$ is an *odd hole* of S_n if $|H|$ is odd and the subgraph induced by H in S_n is a circuit. There is a one-to-one correspondence between the vertices of S_n and the arcs of G_n. However, if every odd circuit of G_n provides an odd hole of S_n, the converse is not true (see Example 4.2.12). Hence, the following corollary generalizes Corollary 4.2.11 in the two stack case.

Corollary 4.2.14. *Inequalities*

$$y^1(H) + y^2(H) \geq \frac{|H|+1}{2} \text{ for all odd circuits } H \text{ of } S_n, \tag{4.25}$$

are valid for $DTSPMS_{n,2}$. If H is not an odd hole of S_n then (4.25) cannot define a facet of $DTSPMS_{n,2}$.

Proof. It is well-known [151] that $y(H) \geq \frac{|H|+1}{2}$ is valid for the vertex cover polytope of a graph when H is one of its odd circuits. Therefore, $y(H) \geq \frac{|H|+1}{2}$ is valid for $RSC_{n,2}$ for every odd circuit H of S_n. By Claim 4.2.5 we get that (4.25) is valid for $SC_{n,2}$ and thus for $DTSPMS_{n,2}$. The last part of the proposition follows from Proposition 4.2.3 and from the observation that $y(H) \geq \frac{|H|+1}{2}$ does not define a facet of $RSC_{n,2}$ if H is not an odd hole of S_n, see *e.g.,* [163]. □

Inequalities of the form (4.25) are called *odd hole inequalities* when H is an odd hole of S_n. The odd hole inequalities have been discovered by Padberg [151] in the context of the stable set polytope of a generic graph G.

Wheel Inequalities. Using the vertex cover approach we now give a new family of valid inequalities for $RSC_{n,2}$ that cannot be obtained, in general, as generalized cycle inequalities. These are called non-simple wheel inequalities and have appeared in several flavors in the literature on stable set polytopes [35, 48]. For their description we follow to a large extent the approach of de Vries [48].

For $k \geq 1$, we define a *k-wheel* as the graph (W, R) with $W = \{w_0, w_1, \ldots, w_{2k+1}\}$ and $R = \{\{w_0, w_i\}, \{w_i, w_{i+1}\} \colon i = 1, 2, \ldots, 2k+1\}$ (assuming $2k+2 \equiv 1$). The vertex w_0 is called *hub*, the other vertices are called *spoke ends* and the subgraphs induced by

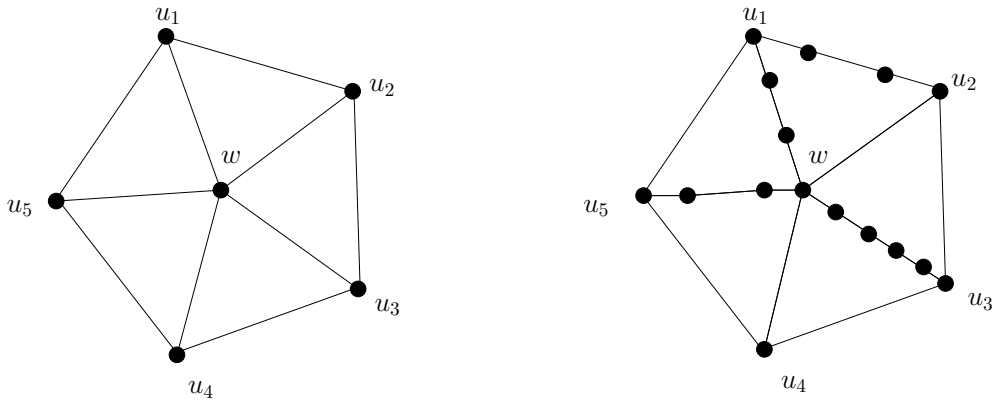


Figure 4.7: On the left, a 2-wheel, with hub in w and spoke ends in $u_1, \dots u_5$. On the right one of its odd subdivisions.

$\{w_0, w_i, w_{i+1}\}$ are called *faces* of the k-wheel, for $i = 1, \dots, 2k+1$ (with $2k+2 \equiv 1$). Note that $H = W \setminus \{w_0\}$ is an odd hole of (W, R).

A *subdivision* of a k-wheel is obtained from a k-wheel by subdividing some of its edges in such a way that each face remains a circuit of odd length. Given a k-wheel with hub w_0 and spoke ends $w_1, \dots, w_{2k+1}$, we distinguish the following types of subdivisions:

- *odd* subdivision of the k-wheel, in which, in addition, we require that each subdivision of an edge incident in w_0 is a path with an odd number of edges;
- *even* subdivision of the k-wheel, in which, in addition, we require that each subdivision of an edge incident in w_0 is a path with an even number of edges.

As a consequence, in both cases the subdivision of the edge $\{w_i, w_{i+1}\}$ is a path with an odd number of edges, for every $i = 1, \dots, 2k+1$ and $2k+2 \equiv 1$. See Figure 4.7 for the illustration of an odd subdivision of a 2-wheel.

Odd (or even) subdivisions of a k-wheel give rise to inequalities valid for the vertex cover polytope of a graph G. Let us suppose that a subgraph $\mathcal{W}$ of G is an odd subdivision of a k-wheel. Let w be the vertex of G corresponding to the hub of $\mathcal{W}$ and let W be the vertex set of $\mathcal{W}$. Then the following inequality is valid for the vertex cover polytope of G, see [35]:

$$ky_w + y(W \setminus \{w\}) \geq \frac{|W|}{2} + k. \tag{4.26}$$

Similarly, if $\mathcal{W}$ is an even subdivision a k-wheel having W as vertex set and w as hub, and $\mathcal{W}$ is a subgraph of G, then the inequality

$$(k+1)y_w + y(W \setminus \{w\}) \geq \frac{|W|+1}{2} + k \tag{4.27}$$

is valid for the vertex cover polytope of G. We call (4.26) and (4.27) *odd wheel inequality* and *even wheel inequality*, respectively. Applying them to the vertex cover polytope of S_n (*i.e.,* to $RSC_{n,2}$), we immediately get valid inequalities for $DTSPMS_{n,2}$.

Corollary 4.2.15. *Let us consider a subgraph $\mathcal{W}$ of S_n. Let W be the vertex set of $\mathcal{W}$. Then:*

- *if $\mathcal{W}$ is an odd subdivision of a k-wheel with w being its hub, the inequality*

$$ky_w^1 + ky_w^2 + y^1(W \setminus \{w\}) + y^2(W \setminus \{w\}) \geq \frac{|W|}{2} + k \tag{4.28}$$

  is valid for $DTSPMS_{n,2}$;

- *if $\mathcal{W}$ is an even subdivision of a k-wheel, the inequality*

$$(k+1)y_w^1 + (k+1)y_w^2 + y^1(W \setminus \{w\}) + y^2(W \setminus \{w\}) \geq \frac{|W|+1}{2} + k \tag{4.29}$$

  is valid for $DTSPMS_{n,2}$.

When no confusion may arise, inequalities (4.28) and (4.29) are also called odd wheel inequality and even wheel inequality, respectively.

Now, we present an important example whose goal is twofold. Firstly, it shows that subdivisions of k-wheels do appear in the graph S_n. Secondly, it provides an example of an odd wheel inequality that is not implied by the generalized cycle inequalities (4.20) of $RSC_{n,2}$.

The odd wheel inequality we consider is $y(W) \geq 6$, where W is the vertex set of the graph $\mathcal{W}$ depicted in Figure 4.8. Such a graph is a subgraph of S_8 and it is actually an odd subdivision of a 1-wheel, having its hub in u_{56} and spoke ends in u_{12}, u_{23} and u_{31}. Let us consider the vertex cover polytope of $\mathcal{W}$. It is the convex hull of the binary solutions to $A'y \geq \mathbf{1}$, where A' is the left-hand-side matrix of the subsystem of constraints defining $RSC_{8,2}$ that corresponds to the edges of $\mathcal{W}$. The matrix A' has 10 columns and each row contains exactly two nonzero entries. Therefore, the vertex cover polytope of $\mathcal{W}$ is full-dimensional. It can be proven by exhibiting an affine basis, that the inequality $y(W) \geq 6$ is facet-defining for the vertex cover polytope of $\mathcal{W}$.

As a consequence, up to multiplying it by a positive scalar, $y(W) \geq 6$ is not implied by any of the inequalities valid for the vertex cover polytope of $\mathcal{W}$.

We now prove that $y(W) \geq 6$ cannot be obtained as a generalized cycle inequality from the constraints $A'y \geq \mathbf{1}$. By contradiction, suppose that $\sum_{w \in W} \mu y_w \geq 6\mu$, for some integer $\mu \geq 1$ can be obtained as a generalized cycle inequality starting from constraints $A'y \geq \mathbf{1}$. This means that there are $N_0, \ldots, N_{p-1}$ such that $\sum_{w \in W} \mu y_w = \sum_{i=0}^{p-1} y(N_i)$ and $\left\lceil \frac{p}{q} \right\rceil = 6\mu$, for some $2 \leq q < p \leq 12$ (because A' has 12 rows). Hence, we immediately get that $\mu = 1$, implying that each variable of the generalized cycle inequality appears in exactly one N_j, for some $j = 0, \ldots, p-1$. Therefore the nonempty N_j's form a partition

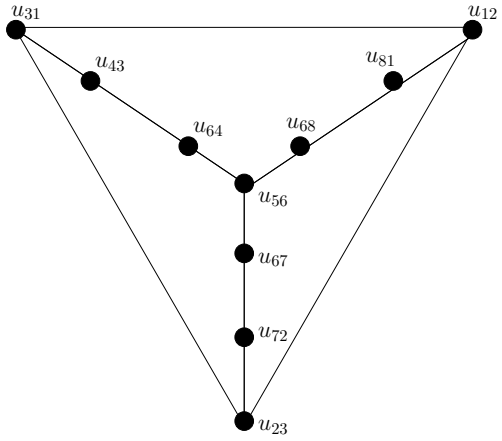


Figure 4.8: An odd subdivision of the 1-wheel. Each edge corresponds to a constraint of $RSC_{8,2}$ whose variables correspond to its endpoints.

of the variables of the generalized cycle inequality. Now if $q = 2$, we have that the variable associated to vertex u_{43} belongs to $N_k \cup N_{k+1}$ ($k \in \{0, \ldots, p-2\}$), the variable associated with vertex u_{12} belongs to $N_\ell \cup N_{\ell+1}$ ($\ell \in \{0, \ldots, p-2\}$) and the variable associated with u_{23} belongs to $N_r \cup N_{r+1}$ ($r \in \{0, \ldots, p-2\}$). Since u_{31} is the neighbor of u_{12}, u_{23} and u_{43} its corresponding variable belongs to $N_i \cup N_{i+1}$ for $i \in \{k, \ell, r\}$, a contradiction because the N_j's form a partition of the variables. Therefore $q \geq 3$ and this implies $\left\lceil \frac{p}{q} \right\rceil \leq 4$ for every $2 < p \leq 12$, a contradiction again. Finally, all the constraints defining $RSC_{8,2}$ that are not in $A'y \geq \mathbf{1}$ contain at least one variable corresponding to a vertex of $S_8 \setminus W$. Thus, the inequality $y(W) \geq 6$ is not a generalized cycle inequality obtained from the constraints defining $RSC_{8,2}$.

We conclude by mentioning a further generalization of the odd and even wheel inequalities, first introduced in [35] and refined in [48]. The generalized inequalities are associated with graph structures obtained from the identification of vertices of odd (or even) subdivisions of wheels. *Identifying* two vertices v_1 and v_2 of a graph $G = (W, E)$ *to* w means replacing them by a new vertex w which is incident to all edges previously incident to v_1 or v_2, see p. 55 in Bondy and Murty's book on Graph Theory [22]. This operation transforms G in a new graph having $(W \setminus \{v_1, v_2\}) \cup \{w\}$ as vertex set. Any valid inequality for the vertex cover polytope of a graph can be transformed into a valid inequality of the vertex cover polytope of the graphs obtained by identifying vertices.

Proposition 4.2.16 (see [35, 48])**.** *Let $G = (W, E)$ be a graph. Let $\sum_{v \in W} a_v y_v \geq b$ be a valid inequality of VC_G and let $T \subseteq W$ be a set of mutually non-adjacent vertices of G. Finally let $G' = (W', E')$ be the graph obtained from G by identifying the vertices in T to a single vertex t. Then $\left(\sum_{w \in T} a_w\right) y_t + \sum_{v \in W' \setminus \{w\}} a_v y_v \geq b$ is valid for $VC_{G'}$.*

Let us call *non-simple odd wheel inequality* any inequality obtained by applying Proposition 4.2.16 to an odd wheel inequality. Analogously define the *non-simple even inequality*.

Corollary 4.2.17. *Let $\sum_{v\in U} a_v y_v \geq b$ be an odd (resp. even) wheel inequality valid for $RSC_{n,2}$. Then any non-simple odd (resp. even) wheel inequality obtained from it is valid for $RSC_{n,2}$.*

Odd Hole Inequalities: Facet Condition in $RSC_{n,2}$**.** We now focus on the odd hole inequalities of $RSC_{n,2}$, that is inequalities of the form $y(H) \geq \frac{|H|+1}{2}$ where H is an odd hole of S_n. The condition that H induces a circuit of S_n is not sufficient to guarantee that such an inequality is facet-defining for $RSC_{n,2}$. Indeed, take a vertex u of S_n and suppose that it is adjacent to three vertices u_1, u_2, u_3 of H. If, in the circuit induced by H, the paths linking u_1 to u_2 and u_2 to u_3 are odd, then the graph induced by $H \cup \{u\}$ contains an odd subdivision of a 1-wheel as subgraph. The associated odd wheel inequality $y(H \cup \{u\}) \geq \frac{|H|+1}{2} + 1$, summed to the bound inequality $y_u \leq 1$, implies the odd hole inequality obtained from H. Using the specific structure of S_n we prove that in all other cases, odd hole inequalities do define facets of $RSC_{n,2}$.

Proposition 4.2.18. *Let H be an odd hole of S_n. The odd hole inequality*

$$y(H) \geq \frac{|H|+1}{2}$$

defines a facet of $RSC_{n,2}$ if and only if for every vertex $u \in U \setminus H$, the graph induced in S_n by $H \cup \{u\}$ does not contain an odd subdivision of a 1-wheel.

The necessity has already been observed above. Hence, in order to prove Proposition 4.2.18, we only need to show the sufficiency of the condition. More specifically, we initially fix an odd hole $H = \{u_1, \ldots, u_h\}$ and a vertex u of $S_n \setminus H$. We prove that, under the condition that $H \cup \{u\}$ induces in S_n a graph without any odd subdivision of the 1-wheel, there is a vertex cover of S_n not containing u and whose incidence vector belongs to the face of $RSC_{n,2}$ defined by the odd hole inequality associated with H. The incidence vectors constructed in this way then let us prove the maximality of the face induced by such inequality. For the sake of clarity, we write $y \in \mathbb{R}^{n(n-1)}$ as $y = (y_{u_1}, \ldots, y_{u_h}, y_u, \bar{y})$ with $\bar{y} \in \mathbb{R}^{n(n-1)-h-1}$. In addition, we assume that u_i is adjacent to u_{i+1} in the graph induced by H in S_n, for all $i = 1 \ldots, h$ ($u_{h+1} \equiv u_1$).

Lemma 4.2.19. *Let $H = \{u_1, \ldots, u_h\}$ be an odd hole of S_n and $u \in U \setminus H$ such that the graph induced by $H \cup \{u\}$ in S_n does not contain an odd subdivision of a 1-wheel. Then there exists a vertex cover Q^u of S_n such that $u \notin Q^u$, $v \in Q^u$ for all $v \in U \setminus (H \cup \{u\})$ and whose incidence vector belongs to $RSC_{n,2} \cap \{y(H) = \frac{|H|+1}{2}\}$.*

Proof. Let us first suppose that u has only one neighbor u_j in H for some $j \in \{1, \ldots, h\}$. If j is even, we define $Q^j = \{u_1, u_2, u_4, \ldots, u_j, u_{j+2}, \ldots, u_{h-1}\}$. Similarly, if j is odd we define the set $Q^j = \{u_1, u_3, u_5, \ldots, u_j, u_{j+2}, \ldots, u_h\}$. In both cases, $|Q^j| = \frac{|H|+1}{2}$. Let $z^j \in \{0,1\}^h$ be defined by $z^j_w = 1$ if and only if $w \in Q^j$. The vector $\beta^u = (z^j, 0, \mathbf{1})$ is the incidence vector of a vertex cover Q^u of S_n since it satisfies all constraints of $RSC_{n,2}$. In addition, $z(H) = \frac{|H|+1}{2}$ hence β^u proves the result in this case.

Similarly, let us suppose that u has exactly two neighbors u_j and u_k in H, for distinct $1 \leq j < k \leq h$. We can always construct a vertex cover $Q^{j,k}$ of the graph induced by H in S_n such that $u_j, u_k \in Q^{j,k}$ and $Q^{j,k}$ has cardinality $\frac{|H|+1}{2}$. For example, if both j and k are odd, one could take $Q^{j,k} = \{u_1, u_3, \ldots, u_j, u_{j+2}, \ldots, u_k, u_{k+2}, \ldots, u_{h-2}, u_{h-1}\}$. The other cases are similar. As before, let $z^{j,k} \in \{0,1\}^h$ be defined by $z^{j,k}_w = 1$ whenever $w \in Q^{j,k}$. The vector $\beta^u = (z^{j,k}, 0, \mathbf{1})$ proves the result as before.

We can now suppose that u has at least three neighbors in H. Let us indicate them with u_j, u_k, u_ℓ for distinct $j, k, \ell \in \{1, 2 \ldots, h\}$. By definition H induces a circuit in S_n with an odd number of edges. Since h is odd, it is without loss of generality to assume that u_j and u_k are linked in this circuit by a path with an odd number of edges and not containing any other neighbor of u. Let us call P_{jk} this path. We orient the circuit induced by H in S_n in the sense going from u_j to u_k along P_{jk}. We denote by $Q^{j,k}$ the subset of H given by u_j, its neighbor in P_{jk} and then by alternating vertices in $Q^{j,k}$ to vertices outside $Q^{j,k}$ by following the orientation. The set $Q^{j,k}$ is a vertex cover of the circuit induced by H and having $\frac{|H|+1}{2}$ vertices. In addition, all the neighbors of u in this circuit are contained in $Q^{j,k}$, as otherwise the graph induced in S_n by $H \cup \{u\}$ contains an odd subdivision of a 1-wheel. As above let $z^{j,k} \in \{0,1\}^h$ be defined by $z^{j,k}_w = 1$ if and only if $w \in Q^{j,k}$. Then, the vector $\beta^u = (z^{jk}, 0, \mathbf{1})$ proves the result. □

Remember that $S_n[H]$ is the graph induced by the vertex subset H in the graph S_n. This result is well-known, see *e.g.,* [43].

Lemma 4.2.20. *Let H be an odd hole of S_n. Then $y(H) \geq \frac{|H|+1}{2}$ defines a facet of the vertex cover polytope of $S_n[H]$.*

If $H = \{u_1, \ldots, u_h\}$ is an odd hole of S_n, then $\dim(VC_{S_n[H]}) = h$ and the previous result guarantees the existence of $\kappa^1, \kappa^2, \ldots, \kappa^h$ affinely independent vectors belonging to $VC_{S_n[H]} \cap \{y(H) = \frac{|H|+1}{2}\}$. Without loss of generality, the κ^i's are binary vectors. We are now ready to prove Proposition 4.2.18.

Proof of Proposition 4.2.18 (sufficiency). Let us assume that $H = \{u_1, \ldots, u_h\}$. For every $i = 1, 2, \ldots, h$, we define $\beta^i = (\kappa^i, \mathbf{1}) \in \{0,1\}^{|U|}$, where we remember that U is the vertex set of S_n and $\kappa^1, \ldots, \kappa^h$ are the vectors giving a basis of the face defined by the odd hole inequality associated with H in $VC_{S_n[H]}$. For every, $u \notin H$ we define β^u to be the incidence vector of the vertex cover Q^u defined in Lemma 4.2.19. Now the set $B = \{\beta^i \colon i = 1, \ldots h\} \cup \{\beta^u \colon u \notin H\}$ contains $|U|$ affinely independent points. Indeed, the independence of the vectors in the first set of the previous union follows from the independence of $\kappa^1, \ldots, \kappa^h$; moreover, each β^u in the second set of the union is affinely independent of all other vectors, because it is the only one to have a zero in the coordinate corresponding to $u \notin H$. By construction, all the points in B belong to $RSC_{n,2} \cap \{y(H) = \frac{|H|+1}{2}\}$. We conclude that B is an affine basis for the odd hole inequality defined by H in $RSC_{n,2}$. □

Remember that each facet of $SC_{n,2}$ can be retrieved from a facet of $RSC_{n,2}$. In particular, Proposition 4.2.18 also characterizes the odd hole inequalities that define facets

of $SC_{n,2}$. The question of which conditions should hold so that the odd hole inequalities define facets of $DTSPMS_{n,2}$ remains open.

4.3 Conclusion and Perspectives

In this chapter we have focused our attention on the DTSPMS polytope ($DTSPMS_{n,s}$), *i.e.,* the convex hull of the solutions to our formulation for the double TSP with stacks of infinite capacity presented in Chapter 3. We have studied the links between the DTSPMS polytope and the PATSP polytope, *i.e.,* the convex hull of the solutions of the TSP formulation (3.1)–(3.7). The PATSP polytope has been studied by several authors in the literature on the TSP. In particular the work of Gouveia and Pesneau [84] has introduced several families of inequalities valid for the PATSP polytope. This yields immediately valid inequalities for the DTSPMS polytope. From a polyhedral point of view, we have shown that the dimension of the DTSPMS polytope is twice the dimension of the PATSP polytope and that every facet-defining inequality for the PATSP polytope yields two facet-defining inequalities for the DTSPMS polytope. The latter result characterizes a super-polynomial number of facets of the DTSPMS polytope, using a recent result of Fiorini *et al.* [60]. Moreover it prompts the following open question:

Open Question 4.3.1. Find conditions under which the inequalities presented in Section 4.1.1 define facets of $PATSP_n$.

Note that any answer to the previous question would have consequences in both the TSP theory and the double TSP with multiple stacks theory.

In the second part of the chapter we have shown the link between the DTSPMS polytope and the polytope $SC_{n,s}$, *i.e.,* the convex hull of the binary points satisfying the s-consistency constraints (3.26). We have observed that $SC_{n,s}$ is a set covering polytope. Since $\text{proj}_{(y^1,y^2)}(DTSPMS_{n,s}) \subseteq SC_{n,s}$ we have introduced some new faces for the DTSPMS polytope by resorting to the known literature on set covering polytopes. In addition, the polyhedral structure of $SC_{n,s}$ can be completely retrieved from the polyhedral structure of a slightly simpler set covering polytope, called $RSC_{n,s}$.

