Tesis Doctoral

Enfoques de programación entera para el Problema del Viajante de Comercio con costos en función del tiempo

Miranda Bront, Juan José

2012

UNIVERSIDAD DE BUENOS AIRES

Facultad de Ciencias Exactas y Naturales

Departamento de Computación

Enfoques de programación entera para el Problema del Viajante de Comercio con costos en función del tiempo

Tesis presentada para optar al título de
Doctor de la Universidad de Buenos Aires
en el área Ciencias de la Computación

Juan José Miranda Bront

Directoras de tesis: Dra. Isabel Méndez Díaz y Dra. Paula Zabala

Consejera de estudios: Dra. Isabel Méndez Díaz

Lugar de Trabajo: Departamento de Computación, FCEyN, Universidad de Buenos Aires.

Buenos Aires, 2012

Enfoques de programación entera para el Problema del Viajante de Comercio con costos en función del tiempo

El Problema del Viajante de Comercio Dependiente del Tiempo (PVCDT) es una generalización del clásico Problema del Viajante de Comercio (PVC) en el cual se busca un circuito de costo mínimo que recorra todos los clientes, donde los tiempos (o costos) de viaje entre ellos no son constantes y pueden variar dependiendo de diversos factores. Una de las motivaciones para considerar la dependencia del tiempo es que nos permite tener una mejor aproximación a muchas aplicaciones de la vida real, dentro de la industria y en el sector de servicios. En la literatura relacionada, existen distintas versiones del PVCDT que consideran que las variaciones se producen por diversos motivos y sobre distintos parámetros. Algunas de ellas estipulan que las variaciones se producen sobre los tiempos (y/o costos) de viaje mientras que otras asumen que las variaciones se producen sobre la velocidad de viaje. Más aún, para el primer caso, una variante asume que el tiempo de viaje depende de la posición del arco en el circuito mientras que otra asume que depende del instante en el que se comienza a atravesar el arco. Al igual que el PVC, el PVCDT pertence a la clase $\mathcal{NP}$-Difícil y por lo tanto no se conoce un algoritmo que encuentre una solución en tiempo polinomial.

En esta tesis abordamos dos de las variantes antes mencionadas mediante la formulación de modelos de Programación Lineal Entera. Para cada formulación, realizamos un estudio teórico enfocado en derivar familias de desigualdades válidas que exploten las características particulares del problema. En particular, para una de las variantes, demostramos que varias de ellas definen facetas del poliedro. Desde el punto de vista práctico, para las variantes consideradas desarrollamos algoritmos exactos de tipo Branch and Cut que incorporan estas familias de desigualdades válidas y los evaluamos considerando instancias propuestas en la literatura, obteniendo buenos resultados que muestran que ambos enfoques son factibles para ser utilizados en la práctica.

Palabras clave: *problema del viajante de comercio dependiente del tiempo, optimización combinatoria, programación lineal entera mixta, algoritmos branch and cut, desigualdades válidas, ventanas de tiempo, caminos infactibles.*

Integer Programming approaches to the Time Dependent Travelling Salesman Problem

The Time Dependent Travelling Salesman Problem (TDTSP) is a generalization of the well known Travelling Salesman Problem (TSP) in which we look for a minimum cost tour that visits each client exactly once where the travel times (or costs) among them are not assumed to be constant and may vary depending on many different factors. The motivation to consider the time dependence factor is that it enables to have better approximations to many problems from practice, mainly from the industry and the service sector. In the related literature there are different versions of the TDTSP that consider that variations occur for different reasons and over distinct parameters. For example, some of them consider that variations affect travel times (and/or costs) while others assume that variations influence travel speeds. Moreover, in the first case, one version assumes that the travel time depends on the position of the arc in the tour while another one assumes that it depends on the particular starting instant for travelling the arc. As well as the TSP, the TDTSP belongs to the class $\mathcal{NP}$-Hard and therefore it is not known an algorithm that solves it in polynomial time.

In this thesis we approach two of the versions mentioned above by means of Integer Linear Programming formulations. For each formulation, we perform a theoretical study focused on deriving families of valid inequalities that exploit the particular characteristics of the problem. In particular, for one of the variants, we prove that some of them are facet defining. From a practical standpoint, for the versions considered we develop exact Branch and Cut algorithms that incorporate these families of valid inequalities and we evaluate them on instances from the related literature, obtaining good computational results that show that both approaches are feasible to be used in practice.

Keywords: *time dependent travelling salesman problem, combinatorial optimization, mixed integer linear programming, branch and cut algorithm, valid inequalities, time windows, infeasible paths.*

A Lupe y Mostaza,

"No permitan que el fracaso les deteriore la autoestima. Cuando ganás, el mensaje de admiración es tan confuso, te estimula el amor hacia uno mismo y eso deforma mucho. Y cuando perdés sucede todo lo contrario, hay una tendencia morbosa a desprestigiarte, a ofenderte, sólo porque perdiste. En cualquier tarea se puede ganar o perder, lo importante es la nobleza de los recursos utilizados, lo importante es el tránsito. La dignidad con que recorrí el camino en la búsqueda del objetivo. Lo otro es cuento para vendernos una realidad que no es tal."

"Do not let failure damage your self-esteem. When you win, the message of admiration is so confusing, it encourages the love to yourself and that deforms you a lot. And when you loose, the exact opposite happens, there is a morbid tendency to disqualify, to offend, only because you have lost. In any activity, you can win or loose, but the important thing is nobility of the resources used, the passage. The dignity employed to walk the path towards the objective. The rest is a tale to sell you a reality that is not such."

Marcelo Bielsa.

(La vida por el fútbol: Marcelo Bielsa, el último romántico. Román Iutch, 2010.)

Acknowledgements

This Thesis is the result of five years of study, learning, researching, teaching, getting married, attending to conferences, adopting a dog, coffees, *mates*, discussions and seminars, among many other activities. All these gave me the opportunity of learning a lot about integer programming, combinatorial optimization and vehicle routing problems, discovering how fascinating this area of research is. Even though, there are two other people responsible of all these findings: Isabel Méndez-Díaz and Paula Zabala. Their advices, teaching, thoughts, ideas and dedication were priceless; their passion for the work, kindness and generosity were essential. Working with them was a privilege and a pleasure. They really went above and beyond what an advisor is required to do, exceeding the limits of work. When I started my PhD, I had two advisors; now I have two friends. I will always be grateful for this.

I want to thank specially to Paolo Toth, who worked with us in the time dependent tsp with time windows. I want to thank him for his ideas, attention, kindness and consideration, giving me the opportunity of spending six wonderful months in Bologna. They were for sure one of the most enriching experiences I've ever had, both at a professional and personal level. I also want to thank Profs. Cid Carvalho de Souza, Nicolás Stier and Daniele Vigo for reading the thesis, for their cooperation and for the invaluable comments and suggestions which improved the earlier version of this document.

In addition, I want to thank to the members of the Operations Research, Combinatorial Optimization and Graphs Group for letting me work with them since I was finishing my undergraduate studies. This is extensive to all the people that is (or was) in this hidden corridor in the second floor during this period. I also want to thank to Pablo, a comrade of journeys, with whom I shared these five years of struggles, victories and defeats included.

Finally, but for sure not last, I want to thank my wife, Laura, the other essential part of this story. Without you, none of this could have been possible. I want to thank you for being always there; for making tomorrow better than today; for Bologna. For reading three times this document correcting the English; for spending the weekends of the last six months at home with me; for making a second division soccer match the funniest part of the day. For having Mostaza, our dog, which completes our family. For being you. This achievement is not mine, is ours. And do not worry. Finally, vacations are around the corner.

April 9^{th}, 2012.

Contents

List of Figures

List of Tables

List of Algorithms

1. Introduction

1.1. Vehicle Routing Problems

Combinatorial Optimization problems arise in many real-world situations and disciplines, from Genetics, Physics and Chemistry to Finances, Marketing and Industry. Generally, this kind of problems are easy to model mathematically but hard to solve computationally.

In the industry and the services sector, transportation costs represent a significant portion of the total goods' or services' value. Vehicle Routing Problems (VRPs) is the name given to a wide family of problems related to goods distribution and service provision across a network. Vehicles depart from fixed points, usually called depots. Each path (arc) between any two clients (locations) has an associated cost that may vary depending on many different factors, such as the type of the vehicle or the moment of the day it is taken.

Some examples of this kind of problems are: the mail pickup and distribution, the route of a doctor that visits patients at their homes, food or goods delivery to the corresponding clients, etc. One of the VRPs that caught most of the attention in the last decades is the Travelling Salesman Problem (TSP). In this case, a vehicle must visit each city (or client) exactly once, starting from and returning to the depot. The objective is to minimize the total travel time.

The main characteristics of VRPs are given by the operating constraints or feasibility rules that the routes for each vehicle must satisfy, such as not to exceed the capacity of the vehicle or to satisfy the precedence relations among the clients. Furthermore, other common constraints are:

- every client has an associated demand. Moreover, there may be restrictions on the way such demand is supplied (for example, by only one vehicle or by more than one).
- every client can be visited within a particular time-range during the day. This constraint usually arises in supply problems in big cities. Transit regulations often forbid load/unload labours during certain hours to avoid traffic jams. Depending on the type of the industry, the delivery of goods must be done before a particular time limit (for example, newspaper distribution). This constraint is usually referred in the literature as *time window*.
- the number and capacity of available vehicles. The vehicle fleet can be homogeneous (vehicles with the same capacity) or heterogeneous (different capacities, size, or other characteristics). In some cases, clients may have restrictions with respect to the size of the vehicle supplying them (for example, weight limits on the highways required to visit one of the clients).

- number of depots. The company can have several supply depots, each of them having associated a subset of the clients.
- starting and ending points of the routes. In general, vehicles return to the departing depot after finishing the work. However, this condition can be relaxed in some situations (for example, presence of multiple depots, a schoolbus driver who owns the bus, etc).
- transportation network. Usually, it is assumed that there exists a communication path between every pair of instances. However, in some problems, the network cannot be assumed to be complete.
- travel times and costs. Between any pair of clients, the travel cost, time and velocity can vary depending on different circumstances, such as the distance between the clients, the size of the vehicle, the time of the day the arc is travelled (modelling rush hours and traffic conditions on big cities), etc.

In addition to the feasibility constraints, there is a function to be optimized associated to each possible route. It can be different from one situation to another, depending on the interests and objectives wanted in every particular situation. Some of these objectives could be:

- minimizing the total travel time (or cost).
- minimizing the waiting at the clients.
- minimizing the number of vehicles used.
- minimizing the total duration of the routes.

Each combination of the factors mentioned above leads to a particular vehicle routing problem. Considering or not a particular feature may change the difficulty of the problem and techniques which are proven to be effective for one problem may not be a good approach for others. For these reasons, it is very important to study each problem separately, considering its particularities and exploiting them. We refer to Toth and Vigo [58] for a complete and detailed description of many different vehicle routing problems. In this work, we tackle one of such problems, which can be applied in many real world situations.

1.2. The Time-Dependent Travelling Salesman Problem

In terms of graph theory, vehicle routing problems can be modelled naturally by considering a complete digraph, where vertices represent clients and an arc the possibility of travelling directly between two clients. Possible Hamiltonian tours or paths of the graph stand for different orders in which clients can be visited. As mentioned in the previous section, depending on the objective function considered, we can define different problems.

The classical TSP can be modelled as follows. Consider a complete digraph $D = (V, A)$, where $V = \{0, 1, \ldots, n\}$ is the set of clients and A the set of arcs linking them. Each arc $(i, j) \in A$, $i \neq j$, has associated a positive cost -or distance, or travel time- c_{ij}. The objective is to find a

tour visiting all vertices in V at minimum total cost. This problem belongs to the class $\mathcal{NP}$-Hard and has received great attention in the last decades (see, for example, [4, 34]), providing a wide variety of ideas and algorithms to solve it. One example of this process is *Concorde* [14], a computer code for the *symmetric* TSP, i.e., $c_{ij} = c_{ji}$, that is able to solve instances with hundreds of clients in just a few seconds. Indeed, these advances on the TSP have been strongly related with the development and techniques for general mathematical programming problems, in particular with general purpose *Mixed Integer Linear Programming* problems which are briefly described in the following chapter.

In practical situations, traffic conditions are heterogeneous and usually produce significant changes on a priori estimated parameters. In the case of big cities, assuming that travel times or costs are constant during the complete planning horizon may generate a missing of valuable information and therefore incur in considerable error when planning the travel. For example, in Buenos Aires, driving downtown during rush hours may take twice the time than doing it at noon. Even intra-city transportation can vary widely depending on the hour or the neighbourhood to be visited.

Similar situations arise from job scheduling applications. Given a set of jobs to be processed on a machine with transitions among them that require a certain time and/or cost. In these problems, the objective is to provide a schedule that minimizes a certain function (cost, tardiness, makespan, etc.). As well as with routing problems, transitions between a pair of jobs may depend on many different factors, for example, staff availability, restrictions depending on the moment of the transition, etc. Thus, in many situations, considering a non-fixed cost for the transitions between pairs of jobs can improve the decisions taken.

The *Time-Dependent Travelling Salesman Problem* (TDTSP) is a generalization of the classical TSP in which the cost of the travel between two clients is not assumed to be constant and depends not only on the distance between them, but also on other factors such as, for example, the time of the day the arc is travelled. This fact represents a better approximation to situations as described above, where the travel times or costs are usually variable.

In its general version, the TDTSP can be stated as follows: Consider a complete digraph $D = (V, A)$, with $V = \{0, 1, \ldots, n\}$ the set of vertices and A the set of directed arcs. Assume that there exists an $(n+1) \times (n+1)$ time-dependent cost matrix $C(t_i) = [c_{ij}(t_i)]$ that associates a cost $c_{ij}(t_i)$ to each arc (i, j) in A, where $c_{ij}(t_i)$ is a time-dependent cost function for the arc (i, j). Vertex 0 is a special vertex representing the depot. The TDTSP involves finding a minimum cost tour that visits each vertex exactly once, starting at and returning to vertex 0.

In the related literature, the TDTSP is used indistinctly to refer to a wide variety of problems where travel times, costs or even velocities are assumed to be variable. Depending on the assumptions stated and the characteristics of the problem under consideration we can obtain different approaches to solve it. We give next a summary of some of these TDTSP variations.

1. *Position dependent travel times.* This version of the TDTSP assumes that the travel times vary depending on the position of the arc in the tour, i.e., $c_{ij}(k) = c_{ijk}$ if arc (i, j) is placed in the k-th position of the tour. It can be applied to scheduling problems on one machine with sequence dependent setup times, and is considered by Fox [24]. Furthermore, it generalizes the well known *Travelling Deliveryman Problem* (TDP), in which the objective

is to find a tour that visits all clients and minimizes the average waiting time for clients to be served. Exact approaches for the TDTSP can be found in [13, 25, 29, 53, 61] and for the TDP in [12, 22, 40, 45, 59, 64]. A very recent paper by Abeledo et al. [1] shows good computational results considering a *Branch and Cut and Price* approach to the TDTSP over TDP benchmark instances from the literature. Heuristics for the TDTSP and the TDP can be found in [60] and [54, 63], respectively.

2. *Time dependent travel times.* This version assumes that the travel time depends on the starting moment for the arc to be travelled. It is introduced by Malandraki and Daskin [42] for the multiple vehicle case, which is motivated mainly to achieve a better approximation for urban congestion. The approach assumes that the travel time function of each arc is not constant over time and variations take place at different points in time. The objective is to minimize the makespan of the route. Within each of these *time periods* the travel time function is assumed to be constant. Stecco et al. [57] also consider this version in the single-vehicle version to solve a scheduling problem where transition times follow a similar pattern, having restricted intervals where some activities cannot be carried out. One of the characteristics of this approach is that the *FIFO* (First-in, First-out) condition does not hold and sometimes it may be better to delay the departure in order to arrive earlier at the destination. However, defining time periods for each arc with fine enough granularity an almost-FIFO condition can be imposed by choosing the dynamic travel times accordingly.

   Albiach et al. [3] also consider the single-vehicle version and include time windows. They consider time-dependent travel costs but waiting times at the departure are not allowed. In a follow-up paper, Soler et al. [56] generalize the approach to multiple vehicles and also allow waiting for the departure. In a recent research, Persiani [52] solves an aviation problem which is a particular case of this version of the TDTSP. Several heuristics have been proposed for similar versions of this problem (see, for example, Malandraki and Dial [43], Li et al. [38], Bentner [11]). Some of them consider a proportional travel time calculation concerning time periods changes.

3. *Time dependent travel speeds.* This version of the problem is similar to the previous one, but considering travel speed changes which, of course, influence travel times. Hill and Benton [35] first introduced a model following this idea, which showed to be quite effective in practice. Ichoua et al. [36] consider an improved version of this model which satisfies the FIFO condition. They develop a metaheuristic for the multiple-vehicle case. In a very recent paper, Cordeau et al. [15] propose an exact approach for the single-vehicle version of this problem, the TDTSP.

All these versions show that the TDTSP has a wide variety of applications in routing and scheduling problems and further improvements can be achieved by including explicitly the time dependency factor in the problem formulation. Furthermore, from the results shown in the articles mentioned above, the TDTSP appears as a challenging problem from a computational standpoint. For these reasons, as we mention in the following section, we will concentrate on the first two variants of the TDTSP mentioned, hoping to contribute to the advance and development of efficient algorithms for the TDTSP.

1.3. Our contributions

The different versions of the TDTSP are natural generalizations of the classical TSP where the travel times (or costs, or speeds) can vary depending on different circumstances and the case under study. The benefit of considering these generalizations relies on obtaining a better approximation to real world problems, being able to account for situations that the TSP cannot deal with. As a counterpart, the structure of the problems becomes more complex and therefore improved models and algorithms are required to tackle the problem. For example, for the TSP instances with hundreds of vertices are solved to optimality in just a few seconds, while for the different versions of the TDTSP exact approaches are capable of solving instances with only tens of vertices in a reasonable amount of time. Intuitively, this difference on the difficulty appears as reasonable given that any of the variants of the TDTSP represents a generalization of the TSP.

Mixed Integer Linear Programming (MILP) techniques have proven to be a very effective approach to develop exact algorithms for vehicle routing problems and combinatorial optimization problems in general. Mostly, the success of these type of algorithms relies in performing a deep study of the problem under consideration and exploiting this particular information by including it in specific-purpose algorithms. The aim of this thesis goes in that direction. We consider two of the versions of the TDTSP mentioned in the previous section and the objective is to address these problems with exact algorithms, providing new developments and techniques that incorporate the special characteristics of each version.

In Chapter 3 we consider the *position dependent* version of the TDTSP, emphasizing our work on the theoretical aspect. Picard and Queyranne [53] present two ILP models for the problem and report computational results for a B&B algorithms. Later, in parallel, Vander Wiel and Sahinidis [60] and Gouveia and Voss [29] propose new models for this version of the TDTSP, which turns out to be the same despite some minor differences regarding integrality conditions on the variables. In a follow up paper, Vander Wiel and Sahinidis [61] consider a Benders Decomposition based approach using general purpose cuts. In fact, they mention that it would be interesting to study the polytope to derive new valid inequalities for the model. We focus our efforts in this aspect. We consider the flow-based formulation from Picard and Queyranne [53] and the formulation proposed by Vander Wiel and Sahinidis [60, 61] and Gouveia and Voss [29]. In particular, using results from Balas and Oosten [9], we prove that there is a strong relation between the two polytopes defined by each of these formulations and that inequalities which are valid for the latter can be adapted to be valid for the former and vice versa. Intuitively, the main difference between the two formulations is that the latter includes also vertex variables, but the idea behind the model is the same as in the former. Based on this fact, we perform a polyhedral study and derive ten families of valid inequalities for both polytopes. Six of these families are based on the idea of *time dependent cycles*. One of these families is called *Time Dependent Cycle Inequalities* and its objective is to forbid cycles that do not visit all vertices. However, we show that this family can be strengthened by applying a sequential lifting procedure over some specific variables. We derive five different families of valid inequalities, the *Lifted TDCI*, obtained as the result of applying five different liftings and we prove that they are facet defining. In parallel with our research, Abeledo et al. [1] provide several families of valid inequalities for the TDTSP. In particular, one of the families proposed

by them, the *r-cycle* inequalities, is a particular case of the lifted TDCI described in Chapter 3. Indeed, the theoretical framework we provide allows to derive more families of facets by means of applying a lifting procedure considering a different order of the variables involved. For practical purposes, we develop a Branch and Cut algorithm that incorporates the valid inequalities derived as well as two primal heuristics. We tested the algorithm on benchmark instances from the related TDP literature, obtaining good computational results and showing that the approach is suitable to be used in practice.

In Chapter 4 we study the version with *time dependent travel times and costs* and time windows, named TDTSP-TW. Waiting times both at the arrival at and the departure from a vertex are also allowed. In this case, we emphasize on the computational aspect of the problem. In the related literature, MILP formulations for the TDTSP usually include big-M constraints to handle the time dependency, known to produce poor LP relaxations. This is the case of the formulations from Malandraki and Daskin [42] and Stecco et al. [57]. Similarly to the *TSP with time windows* (TSPTW), we consider *infeasible paths* to handle the time dependency and the time windows without including big-M inequalities in the formulation. The results presented in this chapter are built mainly upon the papers from Ascheuer et al. [5] regarding infeasible paths and the very recent paper from Dash et al. [19] which proposes a *Time Bucket Formulation* (TBF) for the TSPTW. The latter provides strengthened versions of several families of valid inequalities used for the TSPTW by means of including specific information provided by the formulation. We generalize and adapt the definitions of infeasible paths and the TBF to the time dependent case. In the TDTSP-TW, due to the nature of the problem, we do not assume that the *triangle inequality* holds for the travel times and also the infeasibility of a path depends on the time periods each arc is travelled. Under these conditions, theoretical and practical results from previous research developed for the TSPTW cannot be directly applied to the TDTSP-TW. Thus, we generalize the models proposed in Ascheuer et al. [5] and the TBF from Dash et al. [19] to account for the time dependency factor and consider adapted versions of the formulations from Malandraki and Daskin [42] and Stecco et al. [57] to minimize the total travel cost of the tour instead of the makespan. On preliminary computational results, we observed that the *Time Dependent* version of the TBF, named TDTBF, produces the best results in terms of computational times. Thus, we focus on this formulation and derive time dependent generalizations of several of the valid inequalities derived for the TSPTW as well as strengthened versions of inequalities derived for the TDTSP using specific information from the TDTBF. In order to provide an exact approach for the TDTSP-TW, we develop a Branch and Cut algorithm that incorporates the results mentioned above as well as a preprocessing stage and an initial heuristic. The algorithm is evaluated on a set of instances generated based on criteria from the related literature, obtaining a good performance in general and showing that the overall approach is quite effective.

The remaining of this document is organized as follows. In Chapter 2 we provide a brief review of general algorithms and approaches for MILP problems. Then, in Chapter 3, we consider the position-dependent version of the TDTSP and present the theoretical and computational results concerning this version of the TDSTP. Similarly, in Chapter 4 we show the formulations and results for the TDTSP-TW. Finally, in Chapter 5 we show conclusions of the work developed during this research and present some open problems and possible new research lines derived from our work.

2. Preliminaries

Many combinatorial optimization problems can be modelled with Mixed Integer Linear Programming (MILP) formulations. Generally speaking, MILP models are easy to formulate and there is usually more than one alternative to consider. The purpose of this chapter is to describe the main concepts and algorithms for MILPs, assuming the reader is familiarized with the basic ideas of Linear Programming (LP).

2.1. Mixed Integer Linear Programming

During the last decades, the use of MILP models to solve combinatorial optimization problems has been widely accepted and it is still one of the most effective techniques to solve hard combinatorial problems. By means of such formulations we can model situations where we aim to minimize or maximize a linear function for which feasible solutions must satisfy a set of constraints, also linear, where some subset of the variables defined for the problem (eventually, all of them) are required to be integer. These are the cases of the well known *Travelling Salesman Problem* (TSP), or optimization problems in networks, resource assignment, graph theory, and many others coming from a great diversity of disciplines.

Formally, an MILP problem looks for the optimum solution of a linear function among the vector space in a polyhedron characterized by linear equations and inequalities. Then, the formulation of this type of problems reads:

$$\begin{aligned} \min \quad & \sum_{j \in I} c_j x_j + \sum_{j \in C} c_j x_j \\ \text{s.t.} \quad & \sum_{j \in I} a_{ij} x_j + \sum_{j \in C} a_{ij} x_j \geq b_i, \qquad i = 1, \ldots, m \\ & x_j \in \mathbb{Z}_+ \qquad \forall\ j \in I \\ & x_j \in \mathbb{R}_+ \qquad \forall\ j \in C \end{aligned} \tag{2.1}$$

where I is the set of integer variables and C the set of continuous variables, with $I \cup C = \{1, \ldots, n\}$. Vector $c \in \mathbb{R}^n$ represents the objective function and coefficients a_{ij} can be expressed as a constraint matrix $A \in \mathbb{R}^{m \times n}$. If $C = \emptyset$, the problem is referred as *pure Integer Linear Program* (ILP). For notational purposes, problem (2.1) can be also expressed as

$$\min\{cx \ :\ Ax \geq b,\ x \geq 0,\ x_j \in \mathbb{Z}_+\ \forall j \in I\}.$$

This problem in its general form belongs to the class $\mathcal{NP}$-Hard, that means that so far it is not known a polynomial time algorithm able to solve it.

The polyhedron associated with an MILP formulation is defined as $P = \{x \in \mathbb{R}^n_+ \ : \ Ax \leq b\}$ and the set of feasible solutions as $S = P \cap \{x \in \mathbb{R}^n \ : \ x_j \in \mathbb{Z} \ \forall \ j \in I\}$. The *linear relaxation* (LP) of problem (2.1) is

$$\min\{cx \ : \ Ax \geq b, \ x \geq 0\}. \tag{2.2}$$

The most important difference between the LP and problem (2.1) is that the former has a polynomial time algorithm capable of solving it (see Karmarkar [37]). Moreover, LP algorithms work quite well in practice, solving instances with a big number of variables and constraints efficiently.

We denote by $conv(S)$ to the convex hull of S, the smallest polyhedron containing S. There are instances of MILPs which have the particularity that can be solved in polynomial time. For example, if $P = conv(S)$ and the number of constraints required to describe $conv(S)$ is polynomial, then any LP algorithm can be used to solve it to optimality. However, this occurs only in a limited number of cases and all problems belonging to the class $\mathcal{NP}$-Hard do not satisfy this condition.

When the characterization of $conv(S)$ by linear constraints is not known, the LP relaxation proves very useful. If it turns out to be an infeasible problem, so it is the MILP problem as well. If the optimal solution satisfies the integrality constraints, it is also a solution of the MILP problem. In case it is feasible but not integral, the optimum value of the LP relaxation is a lower bound of the optimum value of the MILP problem. Hence, providing with LP relaxations as tight as possible gives more importance to this value. The difference between the optimum value of the MILP problem and the one of the LP relaxation (*gap*) is a measure that is generally used to evaluate the quality of the relaxation.

Then the difference between the value of a feasible solution of the MILP problem and the optimal value of the LP relaxation gives an upper bound for the gap. This is useful to evaluate the quality of a solution and in some cases to certify its optimality.

Many of the algorithms for solving MILP problems are based on the relation between the MILP problem and the LP relaxation. Some of the algorithms most used in practice are described in the next section.

2.2. Algorithms for MILP problems

Most of the exact methods and algorithms for solving an MILP model fall into one of the following schemes:

- *Cutting planes* algorithms
- *Branch and Bound* (B&B) algorithms
- *Branch and Cut* (B&C) algorithms
- *Branch and Price* (B&P) algorithms

2.2.1. Cutting plane algorithms

The main idea behind a cutting plane algorithm lies on successive improvements of the LP relaxation by means of the addition of valid linear inequalities for $conv(S)$ to eliminate (cut) LP solutions which do not belong to $conv(S)$. The general scheme of the algorithm begins by omitting the integrality constraints for the variables and solving the LP relaxation. If at least one integer variable has a fractional value, then it tries to identify a valid linear inequality for $conv(S)$ that *separates* the present LP solution from the set of integer feasible solutions S. Adding this inequality to the formulation results in a new and tighter relaxation of the problem, which allows us to repeat the procedure.

The success of this methodology depends mostly on the possibility and the efficiency of finding violated valid inequalities (*cutting planes*, or simply *cuts*) that can be added to the formulation to separate fractional solutions. That means that a *separation algorithm* is required.

The basic scheme of a cutting plane algorithm reads as follows

Algorithm 2.1 General Cutting Plane Algorithm

Input: An MILP formulation.

1. *Initialization*: Consider the LP relaxation of the problem.
2. *Solving the LP relaxation*: Let x^* be the optimum solution of the LP relaxation. If $x_j^* \in \mathbb{Z}\ \forall\ j \in I$, then stop. Otherwise, go to 3.
3. *Separation*: Search for valid inequalities violated by x^*. If found, add them to the current problem and go to 2. Otherwise, stop. The problem could not be solved.

Cutting planes can be generated by two different approaches:

- General purpose tools applicable to any MILP problem.

  In the early 60's, Gomory [28] developed a general algorithm to generate valid inequalities to cut the current solution of an LP relaxation. In each iteration, the inequality is generated based on the expression of the basic variables in terms of the non-basic ones using exclusively integrality arguments. It is a general algorithm, considering that it does not use any particular information about the problem (besides integrality). Under certain conditions, it converges to the optimal solution of the original problem. This result is extremely important from the theoretical point of view but in practice it is not efficient to solve MILPs in a reasonable amount of time.

  Subsequently, many algorithms have been developed using a variety of general purpose cuts such as *disjunctive*, *clique*, *cover*, etc (see, e.g., Nemhauser and Wolsey [47]). While these algorithms have very interesting properties from the theoretical aspect, their success is controversial. On one side, their main advantage is that they can be applied to any MILP problem independently of its structure. On the other hand, they are not always the most adequate tool for particular cases. In general, a more specific and structure-dependent study provides better results and is more helpful on devising accurate procedures.

- Exploiting the particular structure of the problem

  There exist properties inherent to each particular problem that may help to identify better cutting planes. The first paper in the literature to consider this approach is the one by Dantzig et al. [18] in 1954 for the TSP. With the technique proposed, the authors were able to solve an instance with 49 cities (large for that time) and laid the foundations of what today is one of the most effective tools: polyhedral techniques.

  A desired property for a cutting plane is to eliminate as much as possible non-integer solutions. Since cutting planes are valid inequalities for $conv(S)$, it is reasonable to expect the *facets* of $conv(S)$ to be more useful. The first polyhedral studies were performed during the 70's for the *Independent Set Problem* by Padberg [51] and for the TSP by Gröetschel and Padberg [32, 33]. These works meant a remarkable advance on the resolution of such type of problems.

  Aiming to obtain new and particular algorithms, the polyhedral study has to be carried out together with the development of efficient separation routines. In this direction, Grötschel et al. [31] established one of the most important theoretical results that relate the complexity between the separation and the optimization problems. The result states that the optimization problem $\max\{cx : x \in conv(S)\}$ can be solved in polynomial time if and only if the separation problem (i.e., $x \in conv(S)$ or finding a violated valid inequality) as well. This somehow also shows a measure of the difficulty to find a complete description of $conv(S)$ and of the development of separation routines.

  Similarly to the previous approach, the drawback of performing a polyhedral study lies in the fact that the valid inequalities identified can be only applied to the problem under consideration (or to variations of it). However, some inequalities obtained for particular problems can be used or adapted to general ones. For example, valid inequalities devised for the *Knapsack Problem* (see, e.g., Martello and Toth [44] and Nemhauser and Wolsey [47]) and some further generalizations are included as general purpose cutting planes in several of the most important commercial software products, such as CPLEX [16].

Finally, and based on this discussion, it is possible to conclude that a cutting plane algorithm may not solve the problem to optimality, either because it is not able to find a violated cut or it exceeds a pre-established time limit for the execution. However, it can be used to obtain good lower bounds for the overall optimum value of the MILP problem. Furthermore, based on the final optimal solution of the LP relaxation, it is also possible to find a good feasible solution by means of a heuristic procedure.

2.2.2. Branch and Bound algorithms

The main idea behind B&B algorithms is to perform an intelligent enumeration of the feasible solutions, using certain information related to the problem to reduce the size of the enumeration tree. These algorithms can also be associated with the *Divide and Conquer* paradigm, which idea is to separate the search space into smaller sets in order to make the search easier.

This scheme is represented by an enumeration tree whose root node corresponds to the original problem and the *branches* are the result of the partition of the search space. Each node

of the tree has an associated subproblem which involves finding the optimum within a part of the solution space. In addition, dominance and feasibility are arguments that will allow to discard (*prune*) certain branches during the search process.

A possibility for pruning branches, usually called *bounding*, is to calculate at each node of the tree lower bounds of the optimum value restricted to the portion of the solution space under consideration. If the lower bound obtained is greater than the value of a known overall upper bound for the problem then it is not necessary to continue exploring that branch. The calculus of such bounds should achieve a tradeoff between their quality and the effort required for the computation. A weak bound may produce the unnecessary exploration of branches, but a procedure that computes tight bounds at high computational effort can be inefficient for the overall algorithm.

The calculation of lower and upper bounds can be performed in several ways. For the former, any relaxation of the current problem may produce a valid lower bound. For example, this value can be obtained by the LP relaxation, a *Lagrangean Relaxation* (see, for example, Beasley [10]), etc. The first one is the most natural and used approach in practice. As regards the upper bound, for example, the value of any feasible solution of the original MILP problem (known in advance or found during the process) can be considered.

The partition of the solution space, *branching*, involves the division of the feasible region of the current analyzed node into two or more smaller feasible regions. Each new region generates a new subproblem or child node, originated by the inclusion of a new constraint to the problem associated with the parent node. An essential requirement is that every feasible solution of the parent node belongs to, at least, one of these new nodes. These new subproblems are inserted into the list of unexplored nodes. The simplest branching rule is to consider an integer variable with a fractional value, d, in the current solution of the LP relaxation. Hence, the problem can be split into two new child nodes, imposing $\lfloor d \rfloor$ as an upper bound of the variable in one of them and $\lceil d \rceil$ as a lower bound in the other. This procedure is applied recursively to each generated node in the tree.

The pseudocode for the general scheme of a B&B algorithm is shown in Algorithm 2.2. It must be taken into account that, in this scheme, two important ingredients that may notoriously affect the behaviour are not specified: the *node selection strategy* and the process for generating the subproblems.

Concerning the node selection, among the most used strategies are *Depth First Search* (last node of the list), *Breadth First Search* (first node of the list) and *Best Bound* (the node with smaller optimum value of the LP relaxation). As regards the branching, it can be done as mentioned before by selecting a particular variable or by adding to the parent node inequalities to divide the region into smaller ones. However, the selection of the variable (or the inequalities) is not specified. This can be achieved in different ways, for example, *minimum infeasibility*, *maximum infeasibility*, *pseudo costs*, *strong branching*, etc, each of them producing different behaviours and results. Achterberg et al. [2] provide a very complete and updated review on different branching strategies.

There is no combination of these two factors that produces always better results for all problems. It is necessary to include criteria based in a combination of theory, common sense

and experimentation.

Algorithm 2.2 General B&B Algorithm

Input: An MILP formulation.

1. *Initialization*: Create a list L with the root node and the LP relaxation of the problem. Set $Z^{\text{ub}} = \infty$.
2. *Node Selection*: If L is empty, then stop and report Z^{ub}. Otherwise, select a node from L and remove it from the list.
3. *Bounding*: Solve the linear relaxation associated to the selected node. If it is infeasible, return to 2. Otherwise, let x^* be the optimal solution and Z^* the value of the objective function.
   - If x^* is feasible for the original problem, set $Z^{\text{ub}} = \min\{Z^{\text{ub}}, Z^*\}$. Return to 2
   - If $Z^* \geq Z^{\text{ub}}$, then in this branch there cannot be a feasible solution better than the current one. Return to 2.
4. *Branching*: Generate subproblems for the current node and add them to L. Return to 2.

2.2.3. Branch and Cut algorithms

At the beginning of the 80's, Crowder et al. [17] were very successful applying a mixed technique to solve pure binary problems. They considered a B&B algorithm but, before starting the Branching phase, they applied a cutting plane algorithm to the linear relaxation associated to root node of the tree. In this manner they improved the lower bound obtained by the LP relaxation and decreased the size of the explored tree.

In the middle of the decade appeared the first papers to extend this idea by applying a cutting phase to other nodes in the tree. Grötschel et al. [30] use this approach for the *Linear Ordering Problem* (LOP) and Padberg et al. [50] for the TSP. In the latter it was introduced the term *Branch and Cut*. The general scheme for this type of algorithms is shown in Algorithm 2.3.

The description of the algorithm leaves without specification several important items, besides the ones mentioned in Section 2.2.2 for the B&B. For example, how many iterations should perform the cutting plane algorithms, how many cuts should be added in each iteration, what to do with the cuts generated in different nodes of the tree, etc. All these represent design decisions which may affect the behaviour, effectiveness and computational time required by the algorithm. In practice, finding a tradeoff among the different strategies adopted is important and depends on the problem under study.

Generally speaking, we can point out two key issues based on the good results obtained with this technique:

- The use of cutting planes which, beside being generated at a particular node in the tree, are valid for the whole tree. This takes advantage of the effort spent on identifying valid inequalities.
- The use of specific valid inequalities for the problem, as the result of the study of its polyhedral structure.

In the last decades, these type of algorithms have proven to be very effective in solving a great variety of combinatorial optimization problems. The use of specific valid inequalities for the problem can be identified as a key factor for computational success, especially for hard problems.

Algorithm 2.3 General B&C Algorithm

Input: An MILP formulation.

1. *Initialization*: Create a list L with the root node and the LP relaxation of the problem. Set $Z^{\text{ub}} = \infty$.
2. *Node Selection*: If L is empty, then stop and report Z^{ub}. Otherwise, select a node from L and remove it from the list.
3. *Bounding*: Solve the linear relaxation associated to the selected node. If it is infeasible, return to 2. Otherwise, let x^* be the optimal solution and Z^* the value of the objective function.
   - If x^* is feasible for the original problem, set $Z^{\text{ub}} = \min\{Z^{\text{ub}}, Z^*\}$. Return to 2
   - If $Z^* \geq Z^{\text{ub}}$, then in this branch there cannot be a feasible solution better than the current one. Return to 2.
4. *Branching vs. Cutting*: Decide whether to seek for cutting planes or not. If the answer is no, go to 6.
5. *Separation*: Look for valid inequalities violated by x^*. If none is found, go to 6. Otherwise, add them to the formulation and go to 3.
6. *Branching*: Generate subproblems for the current node and add them to L. Return to 2.

2.2.4. Branch and Price algorithms

Branch and Price algorithms (B&P) are a generalization of B&B which were introduced by Savelsbergh [55]. They are designed specially for formulations that have a big number of variables (eventually, an exponential one) which cannot be handled explicitly. These models are very frequent in practice and can be considered due to different reasons. For instance, they could be the only approach to model some particular problem. It may also happen that models with fewer variables have weaker LP relaxations that affect the behaviour of the known algorithms.

Due to the size of the model, the resolution of the LP relaxations associated to each node of the tree can be very expensive in terms of computational times. The strategy in these cases is to solve the LP relaxation restricted to a (manageable) subset of variables. The solution obtained is feasible for the original relaxation, but not necessarily optimal. To check optimality it is imperative to search within the non-considered variables for a candidate to enter the basis. That is, let y^* be the optimal solution of the dual of the restricted problem, we have to determine whether there exists a column a with $c_a - y^* a < 0$, with c_a the cost of column a in the objective function. If a column satisfying this condition exists, column a is included into the restricted set of variables and the problem is re-optimized. Otherwise, the actual solution is optimal and the algorithm continues with the traditional scheme for B&B algorithms. This technique for solving the LP-relaxations is known as *column generation*.

As columns are not considered explicitly, the procedure used to check the optimality of an LP solution is a key factor for the success of this technique. In many situations, the columns

of the constraints' matrix can be described implicitly as characteristic vectors for subsets of a particular set. For example, given a set of boxes with different weights, the subsets of boxes having a total weight below a particular value. In these cases, it is possible to formulate the *column generation subproblem* as an MILP model that can be solved by means of a heuristic or an exact algorithm. In the example of the boxes, we obtain a *Knapsack Problem*: among all possible subsets of boxes that are below this particular value, select the one with minimum reduced cost. One of the first articles applying this technique is the one from Gilmore and Gomory [27] for the *cutting stock problem.*

Within the context of a B&P algorithm it is important to develop an efficient column generation algorithm that allows to solve LP relaxations in a reasonable amount of time. This depends essentially on the column generation subproblem structure and on the fact that the branching strategy does not affect this structure considerably, making it harder to solve.

2.2.5. Further comments

Even when it is not possible to find the optimal solution of an MILP problem, the procedures described before are really helpful in practice. The optimal value of the LP relaxation is a lower bound for the integer optimal value. Moreover, one of the main characteristics of these type of algorithms is to iteratively improve this lower bound and, in the case of B&B and B&C algorithms, new feasible solutions may arise during the enumeration, improving also the upper bound for the problem. In practice, sometimes it is not essential to find the optimal solution of the problem under consideration and having a feasible solution together with a measure for the error of an approximate solution represents a useful approach. Achieving good lower and upper bounds for the problem is a useful way for estimating and evaluating the quality of a feasible solution.

None of the methods presented above results in polynomial-time algorithms and it is quite difficult to estimate the overall time required to find an optimal solution of an MILP problem. The size, i.e. the number of variables and constraints, can give us a superficial idea of the difficulty but, in general, it is not enough. In many cases, the difficulty in solving an MILP problem depends not only on the number of variables and constraints, but also on the structure and characteristics of the formulation.

This last statement may lead us to have a pessimistic approach of the situation. However, this is not the case. During the last decades, many researchers have put a great effort in developing new ideas and producing better and more efficient implementations aiming to reduce the running times of the algorithms. This, in conjunction with numerous technological advances, makes possible to count today with solid and stable tools to solve MILP instances that were intractable a few years ago.

2.3. An example: The Asymmetric Travelling Salesman Problem

In this section we present the classical formulation for the Asymmetric Travelling Salesman Problem (ATSP) to illustrate some of the concepts mentioned in this chapter.

Recall the definition for the ATSP given Chapter 1. We consider a complete digraph $D = (V, A)$, where $V = \{0, 1, \ldots, n\}$ is the set of vertices and A the set of arcs linking them. Each arc $(i, j) \in A$, $i \neq j$, has associated a positive cost c_{ij}. The objective is to find a tour visiting all vertices in V exactly once with minimum total cost.

Figure 2.1 shows an example of $D = (V, A)$ for $n = 3$. It is a complete digraph, since for every pair of vertices i, j, $i \neq j$, there is an arc $(i, j) \in A$ linking them. In addition, the number associated with each $(i, j) \in A$ represents the associated cost c_{ij}.

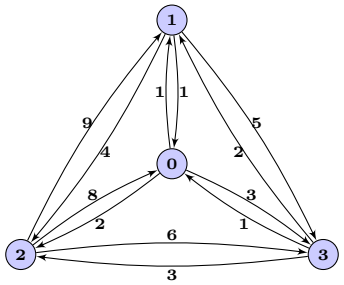


Figure 2.1.: Complete network ($n = 3$).

In order to formulate an ILP model for the ATSP, we begin with the definition of the decision variables. We consider binary variables x_{ij} for $(i, j) \in A$, where $x_{ij} = 1$ if and only if vertex j is visited immediately after vertex i in the tour. Then, the next step is to model the objective function that minimizes the total cost of the tour and the set of constraints that define a feasible solution for the problem using linear expressions. The classical model for the ATSP is defined by the ILP formulation (2.3) - (2.7).

$$\min \quad \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{2.3}$$

$$\text{s.t.} \quad \sum_{j\in V} x_{ij} = 1 \qquad i \in V \tag{2.4}$$

$$\sum_{i\in V} x_{ij} = 1 \qquad j \in V \tag{2.5}$$

$$\sum_{i\in S} \sum_{j\in S} x_{ij} \leq |S| - 1 \qquad S \subset V,\ |S| \geq 2 \tag{2.6}$$

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

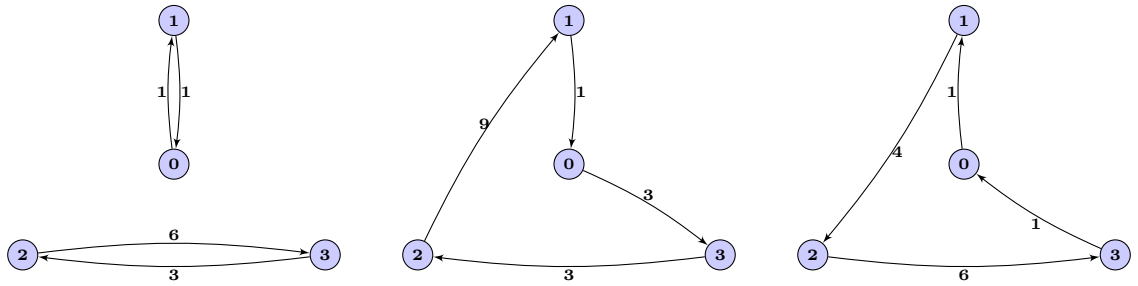
The objective function (2.3) minimizes the sum of the costs associated with the arcs involved in the tour. Equations (2.4) and (2.5) establish that the vehicle must arrive to and depart from each vertex exactly once, respectively. They are known as the *in-degree* and the *out-degree* constraints. However, imposing only these two conditions is not enough to obtain a feasible solution for the ATSP. In addition, the well-known *Subtour Elimination Constraints* (SEC)

have to be considered, which are expressed in constraints (2.6). Finally, in (2.7) we establish integrality conditions for the variables x_{ij}.

The SEC impose an exponential number of constraints, which clearly cannot be included explicitly even for small values of n. Therefore, these constraints are usually considered as cuts in a B&C algorithm, as explained in Section 2.2.3, and added on demand to the formulation within the B&C tree. It is important to remark that the separation problem for the SEC can be solved in polynomial time and in practice a separation procedure can be implemented efficiently (see, e.g., Padberg and Rinaldi [49]).

Figure 2.2 illustrates three different situations related with the ATSP example shown in Figure 2.1. In Figure 2.2(a) we can observe an infeasible solution for the ATSP known as *subtours*, which correspond to partial circuits. In general, solutions following this pattern are the result of the *Assignment Problem* defined by the objective function (2.3) and the constraints (2.4), (2.5), together with $x_{ij} \geq 0$ for $(i, j) \in A$. This solution can be forbidden by including the SEC induced, for example, by $S = \{2, 3\}$.

Figures 2.2(b) and 2.2(c) show two different feasible solutions for the ATSP. The former corresponds to the tour defined by the sequence of vertices $(0, 1, 2, 3)$ and has a total cost of 16. The latter represents the solution determined by $(0, 3, 2, 1)$ and the total cost is 12. In particular, it corresponds to the optimal solution for the instance described in Figure 2.1.



(a) Infeasible solution (subtours). Cost: 11.

(b) Feasible solution. Cost: 16.

(c) Optimal solution. Cost: 12.

Figure 2.2.: Examples related with the ATSP.

As mentioned in the introduction, the TSP (and the ATSP) has received a lot of attention during the last decades. The model presented in this section is only one of the possibilities for the ATSP. Alternative formulations with different variables, constraints, properties and characteristics are available in the related literature.

3. TDTSP: the position-dependent case

3.1. Introduction and literature review

The *Time-Dependent Travelling Salesman Problem* (TDTSP) is a generalization of the classical Travelling Salesman Problem (TSP) in which the cost of the travel between two cities depends not only on the distance between them, but also on the time of the day the arc is travelled.

This problem can be stated as follows: Consider a complete digraph $D = (V, A)$, with $V = \{0, 1, \ldots, n\}$ the set of vertices and A the set of directed arcs. Assume that there exists an $(n+1) \times (n+1)$ time-dependent cost matrix $C(t_i) = [c_{ij}(t_i)]$ that associates a cost $c_{ij}(t_i)$ to each arc (i, j) in A, where $c_{ij}(t_i)$ is a time-dependent cost function for the arc (i, j). Vertex 0 is a special vertex representing the depot. The TDTSP involves finding a minimum cost tour that visits each vertex exactly once, starting at and returning to vertex 0.

The TDTSP offers many interesting practical applications. One of the main advantages is that it allows us to model some real world situations that the classical TSP cannot deal with. For example, the travel time between any two cities may be different if the arc is taken early in the morning or at noon, or if it is known that traffic jams may occur during rush hours, etc. The TDTSP can model this kind of scenarios by using an appropriate cost function for each arc.

In its simplest version, TDTSP assumes that the travel time between any two cities is one time period, meaning that the travel time function depends on the distance between the cities and on the position of the arc in the tour. The time-dependent cost function for arc (i, j), $c_{ij}(t_i)$, can be expressed as a step function with one constant value for each time period $k = 1, \ldots, n$, named c_{ijk}. Additionally to the advantages mentioned above, this version of TDTSP can be used to model different scheduling and assignment problems.

Fox [24] proposes TDTSP formulations to solve the problem of scheduling production with time-dependent staff requirements. Picard and Queyranne [53] give three integer programming formulations -two of them are linear- for TDTSP to model a scheduling problem with time-dependent transition costs. They use these formulations to minimize the tardiness costs in one machine scheduling and they report instances with up to 20 vertices solved to optimality. Fox et al. [25] give a new formulation with $O(n^3)$ variables and only n constraints, but no computational results are reported. Gouveia and Voss [29] present two new formulations for the TDTSP derived from a quadratic assignment problem (QAP), and establish a relation among the previously mentioned formulations and the new ones in terms of the tightness of the linear relaxations.

Almost at the same time, Vander Wiel and Sahinidis [60] present a new formulation for the TDTSP which results to be the same as the one proposed by Gouveia and Voss [29]. The

only difference between them is that Vander Wiel and Sahinidis [60] prove that the integrality condition of some variables can be relaxed. As a result, their formulation has $O(n^3)$ variables, but only $O(n^2)$ are required to be binaries. The main idea of the model is to reformulate the TDTSP as the problem of finding the shortest constrained path in a directed multi-partite graph. They use a Benders-based heuristic to compute upper and lower bounds for their formulation of TDTSP. Computational experience shows that this procedure achieves very good bounds with minimal computational effort. In a follow up paper, Vander Wiel and Sahinidis [61] develop an exact algorithm for the TDTSP. Preliminary computational results show instances with at most 20 cities solved to optimality.

A very interesting approach is proposed by Bigras et al. [13]. The authors study the path formulation proposed in Picard and Queyranne [53] and evaluate the improvements in the LP relaxations of pricing path without 4 cycles. Cutting planes are applied at the root node, considering different families of valid inequalities of TSP as well as clique inequalities obtained by constructing the conflict graph using incompatibilities specially inferred for the TDTSP. They evaluate B&P algorithm on different instances and are able to solve randomly generated instances having up to 50 vertices as well as some instances from the TSPLIB. In a very recent paper, Abeledo et al. [1] provide good computational results using a Branch and Cut and Price based on the approach of Bigras et al. [13].

This simplified version of TDTSP also generalizes the well-known *Travelling Deliveryman Problem* (TDP). Given a vehicle depot and a number of customers with known travel times between each pair of them, the TDP involves finding a path starting at the depot and visiting every customer exactly once that minimizes the sum of the times required to reach every customer. In particular, this objective function can be caputred defining $c_{ijk} = (n-k+1)c_{ij}$, where c_{ij} denotes the travel time from vertex i to vertex j. Exact solution algorithms for the TDP are available in Lucena [40], Fischetti et al. [22], van Eijl [59] and Méndez-Díaz et al. [45].

In Picard and Queyranne [53], one of the models is a three-index integer linear programming formulation. Méndez-Díaz et al. [45] tested it for TDP instances and the results obtained are quite reasonable. In addition, Gouveia and Voss [29] prove that this formulation is equivalent to the one presented in Vander Wiel and Sahinidis [60] in terms of the linear relaxation, and that this value is an upper bound for the rest of the formulations considered.

These results suggest that both models look promissory to be used in a B&C algorithm. Picard and Queyranne [53] use a B&B algorithm. Vander Wiel and Sahinidis [61] developed a B&C algorithm to solve the master problem of the Benders decomposition, but they only use general purpose cuts for a restricted set of inequalities. They suggest that, as future research, it would be interesting to study the polyhedron of the TDTSP. The aim of this chapter goes in that direction. We consider the models presented in Picard and Queyranne [53] and Vander Wiel and Sahinidis [60]. Since both models are a linearization of QAP, it results that their polytopes are strongly related. We derive several families of valid inequalities and facets and evaluate them in a B&C algorithm.

The chapter is organized as follows. In Section 3.2 we introduce the models from Picard and Queyranne [53] and Vander Wiel and Sahinidis [60] and establish a strong relation between them. In Section 3.3 we introduce a new family of valid inequalities based on the well known cycle inequalities for the ATSP. We also analyze some polyhedral properties for this family, which is

used to derive five families of facets, and present another four families of valid inequalities. In Section 3.4 we give the outline of the B&C algorithm, which is tested and compared with other exact algorithms in Section 3.5. Finally, in Section 3.6, we present some conclusions and future research.

3.2. Models

3.2.1. Picard and Queyranne

This model is proposed in Picard and Queyranne [53]. It uses a set of binary decision variables y_{ijk}, where $y_{ijk} = 1$ if city j is visited in time period $k+1$ after city i was visited in time period k. By forcing vertex 0 to be the depot, we can remove from the formulation the variables $y_{ij0}\ \forall i \geq 1,\ y_{ijn}\ \forall j \geq 1,\ y_{i0k}\ \forall k \leq n-1,\ y_{0jk}\ \forall k \geq 1$, given that they always take value zero. The formulation is shown below.

$$\min \sum_{j=1}^{n} c_{0j0}y_{0j0} + \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}\sum_{k=1}^{n} c_{ijk}y_{ijk} + \sum_{j=1}^{n} c_{j0n}y_{j0n} \qquad \text{(PQ)}$$

$$\text{s.t.} \sum_{j=1}^{n} y_{0j0} = 1 \qquad (3.1)$$

$$\sum_{i=1,i\neq j} y_{ji1} = y_{0j0} \qquad j = 1,\ldots,n \qquad (3.2)$$

$$\sum_{i=1,i\neq j}^{n} y_{ijk} = \sum_{i=1,i\neq j}^{n} y_{jik+1} \qquad j = 1,\ldots,n;\ k = 1,\ldots,n-2 \qquad (3.3)$$

$$\sum_{i=1,i\neq j}^{n} y_{ijn-1} = y_{j0n} \qquad j = 1,\ldots,n \qquad (3.4)$$

$$y_{0j0} + \sum_{i=1,i\neq j}^{n}\sum_{k=1}^{n-1} y_{ijk} = 1 \qquad j = 1,\ldots,n \qquad (3.5)$$

$$y_{ijk} \in \{0,1\} \qquad i = 1,\ldots,n;\ j = 1,\ldots,n$$
$$k = 1,\ldots,n-1;\ i \neq j$$

$$y_{j0n}, y_{0j0} \in \{0,1\} \qquad j = 1,\ldots,n,$$

The objective function minimizes the total travel cost of the tour. Equation (3.1) forces the vehicle to depart from the depot in time period 0. Equations (3.2), (3.3) and (3.4) ensure that the selected transitions are travelled in consecutive time periods. Constraints (3.5) establish that the vehicle must arrive to each vertex $j \in V\backslash\{0\}$ in exactly one time period. Finally, integrallity conditions on variables are imposed.

Feasible solutions satisfying constraints (3.1) - (3.5) correspond to tours with transitions travelled in consecutive time periods, starting from the depot in time period 0. It is important to remark that this formulation does not allow subtours.

3.2.2. Vander Wiel and Sahinidis

We also consider the TDTSP formulation of Vander Wiel and Sahinidis [60] (VW). The model is a linearization of the QAP presented in Picard and Queyranne [53], and it can be seen as the problem of finding the shortest constrained path in a directed multi-partite graph. We show this formulation in a slightly different way than the one in Vander Wiel and Sahinidis [60] because we force the vertex 0 to be the depot.

The QAP formulation uses a set of binary decision variables x_{ik}, where $x_{ik} = 1$ if city i is assigned to time period k, and $x_{ik} = 0$ otherwise. This model is quadratic because of the presence of the product between x_{ik-1} and x_{ik} in the objective function for each possible combination of (i, j) and k.

In Vander Wiel and Sahinidis [60], variables x_{ik} are referred as the *assignment variables*. To linearize the objective function of the QAP, they define the *transition variables*, y_{ijk}, which have the same meaning as the ones defined in the previous section. Moreover, they prove (see Proposition 1 of Vander Wiel and Sahinidis [60] for a detailed proof) that $y_{ijk} = 1$ if and only if $x_{ik-1}x_{jk} = 1$, even when variables y_{ijk} are considered as positive continuous variables. The advantage of this linearization is that it only introduces continuous variables to the original formulation of the QAP. See Vander Wiel and Sahinidis [60, Section 1] for a detailed explanation and examples.

$$\min \sum_{j=1}^{n} c_{0j0}y_{0j0} + \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}\sum_{k=1}^{n} c_{ijk}y_{ijk} + \sum_{j=1}^{n} c_{j0n}y_{j0n} \qquad \text{(VW)}$$

$$\text{s.t.} \sum_{k=1}^{n} x_{ik} = 1 \qquad i = 1, \ldots, n \tag{3.6}$$

$$\sum_{i=1}^{n} x_{ik} = 1 \qquad k = 1, \ldots, n \tag{3.7}$$

$$y_{0j0} = x_{j1} \qquad j = 1, \ldots, n \tag{3.8}$$

$$\sum_{\substack{i=1 \\ i\neq j}}^{n} y_{ijk} = x_{jk+1} \qquad j = 1, \ldots, n; k = 1, \ldots, n-1 \tag{3.9}$$

$$y_{i0n} = x_{in} \qquad i = 1, \ldots, n \tag{3.10}$$

$$\sum_{\substack{j=1 \\ j\neq i}}^{n} y_{ijk} = x_{ik} \qquad i = 1, \ldots, n; k = 1, \ldots, n-1 \tag{3.11}$$

$$\sum_{j=1}^{n} y_{0j0} = 1 \tag{3.12}$$

$$\sum_{j=1}^{n} y_{j0n} = 1 \tag{3.13}$$

$$y_{ijk} \geq 0, x_{ik} \in \{0,1\} \tag{3.14}$$

Similarly to model (PQ), the objective function minimizes the total travel cost of the tour. Equations (3.6) and (3.7) establish that each vertex must be assigned to exactly one time period and that each time period can be assigned to exactly one vertex, respectively. Equations (3.8) and (3.9) ensure that the arrival to a vertex is done according to the time period in which it is assigned. An analogous relation is established in equations (3.10) and (3.11) for the departure of each vertex. Equations (3.12) and (3.13) establish that the vehicle must depart from the depot in time period 0 and arrive in time period n, respectively. Finally, integrality conditions on variables are imposed.

By forcing vertex 0 to be the depot, we can remove from the formulation the variables $x_{i0}\ \forall i \geq 1,\ x_{0k}\ \forall k \geq 1,\ y_{ij0}\ \forall i \geq 1,\ y_{ijn}\ \forall j \geq 1,\ y_{i0k}\ \forall k \leq n-1,\ y_{0jk}\ \forall k \geq 1$, given that they always take value zero.

3.2.3. Relation between both models

As we mentioned in the introduction, (PQ) and (VW) are strongly related. It is easy to see that (PQ) is the projection of (VW) onto variables y_{ijk}, and Gouveia and Voss [29] prove that these formulations are equivalent in terms of the linear relaxation. Formulation (VW) expresses each assignment variable, x_{ik}, in terms of the transition variables y_{ijk}. Considering the results shown in Balas and Oosten [9], we deduce that there is a 1-1 correspondence between the faces of (PQ) and the faces of (VW). Moreover, if P_{PQ} and P_{VW} are the polytopes associated with models PQ and VW, respectively, we can also state that $dim(P_{PQ}) = dim(P_{VW})$. From Müller [46] we also know the dimension of P_{PQ}.

Theorem 3.2.1 (Müller [46]). *The dimension of P_{PQ} is $n(n-1)(n-2)$ for $n \geq 5$.*

Then, if an inequality is valid for P_{PQ}, it is also valid for P_{VW} since variables y_{ijk} are considered in formulation (VW). Conversely, if an inequality is valid for P_{VW}, the projected version of the inequality is valid for P_{VW} since variables x_{ik} can be replaced by their expression in terms of variables y_{ijk}. Overloading notation slightly, we will refer to both P_{PQ} and P_{VW} as P^n_{TD}.

3.3. Polyhedral investigations

In this section we focus on deriving families of valid inequalities, aiming to include them in the cutting phase of a B&C algorithm in order to improve the lower bounds produced by the LP relaxation. Based on the results presented in Section 3.2.3, these inequalities are valid for both P_{PQ} and P_{VW}, although most of the inequalities consider variables x_{ik}.

In Section 3.3.1 we propose a new family of valid inequalities exploting the idea of *time dependent cycles*. For this family, we prove that they are facet defining on a polytope obtained

by restricting P_{TD}^n and, from this characterization, we provide in Section 3.3.2 a procedure to obtain facets of P_{TD}^n by means of maximum sequential lifting. Finally, in Section 3.3.3 we derive further valid inequalities that express certain properties regarding time periods.

3.3.1. Time-dependent cycle inequalities

In this section we introduce a new family of valid inequalities based on the idea of the well known cycle inequalities for the asymmetric TSP. The main characteristic is that they include the time dependency of the transitions between vertices. Time dependent cycles are not allowed by the formulations considered in this chapter. However, this family can be used to cut fractional solutions by including them in a B&C algorithm. Indeed, in the next section we will strengthen these inequalities by applying a lifting procedure.

Figure 3.1 illustrates the idea behind time dependent cycles. The value associated with each arc stands for the time period in which it is travelled. Let $C =< v_1, v_2, v_3, v_4, v_1 >$ be a simple cycle with transitions between consecutive vertices travelled in time periods $k, k+1, k+2, k+3$. If we consider $n \geq 4$, then this time dependent cycle cannot be part of a feasible solution.

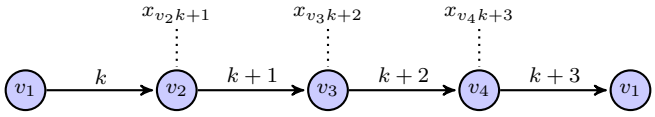


Figure 3.1.: TDCI for $l = 4$.

To forbid this situation, we can consider the following inequality

$$y_{v_1v_2k} + y_{v_2v_3k+1} + y_{v_3v_4k+2} + y_{v_4v_1k+3} \leq 3.$$

However, as we show next in Proposition 3.3.1, we can replace the rhs by some of the vertex variables x_{ik} from formulation (VW) involved in the cycle, as shown below. This new expression dominates the previous one, since by definition variables x_{ik} are upper bounded by 1.

$$y_{v_1v_2k} + y_{v_2v_3k+1} + y_{v_3v_4k+2} + y_{v_4v_1k+3} \leq x_{v_2k+1} + x_{v_3k+2} + x_{v_4k+3}$$

We will refer to this family of inequalities as the Time-Dependent Cycle Inequalities (TDCI). For the sake of notation, we express them in terms of both variables x_{ik} and y_{ijk}.

Proposition 3.3.1 (TDCI)**.** *Let $C =< v_1, v_2, \ldots, v_l, v_{l+1} = v_1 >$, $l \leq n$, be a simple cycle with transitions between consecutive vertices travelled in time intervals $k, k+1, \ldots, k+l-1$, $k+l \leq n$. Then, inequality*

$$\sum_{i=1}^{l} y_{v_iv_{i+1}k+i-1} \leq \sum_{i=1}^{l-1} x_{v_{i+1}k+i} \tag{3.15}$$

is valid for P_{TD}^n.

Proof. From Equation (3.9) we know that $y_{v_i v_{i+1} k+i-1} \leq x_{v_{i+1} k+i}$ for $i = 1, \ldots, l-1$ are valid. Adding these inequalities we get that

$$\sum_{i=1}^{l-1} y_{v_i v_{i+1} k+i-1} \leq \sum_{i=1}^{l-1} x_{v_{i+1} k+i} \tag{3.16}$$

is a valid inequality. Using a similar argument, but now considering Equation (3.11), inequalities $y_{v_i v_{i+1} k+i-1} \leq x_{v_i k+i-1}$ for $i = 2, \ldots, l$ are valid. Adding them and rewriting the right hand side we get that

$$\sum_{i=2}^{l} y_{v_i v_{i+1} k+i-1} \leq \sum_{i=1}^{l-1} x_{v_{i+1} k+i} \tag{3.17}$$

is also a valid inequality.