An important property of $RSC_{n,s}$ is that when $s = 2$ (*i.e.,* in the case of the double TSP with two stacks) it is the vertex cover polytope of a suitable graph. Using this correspondence we were able to give specific inequalities for the DTSPMS polytope based on odd hole and wheel structures of a suitable graph. Finally, we have presented a preliminary study on the facet-defining property of the inequalities arising from the odd holes of this graph. Our result gives an equivalent condition for such an inequality to be facet-defining for $RSC_{n,2}$.

Concerning our set covering approach, a first direction for future work would be to give a better understanding of $RSC_{n,s}$.

Open Question 4.3.2. Find inequalities valid for $RSC_{n,2}$ different from the odd hole and the wheel inequalities presented in this chapter. Generalize to $RSC_{n,s}$.

The following direction of work is instead more related to the double TSP with multiple stacks. We would like to find conditions under which the facet-defining inequalities of $RSC_{n,s}$ induce facets of the corresponding DTSPMS polytope. More precisely we have this open question.

Open Question 4.3.3. Under which conditions an inequality defining a facet of $SC_{n,s}$ is also facet-defining for the DTSPMS polytope?

We think that this question is not easy to answer. In fact, the DTSPMS polytope is contained in another, more complex, set covering polytope obtained from transitivity constraints (3.24) and from a relaxed form of the antisymmetry constraints (3.23). That is, one could think to study the convex hull $\mathrm{CAT}_{n,s}$ of the binary solutions to the following system of constraints:[4]

$$\sum_{i=1}^{s}(y^1_{j_ij_{i+1}} + y^2_{j_ij_{i+1}}) \geq 1 \qquad \text{for all distinct } j_1, \ldots, j_{s+1} \in V \setminus \{0\}, \tag{4.30}$$

$$y^T_{ij} + y^T_{ji} \geq 1 \qquad \text{for all } i \neq j \in V \setminus \{0\} \text{ and } T = 1, 2, \tag{4.31}$$

$$y^T_{ij} + y^T_{jk} + y^T_{ki} \geq 1 \qquad \text{for all } i \neq j \neq k \neq i \in V \setminus \{0\} \text{ and } T = 1, 2. \tag{4.32}$$

The interest of polytope $\mathrm{CAT}_{n,s}$ is that $\mathrm{proj}_{(y^1,y^2)}(DTSPMS_{n,s})$ is one of its faces, as it can be seen by setting at equality constraints (4.31). Hence, a third research direction is to give a polyhedral understanding of this new set covering polytope. This understanding could be valuable to further improve the set covering approach for the double TSP with multiple stacks devised in this chapter.

[4]CAT is for Consistency–Antisymmetry–Transitivity.

Chapter 5

A Branch-and-Cut Algorithm

Contents

In this chapter we describe a branch-and-cut algorithm for the double TSP with two stacks. The algorithm is based on several families of inequalities that are added to our formulation for the problem in order to strengthen its linear relaxation. The inequalities used in our branch-and-cut algorithm correspond to faces of the DTSPMS polytope $DTSPMS_{n,2}$ presented in Chapter 4. We recall that they can be divided into two categories: routing inequalities, arising from the PATSP polytope and consistency inequalities, arising from a set covering polytope. Not surprisingly, and as we also report in this chapter, the time complexity of the separation routines for the routing inequalities does not depend on the number of stacks. In addition, for the routing inequalities presented in Chapter 4, Gouveia and Pesneau have described in [84] polynomial-time exact separation algorithms. On the other hand, in the two stack case we give a polynomial-time exact separation algorithm for the odd hole inequalities (4.25), easily embeddable in a branch-and-cut algorithm. Hence the two stack case lets us test the effectiveness of inequalities of both routing and consistency types.

Although the focus on the two stack case can seem restrictive at first, previous works on the double TSP with multiple stacks have shown that, computationally, it is the hardest

case to deal with, when the capacity is finite. More precisely, the exact algorithms designed for a general number of stacks and presented in [2, 131, 157] are able to solve instances of the double TSP with two stacks with up to 14 items and only the algorithm in [44] solves three instances of the problem with two stacks and 16 items within three hours of computational time. This is much less than in the double TSP with three or four stacks for which the state-of-the-art algorithm of [2] solves instances respectively with up to 21 and 28 items. Thus, we believe that it is important to address the design of faster methods for the special case of the double TSP with two stacks.

The chapter is organized as follows. In Section 5.1 we describe the main features of the branch-and-cut algorithm, such as the starting formulation, the strengthening inequalities and the order of separation. In Section 5.2 we give polynomial-time algorithms to solve the separation problem of the strengthening inequalities embedded in our algorithm. In Section 5.4 we present the computational results obtained by our algorithm. In Section 5.5 we draw some conclusions and present possible extensions of our work.

5.1 Overall Description of the Algorithm

In this section, we introduce the general branch-and-cut framework subsequently used to tackle the double TSP with two stacks, that is we consider the case $s = 2$. We present it in the infinite capacity case. In Section 5.3 we explain how to adapt it to the case of stacks of finite capacity.

Unless differently stated, we understand routing inequalities (those not mixing pickup and delivery variables) to be duplicated in the pickup and delivery variables.

The goal of our branch-and-cut algorithm is to find an optimal solution to the problem:

$$\begin{aligned} \min \quad & c^1x^1 + c^2x^2 \\ & (x^1, y^1, x^2, y^2) \text{ satisfies (3.21)–(3.28).} \end{aligned} \tag{5.1}$$

Our algorithm is based on some modifications to the formulation given in Section 3.3.1 and defining the constraint set of problem (5.1) above. We first motivate these modifications, then we describe the algorithm.

Starting Formulation. A first modification to the constraint set of problem (5.1) consists in replacing the transitivity constraints (3.24) with constraints (4.3). This is motivated by the fact that (4.3) is stronger than (3.24), as explained in Section 4.1.1.

According to the description provided in Section 1.6.3, a branch-and-cut algorithm seeks an optimal solution for a combinatorial optimization method by exploring the feasibility region of the problem. Bounding arguments are used to avoid a complete exploration. In a minimization problem like the double TSP with multiple stacks the computation of lower bounds is particularly important.

In our branch-and-cut algorithm the initial lower bound is obtained from the linear relaxation of an integer linear programming formulation. We describe its specific features. The starting formulation involves arc and precedence variables. It contains constraints (3.21)–(3.23) and (3.25). In addition, the starting formulation contains the $O(n^2)$ size families (4.1) and (4.2). Finally, if the instance is defined by symmetrical cost vectors (*symmetrical instance*), the resulting symmetry is reduced by including in the formulation the equation:

$$y^1_{12} = 1.$$

We detail more on this latter point. If $c^1, c^2 \in \mathbb{R}_+^{|A|}$ are symmetrical cost vectors over the arcs of $G_n = (V, A)$, it is immediate to see that $c^1(H_1) + c^2(H_2) = c^1(\overleftarrow{H}_1) + c^2(\overleftarrow{H}_2)$ for every pair of 2-consistent Hamiltonian circuits (H_1, H_2) of G_n. In addition, also $(\overleftarrow{H}_1, \overleftarrow{H}_2)$ is 2-consistent. Thus, if an instance of the double TSP with multiple stacks is defined by symmetrical cost vectors we can remove exactly one between (H_1, H_2) and $(\overleftarrow{H}_1, \overleftarrow{H}_2)$ without removing all optimal solutions to the problem. This removal reduces the feasibility space speeding up the whole algorithm. In the case of the double TSP with multiple stacks defined on symmetrical instances, this symmetry reduction can be accomplished *e.g.,* by supposing that item 1 is always picked up before item 2, in formulas $y^1_{12} = 1$ holds for all (x^1, y^1, x^2, y^2) representing pairs of Hamiltonian circuits of G_n.

We point out that the lifted transitivity constraints (4.3) and the 2-consistency constraints (3.26) are not considered in the starting formulation. The removal of constraints (4.3) and (3.26) is motivated by the fact that they are the largest families of problem defining constraints, each containing $O(n^3)$ constraints, where n is the number of items of an instance. They can be added by means of row generation procedures, described in Section 5.2.

The algorithm. Let us suppose to have an instance of the double TSP with two stacks with n items and cost vectors c^1 and c^2 on the graph $G_n = (V, A)$. The branch-and-cut algorithm starts by solving the following linear program:

$$\begin{aligned}
\min \quad & c^1x^1 + c^2x^2 \\
& (x^1, y^1, x^2, y^2) \text{ satisfies (3.21)–(3.23),(3.25),(4.1),(4.2),} \\
& y^1_{12} = 1, \\
& x^T_{ij} \in [0,1] \quad \forall \text{ distinct } i, j \in V,\ T = 1, 2, \\
& y^T_{ij} \in [0,1] \quad \forall \text{ distinct } i, j \in V \setminus \{0\},\ T = 1, 2.
\end{aligned} \tag{5.2}$$

Let us suppose that we obtain as optimal solution to the linear program above a binary point $(\bar{x}^1, \bar{y}^1, \bar{x}^2, \bar{y}^2)$. This point is feasible for the double TSP with two stacks exactly when it satisfies (3.26) and $(\bar{x}^T, \bar{y}^T)$ satisfies (4.3) for $T = 1, 2$. In general, this is not the case. Then, the separation step takes place and the branch-and-cut algorithm generates violated constraints from both families (3.26) and (4.3) and adds them to the initial linear program. Now, the resulting linear program is solved. The algorithm continues in this way by adding violated constraints to the current linear program.

We point out that, at any iteration of this process, the optimal solution to the current linear program could be fractional. In this case, the algorithm separates the family of constraints (3.26), (4.3) and other strengthening families. In our algorithm we consider several families of strengthening inequalities presented in Chapter 4: the GDDL inequalities (4.4), the Simple Cut inequalities (4.5)–(4.7) and the odd hole inequalities (4.25). The algorithm looks for a violation of each family following a prescribed order. When a family yields a violation, violated inequalities from that family are generated and added to the current linear program. The algorithm then does not check the violation of the subsequent families and proceeds instead with the resolution of the resulting linear program. We mention however that in our implementation we let the solver CPLEX add automatically all families of strengthening inequalities it provides. The order in which the algorithm checks a violation of the families above is the following:

- 2-consistency constraints (3.26),
- GDDL inequalities (4.4),
- Simple Cut inequalities (4.5)–(4.7),
- lifted transitivity constraints (4.3),
- odd hole inequalities (4.25).

When in the separation step no violation is found, either the last optimal solution is binary, or it is not. In the first case, it is also a feasible solution to the double TSP with multiple stacks. In the second case, the algorithm branches on a fractional coordinate of the last optimal solution. The choice of the variable to branch on is made according to the strong branching strategy embedded in CPLEX [6, 107]. After the branching, two new linear programs are generated. The procedure described above is repeated on an unsolved linear program. The next linear program on which to apply the separation step is the one with the best objective function. When no linear program is left unsolved, the algorithm takes the best integer feasible solution found so far as the optimal solution to the problem. For a more schematic presentation of our branch-and-cut algorithm, please see Algorithm 1.

We point out that all the inequalities added during the separation steps are globally valid. However we decided to add them *locally i.e.,* they are discarded when the algorithm backtracks in the search tree. This lets the algorithm solve linear programs of moderate size along its execution.

The next section addresses the separation problems of each family of strengthening inequalities used in our algorithm.

5.2 Separation Algorithms

In this section we present the routines used in our branch-and-cut algorithm to find violations of the inequalities considered in the separation step.

Algorithm 1: Branch-and-cut algorithm for the double TSP with two stacks

Data: A digraph $G = (V, A)$, two cost vectors $c^1, c^2 \in \mathbb{R}^{|A|}$
Result: Optimal solution to (5.1)

```
1  PL← (5.2);
2  Solve PL;
3  z* ←optimal solution to PL;
4  if z* is binary then
5  |  if z* satisfies (3.26) and (4.3) then
6  |  |  z* is a feasible solution to (5.1);
7  |  |  go to 22;
8  |  else
9  |  |  Add to PL violated constraints from (3.26) and (4.3);
10 |  |  Go to 2;
11 |  end
12 else
13 |  for family=(3.26),(4.4),(4.5)-(4.7),(4.3),(4.25) do
14 |  |  Check whether z* violates inequalities from family;
15 |  |  if a violation is found then
16 |  |  |  Add violated inequalities from family to LP;
17 |  |  |  Go to 2;
18 |  |  end
19 |  end
20 end
21 Branch on a fractional coordinate of z*;
22 Choose an unsolved linear program LP' among those generated by branching;
23 if LP' does not exist then
24 |  pick the best feasible solution found so far. STOP;
25 else
26 |  LP←LP';
27 |  Go to 2;
28 end
```

In all the separation algorithms presented below, we assume that $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$ is a solution found by solving a linear programming generated during the branch-and-cut process. All families but the 2-consistency and the odd hole families contain routing inequalities. For the routing inequalities, we describe the separation algorithms by considering only the pickup part of the solution $(\tilde{x}^1, \tilde{y}^1)$.

Lifted Transitivity Constraints

We split the discussion into two cases. In the first case we assume that $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$ is binary. In the second case we suppose that it is fractional. In the first case, we check whether there is a violation in the $O(n^3)$ family of constraints (4.3) in $O(n^2)$ time.

Binary Solution. If $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$ is binary, then $(\tilde{x}^1, \tilde{y}^1)$ corresponds to the characteristic point of a subgraph $\mathcal{S}$ of G_n. If $\mathcal{S}$ is a Hamiltonian circuit of G_n no lifted transitivity constraints (4.3) is violated, since constraints (4.3) define faces of the PATSP polytope. Using $(\tilde{x}^1, \tilde{y}^1)$, we decide in $O(n^2)$ time whether $\mathcal{S}$ is a Hamiltonian circuit of G_n.

Let us suppose that $\mathcal{S}$ is not a Hamiltonian circuit. By the assignment constraints (3.21) and (3.22), we have that $\mathcal{S}$ is the union of several circuits of G_n. We can always find one such a circuit $\mathcal{C}$ not containing the vertex 0. For notational convenience let us suppose that $\mathcal{C} = \{(c_1, c_2), (c_2, c_3), \ldots, (c_{k-1}, c_k), (c_k, c_1)\}$, for some vertices $c_1, c_2, \ldots, c_k$ of $G_n \setminus \{0\}$. Finding $\mathcal{C}$ takes $O(n^2)$ time. Note that $k \geq 3$ since $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$ satisfies the antisymmetry constraints (3.23).

Let $3 \leq \ell \leq k$ be the first index such that $\tilde{y}^1_{c_\ell c_1} > 0$. Then $\tilde{y}^1_{c_1 c_{\ell-1}} + \tilde{y}^1_{c_{\ell-1} c_\ell} + \tilde{y}^1_{c_\ell c_1} > 2$. This is a violated transitivity constraint (3.4). Thus also its lifted form (4.3) is violated by $(\tilde{x}^1, \tilde{y}^1)$ and added to the current linear program. See Figure 5.1 for an example.

Discovering the index ℓ as above takes $O(n)$ time, since it suffices to fix one vertex c_1 of $\mathcal{C}$ and, for every $j = 3, \ldots, k-1$, to check whether $\tilde{y}^1_{c_j c_1} = 1$.

The procedure described above is an exact separation routine for the transitivity constraints (4.3) when $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$ is binary.

Fractional Solution. When $(\tilde{x}^1, \tilde{y}^1)$ is fractional, a violation to constraints (4.3) is found by enumerating (in the worst case) all these constraints. In this case all violated constraints are added to the current linear program. This is clearly an exact separation routine for constraints (4.3) that runs in $O(n^3)$ time.

Consistency Constraints

The family of 2-consistency constraints (3.26) contains $\Theta(n^3)$ inequalities. We can separate them exactly in $O(n^2)$ time. Consider $j^\star \in V \setminus \{0\}$. Let

$$i^\star = \operatorname*{arg\,min}_{i \in V \setminus \{0, j^\star\}} \{\tilde{y}^1_{ij^\star} + \tilde{y}^2_{ij^\star}\}$$
$$k^\star = \operatorname*{arg\,min}_{i \in V \setminus \{0, j^\star\}} \{\tilde{y}^1_{j^\star k} + \tilde{y}^2_{j^\star k}\}.$$

Algorithm 2: Exact separation algorithm for the lifted transitivity constraints

Data: A point $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$, a value $0 < \Delta_{\texttt{tr}} \leq 1$
Result: A set of constraints (4.3) violated by $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$ of at least $\Delta_{\texttt{tr}}$

if $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$ *is binary* **then**
  $\mathcal{S} \leftarrow$the subgraph of G_n corresponding to $(\tilde{x}^1, \tilde{y}^1)$;
  if $\mathcal{S}$ *is a Hamiltonian circuit* **then**
    No constraint (4.3) can be violated. STOP;
  else
    find a circuit $\mathcal{C}$ contained in $\mathcal{S} \setminus \{0\}$;
    $v \leftarrow$arbitrary vertex of $\mathcal{C}$;
    Following $\mathcal{C}$ find its first vertex t such that $\tilde{y}^1_{tv} = 1$;
    $u \leftarrow$last index visited before t;
    Return $y^1_{vt} + y^1_{tu} + y^1_{uv} - x^1_{uv} \geq 1$. STOP;
  end
else
  for *distinct* $1 \leq v, u, t \leq n$ **do**
    if $\tilde{y}^1_{vt} + \tilde{y}^1_{tu} + \tilde{y}^1_{uv} - \tilde{x}^1_{uv} \leq 1 - \Delta_{tr}$ **then**
      Return $y^1_{vt} + y^1_{tu} + y^1_{uv} - x^1_{uv} \geq 1$;
    end
  end
end

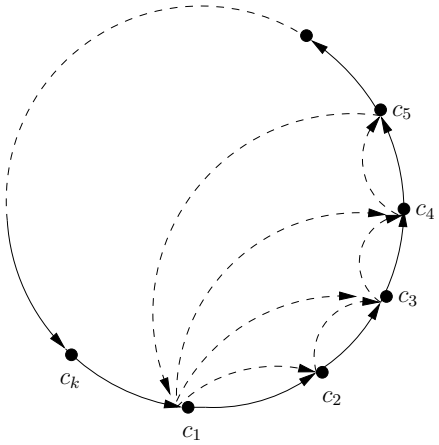


Figure 5.1: Black arrows are variables x^1, dashed ones are variables y^1; here vertices c_1, c_4, c_5 yield a violated transitivity constraint.

If $\tilde{y}^1_{i^\star j^\star} + \tilde{y}^2_{i^\star j^\star} + \tilde{y}^1_{j^\star k^\star} + \bar{y}^2_{j^\star k^\star} < 1$, then the inequality (3.26) associated with $i^\star$, $j^\star$ and $k^\star$ is violated because (3.3) guarantees that $i^\star \neq k^\star$. Otherwise, no inequality (3.26) associated with i, $j^\star$ and k is violated by $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$. Applying this procedure to each $j^\star \in V \setminus \{0\}$ gives an exact separation of the 2-consistency constraints in $O(n^2)$.

Algorithm 3: Exact separation algorithm for the 2-consistency constraints

Data: A point $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$, a value $\Delta_{\mathsf{cc}} > 0$
Result: A set of constraints (3.26) violated by $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$ of at least Δ_{cc}

$\iota = +\infty$ `// will hold the min over indices` i
$\kappa = +\infty$ `// will hold the min over indices` k
for $1 \leq j \leq n$ **do**
  $\iota \leftarrow \min_{i \in V \setminus \{0,j\}} \{\tilde{y}^1_{ij} + \tilde{y}^2_{ij}\}$;
  $i^\star \leftarrow$an index attaining the minimum above;
  $\kappa \leftarrow \min_{k \in V \setminus \{0,j\}} \{\tilde{y}^1_{ki} + \tilde{y}^2_{ki}\}$;
  $k^\star \leftarrow$an index attaining the minimum above;
  if $\iota + \kappa \leq 1 - \Delta_{\mathsf{cc}}$ **then**
    Return $y^1_{i^\star j} + y^2_{i^\star j} + y^1_{jk^\star} + y^2_{jk^\star} \geq 1$
  end
end

The Generalized Disaggregated Desrochers and Laporte Inequalities

The GDDL inequalities (4.4) can be separated exactly in $O(n^6)$ time by computing a minimum weight cut of $G_n = (V, A)$ for every pair of two vertices in V, see Gouveia and Pesneau [84]. We instead use a heuristic method to detect a violation of (4.4) having a

$O(n^4)$ time complexity. Our method is very close to the exact method presented in [84], but we reduce the number of minimum cuts to be computed, attempting to limit this computation to the most promising cases. For fixed $i, j, k \in V \setminus \{0\}$, and $S \subseteq V \setminus \{0, k\}$ and $i, j \in S$, the corresponding GDDL inequality expressed in pickup variables can be rewritten as:

$$y^1_{ki} - y^1_{kj} + 1 \leq x^1(\delta^+(S)).$$

Our heuristic method computes for every $k \in V \setminus \{0\}$ two indices i^k and j^k such that $\tilde{y}^1_{ki^k} \geq \tilde{y}^1_{ki}$ for all $i \in V \setminus \{0, k\}$ and $\tilde{y}^1_{kj^k} \leq \tilde{y}^1_{kj}$ for all $j \in V \setminus \{0, k\}$. Subsequently, if $i^k \neq j^k$, it seeks a subset $W^\star$ of the vertices of G_n such that

$$x^1(\delta^+(W^\star)) \leq \tilde{x}^1(\delta^+(W))$$

for all subsets W of vertices of G_n such that $i^k, j^k \in W$, $0, k \notin W$. This problem amounts to compute a minimum $(i^k, 0)$-cut in $(G_n, \tilde{x}')$ where $\tilde{x}'_{i^k j^k} = \tilde{x}'_{j^k i^k} = M$ and $\tilde{x}'_{0k} = \tilde{x}'_{k0} = M$ for some big value of M, whereas all other coordinates of $\tilde{x}'$ are identical to the corresponding coordinates of $\tilde{x}^1$. Let m^k be the weight of this cut and S^k be the set of vertices that originates the cut and that contains i^k and j^k. If $m^k < \tilde{y}^1_{ki^k} - \tilde{y}^1_{kj^k} + 1$ then the GDDL constraint above with $i = i^k$, $j = j^k$ and $S = S^k$ is violated by $(\tilde{x}^1, \tilde{y}^1)$. The process described above takes place for all $k \in V \setminus \{0\}$ and it is interrupted when a violation is found. The computation of the minimum weighted cuts of $(G_n, \tilde{x}')$ is performed with the preflow push-relabel algorithm of Goldberg and Tarjan [81] having running time in $O(|V|^2\sqrt{|A|})$. As a consequence, the entire heuristic algorithm has running time in $O(n^4)$.