Considering inequalities (3.16) and (3.17), we can rewrite them as

$$\begin{aligned} y_{v_1 v_2 k} + \sum_{i=2}^{l-1} y_{v_i v_{i+1} k+i-1} &\leq \sum_{i=1}^{l-1} x_{v_{i+1} k+i} \\ \sum_{i=2}^{l-1} y_{v_i v_{i+1} k+i-1} + y_{v_l v_1 k+l-1} &\leq \sum_{i=1}^{l-1} x_{v_{i+1} k+i} \end{aligned}$$

Given that $y_{v_1 v_2 k}$ and $y_{v_l v_1 k+l-1}$ cannot both take value one in a feasible solution, the inequality

$$\sum_{i=1}^{l} y_{v_i v_{i+1} k+i-1} \leq \sum_{i=1}^{l-1} x_{v_{i+1} k+i}$$

is valid for P_{TD}^n. □

The TDCI do not define facets for P_{TD}^n. However, similarly to the cycle inequalities for the ATSP, TDCI define facets of a projection of P_{TD}^n over some specific variables. Let $C =< v_1, v_2, \ldots, v_l, v_{l+1} = v_1 >$, $l \leq n$ and k as defined in Proposition 3.3.1. We define the following sets of variables

- $F_1 = \{y_{v_l v_j k+j-2} = 0 \ : \ j = 2, \ldots, l-1\}$
- $F_2 = \{y_{v_l v_j k+l-1} = 0 \ : \ j = 2, \ldots, l-1\}$
- $F_3 = \{y_{v_i v_j k+j-2} = 0 \ : i = 3, \ldots, l-1, \ j = 2, \ldots, i-1\}$
- $F_4 = \{y_{v_i v_j k+i-1} = 0 \ : i = 2, \ldots, l-1, \ j = 1, \ldots, i-1\}$
- $F_5 = \{y_{v_1 v_j k+j-2} = 0 \ : \ j = 3, \ldots, l\}$,

and

$$P_{TD}^n(C,k) = P_{TD}^n \cap \bigcap_{i=1}^{5} F_i.$$

Variables characterizing subspaces $F_1, \ldots, F_5$ represent chords (v_i, v_j) of C, $v_i, v_j \in C$, travelled in valid time periods of departure from v_i or of arrival at v_j with respect to their corresponding positions in the cycle.

We begin characterizing the dimension of the restricted polytope $P^n_{TD}(C,k)$ in Theorem 3.3.2. Then, based on this result, we prove that the TDCI (3.15) are indeed facet defining of $P^n_{TD}(C,k)$ in Theorem 3.3.3. The importance of this result is twofold. First, it implies that the TDCI cannot be strengthened when considering $P^n_{TD}(C,k)$. Secondly, based on the definition of $F_1, \ldots, F_5$ we can devise which variables may be considered to obtain strengthened versions of the TDCI for the general polytope P^n_{TD}. This topic is addressed in detail in Section 3.3.2.

We now establish the following results.

Theorem 3.3.2. *Let $C =< v_1, v_2, \ldots, v_l, v_{l+1} = v_1 >$, $4 \leq l \leq n$ and k the set of vertices and the starting time period associated with a TDCI as defined in Proposition 3.3.1. The dimension of $P^n_{TD}(C,k)$ is $n(n-1)(n-2) - (l+1)(l-2)$ for $n \geq 5$.*

Proof. For the sake of simplicity, w.l.o.g. we assume that vertices in the cycle C are labelled as $< 1, 2, \ldots, l, 1 >$, that the time period k is 1 and $n \geq 7$. In addition, we will refer indistinctly to a tour and to its incidence vector.

Similarly to the idea used in Fischetti [21], the proof is carried out by induction on n and the sketch is the following. In the base case we prove the result on the dimension of $P^n_{TD}(C,k)$, with $n = l+1$, by showing $n(n-1)(n-2) - n(n-3) + 1$ affinely independent tours. In the inductive step, we assume to have the $n(n-1)(n-2) - (l+1)(l-2) + 1$ affinely independent tours for $P^n_{TD}(C,k)$, and based on them we prove the result on the dimension of $P^{n+1}_{TD}(C,k)$. Specifically, we extend these tours by adding vertex $n+1$ in position $n+1$, which by inductive hypothesis, are still affinely independent. We also construct the remaining $3n(n-1)$ tours, and finally we prove that the $(n+1)n(n-1) - (l+1)(l-2) + 1$ tours are affinely independent.

In order to check the affine independence, we show that the tours are linearly independent. Basically, we show an ordering - i.e., the order in which they are presented - in such a way that, considering all the remaining tours, the tour is the only one having a particular variable with value one. In other words, consider a matrix where each row is the incidence vector of one of the proposed tours. Then, the ordering we provide is a permutation of rows (tours) and columns (variables) such that the resulting matrix is upper triangular with non-zero diagonal entries.

The tours are presented in tables that assign vertices to time periods. In each table, we specify only the important positions, i.e., those that are essential for the definition of the tour. For the non-specified positions, the remaining vertices are assigned in increasing order. The last column of each table states which is the variable considered for the linear independence part of the proof. This is noted as a tuple (i,j,k), meaning that variable $y_{ijk} = 1$ for that tour.

Basis: The tours are divided into three different groups, according to the similarities in their particular structure. The only exception is shown below, and corresponds to the tour assigning vertex i to position i, for $i = 1, \ldots, n$.

1	2	...	$n-1$	n	Variable
1	2	...	$n-1$	n	$(n-1, n, n-1)$

We now present the definition of the tours within each group.

Group 1: Tours within this group share the characteristic that vertex n is not assigned to position n. For the linear-independence part, the variables to be used are of the form y_{int}, y_{nit} or y_{ijn}, for some $1 \leq t \leq n-1$. These variables have always value zero in the tours considered in groups 2 and 3, given that the latter have vertex n in position n.

1	...	$j-1$	j	$j+1$	...	$n-1$	n		Variable
n		i	j	$j+1$	...	$n-1$	1	$j=5,\ldots,n-1$, $i=2,\ldots,j-2$	$(i,j,j-1)$

1	2	...	s	$s+1$	$s+2$	...	n		Variable
			$n-1$	n	1			$s=1,\ldots,n-4$	$(n-1,n,s)$
n	$n-1$								$(n,n-1,1)$

1	...	s	$s+1$	$s+2$	...	$n-2$	$n-1$	n		Variable
		n	j						$s=2,\ldots,n-3$, $j=1,\ldots,n-2$, $j\neq s+1$	(n,j,s)
		n	$s+1$					1	$s=2,\ldots,n-3$	$(n,s+1,s)$
		j	n	$n-1$					$s=1,\ldots,n-4$, $j=1,\ldots,n-2$	(j,n,s)
							n	j	$j=2,\ldots,n-2$	$(n,j,n-1)$
						j	n	$n-1$	$j=1,\ldots,n-2$, $j\neq 2$	$(j,n,n-2)$
						$n-1$	n	1		$(n-1,n,n-2)$
						2	n	1		$(n,1,n-1)$
						2	n	$n-1$		$(2,n,n-1)$

1	...	$n-3$	$n-2$	$n-1$	n		Variable
		$n-1$	n	i	j	$i=1,\ldots,n-2$, $j=1,\ldots,n-2$, $(i,j)\neq(1,n-2)$	$(i,j,n-1)$
		$n-1$	n	1	$n-2$		$(n-1,n,n-3)$
		2	n	1	$n-2$		$(1,n-2,n-1)$
			n	j	$n-1$	$j=1,\ldots,n-3$	$(n,j,n-2)$
		j	n	$n-2$	$n-1$	$j=1,n-3$	$(j,n,n-3)$
		$n-4$	n	$n-2$	$n-1$		$(n,n-2,n-2)$
		j	n	$n-1$	1	$j=2,\ldots,n-2$, $j\neq n-3$	$(j,n,n-3)$

1	2	...				$n-1$	n		Variable
n	$j+2$	...	$n-2$	1	...	j	$n-1$	$j=1,\ldots,n-4$	$(j,n-1,n-1)$

1	2	...	$n-1$	n		Variable
n	$n-2$					$(n,n-2,1)$
n			$n-3$	$n-1$		$(n-3,n-1,n-1)$
n	j				$j=1,\ldots,n-3$, $j\neq 2,3$	$(n,j,1)$
n	2			1		$(n,2,1)$
n	3		$n-2$	$n-1$		$(n-2,n-1,n-1)$
n	3		$n-1$	1		$(n,3,1)$

Group 2: Tours within this group have the characteristic that the last $n-k-1$ vertices visited are $k+1, k+2, \ldots, n-2, n$, for $k = 5, \ldots, n-2$. The variables used for the linear independence are of the form y_{ikt}, y_{kit} or y_{ijt}, for some $1 \leq t \leq n-2$. These variables are zero in the tours

considered in the next group for these values of k. For the definition of the tours that follows, k ranges from 5 to $n-2$.

1	2	...	k	$k+1$	$k+2$	...	$n-1$	n		Variable
k	$k-1$		j	$n-1$	$k+1$		$n-2$	n	$j=1,\ldots,k-2$, $k\neq n-2$	$(j,n-1,k)$
k	1		$k-1$	$n-1$	$k+1$		$n-2$	n	$k\neq n-2$	$(k-1,n-1,k)$
			k	$n-1$	$k+1$		$n-2$	n	$k\neq n-2$	$(k,n-1,k)$

1	2	...	s	$s+1$	$s+2$	...	$k+2$	...	$n-1$	n		Variable
			$n-1$	k			$k+1$		$n-2$	n	$s=1,\ldots,k-3$	$(n-1,k,s)$
			k	j	$n-1$		$k+1$		$n-2$	n	$s=2,\ldots,k-2$, $j=1,\ldots,k-1$, $j\neq s+1$	(k,j,s)
			j	k	$n-1$		$k+1$		$n-2$	n	$s=1,\ldots,k-3$, $j=1,\ldots,k-1$	(j,k,s)

1	2	...	$k-2$	$k-1$	k	$k+1$	$k+2$	...	$n-1$	n		Variable
			j	k	$n-1$		$k+1$		$n-2$	n	$j=1,\ldots,k-1$ $j\neq k-3$	$(j,k,k-2)$
k	$k-1$				$n-1$	j	$k+1$		$n-2$	n	$j=1,\ldots,k-2$	$(n-1,j,k)$

1	2	3	...	$k+1$	$k+2$	...	$n-1$	n		Variable
k	j	$n-1$			$k+1$		$n-2$	n	$j=3,\ldots,k-1$	$(k,j,1)$

1	...	$k-2$	$k-1$	k	$k+1$	$k+2$	...	$n-1$	n		Variable
			k	i	j	$k+1$		$n-2$	n	$i,j=1,\ldots,k-2$, $i\neq j$	(i,j,k)
		$n-1$	k	j		$k+1$		$n-2$	n	$j=1,\ldots,k-1$, $j\neq k-2$	$(k,j,k-1)$
		$n-1$	k	$k-2$		$k+1$		$n-2$	n		$(n-1,k,k-2)$
$n-1$			k	$k-2$		$k+1$		$n-2$	n		$(k,k-2,k-1)$
		$k-3$	k	$n-1$		$k+1$		$n-2$	n		$(k-3,k,k-2)$

1	2	3	...	k	$k+1$	$k+2$	...	$n-1$	n		Variable
k				$n-1$	$k-1$	$k+1$		$n-2$	n		$(n-1,k-1,k)$
k	1	$n-1$				$k+1$		$n-2$	n		$(k,1,1)$
k	$n-1$			j	$k-1$	$k+1$		$n-2$	n	$j=1,\ldots,k-3$	$(j,k-1,k)$
k	$n-1$			$k-1$	j	$k+1$		$n-2$	n	$j=1,\ldots,k-2$	$(k-1,j,k)$
k	$n-1$					$k+1$		$n-2$	n		$(k,n-1,1)$

Group 3: This group considers tours such that vertices $5,6,\ldots,n-2,n$ are visited in periods $6,7,\ldots,n$, respectively.

1	2	3	4	5	6	...	$n-1$	n	Variable
3	1	2	4	$n-1$	5	...	$n-2$	n	$(2,4,3)^*$
4	3	2	1	$n-1$	5	...	$n-2$	n	$(1,n-1,4)$
4	1	2	3	$n-1$	5	...	$n-2$	n	$(3,n-1,4)$
1	2	3	4	$n-1$	5	...	$n-2$	n	$(4,n-1,4)$
4	3	2	$n-1$	1	5	...	$n-2$	n	$(n-1,1,4)$
$n-1$	4	1	2	3	5	...	$n-2$	n	$(n-1,4,1)$
1	4	2	$n-1$	3	5	...	$n-2$	n	$(4,2,2)$
1	2	4	$n-1$	3	5	...	$n-2$	n	$(2,4,2)$
1	$n-1$	4	3	2	5	...	$n-2$	n	$(4,3,3)$
4	$n-1$	1	3	2	5	...	$n-2$	n	$(3,2,4)$
3	4	$n-1$	1	2	5	...	$n-2$	n	$(3,4,1)$
1	2	3	$n-1$	4	5	...	$n-2$	n	$(3,n-1,3)$
2	4	1	$n-1$	3	5	...	$n-2$	n	$(4,1,2)$
1	4	$n-1$	2	3	5	...	$n-2$	n	$(1,4,1)$
1	2	$n-1$	3	4	5	...	$n-2$	n	$(1,2,1)$
2	$n-1$	1	3	4	5	...	$n-2$	n	$(1,3,3)$
2	$n-1$	4	1	3	5	...	$n-2$	n	$(2,n-1,1)$
3	$n-1$	4	1	2	5	...	$n-2$	n	$(4,1,3)$
4	3	$n-1$	1	2	5	...	$n-2$	n	$(1,2,4)$
4	3	1	2	$n-1$	5	...	$n-2$	n	$(2,n-1,4)$
4	3	1	$n-1$	2	5	...	$n-2$	n	$(4,3,1)$
1	3	4	$n-1$	2	5	...	$n-2$	n	$(n-1,2,4)$
$n-1$	3	4	2	1	5	...	$n-2$	n	$(3,4,2)$
3	$n-1$	4	2	1	5	...	$n-2$	n	$(2,1,4)$
2	1	4	$n-1$	3	5	...	$n-2$	n	$(4,n-1,3)$
4	1	2	$n-1$	3	5	...	$n-2$	n	$(n-1,3,4)$
2	3	1	$n-1$	4	5	...	$n-2$	n	$(1,n-1,3)$
2	3	$n-1$	1	4	5	...	$n-2$	n	$(2,3,1)$
1	3	$n-1$	2	4	5	...	$n-2$	n	$(3,n-1,2)$
1	3	2	$n-1$	4	5	...	$n-2$	n	$(1,3,1)$
$n-1$	3	2	1	4	5	...	$n-2$	n	$(3,2,2)$
$n-1$	3	1	2	4	5	...	$n-2$	n	$(n-1,3,1)$
1	$n-1$	4	2	3	5	...	$n-2$	n	$(n-1,4,2)$
3	1	2	$n-1$	4	5	...	$n-2$	n	$(2,n-1,3)$
3	$n-1$	2	1	4	5	...	$n-2$	n	$(1,4,4)$
4	$n-1$	2	1	3	5	...	$n-2$	n	$(2,1,3)$
2	4	$n-1$	1	3	5	...	$n-2$	n	$(1,3,4)$
2	1	$n-1$	3	4	5	...	$n-2$	n	$(2,1,1)$
1	$n-1$	2	3	4	5	...	$n-2$	n	$(1,n-1,1)$
3	1	$n-1$	2	4	5	...	$n-2$	n	$(3,1,1)$
$n-1$	1	2	3	4	5	...	$n-2$	n	$(3,4,4)$
4	1	$n-1$	2	3	5	...	$n-2$	n	$(4,1,1)$
$n-1$	1	4	2	3	5	...	$n-2$	n	$(n-1,1,1)$
3	$n-1$	1	2	4	5	...	$n-2$	n	$(2,4,4)$
4	$n-1$	1	2	3	5	...	$n-2$	n	$(n-1,1,2)$
4	$n-1$	2	3	1	5	...	$n-2$	n	$(4,n-1,1)$
2	4	$n-1$	3	1	5	...	$n-2$	n	$(3,1,4)$

Inductive Step: As we mentioned before, we first extend the affinely independent tours for $P^n_{TD}(C,k)$ by adding vertex $n+1$ in position $n+1$. In order to complete the proof of dimension for $P^{n+1}_{TD}(C,k)$, we need to provide another $3n(n-1)$ affinely independent tours.

The main idea behind the construction of these tours is to place vertex $n+1$ in different periods. In order to prove the affine independence of the whole set, and considering the structure of the tours used for $P^n_{TD}(C,k)$, vertex $n+1$ is never assigned to position $n+1$. In addition, the variable considered for the linear independence part always involves either the vertex $n+1$ or the transition from period n to period $n+1$. Since these arguments are not used for the extended

tours arising from $P^n_{TD}(C,k)$, by inductive hypothesis, they are still linearly independent and therefore the whole set of tours is affinely independent.

As well as in the base case, tours are divided into seven different groups according to the similarities in their structure. For groups 1 to 6, the technique used to prove the affine independence is the same as in the base case: identifying for each tour a unique variable set to one, regarding the remaining tours in the ordering. However, in Group 7, there is a small set of tours for which this idea cannot be applied, but they are proved to be linearly independent as well.

Group 1: Vertex $n+1$ is visited in periods between 2 and $n-3$. In particular, when it is visited in period 2, the vertex assigned to period 1 is different from vertices 1 and n.

1	...	k	$k+1$	$k+2$	...	n	$n+1$		Variable
		j	$n+1$	1		$n-1$	n	$k=1,\dots,n-4$, $j=2\dots n-1$, $k\neq j \vee j>l$	$(j,n+1,k)$

1	...	j	$j+1$	$j+2$	...	$n-1$	n	$n+1$		Variable
1	...	j	$n+1$	$j+1$	...	$n-2$	$n-1$	n	$j=2,\dots,n-4$, $j\leq l$	$(j,n+1,j)$

1	...	k	$k+1$	$k+2$	...	n	$n+1$		Variable
		n	$n+1$	1	...	$n-2$	$n-1$	$k=2,\dots,n-4$	$(n+1,1,k+1)$

1	...	$j-2$	$j-1$	j	...	l	$l+1$	$l+2$	...	$n+1$		Variable
	...	n	$n+1$	j	...	l	1	2	...		$j=4,\dots,n-2$ $j\leq l$	$(n+1,j,j-1)$

1	...	k	$k+1$	$k+2$	...	n	$n+1$		Variable
		n	$n+1$	2		$n-2$	$n-1$	$k=2,\dots,n-4$	$(n,n+1,k)$

1	...	k	$k+1$	$k+2$	...	n	$n+1$		Variable
		1	$n+1$	j		$n-1$	n	$k=2,\dots,n-4$, $j=2\dots n$, $k\neq j-2 \vee j<4 \vee j>l$	$(n+1,j,k+1)$

Group 2: Vertex $n+1$ is visited in period n.

1	...	$n-2$	$n-1$	n	$n+1$		Variable
		n	$j+1$	$n+1$	j	$j=1,\dots,n-2$, $j+1\neq n-1 \vee l<n-1$	$(j+1,n+1,n-1)$
			$n-1$	$n+1$	n	$l\geq n-1$	$(n-1,n+1,n-1)$

1	...	$j-1$	...	$n-2$	$n-1$	n	$n+1$		Variable
		1			n	$n+1$	j	$j=2,\dots,n-2$	$(n+1,j,n)$
					n	$n+1$	1		$(n+1,1,n)$
				$n-2$	n	$n+1$	$n-1$		$(n,n+1,n-1)$

1	2	3	...	$n-3$	$n-2$	$n-1$	n	$n+1$	Variable
			...			1	$n+1$	n	$(n+1,n,n)$
			...		n	1	$n+1$	$n-1$	$(n+1,n-1,n)$

Group 3: Vertex $n+1$ is visited in period $n-2$.

1	2	3	...	$n-3$	$n-2$	$n-1$	n	$n+1$		Variable
1	4	5	...	n	$n+1$	2	3	$n-1$		$(n+1,2,n-2)$
2	3	4	...	n	$n+1$	1	$n-2$	$n-1$		$(n,n+1,n-3)$
2	3	4	...	1	$n+1$	n	$n-2$	$n-1$		$(1,n+1,n-3)$
1	3	4	...	2	$n+1$	n	$n-1$	$n-2$		$(n+1,n,n-2)$
			...	j	$n+1$	1	$n-1$	n	$j=2\dots n-1$, $j\neq n-3 \vee j>l$	$(j,n+1,n-3)$
1	2	3	...	$n-3$	$n+1$	$n-2$	$n-1$	n	$n-3\leq l$	$(n-3,n+1,n-3)$
2	3	4	...	$n-1$	$n+1$	j	1	n	$j=3\dots n-1$	$(n+1,j,n-2)$

Group 4: Vertex $n+1$ is visited in period $n-1$.

1	...	$n-3$	$n-2$	$n-1$	n	$n+1$		Variable
	...		j	$n+1$	1	n	$j=3\dots n-1$, $j\neq n-2 \vee j>l$	$(j,n+1,n-2)$
1	...	$n-3$	$n-2$	$n+1$	$n-1$	n	$n-2\leq l$	$(n-2,n+1,n-2)$
	...		1	$n+1$	n	j	$j=2\dots n-1$	(n,j,n)
	...		2	$n+1$	n	1		$(n+1,n,n-1)$
	...		n	$n+1$	j	$n-1$	$j=1\dots n-4,\ j\neq 2,3$	$(j,n-1,n)$
	...		n	$n+1$	2	1		$(2,1,n)$
	...		2	$n+1$	1	n		$(1,n,n)$
4	...	n	2	$n+1$	1	3		$(2,n+1,n-2)$
	...		n	$n+1$	3	$n-1$		$(3,n-1,n)$
	...		n	$n+1$	i	j	$i=1\dots n-1$, $j=1\dots n-2$, $i\neq 2, j-1, j$	(i,j,n)
	...		1	$n+1$	j	n	$j=2\dots n-1$, $j\neq n-2$	(j,n,n)
	...	n	1	$n+1$	2	j	$j=4\dots n-1$	$(2,j,n)$
	...		n	$n+1$	3	4		$(n+1,3,n-1)$
	...		n	$n+1$	j	$j+1$	$j=1\dots n-4, j\neq 2,3$	$(n+1,j,n-1)$

Group 5: Vertex $n+1$ is visited in period 1.

1	2	...	l	$l+1$	$l+2$	...	n	$n+1$	Variable
$n+1$	2	...	l	1	$l+1$	...	$n-1$	n	$(n+1,2,1)$

1	2	3	4	...	$l-2$	$l-1$	l	$l+1$	...	n	$n+1$	Variable
$n+1$	3	4	5	...	$l-1$	l	2	$l+1$	...	n	1	$(n,1,n)$

1	2	...	$n-j+2$	$n-j+3$	...	n	$n+1$		Variable
$n+1$	j	...	n	1	...	$j-2$	$j-1$	$j=3\dots n-2$, $j\neq 4$	$(j-2,j-1,n)$

1	2	3	...	$n-3$	$n-2$	$n-1$	n	$n+1$		Variable
$n+1$	j								$j=1\dots n$, $j\neq 2,4$	$(n+1,j,1)$

Group 6: Vertex $n+1$ is visited in period 2, and the vertex assigned to period 1 is either 1 or n.

1	2	3	...	l	$l+1$	$l+2$	$l+3$	...	$n+1$	Variable
n	$n+1$	3	...	l	1	2	$l+1$	...	$n-1$	$(n+1,3,2)$

1	2	3	...	$n-3$	$n-2$	$n-1$	n	$n+1$	Variable
n	$n+1$	1	...				$n-2$	$n-1$	$(n+1,1,2)$

1	2	3	...	$n-1$	n	$n+1$		Variable
1	$n+1$	j				n	$j = 4 \dots n-3$	$(n+1, j, 2)$
1	$n+1$	n						$(n+1, n, 2)$

Group 7: The tours within this group do not share any particular structure besides that vertex $n+1$ is never assigned to position $n+1$. In addition, the procedure considered so far to prove the linear independence cannot be applied, since every variable regarding vertex $n+1$ or period n appears in at least two tours. For that reason, we prove that these tours are linearly independent by formulating the corresponding system of equations, and showing that they have full column rank.

It is important to appreciate that it is sufficient to consider only the tours in this group because for the remaining tours, i.e., the extended tours arising from $P^n_{TD}(C,k)$, these variables are always set to zero.

1	2	3	$4 \dots n-4$	$n-3$	$n-2$	$n-1$	n	$n+1$	
					1	$n+1$	$n-2$	n	α_1
					1	$n+1$	2	3	α_2
					n	$n+1$	2	3	α_3
					n	$n+1$	$n-3$	$n-2$	α_4
					n	$n+1$	$n-2$	$n-1$	α_5
					n	$n+1$	$n-3$	$n-1$	α_6
n	$n+1$	2					$n-2$	$n-1$	α_7
n	$n+1$	$n-2$					$n-3$	$n-1$	α_8
n	$n+1$	$n-1$					$n-3$	$n-2$	α_9
1	$n+1$	2					$n-1$	n	α_{10}
1	$n+1$	$n-2$					$n-1$	n	α_{11}
1	$n+1$	$n-1$					$n-2$	n	α_{12}
$n+1$	4	5			n	1	2	3	α_{13}
$n+1$	4	1				$n-2$	$n-1$	n	α_{14}

We next show the associated system of equations. Between parenthesis we specify the variable that generates each equation.

1. $\alpha_1 + \alpha_2 = 0$ $(y_{1n+1n-2})$
2. $\alpha_3 + \alpha_4 + \alpha_5 + \alpha_6 = 0$ $(y_{nn+1n-2})$
3. $\alpha_2 + \alpha_3 = 0$ $(y_{n+12n-1})$
4. $\alpha_1 + \alpha_5 = 0$ $(y_{n+1n-2n-1})$
5. $\alpha_2 + \alpha_3 + \alpha_{13} = 0$ (y_{23n})
6. $\alpha_1 + \alpha_{12} = 0$ (y_{n-2nn})
7. $\alpha_5 + \alpha_7 = 0$ $(y_{n-2n-1n})$
8. $\alpha_6 + \alpha_8 = 0$ $(y_{n-3n-1n})$
9. $\alpha_4 + \alpha_9 = 0$ $(y_{n-3n-2n})$
10. $\alpha_7 + \alpha_8 + \alpha_9 = 0$ (y_{nn+11})
11. $\alpha_{13} + \alpha_{14} = 0$ (y_{n+141})
12. $\alpha_7 + \alpha_{10} = 0$ (y_{n+122})
13. $\alpha_8 + \alpha_{11} = 0$ $(y_{n+1n-22})$
14. $\alpha_9 + \alpha_{12} = 0$ $(y_{n+1n-12})$
15. $\alpha_4 + \alpha_6 = 0$ $(y_{n+1n-3n-1})$
16. $\alpha_{10} + \alpha_{11} + \alpha_{14} = 0$ (y_{n-1nn})
17. $\alpha_{10} + \alpha_{11} + \alpha_{12} = 0$ (y_{1n+11})

It is quite easy to check that equations 1 to 14 are linearly independent. Therefore, the only solution to the system is $\alpha_i = 0$, $i = 1, \dots, 14$, and the tours are linearly independent.

Finally, we know by inductive hypothesis that the extended tours used to prove the dimension of $P^n_{TD}(C,k)$ are linearly independent. This completes the proof, since we have $(n+1)n(n-1) - (l+1)(l-2) + 1$ affinely independent tours. □

Based on this theorem, we now state the following result.

Theorem 3.3.3. *Let $C =< v_1, v_2, \ldots, v_l, v_{l+1} = v_1 >$, $4 \leq l \leq n$ and k as defined in Proposition 3.3.1. The TDCI (3.15) are facet-defining for $P^n_{TD}(C, k)$ for $n \geq 5$.*

Proof. Following the same notation as in Theorem 3.3.2 for C and k, the tours that satisfy inequality (3.15) by equality must conform to one of the following patterns:

- vertex i is assigned to time period i, for $i = 1, \ldots, l$;
- vertex i is not assigned to time period i, for $i = 1, \ldots, l$;
- for $1 \leq i_0 \leq l$, vertex i is assigned to time period i, $1 \leq i \leq i_0$, and vertex j is not assigned to time period j, $i_0 < j \leq l$;
- for $2 \leq i_0 \leq l$, vertex j is assigned to time period j, $i_0 \leq j \leq l$, vertex 1 is assigned to time period $l + 1$, and vertex i is not assigned to time period i, $2 \leq i < i_0$.

All tours proposed in Theorem 3.3.2 follow one of these patterns, except for the first tour in Group 3 of the basis, which is shown below.

1	2	3	4	5	6	...	$n-1$	n
3	1	2	4	$n-1$	5	...	$n-2$	n

Therefore, using the same arguments as in the previous result, we have $n(n-1)(n-2)-(l+1)(l-2)$ affinely independent tours satisfying (3.15) by equality, which proves the result. □

Separation of time-dependent cycle inequalities

In this section we study the complexity of the separation problem associated with TDCI inequalities. We consider the reformulation of the TDTSP as the problem of finding the shortest constrained path in a directed multi-partite graph, as suggested by Picard and Queyranne [53] and Vander Wiel and Sahinidis [60]. Let $D_M = (V_M, A_M)$ be this graph. A vertex $u \in V_M$ is a combination between a vertex $i \in V$ and a time period k, where $0 \leq k \leq n$. We will denote such vertex as $u = ik$. In addition, we will refer to arcs $e \in A_M$ as $e = (ik, jk + 1)$, with $ik, jk + 1 \in V_M$. Figure 3.2 shows a graphical representation of D_M.

The separation problem for this family of inequalities can be formulated as follows:

	Time-Dependent Cycle Inequalities Separation
Instance:	A point $\tilde{z} = (\tilde{y}, \tilde{x}) \in P_{TDTSP}$
Question:	Does $\tilde{z}$ violates some Time-Dependent Cycle Inequality?.

We begin showing that this family can be separated considering the support graph, $D_M(\tilde{z})$, instead of the complete graph. The following proposition proves this result.

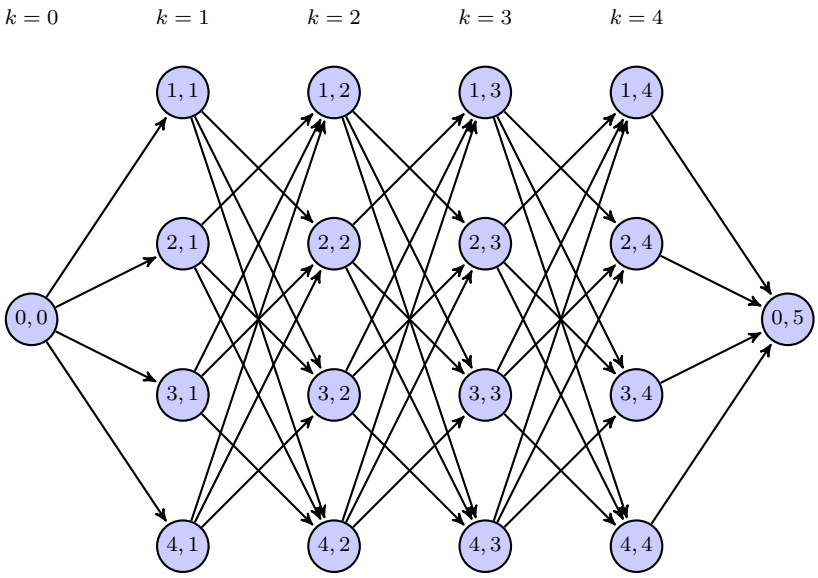


Figure 3.2.: Graphical representation of D_M.

Proposition 3.3.4. *Let $\tilde{z} = (\tilde{y}, \tilde{x})$ be a solution of the relaxation that satisfies (3.6) - (3.13) and $\tilde{z} \geq 0$, $C = \{v_1, \ldots, v_l\} \subset V$ and k the set of nodes and the starting time period associated with a TDCI, and $i_0 \in \{1, \ldots, l\}$. If $\tilde{y}_{v_{i_0} v_{i_0+1} k+i_0-1} = 0$, then the TDCI induced by C and k is trivially satisfied.*

Proof. From Proposition 3.3.1, the TDCI associated with C and k, can be rewritten considering v_{i_0} as

$$\underbrace{\left(\sum_{i=1}^{i_0-1} \tilde{y}_{v_i v_{i+1} k+i-1} - \sum_{i=1}^{i_0-1} \tilde{x}_{v_{i+1} k+i}\right)}_{A} + \tilde{y}_{v_{i_0} v_{i_0+1} k+i_0-1} + \underbrace{\left(\sum_{i=i_0+1}^{l} \tilde{y}_{v_i v_{i+1} k+i-1} - \sum_{i=i_0}^{l-1} \tilde{x}_{v_{i+1} k+i}\right)}_{B} \leq 0.$$

It is easy to see that $A \leq 0$ and $B \leq 0$, since solution $\tilde{z}$ satisfies equations (3.9) and (3.11), respectively. Then, knowing that $\tilde{y}_{v_{i_0} v_{i_0+1} k+i_0-1} = 0$, the inequality is satisfied by $\tilde{z}$. □

Based on this result, we first identify some necessary conditions for the existence of a violated TDCI. As we have seen before, the main idea behind TDCI is the presence of a time-dependent cycle, starting and ending in the same node but with different time-periods. Then, there must exist vertices vk, $vk' \in V_M$, with $k' = k+l$, for some $l > 1$, $k+l \leq n$. In addition, even when this condition holds, it is also necessary the existence of a simple path - in terms of vertices $v_i \in V$

- going from vk to vk', which is not always the case. Let v_1k, $v_1k' \in V_M$ be those vertices as defined before. We want to determine if there exists or not a set of vertices $C = \{v_2, \ldots, v_l\} \subset V$ such that

$$\sum_{i=1}^{l} \tilde{y}_{v_i v_{i+1} k+i-1} - \sum_{i=1}^{l-1} \tilde{x}_{v_{i+1} k+i} \quad > \quad 0, \tag{3.18}$$

where $v_{l+1} = v_1$. We now restrict our search to the subgraph defined by those vertices in the support graph with time periods between k and k', i.e., $u = vt$ such that $v \in V$ and $k < t < k'$. For this subgraph, we also define the arc weight function $w : A_M \to \mathbb{R}$ as

$$w(it, jt+1) = \begin{cases} \tilde{y}_{v_i v_j t} & \text{if } i = 1,\ t = k \\ \tilde{y}_{v_i v_j t} - \tilde{x}_{v_i t} & \text{otherwise.} \end{cases}$$

Considering these two definitions, we seek for a maximum-weight path connecting v_1k and v_1k'. This can be done in polynomial time with a straightforward application of dynamic programming.

However, the optimal solution returned by the maximum path algorithm may not be simple in terms of vertices v from the original graph (i.e., it is possible that the path containts vertices $v_it, v_it' \in V_M$, with $t \neq t'$, visiting more than one time the row corresponding to vertex v_i in different time periods). Let P be this optimal solution, and w^* the weight of P. Clearly, if $w^* \leq 0$, then $\tilde{z}$ does not violate any TDCI starting and ending in v_1k and v_1k', respectively, since w^* is an upper bound for the value of any simple path connecting these two vertices. On the contrary, if $w^* > 0$, there might exist a TDCI that is violated by $\tilde{z}$. The following proposition proves that if this condition on w^* holds, then we can separate $\tilde{z}$.

Proposition 3.3.5. *Let $\tilde{z} = (\tilde{y}, \tilde{x})$ be a solution of the relaxation that satisfies (3.6) - (3.13) and $\tilde{z} \geq 0$, and $P =< v_1, \ldots, v_l >$, $v_{l+1} = v_1$, the sequence of nodes returned by the maximum path algorithm with optimum value $w^* > 0$. Then, there exists a subsequence $P_0 \subseteq P$ and a time period k_0 such that the TDCI defined by P_0' and k_0 is violated by $\tilde{z}$.*

Proof. We start by pointing out that if sequence P has no repetitions, then clearly P and k define a violated TDCI, since $w^* > 0$.

Otherwise, P has at least one repeated vertex and each vertex may appear even more than two times. We also know that these repetitions cannot be in consecutive time periods, since this situation is not allowed even in graph D_M. However, it is not difficult to see that there exists a repeated vertex $v_{i_0} = v_{i_1} \in P$, $i_0 < i_1$, appearing in time periods $k_0 = k+i_0-1$ and $k_0' = k+i_1-1$, respectively, such that the vertices of P between two of its repetitions defines a subsequence of P which is a simple path connecting $v_{i_0}k_0$ and $v_{i_0}k_0'$. Let $P_0 = \{v_{i_0}, \ldots, v_{i_1-1}\} \subset P$ be this sequence, where $|P_0| \geq 3$. We now analyze the TDCI defined by P and k in terms of subsequence

P_0 and k_0. Expression (3.18) can be rewritten as

$$\sum_{i=1}^{i_0-1} \tilde{y}_{v_i v_{i+1} k+i-1} - \sum_{i=1}^{i_0-1} \tilde{x}_{v_{i+1} k+i} + \tag{3.19}$$

$$\sum_{i=i_0}^{i_1-1} \tilde{y}_{v_i v_{i+1} k+i-1} - \sum_{i=i_0}^{i_1-2} \tilde{x}_{v_{i+1} k+i} + \tag{3.20}$$

$$\sum_{i=i_1}^{l} \tilde{y}_{v_i v_{i+1} k+i-1} - \sum_{i=i_1-1}^{l-1} \tilde{x}_{v_{i+1} k+i} \quad > \quad 0, \tag{3.21}$$

where (3.20) is the expression related to P_0. We consider each part separately:

1. The value of expression (3.19) is at most zero, since solution $\tilde{z}$ satisfies equation (3.9).

2. Analogously, expression (3.21) is also upper bounded by zero, given that $\tilde{z}$ satisfies (3.11).

3. From the previous two items, it follows that (3.20) is strictly greater than zero. In addittion to this fact, by construction, P_0 has no repeated vertices. Moreover, if we consider $k_0 = k + i_0 - 1$, expression (3.20) defines a TDCI for P_0 and k_0. Then, we have found a sequence of vertices P_0 and time period k_0 that represent a TDCI violated by $\tilde{z}$.

To sum up, given the sequence of nodes P for whose optimal value w^* is strictly greater than zero, we are able to find a TDCI that is violated by $\tilde{z}$. □

Finally, considering all we have seen so far, we can state that the separation problem for the TDCI can be solved in polynomial time. A pseudocode for the separation procedure summarizing all steps from this section is shown in Algorithm 3.1.

Algorithm 3.1 TDCI Separation Algorithm

Input: $\tilde{z} = (\tilde{y}, \tilde{x}) \in P_{TDTSP}$.

1. **for** $ik, ik' \in D_M(\tilde{z})$, $k' > k+1$, **do:**

2. Calculate the maximum path on $D_M(\tilde{z})$ considering $w : A_M \to \mathbb{R}$ as the arc weigth function. Let P be the optimal solution and w^* its cost.

3. **If** $w^* > 0$ **then**

4. If P is simple, then answer *yes* and return the TDCI induced by P.

5. **else**

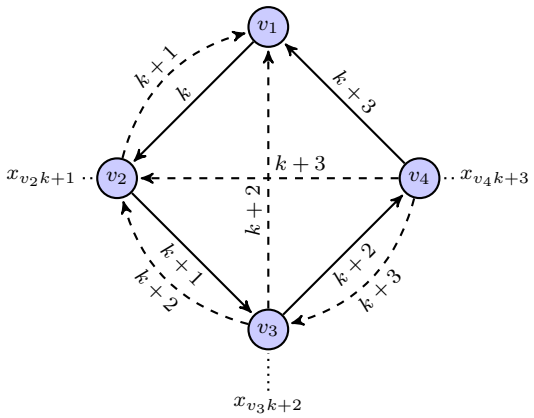
6. Determine the simple subpath $P_0 \subseteq P$ as shown in Proposition 3.3.5 answer *yes* and return the TDCI induced by P_0.

7. **end if**

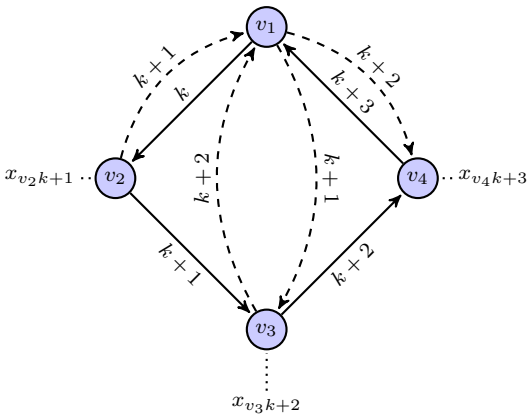
8. **end for**

3.3.2. Lifted time-dependent cycle inequalities

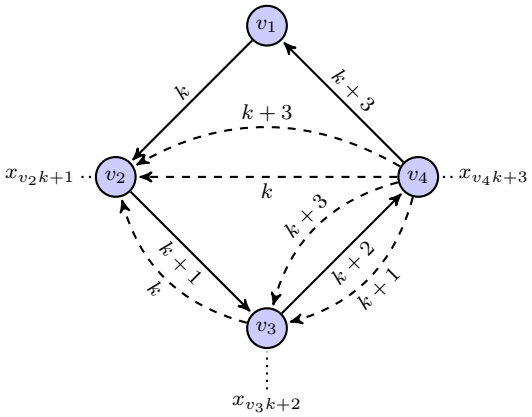
Based on the ideas from Balas and Fischetti [7, 34], from Proposition 3.3.3 we can derive facets of P^n_{TD} applying a maximum sequential lifting over the variables present in $F_1, \ldots, F_5$. It is well known that the order in which variables are lifted may generate different inequalities. Indeed, we lifted these variables in five different ways to obtain five families of facets. The support graph for some of these inequalities with $l = 4$ is shown in Figure 3.3. In all cases, a solid arc corresponds to a variable in the original TDCI, and a dashed arc represents a variable obtained by the lifting process. In addition, the x_{ik} variables involved in the inequality are placed next to the corresponding vertex.



(a) Support for Lifted TDCI (3.22).



(b) Support for Lifted TDCI (3.23).



(c) Support for Lifted TDCI (3.24).

Figure 3.3.: Support for Lifted TDCI inequalities.

The following proposition shows the five different families of facets.

Proposition 3.3.6. *Let $C =< v_1, v_2, \ldots, v_l, v_{l+1} = v_1 >$, $l \leq n$, be a simple cycle with transitions between consecutive vertices taken in time intervals $k, k+1, \ldots, k+l-1$, $k+l \leq n$. Then,*

inequalities

$$\sum_{i=1}^{l} y_{v_i v_{i+1} k+i-1} + \sum_{i=2}^{l-1}\sum_{j=1}^{i-1} y_{v_i v_j k+i-1} + \sum_{j=2}^{l-1} y_{v_l v_j k+l-1} \leq \sum_{i=1}^{l-1} x_{v_{i+1} k+i} \tag{3.22}$$

$$\sum_{i=1}^{l} y_{v_i v_{i+1} k+i-1} + \sum_{j=3}^{l} y_{v_1 v_j k+j-2} + \sum_{j=2}^{l-1} y_{v_j v_1 k+j-1} \leq \sum_{i=1}^{l-1} x_{v_{i+1} k+i} \tag{3.23}$$

$$\sum_{i=1}^{l} y_{v_i v_{i+1} k+i-1} + \sum_{i=3}^{l-1}\sum_{j=2}^{i-1} y_{v_i v_j k+j-2} + \sum_{j=2}^{l-1} y_{v_l v_j k+l-1} + \sum_{j=2}^{l-1} y_{v_l v_j k+j-2} \leq \sum_{i=1}^{l-1} x_{v_{i+1} k+i} \tag{3.24}$$

$$\sum_{i=1}^{l} y_{v_i v_{i+1} k+i-1} + \sum_{i=3}^{l-1}\sum_{j=2}^{i-1} y_{v_i v_j k+j-2} + \sum_{j=3}^{l} y_{v_1 v_j k+j-2} + \sum_{j=2}^{l-1} y_{v_l v_j k+j-2} \leq \sum_{i=1}^{l-1} x_{v_{i+1} k+i} \tag{3.25}$$

$$\sum_{i=1}^{l} y_{v_i v_{i+1} k+i-1} + \sum_{i=3}^{l-1}\sum_{j=2}^{i-1} y_{v_i v_j k+j-2} + \sum_{j=3}^{l} y_{v_1 v_j k+j-2} + y_{v_{l-1} v_1 k+l-2} \leq \sum_{i=1}^{l-1} x_{v_{i+1} k+i} \tag{3.26}$$

define facets of P_{TD}^n.

Proof. We show the detailed proof for inequalities (3.23). Similar arguments are used for the remaining inequalities, for which we only provide the order of the variables for the sequential maximum lifting.

Recall the definition of sets $F_1, \ldots, F_5$. The variables lifted in inequalities (3.23) correspond to the ones in set F_5, i.e. $y_{v_1 v_j k+j-2}$ for $j = 3, \ldots, l$, and variables from set F_4 for $j = 1$, i.e. $y_{v_i v_1 k+j-1}$ for $i = 2, \ldots, l-1$. This is in fact the order in which the first variables are lifted.

We divide the proof in four parts. First, since we consider the maximum lifting theorem, for the sake of completeness we include again the characterization of the feasible solutions satisfying the TDCI by equality from Theorem 3.3.3, which will be used in different parts of the proof. Then, we apply the maximum lifting procedure for variables in F_5 and subsequently to the subset of variables from F_4, showing that the coefficient obtained in each case is one. Finally, we prove that the coefficient obtained by means of maximum lifting for the remaining variables in sets $F_1, \ldots, F_4$ are zero, which completes the proof.

1. *Analyzing the TDCI.* Due to its feasibility, we know that the expression

$$\sum_{i=1}^{l} y_{v_i v_{i+1} k+i-1} - \sum_{i=1}^{l-1} x_{v_{i+1} k+i} \leq 0.$$

In particular, it is relatively easy to see that for a feasible solution equality occurs if and only if the solution satisfies:

- Vertices $v_2, \ldots, v_l$ are not assigned to positions $k+1, \ldots, k+l-1$, respectively. In other words, $x_{v_{i+1} k+i} = 0$ for $i = 1, \ldots, l-1$.

- Vertices $v_1, \ldots, v_{l_0}$, with $l_0 \leq l$, are assigned to positions $k, k+1, \ldots, k+l_0-1$ and vertices $v_{l_0+1}, \ldots, v_l$ are not assigned to positions $k+l_0, \ldots, k+l-1$, respectively. That is, regarding the variables involved in the TDCI induced by C the feasible solution only contains a time dependent proper subpath of C starting at vertex v_1.
- Vertices $v_{l_0}, \ldots, v_l, v_{l+1} = v_1$, with $l_0 > 1$, are assigned to positions $k+l_0-1, \ldots, k+l-1, k+l$ and vertices $v_2, \ldots, v_{l_0-1}$ are not assigned to positions $k+1, \ldots, k+l-1$, respectively. That is, regarding the variables involved in the TDTI induced by C the feasible solution only contains a proper time dependent subpath of C ending at vertex $v_{l+1} = v_1$.

For feasible solutions which do not satisfy this characterization the inequality is satisfied strictly, since other partial subpaths will produce more assignment variables (x_{ik}) than transition variables (y_{ijk}) having value one.

2. *Lifting variables in F_5.* We begin by lifting variable $y_{v_1v_3k+1}$ from set F_5. As established in the maximum lifting theorem, to determine the coefficient for variable $y_{v_1v_3k+1}$ we maximize the following expression

$$z^0 = \sum_{i=1}^{l} y_{v_iv_{i+1}k+i-1} - \sum_{i=1}^{l-1} x_{v_{i+1}k+i}$$

subject to $y_{v_1v_3k+1} = 1$ and constraints (3.6) - (3.14). For this problem, $y_{v_1v_3k+1} = 1$ implies that $x_{v_1k+1} = 1$ and $x_{v_3k+2} = 1$. Thus, any feasible solution for this problem does not satisfy conditions from item 1 and it is easy to see that the maximum value is $z^0 = -1$. Therefore, the coefficient obtained by performing a maximum lifting for variable $y_{v_1v_3k+1}$ is 1.

We now argue recursively. Suppose we already lifted variables $y_{v_1v_3k+1}, \ldots, y_{v_1v_{l_0-1}k+l_0-3}$, $4 \leq l_0 \leq l$, and the resulting coefficients in all cases is 1. In order to lift variable $y_{v_1v_{l_0}k+l_0-2}$ we maximize the expression

$$z^1 = \sum_{i=1}^{l} y_{v_iv_{i+1}k+i-1} + \sum_{j=3}^{l_0-1} y_{v_1v_jk+j-2} - \sum_{i=1}^{l-1} x_{v_{i+1}k+i}$$

subject to $y_{v_1v_{l_0}k+l_0-2} = 1$ and constraints (3.6) - (3.14). Similarly to the previous case, $y_{v_1v_{l_0}k+l_0-2} = 1$ implies $x_{v_1k+l_0-2} = 1$ and $x_{v_{l_0}k+l_0-1} = 1$. Also, the variables included from previous liftings, i.e. $y_{v_1v_jk+j-2}$, must have value 0 because of $y_{v_1v_{l_0}k+l_0-2} = 1$ and degree constraints. Then, we are in similar case as in the first lifting. Since by hypothesis $4 \leq l_0 \leq l$, vertex v_1 is neither in position k or $k+l$ and, due to $x_{v_{l_0}k+l_0-1} = 1$, all feasible solutions do not satisfy conditions from the previous item 1. Thus, in this case we also have $z^1 = -1$ and therefore the coefficient for variable $y_{v_1v_{l_0}k+l_0-2}$ is 1.

By this, we proved that by applying a sequential maximum lifting procedure the coefficients for all variables in F_5 is one, and therefore

$$\sum_{i=1}^{l} y_{v_iv_{i+1}k+i-1} + \sum_{j=3}^{l} y_{v_1v_jk+j-2} \leq \sum_{i=1}^{l-1} x_{v_{i+1}k+i} \tag{3.27}$$

is a valid inequality for P^n_{TD}.

3. *Lifting variables included in F_4.* We will argue in an analogous fashion as in item 2 but considering variables $y_{v_j v_1 k+j-1}$, for $j = 2, \ldots, l-1$, and starting from inequality (3.27).

   We begin by considering variable $y_{v_2 v_1 k+1}$. For the maximum lifting, we maximize

   $$z^0 = \sum_{i=1}^{l} y_{v_i v_{i+1} k+i-1} + \sum_{j=3}^{l} y_{v_1 v_j k+j-2} - \sum_{i=1}^{l-1} x_{v_{i+1} k+i}$$

   subject to $y_{v_2 v_1 k+1} = 1$ and constraints (3.6) - (3.14). In this case, $y_{v_2 v_1 k+1} = 1$ implies that $x_{v_2 k+1} = x_{v_1 k+2} = 1$ and $y_{v_1 v_j k+j-2} = 0$, for $j = 3, \ldots, l$, $j \neq 4$. We analyze the two possible values for this special case when $j = 4$.

   If $y_{v_1 v_4 k+2} = 0$, the argument is similar as the ones from item 2 and since v_1 is in position $k+2$ and vertex v_2 in position $k+1$, then $z^0 = -1$. Otherwise, $y_{v_1 v_4 k+2} = 1$ implies that $x_{v_4 k+3} = 1$ and the same argument holds. Then, $z^0 = -1$ and the coefficient for variable $y_{v_2 v_1 k+1}$ is 1.

   For the recursive step, when lifting variable $y_{v_{l_0} v_1 k+l_0-1}$ for $3 \leq l_0 \leq l-1$, the argument is rather the same. Having $y_{v_{l_0} v_1 k+l_0-1} = 1$ forces all variables from F_4 lifted previously to take value zero and also $x_{v_{l_0} k+l_0-1} = x_{v_1 k+l_0} = 1$. Then, the coefficient obtained by applying a maximum lifting is 1.

   Then, we proved that inequality (3.23) is valid for P_{TD}^n. We are left showing that the remaining variables in sets $F_1, \ldots, F_4$, the coefficient obtained when applying a sequential maximum lifting is zero.

4. *Lifting the remaining variables in $F_1, \ldots, F_4$.* Starting form inequality (3.23), for the remaining variables it is sufficient to prove that if each of them is set to 1, there exists an assignment of variables that satisfy constraint (3.23) by equality. Therefore, the coefficient obtained by applying maximum lifting is zero.

   - The other variables in F_4 are $y_{v_i v_j k+i-1}$, for $i = 3, \ldots, l-1$ and $j = 2, \ldots, i-1$. Consider feasible solutions having $y_{v_i v_j k+i-1} = 1$, with i, j fixed. In particular, if we consider the ones having $y_{v_1 v_i k+i-2} = 1$ and the the remaining vertices from C, v_t, not assigned to positions $k+t-1$, for $t = 2, \ldots, l$, $t \neq i, j$, inequality (3.23) is satisfied by equality.
   - For variables in F_3 we argue in a similar fashion. Feasible solutions having variables $y_{v_i v_j k+j-2} = 1$, for $i = 3, \ldots, l-1$ and $j = 2, \ldots, i-1$, $y_{v_j v_1 k+j-1} = 1$ and the remaining vertices in C not assigned to the positions considered in C satisfy (3.23) by equality.
   - Variables in F_2 are $y_{v_l v_j k+l-1}$, $j = 2, \ldots, l-1$. In each case, feasible solutions having also $y_{v_1 v_l k+l-2} = 1$ and the remaining vertices assigned to different positions as in C satisfies (3.23) by equality.
   - Finally, for variables in F_1, $y_{v_l v_j k+j-2}$, feasible solutions having also $y_{v_j v_1 k+j-1} = 1$ and the remaining vertices assigned to different positions as in C satisfies (3.23) by equality.

Then, coefficients for the remaining variables are zero.

These four items prove that constraints (3.23) are facet defining since for all variables in sets $F_1, \ldots, F_5$ a sequential maximum lifting is applied.

For the remaining cases, the arguments are similar to the ones used for constraints (3.23). We show below the order in which sets must be considered to obtain the corresponding facet. Within each set variables can be lifted in any order, obtaining the same result.

- Constraints (3.22): F_4, F_2
- Constraints (3.24): F_1, F_2, F_3
- Constraints (3.25): F_5, F_1, F_3
- Constraints (3.26): F_5, F_3, F_4

□

This proposition proves that constraints (3.22) - (3.26) are facet defining. We next remark that these five lifting of a TDCI indeed define different facets of P^n_{TD}. For this purpose, we use feasible solutions satisfying them by equality. Let $C =< v_1, v_2, v_3, v_4 >$ be a time dependent cycle. We consider feasible solutions with the following permutations of the vertices.

- $v_4v_1v_3v_2$ satisfies by equality (3.22), (3.23), (3.25) and (3.26), and strictly (3.24). Then, constraint (3.24) is different from the other Lifted TDCIs.
- $v_4v_2v_1v_3$ satisfies by equality (3.22), (3.23), (3.24) and (3.25), and strictly (3.26). Then, constraint (3.26) is different from (3.22), (3.23) and (3.25).
- $v_3v_2v_4v_1$ satisfies by equality (3.24), (3.25), (3.26), and strictly (3.22) and (3.23). Then, constraint (3.25) is different from (3.22) and (3.23).
- $v_3v_1v_5v_4v_2$ satisfies by equality (3.22) and (3.24), and strictly (3.23), (3.25) and (3.26). Then, constraint (3.22) is different from (3.23).

Separation procedure

In order to incorporate these inequalities to the B&C algorithm, we use a heuristic separation routine based on the separation algorithm for the TDCI. For every two vertices of the form ik and ik' that are present in the support graph, we execute the maximum path algorithm. If the returned path is simple (in terms of vertices in V of the original graph D), then we add the value of the lifted variables involved in the inequality to the cost of this path. The outline of the generic procedure is described in Algorithm 3.2.

Although this approach is heuristic, in practice it is quite effective for all families, finding a considerable number of violated cuts.

Algorithm 3.2 LIFTED TDCI SEPARATION PROCEDURE

Input: $\tilde{z} = (\tilde{y}, \tilde{x}) \in P_{TDTSP}$.

1. **for** $ik, ik' \in D_M(\tilde{z})$, $k' > k+1$, **do:**
2. Calculate the maximum path on $D_M(\tilde{z})$ considering $w : A_M \to \mathbb{R}$ as the arc weight function. Let P be the optimal solution and w^* its cost.
3. **If** P is simple **then**
4. Add to w^* the values from $\tilde{y}$ corresponding to the variables according to the lifting under consideration. If this new value is strictly positive add the corresponding Lifted TDCI to the cut pool.
5. **end if**
6. **end for**

3.3.3. Polynomially-sized families

In this section we present four new families of valid inequalities for P^n_{TD}that are also incorporated in the B&C algorithm. The following proposition introduces the first of them. The idea behind this family is to bound a particular variable, $y_{i_0j_0k_0}$ based on the values of the variables representing arcs leaving a particular vertex, l_0, with $l_0 \neq i_0, j_0$. Figure 3.4 shows the support for constraints (3.28) in Family 1. The remaining constraints follow a similar idea.

Proposition 3.3.7 (Family 1)**.** *For $i_0, j_0, l_0 = 1, \ldots, n$, $i_0 \neq j_0 \neq l_0$, inequalities*

$$y_{i_0j_0k_0} + y_{j_0l_0n-1} + y_{i_0l_0n-1} \leq y_{l_0i_0k_0-1} + \sum_{\substack{t=1 \\ t\neq k_0-1,k_0,k_0+1}}^{n-1} \sum_{\substack{w=1 \\ w\neq i_0,j_0,l_0}}^{n} y_{l_0wt} + y_{l_00n} \tag{3.28}$$

for $k_0 = 2, \ldots, n-3$, and

$$y_{i_0j_0n-2} + \sum_{\substack{w=1 \\ w\neq j_0,l_0}}^{n} y_{wl_0n-1} \leq y_{l_0i_0n-3} + \sum_{t=1}^{n-4} \sum_{\substack{w=1 \\ w\neq i_0,j_0,l_0}}^{n} y_{l_0wt} + y_{l_00n} \tag{3.29}$$

$$y_{i_0j_0n-1} \leq y_{l_0i_0n-2} + \sum_{t=1}^{n-3} \sum_{\substack{w=1 \\ w\neq i_0,j_0,l_0}}^{n} y_{l_0wt} \tag{3.30}$$

are valid for P^n_{TD}.

Proof. We start by noting an identity that will be used several times within the proof. If we focus in the case where $y_{i_0j_0k_0} = 1$, and considering equations (3.2), (3.3), (3.4) and (3.5), it is quite easy to see that for $1 \leq k \leq n-1$,

$$y_{l_0i_0k-1} + \sum_{\substack{w=1 \\ w\neq i_0, \\ j_0,l_0}}^{n} \sum_{\substack{t=1 \\ t\neq k-1, \\ k,k+1}}^{n-1} y_{l_0wt} + y_{l_00n} = y_{0l_00} + \sum_{\substack{w=1 \\ w\neq l_0}}^{n} \sum_{t=1}^{n-1} y_{wl_0t} = 1. \tag{3.31}$$

Now we prove the validity of each inequality separately.

1. For inequality (3.28), we consider three different cases:

   - If $y_{i_0j_0k_0} = y_{j_0l_0n-1} = y_{i_0l_0n-1} = 0$, the inequality is trivially satisfied since all variables are non-negative.
   - If $y_{i_0j_0k_0} = 1$, it follows that $y_{j_0l_0n-1}$ and $y_{i_0l_0n-1}$ have value zero since $k_0 \leq n-3$. Then, by (3.31) the inequality is satisfied.
   - If either $y_{j_0l_0n-1} = 1$ or $y_{i_0l_0n-1} = 1$, variable $y_{i_0j_0k_0} = 0$ and the inequality is satisfied since by equations (3.4) $y_{l_00n} = 1$.

   These three cases cover all possible situations, and therefore inequality (3.28) is valid for P^n_{TD}.

2. For inequality (3.29), we separate again the proof in three cases:

   - If $y_{i_0j_0n-2}$ and y_{wl_0n-1} are zero, for $w = 1, \ldots, n$, $w \neq l_0, j_0$, then the inequality is trivially satisfied.
   - If $y_{i_0j_0n-2} = 1$, $y_{wl_0n-1} = 0$ for $w = 1, \ldots, n$, $w \neq l_0, j_0$. Using equation (3.31) for $k = n-2$, we get the expression in the rhs of the inequality, and therefore it is satisfied.
   - If for some $w = 1, \ldots, n$, $w \neq l_0, j_0$, $y_{wl_0n-1} = 1$, it follows that $y_{i_0j_0n-2} = 0$ and from equations (3.4), $y_{l_00n} = 1$ and the inequality is satisfied too.

   Therefore, the inequality is valid from P^n_{TD}.

3. Finally, we consider inequality (3.30). If $y_{i_0j_0n-1} = 0$, the inequality is satisfied. Otherwise, considering equation (3.31) and the fact $y_{i_0j_0n-1} = 1$ implies that $y_{l_00n} = 0$, the inequality is also valid for P^n_{TD}.

Therefore, inequalities (3.28), (3.29) and (3.30) are valid for P^n_{TD}. □

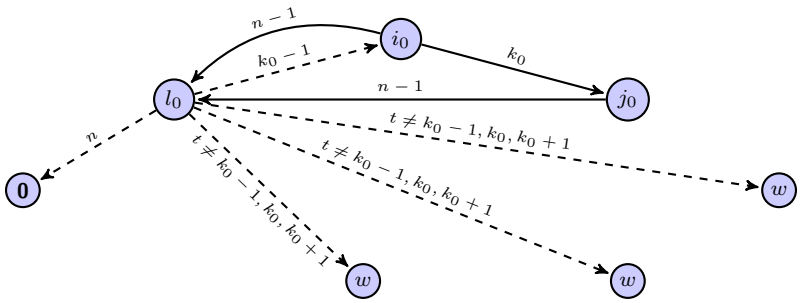


Figure 3.4.: Support for constraints (3.4) in Family 1.

The next family uses mainly the same idea as in the previous proposition, with the only difference that instead of looking at the arcs leaving l_0, we consider the entering ones. Figure 3.5 shows the support for the first group of constraints in this family. Regarding the proof for this family, since it is analogous to the one in Proposition 3.3.7, we omit the details.

Proposition 3.3.8 (Family 2). *For $i_0, j_0, l_0 = 1, \ldots, n$, $i_0 \neq j_0 \neq l_0$, inequalities*

$$y_{i_0 j_0 k_0} + y_{l_0 i_0 1} + y_{l_0 j_0 1} \leq y_{j_0 l_0 k_0+1} + \sum_{\substack{t=1 \\ t \neq k_0-1, k_0, k_0+1}}^{n-1} \sum_{\substack{w=1 \\ w \neq i_0, j_0, l_0}}^{n} y_{w l_0 t} + y_{0 l_0 0}$$

for $k_0 = 3, \ldots, n-2$, and

$$\begin{aligned} y_{i_0 j_0 2} + \sum_{\substack{w=1 \\ w \neq i_0, l_0}}^{n} y_{l_0 w 1} &\leq y_{j_0 l_0 3} + \sum_{t=4}^{n-1} \sum_{\substack{w=1 \\ w \neq i_0, j_0, l_0}}^{n} y_{w l_0 t} + y_{0 l_0 0} \\ y_{i_0 j_0 1} &\leq y_{j_0 l_0 2} + \sum_{t=3}^{n-1} \sum_{\substack{w=1 \\ w \neq i_0, j_0, l_0}}^{n} y_{w l_0 t} \end{aligned}$$

are valid for P^n_{TD}.

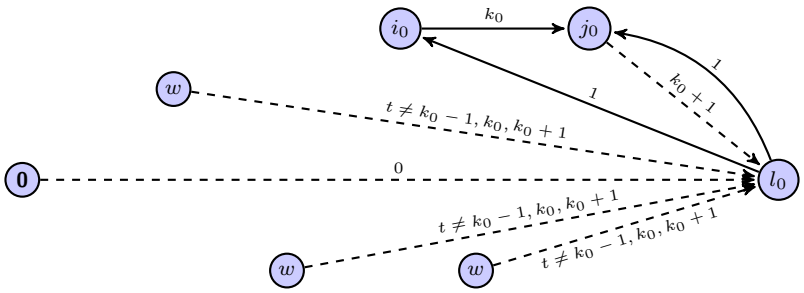


Figure 3.5.: Support for Family 2.

By combining inequalities from propositions 3.3.7 and 3.3.8 in a particular way, we derive the two other families. Next we introduce the inequalities resulting of the combination of inequalities from Proposition 3.3.8.

Proposition 3.3.9 (Family 3). *For $i_0, j_0, l_0 = 1, \ldots, n$, $i_0 \neq j_0 \neq l_0$, inequalities*

$$y_{i_0 j_0 k_0} + y_{j_0 i_0 k_0} + y_{l_0 i_0 1} + y_{l_0 j_0 1} \leq y_{i_0 l_0 k_0+1} + y_{j_0 l_0 k_0+1} + \sum_{\substack{t=1 \\ t \neq k_0-1, \\ k_0, k_0+1}}^{n-1} \sum_{\substack{w=1 \\ w \neq i_0, \\ j_0, l_0}}^{n} y_{w l_0 t} + y_{0 l_0 0} \tag{3.32}$$

for $k_0 = 3, \ldots, n-2$, and

$$y_{i_0 j_0 2} + y_{j_0 i_0 2} + \sum_{\substack{w=1 \\ w \neq i_0, j_0, l_0}}^{n} y_{l_0 w 1} \leq y_{i_0 l_0 3} + y_{j_0 l_0 3} + \sum_{t=4}^{n-1} \sum_{\substack{w=1 \\ w \neq i_0, j_0, l_0}}^{n} y_{w l_0 t} + y_{0 l_0 0} \tag{3.33}$$

$$y_{i_0 j_0 1} + y_{j_0 i_0 1} \leq y_{i_0 l_0 2} + y_{j_0 l_0 2} + \sum_{t=3}^{n-1} \sum_{\substack{w=1 \\ w \neq i_0, j_0, l_0}}^{n} y_{w l_0 t} \tag{3.34}$$

are valid for P^n_{TD}.