The Simple Cut Inequalities

The Simple Cut inequalities can be separated exactly in $O(n^5)$ time by computing a minimum cut of $G_n = (V, A)$ for every pair of vertices in V, see Gouveia and Pesneau [84]. We instead use a heuristic method to detect a violation of (4.5)–(4.7) having a $O(n^4)$ time complexity. Our method is very close to the exact method presented in [84]. It simply reduces the number of minimum cuts to be computed. We give details only for the family (4.5), since the other cases are analogous. For the sake of clarity, we recall constraints (4.5) expressed in the pickup variables:

$$y^1_{ij} \leq \sum_{u \in V \setminus (S \cup \{j\})} \sum_{t \in S} x^1_{ut} \qquad S \subseteq V \text{ such that } 0, j \notin S, i \in S.$$

In our heuristic algorithm we find for each $j \in V \setminus \{0\}$ the index i^j such that $\tilde{y}^1_{i^j j} \geq \tilde{y}^1_{ij}$ for all $i \in V \setminus \{0, j\}$. Then for each $j \in V \setminus \{0\}$ we compute a minimum $(0, i^j)$-cut in the weighted graph $(G', \tilde{x}')$ where $G' = G_n \setminus \{j\}$, and $\tilde{x}'$ is the restriction of $\tilde{x}^1$ to the variables corresponding to arcs not incident to j in G_n. Let m^j be the weight of the minimum cut obtained above and S^j be the set of vertices originating this cut and containing i^j. If

Algorithm 4: Heuristic separation algorithm for the GDDL inequalities (4.4).

Data: A point $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$, a value $\Delta_{\texttt{gddl}} > 0$
Result: A constraint (4.4) violated by $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$ of at least $\Delta_{\texttt{gddl}}$

```
for k = 1, ..., n do
    Find the index i^k ∈ V \ {0, k} such that ỹ¹_{ki^k} ≥ ỹ¹_{ki} for all i ∈ V \ {0, k};
    Find the index j^k ∈ V \ {0, k} such that ỹ¹_{kj^k} ≤ ỹ¹_{kj} for all j ∈ V \ {0, k};
    if i^k ≠ j^k then
        Choose a big value M;
        x̃¹_{i^k j^k} ← M, x̃¹_{j^k i^k} ← M;
        x̃¹_{0k} ← M, x̃¹_{k0} ← M;
        m^k ←value of a minimum (i^k, 0)-cut in (G', x̃');
        S^k ←set of the tails of the cut above;
        if m^k ≤ ỹ¹_{ki^k} − ỹ¹_{kj^k} + 1 − Δ_gddl then
            Return y¹_{ki^k} − y¹_{kj^k} + 1 ≤ Σ_{u∈S^k} Σ_{t∈V\S^k} x¹_{ut}. STOP;
        end
    end
    Recover the original values of x̃¹;
end
```

$m^j < \tilde{y}^1_{i^j j}$ the constraint above with $S = S^j$ and $i = i^j$ is violated. The computation of the minimum weighted cuts is executed for decreasing values of $\tilde{y}^1_{i^j j}$. When a violation is found the subsequent minimum cuts are not computed. However, the algorithm is repeated, with the obvious modifications, to find violations of inequalities (4.6) and (4.7). Hence at most three Simple Cut inequalities are added each time the separation routine is called.

The computation of the minimum weighted cuts of $(G', \tilde{x}')$ is performed as for the GDDL inequalities. As a consequence, the entire heuristic algorithm has worst-case running time in $O(n^4)$.

Odd Hole Inequalities

We recall that the odd hole inequalities (4.25) are associated with odd holes of the graph $S_n = (U, F)$ with U given by the ordered pairs of distinct indices in $\{1, \ldots, n\}$. Note that if H is an odd hole of S_n then $\tilde{y}^1(H) + \tilde{y}^2(H) < \left\lceil \frac{H}{2} \right\rceil = \frac{|H|+1}{2}$ if and only if $w(H) < \frac{1}{2}$, where the weight vector $w \in \mathbb{R}^{|F|}$ is defined by $w_{\{u,v\}} = \frac{\tilde{y}^1_u + \tilde{y}^2_u + \tilde{y}^1_v + \tilde{y}^2_v - 1}{2}$ for all $u, v \in U$. If we construct the vector w under the assumption that $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$ satisfies the 2-consistency constraints, then $w_{\{u,v\}} \geq 0$. This is in fact the case, because of the separation order given in Section 5.1. Thus $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$ violates (4.25) if and only if the odd circuits of minimum weight in the edge-weighted graph (S_n, w) have weight smaller than $\frac{1}{2}$.

In order to find an odd circuit of minimum weight in (S_n, w) we use a classical algorithm first described by Gerards and Schrijver [75]. It consists in computing $|U|$ shortest paths in an auxiliary graph $\mathcal{S}_n = (\mathcal{U}, \mathcal{F})$ constructed as follows. We obtain the vertex set $\mathcal{U}$

Algorithm 5: Heuristic separation algorithm for the Simple Cut inequalities of type (4.5). It is repeated for (4.6) and (4.7).

Data: A point $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$, a value $\Delta_{\texttt{sc}} > 0$
Result: A constraint (4.5) violated by $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$ of at least $\Delta_{\texttt{sc}}$

for $j \in V \setminus \{0\}$ **do**
  Find the index $i^j \in V \setminus \{0, j\}$ such that $\tilde{y}^1_{i^j j} \leq \tilde{y}^1_{ij}$ for all $i \in V \setminus \{0, j\}$;
end
Sort the indices j for decreasing value of $\tilde{y}^1_{i^j j}$;
for *each* j *in the order obtained above* **do**
  Create the digraph G' from G_n by removing j from the vertex set;
  Create the weight vector x' from $\tilde{x}^1$ by removing coordinates with j in the subscript;
  $m^j \leftarrow$value of a minimum weight $(0, i^j)$-cut in $(G', \tilde{x}')$;
  $S^j \leftarrow$the set of vertices of G' containing the heads of the cut above;
  if $m^j \leq \tilde{y}^1_{i^j j} - \Delta_{sc}$ **then**
    Return $y^1_{i^j j} \leq \sum_{u \in V \setminus (S^j \cup \{j\})} \sum_{t \in S^j} x^1_{ut}$. STOP;
  end
end

by creating two copies u^1 and u^2 for each vertex $u \in U$. We say that u is the *original vertex* of u^1 and u^2. In addition, for every edge $e = \{u, v\}$ of S_n we create the two edges $e^1 = \{u^1, v^2\}$ and $e^2 = \{u^2, v^1\}$ of $\mathcal{S}_n$. We say that e is the *original edge* of e^1 and e^2. We associate with e^1 and e^2 the weight w_e of their original edge. Now, every walk P_u of $\mathcal{S}_n$ with extremities u^1 and u^2 corresponds to an odd closed walk of S_n containing u, the original vertex of u^1 and u^2. The correspondence is found by replacing in sequence, starting with u^1, each vertex of P_u with its original vertex and each edge of P_u with its original edge. The weight of the path P_u is the same as the weight of the resulting odd closed walk. Each odd closed walk contains an odd hole having the same weight, since the weights on the edges of $\mathcal{S}_n$ are nonnegative. Therefore, we can find the odd hole of $\mathcal{S}_n$ of minimum weight by computing for each $u \in \mathcal{U}$ the shortest path having u^1 and u^2 as extremities. Note that, since the edge-weights in $\mathcal{S}_n$ are nonnegative, the shortest paths can be computed by using Dijkstra's algorithm [51], see [66] for a $O(|\mathcal{F}| + |\mathcal{U}| \log |\mathcal{U}|)$ time implementation.

We did not implement directly the procedure described above. We instead used the odd hole inequality separation algorithm described in [164], whose source code was available at the URL `http://rebennack.net/Publications.htm` at the time of this writing. It is identical to the procedure described above apart from two details: first, the weighted graph (S_n, w) is preprocessed by eliminating the vertices $u \in U$ such that $w_u = 0$, since they cannot belong to an odd hole H yielding a violated odd hole inequality; second, the computation of the shortest path linking u^1 to u^2 is not performed if their original vertex u belongs to an already found odd hole of S_n yielding a violated inequality. Finally, when the algorithm finds an odd hole yielding a violation and containing a vertex u, it

does not look for additional odd holes containing u but it proceeds to the next available vertex of S_n on which to execute the shortest path routine. The total running time is in $O(|U| \cdot (|\mathcal{F}| + |\mathcal{U}| \log |\mathcal{U}|))$. Since $|U|, |\mathcal{U}| \in O(n^2)$ and $|\mathcal{F}| \in O(n^3)$ we get a $O(n^5)$ time separation algorithm.

Algorithm 6: Separation algorithm for the odd hole inequalities (4.25).

Data: A point $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$, the graph $S_n = (U, F)$ and a value $\Delta_{\texttt{oh}} > 0$
Result: A set of constraints (4.25) violated by $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$ of at least $\Delta_{\texttt{oh}}$

$\mathcal{U} \leftarrow \emptyset$;
$\mathcal{F} \leftarrow \emptyset$;
for $u \in U$ **do**
  $\mathcal{U} \leftarrow \mathcal{U} \cup \{u^1, u^2\}$;
end
for $\{u, v\} \in F$ **do**
  $\mathcal{F} \leftarrow \mathcal{F} \cup \{\{u^1, v^2\}, \{u^2, v^1\}\}$;
  $w_{\{u^1 v^2\}} \leftarrow \frac{\tilde{y}^1_u + \tilde{y}^2_u + \tilde{y}^1_v + \tilde{y}^2_v - 1}{2}$;
  $w_{\{u^2 v^1\}} \leftarrow \frac{\tilde{y}^1_u + \tilde{y}^2_u + \tilde{y}^1_v + \tilde{y}^2_v - 1}{2}$;
end
for $u \in U$ **do**
  if *u is not in an odd hole yielding a violation* **then**
    Compute a shortest path P_u between u^1 and u^2 with Dijkstra's algorithm;
    `val`$\leftarrow$the value of this shortest path;
    if `val`$\leq \frac{1}{2} - \Delta_{oh}$ **then**
      Find an odd hole H^u of S_n contained in the odd closed walk of S_n obtained from P_u;
      Return $y^1(H^u) + y^2(H^u) \geq \left\lceil \frac{|H^u|}{2} \right\rceil$. Continue the **for** loop with a vertex $\neq u$;
    end
  end
end

5.3 Finite Capacity Case

In this section we explain how our branch-and-cut algorithm can be adapted to the double TSP with two stacks of capacity q.

We point out that all the constraints and strengthening inequalities described in Section 5.1 can be used in the finite capacity case. Hence we separate all those constraints also in the branch-and-cut algorithm for the double TSP with two stacks of finite capacity, by applying the same separation routines described in Section 5.2.

Our formulation for the finite capacity case differs from the formulation for the infinite capacity case only for the presence of the y-infeasible path constraints (3.14). Since these are exponentially-many, we need to separate them during our algorithm. To ensure the correctness of our algorithm it is only necessary to separate constraints (3.14) on binary solutions to the linear programs obtained during the branch-and-cut algorithm. We show that this task can be performed in polynomial time.

Let $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$ be an optimal solution to a linear program solved during the branch-and-cut algorithm and let us suppose that it is binary. Our separation algorithm for (3.14) works under the assumption that $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$ satisfies all the lifted transitivity constraints. Hence it takes place only after the separation routine for the lifted transitivity constraints described in Section 5.2 and only if this routine returns no violation to (4.3). In this case, $(\tilde{x}^1, \tilde{y}^1)$ and $(\tilde{x}^2, \tilde{y}^2)$ correspond to two Hamiltonian circuits H_1 and H_2. For later convenience, let us assume that $H_1 = 0, P_1, 0$ and that $H_2 = 0, P_2, 0$ for some paths P_1 and P_2 of $G_n \setminus \{0\}$. From now on we consider P_1 and P_2 as sets of arcs.

By solving the knapsack problem (K) of Claim 3.2.11 we test the $2, q$-consistency of (H_1, H_2). If the pair is $2, q$-consistent, no y-infeasible path constraint can be violated by $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$. Otherwise, we know (Proposition 3.2.3) that the y-infeasible path constraint $y^1(P_1) + y^2(P_2) \leq |P_1| + |P_2| - 1$ is violated by $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$.

Since (K) can be solved in $O(n^2)$ time, the separation of the y-infeasible path constraints on a binary point can be performed in $O(n^2)$ provided that $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$ also satisfies (4.3).

It is important to remark that a solution to problem (K) can also certify that (H_1, H_2) is not 2-consistent. In this case one should add to the formulation a violated 2-consistency constraint (3.26) because by minimality it is stronger than $y^1(P_1)+y^2(P_2) \leq |P_1|+|P_2|-1$. For this reason we run the separation routine of the 2-consistency constraints (3.26) before the separation algorithm for the y-infeasible path constraints. Under this assumption, a solution to (K) certifies a violation of a y-infeasible path constraint due to the finiteness of the capacity. Also in this case, it is more convenient to add a violated y-infeasible path constraint associated with minimally $2, q$-infeasible paths. In general, the paths P_1 and P_2 above have not this property. In order to find minimally $2, q$-infeasible paths we proceed as follows. We remove a vertex v from both P_1 and P_2. Then we check, by solving (K), whether the two resulting paths $P_1' = P_1/v$ and $P_2' = P_2/v$ are $2, q$-infeasible. If not, we reinsert v in its original position in both P_1 and P_2 and we repeat the previous procedure on a new vertex. If we do not generate any new infeasible pair of paths, P_1 and P_2 are minimally $2, q$-infeasible. Otherwise by removing one vertex from both P_1 and P_2 we get two new $2, q$-infeasible paths P_1' and P_2'. Now the process can be iterated on these new paths. The whole procedure takes $O(n^4)$ time since it generates at most n pairs of new s, q-infeasible paths from which we remove at most n vertices. Solving (K) to check whether each new pair of paths is $2, q$-infeasible takes $O(n^2)$ time. We resume in Algorithm 7 the separation routine for the y-infeasible path constraints.

Algorithm 7: Separation algorithm for the y-infeasible path inequalities (3.8).

Data: The characteristic point $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$ of a pair of Hamiltonian circuits
Result: A constraint (3.8) violated by $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$

$(H_1, H_2) \leftarrow$Hamiltonian circuits corresponding to $(\tilde{x}^1, \tilde{y}^1)$ and $(\tilde{x}^2, \tilde{y}^2)$;
$P_1 \leftarrow H_1 \setminus \{0\}$;
$P_2 \leftarrow H_2 \setminus \{0\}$;
for $v \in V(P_1)$ **do**
    $P_1' \leftarrow P_1/v$ `// obtained by skipping` v `in` P_1
    $P_2' \leftarrow P_2/v$ `// obtained by skipping` v `in` P_2
    if P_1' *and* P_2' *are* $2, q$*-infeasible* **then**
        $P_1 \leftarrow P_1'$;
        $P_2 \leftarrow P_2'$;
        Go to 4;
    end
end
Return $y^1(P_1) + y^2(P_2) \leq |P_1| + |P_2| - 1$;

5.4 Experimental Results

In this section we present the experimental results obtained with our branch-and-cut algorithm. We provide informations about the implementation details, the instances used for testing the algorithm and finally we show tables of results. The results obtained are useful to draw some conclusions on the effectiveness of families of valid inequalities found in Chapter 4.

5.4.1 Implementation Details

The algorithm has been coded in C++ and uses CPLEX 12.5 [106] as underlying LP solver and branch-and-bound framework. The routines that compute cuts in graphs have been coded using the library LEMON COIN-OR [38].

The code has been compiled using `g++ 4.7.2` and run in a Debian environment on an Intel Core i7-3770 of frequency 3.40GHz and using 8 GB of RAM. The code has been run in sequential mode (1 thread).

We let CPLEX choose automatically the LP solver for both the root and the subproblem nodes of branch-and-cut tree. Since we experienced numerical issues during preliminary test session, we set the Markowitz tolerance of CPLEX to a value of 0.08 and 0.09 depending on the models used in the different tests. Its default value in CPLEX is 0.01. An increased value of the Markowitz tolerance results in numerically stable solutions to the linear program solved by CPLEX.

In Algorithms 2–6 described in Section 5.2 we use Δ values to control the degree of violation of each inequality added to the current linear program. In our final implementation

we set $\Delta_{\texttt{cc}} = \Delta_{\texttt{sc}} = \Delta_{\texttt{tr}} = 0.1$, $\Delta_{\texttt{gddl}} = 0.7$ and $\Delta_{\texttt{oh}} = 0.4$.

In our tests we used the heuristic of Côté *et al.* [45] to get an initial upper bound in the branch-and-cut algorithm. We impose a CPU time limit of three hours to solve each instance.

5.4.2 Instances

The instances used for the tests are taken from the seminal paper on the double TSP with multiple stacks [158] and have been used in all previous works on the double TSP with multiple stacks. We describe them. Each instance is retrieved from two files, one containing the vertices of the pickup city the other containing the vertices of the delivery city. Each vertex is represented by two real coordinates randomly chosen in the 100×100 square centered in $(50, 50)$. The depot of each city is always in the center of the square. The cost of an arc is the Euclidean distance between its endpoints rounded to the nearest integer. Hence the instances are symmetrical. Each file contains 34 vertices, depot included. If we want to deal with k items and customers we limit the computation of the distance to the first k vertices in each file. In our tests we consider two stacks and even values of k from 12 to 16 or to 18, depending on the model used.

5.4.3 Results in the Infinite Capacity Case

We describe the computational results obtained by the branch-and-cut algorithm described in the previous sections for the the double TSP with two stacks of infinite capacity.

In order to assess the impact of the families of strengthening inequalities used in our algorithm we run several tests. For sake of clarity, we call (A1) the branch-and-cut algorithm with no strengthening inequalities at all. This means that only the 2-consistency constraints and the transitivity constraints are separated during the separation step. We call (A2) the branch-and-cut algorithm in which also the odd hole inequalities are separated. That is, in (A2) only the Simple Cut inequalities and the GDDL inequalities are not separated. We call (A3) the algorithm obtained from (A1) by separating also the Simple Cut inequalities and the GDDL inequalities, *i.e.,* only the odd hole inequalities are not separated. Finally, we call (A4) the algorithm in which all families of inequalities are separated. The Markovitz tolerance has been set to 0.08 for the tests based on (A1) and (A2) and to 0.09 for the tests based on (A3) and (A4).

Due to space limits, we present the experimental results obtained by these four algorithms in three tables. Each table reports the performance of two algorithms for an increasing and even number of items starting from 12. Table 5.1 resumes the results of algorithms (A1) and (A2). The largest instances tested with these two algorithms have 16 items. The results of (A3) and (A4) are resumed in Table 5.2 and Table 5.3, respectively for instances with up to 16 items and for instances with 18 or 20 items.

In each table the first two columns are independent of the algorithm version: in column labeled *Instance* we report the instance name, as given in [158], and in column *Items* we

report the number of items in the instance. Hence each line of the table corresponds to exactly one instance of the problem. For each algorithm the remainder of the tables reports the following informations divided into columns:

- column UB: it reports the best upper bound found within the time limit.
- column LB: it reports the best lower bound found within the time limit.
- column Time: it reports the CPU time in seconds spent to solve to optimality the instance. Since this is not always the case, a value of 10800 seconds indicates that the time limit has been reached without a proof of optimality.
- column Nodes: it reports the number of nodes in the branch-and-cut tree.
- column Gap: it reports the gap in percentage between the values of columns UB and LB in the same line. The value of this column is calculated as $100 \cdot (UB - LB)/UB$. Gap values are highlighted in boldface if they are 0 *i.e.,* if the corresponding instances have been solved to optimality.

The only exception is in Table 5.2 where the column Gap is not present since the algorithms (A3) and (A4) solve all the tested instances to optimality within the time limit. In all tables we also report, in the lines labeled *Average*, the average values of columns Time, Nodes and Gap. The average is calculated over the instances having the same number of items.

Table 5.1 and Table 5.2 show that all the four algorithms solve to optimality all the instances with 12 and 14 items within the time limit. Concerning the instances with 16 items, only five of them are solved to optimality by (A1) and only four of them are solved to optimality by algorithm (A2). Algorithms (A3) and (A4) solve all the instances with 16 items.

Table 5.3 shows that both (A3) and (A4) are able to solve 11 out of the 20 instances with 18 items within the time limit. Both algorithms solve the same instances with 18 items. Finally, instance R13 with 20 items is solved to optimality within the time limit by both (A3) and (A4). Algorithm (A3) additionally solves instance R09 with 20 items.

We now compare the four algorithms. Looking at the average running times of (A1) and (A2) reported in Table 5.1 we conclude that both algorithms exhibit similar performances, with (A2) slightly slower in average for all the considered numbers of items in the instances. Recall that (A2) differs from (A1) for the presence of the odd hole inequalities. To see on which instances (A2) is faster than (A1), we highlighted in boldface those running times obtained by (A2) that are at least 10% smaller than the corresponding running times obtained by (A1). Table 5.1 shows that (A2) is faster than (A1) on six out of the 20 instances with 12 items and on five instances with 14 items. With 16 items (A2) is faster than (A1) only on instances R01 and R06 (this latter is not solved to optimality by algorithm (A1)). The average gap of (A1) is smaller than the average gap of (A2) on the instances with 16 items.

Table 5.1: Results of (A1) and (A2) on instances with up to 16 items and stacks of infinite capacity.