Proof. We prove first the validity of inequality (3.32). From the previous proposition, we know that

$$y_{i_0j_0k_0} + y_{l_0i_01} + y_{l_0j_01} \leq y_{j_0l_0k_0+1} + \sum_{\substack{t=1 \\ t\neq k_0-1,k_0,k_0+1}}^{n-1} \sum_{\substack{w=1 \\ w\neq i_0,j_0,l_0}}^{n} y_{wl_0t} + y_{0l_00}$$

$$y_{j_0i_0k_0} + y_{l_0i_01} + y_{l_0j_01} \leq y_{i_0l_0k_0+1} + \sum_{\substack{t=1 \\ t\neq k_0-1,k_0,k_0+1}}^{n-1} \sum_{\substack{w=1 \\ w\neq i_0,j_0,l_0}}^{n} y_{wl_0t} + y_{0l_00}$$

are valid inequalities. Given that $y_{i_0j_0k_0}$ and $y_{j_0i_0k_0}$ are mutually exclusive, the proposed inequality is valid for P^n_{TD}. The same argument can be used for inequalities (3.33) and (3.34). However, the former is slightly modified by excluding both $y_{l_0i_01}$ and $y_{l_0j_01}$ from the sum on the left side of the expression. □

Finally, the remaining family exploits the same idea as Proposition 3.3.9 but for inequalities present in Proposition 3.3.7. Again, due to the similarity with the previous result, the details of the proof are not provided.

Proposition 3.3.10 (Family 4)**.** *For $i_0, j_0, l_0 = 1, \ldots, n$, $i_0 \neq j_0 \neq l_0$, inequalities*

$$y_{i_0j_0k_0} + y_{j_0i_0k_0} + y_{i_0l_0n-1} + y_{j_0l_0n-1} \leq y_{l_0i_0k_0-1} + y_{l_0j_0k_0-1} + \sum_{\substack{t=1 \\ t\neq k_0-1, \\ k_0,k_0+1}}^{n-1} \sum_{\substack{w=1 \\ w\neq i_0, \\ j_0,l_0}}^{n} y_{l_0wt} + y_{l_00n}$$

for $k_0 = 2, \ldots, n-3$, and

$$y_{i_0j_0n-2} + y_{j_0i_0n-2} + \sum_{\substack{w=1 \\ w\neq i_0, \\ j_0,l_0}}^{n} y_{wl_0n-1} \leq y_{l_0i_0n-3} + y_{l_0j_0n-3} + \sum_{t=1}^{n-4} \sum_{\substack{w=1 \\ w\neq i_0, \\ j_0,l_0}}^{n} y_{l_0wt} + y_{l_00n}$$

$$y_{i_0j_0n-1} + y_{j_0i_0n-1} \leq y_{l_0i_0n-2} + y_{l_0j_0n-2} + \sum_{t=1}^{n-3} \sum_{\substack{w=1 \\ w\neq i_0,j_0,l_0}}^{n} y_{l_0wt}$$

are valid for P^n_{TD}.

Regarding the separation problems for these families, since each of them is composed by a polynomial number of members, we explicitly enumerate them in order to find a violated cut.

3.4. B&C algorithm

In order to evaluate the strength of the inequalities introduced in Section 3.3, we develop a B&C algorithm considering the model (PQ). We focus mainly on the cutting planes and a primal heuristic leaving, among other parameters, the branching and the node selection strategies as

CPLEX's defaults. For notational convenience, sometimes we refer to variables x_{ik} from (VW) formulation, which value can be calculated in termes of variables y_{ijk}.

We observed that in some instances applying a simple preprocessing improved drastically the overall computing times. This preprocessing phase aims to eliminate feasible solutions for which we can assure that either they are not optimal or there exists an alternative optimal solution.

The main idea is quite simple and looks for identifying time dependent arcs which cannot be present in an optimal solution. Given $i, j \in V\backslash\{0\}$, $i < j$, $1 \leq k \leq n-2$, if for all $v, w \in V$, $v, w \neq i, j$, if

$$c_{vik-1} + c_{ijk} + c_{jwk+1} \leq c_{vjk-1} + c_{jik} + c_{iwk+1},$$

then we can fix variable $y_{jik} = 0$. Otherwise, we check whether swapping i and j the condition is satisfied to fix $y_{ijk} = 0$. It is important to remark the second test is applied only when the first one fails, since otherwise the optimal solution may be cut off.

For some particular cases of the TDTSP, such as the TSP and the TDP, this test can be executed without considering the particular position of the arc in the tour and, therefore, the implication is valid for all possible time periods k.

3.4.1. Primal heuristic

The use of heuristic procedures to obtain feasible integer solutions based on the information provided by the solution of the LP relaxation have been proven to be very effective to obtain good quality upper bounds. The main purpose is to reduce the number of nodes explored in the B&C tree, aiming to reduce the overall running time of the algorithm. However, it is important to find a reasonable tradeoff between the effectiveness of the algorithm and the computational effort.

In our algorithm we consider a primal heuristic consisting of two phases. The first one is a construction phase, which generates different feasible solutions based on a heuristic. The second phase consists on applying for each solution an improvement procedure. We give further details about each phase in the following sections.

Construction Phase

The greedy heuristic considered in this phase iteratively adds vertices to positions $0, 1, \ldots, n-1$ using the information of the fractional solution of the current node. In particular, in iteration k selects as the next vertex v to be added to the partial tour the one having the greatest value its corresponding assignment variable $y_{v_{\text{last}}vk}$, where v_{last} is the last vertex aded to the current partial path. In case of tie, we select the vertex with minimum cost for the corresponding arc.

In order to generate several possibilities, we execute this procedure considering each vertex $w \in V$ as the first one in the tour. The outline of the heuristic is presented in Algorithm 3.3

Algorithm 3.3 GREEDY CONSTRUCTIVE PRIMAL HEURISTIC

Input: $\tilde{z} = (\tilde{y}, \tilde{x}) \in P_{TDTSP}$.

1. for $w \in V$, **do:**

2. Set w as the first vertex in the tour and mark it as used. Set also $v_{\text{last}} = w$ and $k = 1$.

3. **for** $k = 1, \ldots, n-1$ **do:**

4. Set $v = \arg\max_{i \text{ not used}} \tilde{y}_{v_{\text{last}} ik}$. In case of more than one possibility, select the vertex having minimum cost $c_{v_{\text{last}} ik}$.

5. Set $k = k + 1$, $v_{\text{last}} = v$, mark v as used.

6. **end for**

7. end for

Improvement Phase

The aim of this phase is, given a feasible solution for the problem, to find a new one with a better total cost by applying a local search procedure. We consider two different improvements, which are executed sequentially. It is important to remark that this improvement phase is applied to all solutions generated in the construction phase.

First we consider a vertex interchange heuristic. Given $C =< 0, v_1, \ldots, v_n, 0 >$ a feasible tour, we define the neighbourhood of C, $N(C)$, as all the possible tours obtained by interchanging the positions of any two different vertices of C. From all of the tours in $N(C)$ that improve the cost of C, in case there is any, we choose as our new solution the one with the smallest cost. We apply this procedure iteratively until no improvement is achieved or a maximum of fifteen iterations is reached. Secondly, when this procedure is finished, we execute a 3-opt procedure (Lin [39]).

3.4.2. Cutting planes

In this section we specify the cutting plane algorithm considered. Based on restricted preliminary computational results, considering together inequalities (3.22) - (3.26) slows the resolution of the LP relaxations. This is due to their similarity, given that some of them share many of the variables involved in the inequality. As a consequence, more inequalities are added to the formulation but obtaining similar results in terms of improvements of the lower bound. We observed that the best results are produced considering together constraints (3.22) and (3.24).

In addition to the inequalities presented in Section 3.3 we included also the *Subtour Elimination Constraints* (SEC, see Section 2.3, constraints (2.6)). In order to include these constraints, variables z_{ij} can be defined as

$$z_{ij} = \sum_{k=1}^{n} y_{ijk},$$

indicating if we are travelling from vertex i to vertex j in one of time periods.[1] We consider adding more than one of these constraints instead of only the most violated one. At the root node we perform a maximum of fifteen iterations and in the internal nodes we only apply one round of cutting planes.

On preliminary computational results, we observed that the best improvements at the root node is produced by a combination of the SEC, the TDCI with $l = 2$ and the Lifted TDCI (3.22) and (3.24). For example, in some instances considering only SEC or TDCI with $l = 2$, constraints (3.22) and (3.24) the gap at the root node is reduced to nearly an 8% in both cases. However, when considered toghether, this value drops below 2%. For this reason, we include the three sets of constraints in the cutting plane algorithm.

The constraints considered in the cutting plane algorithm are enumerated below.

- SEC. At most 30 per round.
- TDCI with $l = 2$.
- Lifted TDCI (3.22) and (3.24). At most 100 per round.
- Families from Section 3.3.3. At most 100 for each of them per round.

3.5. Computational results

The experiments were conducted on a workstation with Intel Core i7 with 16 Gb of RAM running a Fedora Linux distribution. The algorithms are coded in C++ and combined with Ilog CPLEX 12.2 callable library for the optimization routines.

We use benchmark instances that are divided into three groups. The first group includes instances from TSPLIB. These instances are considered both as TSP instances (i.e., $c_{ijk} = c_{ij}$) and as TDP instances (i.e., $c_{ijk} = (n - k + 1)c_{ij}$). A second group regards randomly generated instances for TDP from Méndez-Díaz et al. [45]. Finally, the third group considers the instances proposed in Rubin and Ragatz [54] and considered also in Bigras et al. [13]. We slightly modify the original instances of the third group by discarding the corresponding due dates, which results in $1|s_{ij}|\sum C_j$ scheduling instances (equivalent to TDP).

As regards the methods evaluated, we consider the follwing ones:

- BC: the B&C algorithm described in Section 3.4 considering the PQ model. All CPLEX cuts and heuristics are disabled.
- BC-R: the B&C algorithm for the TDP proposed in Méndez-Díaz et al. [45]. This algorithm is based on a special formulation for the TDP, whose variables capture the cummulative nature of costs. It includes several facet defining inequalities as well as a primal heuristic, producing good and competitive computational results. Taking to account the benchmark instances tested, the results produced by this algorithm represents a good baseline for the

[1] To avoid confusions with variables x_{ik} from model (VW), we rename variables x_{ij} from Section 2.3 as z_{ij}. It is important to remark that variables z_{ij} are not included explicitly in the formulation (PQ).

evaluation of BC. Since we have access to the code, the computational experiments for this algorithm are carried out in the same environment described before.

- CPLEX: CPLEX's default algorithm considering the PQ model. It also includes the variable fixing phase explained in Section 3.4 and all CPLEX's general features. The comparison with this algorithm will show the benefits obtained by including special purpose features in BC.

For each algorithm we report the computational time (in seconds) and the number of tree nodes explored in the B&C algorithm. A cell filled with $(***)$ means that the instance was not solved within two hours by that algorithm. We also show the gaps at the root node (%rG) and at the end of the execution (%fG). The %rG is calculated as $100*(BESTUB-LB)/BESTUB$, where $BESTUB$ is the objective value of the best solution obtained considering all algorithms tested. For %fG we adopted a different criterion, and is computed as $100*(UB-LB)/UB$, where UB stands for the objective value of the best solution found by the algorithm under consideration. In this way, %fG gives us a measure of the progress made by the algorithm when the time limit is reached. For the BC-R algorithm we do not report results of TSP instances, since BC-R is specifically developed for the TDP.

3.5.1. Comparison of B&C algorithms

In tables 3.1 and 3.2 we show the computational times for the TSPLIB instances considered as TSP and TDP instances, respectively. One of the messages of this table is that our B&C algorithm outperforms CPLEX in all the instances considered, producing much better results and solving almost all instances considered. This lies on the fact that the inequalities incorporated to the cutting phase are quite effective, specially the combination of the TDCI of size 2, the Lifted TDCI and SECs. The best gains are obtained at the root node, where the gap with respect to the optimal solution is considerably reduced. The average of %rG over all TSP instances considered is 0.36% for BC while for CPLEX is 13.84%. For TDP instances, these values are 1.82% for BC and 15.85% for CPLEX. It is important to note that in both cases the %rG for TDP instances tend to be larger than for TSP ones. Furthermore, the number of instances solved at the root node by BC is smaller when considered as TDP.

The most interesting results are the ones regarding instances with 40 vertices or more. CPLEX algorithm is able to solve to optimality only 2 of the 26 instances (ftv44 TSP; ftv47 TDP) within two hours, while our BC solves 23 of them. This is also expressed in the number of nodes explored in the B&C tree, where CPLEX requires to visit an extremely higher number of them compared to BC.

Considering BC-R algorithm, it is able to solve 4 of the 13 TDP instances with more than 40 vertices, The computational times are considerable higher compared to the ones produced by BC. This can be explained by the fact that BC-R LP relaxations are difficult to solve and, even when they produce good lower bounds, the procedure of BC is more effective.

We now turn our attention to two specific instances for which computational times are affected by the preprocessing explained in Section 3.4. Instance br17, when considered as a TDP instance, without this variable fixing requires 209.24 seconds and 1549 nodes in the B&C tree

Type	Instance	n	BC				CPLEX			
			Time	Nodes	%rG	%fG	Time	Nodes	%rG	%fG
TSP	burma14	14	0,02	0	0,00	0,00	0,26	125	7,45	0,00
	ulysses16	16	0,07	0	0,00	0,00	0,83	364	10,95	0,00
	gr17	17	0,09	0	0,00	0,00	19,86	20001	13,25	0,00
	gr21	21	0,09	0	0,00	0,00	8,58	2040	6,75	0,00
	ulysses22	22	0,37	0	0,00	0,00	26,27	6128	16,05	0,00
	gr24	24	0,47	0	0,00	0,00	25,98	3496	10,68	0,00
	fri26	26	0,55	0	0,00	0,00	204,39	29943	9,14	0,00
	bayg29	29	1,01	0	0,00	0,00	609,94	54874	7,25	0,00
	bays29	29	1,83	0	0,00	0,00	1179,66	118294	8,69	0,00
	dantzig42	42	10,88	0	0,00	0,00	* * *	* * *	12,50	7,68
	swiss42	42	6,94	0	0,00	0,00	* * *	* * *	17,92	12,28
	att48	48	76,29	35	0,17	0,00	* * *	* * *	16,70	19,57
	gr48	48	* * *	* * *	1,68	0,33	* * *	* * *	15,47	21,56
	hk48	48	24,51	0	0,00	0,00	* * *	* * *	11,10	14,21
	eil51	51	1306,8	84	0,82	0,00	* * *	* * *	9,91	32,64
	berlin52	52	26,1	0	0,00	0,00	* * *	* * *	13,15	32,39
	brazil58	58	220,33	25	0,11	0,00	* * *	* * *	25,73	81,74
	br17	17	0,03	0	0,00	0,00	0,56	132	16,03	0,00
	ftv33	33	1,81	0	0,00	0,00	646,18	18158	6,80	0,00
	ftv35	35	30,99	39	0,82	0,00	269,58	11757	5,44	0,00
	ftv38	38	59,96	55	0,80	0,00	788,7	14998	5,28	0,00
	p43	43	5080,08	63	0,14	0,00	* * *	* * *	84,13	81,84
	ftv44	44	347,23	35	1,43	0,00	5383,61	48305	5,02	0,00
	ftv47	47	528,55	73	1,50	0,00	* * *	* * *	3,63	5,30
	ry48p	48	521,12	95	0,85	0,00	* * *	* * *	10,72	9,03
	ftv55	55	3164,68	85	1,28	0,00	* * *	* * *	10,17	19,62

Table 3.1.: Computational times (in seconds) and number of tree nodes explored for TSP instances from TSPLIB.

to be solved. In Table 3.2 we can appreciate that it is solved in less than a second at the root node. A similar observation can be done for p43, which cannot be solved within the time limit imposed without the preprocessing, neither in its TSP version nor in its TDP one. Although it is a quite restrictive condition, it shows to be very effective and produces significant changes in the overall computational times.

As regards the %fG, we can observe that BC produces the best results in the instances that cannot be solved within the time limit imposed. For TSP instances, BC is not able to solve only one instance and %fG is 0.33 %, while for CPLEX the average %fG over the unsolved instances is 28.15%. For TDP ones, the average %fG over the unsolved instances is 1.99% for BC, 14.61% for BC-R and 17.64% for CPLEX. Based on this values and the number of instances solved we can deduce that BC represents a more robust approach than BC-R and CPLEX. This is due to the combination between the improvements obtained by means of the cutting planes and the effectiveness of the primal heuristic in finding good feasible solutions early in the B&C tree.

The results for the instances from Méndez-Díaz et al. [45] are presented in Table 3.3. Each

Type	Instance	n	BC				BC-R				CPLEX			
			Time	Nodes	%rG	%fG	Time	Nodes	%rG	%fG	Time	Nodes	%rG	%fG
TDP	burma14	14	0,03	0	0,00	0,00	0,53	0	0,00	0,00	0,58	220	8,70	0,00
	ulysses16	16	0,14	0	0,00	0,00	6,83	7	1,74	0,00	0,87	273	17,95	0,00
	gr17	17	0,1	0	0,00	0,00	1,68	0	0,00	0,00	1,34	380	15,16	0,00
	gr21	21	0,37	0	0,00	0,00	2,98	0	0,00	0,00	14,8	4123	16,29	0,00
	ulysses22	22	4,65	33	4,18	0,00	129,93	32	4,27	0,00	20,33	4648	27,43	0,00
	gr24	24	0,48	0	0,00	0,00	4,54	0	0,00	0,00	22,89	3026	14,68	0,00
	fri26	26	2,74	7	0,71	0,00	47,36	7	1,97	0,00	164,91	21903	13,67	0,00
	bayg29	29	94,53	95	3,00	0,00	250,07	27	2,18	0,00	705,15	67719	12,80	0,00
	bays29	29	14,36	39	1,86	0,00	618,08	49	1,62	0,00	345,76	31839	13,77	0,00
	dantzig42	42	39,93	33	0,90	0,00	2278,71	41	2,30	0,00	* * *	* * *	19,10	11,33
	swiss42	42	30,33	13	1,20	0,00	1422,74	21	4,01	0,00	* * *	* * *	18,53	13,51
	att48	48	70,79	39	0,85	0,00	3377,31	33	4,30	0,00	* * *	* * *	16,12	7,89
	gr48	48	2014,95	407	3,99	0,00	* * *	* * *	6,84	8,32	* * *	* * *	20,66	17,70
	hk48	48	131,65	39	1,61	0,00	* * *	* * *	4,60	4,60	* * *	* * *	16,48	12,80
	eil51	51	90,55	25	0,71	0,00	* * *	* * *	1,94	4,14	* * *	* * *	14,75	19,86
	berlin52	52	* * *	* * *	3,54	1,01	* * *	* * *	13,45	14,30	* * *	* * *	21,83	23,71
	brazil58	58	* * *	* * *	8,77	2,96	* * *	* * *	17,70	19,15	* * *	* * *	21,83	36,87
	br17	17	0,04	0	0,00	0,00	295,08	171	12,45	0,00	0,67	16	7,32	0,00
	ftv33	33	22,68	27	1,57	0,00	887,52	47	2,40	0,00	807,67	41585	11,62	0,00
	ftv35	35	23,57	19	1,34	0,00	778,47	38	1,59	0,00	1645,47	72258	7,87	0,00
	ftv38	38	41,74	23	1,38	0,00	341,31	11	2,06	0,00	1426,38	36485	8,68	0,00
	p43	43	3241,23	691	4,54	0,00	* * *	* * *	62,02	62,14	* * *	* * *	50,27	42,39
	ftv44	44	449,35	117	3,54	0,00	* * *	* * *	4,44	5,25	* * *	* * *	9,86	7,19
	ftv47	47	41,27	13	1,52	0,00	* * *	* * *	3,20	6,25	703,7	4600	6,44	0,00
	ry48p	48	39,96	7	0,52	0,00	2550,81	27	3,54	0,00	* * *	* * *	8,03	6,28
	ftv55	55	156,89	33	1,48	0,00	* * *	* * *	7,33	7,33	* * *	* * *	12,43	12,18

Table 3.2.: Computational times (in seconds) and number of tree nodes explored for TDP instances from TSPLIB.

row shows the average value for the computational time, number of nodes explored and gaps over five instances. If present, a number between parenthesis indicates how many of the five instances are actually solved to optimality within two hours and the value in the cell represents the average considered only these instances.

In general, results are similar to the ones for the TSPLIB instances. The BC algorithm produces considerable reductions in the computational times, and these differences become larger as the number of vertices of the instances increase. Both BC and BC-R are capable of solving all instances, in general enumerating a small number of nodes in the B&C tree. CPLEX begins to fail in symmetric and euclidean instances with 35 and 40 vertices. The gaps at the root node for BC-R are very good in general, but the algorithm is more time consuming than BC because of the time required to solve each LP relaxation. As expected, CPLEX enumerates a great number of nodes in the B&C tree, showing the benefit of the cutting plane algorithm of BC.

An interesting observation from the results observed in this table regards the difficulty of the instances depending on its type. Both BC and BC-R are able to solve asymmetric and symmetric instances within a reasonable time, and for each algorithm the times are comparable. However, we can observe an increment on the times as well as in the number of nodes explored for euclidean instances, which seem to be harder to solve than the ones mentioned previously.

In Table 3.4 we show the average computational times for the scheduling instances from Rubin and Ragatz [54]. The average value is calculated over eight instances for each $n = 15, 25, 35, 45$. These results are aligned with the ones from the previous tables. Our B&C performs better than CPLEX's default algorithm, both in the computational times and the number of nodes explored. It is important to note that this difference is significantly higher when $n = 45$, since the other instances are easily solved by both methods. Again, this behaviour is due to the strengthening of the bounds produced by the cutting phase and the primal heuristic.

Finally, we compare our results with the ones reported in Bigras et al. [13]. As mentioned in the introduction of this chapter, the authors study the path formulation proposed in Picard and Queyranne [53]. The formulations has an exponential number of variables and therefore they consider a B&P algorithm. For the pricing phase, they consider adding paths (columns) without 4 cycles. They also include several families of valid inequalities of the TSP and clique inequalities obtained by constructing the conflict graph using incompatibilites inferred for the TDTSP.

The authors report computational times in all cases, exceeding our time limit of two hours in four instances. It is important to remark that the computer used by Bigras et al. dates from 2008. For this reason, we made the assumption that instances in which runtimes for BC are 600 seconds or less are easy and therefore comparably fast to be solved by both algorithm, given that the relationship between computational times for both algorithms is linear (corr coef = 95%). Using this information, we calculate a linear least squares regression and obtained that our computer has a speedup of nearly 5.2 compared to theirs. Thus, we report $(***)$ for their computing times whenever they exceed $5.2 * 7200 = 37440$.[2] Besides this, we can observe some interesting results in the comparison.

[2]It is important to remark that this estimation implicitly assumes that there are mainly no differences between both algorithms. However, we consider this simplification in order to contemplate the difference in the computers for establishing the time limit in their computational times.

Inst.	n	BC				BC-R				CPLEX			
		Time	Nodes	%rG	%fG	Time	Nodes	%rG	%fG	Time	Nodes	%rG	%fG
Asymmetric	20	0,05	0,00	0,00	0,00	0,68	0,00	0,00	0,00	0,20	0,00	0,00	0,00
	22	0,08	0,00	0,00	0,00	0,84	0,00	0,00	0,00	0,48	10,80	0,41	0,00
	24	0,13	0,00	0,00	0,00	2,38	0,60	0,15	0,00	0,75	21,20	2,01	0,00
	26	0,13	0,00	0,00	0,00	2,11	0,00	0,00	0,00	0,74	4,20	0,65	0,00
	28	0,44	0,60	0,08	0,00	14,54	6,40	0,47	0,00	2,35	25,80	1,70	0,00
	30	0,98	2,80	0,50	0,00	34,09	8,00	0,58	0,00	4,30	104,20	3,25	0,00
	35	0,99	0,60	0,12	0,00	21,24	0,80	0,10	0,00	4,28	7,20	0,55	0,00
	40	8,17	10,00	1,22	0,00	489,50	25,00	1,90	0,00	53,42	952,00	5,48	0,00
Symmetric	20	0,15	0,00	0,00	0,00	1,39	0,00	0,00	0,00	3,28	519,40	18,77	0,00
	22	0,28	0,00	0,00	0,00	2,56	0,00	0,00	0,00	8,08	1385,60	17,12	0,00
	24	0,47	0,00	0,00	0,00	7,05	1,00	0,11	0,00	23,94	2943,00	27,27	0,00
	26	0,58	0,00	0,00	0,00	6,82	1,00	0,23	0,00	63,33	6077,20	23,45	0,00
	28	1,00	0,00	0,00	0,00	12,21	0,60	0,04	0,00	113,60	9563,40	26,04	0,00
	30	1,46	0,00	0,00	0,00	24,94	1,00	0,22	0,00	466,21	32789,00	27,55	0,00
	35	3,67	0,00	0,00	0,00	57,52	0,60	0,07	0,00	(4) 1242,66	47799	34,61	10,05
	40	9,58	2,00	0,19	0,00	391,15	11,00	1,35	0,00	(1) 6749,43	158253	34,00	19,77
Euclidean	20	0,15	0,00	0,00	0,00	1,27	0,00	0,00	0,00	3,79	526,80	10,47	0,00
	22	0,44	0,00	0,00	0,00	6,75	2,80	0,51	0,00	9,90	1199,80	13,32	0,00
	24	0,78	0,00	0,00	0,00	10,44	2,00	0,60	0,00	34,84	3689,00	16,12	0,00
	26	1,19	1,00	0,13	0,00	19,31	4,40	0,26	0,00	61,93	5772,00	13,37	0,00
	28	5,02	10,80	0,69	0,00	37,17	5,60	0,67	0,00	233,05	17746,20	18,25	0,00
	30	4,01	2,60	0,22	0,00	133,51	14,20	0,64	0,00	234,90	12049,60	16,24	0,00
	35	8,53	3,20	0,32	0,00	139,52	6,20	0,53	0,00	(4) 3164,09	112825,75	20,05	7,74
	40	44,35	27,80	1,50	0,00	1421,73	33,80	1,87	0,00	(1) 4984,88	90391	21,62	10,75

Table 3.3.: Average computational times (in seconds) and number of tree nodes explored for Méndez-Díaz et al. instances.

Instances	n	BC			CPLEX		
		Time	Nodes	%rG	Time	Nodes	%rG
PROB40x.TXT	15	0,02	0,00	0,00	0,23	2,88	0,04
PROB50x.TXT	25	0,15	0,00	0,00	1,06	21,75	0,06
PROB60x.TXT	35	1,68	4,75	0,02	9,24	80,00	0,08
PROB70x.TXT	45	10,35	7,00	0,02	104,56	923,63	0,09

Table 3.4.: Average computational times (in seconds) for scheduling instances.

Instance	BC		Bigras et al.		BC-R		CPLEX	
	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes
gr17	0,1	0	3	1	1,68	0	1,34	380
gr21	0,37	0	10	1	2,98	0	14,80	4123
gr24	0,48	0	15	1	4,54	0	22,89	3026
bays29	14,36	39	76	16	618,08	49	345,76	31839
bayg29	94,53	95	191	51	250,07	27	705,15	67719
rbg016a	0,03	0	15	1	1,01	0	0,2	0
rbg031a	1,29	0	90	1	279,37	5	21,31	290
rbg050b	620,46	602	37342	39	* * *	* * *	1239,92	16125
dTSP40.0	3571,54	1119	6473	377	* * *	* * *	* * *	* * *
dTSP40.1	224,94	95	1452	50	* * *	* * *	* * *	* * *
dTSP40.2	342,23	125	1068	28	5662,62	127	* * *	* * *
dTSP40.3	6410,05	1431	629	24	* * *	* * *	* * *	* * *
dTSP40.4	26,83	29	299	4	1184,86	30	* * *	* * *
dTSP50.0	363,57	91	1364	5	* * *	* * *	* * *	* * *
dTSP50.1	3088,25	499	* * *	* * *	* * *	* * *	* * *	* * *
dTSP50.2	594,8	205	3668	8	* * *	* * *	* * *	* * *
dTSP50.3	* * *	* * *	* * *	* * *	* * *	* * *	* * *	* * *
dTSP50.4	* * *	* * *	* * *	* * *	* * *	* * *	* * *	* * *

Table 3.5.: Comparison of computational times (in seconds) with Bigras et al. [13]

First we note that BC solves more instances than all the other methods. In some cases, computational times obtained by Bigras et al. are reduced by BC to a 10% of the time. Furthermore, computational times reported by the authors for instances dTSP50.1, dTSP50.3 and dTSP50.4 are 56240, 47771 and 109586 seconds, respectively. Regarding BC, it can solve dTSP50.1 in less than an hour. It is also intersting the reduction obtained for instance rbg050b, where BC requires approximately 10 minutes to solve it. Despite the difference on the computers, the reductions on the computational times are also due to the inclusion of specific purpose cuts and the primal heuristic. On the other hand, for the instance dTSP40.3 their algorithm performs much better than BC, which is able to find the optimal solution but the initial gap is large and encounters difficulties to close it. This is related to the nature of their approach, which can also be observed in the fact that their B&P algorithm tends to enumerate less nodes than BC. As

regards BC-R and CPLEX, the behaviour is similar to the previous experiments.

3.5.2. Impact of the primal heuristic

In this section we analyze the impact of the primal heuristic described in Section 3.4. Tables 3.6 and 3.7 show the computational times and the number of nodes explored of BC with the primal heuristic and without it (BC-NPH) on the TSPLIB instances considered.

Type	Instance	n	BC		BC-NPH	
			Time	Nodes	Time	Nodes
TSP	burma14	14	0,02	0	0,02	0
	ulysses16	16	0,07	0	0,08	0
	gr17	17	0,09	0	0,1	0
	gr21	21	0,09	0	0,11	0
	ulysses22	22	0,37	0	0,46	0
	gr24	24	0,47	0	0,95	0
	fri26	26	0,55	0	0,78	0
	bayg29	29	1,01	0	2,55	0
	bays29	29	1,83	0	3,32	0
	dantzig42	42	10,88	0	61,34	9
	swiss42	42	6,94	0	16,57	0
	att48	48	76,29	35	215,2	41
	gr48	48	* * *	* * *	* * *	* * *
	hk48	48	24,51	0	98,1	0
	eil51	51	1306,8	84	* * *	* * *
	berlin52	52	26,1	0	35,97	0
	brazil58	58	220,33	25	419,58	25
	br17	17	0,03	0	0,06	0
	ftv33	33	1,81	0	2,3	0
	ftv35	35	30,99	39	120,05	43
	ftv38	38	59.96	55	294,4	52
	p43	43	5080,08	63	* * *	* * *
	ftv44	44	347,23	35	2077,1	47
	ftv47	47	528,55	73	2396,92	69
	ry48p	48	521,12	95	1399	93
	ftv55	55	3164,68	85	* * *	* * *

Table 3.6.: Evaluation of the primal heuristic (TSP instances).

For TSP instances, we can observe that BC produces in all cases considerably smaller computational times than BC-NPH. For instances solved at the root node, this reduction is because BC requires less iterations of cutting planes since the optimal solution is found by the primal heuristic. Moreover, the number of nodes explored by BC is reduced in the remaining cases.

Regarding TDP instances, the reductions in the computational times are similar. However, we can appreciate that, even when the computational times are smaller, in some cases BC explores a slightly larger number of nodes than BC-NPH. We illustrate this situation analyzing in detail instance bayg29. The remaining cases are similar.