		(A1)					(A2)				
Instance	Items	UB	LB	Time	Nodes	Gap	UB	LB	Time	Nodes	Gap
R00	12	716	716,00	21,56	595	0,00	716	716,00	36,04	887	**0,00**
R01	12	741	741,00	15,31	471	0,00	741	741,00	15,87	471	**0,00**
R02	12	651	651,00	72,69	2098	0,00	651	651,00	69,16	1787	**0,00**
R03	12	690	690,00	4,84	133	0,00	690	690,00	**3,54**	89	**0,00**
R04	12	659	659,00	46,95	1405	0,00	659	659,00	68,82	1971	**0,00**
R05	12	627	627,00	180,04	5808	0,00	627	627,00	206,41	5916	**0,00**
R06	12	789	789,00	14,58	488	0,00	789	789,00	14,53	401	**0,00**
R07	12	589	589,00	111,70	3744	0,00	589	589,00	150,58	4241	**0,00**
R08	12	749	749,00	67,09	1901	0,00	749	749,00	87,49	2349	**0,00**
R09	12	686	686,00	8,43	273	0,00	686	686,00	**6,38**	200	**0,00**
R10	12	663	663,00	170,86	5493	0,00	663	663,00	**124,85**	3666	**0,00**
R11	12	622	622,00	26,89	794	0,00	622	622,00	**21,07**	568	**0,00**
R12	12	741	741,00	14,12	469	0,00	741	741,00	18,26	557	**0,00**
R13	12	683	683,00	10,94	323	0,00	683	683,00	**9,65**	235	**0,00**
R14	12	680	680,00	127,85	4090	0,00	680	680,00	151,54	4301	**0,00**
R15	12	624	624,00	67,52	2206	0,00	624	624,00	85,32	2427	**0,00**
R16	12	610	610,00	22,07	557	0,00	610	610,00	**18,79**	436	**0,00**
R17	12	780	780,00	264,27	7298	0,00	780	780,00	253,38	6529	**0,00**
R18	12	735	735,00	6,63	227	0,00	735	735,00	5,99	170	**0,00**
R19	12	782	782,00	38,92	1117	0,00	782	782,00	60,70	1599	**0,00**
Average				*64,66*	*1974,50*	*0,00*			*70,42*	*1940,00*	*0,00*
R00	14	766	766,00	328,65	5786	0,00	766	766,00	475,96	8011	**0,00**
R01	14	761	761,00	205,69	4088	0,00	761	761,00	589,56	10776	**0,00**
R02	14	690	690,00	117,58	2234	0,00	690	690,00	143,16	2500	**0,00**
R03	14	791	791,00	188,99	3635	0,00	791	791,00	**90,05**	1590	**0,00**
R04	14	756	756,00	2744,00	47198	0,00	756	756,00	3081,46	50116	**0,00**
R05	14	773	773,00	3402,97	64479	0,00	773	773,00	3440,48	60995	**0,00**
R06	14	811	811,00	131,76	2601	0,00	811	811,00	**92,26**	1616	**0,00**
R07	14	693	693,00	744,80	14224	0,00	693	693,00	994,74	18779	**0,00**
R08	14	824	824,00	753,88	13351	0,00	824	824,00	776,23	13043	**0,00**
R09	14	733	733,00	27,90	655	0,00	733	733,00	26,62	491	**0,00**
R10	14	733	733,00	2108,38	39234	0,00	733	733,00	3200,05	56914	**0,00**
R11	14	719	719,00	704,38	13181	0,00	719	719,00	883,69	15617	**0,00**
R12	14	803	803,00	494,62	8819	0,00	803	803,00	**228,78**	3903	**0,00**
R13	14	743	743,00	978,18	17743	0,00	743	743,00	999,01	17418	**0,00**
R14	14	747	747,00	4322,62	79610	0,00	747	747,00	4780,96	84509	**0,00**
R15	14	765	765,00	249,85	4525	0,00	765	765,00	445,94	7459	**0,00**
R16	14	685	685,00	191,61	3585	0,00	685	685,00	**137,87**	2287	**0,00**
R17	14	818	818,00	1607,63	28627	0,00	818	818,00	1698,15	28615	**0,00**
R18	14	774	774,00	172,45	3360	0,00	774	774,00	212,07	3734	**0,00**
R19	14	833	833,00	2184,96	39776	0,00	833	833,00	**1002,02**	16302	**0,00**
Average				*1083,05*	*19835,55*	*0,00*			*1164,95*	*20233,75*	*0,00*
R00	16	804	759,12	10800,00	102463	5,58	795	763,00	10800,00	98707	4,03
R01	16	794	794,00	1610,67	19886	**0,00**	794	794,00	**1237,40**	13813	**0,00**
R02	16	752	702,00	10800,00	95493	6,65	752	694,50	10800,00	84208	7,65
R03	16	855	790,00	10800,00	101799	7,60	855	796,67	10800,00	94340	6,82
R04	16	792	792,00	4655,07	46526	**0,00**	792	792,00	4665,08	43764	**0,00**
R05	16	823	760,27	10800,00	103347	7,62	823	759,42	10800,00	97171	7,73
R06	16	900	898,58	10800,00	128224	0,16	900	900,00	**8815,20**	104367	**0,00**
R07	16	756	756,00	9120,89	113688	**0,00**	756	750,75	10800,00	122900	0,69
R08	16	909	866,50	10800,00	90976	4,68	908	868,94	10800,00	83127	4,30
R09	16	796	796,00	375,14	4737	**0,00**	796	796,00	455,08	5555	**0,00**
R10	16	755	706,58	10800,00	98524	6,41	755	707,50	10800,00	93244	6,29
R11	16	759	759,00	5194,92	62263	**0,00**	769	754,50	10800,00	112312	1,89
R12	16	825	808,67	10800,00	103212	1,98	825	804,00	10800,00	93500	2,55
R13	16	824	818,16	10800,00	125047	0,71	824	815,50	10800,00	116766	1,03
R14	16	823	728,06	10800,00	97766	11,54	823	724,33	10800,00	90494	11,99
R15	16	807	777,75	10800,00	101719	3,62	807	765,62	10800,00	93402	5,13
R16	16	781	762,00	10800,00	112255	2,43	781	766,83	10800,00	111309	1,81
R17	16	857	812,72	10800,00	105049	5,17	852	807,00	10800,00	95960	5,28
R18	16	846	826,73	10800,00	107810	2,28	846	816,00	10800,00	97441	3,55
R19	16	882	863,27	10800,00	108122	2,12	882	858,29	10800,00	97295	2,69
Average				*9147,83*	*91445,30*	*3,43*			*9398,64*	*87483,75*	*3,67*

Table 5.2: Results of (A3) and (A4) on instances with up to 16 items and stacks of infinite capacity.

				(A3)		(A4)	
Instance	Items	UB	LB	Time	Nodes	Time	Nodes
R00	12	716	716,00	6,75	115	12,19	193
R01	12	741	741,00	5,29	109	5,05	79
R02	12	651	651,00	31,09	661	**24,3**	441
R03	12	690	690,00	1,66	24	2,02	14
R04	12	659	659,00	13,95	281	15,24	297
R05	12	627	627,00	19,01	353	23,83	474
R06	12	789	789,00	5,22	89	6,73	114
R07	12	589	589,00	7,11	128	9,21	139
R08	12	749	749,00	35,06	787	**20,84**	418
R09	12	686	686,00	3,46	75	4,79	72
R10	12	663	663,00	5,17	108	5,26	96
R11	12	622	622,00	11,20	243	15,52	298
R12	12	741	741,00	2,66	41	2,99	35
R13	12	683	683,00	1,92	28	4,5	61
R14	12	680	680,00	3,69	65	4,1	59
R15	12	624	624,00	14,65	297	17,5	331
R16	12	610	610,00	9,97	216	**6,35**	107
R17	12	780	780,00	27,18	546	25,29	477
R18	12	735	735,00	3,95	93	3,65	66
R19	12	782	782,00	15,12	317	25,02	472
Average				*11,21*	*228,80*	*11,72*	*212,15*
R00	14	766	766,00	112,78	1431	161,6	1924
R01	14	761	761,00	26,49	304	**22,23**	219
R02	14	690	690,00	56,59	714	51,23	567
R03	14	791	791,00	75,85	1116	**24,67**	312
R04	14	756	756,00	401,77	5208	697,29	8983
R05	14	773	773,00	127,48	1536	**81,49**	896
R06	14	811	811,00	16,68	155	19,7	194
R07	14	693	693,00	27,50	341	**23,5**	238
R08	14	824	824,00	297,97	4081	288,75	3750
R09	14	733	733,00	9,16	82	9,58	63
R10	14	733	733,00	98,83	1428	**72,99**	950
R11	14	719	719,00	206,65	2865	209,36	2700
R12	14	803	803,00	95,39	1239	**58,34**	685
R13	14	743	743,00	22,01	265	36,12	429
R14	14	747	747,00	177,59	2301	**136,17**	1565
R15	14	765	765,00	22,31	245	22,08	209
R16	14	685	685,00	33,97	449	**27,39**	308
R17	14	818	818,00	110,51	1293	122,7	1372
R18	14	774	774,00	85,21	1256	78,98	1151
R19	14	833	833,00	240,87	3164	**142,41**	1613
Average				*112,28*	*1473,65*	*114,33*	*1406,40*
R00	16	795	795,00	1414,19	10685	1783,13	13298
R01	16	794	794,00	120,46	872	**97,56**	642
R02	16	752	752,00	5738,98	45042	5427,46	40629
R03	16	855	855,00	2487,10	19324	3521,08	26700
R04	16	792	792,00	2676,19	21745	2707,43	20325
R05	16	820	820,00	995,85	7741	1882,26	13849
R06	16	900	900,00	668,24	5403	649,76	4862
R07	16	756	756,00	166,54	1276	**98,1**	725
R08	16	907	907,00	1408,09	11761	**1214,73**	10178
R09	16	796	796,00	53,16	427	85,89	697
R10	16	755	755,00	424,26	3443	412,44	3250
R11	16	759	759,00	760,43	6547	793,8	5841
R12	16	825	825,00	643,64	5235	**522,88**	4136
R13	16	824	824,00	408,69	3258	389,07	2809
R14	16	823	823,00	3140,47	24722	5288,23	39818
R15	16	807	807,00	406,53	3091	916,95	6824
R16	16	781	781,00	1075,12	9035	**472,86**	3864
R17	16	852	852,00	2309,26	18161	**1704,2**	12833
R18	16	846	846,00	1390,37	11996	**1044,22**	9193
R19	16	882	882,00	1299,35	9810	**1044,82**	7670
Average				*1379,35*	*10978,70*	*1502,84*	*11407,15*

Table 5.3: Results of (A3) and (A4) on instances with 18 and 20 items and stacks of infinite capacity.

Instance	Items	(A3)					(A4)				
		UB	LB	Time	Nodes	Gap	UB	LB	Time	Nodes	Gap
R00	18	839	839,00	3488,88	17894	**0,00**	839	839,00	3972,65	20517	**0,00**
R01	18	825	825,00	791,71	3496	**0,00**	825	825,00	**694,46**	3415	**0,00**
R02	18	793	753,02	10800,00	46344	5,04	793	758,21	10800,00	48600	4,39
R03	18	899	851,68	10800,00	43227	5,26	899	859,00	10800,00	42919	4,45
R04	18	832	785,67	10800,00	43162	5,57	832	783,90	10800,00	42166	5,78
R05	18	873	873,00	8984,39	48919	**0,00**	873	873,00	9181,43	47196	**0,00**
R06	18	930	930,00	8001,76	43377	**0,00**	930	930,00	**3622,55**	18763	**0,00**
R07	18	805	805,00	1048,62	5653	**0,00**	805	805,00	2700,83	14181	**0,00**
R08	18	959	919,44	10800,00	47286	4,13	962	923,44	10800,00	48645	4,01
R09	18	815	815,00	151,61	750	**0,00**	815	815,00	187,73	941	**0,00**
R10	18	856	830,15	10800,00	49487	3,02	856	832,77	10800,00	47435	2,71
R11	18	823	787,69	10800,00	51464	4,29	813	791,97	10800,00	53444	2,59
R12	18	871	871,00	3982,70	20864	**0,00**	871	871,00	**2245,87**	11038	**0,00**
R13	18	845	845,00	2185,50	10921	**0,00**	845	845,00	**1791,80**	8940	**0,00**
R14	18	874	827,28	10800,00	44578	5,35	866	828,97	10800,00	45528	4,28
R15	18	868	838,87	10800,00	46897	3,36	869	836,34	10800,00	47538	3,76
R16	18	811	811,00	5787,25	31662	**0,00**	811	811,00	**3297,92**	16891	**0,00**
R17	18	900	859,24	10800,00	44310	4,53	900	859,86	10800,00	43181	4,46
R18	18	883	883,00	10274,22	52388	**0,00**	883	883,00	10700,68	55468	**0,00**
R19	18	909	909,00	5978,36	29985	**0,00**	909	909,00	6697,51	30803	**0,00**
Average				*7393,75*	*34133,20*	*2,03*			*7114,67*	*32380,45*	*1,82*
R00	20	879	824,25	10800,00	27952	6,23	879	820,72	10800,00	28855	6,63
R01	20	879	859,94	10800,00	31929	2,17	879	863,72	10800,00	33700	1,74
R02	20	830	759,19	10800,00	31140	8,53	830	757,18	10800,00	31900	8,77
R03	20	951	843,89	10800,00	29599	11,26	951	854,24	10800,00	28160	10,17
R04	20	874	787,50	10800,00	27808	9,90	874	786,65	10800,00	28117	9,99
R05	20	917	863,00	10800,00	31686	5,89	917	864,57	10800,00	31312	5,72
R06	20	978	927,12	10800,00	33000	5,20	971	936,67	10800,00	34317	3,54
R07	20	916	843,29	10800,00	30460	7,94	920	841,65	10800,00	30403	8,52
R08	20	979	894,18	10800,00	30317	8,66	979	898,75	10800,00	30534	8,20
R09	20	888	888,00	10164,86	36150	**0,00**	888	861,04	10800,00	34360	3,04
R10	20	945	847,52	10800,00	30437	10,32	945	853,53	10800,00	29088	9,68
R11	20	846	786,19	10800,00	33091	7,07	846	785,00	10800,00	30834	7,21
R12	20	922	863,16	10800,00	30141	6,38	925	865,73	10800,00	29607	6,41
R13	20	888	888,00	5817,28	19704	**0,00**	888	888,00	7385,40	24509	**0,00**
R14	20	921	832,09	10800,00	29971	9,65	921	841,43	10800,00	28138	8,64
R15	20	947	855,85	10800,00	26330	9,62	947	858,13	10800,00	25509	9,38
R16	20	869	837,27	10800,00	32738	3,65	869	822,18	10800,00	32574	5,39
R17	20	928	869,69	10800,00	28961	6,28	928	865,15	10800,00	29685	6,77
R18	20	928	855,17	10800,00	32691	7,85	928	865,62	10800,00	32442	6,72
R19	20	969	893,55	10800,00	29667	7,79	969	895,98	10800,00	28464	7,54
Average				*10519,11*	*30188,60*	*6,72*			*10629,27*	*30125,40*	*6,70*

We now compare the average running times reported in Table 5.1 with the average running times of (A3) in Table 5.2. We see that (A3) is roughly six times faster than (A1) and (A2) on instances with 12 items, ten times faster on instances with 14 items and seven times faster in the case of 16 items. Similar ratios are observed by comparing the average running times of (A4) with those of (A1) and (A2). We also see that the algorithms in Table 5.2 enumerate considerably less branch-and-cut nodes than the algorithms in Table 5.1.

We finally compare our best algorithms (A3) and (A4). In Table 5.2 we observe that in average the running times of (A3) and (A4) are very close on the instances with 12 and 14 items, but (A3) is faster than (A4) on the instances with 16 items. Additionally, we recall that both (A3) and (A4) solve to optimality the same set of 11 instances with 18 items. Table 5.3 shows that (A4) is faster than (A3) in average on these instances. We already observed that (A3) solves to optimality just two instances with 20 items, one more than (A4). Hence a comparison of the running times of (A3) and (A4) on these instances would not be very telling.

Please, remember that (A4) differs from (A3) for the presence of the odd hole inequalities. In Table 5.2 and Table 5.3 we highlight in boldface the running times of (A4) that are at least 10% less than the corresponding running times of (A3). We see that, if we do not consider the easy instances with 12 items, the presence of the odd hole inequalities improves the running times in roughly the 40% of the instances solved by both algorithms.

The number of branch-and-cut nodes enumerated by both algorithms is always comparable. Finally, for the unsolved instances in Table 5.3 we see that the average gap obtained by (A4) is smaller than the average gap obtained by (A3) in both the cases of instances with 18 and 20 items.

From the analysis of Tables 5.1–5.3 we draw several conclusions. The computational results of algorithm (A1) indicate that our formulation can be used to solve easily instances of the double TSP with two stacks of infinite capacity with up to 14 items.

In order to increase the size of the instances solved with our approach, our analysis clearly shows that the routing inequalities have to be included in the branch-and-cut algorithm.

From a running time point of view the odd hole inequalities did not show a clear beneficial impact in average. However, we have observed that they are able to speed up the algorithm on several instances and for every instance size considered in this test session. In addition, the odd hole inequalities help in reducing the optimality gap for the unsolved instances.

Finally, we point out that our algorithm (A3) solves to optimality 73 out of the 100 tested instances within three hours of computational time. The heuristic of Côté *et al.* [45] yields the optimal value of 41 instances out of the 73 solved to optimality by (A3). The largest instances solved to optimality with our approach have 20 items.

5.4.4 Results in the Finite Capacity Case

The existing exact algorithm for the double TSP with multiple stacks consider the case in which the stacks have a finite capacity [16, 30, 44, 109, 131, 132, 2, 157]. In this section we report the computational results obtained by our algorithm for the double TSP with two stacks of finite capacity.

The study presented in this section is the starting point for several discussions. Firstly, it lets us assess the quality of the strengthening cuts in the finite capacity case. It also clarifies the contribution of our approach with respect to the existing computational studies on the double TSP with two stacks of finite capacity. Finally, it enables a comparison between the results obtained in the finite capacity case with those obtained in the infinite capacity case.

To this end, we adapt the four versions of the algorithm described in Section 5.4.3 by including also the separation routine of the y-infeasible path constraints described in Section 5.3. In particular, the y-infeasible path constraints are separated only when the current solution is binary and yielded no violation to the transitivity and 2-consistency constraints.

For the sake of clarity we define the four "capacitated" versions of the algorithm schematically as follows:

- (CA1): it is the branch-and-cut algorithm with no strengthening cuts at all *i.e.,* only the 2-consistency, transitivity and y-infeasible path constraints are separated during the separation step;
- (CA2): with respect to (CA1), the odd hole inequalities are also separated. That is, only the Simple Cut and the GDDL inequalities are not separated;
- (CA3): with respect to (CA1), the Simple Cut and the GDDL inequalities are also separated, hence only the odd hole inequalities are not separated;
- (CA4): all strengthening cuts are separated.

The results obtained by Algorithms (CA1)–(CA4) are collected in Tables 5.4–5.6. The informations reported in such tables are organized as described in Section 5.4.3.

We proceed now to the discussion of the computational results. From Table 5.4 we see that both (CA1) and (CA2) solve to optimality all the instances with 12 items and all instances but instance R14 with 14 items. Instead both algorithms mostly fail when passing to instances with 16 items.

Table 5.5 shows that (CA3) and (CA4) are able to solve all instances with up to 16 items within the time limit. In addition, as reported in Table 5.6, both (CA3) and (CA4) solve the same set of 11 instances out of the 20 instances with 18 items. Finally, we observe that (CA3) solve to optimality only instance R09 with 20 items whereas no instance with 20 items is solved within the time limit by (CA4).

As in the infinite capacity case, we see that using only the odd hole inequalities as strengthening cuts does not reduce, in general, the running time of the algorithm. Indeed,

in Table 5.4 Algorithm (CA1) appears slightly faster than (CA2). In addition, (CA1) solves one instance with 16 items more than (CA2). We also observe that the inclusion of the odd hole inequalities reduces the optimality gap in six out of the 17 instances not solved to optimality by (CA1).

As was the case for stacks of infinite capacity, one gets significantly better performances by including the Simple Cut and the GDDL inequalities as strengthening cuts. Indeed, the comparison between the computational results in Table 5.4 and Table 5.5 shows that both (CA3) and (CA4) are superior to (CA1) and (CA2) in terms of running time and number of branch-and-cut nodes.

Algorithms (CA3) and (CA4) exhibit very similar average running times for the instances with 12 and 14 items, as shown in Table 5.5. On instances with 16 items (CA3) is faster than (CA4) in average. Conversely, Table 5.6 reveals that (CA4) reaches optimal solutions of the solved instances with 18 items in less time than (CA3). Also, the optimality gap for the unsolved instances with 18 items is oftentimes smaller when using (CA4). In Table 5.5 and Table 5.6, we highlighted in boldface those running times obtained by (CA4) that are at least 10% smaller than the corresponding running times obtained by (CA3). Then, if we consider the instances with at least 14 items, we get that (CA4) improves of at least the 10% the running times of (CA3) in roughly the 42% of the cases. Finally, we observe that, with respect to (CA3), Algorithm (CA4) reduces the optimality gap in seven out of the nine unsolved instances with 18 items.

We now compare the results described in this section with those obtained in the existing literature on the double TSP with two stacks of finite capacity. As far as we know, the best computational results for this problem have been obtained by Alba Martínez *et al.* in [2]. The code of [2] has been implemented in C++ using CPLEX 12 and tested with a 3GHz Intel Core 2 Duo processor, with a CPU time limit of three hours. In [2] optimal solutions for instances with up to 14 items are reported.

In Table 5.7 we compare our best algorithms (CA3) and (CA4) with the algorithm of [2]. The lines of Table 5.7 correspond to the instances identified by their name and the number of items, as reported in the first two columns. We have three additional columns each corresponding to one algorithm. For every instance these three columns contain the CPU times of the corresponding algorithms. We highlighted in boldface those running times of (CA3) and (CA4) that are smaller than the corresponding running times of the algorithm of [2].

We see that both (CA3) and (CA4) are faster than the algorithm of [2] in 23 out of the 40 considered instances. In addition, the average running time of (CA3) and (CA4) is considerably smaller for the instances with 14 items. We explain this latter result by observing that the algorithm of [2] has high running times on difficult instances with 14 items, such as instances R04, R08, R11, R14 and R17, whereas (CA3) and (CA4) exhibit more uniform performances. From the discussion above we conclude that our approach improves the existing computational results of the exact algorithms for the double TSP with two stacks of finite capacity.

Table 5.4: Results of (CA1) and (CA2) on instances with up to 16 items and stacks of finite capacity.