Type	Instance	n	BC		BC-NPH	
			Time	Nodes	Time	Nodes
TDP	burma14	14	0,03	0	0,03	0
	ulysses16	16	0,14	0	0,3	5
	gr17	17	0,1	0	0,13	0
	gr21	21	0,37	0	0,47	0
	ulysses22	22	4,65	33	9,38	61
	gr24	24	0,48	0	0,66	0
	fri26	26	2,74	7	3,97	7
	bayg29	29	94,53	95	181,57	89
	bays29	29	14,36	39	23,07	33
	dantzig42	42	39,93	33	106,54	31
	swiss42	42	30,33	13	86,19	11
	att48	48	70,79	39	218,62	39
	gr48	48	2014,95	407	4648,02	359
	hk48	48	131,65	39	373,05	35
	eil51	51	90,55	25	286.19	19
	berlin52	52	* * *	* * *	* * *	* * *
	brazil58	58	* * *	* * *	* * *	* * *
	br17	17	0,04	0	0,06	0
	ftv33	33	22,68	27	31,97	29
	ftv35	35	23,57	19	49,24	19
	ftv38	38	41,74	23	101,16	17
	p43	43	3241,23	691	5801,13	833
	ftv44	44	449,35	117	1840,65	247
	ftv47	47	41,27	13	111,62	13
	ry48p	48	39,96	7	114,79	9
	ftv55	55	156,89	33	768,1	29

Table 3.7.: Evaluation of the primal heuristic (TDP instances).

In this instance, BC requires half the time compared to BC-NPH although it explores 6 nodes more. Observing in detail the log files of both BC and BC-NPH, we observed that even when it explores more nodes the number of LP relaxations solved is smaller. As regards BC, the optimal solution is found at the root node. In the B&C tree, all nodes remain open until node 26. From node 26 to node 95, 44 of them are pruned solving only the first LP relaxation (recall that in the internal nodes the maximum number of rounds is 1).

Considering BC-NPH, the situation is quite different. The optimal solution is found at node 71. Until then, all nodes remain open (up to node 70, there are 68 nodes open, since it finds two other feasible solutions during the enumeration). The remaining nodes are pruned solving only the first LP relaxation of the node. For each of the first 70 nodes (with the exception of the root node and the two in which an feasible integral solution is found), BC-NPH solves 2 LP relaxations and executes the cutting plane algorithm, causing the increment in the overall computational times of the algorithms. Furthermore, due to the difference in the cutting planes included in the formulations, LP relaxations may become harder to solve, specially when reoptimzing.

This shows the benefits produced by the primal heuristic, reducing considerably the overall

computational time of the algorithm in all cases.

3.6. Conclusions

In this chapter we consider the TDTSP formulations of Picard and Queyranne [53] and Vander Wiel and Sahinidis [61]. We analyze both polytopes and derive several families of valid inequalities for both models. We generalize the idea of the well-known cycle inequalities for the ATSP, and derive five families of facets by applying a lifting procedure. We develop a B&C algorithm in order to evaluate these inequalities which, together with a primal heuristic, prove to be very effective. The overall approach produces good computational results over different benchmark instances with respect to CPLEX's default algorithm and the B&C algorithm from Méndez-Díaz et al. [45].

As future research, it would be interesting to analyze the complexity of the separation problems for the lifted TDCI in order to improve the separation routines implemented so far. Regarding the polyhedral study performed in this chapter, although we do not have a proof, we conjecture that valid inequalities proposed in Section 3.3.3 are facet defining. More work could be done in this direction.

Investigating how to improve the computational times required to solve the LP relaxations of BC, by means of studying particular characteristics of the formulation (PQ), could lead to significant reductions in the overall computational times, specially for instances with large values of n. In addition, as mentioned in the computational experiments, the preprocessing proposed showed to be very effective on instances satisfying the condition. It is worth to investigate further this aspect of the problem to derive new rules to fix variables in the model.

4. The TDTSP with time dependent travel times and time windows

4.1. Introduction

The *Time-Dependent TSP with Time Windows* (TDTSP-TW) is a generalization of the classical TSP with Time Windows (TSPTW) in which the cost of the travel between two cities depends on the time of the day the arc is travelled.

The problem studied in this chapter can be stated as follows: Consider a complete digraph $D = (V, A)$, with $V = \{0, 1, \ldots, n, n+1\}$ the set of vertices and A the set of arcs. Vertices 0 and $n+1$ represent the depot, for which we do not consider the incoming and outcoming arcs, respectively. There is a discrete time horizon $0, 1, \ldots, T$ in which the vehicle moves along the network. We assume that, for each arc $(i, j) \in A$, the travel time function is discretized into M different time periods, and that it is constant (and integer) within each of them. Therefore, the problem is formulated on an expanded network where we replace each arc $(i, j) \in A$ by M parallel links going from i to j, one for each possible time period. Time period m for arc $(i, j) \in A$ is bounded by $(T_{ij}^{m-1}, T_{ij}^{m}]$, for $m = 1, \ldots, M$, where $T_{ij}^{0} = 0$. We name c_{ij}^{m} the travel time and θ_{ij}^{m} the travel cost when travelling from i to j starting at i in time period m. For each vertex $i \in V$, integer values p_i represent its processing time and $W_i = [r_i, d_i]$ the corresponding time window, where r_i and d_i are the release and deadline times, respectively. The TDTSP-TW involves finding a minimum cost tour, starting from vertex 0 and ending at vertex $n+1$, that starts processing each vertex within its time window exactly once. It is important to remark that waiting times at both the arrival and the departure of each vertex are allowed. As regards the waiting at the departure, following the related literature, the vehicle must depart at the latest at $d_i + p_i$. Due to the nature of the problem, we do not assume that the triangle inequality holds on the travel times.

This version of the problem is proposed in Malandraki and Daskin [42] with an objective function that minimizes the makespan of the route. They propose an MILP formulation for the problem and develop a nearest neighbour heuristic. To model the time-dependency of the travel function they use big M-like inequalities, known by their poor performance in terms of tightness of the linear relaxations. Indeed, they report large gaps between the best feasible solution obtained and the value of the LP relaxation of a cutting plane algorithm. In a follow up paper, Malandraki and Dial [43] develop a restricted dynamic programming heuristic and apply it to instances with up to 55 vertices, obtaining near optimal solutions. In the latter, neither waiting times at the vertices nor time windows are considered.

This version of TDTSP is also considered in Stecco, Cordeau and Moretti [57] in order to solve a Sequence and Time Dependent Scheduling Problem (STDSP) by allowing waiting

times at the vertices. Based on the model from Malandraki and Daskin [42], Stecco et al. propose a new formulation of the STDSP considering a particular travel time function from a real-world problem and also with the objective of minimizing the makespan. They prove that this formulation has stronger LP relaxation bounds and perform a polyhedral study to develop a B&C algorithm. Computational results show instances with up to 35 vertices solved to optimality.

A different approach is proposed in Albiach et al. [3]. By applying a succession of transformations, they reduce the TDTSP-TW into a Graphical Asymmetric TSP (GATSP) and apply known exact algorithms of the Mixed General Routing Problem. With this framework they are able to solve to optimality instances with up to 60 vertices. However, due to the nature of the transformation, all these instances have narrow time windows (with lengths not exceeding one hour), which might be crucial regarding the performance of the algorithms and the number of variables in the corresponding models. In a follow up paper, Soler et al. [56] extend the problem to the multiple-vehicle case where they allow waiting times at the departure. No computational results are reported. In a recent research, Persiani [52] addresses an aviation problem which is a special case of the TDTSP.

To the best of our knowledge, these are the only exact approaches in the related literature regarding the general case of the TDTSP and TDTSP-TW. In a general analysis, the model from Malandraki and Daskin [42] generates weaker lower bounds when minimizing the makespan. In addition, although they prove that the relaxations are stronger, the model from Stecco et al. [57] is strongly related to the particular time-dependent transition function considered. Finally, the approach proposed by Albiach et al. [3] does not allow waiting times at the departure from a vertex.

In this chapter, we propose two formulations for the TDTSP-TW. We generalize the idea of *infeasible paths* from the classical TSPTW [5, 6], which we name *time-dependent infeasible paths*. One of the formulations is based on this idea. The second one is built upon a very interesting and recent article by Dash et al. [19] that proposes a new formulation for the TSPTW based on the idea of *time buckets*, which shows to be very effective in solving TSPTW benchmark instances, producing good quality LP relaxations. We generalize this approach to the time-dependent case. We compare and evaluate these formulations with adapted versions of the ones from Malandraki and Daskin [42] and Stecco et al. [57] on instances generated from TSPTW benchmark ones. Preliminary computational results show that the formulation based on the approach from Dash et al. produces the best results. For this reason, we provide several families of valid inequalities for the TDTSP-TW which are included in a B&C algorithm, together with some of the valid inequalities proposed by Dash et al. [19]. We conducted further experiments to evaluate and determine different parameters of the algorithm, obtaining good computational results and showing that the overall approach is proven effective.

The chapter is organized as follows. In Section 4.1.1 we present the formulations of Malandraki and Daskin [42] and Stecco et al. [57]. In Section 4.2 we present the generalization of infeasible paths to the time dependent case and the two formulations proposed for the TDTSP-TW using this idea. Section 4.3 we show the valid inequalities derived for the TDTSP-TW and Section 4.4 explains the details of the B&C algorithm developed. Computational experiments are reported in Section 4.5 and finally we show some conclusions for the chapter in Section 4.6.

4.1.1. Exact approaches for the TDTSP and TDTSP-TW

Malandraki and Daskin

As we have mentioned in the introduction, the TDVRP is introduced in Malandraki and Daskin [42]. The authors propose a very general MILP model to solve it, considering constraints such as time windows and capacity restrictions. The objective function minimizes the total makespan of the routes. For the sake of simplicity, we will show in this section a slightly modified version of the formulation proposed in Stecco et al. [57] -with fewer constraints and the same value of the linear relaxation when minimizing the makespan of the tour- for the single vehicle case and only considering time windows.

This formulation considers two sets of variables which are the natural time-dependent extension of the classical TSPTW variables. First, let binary variable x_{ij}^m take value 1 if and only if we are travelling from vertex i to vertex j in time period $(T_{ij}^{m-1}, T_{ij}^m]$. In addition, to account for the departure times, there exist real-valued variables t_i, for $i = 0, \ldots, n+1$, which represent the time at which the vehicle departs from vertex i. In the formulation shown below, M_1 stands for a constant value which is assumed to be big enough.

$$\min \quad t_{n+1} \tag{4.1}$$

s.t.

$$\sum_{i=0}^{n} \sum_{m=1}^{M} x_{ij}^m = 1 \qquad \forall j \in V \backslash \{0\} \tag{4.2}$$

$$\sum_{j=1}^{n+1} \sum_{m=1}^{M} x_{ij}^m = 1 \qquad \forall i \in V \backslash \{n+1\} \tag{4.3}$$

$$t_j - t_i - M_1 \sum_{m=1}^{M} x_{ij}^m \geq \sum_{m=1}^{M} (c_{ij}^m + p_j) x_{ij}^m - M_1 \quad \forall i \in V \backslash \{n+1\},\ j \in V \backslash \{0\} \tag{4.4}$$

$$t_i \leq \sum_{j=1}^{n+1} \sum_{m=1}^{M} T_{ij}^m x_{ij}^m \qquad \forall\ i \in V \backslash \{n+1\} \tag{4.5}$$

$$t_i \geq \sum_{j=1}^{n+1} \sum_{m=1}^{M} (T_{ij}^{m-1} + 1) x_{ij}^m \qquad \forall\ i \in V \backslash \{n+1\} \tag{4.6}$$

$$r_i + p_i \leq t_i \leq d_i + p_i \qquad \forall\ i\ \in\ V \backslash \{n+1\} \tag{4.7}$$

$$x_{ij}^m \in \{0,1\}, t_i \geq 0 \tag{4.8}$$

The objective function (4.1) minimizes the departure time from vertex $n+1$, which represents the final depot and therefore, the makespan of the route. Equations (4.2) and (4.3) establish that exactly one arc enters vertices $i \in V \backslash \{0\}$ and that exactly one arc leaves vertices $i \in V \backslash \{n+1\}$. Inequalities (4.4) relate the departure time from vertex $i \in V \backslash \{n+1\}$, t_i, with the departure time of vertex $j \in V \backslash \{0\}$, t_j, if travelling from i to j during time period m, for $1 \leq m \leq M$.

Inequalities (4.5), (4.6) and (4.7) consider that the departure time from any vertex is within the selected time period and that each vertex is visited within its corresponding time window, respectively.

In our problem, instead of minimizing the makespan, we are considering the minimization of the total cost of the tour. Thus, in order to use this model for our problem, we replace the objective function (4.1) by

$$z_{\text{cost}} = \sum_{i=0}^{n} \sum_{j=1, j\neq i}^{n+1} \sum_{m=1}^{M} \theta_{ij}^{m} x_{ij}^{m}, \tag{4.9}$$

leaving the rest of the model without changes.

Stecco, Cordeau and Moretti

In Stecco et al. [57] the authors study a scheduling problem with sequence and time-dependent setup times from the plastic container manufacturer industry, which they name STDSP. In this problem, the transition between two jobs requires two different activities. One of them, called *unrestricted setup*, can be performed at any time. The other activity, the *restricted setup*, should be executed outside a given time interval. The objective is to find a jobs' schedule minimizing the makespan.

The planning horizon considered is composed by several periods (e.g., days), where the forbidden interval occurs in the same relative position, giving a sort of periodicity to the time-dependent transition time function. To tackle this problem, the authors formulate the STDSP as a TDTSP and propose an MILP formulation that takes advantage of the particular characteristics of the time-dependent travel function and develop a B&C algorithm.

Formally, V represents the set of vertices (jobs) to be visited (scheduled), where 0 and $n+1$ are two dummy vertices indicating the beginning and the end of the tour. Let also $P = \{0, \ldots, n\}$ and $S = \{1, \ldots, n+1\}$ be the sets of possible predecessors and successors, respectively. Each vertex $j \in S$ has a processing time p_j, with $p_{n+1} = 0$. Let r_{ij} be the restricted setup time when processing vertex j immediately after vertex i, and u_{ij} the corresponding unrestricted setup time. Let also d denote the length of a planning period and $[a, d]$ the forbidden interval during which the restricted setup cannot be performed.

For $j \in V$, let t_j be a fractional decision variable representing the completion time of vertex j. By allowing waiting times, the time-dependent travel time function $c_{ij}(t_i)$ for arc (i, j) can be defined on interval $[0, d]$ as

$$c_{ij}(t_i) = \begin{cases} r_{ij} + u_{ij} & \text{if } t_i \in [0, a - r_{ij}], \\ r_{ij} + u_{ij} + d - a & \text{if } t_i \in (a - r_{ij}, d - u_{ij}), \\ r_{ij} + u_{ij} & \text{if } t_i \in [d - u_{ij}, d]. \end{cases}$$

Thus, period $[0, d]$ is divided into three disjoint intervals where the travel time is constant within each of them. Given that this pattern is repeated in successive periods, i.e., $[d, 2d], [2d, 3d]$, etc., Stecco et al. propose a refined formulation based on the one from Malandraky and Daskin [42] which exploits the structure of the objective function.

First, let K be the number of different intervals within a period, say $[0, d]$, I_{ij}^k the upper bound of the time interval $1 \leq k \leq K$ and c_{ij}^k the travel time from vertex i to vertex j if starting at i during interval k. Similarly to the model from the previous section, Stecco et al. define binary variables x_{ij}^k that take value 1 if and only if vertex j is visited immediately after vertex i during interval k and continuous variables t_i that represent the departure time from vertex i.

In order to account for the periodicity of the objective function they introduce a new set of integer variables h_i, $i \in V$, that represent the number of the period in which the departure from vertex i takes place. Furthermore, to improve the bound provided by the model's LP relaxation, they decompose variables t_j and h_j, $j \in V$, as

$$t_j = \sum_{i \in P} t_{ij} \qquad \text{and} \qquad h_j = \sum_{i \in P} h_{ij}, \tag{4.10}$$

where

$$t_{ij} = \begin{cases} t_j & \text{if } j \text{ is visited right after } i \\ 0 & \text{otherwise} \end{cases} \qquad \text{and} \qquad h_{ij} = \begin{cases} h_j & \text{if } j \text{ is visited right after } i \\ 0 & \text{otherwise.} \end{cases} \tag{4.11}$$

With these definitions, we show below the final formulation proposed in Stecco et al. [57]. The objective function (4.12) minimizes the makespan of the tour. Constraints (4.13) and (4.14) establish that each vertex must be visited exactly once. Constraints (4.15) and (4.16) compute the departure time from vertex j if visited after vertex $i \in S$ or after vertex 0, respectively. Constraints (4.17) and (4.19) force that the correct interval k is selected when travelling from i to j according to the departure time from vertex i. A similar condition ensures constraints (4.18) and (4.20) but when travelling from vertex 0 to a vertex $j \in S$. Constraints (4.21) and (4.22) ensure that condition (4.11) over variables t_{ij} and h_{ij} is satisfied. Finally, (4.23) defines the domain of all the variables present in the model.

$$\min \sum_{i=1}^{n} t_{i,n+1} \tag{4.12}$$

s.t.:

$$\sum_{i \in P} \sum_{k \in K} x_{ij}^k = 1 \qquad \forall j \in S \tag{4.13}$$

$$\sum_{j \in S} \sum_{k \in K} x_{ij}^k = 1 \qquad \forall i \in P \tag{4.14}$$

$$\sum_{j \in S} t_{ij} \geq \sum_{l \in P} t_{li} + \sum_{j \in S} \sum_{k \in K} (c_{ij}^k + p_j) x_{ij}^k \qquad i = 1, \ldots, n \tag{4.15}$$

$$\sum_{j=1}^{n} t_{0j} \geq t_0 + \sum_{j=1}^{n} \sum_{k \in K} (c_{0j}^k + p_j) x_{0j}^k \tag{4.16}$$

$$\sum_{l \in P} t_{li} \geq \sum_{j \in S} \sum_{k \in K} (I_{ij}^{k-1} + 1) x_{ij}^k + d \sum_{m \in P} h_{mi} \qquad i = 1, \ldots, n \tag{4.17}$$

$$t_0 \geq \sum_{j=1}^{n} \sum_{k \in K} (I_{0j}^{k-1} + 1) x_{0j}^k \tag{4.18}$$

$$\sum_{l \in S} t_{li} \leq \sum_{j \in S} \sum_{k \in K} I_{ij}^k x_{ij}^k + d \sum_{m \in S} h_{mi} \qquad i = 1, \ldots, n \tag{4.19}$$

$$t_0 \leq \sum_{j=1}^{n} \sum_{k \in K} I_{0j}^k x_{0j}^k \tag{4.20}$$

$$t_{ij} \leq M_1 \sum_{k \in K} x_{ij}^k \qquad \forall i \in P, j \in S \tag{4.21}$$

$$h_{ij} \leq M_2 \sum_{k \in K} x_{ij}^k \qquad \forall i \in P, j \in S \tag{4.22}$$

$$h_{ij} \in \mathbb{N}_0, \; x_{ij}^k \in \{0, 1\}, \; t_{ij} \geq 0 \tag{4.23}$$

This formulation can be easily adapted to the case where the travel time function is not periodic by considering a unique period with all the relevant intervals. This is achieved by setting $K = M$ and $I_{ij}^m = T_{ij}^m$, for $1 \leq m \leq M$ and $(i, j) \in A$.

As well as with the model from Malandraki and Daskin [42], in order to minimize the total travel cost of the tour, we replace the objective function (4.12) by the objective function defined in (4.9). Furthermore, considering the definitions of variables t_j in (4.10), we can incorporate time windows in a similar way than the model proposed in Malandraki and Daskin [42] using an adapted version of inequalities (4.7).

4.2. New ILP models

To account for the time dependency factor, the formulations proposed by Malandraki and Daskin [42] and Stecco et al. [57] include big M-like constraints. These type of constraints, in general, produce weak LP relaxations and suffer from numerical instability.

Aiming to avoid these problems, we present in this section two different models based on the idea of *infeasible paths* proposed by Ascheuer et al. [5, 6]. For this purpose, we first generalize the definition of infeasible paths to the time-dependent case. Then, based on this generalization, we present two new formulations for the TDTSP-TW.

4.2.1. Time dependent infeasible paths

In order to define a new model for the TDTSP-TW, we start adapting and generalizing to the time-dependent case the definitions from Ascheuer et al. [5, 6] regarding infeasible paths for the TSPTW. In this case, the idea behind infeasible paths is to model implicitly time windows constraints forbidding paths which are infeasible by the so-called *infeasible path constraints*. For a Hamiltonian path, checking if it is infeasible can be done easily. However, for partial paths this is an $\mathcal{NP}$-Complete problem and in general sufficient -and easy to check- conditions are considered.

In the TDTSP-TW, for a particular arc (i, j) its travel time is c_{ij}^m only if the vehicle departs from vertex i to vertex j between T_{ij}^{m-1} and T_{ij}^m. The bounds for time period m and arc (i, j), $(T_{ij}^{m-1}, T_{ij}^m]$, act as a sort of time window for the departure from vertex i. Therefore, for each combination between an arc leaving vertex i and a time period $1 \leq m \leq M$ there is an associated "time window" which is active only if vertex j is visited right after leaving vertex i in period m.

The definitions presented in Ascheuer et al. [5, 6] are slightly modified due to the nature of our problem, which also focus on the departure time from a vertex to select the appropriate time period.

Let $P = (v_1, v_2, \ldots, v_k)$ be a simple path and $T = (m_1, m_2, \ldots, m_{k-1})$ the corresponding sequence of time periods. We define a *time dependent simple path* as a combination between P and T, $[P, T]$, such that arc (v_i, v_{i+1}) is travelled in time period m_i, for $i = 1, \ldots, k-1$. Let t_{v_i} and s_{v_i} be, respectively, the *earliest* departure and arrival time for vertex v_i given $[P, T]$. We next show how to generalize the definition of infeasible paths to the time dependent case in two different situations: with and without considering time windows. Recall that input data are assumed to be integer.

1. **Time windows not allowed:** given a time dependent simple path $[P, T]$, the *earliest departure time* t_{v_i} for vertex v_i over $[P, T]$ is computed by

$$\begin{aligned} t_{v_1} &= T_{v_1 v_2}^{m_1 - 1} + 1 \\ t_{v_i} &= \max\{T_{v_i v_{i+1}}^{m_i - 1} + 1, t_{v_{i-1}} + c_{v_{i-1} v_i}^{m_{i-1}} + p_{v_i}\}, \quad \text{for } i = 2, \ldots, k-1. \end{aligned}$$

   This formula considers waiting times at the departure from a vertex.

   A Hamiltonian time dependent path $[P, T]$, where $P = (v_0 = 0, v_1, v_2, v_3, \ldots, v_n, v_{n+1} = n+1)$ and $T = (m_1, m_2, \ldots, m_n)$ is called feasible if each arc is travelled within its corresponding time period, i.e., $T_{v_i v_{i+1}}^{m_i - 1} < t_{v_i} \leq T_{v_i v_{i+1}}^{m_i}$, for $i = 1, \ldots, n$. A path $P = (v_1, \ldots, v_k)$ is called *infeasible* if it is not contained as a subpath in any feasible Hamiltonian time dependent path.

2. **Time windows allowed:** in this situation, if we also allow waiting times for the arrival at a particular vertex, we need to modify the definitions from the previous cases. We consider the possibility of waiting at the arrival at a vertex as well as at the departure. Let t_{v_i} and s_{v_i} be the earliest departure and arrival time for vertex v_i, respectively, for the time dependent simple path $[P, T]$. These values are calculated as

$$\begin{aligned} t_{v_1} &= \max\{r_{v_1} + p_{v_1}, T_{v_1 v_2}^{m_1 - 1} + 1\} \\ s_{v_i} &= \max\{t_{v_{i-1}} + c_{v_{i-1} v_i}^{m_{i-1}}, r_{v_i}\}, \quad i = 2, \ldots, k \\ t_{v_i} &= \max\{s_{v_i} + p_{v_i}, T_{v_i v_{i+1}}^{m_i - 1} + 1\}, \quad i = 2, \ldots, k-1 \end{aligned}$$

   These formulae allow waiting times at the departure of and the arrival at v_i.

   A Hamiltonian time dependent path $[P, T]$, where $P = (v_0 = 0, v_1, v_2, v_3, \ldots, v_n, v_{n+1} = n+1)$ and $T = (m_1, m_2, \ldots, m_n)$ is called feasible if $T_{v_i v_{i+1}}^{m_{i-1}} < t_{v_i} \leq \min\{T_{v_i v_{i+1}}^{m_i}, d_{v_i} + p_{v_i}\}$ and $r_{v_i} \leq s_{v_i} \leq d_{v_i}$ for $v_i \in H$. A time dependent simple path is *infeasible* if it cannot be a subpath in any feasible Hamiltonian path.

With these definitions, given a Hamiltonian time-dependent path $H = [P, T]$, we can easily compute the above formulae in order to decide whether H corresponds to a feasible solution or not. However, for time-dependent simple paths which are not Hamiltonian, deciding whether it is feasible or not is an $\mathcal{NP}$-Complete problem, since it requires to determine if it can be a subpath of a feasible solution for the problem or not. However, as well as in Ascheuer et al. [5] for the classical TSPTW, we can establish sufficient conditions for $[P, T]$ to be infeasible. They are presented in the following lemma.

Lemma 4.2.1. *A time dependent simple path $[P, T]$, with $P = (v_1, v_2, \ldots, v_k)$ and $T = (m_1, m_2, \ldots, m_{k-1})$, is infeasible if at least one of the following conditions holds:*

1. *$[P, T]$ violates the deadline for its last vertex v_k, that is, $s_{v_k} > d_{v_k}$.*
2. *$[P, T]$ violates the upper bound on the time period assigned to vertex v_{k-1}, that is, $t_{v_{k-1}} > T^{m_{k-1}}_{v_{k-1}v_k}$.*

In order to include time dependent infeasible paths in our formulations, let x^m_{ij} be a binary variable with the same meaning as in the previous formulations. Let also $[P, T]$ be a time dependent infeasible simple path, with $P = (v_1, v_2, \ldots, v_k)$ and $T = (m_1, m_2, \ldots, m_{k-1})$. Then, we can forbid $[P, T]$ to be part of a feasible solution with the following inequality

$$\sum_{i=1}^{k-1} x^{m_i}_{v_i v_{i+1}} \leq k - 2. \tag{4.24}$$

We name this family of inequalities *Time-Dependent Infeasible Path Elimination Constraints* (TDIPECs). Based on this generalization, we propose in the following sections two different formulations for the TDTSP-TW.

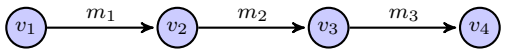


Figure 4.1.: Support for TDIPECs ($k = 4$).

Figure 4.1 shows the support a constraint (4.24) with $k = 4$. For this example, the associated TDIPEC is

$$x^{m_1}_{v_1v_2} + x^{m_2}_{v_2v_3} + x^{m_3}_{v_3v_4} \leq 2.$$

4.2.2. An ILP formulation

In this section we present the natural generalization to the time-dependent case of the formulation proposed by Ascheuer et al. [5, 6]. By means of the time-dependent infeasible paths defined previously, the aim of this formulation is to avoid using big M-like inequalities.

Let binary variables x^m_{ij} as in the previous models, i.e. x^m_{ij} take value one if and only if the tour visits vertex j immediately after vertex i departing from i in time period m. In the formulation shown below, we need to include the well-known *Subtour Elimination Constraints*

(SEC, see Section 2.3, (2.6)) to forbid subtours as feasible solutions. For notational convenience, we define for $(i,j) \in A$ the traditional binary variables x_{ij} in terms of x_{ij}^m as

$$x_{ij} = \sum_{m=1}^{M} x_{ij}^m \tag{4.25}$$

to include such inequalities in the model. However, it is important remark that variables x_{ij} are not required to be added explicitly since they can be expressed in terms of variables x_{ij}^m. The first model based on time-dependent infeasible paths is shown in (4.26) - (4.32), named TDIPF.

$$\min \; z_{\text{cost}} = \sum_{(i,j)\in A} \sum_{m=1}^{M} \theta_{ij}^m x_{ij}^m \tag{4.26}$$

s.t.

$$\sum_{i=0, i\neq j}^{n} \sum_{m=1}^{M} x_{ij}^m = 1 \qquad j \in V\backslash\{0\} \tag{4.27}$$

$$\sum_{j=1, j\neq i}^{n+1} \sum_{m=1}^{M} x_{ij}^m = 1 \qquad i \in V\backslash\{0, n+1\} \tag{4.28}$$

$$\sum_{j=1}^{n} \sum_{m=1}^{M} x_{0j}^m = 1 \tag{4.29}$$

$$\sum_{i,j\in W, i\neq j} x_{ij} \leq |W| - 1 \qquad \forall\, W \subset V,\ |W| \geq 2 \tag{4.30}$$

$$\sum_{i=1}^{k-1} x_{v_i v_{i+1}}^{m_i} \leq k - 2 \qquad \forall \text{ infeasible time dependent path } [P,T] \tag{4.31}$$

$$x_{ij}^m \in \{0,1\} \qquad \forall (i,j) \in A,\ 1 \leq m \leq M \tag{4.32}$$

The objective function (4.26) minimizes the cost of the tour. Equations (4.27) and (4.28) correspond to in-degree and out-degree constraints, respectively, while equation (4.29) forces the vehicle to leave the depot. Inequalities (4.30) correspond to the SEC. Inequalities (4.31) are the TDIPECs proposed in the previous section. Their presence ensure that the correct time period is chosen for each arc according to the departure time from the origin vertex and, in case of having time windows, that clients are visited within the corresponding release and deadline times. Finally, (4.32) establishes that variables x_{ij}^m must be binary. Clearly, from the definition of infeasible paths, the TDIPF defines a correct model for the TDTSP-TW.

4.2.3. A refined formulation for the TDTSP-TW

The following formulation is based on the one proposed in Dash et al. [19] for the TSPTW, named the *Time Bucket Formulation* (TBF). The idea behind the TBF is the application of the *under-*

constrained method from Wang and Regan [62] to the *Time Index Formulation* (TIF) derived from the transformations proposed by Albiach et al. [3] for the TDTSP-TW -but without the time dependency. Instead of considering every possible time instant, the approach relies on the introduction of the so-called *time buckets* (i.e. subintervals defined by collection of consecutive time instants) to reduce the number of variables in the model. This approach may allow solutions that are infeasible for the original problem but, to overcome this issue, the authors consider as well the *Subtour Elimination Constraints* (SEC) and the *Infeasible Path Elimination Constraints* (IPEC) proposed in Ascheuer et al. [5, 6]. In order to apply these ideas and formulate a model for the TDTSP-TW, we need to modify and adapt the definitions from Dash et al. [19] and consider the TDIP defined in Section 4.2.1. Following the notation proposed by Dash et al. we generalize the definition of buckets to the time dependent case.

Definition 4.1 (Buckets' minimal conditions)**.** *For each node $i \in V$ its corresponding time window W_i is divided into a set of buckets $B_i = \{b_1^i, \ldots, b_{L_i}^i\}$ in such a way that each bucket $b_l^i = [r_{b_l^i}, d_{b_l^i}]$, $r_{b_l^i} \leq d_{b_l^i}$, satisfies the following conditions:*

1. $r_{b_1^i} = r_i$,
2. $d_{b_{L_i}^i} = d_i + p_i$,
3. $r_{b_{l+1}^i} > d_{b_l^i}$, *for* $l = 1, \ldots, L_i - 1$, *and*
4. *If* $T_{ij}^{m-1} + 1 \in [r_i, d_i + p_i]$ *for* $j \in V$ *and* $1 \leq m \leq M$, *then for some* $1 \leq l \leq L_i$, $r_{b_l^i} = T_{ij}^{m-1} + 1$.

These conditions allow the buckets not to cover all the time instants in $[r_i, d_i + p_i]$. However, these time instants cannot be a valid arrival or departure of a feasible time dependent tour at vertex i. If for a vertex $i \in V$ a time instant $t \in [r_i, d_i + p_i]$ is not a valid arrival or departure instant for any feasible solution for the TDTSP-TW, then t can be excluded from the buckets' definition for vertex i.

Condition 4 is new with respect to the original ones. To account for time-dependency, we force time period changes taking place during the vertex's time window to match the beginning of one of the buckets defined. The inclusion of this condition is due to the definition of the decision variables and the objective function of the model for the TDTSP-TW. Figure 4.2 shows a graphical representation of conditions from Definition 4.1.

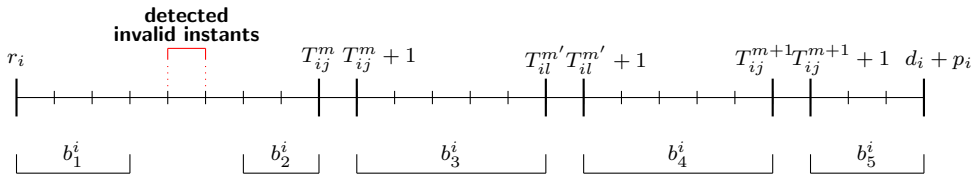


Figure 4.2.: Graphical representation of buckets for $i \in V$.