		(CA1)					(CA2)				
Instance	Items	UB	LB	Time	Nodes	Gap	UB	LB	Time	Nodes	Gap
R00	12	726	726,00	37,55	1086	0,00	726	726,00	45,66	1204	**0,00**
R01	12	741	741,00	15,27	471	0,00	741	741,00	15,91	471	**0,00**
R02	12	660	660,00	128,75	3681	0,00	660	660,00	**110,77**	2901	**0,00**
R03	12	690	690,00	4,85	133	0,00	690	690,00	**3,55**	89	**0,00**
R04	12	659	659,00	47,08	1405	0,00	659	659,00	69,01	1971	**0,00**
R05	12	631	631,00	208,82	6810	0,00	631	631,00	250,78	7188	**0,00**
R06	12	793	793,00	18,04	597	0,00	793	793,00	17,99	493	**0,00**
R07	12	593	593,00	128,05	4353	0,00	593	593,00	167,41	4716	**0,00**
R08	12	749	749,00	67,22	1901	0,00	749	749,00	87,74	2349	**0,00**
R09	12	692	692,00	11,71	397	0,00	692	692,00	12,58	402	**0,00**
R10	12	663	663,00	170,67	5493	0,00	663	663,00	**125,59**	3666	**0,00**
R11	12	625	625,00	33,18	1007	0,00	625	625,00	**28,07**	749	**0,00**
R12	12	741	741,00	14,12	469	0,00	741	741,00	18,28	557	**0,00**
R13	12	694	694,00	16,15	486	0,00	694	694,00	20,87	532	**0,00**
R14	12	680	680,00	127,90	4090	0,00	680	680,00	152,15	4301	**0,00**
R15	12	628	628,00	77,84	2537	0,00	628	628,00	91,94	2666	**0,00**
R16	12	610	610,00	22,11	557	0,00	610	610,00	**18,84**	436	**0,00**
R17	12	780	780,00	264,64	7298	0,00	780	780,00	254,53	6529	**0,00**
R18	12	735	735,00	6,63	227	0,00	735	735,00	6,01	170	**0,00**
R19	12	789	789,00	66,35	1979	0,00	789	789,00	75,78	2080	**0,00**
Average				*73,35*	*2248,85*	*0,00*			*78,67*	*2173,50*	*0,00*
R00	14	774	774,00	439,66	8051	0,00	774	774,00	602,09	10469	**0,00**
R01	14	761	761,00	205,94	4088	0,00	761	761,00	592,49	10776	**0,00**
R02	14	690	690,00	117,72	2234	0,00	690	690,00	143,96	2500	**0,00**
R03	14	791	791,00	189,16	3635	0,00	791	791,00	**90,35**	1590	**0,00**
R04	14	756	756,00	2748,03	47198	0,00	756	756,00	3089,76	50116	**0,00**
R05	14	775	775,00	3429,03	65560	0,00	775	775,00	3777,32	65715	**0,00**
R06	14	824	824,00	190,40	4105	0,00	824	824,00	188,75	3539	**0,00**
R07	14	697	697,00	896,31	17412	0,00	697	697,00	1260,95	23721	**0,00**
R08	14	824	824,00	754,41	13351	0,00	824	824,00	779,01	13043	**0,00**
R09	14	739	739,00	53,92	1255	0,00	739	739,00	**46,47**	901	**0,00**
R10	14	733	733,00	2112,82	39234	0,00	733	733,00	3211,25	56914	**0,00**
R11	14	725	725,00	989,19	18623	0,00	725	725,00	1129,00	20288	**0,00**
R12	14	803	803,00	495,29	8819	0,00	803	803,00	**229,27**	3903	**0,00**
R13	14	746	746,00	1100,57	20052	0,00	746	746,00	1206,87	20980	**0,00**
R14	14	765	745,00	10800,00	173328	2,61	765	750,40	10800,00	175262	1,91
R15	14	765	765,00	250,34	4525	0,00	765	765,00	446,45	7459	**0,00**
R16	14	685	685,00	192,04	3585	0,00	685	685,00	**138,15**	2287	**0,00**
R17	14	818	818,00	1609,22	28627	0,00	818	818,00	1704,45	28615	**0,00**
R18	14	774	774,00	172,51	3360	0,00	774	774,00	213,38	3734	**0,00**
R19	14	836	836,00	2696,59	49004	0,00	836	836,00	1280,37	20768	**0,00**
Average				*1472,16*	*25802,30*	*0,13*			*1546,52*	*26129,00*	*0,10*
R00	16	804	759,14	10800,00	102542	5,58	804	761,06	10800,00	96488	5,34
R01	16	794	794,00	1614,41	19886	**0,00**	794	794,00	**1238,08**	13813	**0,00**
R02	16	752	702,00	10800,00	95513	6,65	752	694,50	10800,00	84266	7,65
R03	16	855	790,00	10800,00	100850	7,60	855	796,33	10800,00	92645	6,86
R04	16	801	801,00	9139,49	91741	**0,00**	801	801,00	8517,75	81307	**0,00**
R05	16	823	760,29	10800,00	103342	7,62	823	759,40	10800,00	97199	7,73
R06	16	906	891,72	10800,00	118573	1,58	906	893,54	10800,00	116759	1,37
R07	16	756	756,00	9116,20	113688	**0,00**	756	750,83	10800,00	123042	0,68
R08	16	909	866,50	10800,00	90806	4,68	909	868,67	10800,00	82935	4,44
R09	16	800	800,00	521,78	6792	**0,00**	800	800,00	524,97	6540	**0,00**
R10	16	755	706,58	10800,00	98501	6,41	755	707,50	10800,00	93264	6,29
R11	16	777	754,16	10800,00	112459	2,94	777	750,00	10800,00	104998	3,47
R12	16	825	808,67	10800,00	103184	1,98	825	804,00	10800,00	93470	2,55
R13	16	831	814,00	10800,00	116390	2,05	831	808,76	10800,00	106639	2,68
R14	16	823	728,00	10800,00	97544	11,54	823	724,33	10800,00	90475	11,99
R15	16	807	777,75	10800,00	101708	3,62	807	765,62	10800,00	93377	5,13
R16	16	781	762,01	10800,00	113125	2,43	781	766,93	10800,00	111656	1,80
R17	16	858	812,50	10800,00	104629	5,30	858	806,34	10800,00	95232	6,02
R18	16	846	826,71	10800,00	107772	2,28	846	816,00	10800,00	97525	3,55
R19	16	882	863,31	10800,00	108247	2,12	882	858,25	10800,00	97041	2,69
Average				*9659,59*	*95364,60*	*3,72*			*9694,04*	*88933,55*	*4,01*

Table 5.5: Results of (CA3) and (CA4) on instances with up to 16 items and stacks of finite capacity.

				(CA3)		(CA4)	
Instance	Items	UB	LB	Time	Nodes	Time	Nodes
R00	12	726	726,00	15,12	299	19,27	351
R01	12	741	741,00	5,37	109	5,06	79
R02	12	660	660,00	53,03	1134	**37,39**	709
R03	12	690	690,00	1,69	24	2,02	14
R04	12	659	659,00	14,18	281	15,25	297
R05	12	631	631,00	28,41	559	29,08	596
R06	12	793	793,00	6,64	120	7,93	136
R07	12	593	593,00	7,75	139	10,34	165
R08	12	749	749,00	35,69	787	**20,87**	418
R09	12	692	692,00	4,87	122	8,42	157
R10	12	663	663,00	5,23	108	5,29	96
R11	12	625	625,00	12,83	273	16,54	322
R12	12	741	741,00	2,72	41	3,00	35
R13	12	694	694,00	3,70	68	5,07	83
R14	12	680	680,00	3,75	65	4,12	59
R15	12	628	628,00	16,53	341	17,95	356
R16	12	610	610,00	10,12	216	**6,37**	107
R17	12	780	780,00	27,65	546	25,44	477
R18	12	735	735,00	4,02	93	3,66	66
R19	12	789	789,00	26,46	558	33,68	649
Average				*14,29*	*294,15*	*13,84*	*258,60*
R00	14	774	774,00	152,69	1979	247,86	3032
R01	14	761	761,00	27,29	304	**22,25**	219
R02	14	690	690,00	58,12	714	**51,24**	567
R03	14	791	791,00	77,95	1116	**24,71**	312
R04	14	756	756,00	412,28	5208	698,79	8983
R05	14	775	775,00	141,05	1674	**95,55**	1078
R06	14	824	824,00	30,83	346	27,91	333
R07	14	697	697,00	30,42	380	**27,03**	291
R08	14	824	824,00	305,95	4081	289,18	3750
R09	14	739	739,00	14,92	166	15,12	154
R10	14	733	733,00	101,40	1428	**73,53**	950
R11	14	725	725,00	319,65	4370	305,59	3917
R12	14	803	803,00	97,67	1239	**58,74**	685
R13	14	746	746,00	25,93	306	40,47	480
R14	14	765	765,00	516,63	6245	**395,94**	4688
R15	14	765	765,00	22,87	245	22,26	209
R16	14	685	685,00	34,84	449	**27,54**	308
R17	14	818	818,00	113,29	1293	123,65	1372
R18	14	774	774,00	87,33	1256	79,56	1151
R19	14	836	836,00	274,21	3476	**177,28**	2045
Average				*142,27*	*1813,75*	*140,21*	*1726,20*
R00	16	804	804,00	2424,13	18632	3046,54	22601
R01	16	794	794,00	120,06	872	**97,56**	642
R02	16	752	752,00	5724,79	45042	5412,28	40629
R03	16	855	855,00	2474,61	19324	3532,05	26700
R04	16	801	801,00	4789,36	38639	5102,74	38661
R05	16	823	823,00	1255,92	9752	2175,92	16069
R06	16	906	906,00	846,31	6941	817,02	6164
R07	16	756	756,00	166,15	1276	**98,41**	725
R08	16	909	909,00	1684,53	13790	**1359,68**	11359
R09	16	800	800,00	65,90	554	102,12	855
R10	16	755	755,00	423,26	3443	412,80	3250
R11	16	777	777,00	3343,26	27766	**2135,59**	16776
R12	16	825	825,00	643,16	5235	**523,26**	4136
R13	16	831	831,00	499,17	4098	464,15	3464
R14	16	823	823,00	3135,90	24722	5316,48	39818
R15	16	807	807,00	405,52	3091	920,19	6824
R16	16	781	781,00	1072,41	9035	**473,23**	3864
R17	16	858	858,00	3223,62	25471	**2547,03**	19180
R18	16	846	846,00	1377,10	11996	**1053,68**	9193
R19	16	882	882,00	1286,18	9810	**1058,20**	7670
Average				*1748,07*	*13974,45*	*1832,45*	*13929,00*

Table 5.6: Results of (CA3) and (CA4) on instances with 18 and 20 items and stacks of finite capacity.

Instance	Items	(CA3)					(CA4)				
		UB	LB	Time	Nodes	Gap	UB	LB	Time	Nodes	Gap
R00	18	839	839,00	3441,34	17894	**0,00**	839	839,00	3979,04	20517	**0,00**
R01	18	857	857,00	4379,60	22658	**0,00**	857	857,00	4824,54	25717	**0,00**
R02	18	793	753,32	10800,00	47100	5,00	793	758,28	10800,00	48738	4,38
R03	18	899	851,72	10800,00	43348	5,26	899	859,00	10800,00	42938	4,45
R04	18	832	785,77	10800,00	43383	5,56	832	784,00	10800,00	42528	5,77
R05	18	873	873,00	8944,67	48919	**0,00**	873	873,00	9074,50	47196	**0,00**
R06	18	930	930,00	7886,89	43377	**0,00**	930	930,00	**3571,23**	18763	**0,00**
R07	18	805	805,00	1035,03	5653	**0,00**	805	805,00	2664,87	14181	**0,00**
R08	18	962	919,42	10800,00	47888	4,43	962	923,59	10800,00	49173	3,99
R09	18	815	815,00	150,07	750	**0,00**	815	815,00	185,57	941	**0,00**
R10	18	856	830,51	10800,00	50350	2,98	856	833,07	10800,00	48204	2,68
R11	18	823	787,95	10800,00	52500	4,26	823	788,14	10800,00	51696	4,24
R12	18	871	871,00	3928,58	20864	**0,00**	871	871,00	**2211,18**	11038	**0,00**
R13	18	860	860,00	3542,27	19592	**0,00**	860	860,00	4120,56	21969	**0,00**
R14	18	874	827,53	10800,00	45315	5,32	874	828,90	10800,00	46280	5,16
R15	18	869	838,92	10800,00	47466	3,46	869	836,61	10800,00	48230	3,73
R16	18	819	819,00	9823,54	55071	**0,00**	819	819,00	**5112,37**	27047	**0,00**
R17	18	900	859,50	10800,00	45200	4,50	900	860,07	10800,00	43971	4,44
R18	18	883	883,00	10127,94	52388	**0,00**	883	883,00	10553,73	55475	**0,00**
R19	18	909	909,00	5956,23	29985	**0,00**	909	909,00	6598,92	30803	**0,00**
Average				*7820,81*	*36985,05*	*2,04*			*7504,83*	*34770,25*	*1,94*
R00	20	879	824,28	10800,00	28009	6,22	879	820,72	10800,00	28853	6,63
R01	20	879	859,97	10800,00	31953	2,17	879	863,73	10800,00	33709	1,74
R02	20	830	759,21	10800,00	31194	8,53	830	757,22	10800,00	31936	8,77
R03	20	951	843,89	10800,00	29609	11,26	951	854,27	10800,00	28204	10,17
R04	20	874	787,49	10800,00	27855	9,90	874	786,66	10800,00	28135	9,99
R05	20	917	863,01	10800,00	31722	5,89	917	864,57	10800,00	31301	5,72
R06	20	983	926,50	10800,00	32733	5,75	983	934,31	10800,00	33817	4,95
R07	20	920	843,24	10800,00	30450	8,34	920	841,65	10800,00	30385	8,52
R08	20	979	894,18	10800,00	30307	8,66	979	898,78	10800,00	30530	8,19
R09	20	888	888,00	10167,85	36150	**0,00**	888	861,03	10800,00	34343	3,04
R10	20	945	847,51	10800,00	30411	10,32	945	853,50	10800,00	29032	9,68
R11	20	846	786,20	10800,00	33100	7,07	846	785,00	10800,00	30858	7,21
R12	20	925	863,16	10800,00	30264	6,69	925	865,73	10800,00	29567	6,41
R13	20	919	881,83	10800,00	33579	4,04	919	879,74	10800,00	32049	4,27
R14	20	921	832,09	10800,00	29951	9,65	921	841,42	10800,00	28117	8,64
R15	20	947	855,88	10800,00	26335	9,62	947	858,25	10800,00	25633	9,37
R16	20	869	837,35	10800,00	32958	3,64	869	822,18	10800,00	32604	5,39
R17	20	928	869,69	10800,00	29008	6,28	928	865,15	10800,00	29689	6,77
R18	20	928	855,17	10800,00	32682	7,85	928	865,62	10800,00	32483	6,72
R19	20	969	893,57	10800,00	29684	7,78	969	895,98	10800,00	28458	7,54
Average				*10766,88*	*30897,70*	*6,98*			*10800,00*	*30485,15*	*6,99*

Table 5.7: Running times of our best algorithms (CA3) and (CA4) and of the algorithm of [2].

Instance	Items	(CA3)	(CA4)	[2]
R00	12	15,12	19,27	8,30
R01	12	5,37	5,06	1,80
R02	12	53,03	**37,39**	42,80
R03	12	1,69	2,02	0,20
R04	12	**14,18**	**15,25**	40,70
R05	12	**28,41**	**29,08**	176,80
R06	12	6,64	7,93	3,30
R07	12	**7,75**	10,34	8,50
R08	12	35,69	**20,87**	30,10
R09	12	4,87	8,42	1,80
R10	12	**5,23**	**5,29**	12,00
R11	12	**12,83**	**16,54**	17,90
R12	12	2,72	3,00	0,30
R13	12	3,70	5,07	1,40
R14	12	**3,75**	**4,12**	4,20
R15	12	**16,53**	**17,95**	20,60
R16	12	**10,12**	**6,37**	25,40
R17	12	**27,65**	**25,44**	92,30
R18	12	4,02	3,66	0,70
R19	12	26,46	33,68	9,00
Average		*14,29*	*13,84*	*24,91*
R00	14	**152,69**	247,86	156,60
R01	14	**27,29**	**22,25**	41,90
R02	14	**58,12**	**51,24**	125,50
R03	14	77,95	24,71	24,10
R04	14	**412,28**	**698,79**	3815,10
R05	14	**141,05**	**95,55**	392,60
R06	14	**30,83**	**27,91**	33,40
R07	14	**30,42**	**27,03**	48,40
R08	14	**305,95**	**289,18**	672,50
R09	14	14,92	15,12	5,30
R10	14	**101,40**	**73,53**	390,80
R11	14	**319,65**	**305,59**	989,00
R12	14	97,67	**58,74**	86,50
R13	14	**25,93**	40,47	26,40
R14	14	**516,63**	**395,94**	10134,90
R15	14	22,87	22,26	22,10
R16	14	**34,84**	**27,54**	66,60
R17	14	**113,29**	**123,65**	956,70
R18	14	87,33	79,56	31,20
R19	14	274,21	177,28	149,30
Average		*142,27*	*140,21*	*908,45*

Another observation is that the best versions of our algorithm are able to solve to optimality previously unsolved instances of the double TSP with two stacks of finite capacity. Indeed, to the best of our knowledge only three instances with 16 items have been solved to optimality in [44] within three hours of CPU time and no larger instance has been solved before our work. Conversely, both (CA3) and (CA4) solve to optimality all instances with 16 items and also solve to optimality 11 out of the 20 instances with 18 items — see Table 5.5 and Table 5.6.

We finally point out that, for every instance of the double TSP with two stacks of finite capacity solved to optimality by our algorithm, the upper bound returned by the heuristic of Côté *et al.* [45] used for our tests coincides with the optimal value.

We conclude by presenting a comparison between the results described in this section with those described in Section 5.4.3 in the infinite capacity case. We focus on the best versions of our algorithm. Note that (CA3) (resp. (CA4)) is the "capacitated" version of (A3) (resp. (A4)), in the sense that it is obtained from (A3) (resp. (A4)) by just including the y-infeasible path constraints.

If we consider instances with up to 18 items, we observe that (CA3) and (CA4) solve to optimality exactly the same instances solved by (A3) and (A4). For 20 items, our best algorithm for the infinite capacity case (A3) solves one instance more than its "capacitated" version. Hence we have that the same set of 72 instances is solved to optimality in both the cases where capacity is infinite or finite. In 41 of these 72 instances we observe that the optimal values in the infinite capacity and finite capacity cases are the same.

As expected, (CA3) and (CA4) are in general slower than (A3) and (A4). Nevertheless, if we compare the average running times of Table 5.2 and Table 5.5 we see that they are of the same order of magnitude. More precisely, the finiteness of the capacity increases the average running times of less than 30% for every instance size.

From the above observations we infer that, with our approach, the finiteness of the capacity is not the major computational difficulty in solving to optimality benchmark instances involving two stacks.

5.5 Conclusions and Perspectives

In this chapter we have presented a branch-and-cut algorithm for the double TSP with two stacks. We have proposed four versions of our algorithm, each corresponding to a different combination of strengthening inequalities among those presented in Chapter 4. For each of the used families of inequalities we have presented polynomial-time separation routines subsequently embedded in our algorithm. The four versions of the algorithm have been tested on instances of the double TSP with two stacks of infinite capacity as well as on instances where the capacity is finite.

We have observed that the finiteness of the capacity does not affect the overall behavior of our algorithm. With respect to the existing literature on the problem, our approach reduces the computational time needed to solve instances of the double TSP with two

stacks of finite capacity and also solves previously unsolved instances. The largest instance solved to optimality has 20 items.

By comparing the four versions of our algorithm we have concluded that the formulation presented in Chapter 3 can be successfully used in a branch-and-cut algorithm to tackle instances of this problem with up to 14 items. To solve larger instances, our computational tests indicate that the routing inequalities presented in Section 4.1.1 should be used as strengthening inequalities. Since the routing inequalities do not carry any information on the consistency requirement that characterizes the double TSP with multiple stacks, we used the odd hole inequalities arising from the vertex cover approach to further improve our algorithm. Our tests have shown that the odd hole inequalities speed up our algorithm on several instances and reduce the average optimality gap for large unsolved instances.

From these observations we believe that a promising direction for future works is to enhance our algorithm for the double TSP with two stacks by exploiting the vertex cover approach. To this end, it is necessary to design fast heuristic separation routines for the odd hole inequalities. Heuristic separation routines should also be designed for the wheel inequalities presented in Corollary 4.2.15. Indeed, a polynomial separation algorithm for these inequalities is given in [48] but, in the case of the double TSP with two stacks and n items, its time complexity is in $O(n^7)$ hence we decided to not implement it in our algorithm. From the same point of view, we also believe that it is important to find new strengthening inequalities arising from the vertex cover approach.

Another interesting research direction is to consider the adaptation of our branch-and-cut algorithm to the important case of the double TSP with more than two stacks. In particular, we would like to investigate the computational effectiveness of the generalized cycle inequalities (4.20) arising from the set covering approach presented in Chapter 4. We point out that the existence of an exact separation routine for inequalities (4.20) was proven by Letchford in [126]. Unfortunately, this positive result does not seem easily transferable to separation algorithms likely to be effective in practice, since it relies on a disjunctive argument combined with the equivalence of separation and optimization. Hence, we think that the design of both heuristic and exact algorithms for the separation of such inequalities based on more combinatorial arguments needs to be addressed.

Part II

Lexicographical Polytopes

Chapter 6

Polyhedral Study of Lexicographical Polytopes

In this chapter we study the lexicographical polytopes. Within a fixed integer box of $\mathbb{R}^n$, a *lexicographical polytope* is the convex hull of the integer points of the box that are lexicographically between two given integer points of the box . Similarly, a *top-lexicographical polytope* (resp. *bottom-lexicographical polytope*) is the convex hull of the integer points of the box that are lexicographically equal or smaller (resp. greater) than a given integer point of the box.

Lexicographical polytopes can be seen in the more common context of knapsack polytopes *i.e.,* the convex hulls of the solutions to a binary knapsack problem. More precisely, every top-lexicographical polytope is a so-called superdecreasing knapsack polytope, and conversely (see Section 6.2 for the proof of this equivalence). The definition of superdecreasing knapsack polytopes that will be given in Section 6.2 is similar to the one of superincreasing knapsack polytope already present in the literature, see *e.g.,* [172, 46, 94, 144]. In fact, both superincreasing and superdecreasing knapsack polytopes can be studied with the same techniques and we can refer indifferently to any of them. Superincreasing knapsack polytopes have a wide range of applications and, by the equivalence above, they motivate the interest in lexicographical polytopes.

A first application is in the design of *knapsack cryptosystems*. These are cryptosystems that define a public "hard" binary knapsack problem to hide a corresponding binary superincreasing knapsack problem, whose resolution is tractable. The resolution of the superincreasing knapsack problem lets a receiver easily decipher any message encrypted according to specific rules, whereas an eavesdropper can decipher the same message only by solving the hard knapsack problem. See [144] for more details.

A second application is in the design of wireless networks [46]. Some standard available models for this task exhibit numerical issues. To overcome these difficulties the models can be reformulated as binary integer linear programs. The resulting knapsack constraints in the new programs have the superincreasing property hence they can be replaced by the set of inequalities describing the associated binary superincreasing knapsack polytopes. This operation partially solves the numerical issues.