By definition, given an arc $(i, j) \in A$, a bucket $b \in B_i$ belongs exactly to one time period. In other words, there exists a unique $1 \leq m \leq M$ such that $b \subseteq (T_{ij}^{m-1}, T_{ij}^m]$. Therefore, we

denote by $c_{ij}^b = c_{ij}^m$ the time-dependent travel time when travelling arc (i,j) starting at vertex i during bucket b. Conversely, for $(i,j) \in A$ a time period $1 \leq m \leq M$ is covered by a unique set of buckets $b \in B_i$. In this sense, we define $S_{ij}(m) = \{b \in B_i \ : \ b \subseteq (T_{ij}^{m-1}, T_{ij}^m]\}$ as the set of buckets that defines the time interval m over the arc $(i,j) \in A$.

For each vertex i and bucket $b_l \in B_i$, we define $I_k(i, b_l) = \{b \in B_k \ : d_{b_{l-1}} < r_b + c_{ki}^b \leq d_{b_l}\}$ as the collection of possible departure buckets at vertex k given that we select arc (k,i) and the arrival bucket at vertex i is b_l, with $d_{b_0} = -\infty$. In addition, we define $N_i(k,b)$ as the arrival bucket at vertex i given that the arc (k,i) is selected and b is the departure bucket at k. Formally, $N_i(k,b) = \beta \in B_i$ such that $b \in I_k(i,\beta)$. If $r_b + c_{ki}^b > d_i$, this means that, even when departing from vertex k at the very beginning of time bucket b, the arrival at vertex i exceeds the deadline d_i of its time window and $N_i(k,b)$ is undefined.

Similarly to Dash et al. [19], we define that binary variables y_{ij}^b take value 1 if and only if the vehicle travels from vertex i to vertex j starting from i in $b \in B_i$, and z_i^b take value 1 if and only if the vehicle arrives at i in bucket b. To account for waiting times at the departure of a vertex, we introduce binary variables y_i^b which take value 1 if and only if the vehicle leaves vertex i in bucket b. Variables x_{ij}^m, and therefore x_{ij} considering the definition in (4.25), can be defined correctly in terms of y_{ij}^b as

$$x_{ij}^m = \sum_{b \in S_{ij}(m)} y_{ij}^b.$$

We assume that, for any vertex $i \in V$, if due to the presence of a time window $W_i = [r_i, d_i]$ travelling from i to j in time period m is not possible, then $x_{ij}^m = 0$. This is expressed in the first formula since $S_{ij}(m) = \emptyset$. An analogous assumption is made for x_{ij} variables. By considering (4.33) - (4.42) we obtain another formulation for the TDTSP-TW that we name *Time Dependent Time Bucket Formulation* (TDTBF). It is important to remark that x_{ij} variables are included in the formulation only for the sake of simplicity. In practice, we consider their definition in terms of x_{ij}^m variables.

$$\min z_{\text{cost}} = \sum_{(i,j) \in A} \sum_{m=1}^{M} \theta_{ij}^m x_{ij}^m \tag{4.33}$$

s.t.

$$\sum_{b \in B_i} z_i^b = 1 \qquad \forall\ i \in V \backslash \{0\} \tag{4.34}$$

$$\sum_{b \in B_i} y_i^b = 1 \qquad \forall\ i \in V \backslash \{n+1\} \tag{4.35}$$

$$\sum_{\substack{\beta \geq b \\ \beta \in B_i}} z_i^\beta \leq \sum_{\substack{\beta \in B_i, \beta \geq b \\ r_b + p_i \leq d_\beta}} y_i^\beta \qquad \forall\ i \in V \backslash \{0, n+1\},\ b \in B_i \tag{4.36}$$

$$\sum_{\substack{j=1, j \neq i \\ N_j(i,b) \neq \emptyset}}^{n} y_{ij}^b = y_i^b \qquad \forall\ i \in V \backslash \{n+1\},\ \forall\ b \in B_i \tag{4.37}$$

$$\sum_{k=1,k\neq i}^{n} \sum_{\beta\in I_k(i,b)} y_{ki}^{\beta} = z_i^b \qquad \forall\, i \in V\backslash\{0\},\ \forall\, b \in B_i \tag{4.38}$$

$$\sum_{b\in S_{ij}(m)} y_{ij}^b = x_{ij}^m \qquad \forall\, (i,j)\ \in A,\ 1 \leq m \leq M \tag{4.39}$$

$$\sum_{i,j\in W, i\neq j} x_{ij} \leq |W| - 1 \qquad \forall\, W \subset V,\ |W| \geq 2 \tag{4.40}$$

$$\sum_{i=1}^{k-1} x_{v_i v_{i+1}}^{m_i} \leq k-2 \qquad \forall \text{ infeasible time dependent path } [P,T] \tag{4.41}$$

$$x_{ij}^m, y_{ij}^b, z_i^b, y_i^b \in \{0,1\} \qquad \forall\, (i,j) \in A,\ \forall\, i \in V,\ \forall\, b \in B_i \tag{4.42}$$

The objective function (4.33) minimizes the cost of the tour. Equations (4.34) and (4.35) establish that each vertex must be visited and (4.36) accounts for waiting times at the departure. Equations (4.37) and (4.38) relate y_{ij}^b variables with z_i^b and y_i^b, respectively. Identity (4.39) defines in an explicit way variables x_{ij}^m. Inequalities (4.40) and (4.41) read for the SEC and TDIPEC. Finally, integrality constraints are established in (4.42).

Proposition 4.2.2. *Any feasible solution for the TDTSP-TW is feasible for the TDTBF.*

Proof. For simplicity, w.l.o.g. let $P = (0, 1, \ldots, n, n+1)$ and $T = (m_0, m_1, \ldots, m_n)$ be a feasible time-dependent Hamiltonian path. Let

also s_i, for $i = 1, \ldots, n+1$, and t_i, for $i = 0, \ldots, n$, be the earliest arrival and departure times, respectively, for $[P,T]$. For all $i \in V$ we identify time instant ϕ_i and buckets $\gamma(i), \beta(i)$, where ϕ_i and $\gamma(i)$ will be related to the arrival and $\beta(i)$ to the departure.

For $i = 0$, we define $\beta(0)$ as the first bucket in B_0 containing t_0, i.e., $d_{\beta(0)-1} < t_0 \leq d_{\beta(0)}$. Considering the definitions of t_i and the conditions imposed to buckets, it is simple to check that $r_{\beta(0)} = t_0$. For $i \geq 1$, we define:

- $\phi_i = \max\{r_i, r_{\beta(i-1)} + c_{i-1,i}^{\beta(i-1)}\}$ as the arrival instant to vertex i departing from $i-1$ en $\beta(i-1)$,
- $\gamma(i) = b_l \in B_i$ such that $d_{b_{l-1}} < \phi_i \leq d_{b_l}$ (recall that $d_{b_0} = -\infty$),
- $\beta(i) = b_l \in B_i$ such that $r_{b_l} \leq t_i \leq d_{b_l}$ (recall that $d_{b_0} = -\infty$).

For $[P,T]$, we set $y_i^{\beta(i)} = y_{i,i+1}^{\beta(i)} = 1$, for $i = 0, \ldots, n$, and $z_i^{\gamma(i)} = 1$, for $i = 1, \ldots, n+1$. We have to prove now that this assignment is correct and that it provides the same cost as $[P,T]$.

1. *Buckets $\beta(i)$ and $\gamma(i)$ are well defined.* First, note that by definition $\beta(i)$ is correct, since t_i is a valid time instant and therefore $r_{\beta(i)} \leq t_i \leq d_{\beta(i)}$ for $i = 0, \ldots, n$. Furthermore, $c_{i,i+1}^{\beta(i)} = c_{i,i+1}^{m_i}$ holds, since condition 4 from the buckets' definition allows us to deduce the following relation
$$T_{i,i+1}^{m_i-1} < r_{\beta(i)} \leq t_i \leq d_{\beta(i)} \leq T_{i,i+1}^{m_i}. \tag{4.43}$$

We also check that the definition of $\gamma(i)$ is correct. For that purpose, we prove that $r_i \leq \phi_i \leq d_i$. First, notice that $r_i \leq \phi_i$ by definition, for $i = 1, \ldots, n+1$. For the second one, it is quite easy to see that

$$\phi_i = \max\{r_i, r_{\beta(i-1)} + c_{i-1,i}^{\beta(i-1)}\} \leq \max\{r_i, t_{i-1} + c_{i-1,i}^{m_{i-1}}\} = s_i, \text{ for } i = 1, \ldots, n+1.$$

and since s_i is a valid time instant, it follows that $s_i \leq d_i$. Then, $r_i \leq \phi_i \leq d_i$ and therefore $\gamma(i)$ is also well defined and $N_i(i-1, \beta(i-1)) = \gamma(i)$.

2. *The solution is feasible for the TDTBF.* First we note that equations (4.34) and (4.35) are satisfied by definition, since we are assigning only one arrival bucket and one departure bucket for each vertex (except for vertices 0 and $n+1$, having only a departure and an arrival bucket, respectively).

   Next we prove that constraints (4.36) are satisfied. Consider $i \in V \backslash \{0, n+1\}$ and $b \in B_i$. We first state that if $\beta(i) \geq b$ and $r_b + p_i \leq d_\beta$, the constraint is satisfied. We now consider the remaining two cases.

   If $\beta(i) < b$, we prove that $\gamma(i) \leq \beta(i)$ and therefore constraints are satisfied. As seen in the previous item, we know that $\phi_i \leq s_i$ and from the definitions of s_i and t_i, $s_i \leq t_i$, we deduce $\phi_i \leq s_i \leq t_i$. Given that $\phi_i \leq d_{\gamma(i)}$, $t_i \leq d_{\beta(i)}$ and the conditions for the definitions of buckets, it follows that $\gamma(i) \leq \beta(i)$.

   Now we consider the case where $\beta(i) \geq b$ and $r_b + p_i > d_{\beta(i)}$. A sufficient condition to prove that the constraint is satisfied is that $\gamma(i) < b$. We argue by contradiction. Suppose $\gamma(i) \geq b$. Then, we have that $r_b \leq r_{\gamma(i)}$ and therefore $r_b + p_i \leq r_{\gamma(i)} + p_i$. First we prove that $r_{\gamma(i)} \leq s_i$. We analyze two possible situations. If $r_{\gamma(i)} \leq \phi_i$, then $r_{\gamma(i)} \leq \phi_i \leq s_i$. Otherwise, ϕ_i is an infeasible time instant. The first feasible time instant following ϕ_i is greater than or equal to $r_{\gamma(i)}$ and since s_i is feasible and $\phi_i \leq s_i$, it follows that $\phi_i < r_{\gamma(i)} \leq s_i$.Then, given that $r_{\gamma(i)} \leq s_i$, we can establish the following relation

   $$r_b + p_i \leq r_{\gamma(i)} + p_i \leq s_i + p_i \leq t_i \leq d_{\beta(i)}$$

   which produces a contradiction because we assumed that $r_b + p_i > d_{\beta(i)}$. Then, we proved that constraints (4.36) are satisfied.

   It is time to constraints (4.37). We prove that $N_{i+1}(i, \beta(i))$ is non-empty. Since t_i is a valid time instant, travelling from i to $i+1$ departing in time period m_i at time t_i is feasible. By definition, $r_{\beta(i)} \leq t_i$, for $i = 0, \ldots, n$, and then travelling from i to $i+1$ in bucket $\beta(i)$ is also feasible. This implies that $N_{i+1}(i, \beta(i))$ is non-empty, $y_{i,i+1}^{\beta(i)}$ is well defined and then constraints (4.37) are satisfied.

   For constraints (4.38), since $N_i(i-1, \beta(i-1)) = \gamma(i)$ we have that $\beta(i-1) \in I_{i-1}(i, \gamma(i))$ and therefore are also satisfied.

   By the definition of buckets and S_{ij}^m, a time period for a given arc is defined by a unique set of buckets. Then considering (4.43) from the previous item we have that $x_{i,i+1}^{m_i} = 1$ since $\beta(i) \in S_{i,i+1}^{m_i}$, for $i = 0, \ldots, n$, and constraints (4.39) are satisfied. Finally, due to the feasibility of $[P, T]$ and the definition of variables x_{ij}, SEC (4.40) and TDIPECs (4.41) are also satisfied.

3. *Equivalent objective value.* This is directly implied by the fact that x_{ij}^m variables are well defined, accordingly to the arguments used in the previous item.

Then, we can conclude that the solution provided is feasible and it has the same objective value than $[P, T]$. □

This proposition establishes a particular form to represent a feasible solution for the TDTSP-TW in terms of buckets. However, there may exist more than one way of expressing TDTSP-TW solutions. We will call the representation adopted in Proposition 4.2.2 *canonical solutions*. Moreover, from the proof we deduce some interesting properties that hold for this kind of solutions. These properties will be further considered in successive sections of this chapter. They are presented in the following corollary.

Corollary 4.2.3. *Let $[P, T]$ be a feasible solution for the TDTSP-TW. Consider the earliest arrival and departure times for $[P, T]$, i.e., s_i and t_i, and arrival and departure buckets $\gamma(i)$ and $\beta(i)$ from a canonical solution. Then, we can state the following properties:*

1. $r_{\gamma(i)} \leq s_i \leq t_i \leq d_{\beta(i)}$, *for* $i = 1, \ldots, n$,
2. $r_{\beta(i)} \leq t_i \leq d_{\beta(i)}$, *for* $i = 0, \ldots, n$, *and*
3. $r_{\gamma(n+1)} \leq s_{n+1}$.

Finally, we compare the TDTBF with the model presented in Section 4.2.2, the TDIPF. First, we note that conditions in Definition 4.1 allow different levels of granularity for the buckets' definition. Therefore, the number of variables and constraints in the TDTBF clearly depends on these definitions, which represents a very important factor to consider. A very fine discretization will increase the size of the formulation TDTBF considerably compared to the TDIPF. As a counterpart, we may obtain a tighter LP relaxation.

4.3. Valid inequalities

In this section we present several families of valid inequalities for the TDIPF and the TDTBF. In particular, families involving only variables x_{ij}^m (and variables x_{ij}, which are defined in terms of x_{ij}^m ones) are valid for both formulations. As regards the TDTBF, as mentioned in the previous section, a solution of the TDTSP-TW can be represented in terms of buckets in several ways. Then, some specific families of valid inequalities derived for the TDTBF are valid for canonical solutions, which assumes a particular representation in terms of buckets. We name P_1^{TD} to the polytope defined by the TDIPF and P_2^{TD} to the polytope defined by the TDTBF.

4.3.1. TDIPECs and Time-Dependent Tournament Constraints

The purpose of this section is to analyze in detail several properties related to the TDIPECs, generalizing some of the results presented in Ascheuer et al. [5] for the time dependent case. The aim of this analysis is to describe a procedure to incorporate a strengthened version of TDIPECs in our B&C algorithm.

The well-known *Tournament Constraints* (TOURs) proposed in Ascheuer et al. [5] are a family of valid inequalities for the TSPTW obtained by strengthening the IPECs. The main advantage of considering TOURs is that they dominate the IPECs.

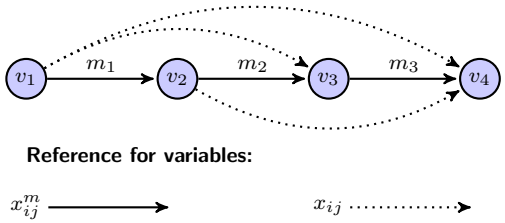


Figure 4.3.: Support for TDTOURs ($k = 4$).

For the TDTSP-TW we state an analogous result. Due to the definitions of TDIPs, by fixing the time periods in which the arcs of the path are travelled, we can also strengthen the TDIPECs by including new variables in the inequality. In the next proposition we introduce the *Time Dependent Tournament Constraints* (TDTOURs) and prove that they are valid. Figure 4.3 shows the support for a TDTOUR with $k = 4$.

Proposition 4.3.1 (TDTOURs)**.** *Given a time dependent infeasible simple path $[P,T]$, with $P = (v_1, v_2, \ldots, v_k)$ and $T = (m_1, m_2, \ldots, m_{k-1})$, the Time Dependent Tournament Constraint (TDTOUR) defined by*

$$\sum_{i=1}^{k-1} x_{v_i v_{i+1}}^{m_i} + \sum_{i=1}^{k-2} \sum_{j=i+2}^{k} x_{v_i v_j} \leq k-2 \tag{4.44}$$

is a valid inequality for P_1^{TD} and P_2^{TD}.

Proof. We argue similarly to Ascheuer et al. [5]. By the degree inequalities and the definition of variables x_{ij}^m and x_{ij} (recall x_{ij} can be defined in terms of the x_{ij}^m ones), we deduce that $x_{v_i v_{i+1}}^{m_i} + \sum_{j=i+2}^{k} x_{v_i v_j} \leq 1$, for $i = 1, \ldots, k-1$. Suppose we have, for a time-dependent infeasible simple path $[P,T]$, a feasible solution such that

$$\sum_{i=1}^{k-1} x_{v_i v_{i+1}}^{m_i} + \sum_{i=1}^{k-2} \sum_{j=i+2}^{k} x_{v_i v_j} > k-2$$

This would imply that $x_{v_i v_{i+1}}^{m_i} + \sum_{j=i+2}^{k} x_{v_i v_j} = 1$, for $i = 1, \ldots, k-1$. Then, we can formulate the following linear system

$$
\begin{aligned}
x^{m_{k-1}}_{v_{k-1}v_k} &= 1 \\
x^{m_{k-2}}_{v_{k-2}v_{k-1}} + x_{v_{k-2}v_k} &= 1 \\
x^{m_{k-3}}_{v_{k-3}v_{k-2}} + \sum_{j=k-1}^{k} x_{v_{k-3}v_k} &= 1 \\
&\vdots \\
x^{m_1}_{v_1v_2} + \sum_{j=3}^{k} x_{v_1v_j} &= 1
\end{aligned}
$$

We now analyze the solution of this system. The first equation sets $x^{m_{k-1}}_{v_{k-1}v_k} = 1$. Considering the second equation, from the degree inequalities and that $x^{m_{k-1}}_{v_{k-1}v_k} = 1$ we deduce that $x_{v_{k-2}v_k} = 0$ and therefore $x^{m_{k-2}}_{v_{k-2}v_{k-1}} = 1$. Using analogous arguments for the remaining equations, we obtain that the system of equations has a unique solution $x^{m_i}_{v_iv_{i+1}} = 1$, for $i = 1, \ldots, k-1$. This produces a contradiction, since $[P, T]$ is a time-dependent infeasible path and therefore cannot be part of a feasible solution. □

This result allows us to use TDTOURs instead of TDIPECs, providing with stronger inequalities to forbid TDIPs as feasible solutions. We next describe the separation procedure for the TDTOURs.

Separation of TDTOURs

In the first place, we focus on a dominance relation among the TDTOURs. Similarly to the TSPTW, in the following proposition we show that if a TDIP $[P', T']$ has a proper subpath $[P, T]$ which is also infeasible, then the TDTOUR associated to $[P, T]$ dominates the one associated to $[P', T']$.

Proposition 4.3.2. *Let $[P', T']$ be a time-dependent simple path and $[P, T]$ be a proper subpath of $[P', T']$. If $[P, T]$ is a TDIP, then the TDTOUR induced by $[P, T]$ dominates the TDTOUR induced by $[P', T']$.*

Proof. W.l.o.g, we assume that $[P', T']$ is an extension of $[P, T]$ having one more vertex and the corresponding time period at the end, i.e. $P = (v_1, v_2, \ldots, v_k)$, $T = (m_1, m_2, \ldots, m_{k-1})$ and that $P' = (v_1, \ldots, v_k, v_{k+1})$ and $T' = (m_1, \ldots, m_{k-1}, m_k)$, with $v_{k+1} \notin P$. We rewrite the TDTOUR induced by $[P', T']$ in terms of the TDTOUR induced by $[P, T]$.

$$
\sum_{i=1}^{k-1} x^{m_i}_{v_iv_{i+1}} + \sum_{i=1}^{k-2} \sum_{j=i+2}^{k} x_{v_iv_j} + \underbrace{x^{m_k}_{v_kv_{k+1}} + \sum_{i=1}^{k-1} x_{v_iv_{k+1}}}_{\leq 1} \leq k - 2 + 1
$$

Clearly, the TDTOUR induced by $[P', T']$ is obtained from that of $[P, T]$ and a linear combiantion with an expression dominated by the in-degree equation for v_{k+1}, and therefore it is weaker.

By an analogous argument, if $[P, T]$ is a proper subpath of $[P', T']$ ending at vertex v_{k+1} and travelling arc (v_k, v_{k+1}) in time period m_{k+1}, the dominance relation also holds. □

Based on the ideas proposed in Fischetti and Toth [23] for the separation of the D_k^+ and D_k^- inequalities for the TSP and the properties analyzed in this section, we devise an heuristic procedure to separate the TDIPECs. A similar approach is followed in Ascheuer et al. [6] and Dash et al. [19] regarding the IPECs for the TSPTW.

The core of the procedure is a backward enumeration of the time-dependent simple paths ending in $v \in V \backslash \{0\}$ that are present in the support graph, checking whether each of them is infeasible using conditions 1 and 2 from Lemma 4.2.1. Since these conditions are sufficient, the procedure will be heuristic for general paths. However, it is important to remark that this procedure is exact for time dependent Hamiltonian paths. In addition, based on Proposition 4.3.2, if the TDTOUR induced by a partial time-dependent path during the enumeration is not violated, then by the dominance relation we stop extending such path.

Although the number of time-dependent paths is exponential, in practice the computational time required by this procedure is reasonable compared with the overall time required by the B&C algorithm. However, it is important to remark that compared with the enumeration procedures used for the TSPTW the computational effort is higher, since the time dependence factor generates more possibilities during the enumeration.

4.3.2. Strengthening TDTOURs

Infeasibility of a time dependent simple path $[P, T]$ depends on the particular time periods each arc is travelled. Specifically, a sequence of vertices $P = (v_1, \ldots, v_k)$ might be infeasible for a sequence of time periods $T = (m_1, \ldots, m_{k-1})$ but feasible for a different assignment of time periods for each transition. In this section we aim to identify conditions and characteristics for time dependent paths in order to strengthen the TDTOURs.

First we notice that, from the proof of Proposition 4.3.1, a TDTOUR is violated if and only if $[P, T]$ is infeasible. We use this fact in the following result to generalize TDTOURs by considering sets of time periods for each arc in $[P, T]$.

Proposition 4.3.3. *Let $S_1, \ldots, S_{k-1} \subseteq \{1, \ldots, M\}$ be a collection of sets of time periods and $P = (v_1, \ldots, v_k)$ a simple path such that $[P, T]$ is infeasible for all $T = (m_1, \ldots, m_{k-1})$, where $m_i \in S_i$ for $i = 1, \ldots, k-1$. Then, inequality*

$$\sum_{i=1}^{k-1} \sum_{m \in S_i} x_{v_i v_{i+1}}^m + \sum_{i=1}^{k-2} \sum_{j=i+2}^{k} x_{v_i v_j} \leq k - 2 \tag{4.45}$$

is valid for P_1^{TD} and P_2^{TD}.

Proof. We argue in a similar way as in the proof of Proposition 4.3.1. Formulating a similar linear system we arrive at the conclusion that there exists a unique solution which satisfies $\sum_{m \in S_i} x^m_{v_i v_{i+1}} = 1$, for $i = 1, \ldots, k-1$. Since by hypothesis $[P, T]$ is infeasible for any sequence of time periods $T = (m_1, \ldots, m_{k-1})$ with $m_i \in S_i$, it follows that the inequality is valid. □

The Tournament Constraints can be seen as a particular case of Proposition 4.3.3. The following corollary states conditions for the validity of the TOURs.

Corollary 4.3.4 (TOURs)**.** *If a simple path $P = (v_1, \ldots, v_k)$ is infeasible for all possible sequences of time periods $T = (m_1, \ldots, m_{k-1})$, $1 \leq m_i \leq M$ for $i = 1, \ldots, k-1$, then the Tournament Constraint (TOUR)*

$$\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} x_{v_i v_j} \leq k - 2 \tag{4.46}$$

is valid for P_1^{TD} and P_2^{TD}.

In practice, paths satisfying conditions established by Corollary 4.3.4 may not appear frequently given the characteristics of the TDTSP-TW. However, in order to provide stronger inequalities than the TDTOURs, given a TDIP we identify particular sets of time periods for each arc that satisfy the condition imposed by Proposition 4.3.3.

We first extend to the time-dependent case the definition of *minimal violation paths*, introduced by Mak and Ernst [41] for the TSPTW.

Definition 4.2. *An infeasible time dependent path $[P, T]$, with $P = (v_1, \ldots, v_k)$ and $T = (m_1, \ldots, m_{k-1})$, is called a minimal violation path if both truncated sequences $[P_1, T_1]$, $P_1 = (v_1, \ldots, v_{k-1})$ and $T_1 = (m_1, \ldots, m_{k-2})$, and $[P_2, T_2]$, $P_2 = (v_2, \ldots, v_k)$ and $T_2 = (m_2, \ldots, m_{k-1})$ are feasible.*

Minimal violation paths can be handled easily from a computational point of view. Indeed, checking whether a time dependent simple path is a minimal violation requires the same computational effort, in terms of complexity, than checking sufficient conditions 1 and 2 for infeasibility from Lemma 4.2.1. In addition, every TDIP has as a subpath at least one minimal violation path, and therefore we can restrict our search only to this particular type of time-dependent paths.

We define the *critical arrival and departure times* for a given time dependent simple path. The intuition behind these definitions lies on identifying two time instants for each vertex after which any arrival or departure, depending on the case, makes the path infeasible.

Definition 4.3. *Let $[P, T]$ be a time dependent simple path, with $P = (v_1, \ldots, v_k)$ and $T = (m_1, \ldots, m_{k-1})$. We define the critical arrival and departure times, z_{v_i} and y_{v_i}, respectively, as*

$$\begin{aligned} z_{v_k} &= d_{v_k} \\ y_{v_i} &= \min\{T^{m_i}_{v_i v_{i+1}}, d_{v_i} + p_{v_i}, z_{v_{i+1}} - c^{m_i}_{v_i v_{i+1}}\}, \quad i = 1, \ldots, k-1 \\ z_{v_i} &= y_{v_i} - p_{v_i}, \quad i = 2, \ldots, k-1 \end{aligned}$$

In particular, minimal violation paths provide some useful characteristics concerning general TDIPs. We prove these results in the following lemma. For example, we can deduce that no waiting times are present neither at the arrival at or the departure from a vertex. Furthermore, if considering also critical arrival and departure times z_{v_i} and y_{v_i}, we can provide useful information regarding the relation between these values and the time periods of the arcs in the path.

Lemma 4.3.5. *If $[P,T]$ is a minimal violation path, with $P = (v_1, \ldots, v_k)$ and $T = (m_1, \ldots, m_{k-1})$, then the following properties hold:*

1. *There are no waiting times at the arrival to v_i, for $i = 2, \ldots, k-1$.*
2. *There are no waiting times at the departure from v_i, for $i = 2, \ldots, k-1$.*
3. $\max\{r_{v_i} + p_{v_i}, T^{m_i-1}_{v_i v_{i+1}} + 1\} \leq y_{v_i} \leq \min\{d_{v_i} + p_{v_i}, T^{m_i}_{v_i v_{i+1}}\}$, *for $i = 2, \ldots, k-1$.*
4. $z_{v_{i+1}} - c^{m_i}_{v_i v_{i+1}} < \min\{T^{m_i}_{v_i v_{i+1}}, d_{v_i} + p_{v_i}\}$, *for $i = 1, \ldots, k-2$.*
5. *Let $[\hat{P}, \hat{T}]$ be an ending subpath of $[P,T]$ starting at v_i, i.e. $\hat{P} = (v_i, v_{i+1}, \ldots, v_k)$ and $\hat{T} = (m_i, m_{i+1}, \ldots, m_{k-1})$. Then, departing from v_i at a time instant greater that y_{v_i}, i.e. $t > y_{v_i}$, is infeasible.*

Proof. We divide the proof in blocks for a better comprehension:

- For items 1 and 2 we consider the truncated path $[P_2, T_2]$ from Definition 4.2. Suppose that for some v_{i_0}, $2 \leq v_{i_0} \leq k-1$, the arrival time $s_{v_{i_0}} < r_{v_{i_0}}$. Consider the subpath $[\hat{P}, \hat{T}]$, with $\hat{P} = (v_{i_0}, \ldots, v_k)$ and $\hat{T} = (m_{i_0}, \ldots, m_{k-1})$. The arrival and departure times for vertices in $[\hat{P}, \hat{T}]$ are the same as in $[P,T]$ and then $[\hat{P}, \hat{T}]$ is infeasible. This is a contradiction, since $[\hat{P}, \hat{T}]$ is a subpath of $[P_2, T_2]$ which is feasible by hypothesis. A similar argument shows that item 2 holds.

- For item 3, we have that $y_{v_i} \leq \min\{d_{v_i} + p_{v_i}, T^{m_i}_{v_i v_{i+1}}\}$ from Definition 4.3. For the remaining inequality, suppose that for some $2 \leq i_0 \leq k-1$, $y_{v_{i_0}} < \max\{r_{v_{i_0}} + p_{v_{i_0}}, T^{m_{i_0}-1}_{v_{i_0} v_{i_0+1}} + 1\}$ and consider $[\hat{P}, \hat{T}]$ as defined before. Then, even departing from v_{i_0} at the earliest possible time $[\hat{P}, \hat{T}]$ is infeasible, which is a contradiction. Thus, item 3 also holds.

- For item 4 we consider the earliest departure and arrival times t_{v_1}, s_{v_k} and s_{v_i}, t_{v_i}, for $i = 2, \ldots, k-1$, for path $[P,T]$. We also consider departure and arrival times for the truncated path $[P_1, T_1]$ from Definition 4.2, i.e., $t^{(1)}_{v_1}$, $s^{(1)}_{v_{k-1}}$ and $s^{(1)}_{v_i}, t^{(1)}_{v_i}$, for $i = i \ldots, k-2$. It is quite easy to see that $t_{v_1} = t^{(1)}_{v_1}$, $s_{v_i} = s^{(1)}_{v_i}$, $t_{v_i} = t^{(1)}_{v_i}$ for $i = 2, \ldots, k-2$ and $s_{v_{k-1}} = s^{(1)}_{v_{k-1}}$.

  From item 1 and 2, we know that there are no waiting times neither at the departure nor at the arrival at any of the vertices in $[P,T]$. Then, we can calculate departure and arrival times for vertices in $[P,T]$ as

$$\begin{aligned} t_{v_1} &= \max\{r_{v_1} + p_{v_1}\}, \\ s_{v_i} &= t_{v_{i-1}} + c^{m_{i-1}}_{v_{i-1} v_i}, \ i = 2, \ldots, k, \\ t_{v_i} &= s_{v_i} + p_{v_i}, \ i = 2, \ldots, k-1. \end{aligned}$$

We now claim that $t_{v_{k-1}} > y_{v_{k-1}}$. Since $[P,T]$ is a minimal violation path, it violates the time period's upper bound for vertex v_{k-1} (i.e., $t_{v_{k-1}} > T^{m_{k-1}}_{v_{k-1}v_k}$) or violates the time window of v_k (i.e., $s_{v_k} = t_{v_{k-1}} + c^{m_{k-1}}_{v_{k-1}v_k} > d_{v_k} = z_{v_k}$). Before continuing, we prove the following intermediate results:

- *Claim 1: If $t_{v_i} > y_{v_i}$, then $s_{v_i} > z_{v_i}$ for $i = 2,\dots,k-1$.* Considering the formulae mentioned above for the computation of the earliest arrival and departure times, we have that
$$s_{v_i} = t_{v_i} - p_{v_i} > y_{v_i} - p_{v_i} = z_{v_i}.$$
- *Claim 2: If $s_{v_i} > z_{v_i}$, then $t_{v_{i-1}} > y_{v_{i-1}}$ for $i = 2,\dots,k-2$.* Considering again the formulae for the earliest arrival and departure times,
$$s_{v_i} = t_{v_{i-1}} + c^{m_{i-1}}_{v_{i-1}v_i} > z_{v_i}$$
and therefore
$$t_{v_{i-1}} > z_{v_i} - c^{m_{i-1}}_{v_{i-1}v_i} \geq \min\{T^{m_i}_{v_{i-1}v_i}, d_{v_{i-1}} + p_{v_{i-1}}, z_{v_i} - c^{m_{i-1}}_{v_{i-1}v_i}\} = y_{v_{i-1}}.$$

Since $t_{v_{k-1}} > y_{v_{k-1}}$ and by means of applying recursively claims 1 and 2 for $i = 2,\dots,k-1$, we can state that

$$t_{v_i} > y_{v_i}, \text{ for } i = 1,\dots,k-1, \text{ and} \tag{4.47}$$
$$s_{v_i} > z_{v_i}, \text{ for } i = 2,\dots,k-1. \tag{4.48}$$

Then, using that $[P_1,T_1]$ is feasible, the definitions for s_i and t_i and equations (4.48)

$$\min\{T^{m_i}_{v_iv_{i+1}}, d_{v_i} + p_{v_i}\} \geq t^{(1)}_{v_i} = t_{v_i} = s_{v_{i+1}} - c^{m_i}_{v_iv_{i+1}} > z_{v_{i+1}} - c^{m_i}_{v_iv_{i+1}},$$

which is the desired result. As a consequence, for a minimal violation path we can calculate critical values using the following formulae

$$\begin{aligned} z_{v_k} &= d_{v_k}, \\ y_{v_{k-1}} &= \min\{T^{m_{k-1}}_{v_{k-1}v_k}, d_{v_{k-1}} + p_{v_{k-1}}, z_{v_k} - c^{m_{k-1}}_{v_{k-1}v_k}\}, \\ z_{v_i} &= y_{v_i} - p_{v_i},\ i = 2,\dots,k-1, \\ y_{v_i} &= z_{v_{i+1}} - c^{m_i}_{v_iv_{i+1}},\ i = 1,\dots,k-2. \end{aligned}$$

- Finally, we prove item 5. The intuition behind this property is that, since $[P,T]$ is a minimal violation path and we depart from v_i after its critical departure value y_{v_i}, then $[\hat{P},\hat{T}]$ does not have waiting times and therefore is also infeasible.