In general, one reformulates integer linear programs by replacing general integer variables by several binary variables, using the so-called *binary expansion*, see *e.g.,* p.477 [186]. Each variable substitution generates a binary superincreasing knapsack constraint. Thus, all inequalities valid for the corresponding superincreasing knapsack polytope can be added to the reformulated program as strengthening cuts.

We point out that the above mentioned applications are based on binary superincreasing knapsack polytopes. Hence, it is not surprising that several researches have been interested in these polytopes. In particular, a complete linear description of binary superincreasing knapsack polytopes has been found independently by many authors over the past years. From the equivalence mentioned at the beginning of this introduction, the description provides a complete description of binary top-lexicographical polytopes. In this chapter we give a similar result in the general integer case, that is we provide a complete linear description of top-lexicographical and bottom-lexicographical polytopes. In addition we exploit them to obtain a complete linear description of lexicographical polytopes. Finally, we show that the intersection of two lexicographical polytopes yields a lexicographical polytope.

The chapter is organized as follows. In Section 6.1 we survey the known results on lexicographical polytopes. In Section 6.2 we give the fundamental definitions used throughout the chapter, as well as the proof of the equivalence between top-lexicographical and superdecreasing knapsack polytopes. In Section 6.3, we provide a flow based extended formulation of the convex hull of the componentwise maximal points of a top-lexicographical polytope. Projecting this formulation is surprisingly straightforward and thus, we get the description in the original space. In Section 6.4, using the fact that a top-lexicographical polytope is, up to translation, the submissive of the above convex hull, we derive the description of top-lexicographical polytopes. We then show that a lexicographical polytope is the intersection of its top- and bottom-lexicographical polytopes.

6.1 Known Results

Results on superincreasing knapsack polytopes are also a contribution to the lexicographical polytope theory, hence we report known results in both topics.

Superincreasing Knapsack Polytopes. The study of superincreasing knapsack polytopes starts in [123]. The authors show that a binary knapsack polytope is completely described by its minimal cover inequalities [14] if and only if it is a binary superincreasing knapsack polytope. They use an argument on Mengerian clutters due to Seymour [176]. Finally, they explicitly determine the minimal cover inequalities in terms of the coefficients defining the knapsack constraint and deduce that the optimization problem on binary superincreasing polytopes can be solved in polynomial time.

In [3] Angulo *et al.* consider two binary knapsack sets:

$$K^{\leq} = \{x \in \{0,1\}^n \colon ax \leq b\},$$
$$K^{\geq} = \{x \in \{0,1\}^n \colon ax \geq c\},$$

where the coefficients of $ax \leq b$ and $ax \geq c$ satisfy a binary superincreasing property, *i.e.,* $\sum_{i=1}^{k} a_i \leq a_{k+1}$. The goal of the paper is to provide a polynomial-size extended formulation for the polytopes obtained from the unit hypercube by removing a prescribed list of its vertices. Via a result of Balas on the union of polyhedra [9], they show that this is possible, provided that a polynomial-size formulation exists for $\text{conv}(K^{\leq} \cap K^{\geq})$. Hence, they prove that $\text{conv}(K^{\leq} \cap K^{\geq}) = \text{conv}(K^{\leq}) \cap \text{conv}(K^{\geq})$ and conclude using the description of $\text{conv}(K^{\leq})$ and $\text{conv}(K^{\geq})$ given in [123].

Given $u \in \mathbb{Z}^n_+$, Gupte [94] provides a complete description of the convex hull of the points in the knapsack set

$$\mathbf{K}^{\leq} = \{x \in \mathbb{Z}^n \colon ax \leq b, \mathbf{0} \leq x \leq u\},$$

when the coefficients of $ax \leq b$ satisfy a superincreasing property, *i.e.,* $\sum_{i=1}^{k} a_i u_i \leq a_{k+1}$. His result generalizes the one of [123] since the binary superincreasing knapsack polytopes studied in [123] are a special case of $\text{conv}(\mathbf{K}^{\leq})$. The proof in [94] are completely based on polyhedral arguments, first by proving the validity of the so-called packing inequalities for $\text{conv}(\mathbf{K}^{\leq})$, and subsequently showing their sufficiency for a complete description. The author observes that the same results can be obtained by considering the convex hull of the points in:

$$\mathbf{K}^{\geq} = \{x \in \mathbb{Z}^n \colon ax \geq b, \mathbf{0} \leq x \leq u\}.$$

with $ax \geq b$ satisfying the superincreasing property. Finally, he proves that $\text{conv}(\mathbf{K}^{\leq} \cap \mathbf{K}^{\geq}) = \text{conv}(\mathbf{K}^{\leq}) \cap \text{conv}(\mathbf{K}^{\geq})$. In this case he uses a well-known result on the union of polyhedra due to Balas [9]. This latter result generalizes the one of Angulo *et al.* [3] described above.

Lexicographical Polytopes. The study of lexicographical polytopes is initiated in the binary case in [77]. A complete linear description of binary top-lexicographical polytopes is provided, and the authors deduce that the minimum number of facets that a binary d-dimensional polytope with n vertices can have is bounded above by $3d$. The structure of lexicographical polytopes is used to support the hypothesis that sorting lexicographically randomly generated binary points might be a good strategy to have fast incremental algorithms for the construction of their convex hull — an incremental algorithm constructs the convex hull of the first $i+1$ points in a set from the convex hull of the first i points.

In the second part of their paper, the authors study the graph $L_n = (V, E)$ of n-dimensional top-lexicographical polytopes. They show that for an n-dimensional lexicographical polytope, the average degree of a vertex in L_n is at most $4+n$. Finally, they prove that the *edge-expansion* of the same graph, defined as $\min\{\frac{\delta(S)}{|S|} \colon S \subset V, 0 < |S| \leq \frac{|V|}{2}\}$,

is at least one, thus supporting a conjecture of Mihail and Vazirani [135] stating that this holds for every binary polytope.

In [141], Muldoon *et al.* independently find the results of the above mentioned paper by Angulo *et al.* [3] about the intersection of binary superincreasing knapsack polytopes. However, the approach of [141] is initially to consider binary lexicographical polytopes. Subsequently in the paper, the equivalence with the superdecreasing knapsack is revealed. Finally, the authors employ the inequalities describing the binary lexicographical polytope to strengthen the reformulations of challenging integer linear programs via the binary expansion of integer variables. They show that solving the reformulation needs in average only 59% of the time needed to solve the original formulation.

Very recently (April 2016) Adams *et al.* [1] have proven some of the results found in [94] from the lexicographical polytope point of view. That is they provide a linear description of the lexicographical polytope (not necessarily in the binary case). They first present a family of inequalities valid for a top-lexicographical polytope. Subsequently, they show that the coefficients of these inequalities are related to the inverse of structured lower triangular matrices. Exploiting this relation, they show that the proposed inequalities are also sufficient to describe the top-lexicographical polytopes. Using Balas' results on the union of polyhedra [9] they also provide an extended formulation for top lexicographical polytopes as well as an approximation of the convex hull of the matrices with lexicographically nonincreasing columns.

6.2 Definitions and Preliminary Results

Throughout, ℓ, u, r, s will denote integer points satisfying $\ell \leq r \leq u$ and $\ell \leq s \leq u$, that is r and s are within $[\ell, u]$. A point $x \in \mathbb{Z}^n$ is *lexicographically smaller than* $y \in \mathbb{Z}^n$, denoted by $x \preccurlyeq y$, if $x = y$ or the first nonzero coordinate of $y - x$ is positive. We write $x \prec y$ if $x \preccurlyeq y$ and $x \neq y$. Note that $\prec$ is a linear ordering over $\mathbb{Z}^n$. The *lexicographical polytope* $P_{\ell,u}^{r \preccurlyeq s}$ is the convex hull of the integer points within $[\ell, u]$ that are lexicographically between r and s:

$$P_{\ell,u}^{r \preccurlyeq s} = \text{conv}\{x \in \mathbb{Z}^n : \ell \leq x \leq u, r \preccurlyeq x \preccurlyeq s\}.$$

The *top-lexicographical polytope* $P_{\ell,u}^{\preccurlyeq s} = \text{conv}\{x \in \mathbb{Z}^n : \ell \leq x \leq u, x \preccurlyeq s\}$ is the special case when $r = \ell$. Similarly, the *bottom-lexicographical polytope* is $P_{\ell,u}^{r \preccurlyeq} = \text{conv}\{x \in \mathbb{Z}^n : \ell \leq x \leq u, r \preccurlyeq x\}$.

There is a close relations between top-lexicographical polytopes and the so-called superdecreasing knapsack polytopes. Given $a, u \in \mathbb{R}_+^n$ and $b \in \mathbb{R}_+$, the *knapsack polytope* defined by $K_u^{a,b} = \text{conv}\{x \in \mathbb{Z}^n, 0 \leq x \leq u, ax \leq b\}$ is *superdecreasing* if:

$$\sum_{i>k} a_i u_i \leq a_k \qquad \text{for } k = 1, \ldots, n. \tag{6.1}$$

Observe that a superdecreasing knapsack $K_u^{a,b}$ is the top-lexicographical polytope $P_{\mathbf{0},u}^{\preccurlyeq s}$, where s the lexicographically greatest integer point of $K_u^{a,b}$. The non trivial inclusion

actually holds because every integer point x of $P^{\preccurlyeq s}_{\mathbf{0},u}$ satisfies $ax \leq as$. Indeed, by definition, if $x \prec s$, there exists $k \in \{1, \ldots, n\}$ such that $x_k + 1 \leq s_k$ and $x_i = s_i$ for $i < k$. Hence, we have $b - ax \geq as - ax \geq \sum_{i>k} a_i(s_i - x_i) + a_k \geq \sum_{i>k} a_i(s_i - x_i + u_i) \geq 0$, because of (6.1), $s_i \geq 0$ and $u_i \geq x_i$.

It turns out that top-lexicographical polytopes are superdecreasing knapsack polytopes. Indeed, let $P^{\preccurlyeq s}_{\ell,u}$ be a top-lexicographical polytope for some s within $[\ell, u]$. Possibly after translating, we may assume $\ell = \mathbf{0}$. Define a by $a_k = \sum_{i>k} a_i u_i + 1$, for $k = 1, \ldots, n$, and let $b = as$. Since the associated knapsack polytope $K^{a,b}_u$ is superdecreasing, if $x \preccurlyeq s$ then $ax \leq as = b$, for all x within $[\mathbf{0}, u]$. Moreover, the converse holds because, inequalities (6.1) being all strict, $s \prec x$ implies $b = as < ax$. Therefore, $P^{\preccurlyeq s}_{\mathbf{0},u} = K^{a,b}_u$. These observations are summarized in the following.

Remark 6.2.1. Superdecreasing knapsacks are top-lexicographical polytopes, and conversely (up to translations).

The equivalence in Remark 6.2.1 has been observed by several authors. For the 0/1 case, that is when $\ell = \mathbf{0}$ and $u = \mathbf{1}$, Gillmann and Kaibel [77] first noticed that top-lexicographical polytopes are special cases of superdecreasing knapsack ones, and the converse has been later established by Muldoon *et al.* [141]. Recently, Gupte [94] generalized the latter result by showing that all superdecreasing knapsacks are top-lexicographical polytopes.

Example 6.2.2. Let us consider the following knapsack polytope

$$K^{a,b}_u = \operatorname{conv}\{x \in \mathbb{Z}^4 \colon \mathbf{0} \leq x \leq u \\ 594x_1 + 54x_2 + 9x_3 + x_4 \leq 4200\},$$

where $u = (7, 10, 5, 8)$, $a = (594, 54, 9, 1)$ and $b = 4200$. It can be verified that it is a superdecreasing knapsack polytope since $\sum_{i>k} a_i u_i \leq a_k$ for $k = 1, \ldots, 4$. It can be easily verified that $s = (7, 0, 4, 6)$ is the lexicographically greatest point in $K^{a,b}_u$, hence, according to the definitions given above we have $K^{a,b}_u = P^{\preccurlyeq s}_{\mathbf{0},u}$. The forthcoming Theorem 6.4.2 can then be used to show that $K^{a,b}_u$ is the set of points (x_1, x_2, x_3, x_4) satisfying the bound inequalities $0 \leq x_i \leq u_i$ for $i = 1, \ldots, 4$ and the following set of inequalities:

$$\begin{aligned} 4x_1 + 2x_3 + x_4 &\leq 42 \\ 10x_1 + x_2 &\leq 70 \\ x_1 + x_3 &\leq 11 \\ x_1 &\leq 7. \end{aligned}$$

In next sections, we provide the description of the lexicographical polytopes which, thanks to Observation 6.2.1, gives alternative proofs of the results of Gupte [94] and Adams *et al.* [1]. We would like to emphasize that our approach provides very simple and short proofs and conveys a geometrical insight. This is due to the efficient combination of polyhedral arguments together with a suitable extended formulation for the convex hull of componentwise maximal integer points.

6.3 Convex Hull of Componentwise Maximal Points

From now on, $X^{\preccurlyeq s}_{\ell,u}$ will denote the set of the points $p^i = (s_1, \ldots, s_{i-1}, s_i - 1, u_{i+1}, \ldots, u_n)$, for $i = 1, \ldots, n+1$ such that $s_i > \ell_i$, $p^{n+1} = s$ by definition. Note that $X^{\preccurlyeq s}_{\ell,u}$ consists of the componentwise maximal integer points of $P^{\preccurlyeq s}_{\ell,u}$, to which we added, for later convenience, the point $p^n = (s_1, \ldots, s_{n-1}, s_n - 1)$ if $s_n > \ell_n$.

6.3.1 A Flow Model for $X^{\preccurlyeq s}_{\ell,u}$

We first model the points of $X^{\preccurlyeq s}_{\ell,u}$ as paths from 1 to $n+1$ in the digraph given in Figure 6.1.

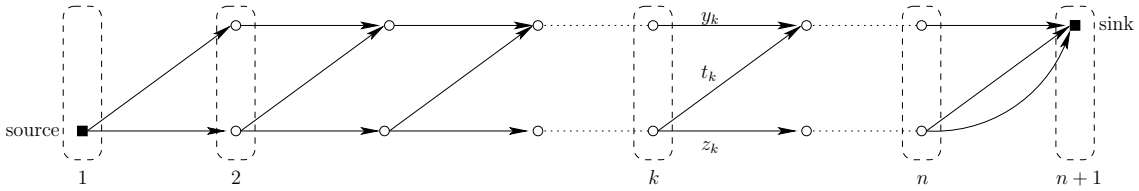


Figure 6.1: Path representation of the points of $X^{\preccurlyeq s}_{\ell,u}$.

Our digraph is composed of $n+1$ layers, each containing two nodes except the first and the last ones. There are three arcs connecting the layer k to the layer $k+1$, an upper arc y_k, a diagonal arc t_k and a lower arc z_k. The only exception concerns the first level, which does not have the upper arc.

The arcs connecting two successive layers correspond to a coordinate of $x \in X^{\preccurlyeq s}_{\ell,u}$. More precisely, given a directed path P from 1 to $n+1$, we define the point x by setting, for $k = 1, \ldots, n$,

$$x_k = \begin{cases} u_k & \text{if } y_k \in P, \\ s_k - 1 & \text{if } t_k \in P, \\ s_k & \text{if } z_k \in P. \end{cases}$$

As shown in Observation 6.3.1, the set of (x, y, z, t) satisfying the following set of inequalities is an extended formulation of $\mathrm{conv}(X^{\preccurlyeq s}_{\ell,u})$:

$$x_i = u_i y_i + (s_i - 1)t_i + s_i z_i \qquad \text{for } i = 1, \ldots, n, \tag{6.2}$$
$$y_1 = 0 \tag{6.3}$$
$$y_i = y_{i-1} + t_{i-1} \qquad \text{for } i = 2, \ldots, n, \tag{6.4}$$
$$z_i = z_{i+1} + t_{i+1} \qquad \text{for } i = 1, \ldots, n-1, \tag{6.5}$$
$$t_i = 0 \qquad \text{whenever } s_i = \ell_i, \tag{6.6}$$
$$y_n + t_n + z_n = 1 \tag{6.7}$$
$$y_i, t_i, z_i \geq 0 \qquad \text{for } i = 1, \ldots, n. \tag{6.8}$$

Observation 6.3.1. $\text{conv}(X^{\preccurlyeq s}_{\ell,u}) = \text{proj}_x\{(x,y,z,t) \text{ satisfying (6.2)-(6.8)}\}$.

Proof. First, note that there is a one-to-one correspondence between the points of $X^{\preccurlyeq s}_{\ell,u}$ and the paths from layer 1 to layer $n+1$ of the digraph. This implies that $X^{\preccurlyeq s}_{\ell,u}$ is the projection onto the x variables of the integer points of $Q = \{(x,y,z,t) \text{ satisfying (6.2)–(6.8)}\}$. The digraph being acyclic, the set of (y,z,t) satisfying (6.3)-(6.8) is a path polytope and thus is an integral polytope. The integrality of u and s implies that Q is integral, hence so is its projection onto the x variables, which concludes the proof. □

6.3.2 Description of $\text{conv}(X^{\preccurlyeq s}_{\ell,u})$

In the following result, we use Observation 6.3.1 to provide a linear description of $\text{conv}(X^{\preccurlyeq s}_{\ell,u})$.

Lemma 6.3.2. $\text{conv}(X^{\preccurlyeq s}_{\ell,u})$ *is described by the inequalities:*

$$\sum_{i=1, s_i > \ell_i}^{n} A_i(x) \geq -1 \tag{6.9}$$

$$A_k(x) \leq 0 \quad \text{for } k = 1, \dots, n, \tag{6.10}$$

$$A_k(x) \geq 0 \quad \text{when } s_k = \ell_k, \tag{6.11}$$

where, for $k = 1, \dots, n$,

$$A_k(x) := (x_k - s_k) + (u_k - s_k) \sum_{i=1, s_i > \ell_i}^{k-1} (x_i - s_i) \prod_{j=i+1, s_j > \ell_j}^{k-1} (u_j - s_j + 1).$$

Proof. By Observation 6.3.1, it suffices to project onto the x variables of the set of x, y, t, z satisfying (6.2)-(6.8).

For $k = 1, \dots, n$, we get $y_k = \sum_{i=1}^{k-1} t_i$ by (6.3) and (6.4). This, combined with (6.5), (6.7), yields $z_k = 1 - \sum_{i=1}^{k} t_i$. Using those two equations in (6.2), and $t_k = 0$ whenever $s_k = \ell_k$, we obtain

$$t_k = s_k - x_k + (u_k - s_k) \sum_{i=1, s_i > \ell_i}^{k-1} t_i, \quad \text{for } k = 1, \dots, n. \tag{6.12}$$

We now show by induction on k that, for all $k = 1, \dots, n$,

$$\sum_{i=1, s_i > \ell_i}^{k} t_i = \sum_{i=1, s_i > \ell_i}^{k} (s_i - x_i) \prod_{j=i+1, s_j > \ell_j}^{k} (u_j - s_j + 1). \tag{6.13}$$

By definition of t_k, (6.13) holds for $k = 1$. Let us suppose that (6.13) holds for $k < n$ and show that it holds for $k+1$. The result is immediate if $s_{k+1} = \ell_{k+1}$, hence assume that

$s_{k+1} > \ell_{k+1}$. We have

$$\sum_{i=1,s_i>\ell_i}^{k+1} t_i = (s_{k+1} - x_{k+1}) + (u_{k+1} - s_{k+1}) \sum_{i=1,s_i>\ell_i}^{k} t_i + \sum_{i=1,s_i>\ell_i}^{k} t_i \tag{6.14}$$

$$= (s_{k+1} - x_{k+1}) + (u_{k+1} - s_{k+1} + 1) \sum_{i=1,s_i>\ell_i}^{k} (s_i - x_i) \prod_{j=i+1,s_j>\ell_j}^{k} (u_j - s_j + 1) \tag{6.15}$$

$$= \sum_{i=1,s_i>\ell_i}^{k+1} (s_i - x_i) \prod_{j=i+1,s_j>\ell_j}^{k+1} (u_j - s_j + 1).$$

Above, equality (6.14) follows from (6.12) applied to t_{k+1} and equality (6.15) follows using (6.13).

Injecting (6.13) in (6.12) leads to

$$t_k = s_k - x_k + (u_k - s_k) \sum_{i=1,s_i>\ell_i}^{k-1} (s_i - x_i) \prod_{j=i+1,s_j>\ell_j}^{k-1} (u_j - s_j + 1) \qquad \text{for } k = 1, \ldots, n. \tag{6.16}$$

Up to now, we only used linear transformations, thus projecting out the variables y, z gives us (6.16), $\sum_{i=1,s_i>\ell_i}^{n} t_i \leq 1$, $t_k = 0$ whenever $s_k = \ell_k$ and $t_k \geq 0$ otherwise. Then, projecting onto the x variable gives the desired result. □

Note that the following derives from the above proof by combining (6.12) and the fact that, by (6.16), we have $t_k = -A_k$:

$$A_k(x) = (x_k - s_k) + (u_k - s_k) \sum_{i=1,s_i>\ell_i}^{k-1} A_i(x), \text{ for } k = 1, \ldots, n. \tag{6.17}$$

6.4 Lexicographical Polytopes

In this section, we first provide the description of top-lexicographical polytopes. We then show that a lexicographical polytope is the intersection of its top- and bottom-lexicographical polytopes.

6.4.1 Description of Top-lexicographical Polytopes

The following observation unveils the polyhedral relation between a top-lexicographical polytope and the convex hull of its componentwise maximal points.

Observation 6.4.1. $P_{\ell,u}^{\preccurlyeq s} = (\text{conv}(X_{\ell,u}^{\preccurlyeq s}) + \mathbb{R}^n_-) \cap \{x \geq \ell\}$.

Proof. Since $\mathrm{conv}(X^{\preccurlyeq s}_{\ell,u})$ is integer and contained in $\{x \geq \ell\}$, the polyhedron on the right is integer. Seen the definitions, the observation follows. □

Remark that, when $\ell = 0$, $P^{\preccurlyeq s}_{\ell,u}$ is precisely the submissive of $\mathrm{conv}(X^{\preccurlyeq s}_{\ell,u})$ — see [12] for the definition of submissive of a polytope. Now, we derive from Lemma 6.3.2 and Observation 6.4.1 the linear description of top-lexicographical polytopes.

Theorem 6.4.2. $P^{\preccurlyeq s}_{\ell,u} = \{x \in \mathbb{R}^n : \ell \leq x \leq u, A_k(x) \leq 0, \textit{for } k = 1, \ldots, n\}$.

Proof. Theorem 6.4.2 immediately follows from Observation 6.4.1 and the following description of $\mathrm{conv}(X^{\preccurlyeq s}_{\ell,u}) + \mathbb{R}^n_-$,

$$\mathrm{conv}(X^{\preccurlyeq s}_{\ell,u}) + \mathbb{R}^n_- = \{x \in \mathbb{R}^n : x \leq u \text{ and } A_k(x) \leq 0, \text{ for } k = 1, \ldots, n\}. \tag{6.18}$$

To prove (6.18), denote by Q its right hand side. By Lemma 6.3.2, the above inequalities are valid for $\mathrm{conv}(X^{\preccurlyeq s}_{\ell,u})$. Since their coefficients for x are nonnegative, they also hold for $\mathrm{conv}(X^{\preccurlyeq s}_{\ell,u}) + \mathbb{R}^n_-$. Note that the latter and Q have the same recession cone, thus it remains to show that the vertices of Q are vertices of $\mathrm{conv}(X^{\preccurlyeq s}_{\ell,u})$. Let us prove it by induction on the dimension, the base case being immediate. We may assume that $u_n > s_n$, as otherwise $A_n(x) = x_n - s_n$ and the induction concludes. Let $\bar{x}$ be a vertex of Q.

Claim 6.4.3. $\displaystyle\sum_{i=1, s_i > \ell_i}^{n} A_i(\bar{x}) \geq -1.$

Proof. The indices i of $A_i(x)$ involved in sums throughout this proof satisfy $s_i > \ell_i$, yet to ease the reading, we will omit the subscripts "$s_i > \ell_i$". By contradiction, assume that $\sum_{i=1}^n A_i(\bar{x}) < -1$. Since $\bar{x}$ is a vertex, and x_n appears only in $x_n \leq u_n$ and $A_n(x) \leq 0$, at least one of them holds with equality. If the latter does, then by (6.17) and $u_n > s_n$, we get the contradiction $0 = A_n(\bar{x}) \leq (u_n - s_n)(1 + A_1(\bar{x}) + \cdots A_{n-1}(\bar{x})) < (u_n - s_n)(1 - 1) = 0$. Therefore $A_n(\bar{x}) < 0$ and $\bar{x}_n = u_n$. For $x \in \mathbb{R}^n$, we denote $x' := (x_1, \ldots, x_{n-1})$. Necessarily, $\bar{x}'$ satisfies to equality $n-1$ linearly independent of the remaining inequalities, and hence $\bar{x}'$ is a vertex of $\{x \in \mathbb{R}^{n-1} : x_k \leq u_k, A_k(x) \leq 0, \text{ for } k = 1, \ldots, n-1\}$. By the induction hypothesis, $\bar{x}'$ is a vertex of $\mathrm{conv}(X^{\preccurlyeq s'}_{\ell',u'}) + \mathbb{R}^{n-1}_-$, hence $\sum_{i=1}^{n-1} A_i(\bar{x}') \geq -1$. But now $A_n(\bar{x}) < 0$, $\bar{x}_n = u_n$ and (6.17) imply $A_1(\bar{x}') + \cdots + A_{n-1}(\bar{x}') < -1$, a contradiction. ■

Let us show that $A_k(\bar{x}) = 0$ whenever $s_k = \ell_k$. Indeed, in this case, $\bar{x}_k$ only appears in $A_k(\bar{x}) \leq 0$ and $\bar{x}_k \leq u_k$, and one is satisfied with equality since $\bar{x}$ is a vertex. If $\bar{x}_k = u_k$, then by (6.17), Claim 6.4.3 and $A_i(\bar{x}) \leq 0$, for $i = 1 \ldots, n$, we get $0 \geq A_k(\bar{x}) = (u_k - s_k)(1 + \sum_{i=1, s_i > \ell_i}^{k-1} A_i(\bar{x})) \geq 0$. Consequently, $\bar{x}$ belongs to $\mathrm{conv}(X^{\preccurlyeq s}_{\ell,u})$ and this proves (6.18). □

Symmetrically, bottom-lexicographical polytopes are described as follows.

Corollary 6.4.4. $P^{r\preccurlyeq}_{\ell,u} = \{x \in \mathbb{R}^n : \ell \leq x \leq u, B_k(x) \leq 0, \textit{for } k = 1, \ldots, n\}$, *where, for* $k = 1, \ldots, n$,

$$B_k(x) = (r_k - x_k) + (r_k - \ell_k) \sum_{i=1, r_i < u_i}^{k-1} (r_i - x_i) \prod_{j=i+1, r_j < u_j}^{k-1} (r_j - \ell_j + 1).$$

6.4.2 Lexicographical Polytopes

By definition, we have $P_{\ell,u}^{r\preccurlyeq s} \subseteq P_{\ell,u}^{r\preccurlyeq} \cap P_{\ell,u}^{\preccurlyeq s}$. It turns out that the converse holds, see Theorem 6.4.5. In particular, $P_{\ell,u}^{r\preccurlyeq} \cap P_{\ell,u}^{\preccurlyeq s}$ is an integer polytope.

Theorem 6.4.5. *A lexicographical polytope is the intersection of its top- and bottom-lexicographical polytopes.*

Proof. It remains to prove that $P_{\ell,u}^{r\preccurlyeq s} \supseteq Q$, where $Q = P_{\ell,u}^{r\preccurlyeq} \cap P_{\ell,u}^{\preccurlyeq s}$. Let us prove it by induction on the dimension, the one-dimensional case being immediate.

If $r_1 = s_1$, then the problem reduces to the $(n-1)$-dimensional case, and using induction concludes.

If $r_1+1 \leq \pi \leq s_1-1$ for some integer π, then let ℓ' be obtained from ℓ by replacing ℓ_1 by π. By $s_1 > \ell'_1$ and the definition of $A_k(x)$, applying Theorem 6.4.2 gives $P_{\ell,u}^{\preccurlyeq s} \cap \{x_1 \geq \pi\} = P_{\ell',u}^{\preccurlyeq s}$. Moreover, since $\pi > r_1$, the latter is contained in $P_{\ell,u}^{r\preccurlyeq}$. Therefore $Q \cap \{x_1 \geq \pi\} = P_{\ell',u}^{\preccurlyeq s}$ is integer. Similarly, $Q \cap \{x_1 \leq \pi\}$ is integer, hence so is Q, and we are done.

The remaining case is when $r_1 = s_1 - 1$. Let $\bar{x} \in P_{\ell,u}^{r\preccurlyeq} \cap P_{\ell,u}^{\preccurlyeq s}$. If $\bar{x}_1 = s_1$, when $\bar{x}$ is written as a convex combination of integer points of $P_{\ell,u}^{\preccurlyeq s}$, all of them have their first coordinate equal to s_1, and hence belong to $P_{\ell,u}^{r\preccurlyeq s}$. By convexity, so does $\bar{x}$ and we are done. A similar argument may be applied if $\bar{x}_1 = r_1$. Therefore, we may assume that $r_1 < \bar{x}_1 < s_1$.

Let $\lambda = \bar{x}_1 - r_1$, and define y by $y_1 = s_1$ and $y_k = u_k + \frac{\bar{x}_k - u_k}{\lambda}$ for $k = 2, \ldots, n$. Similarly, define z by $z_1 = r_1$ and $z_i = \ell_i + \frac{\bar{x}_i - \ell_i}{1-\lambda}$, for $i = 2, \ldots, n$. The following claim finishes the proof, where, given two points v and w of $\mathbb{R}^n$, $\max(v, w)$ (resp. $\min(v, w)$) will denote the point of $\mathbb{R}^n$ whose i^{th} coordinate is $\max\{v_i, w_i\}$ (resp. $\min\{v_i, w_i\}$) for $i = 1, \ldots, n$.

Claim 6.4.6. *$\bar{x}$ is a convex combination of $\bar{y} = \max(y, \ell)$ and $\bar{z} = \min(z, u)$ which both belong to $P_{\ell,u}^{r\preccurlyeq s}$.*

Proof. First, let us show that $y \in \operatorname{conv}(X_{\ell,u}^{\preccurlyeq s}) + \mathbb{R}^n_-$. As $\bar{x} \leq u$, we have $y \leq u$. Moreover, $A_1(y) = y_1 - s_1 = 0$. Now, we prove by induction that $A_k(y) = \frac{1}{\lambda} A_k(\bar{x})$ for $k = 2, \ldots, n$. Using (6.17), $A_1(y) = 0$, the definition of y_k, and the induction hypothesis, we have $A_k(y) = \frac{1}{\lambda}[\bar{x}_k - s_k + (\lambda-1)(u_k - s_k) + (u_k - s_k) \sum_{i=2, s_i > \ell_i}^{k-1} A_i(\bar{x})]$. Since $\lambda - 1 = \bar{x}_1 - s_1 = A_1(\bar{x})$ and $s_1 = r_1 + 1 > \ell_1$, we get by (6.17) that $A_k(y) = \frac{1}{\lambda} A_k(\bar{x})$, for $k = 2, \ldots, n$. Since $A_k(\bar{x}) \leq 0$, we have $A_k(y) \leq 0$. Hence, $y \in \operatorname{conv}(X_{\ell,u}^{\preccurlyeq s}) + \mathbb{R}^n_-$. Therefore, there exists y^+ of $\operatorname{conv}(X_{\ell,u}^{\preccurlyeq s})$ with $y^+ \geq y$. Clearly, $y^+ \geq \ell$ hence $y^+ \geq \max(y, \ell)$. Thus, $\max(y, \ell)$ belongs to $\operatorname{conv}(X_{\ell,u}^{\preccurlyeq s}) + \mathbb{R}^n_-$ and, by Observation 6.4.1, to $P_{\ell,u}^{\preccurlyeq s}$. Moreover, as its first coordinate equals s_1, $\max(y, \ell)$ belongs to $P_{\ell,u}^{r\preccurlyeq s}$. Similarly, $\min(z, u)$ also belongs to $P_{\ell,u}^{r\preccurlyeq s}$.

Finally, we have $(1-\lambda)\bar{z}_1 + \lambda\bar{y}_1 = (1-\lambda)(s_1 - 1) + \lambda s_1 = s_1 - 1 + \lambda = \bar{x}_1$. For $i \in \{2, \ldots, n\}$, we have $(1-\lambda)\bar{z}_i + \lambda\bar{y}_i = \min(\bar{x}_i - \lambda\ell_i, (1-\lambda)u_i) + \max((\lambda-1)u_i + \bar{x}_i, \lambda\ell_i) = \bar{x}_i - \max(\lambda\ell_i, (\lambda-1)u_i + \bar{x}_i) + \max((\lambda-1)u_i + \bar{x}_i, \lambda\ell_i) = \bar{x}_i$. Therefore, $\bar{x} = (1-\lambda)\bar{z} + \lambda\bar{y}$ and we are done. ■ □

Note that the above result implies that the family of lexicographical polytopes defined on a fixed box $[\ell, u]$ is closed by intersection. Beside, combined with Theorem 6.4.2 and Corollary 6.4.4, it provides the description of lexicographical polytopes.

Corollary 6.4.7. *The lexicographical polytope $P_{\ell,u}^{r\preccurlyeq s}$ is described as follows.*

$$P_{\ell,u}^{r\preccurlyeq s} = \left\{ \begin{array}{lll} x \in \mathbb{R}^n : & A_k(x) \leq 0 & \text{for } k = 1, \ldots, n \\ & B_k(x) \leq 0 & \text{for } k = 1, \ldots, n \\ & \ell \leq x \leq u & \end{array} \right\}.$$

Conclusion

In this thesis we have studied two problems arising in combinatorial optimization.

In the first part of the thesis we have considered the double TSP with multiple stacks. In this problem a vehicle picks up a set of items in one region and delivers them to a corresponding set of customers in a far region. The items are put in stacks and can be unloaded only following a last-in-first-out rule.

We have given a new integer linear programming formulation for the double TSP with multiple stacks. Our formulation is based on a model for the TSP involving precedence variables [171] and on the so-called y-infeasible path constraints introduced in this thesis. A solution to our formulation coincides with a pair of Hamiltonian circuits representing a feasible pair of pickup and delivery circuits. In the case of the double TSP with two stacks we have presented an algorithm to retrieve from such a pair a loading plan, *i.e.,* a disposition of the items in the stacks that is consistent with the pair of circuits. Such an algorithm is based on the resolution of a binary knapsack problem and improves the worst-case running time of an algorithm presented in [23]. We have also focused on the special case of the problem in which the stacks have an infinite capacity. In this latter case our formulation contains a polynomial number of constraints when the number of stacks is fixed.

Afterward, we have studied the DTSPMS polytope, *i.e.,* the convex hull of the solutions to our formulation in the infinite capacity case. The study has been divided into two main parts, accordingly with the dual nature of our formulation. In the first part we have focused on the routing aspects. The DTSPMS polytope is related to the PATSP polytope, *i.e.,* the polytope of the solutions to an extended integer linear programming formulation for the TSP. Our main result is that every facet of the PATSP polytope induces two facets of the DTSPMS polytope. We have then considered the consistency part of our formulation, *i.e.,* the one modeling the last-in-first-out rule. From this part we have derived a set covering relaxation that can be exploited to obtain new valid inequalities for our formulation. Not only this set covering approach is theoretically elegant. It also captures some aspects of the double TSP with multiple stack that make this problem so challenging. Our construction has proven to be very advantageous in the two stack case where the set covering relaxation relates to a vertex cover polytope. We have then introduced new families of inequalities based on odd hole and wheel inequalities which are well-known inequalities valid for the vertex cover polytope.

Finally, we have described a branch-and-cut algorithm for the double TSP with two

stacks. This algorithm relies on the polyhedral results described above. In particular several strengthening cuts arising from the TSP and introduced in [84] as well as the odd hole inequalities have been used in our algorithm. This was possible since we could provide polynomial-time separation routines for all the tested families. We have drawn the conclusion that the inequalities arising from the TSP are useful in speeding up our algorithm. In addition, we have observed that our approach based on set covering polytopes is promising to improve our computational results. In the case of the double TSP with two stacks, the best version of our algorithm outperforms the existing algorithms for the problem in terms of computational time and size of solved instances.

In the second part of the thesis we have studied the lexicographical polytopes. Given an n-dimensional box, a lexicographical polytope is the convex hull of the points in the box that are lexicographically between two fixed points in the box. We have provided a complete linear description of these polytopes, thus generalizing results already known in the binary case [3, 141]. Our construction is based on simple polyhedral arguments and offers an alternative point of view on results independently found by Gupte [94] and Adams *et al.* [1].

This thesis opens several directions for future works, all related to the double TSP with multiple stacks. From a polyhedral point of view, it would be interesting to find facets of the DTSPMS polytope. A way to obtain results of this kind would be to exploit the link between the facets of the DTSPMS polytope and the facets of the PATSP polytope. Another perspective is to find conditions under which the inequalities presented in this thesis and arising from the set covering approach define facets for the DTSPMS polytope.

From a computational point of view, we believe that it is important to design heuristic separation routines for the strengthening inequalities presented in this thesis. In addition, we believe that our branch-and-cut algorithm can be enhanced by including new strengthening inequalities arising from the set covering approach. Thus an additional direction to extend our work is to find new inequalities of this type. In this sense, a first step is to focus on the two stack case where we can exploit the vertex cover setting.

Bibliography

[1] W. Adams, P. Belotti, and R. Shen. Convex hull characterizations of lexicographic orderings. *Journal of Global Optimization*, pages 1–19, 2016.

[2] M. A. Alba Martínez, J.-F. Cordeau, M. Dell'Amico, and M. Iori. A branch-and-cut algorithm for the double traveling salesman problem with multiple stacks. *INFORMS Journal on Computing*, 25(1):41–55, 2013.

[3] G. Angulo, S. Ahmed, and S. Dey. Forbidding extreme points from the 0-1 hypercube. *Optimization Online*, May 2012.

[4] S. Anily and J. Bramel. Approximation algorithms for the capacitated traveling salesman problem with pickups and deliveries. *Naval Research Logistics (NRL)*, 46(6):654–670, Sept. 1999.

[5] S. Anily and G. Mosheiov. The Traveling Salesman Problem with Delivery and Backhauls. *Oper. Res. Lett.*, 16(1):11–18, Aug. 1994.

[6] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. *Finding cuts in the TSP (A preliminary report)*, volume 95. Citeseer, 1995.

[7] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Concorde: A code for solving traveling salesman problems. *World Wide Web, http://www.math.princeton.edu/tsp/concorde.html*, 2003.

[8] E. Balas. The asymmetric assignment problem and some new facets of the traveling salesman polytope on a directed graph. *SIAM Journal on Discrete Mathematics*, 2(4):425–451, 1989.

[9] E. Balas. Disjunctive programming: Properties of the convex hull of feasible points. *Discrete Applied Mathematics*, 89(1):3–44, 1998.

[10] E. Balas and N. Christofides. A restricted lagrangean approach to the traveling salesman problem. *Mathematical Programming*, 21(1):19–46, 1981.

[11] E. Balas and M. Fischetti. A lifting procedure for the asymmetric traveling salesman polytope and a large new class of facets. *Mathematical Programming*, 58(1-3):325–352, 1993.

[12] E. Balas and M. Fischetti. On the monotonization of polyhedra. *Mathematical Programming*, 78(1):59–84, 1996.

[13] E. Balas, M. Fischetti, and W. R. Pulleyblank. The precedence-constrained asymmetric traveling salesman polytope. *Mathematical programming*, 68(1-3):241–265, 1995.

[14] E. Balas and R. Jeroslow. Canonical cuts on the unit hypercube. *SIAM Journal on Applied Mathematics*, 23(1):61–69, 1972.

[15] R. Baldacci, E. Hadjiconstantinou, and A. Mingozzi. An exact algorithm for the Traveling Salesman Problem with Deliveries and Collections. *Networks*, 42(1):26–41, Aug. 2003.

[16] M. Batista-Galván, J. Riera-Ledesma, and J. J. Salazar-González. The traveling purchaser problem, with multiple stacks and deliveries: A branch-and-cut approach. *Computers & Operations Research*, 40(8):2103–2115, 2013.

[17] M. Battarra, J.-F. Cordeau, and M. Iori. Pickup-and-delivery problems for goods transportation. In P. Toth and D. Vigo, editors, *Vehicle Routing: Problems, Methods and Applications. MOS/SIAM series on Optimization*, pages 161–192. SIAM, 2014.

[18] M. Bellmore and J. C. Malone. Pathology of traveling-salesman subtour-elimination algorithms. *Operations Research*, 19(2):278–307, 1971.

[19] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische mathematik*, 4(1):238–252, 1962.

[20] G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte. Static pickup and delivery problems: a classification scheme and survey. *Top*, 15(1):1–31, 2007.

[21] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo. The maximum clique problem. In *Handbook of combinatorial optimization*, pages 1–74. Springer, 1999.

[22] J. Bondy and U. Murty. Graph theory, 2008.

[23] F. Bonomo, S. Mattia, and G. Oriolo. Bounded coloring of co-comparability graphs and the pickup and delivery tour combination problem. *Theoretical Computer Science*, 412(45):6261–6268, 2011.

[24] R. Borndörfer. Aspects of set packing partitioning, and covering. *Doctoral Thesis, Technischen Universität Berlin*, 1998.

[25] R. Borndörfer and R. Weismantel. Discrete relaxations of combinatorial programs. *Discrete applied mathematics*, 112(1):11–26, 2001.

[26] S. Borne, R. Grappe, and M. Lacroix. The uncapacitated asymmetric traveling salesman problem with multiple stacks. In A. R. Mahjoub, V. Markakis, I. Milis, and V. T. Paschos, editors, *Combinatorial Optimization*, number 7422 in Lecture Notes in Computer Science, pages 105–116. Springer Berlin Heidelberg, 2012.

[27] G. Carpaneto, M. Dell'Amico, and P. Toth. Exact solution of large-scale, asymmetric traveling salesman problems. *ACM Transactions on Mathematical Software (TOMS)*, 21(4):394–409, 1995.

[28] G. Carpaneto and P. Toth. Some new branching and bounding criteria for the asymmetric travelling salesman problem. *Management Science*, 26(7):736–743, 1980.

[29] F. Carrabs, R. Cerulli, and J.-F. Cordeau. An additive branch-and-bound algorithm for the pickup and delivery traveling salesman problem with lifo or fifo loading. *INFOR: Information Systems and Operational Research*, 45(4):223–238, 2007.

[30] F. Carrabs, R. Cerulli, and M. G. Speranza. A branch-and-bound algorithm for the double travelling salesman problem with two stacks. *Networks*, 61(1):58–75, 2013.

[31] F. Carrabs, J.-F. Cordeau, and G. Laporte. Variable neighborhood search for the pickup and delivery traveling salesman problem with lifo loading. *INFORMS Journal on Computing*, 19(4):618–632, 2007.

[32] M. Casazza, A. Ceselli, and M. Nunkesser. Efficient algorithms for the double traveling salesman problem with multiple stacks. *Computers & Operations Research*, 39(5):1044–1053, 2012.

[33] L. Cassani. Algoritmi euristici per il tsp with rear-loading. *Degree thesis, Università di Milano, Italy*, 2004.

[34] P. Chalasani and R. Motwani. Approximating Capacitated Routing and Delivery Problems. *SIAM Journal on Computing*, 28(6):2133–2149, Jan. 1999.

[35] E. Cheng and W. H. Cunningham. Wheel inequalities for stable set polytopes. *Mathematical Programming*, 77(2):389–421, 1997.

[36] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, DTIC Document, 1976.

[37] V. Chvátal. Edmonds polytopes and weakly hamiltonian graphs. *Mathematical programming*, 5(1):29–40, 1973.

[38] COIN-OR project. LEMON–Library for Efficient Modeling and Optimization in Networks. 2013.

[39] M. Conforti, D. G. Corneil, and A. R. Mahjoub. K_i-covers I: Complexity and polytopes. *Discrete mathematics*, 58(2):121–142, 1986.

[40] M. Conforti, G. Cornuéjols, and G. Zambelli. *Integer programming*, volume 271. Springer, 2014.

[41] J.-F. Cordeau, M. Iori, G. Laporte, and J. J. Salazar González. A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with lifo loading. *Networks*, 55(1):46–59, 2010.

[42] G. Cornuéjols, J. Fonlupt, and D. Naddef. The traveling salesman problem on a graph and some related integer polyhedra. *Mathematical programming*, 33(1):1–27, 1985.

[43] G. Cornuéjols and A. Sassano. On the 0,1 facets of the set covering polytope. *Mathematical Programming*, 43(1-3):45–55, 1989.

[44] J.-F. Côté, C. Archetti, M. G. Speranza, M. Gendreau, and J.-Y. Potvin. A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with multiple stacks. *Networks*, 60(4):212–226, 2012.

[45] J.-F. Côté, M. Gendreau, and J.-Y. Potvin. Large neighborhood search for the pickup and delivery traveling salesman problem with multiple stacks. *Networks*, 60(1):19–30, 2012.

[46] F. D'Andreagiovanni. Pure 0-1 programming approaches to wireless network design. *4OR: A Quarterly Journal of Operations Research*, 10(2):211–212, 2012.

[47] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a Large-Scale Traveling-Salesman Problem. *Journal of the Operations Research Society of America*, 2(4):393–410, Nov. 1954.

[48] S. de Vries. Faster separation of 1-wheel inequalities by graph products. *Discrete Applied Mathematics*, 195:74–83, 2015.

[49] M. Desrochers and G. Laporte. Improvements and extensions to the miller-tucker-zemlin subtour elimination constraints. *Operations Research Letters*, 10(1):27–36, 1991.

[50] J. Desrosiers and M. E. Lübbecke. A primer in column generation. In *Column generation*, pages 1–32. Springer, 2005.

[51] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[52] K. Doerner and J.-J. Salazar-González. Pickup-and-delivery problems for people transportation. In P. Toth and D. Vigo, editors, *Vehicle Routing: Problems, Methods and Applications. MOS/SIAM series on Optimization*, pages 193–212. SIAM, 2014.

[53] I. Dumitrescu, S. Ropke, J.-F. Cordeau, and G. Laporte. The traveling salesman problem with pickup and delivery: polyhedral results and a branch-and-cut algorithm. *Mathematical Programming*, 121(2):269–305, 2010.

[54] J. Edmonds. Maximum matching and a polyhedron with 0, l-vertices. *J. Res. Nat. Bur. Standards B*, 69(1965):125–130, 1965.

[55] G. Erdoğan, G. Laporte, and R. W. Calvo. The one-commodity pickup and delivery traveling salesman problem with demand intervals. *Submitted for publication to Transportation Science*, 2013.

[56] R. Euler, M. Jünger, and G. Reinelt. Generalizations of cliques, odd cycles and anticycles and their relation to independence system polyhedra. *Mathematics of Operations Research*, 12(3):451–462, 1987.

[57] Á. Felipe, M. T. Ortuño, and G. Tirado. The double traveling salesman problem with multiple stacks: a variable neighborhood search approach. *Computers & Operations Research*, 36(11):2983–2993, 2009.

[58] A. Felipe, M. T. Ortuño, and G. Tirado. Using intermediate infeasible solutions to approach vehicle routing problems with precedence and loading constraints. *European journal of operational research*, 211(1):66–75, 2011.

[59] T. A. Feo and M. G. Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133, 1995.

[60] S. Fiorini, S. Massar, S. Pokutta, H. R. Tiwary, and R. D. Wolf. Exponential Lower Bounds for Polytopes in Combinatorial Optimization. *J. ACM*, 62(2):17:1–17:23, 2015.

[61] M. Fischetti. Facets of the asymmetric traveling salesman polytope. *Mathematics of Operations Research*, 16(1):42–56, 1991.

[62] M. Fischetti, A. Lodi, and P. Toth. Solving real-world ATSP instances by branch-and-cut. In *Combinatorial Optimization—Eureka, You Shrink!*, pages 64–77. Springer, 2003.

[63] M. Fischetti and P. Toth. An additive bounding procedure for combinatorial optimization problems. *Operations Research*, 37(2):319–328, 1989.

[64] M. Fischetti and P. Toth. An additive bounding procedure for the asymmetric travelling salesman problem. *Mathematical Programming*, 53(1-3):173–197, 1992.

[65] M. Fischetti and P. Toth. A polyhedral approach to the asymmetric traveling salesman problem. *Management Science*, 43(11):1520–1536, 1997.

[66] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

[67] D. R. Fulkerson. Blocking and anti-blocking pairs of polyhedra. *Mathematical Programming*, 1(1):168–194, Dec. 1971.

[68] M. R. Garey and D. S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4(4):397–411, 1975.

[69] M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 29. WH Freeman, New York, 2002.

[70] R. S. Garfinkel. Technical note—on partitioning the feasible set in a branch-and-bound algorithm for the asymmetric traveling-salesman problem. *Operations Research*, 21(1):340–343, 1973.

[71] M. Gendreau, A. Hertz, and G. Laporte. New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40(6):1086–1094, 1992.

[72] M. Gendreau, A. Hertz, and G. Laporte. The Traveling Salesman Problem with Backhauls. *Comput. Oper. Res.*, 23(5):501–508, May 1996.

[73] M. Gendreau, G. Laporte, and A. Hertz. An Approximation Algorithm for the Traveling Salesman Problem with Backhauls. *Operations Research*, 45(4):639–641, Aug. 1997.

[74] M. Gendreau, G. Laporte, and D. Vigo. Heuristics for the traveling salesman problem with pickup and delivery. *Computers & Operations Research*, 26(7):699–714, July 1999.

[75] A. M. Gerards and A. Schrijver. Matrices with the Edmonds—Johnson property. *Combinatorica*, 6(4):365–379, 1986.

[76] H. Ghaziri and I. H. Osman. A neural network algorithm for the traveling salesman problem with backhauls. *Computers & Industrial Engineering*, 44(2):267–281, Feb. 2003.

[77] R. Gillmann and V. Kaibel. Revlex-initial 0/1-polytopes. *Journal of Combinatorial Theory, Series A*, 113(5):799 – 821, 2006.

[78] F. Glover. Tabu search-part I. *ORSA Journal on computing*, 1(3):190–206, 1989.

[79] F. Glover. Tabu search—part II. *ORSA Journal on computing*, 2(1):4–32, 1990.

[80] F. Glover and M. Laguna. Tabu search, 1997. *Kluwer Academic Publishers*, 1997.

[81] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988.

[82] M. C. Golumbic. *Algorithmic graph theory and perfect graphs*, volume 57. Elsevier, 2004.

[83] R. E. Gomory. An algorithm for integer solutions to linear programs. *Recent advances in mathematical programming*, 64:260–302, 1963.

[84] L. Gouveia and P. Pesneau. On extended formulations for the precedence constrained asymmetric traveling salesman problem. *Networks*, 48(2):77–89, 2006.

[85] L. Gouveia and J. M. Pires. The asymmetric travelling salesman problem and a reformulation of the miller–tucker–zemlin constraints. *European Journal of Operational Research*, 112(1):134–146, 1999.

[86] L. Gouveia and J. M. Pires. The asymmetric travelling salesman problem: on generalizations of disaggregated Miller–Tucker–Zemlin constraints. *Discrete Applied Mathematics*, 112(1–3):129–145, 2001.

[87] M. Grötschel. Polyedrische charakterisierungen kombinatorischer optimierungsprobleme. 1977.

[88] M. Grötschel and M. Jünger. Facets of the linear ordering polytope. *Mathematical Programming*, 33(1):43–60, 1985.

[89] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2. Springer Science & Business Media, 2012.

[90] M. Grötschel, M. Padberg, et al. Polyhedral theory. *The traveling salesman problem*, pages 251–305, 1985.

[91] M. Grötschel and M. W. Padberg. On the symmetric travelling salesman problem i: inequalities. *Mathematical Programming*, 16(1):265–280, 1979.

[92] M. Grötschel and M. W. Padberg. Lineare Charakterisierungen von Travelling Salesman Problemen. *Zeitschrift für Operations Research*, 21(1):33–64, Feb. 1977.

[93] Z. Gu, G. L. Nemhauser, and M. W. Savelsbergh. Sequence independent lifting in mixed integer programming. *Journal of Combinatorial Optimization*, 4(1):109–129, 2000.

[94] A. Gupte. Convex hulls of superincreasing knapsacks and lexicographic orderings. *Discrete Applied Mathematics*, 201:150–163, 2016.

[95] G. Gutin and A. P. Punnen. *The traveling salesman problem and its variations*, volume 12. Springer Science & Business Media, 2006.

[96] P. R. Halmos. *Naive set theory*. Springer Science & Business Media, 1960.

[97] L. Han, B. T. Luong, and S. Ukkusuri. An algorithm for the one commodity pickup and delivery traveling salesman problem with restricted depot. *Networks and Spatial Economics*, pages 1–26, 2015.

[98] P. Hansen, A. Hertz, and J. Kuplinsky. Bounded vertex colorings of graphs. *Discrete Mathematics*, 111(1):305–312, 1993.

[99] H. Hernández-Pérez and J.-J. Salazar-González. Heuristics for the one-commodity pickup-and-delivery traveling salesman problem. *Transportation Science*, 38(2):245–255, 2004.

[100] H. Hernandez-Perez and J.-J. Salazar-Gonzalez. The One-Commodity Pickup-and-Delivery Traveling Salesman Problem : Inequalities and Algorithms. *Networks*, 50(4):258–272, 2007.

[101] H. Hernández-Pérez, I. Rodríguez-Martín, and J. J. Salazar-González. A hybrid GRASP/VND heuristic for the one-commodity pickup-and-delivery traveling salesman problem. *Computers & Operations Research*, 36(5):1639–1645, May 2009.

[102] H. Hernández-Pérez and J.-J. Salazar-González. The One-Commodity Pickup-and-Delivery Travelling Salesman Problem. In M. Jünger, G. Reinelt, and G. Rinaldi, editors, *Combinatorial Optimization — Eureka, You Shrink!*, number 2570 in Lecture Notes in Computer Science, pages 89–104. Springer Berlin Heidelberg, 2003. DOI: 10.1007/3-540-36478-1_10.

[103] H. Hernández-Pérez and J.-J. Salazar-González. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics*, 145(1):126–139, Dec. 2004.

[104] M. I. Hosny and C. L. Mumford. Solving the One-Commodity Pickup and Delivery Problem Using an Adaptive Hybrid VNS/SA Approach. In R. Schaefer, C. Cotta, J. Kołodziej, and G. Rudolph, editors, *Parallel Problem Solving from Nature, PPSN XI*, number 6239 in Lecture Notes in Computer Science, pages 189–198. Springer Berlin Heidelberg, Sept. 2010. DOI: 10.1007/978-3-642-15871-1_20.

[105] J. W. Hunt and T. G. Szymanski. A fast algorithm for computing longest common subsequences. *Communications of the ACM*, 20(5):350–353, 1977.

[106] IBM. IBM ILOG CPLEX Optimization Studio 12.5. 2012.

[107] I. ILOG. Cplex 9.0 reference manual. *ILOG CPLEX Division*, 2003.

[108] M. Iori and S. Martello. Routing problems with loading constraints. *Top*, 18(1):4–27, 2010.

[109] M. Iori and J. Riera-Ledesma. Exact algorithms for the double vehicle routing problem with multiple stacks. *Computers & Operations Research*, 63:83–101, 2015.

[110] S. Irnich and G. Desaulniers. Shortest path problems with resource constraints. In *Column generation*, pages 33–65. Springer, 2005.

[111] K. Jansen. The mutual exclusion scheduling problem for permutation and comparability graphs. *Information and Computation*, 180(2):71–81, 2003.

[112] R. Jonker and T. Volgenant. Transforming asymmetric into symmetric traveling salesman problems. *Operations Research Letters*, 2(4):161–163, 1983.

[113] M. Jünger, G. Reinelt, and G. Rinaldi. The traveling salesman problem. *Handbooks in operations research and management science*, 7:225–330, 1995.

[114] B. Kalantari, A. V. Hill, and S. R. Arora. An algorithm for the traveling salesman problem with pickup and delivery customers. *European Journal of Operational Research*, 22(3):377–386, 1985.

[115] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM, 1984.

[116] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[117] R. M. Karp. A patching algorithm for the nonsymmetric traveling-salesman problem. *SIAM Journal on Computing*, 8(4):561–573, 1979.

[118] J. E. Kelley Jr. Critical-path planning and scheduling: Mathematical basis. *Operations research*, 9(3):296–320, 1961.

[119] L. Khachiyan. A polynomial algorithm for linear programming. *Soviet Math. Dokl.*, (20):191–194, 1979.

[120] S. P. Ladany and A. Mehrez. Optimal routing of a single vehicle with loading and unloading constraints. *Transportation Planning and Technology*, 8(4):301–306, 1984.

[121] S. Lang. *Linear Algebra. Undergraduate texts in mathematics*. Springer-Verlag, New York, 1987.

[122] M. Laurent. A generalization of antiwebs to independence systems and their canonical facets. *Mathematical Programming*, 45(1-3):97–108, 1989.

[123] M. Laurent and A. Sassano. A characterization of knapsacks with the max-flow–min-cut property. *Oper. Res. Lett.*, 11(2):105–110, Mar. 1992.

[124] E. L. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management science*, 18(7):401–405, 1972.

[125] E. L. Lawler. *Combinatorial optimization: networks and matroids*. Courier Corporation, 2001.

[126] A. N. Letchford. On disjunctive cuts for combinatorial optimization. *Journal of Combinatorial Optimization*, 5(3):299–315, 2001.

[127] S. Lin. Computer solutions of the traveling salesman problem. *The Bell System Technical Journal*, 44(10):2245–2269, 1965.

[128] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[129] J. D. Little, K. G. Murty, D. W. Sweeney, and C. Karel. An algorithm for the traveling salesman problem. *Operations research*, 11(6):972–989, 1963.

[130] L. Lovász. A characterization of perfect graphs. *Journal of Combinatorial Theory, Series B*, 13(2):95–98, 1972.

[131] R. M. Lusby and J. Larsen. Improved exact method for the double TSP with multiple stacks. *Networks*, 58(4):290–300, 2011.

[132] R. M. Lusby, J. Larsen, M. Ehrgott, and D. Ryan. An exact method for the double TSP with multiple stacks. *International Transactions in Operational Research*, 17(5):637–652, 2010.

[133] P. C. Mahalanobis. A sample survey of the acreage under jute in bengal. *Sankhyā: The Indian Journal of Statistics*, pages 511–530, 1940.

[134] K. Menger. Das botenproblem. *Ergebnisse eines mathematischen kolloquiums*, 2:11–12, 1932.

[135] M. Mihail. On the expansion of combinatorial polytopes. In *International Symposium on Mathematical Foundations of Computer Science*, pages 37–49. Springer, 1992.

[136] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.

[137] D. Miller and J. Pekny. Results from a parallel branch and bound algorithm for the asymmetric traveling salesman problem. *Operations Research Letters*, 8(3):129–135, 1989.

[138] N. Mladenović, D. Urošević, S. Hanafi, and A. Ilić. A general variable neighborhood search for the one-commodity pickup-and-delivery travelling salesman problem. *European Journal of Operational Research*, 220(1):270–285, July 2012.

[139] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, Nov. 1997.

[140] G. Mosheiov. The Travelling Salesman Problem with pick-up and delivery. *European Journal of Operational Research*, 79(2):299–310, Dec. 1994.

[141] F. Muldoon, W. Adams, and H. Sherali. Ideal representations of lexicographic orderings and base-2 expansion of integer variables. *Operations Research Letters*, 41:32–39, 2013.

[142] G. L. Nemhauser and L. E. Trotter Jr. Properties of vertex packing and independence system polyhedra. *Mathematical Programming*, 6(1):48–61, 1974.

[143] P. Nobili and A. Sassano. Facets and lifting procedures for the set covering polytope. *Mathematical Programming*, 45(1-3):111–137, 1989.

[144] A. Odlyzko. The rise and fall of knapsack cryptosystems. In *Cryptology and Computational Number Theory*, pages 75–88. A.M.S, 1990.

[145] J. A. Pacheco Bonrostro. Problemas de rutas con carga y descarga en sistemas lifo: soluciones exactas. *Estudios de economía aplicada*, (3):69–86, 1995.

[146] J. A. Pacheco Bonrostro. Heurístico para los problemas de rutas con carga y descarga en sistemas lifo. 1997.

[147] J. A. Pacheco Bonrostro. Metaheuristic based on a simulated annealing process for one vehicle pick-up and delivery problem in lifo unloading systems. In *Proceedings of the Tenth Meeting of the Europen Chapter of Combinatorial Optimization (ECCO X). Tenerife, Spain*, 1997.

[148] J. A. Pacheco Bonrostro. *Problemas de rutas con ventanas de tiempo*. Universidad Complutense de Madrid, Servicio de Publicaciones, 2002.

[149] M. Padberg and G. Rinaldi. An efficient algorithm for the minimum capacity cut problem. *Mathematical Programming*, 47(1-3):19–36, 1990.

[150] M. Padberg and G. Rinaldi. Facet identification for the symmetric traveling salesman polytope. *Mathematical programming*, 47(1-3):219–257, 1990.

[151] M. W. Padberg. On the facial structure of set packing polyhedra. *Mathematical programming*, 5(1):199–215, 1973.

[152] M. W. Padberg. Covering, packing and knapsack problems. *Annals of Discrete Mathematics*, 4:265–287, 1979.

[153] S. N. Parragh, K. F. Doerner, and R. F. Hartl. A survey on pickup and delivery problems. Part I: Transportation between customers and depot. *Journal für Betriebswirtschaft*, 58(1):21–51, 2008.

[154] S. N. Parragh, K. F. Doerner, and R. F. Hartl. A survey on pickup and delivery problems. Part II: Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, 58(2):81–117, 2008.

[155] J. F. Pekny and D. L. Miller. A parallel branch and bound algorithm for solving large asymmetric traveling salesman problems. *Mathematical programming*, 55(1-3):17–33, 1992.

[156] J. F. Pekny, D. L. Miller, and D. Stodolsky. A note on exploiting the hamiltonian cycle problem substructure of the asymmetric traveling salesman problem. *Operations research letters*, 10(3):173–176, 1991.

[157] H. L. Petersen, C. Archetti, and M. G. Speranza. Exact solutions to the double travelling salesman problem with multiple stacks. *Networks*, 56(4):229–243, 2010.

[158] H. L. Petersen and O. B. G. Madsen. The double travelling salesman problem with multiple stacks – formulation and heuristic solution approaches. *European Journal of Operational Research*, 198(1):139–147, 2009.

[159] A. Pnueli, A. Lempel, and S. Even. Transitive orientation of graphs and identification of permutation graphs. *Canad. J. math*, 23(1):160–175, 1971.

[160] H. N. Psaraftis. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2):130–154, 1980.

[161] H. N. Psaraftis. Analysis of an $O(N^2)$ heuristic for the single vehicle many-to-many euclidean dial-a-ride problem. *Transportation Research Part B: Methodological*, 17(2):133–145, 1983.

[162] M. Queyranne and Y. Wang. Hamiltonian path and symmetric travelling salesman polytopes. *Mathematical Programming*, 58(1-3):89–110, 1993.

[163] S. Rebennack. Stable set problem: Branch & cut algorithms stable set problem: Branch & cut algorithms. In *Encyclopedia of Optimization*, pages 3676–3688. Springer, 2008.

[164] S. Rebennack, M. Oswald, D. O. Theis, H. Seitz, G. Reinelt, and P. M. Pardalos. A branch and cut solver for the maximum stable set problem. *Journal of combinatorial optimization*, 21(4):434–457, 2011.

[165] G. Reinelt. TSPLIB—A traveling salesman problem library. *ORSA journal on computing*, 3(4):376–384, 1991.

[166] M. G. Resende, C. C. Ribeiro, F. Glover, and R. Martí. Scatter search and path-relinking: Fundamentals, advances, and applications. In *Handbook of metaheuristics*, pages 87–107. Springer, 2010.

[167] R. Roberti and P. Toth. Models and algorithms for the asymmetric traveling salesman problem: an experimental comparison. *EURO Journal on Transportation and Logistics*, 1(1-2):113–133, 2012.

[168] J. Robinson. On the hamiltonian game (a traveling salesman problem). Technical report, DTIC Document, 1949.

[169] K. Ruland and E. Rodin. The pickup and delivery problem: Faces and branch-and-cut algorithm. *Computers & mathematics with applications*, 33(12):1–13, 1997.

[170] A. H. Sampaio and S. Urrutia. New formulation and branch-and-cut algorithm for the pickup and delivery traveling salesman problem with multiple stacks. *International Transactions in Operational Research*, 2016.

[171] S. C. Sarin, H. D. Sherali, and A. Bhootra. New tighter polynomial length formulations for the asymmetric traveling salesman problem with and without precedence constraints. *Operations Research Letters*, 33(1):62–70, 2005.

[172] A. Sassano. On the facial structure of the set covering polytope. *Mathematical Programming*, 44(1-3):181–202, 1989.

[173] C. Schensted. Longest increasing and decreasing subsequences. *Canad. J. Math*, 13(2):179–191, 1961.

[174] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 2001.

[175] Y. Sekiguchi. A note on node packing polytopes on hypergraphs. *Operations Research Letters*, 2(5):243–247, 1983.

[176] P. D. Seymour. The matroids with the max-flow min-cut property. *Journal of Combinatorial Theory, Series B*, 23(2-3):189–222, 1977.

[177] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming—CP98*, pages 417–431. Springer, 1998.

[178] M. Sigurd, D. Pisinger, and M. Sig. Scheduling transportation of live animals to avoid the spread of diseases. *Transportation Science*, 38(2):197–209, 2004.

[179] T. H. Smith, V. Srinivasan, and G. Thompson. Computational performance of three subtour elimination algorithms for solving asymmetric traveling salesman problems. *Annals of Discrete Mathematics*, 1:495–506, 1977.

[180] D. M. Stein. An asymptotic, probabilistic analysis of a routing problem. *Mathematics of Operations Research*, 3(2):89–101, 1978.

[181] R. E. Tarjan. Finding optimum branchings. *Networks*, 7(1):25–35, 1977.

[182] P. Toth and D. Vigo. *Vehicle routing: problems, methods, and applications*, volume 18. Siam, 2014.

[183] S. Toulouse and R. Wolfler Calvo. On the complexity of the multiple stack tsp, kstsp. In *Proceedings of the 6th Annual Conference on Theory and Applications of Models of Computation*, TAMC '09, pages 360–369, Berlin, Heidelberg, 2009. Springer-Verlag.

[184] S. Urrutia, A. Milanés, and A. Løkketangen. A dynamic programming based local search approach for the double traveling salesman problem with multiple stacks. *International Transactions in Operational Research*, 22(1):61–75, 2015.

[185] P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information processing letters*, 6(3):80–82, 1977.

[186] F. Vanderbeck and L. Wolsey. Reformulation and decomposition of integer programs. In M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 431–502. Springer Berlin Heidelberg, 2010.

[187] F. Wang, A. Lim, and Z. Xu. The one-commodity pickup and delivery travelling salesman problem on a path or a tree. *Networks*, 48(1):24–35, Aug. 2006.

[188] D. P. Williamson and D. B. Shmoys. *The design of approximation algorithms*. Cambridge University Press, 2011.

[189] F. Zhao, S. Li, J. Sun, and D. Mei. Genetic algorithm for the one-commodity pickup-and-delivery traveling salesman problem. *Computers & Industrial Engineering*, 56(4):1642–1648, May 2009.