  Let $\hat{t}_{v_i} = t > y_{v_i}$ and $\hat{s}_{v_l}$ and $\hat{t}_{v_l}$ be the arrival and departure times for $[\hat{P},\hat{T}]$ starting at v_i in $\hat{t}_{v_i}$, for $l = i+1,\dots,k-1$. Considering also the formulae from the previous items for the critical values and using similar arguments as before, we can deduce that
  $$z_{v_l} < \hat{s}_{v_l} \quad \text{and} \quad y_{v_l} < \hat{t}_{v_l}, \text{ for } l = i+1,\dots,k-1.$$
  Then, $\hat{t}_{v_{k-1}} \geq y_{v_{k-1}}$ and therefore $[\hat{P},\hat{T}]$ violates the deadline of the time window for vertex $k-1$, the upper bound for time period m_{k-1} for arc (v_{k-1},v_k) or the deadline of the time window for vertex v_k, Then, it is infeasible to depart from v_i in $t > y_{v_i}$.

□

Using these definitions and properties we propose a strengthening of TDTOURs when the underlying time dependent path $[P,T]$ is a minimal violation path. The purpose is to identify for each arc (v_i, v_{i+1}), $i = 1,\dots,k-2$, a set of feasible time periods satisfying sufficient conditions for Proposition 4.3.3. The main idea is to calculate in a backward fashion, starting from v_k, critical values for vertices in $[P,T]$ and sets of time periods for arcs $\hat{S}_{v_i}$ for $i = 1,\dots,k-2$, without modifying the time period for arc (v_{k-1}, v_k).

Proposition 4.3.6. *Let $[P,T]$ be a minimal violation path, with $P = (v_1,\dots,v_k)$ and $T = (m_1,\dots,m_{k-1})$. We define for each v_i, $i = 1,\dots,k-2$, the following set of time periods for arc (v_i, v_{i+1}):*

$$\begin{aligned}\hat{S}(v_i) \;=\; & \{p \text{ time period for } (v_i, v_{i+1}) : p \neq m_i, \\ & \max\{T^{p-1}_{v_i v_{i+1}} + 1, r_{v_i} + p_{v_i}\} + c^p_{v_i v_{i+1}} > z_{v_{i+1}}\} \cup \{m_i\}.\end{aligned}$$

Then, any path of the form $[P,T']$, with $T' = (\hat{m}_1,\dots,\hat{m}_{k-2}, m_{k-1})$ and $\hat{m}_i \in \hat{S}(v_i)$ for $i = 1,\dots,k-2$, is infeasible.

Proof. We start pointing out that if $\hat{m}_i = m_i$ for $i = 1,\dots,k-2$, then $T' = T$ and therefore $[P,T']$ is infeasible. Now suppose that at least one $\hat{m}_i \neq m_i$, and let i_0 be the last of these time periods. We consider the subpath $[\hat{P},\hat{T}]$, with $\hat{P} = (v_{i_0}, v_{i_0+1},\dots,v_k)$ and $\hat{T} = (\hat{m}_{i_0}, m_{i_0+1},\dots,m_{k-1})$, and $\hat{s}_{v_i}$, $\hat{t}_{v_i}$ the earliest arrival and departure times respectively of vertices in $[\hat{P},\hat{T}]$. From the definition of $\hat{S}_{v_{i_0}}$, we know that even when starting at the beginning of time period $\hat{m}_{i_0}$, $\hat{s}_{v_{i_0+1}} > z_{v_{i_0+1}}$ and therefore $\hat{t}_{v_{i_0+1}} > y_{v_{i_0+1}}$. Then, from property 5 in Lemma 4.3.5 $[\hat{P},\hat{T}]$ is infeasible.

Then, we can conclude that for any sequence T' of time periods in $\hat{S}_{v_i}$ the time dependent path $[P,T]$ is infeasible. □

The definition of sets $\hat{S}(v_i)$ satisfies the hypothesis from Proposition 4.3.3 when $[P,T]$ is a minimal violation path. Then, combining this result with the above proposition we obtain the following family of valid inequalities. The proof is omitted since it is a direct combination of the two mentioned results.

Proposition 4.3.7. *Let $[P,T]$ be a minimal violation path, with $P = (v_1,\dots,v_k)$ and $T = (m_1,\dots,m_k)$, and consider the sets of time periods $\hat{S}(v_i)$, for $i = 1,\dots,k-2$, as defined in Proposition 4.3.6. Then, inequality*

$$\sum_{i=1}^{k-2} \sum_{p \in \hat{S}(v_i)} x^p_{v_i v_{i+1}} + x^{m_{k-1}}_{v_{k-1} v_k} + \sum_{i=1}^{k-2} \sum_{j=i+2}^{k} x_{v_i v_j} + \leq k-2 \tag{4.49}$$

is valid for P_1^{TD} and P_2^{TD}.

This family provides with stronger inequalities than TDTOURs to forbid time dependent minimal violation paths from our formulation. However, we can strengthen them by considering information regarding buckets. As mentioned before, the intuition behind sets $\hat{S}(v_i)$ relies in considering other time periods which, even when departing at the very beginning, produce the time dependent path to become infeasible. If a time period $\hat{m}_i \in \hat{S}(v_i)$, then starting in any of the buckets belonging to $S_{v_i v_{i+1}}(\hat{m}_i)$ is also infeasible. However, the converse is not true. If we consider the beginning of a bucket as a lower bound of the departure time, as seen in Proposition 4.2.2, we can establish stronger conditions than inspecting only the beginning of the time period.

The following proposition generalizes the definition of the sets $\hat{S}(v_i)$ so as to include bucket information and proves an analogous result to the one in Proposition 4.3.6.

Proposition 4.3.8. *Let $[P,T]$ be a minimal violation path, with $P = (v_1,\dots,v_k)$ and $T = (m_1,\dots,m_{k-1})$. We extend the definition of set $\hat{S}(v_i)$ to buckets, for $i = 1,\dots,k-2$ as*

$$\hat{S}_B(v_i) = \{b \in B_i : b \notin S_{v_i v_{i+1}}(m_i), r_b + c^b_{v_i v_{i+1}} > z_{v_{i+1}}\} \tag{4.50}$$

Then, any path $[P,\hat{T}]$ that uses departure buckets $b^{(v_i)} \in \hat{S}_B(v_i) \cup S_{v_i v_{i+1}}(m_i)$, for $i = 1,\dots,k-1$ is infeasible.

Proof. To prove this proposition, we will show that any canonical solution, i.e. the ones considered in the proof of Proposition 4.2.2, is infeasible. The main idea is similar to the one in Proposition 4.3.7 but using the beginning of a bucket in a canonical representation as a lower bound for the departure time from the vertex.

Let $\beta(v_i) \in B_i$ be the departure bucket for vertex v_i in path $[P,\hat{T}]$. First, we notice that $\beta(v_{k-1}) \in S_{v_{k-1}v_k}(m_{k-1})$ because $\hat{S}_B(v_{k-1})$ is not defined. Moreover, if the departure bucket $\beta(v_i) \in S_{v_i v_{i+1}}(m_i)$, for $i = 1,\dots,k-2$, the $[P,\hat{T}]$ is infeasible by hypothesis.

Otherwise, let i_0 be the last vertex from $[P,\hat{T}]$ for which $\beta(v_{i_0}) \in \hat{S}_B(v_i)$. Let us also consider the truncated path $\hat{P} = (v_{i_0},\dots,v_k)$ departing from v_{i_0} in bucket $\beta(v_{i_0})$. Transitions for arcs $(v_{i_0+1}, v_{i_0+2}),\dots,(v_{k-1},v_k)$ are travelled in time periods $m_{i_0+1},\dots,m_{k-1}$, respectively, since we are assuming that v_{i_0} is that last vertex not departing in the time period considered in $[P,T]$.

From the canonical representation of any feasible solution containing $\hat{P}$ as a subpath and starting at v_{i_0} in bucket $\beta(v_{i_0})$, we know that $r_{\beta(v_{i_0})} \le t_{v_{i_0}}$. Then, we set $\hat{t}_{v_{i_0}} = r_{\beta(v_{i_0})}$ and calculate the arrival and departure times $\hat{s}_{v_i}, \hat{t}_{v_i}$ for the remaining vertices in $\hat{P}$ considering travel times $c^{\beta(v_{i_0})}_{v_{i_0}v_{i_0+1}}$ for arc (v_{i_0}, v_{i_0+1}) and time periods $m_{i_0+1},\dots,m_{k-1}$ for (v_{i_0+1}, v_{i_0+2}), $\dots$, (v_{k-1},v_k).

The departing time is $\hat{t}_{v_{i_0}} = r_{\beta(v_{i_0})}$ and since $\beta(v_{i_0}) \in \hat{S}_B(v_{i_0})$, then we can deduce that $z_{v_{i_0+1}} < \hat{s}_{v_{i_0+1}}$ and therefore $y_{v_{i_0+1}} < \hat{t}_{v_{i_0+1}}$. Considering again item 5 from Lemma 4.3.5 we have that $[\hat{P},\hat{T}]$ is infeasible, proving the proposition. □

Based on this result, we can derive a family of valid inequalities similar to the one presented in Proposition 4.3.7.

Proposition 4.3.9. *Let $[P,T]$ be a minimal violation path, with $P = (v_1,\ldots,v_k)$ and $T = (m_1,\ldots,m_{k-1})$. Consider also sets $\hat{S}_B(v_i)$, for $i = 1,\ldots,k-2$, as defined in Proposition 4.3.8. Then, inequality*

$$\sum_{i=1}^{k-1} x^{m_i}_{v_i v_{i+1}} + \sum_{i=1}^{k-2}\sum_{j=i+2}^{k} x_{v_i v_j} + \sum_{i=1}^{k-2}\sum_{\beta\in\hat{S}_B(v_i)} y^{\beta}_{v_i v_{i+1}} \leq k-2 \tag{4.51}$$

is satisfied by all canonical solutions.

Proof. We argue in a similar way as in Proposition 4.3.1 for the TDTOURs. By formulating an analogous linear system, we have that the inequality is violated if and only if $x^{m_{k-1}}_{v_{k-1}v_k} = 1$ and $x^{m_i}_{v_i v_{i+1}} + \sum_{\beta\in\hat{S}_B(v_i)} y^{\beta}_{v_i v_{i+1}} = 1$, for $i = 1,\ldots,k-2$. In Lemma 4.3.8 we prove that any of these paths is infeasible and therefore the inequality is valid. □

Separation procedure

Similarly to TDTOURs, the separation of constraints (4.51) is performed by explicitly enumerating time dependent simple paths. The difference lies in the restriction to minimal violation paths. In addition, when a violated constraint (4.51) is identified, we test whether it can be strengthened to a TOUR (4.46). The algorithm used for this test is shown in Section 4.4.

4.3.3. Bucket sequential ordering polytope inequalities

Following the approach from Dash et al. [19], we also consider Sequential Ordering Polytope (SOP) inequalities derived for the *Precedence Constrained TSP* (PCTSP) introduced by Balas et al. [8] which have been proven to be effective using precedences induced by time windows, as reported in Ascheuer et al. [6].

Simple π_B and σ_B-inequalities

A vertex $i \in V$ precedes vertex $j \in V$, $j \neq i$, if i has to be visited before j in any feasible solution. This relation is expressed as $i \prec j$. For $S \subseteq V\backslash\{0, n+1\}$, define $\pi(S) = \{i \in V : i \prec j \text{ for some } j \in S\}$ and $\sigma(S) = \{j \in V : i \prec j \text{ for some } i \in S\}$. In any feasible solution, if $i \in S \cap \pi(S)$ then i cannot be the last vertex of S visited by the tour. Analogously, if $j \in S \cap \sigma(S)$ then j cannot be the first vertex of S visited by the tour. For $S \subseteq V$, let $\bar{S} = V\backslash S$, $\delta(S) = \{(i,j) \in A \ : \ i \in S,\ j \in \bar{S}\}$ and consider also the following definitions

$$\delta_\pi(S) = \delta(S\backslash\pi(S), \bar{S}\backslash\pi(S)) \text{ and } \delta_\sigma(\bar{S}) = \delta(\bar{S}\backslash\sigma(S), S\backslash\sigma(S)).$$

For all set of vertices $S \subset V\backslash\{0, n+1\}$, the π and σ inequalities

$$\sum_{(i,j)\in\delta_\pi(S)} x_{ij} \geq 1, \tag{4.52}$$

$$\sum_{(i,j)\in\delta_\sigma(\bar{S})} x_{ij} \geq 1, \tag{4.53}$$

are valid for the TDTSP-TW. As pointed out in Balas et al. [8], these inequalities dominate the SECs.

Aiming to obtain stronger inequalities, Dash et al. [19] extend precedence relations among vertices to include information given by buckets.

For a vertex $i \in V$, a bucket $b \in B_i$ is said to precede $j \in V$ if all feasible solutions that visit vertex i in bucket b (i.e., $z_i^b = 1$ or $y_i^b = 1$) have to visit vertex j after vertex i and this is noted as $b \prec j$. Similarly, a vertex $j \in V$ precedes a bucket $b \in B_i$ if all feasible solutions visiting i in bucket b have to visit vertex j before i. This precedence is denoted as $j \prec b$. Vertex precedences imply bucket precedences, but the converse is not true.

Consider a set of buckets $B \subseteq \mathcal{B} = \cup_{i\in V} B_i$. The extensions of $\pi(S)$ and $\sigma(S)$ to buckets are given by

$$\begin{aligned} \pi(B) &= \{b \in \mathcal{B} : b \prec i, \text{ where } i \in V \text{ and } B_i \subseteq B\}, \\ \sigma(B) &= \{b \in \mathcal{B} : i \prec b, \text{ where } i \in V \text{ and } B_i \subseteq B\}. \end{aligned}$$

The general version of the extension of π and σ-inequalities presented in Dash et al. [19], i.e., π_B and σ_B-inequalities (see [19]), are not valid in our case because of the difference in the representation of a solution in terms of buckets. However, we can extend the *simple* version of these inequalities to our case. In fact, the authors report that using the simple version of the inequalities they obtain similar results in comparison to the general version and requires less computational time for the separation procedures. The next proposition presents the simple π_B and σ_B-inequalities. We also provide an alternative proof for the case of the TDTSP-TW[1].

Proposition 4.3.10. *Let $S \subset V\backslash\{0, n+1\}$, $S \neq \emptyset$, and define $B = B(S) = \cup_{i\in S} B_i$. Consider also*

$$\begin{aligned} \delta_\pi(B) &= \{(i,j,b) \ : \ B_i \subseteq B, \ B_j \subseteq \bar{B}, \ b \in B_i\backslash\pi(B), \ N_j(i,b) = b' \in B_j\backslash\pi(B)\}, \text{ and} \\ \delta_\sigma(\bar{B}) &= \{(i,j,b) \ : \ B_i \subseteq \bar{B}, \ B_j \subseteq B, \ b \in B_i\backslash\sigma(B), \ N_j(i,b) = b' \in B_j\backslash\sigma(B)\}. \end{aligned}$$

The simple π_B and σ_B inequalities

$$\sum_{(i,j,b)\in\delta_\pi(B)} y_{ij}^b \geq 1 \qquad \textit{(simple } \pi_B\textit{-inequalities)} \tag{4.54}$$

$$\sum_{(i,j,b)\in\delta_\sigma(\bar{B})} y_{ij}^b \geq 1 \qquad \textit{(simple } \sigma_B\textit{-inequalities)} \tag{4.55}$$

[1] The main difference in the proof relies in the representation of a feasible solution for the TDTSP-TW in terms of buckets.

are valid for P_2^{TD}.

Proof. Let $[P,T]$ be a feasible solution of the TDTSP-TW and consider $P_b = (b_0, b_1, \ldots, b_l)$ the sequence of buckets associated with $[P,T]$. Since vertices $0, n+1 \notin S$ and $S \neq \emptyset$, P_b starts at a bucket in $\bar{B}$, visits at least one bucket in B and ends in $\bar{B}$. Let $b_k, b_{k+1} \in P_b$ be the two consecutive vertices in P_b representing the last cross from B to $\bar{B}$ in P_b, with $b_k \in B_i$ and $b_{k+1} \in B_j$. First, we remark that $i \neq j$ since for any vertex $w \in V$, by definition of B it holds that either $B_w \subseteq B$ or $B_w \subseteq \bar{B}$. Then we point out that $b_k, b_{k+1} \notin \pi(B)$ using the same argument considered in Dash et al. [19]. If $b_k \in \pi(B)$, then there exists a vertex $w \in S$, $B_w \subseteq B$, such that $b \prec w$ and therefore some bucket of $B_w \subseteq B$ is visited after b_k, which is a contradiction. The same argument is valid for b_{k+1}. Then, $(i,j,b_k) \in \delta_\pi(B)$ and inequality (4.54) is valid.

A similar argument can be used to prove that if b_k, b_{k+1} is the first cross from $\bar{B}$ to B, then $b_k, b_{k+1} \notin \sigma(B)$ and therefore inequality (4.55) is valid. □

From the proof of Proposition 4 in Dash et al. [19] it can be checked that simple π_B and σ_B-inequalities dominate π and σ-inequalities. We refer to this paper for a detailed proof.

Simple $(\pi,\sigma)_B$-inequalities

In Balas et al. [8] another family of valid inequalities is introduced for the PCTSP based on the definition of precedences. Let $X, Y \subseteq V\backslash\{0, n+1\}$ such that $i \prec j$ for every pair $i \in X$, $j \in Y$, and let $Q = \{0, n+1\} \cup \pi(X) \cup \sigma(Y)$. Then, for $S \subset V$ such that $X \subseteq S$, $Y \subseteq \bar{S}$, the (π,σ)-inequality is given by

$$\sum_{(i,j)\in\delta(S\backslash Q, \bar{S}\backslash Q)} x_{ij} \geq 1. \tag{4.56}$$

If we infer precedences based on time windows, this family of inequalities is also valid for the TSPTW and, in our case, for the TDTSP-TW. Ascheuer et al. [5] derive a strengthened version of these inequalities for the TSPTW and Dash et al. [19] provide an even stronger inequality extending it to buckets with similar arguments to the ones shown in the previous section. However, these two strengthening assume implicitly that the triangle inequality holds on the travel times, which is not our case. Thus, in the following proposition we show a bucket version of inequality (4.56), the simple $(\pi,\sigma)_B$-inequality, that is valid for the TDTSP-TW.

Proposition 4.3.11. *Let* $X, Y \subseteq V\backslash\{0, n+1\}$ *be two disjoint subsets of vertices satisfying* $i \prec j$ *for every pair* $i \in X$, $j \in Y$ *and* $S \subset V\backslash\{0, n+1\}$ *such that* $X \subseteq S$ *and* $Y \subseteq \bar{S}$. *Let also* $B = B(S)$, $Q_B = B(\{0, n+1\}) \cup \pi(B(X)) \cup \sigma(B(Y))$ *and*

$$\delta_{\pi\sigma}(B) = \{(i,j,b) \ : \ B_i \subseteq B, B_j \subseteq \bar{B},\ b \in B_i\backslash Q_B,\ N_j(i,b) = b' \in B_j\backslash Q_B\}.$$

Then, the simple $(\pi,\sigma)_B$*-inequality defined by*

$$\sum_{(i,j,b)\in\delta_{\pi\sigma}(B)} y_{ij}^b \geq 1 \tag{4.57}$$

is valid P_2^{TD}.

Proof. Let $[P,T]$ be a feasible solution for the TDTSP-TW and $P_b = (b_0, b_1, \ldots, b_l)$ a sequence of buckets associated with $[P,T]$. Every pair of $i \in X$, $j \in Y$ satisfies that $i \prec j$ by hypothesis. Let $i_0 \in X \subseteq S$ be the vertex from X visited last and $j_0 \in Y \subseteq \bar{S}$ the vertex from Y visited first by $[P,T]$, and denote by $[\hat{P},\hat{T}]$ the subpath of $[P,T]$ from i_0 to j_0. We also consider $\hat{P}_b = (b_r, \ldots, b_s)$ as the subpath of P_b where b_r is the departure bucket from i_0 and b_s is the arrival bucket to j_0, with $r < s$. Using a similar argument as in Proposition 4.3.10, we can see that buckets from $\hat{P}_b$ cannot travel through any of the buckets in Q_B. Since $i_0 \in S$ and $j_0 \in \bar{S}$, at least one of the arcs in $[\bar{P},\hat{T}]$ goes from a vertex in S to a vertex in $\bar{S}$ in a bucket which does not belong to Q_B and then that transition is considered in the left hand side of inequality (4.57). □

This family of inequalities is *simple* in two different ways. First, they are simple in the sense proposed by Dash et al. [19] because $B = B(S)$ for some set $S \subseteq V$. Secondly, we also apply its simple version in the sense proposed in Balas et al. [8], i.e. when $X = \{i\}$ and $Y = \{j\}$ with $i \prec j$. This is explained in more detail in the next section.

Separation procedures

In order to separate the Bucket SOP inequalities, we follow the approach from Dash et al. [19] inspired in the separation proposed by Balas et al. [8] for the π and σ-inequalities for the PCTSP. Specifically, they propose a polynomial time separation heuristic for simple Bucket SOP inequalities, based on the calculus several maximum flows on the original graph with particular arc capacities. Furthermore, they show that this procedure -as well as in the work by Balas et al. - results in an exact separation procedure for the SECs. We refer the reader to Dash et al. [19] for a detailed explanation on this aspect.

In Algorithm 4.1 we show the separation heuristic for simple π_B-inequalities from Dash et al. [19]. Simple σ_B-inequalities can be separated using an analogous approach, but considering a reversed graph as mentioned in Balas et al. [8] and calculating the maximum flow setting as target vertex 0 instead of $n+1$.

Regarding the separation of simple $(\pi,\sigma)_B$-inequalities, the approach is similar to the previous cases. Following the remarks made in Balas et al. [8] and Dash et al. [19], the procedure considers sets $X = \{u\}$, $Y = \{w\}$, with $u, w \in V\backslash\{0, n+1\}$, such that it is not known a vertex $v \in V\backslash\{0, n+1\}$ satisfying $u \prec v \prec w$. In Algorithm 4.1, for each $u \prec w$ the set A^S from Step 2 is replaced by $A^{u,w} = \{(i,j,b) \ : \ b \in B_i,\ b' = N_j(i,b),\ b, b' \notin Q_B\}$ and in Step 3 the maximum flow is computed from u to w instead than from u to $n+1$. Step 4 is not considered.

4.3.4. Further valid inequalities

The following three families of valid inequalities are inspired in the temporal constraints proposed by Malandraki and Daskin [42]. The intuition behind the original inequalities is to forbid, given an arc and a feasible time period, all time dependent paths of size two that are infeasible due to the time periods definition. Here, we present a slightly different version of these constraints.

Algorithm 4.1 SIMPLE π_B-INEQUALITIES SEPARATION (Dash et al. [19])

Input: A fractional solution of the LP relaxation of the TDTBF.

1. for $k \in V\backslash\{0, n+1\}$ **do:**

2. Set $S = \{k\}$. Define $A^S = \{(i,j,b) \ : \ b \in B_i, \ b, N_j(i,b) \notin \pi(B(S))\}$ and compute $\tilde{x}$ as

$$\tilde{x}_{ij} = \sum_{(i,j,b)\in A^S} (y_{ij}^b)^*.$$

3. Calculate the max flow z^0 from S to $n+1$ using $\tilde{x}_{ij}$ as the capacity for arc (i,j). Let $(S', \bar{S}')$ be the corresponding min cut. If $z^0 < 1$, then add the simple π_B-inequality induced S'.

4. If $\pi(B(S)) \neq \pi(B(S'))$, then set $S = S'$ and go to Step 2 considering the new definition for S.

5. end for

The difference lies mainly in including information about the time windows and buckets and modifying some definitions since instead of minimizing the makespan we are minimizing the time dependent travel cost of the tour (see Malandraki and Daskin [42, Section 4] for a reference).

Proposition 4.3.12. *Let $j,k \in V\backslash\{0\}$, $j \neq k, n+1$, and m a feasible time period for arc (j,k). For $w \in V$, $w \neq j$, we define*

$$\begin{aligned} I^m(j,k,w) \ = \ & \{p \text{ feasible time period for } (w,j) : \\ & \min\{T_{jk}^m, d_k - c_{jk}^m\} < \max\{r_w + p_w, T_{wj}^{p-1} + 1\} + c_{wj}^p + p_j\} \end{aligned}$$

and let $\underline{b}_{jk}^m = \min_{\beta\in S_{jk}(m)} \beta$. Then, inequality

$$\sum_{w=0, \ w\neq j}^{n} \sum_{p\in I^m(j,k,w)} x_{wj}^p + x_{jk}^m + \sum_{\beta\in B_j, \ \beta<\underline{b}_{jk}^m} y_j^\beta \leq 1 \tag{4.58}$$

is satisfied by all canonical solutions.

Proof. We first remark that in any feasible solution, from the in-degree constraints, we have that

$$\sum_{w=0, \ w\neq j}^{n} \sum_{p\in I^m(j,k,w)} x_{wj}^p \leq 1$$

It is simple to check that if $x_{wj}^p = 1$, with $p \in I^m(j,k,w)$, the time dependent path given by $P = (w,j,k)$ and $T = (p,m)$ is infeasible. Then, we can state that

$$\sum_{w=0, \ w\neq j}^{n} \sum_{p\in I^m(j,k,w)} x_{wj}^p + x_{jk}^m \leq 1.$$

We now analyze the remaining part of the inequality. Suppose $y_j^\beta = 1$ for some $\beta < \underline{b}_{jk}^m$. From constraint (4.35), $y_j^b = 0$ for $b \neq \beta, b \in B_j$. In particular, since $\beta < \underline{b}_{jk}^m$, then $y_j^b = 0$ for $b \in S_{jk}^m$ and therefore $x_{jk}^m = 0$. We are left showing that $x_{wj}^p = 0$.

For this, we consider again a canonical solution and we argue similarly to the previous cases. First we notice that since time period m for arc (j,k) is feasible, then it holds that $d_k - c_{jk}^m > T_{jk}^{m-1}$ and therefore $\min\{T_{jk}^m, d_k - c_{jk}^m\} > T_{jk}^{m-1}$. If travelling arc (w,j) in time period $p \in I^m(j,k,w)$, from definition of $I^m(j,k,w)$, the earliest departure time from vertex j is greater than or equal to $T_{jk}^{m-1}+1$. Considering also condition (4) from the bucket definition, a canonical solution has $y_j^b = 1$ with $b \geq \underline{b}_{jk}^m$, $b \in B_j$, which is a contradiction since we are assuming $y_j^\beta = 1$ for $\beta < \underline{b}_{jk}^m$ and this violates constraint (4.35). Thus, $y_j^\beta = 1$ implies $x_{jk}^m = 0$ and $x_{wj}^p = 0$, for $p \in I^m(j,k,w)$ and the inequality is valid. □

As well as with the strengthened version of TDTOURs for minimal violation paths, we can extend the definition of $I^m(j,k,w)$ including bucket information to strengthen inequalities from the previous proposition. The idea is basically the same, using information about canonical solutions of the TDTBF so as to incorporate more variables to the inequality. This is introduced in the following proposition.

Proposition 4.3.13. *Let $j,k \in V\backslash\{0\}$, $j \neq k, n+1$, and m a feasible time period for arc (j,k). For $w \in V$, $w \neq j$, we define*

$$I_B^m(j,k,w) \quad = \quad \{b \in B_w : \min\{T_{jk}^m, d_k - c_{jk}^m\} < r_b + c_{wj}^b + p_j\}$$

and let $\underline{b}_{jk}^m = \min_{\beta \in S_{jk}(m)} \beta$. Then, inequality

$$\sum_{w=0,\ w\neq j}^{n} \sum_{\beta \in I_B^m(j,k,w)} y_{wj}^\beta + x_{jk}^m + \sum_{\beta \in B_j,\ \beta < \underline{b}_{jk}^m} y_j^\beta \leq 1 \tag{4.59}$$

is satisfied by all canonical solutions.

Proof. We argue similarly to Proposition 4.3.12. First, we notice that any feasible solution that travels arc (j,k) in time period m must satisfy $t_j \leq \min\{T_{jk}^m, d_k - c_{jk}^m\}$. In addition, for any canonical solution travelling arc (w,j) in a bucket $\beta \in B_w$, we have that $r_\beta \leq t_w$. Then, considering $\beta \in I_B^m(j,k,w)$ and let $\beta \in S_{wj}^p$ for some valid p time period for arc (w,j), we can state that

$$t_j \leq \min\{T_{jk}^m, d_k - c_{jk}^m\} < r_\beta + c_{wj}^\beta + p_j \leq t_w + c_{wj}^p + p_j.$$

Therefore, vertex w cannot be immediately before j when starting at w in $\beta \in I_B^m(j,k,w)$ and it holds that

$$\sum_{w=0,\ w\neq j}^{n} \sum_{\beta \in I_B^m(j,k,w)} y_{wj}^\beta + x_{jk}^m \leq 1$$

for any canonical solution.

To prove that $y_j^b = 1$, with $b < \underline{b}_{jk}^m$, implies $x_{jk}^m = 0$ and $y_{wj}^\beta = 0$, $\beta \in I_B^m(j,k,w)$, we can use exactly the same argument considered in Proposition 4.3.13. Then, it follows that the inequality is valid. □

The following family takes into account the opposite case compared to the two presented before. Given an arc (i,j) and a feasible time period m, instead of looking at vertices and time periods arriving at i it focuses on infeasible time periods for arcs leaving j. .

Proposition 4.3.14. *Let $i,j \in V$, $i \neq j$, and m a feasible time period for arc (i,j). For $w \in V$, $w \neq j$, we define*

$$\begin{aligned} H^m(i,j,w) \;=\; & \{p \text{ feasible time period for } (j,w) : \\ & \min\{\min\{T_{jw}^p, d_j + p_j\}, d_w - c_{jw}^p\} < \max\{r_i + p_i, T_{ij}^{m-1} + 1\} + c_{ij} + p_j\} \end{aligned}$$

and $\beta_{lim} \in B_j$ such that

$$\beta_{lim} \;\geq\; \max\{\gamma_{ij}^m, \max_{\substack{1 \leq w \leq n+1,\ w \neq j \\ p \in H^m(i,j,w)}} \{\bar{b}_{jw}^p\}\}$$

where $\gamma_{ij}^m = \max_{\beta \in S_{ij}(m)} N_j(i,\beta)$ and $\bar{b}_{jw}^p = \max_{\beta \in S_{jw}(p)}\{\beta\}$. Then, inequality

$$x_{ij}^m + \sum_{w=1, w\neq j}^{n+1} \sum_{p \in H^m(i,j,w)} x_{jw}^p + \sum_{\beta > \beta_{lim}} z_j^\beta \leq 1 \tag{4.60}$$

is valid for P_2^{TD}.

Proof. Similarly to the proof of Proposition 4.3.12, we first note that

$$x_{ij}^m + \sum_{w=1, w\neq j}^{n+1} \sum_{p \in H^m(i,j,w)} x_{jw}^p \leq 1$$

since vertex j satisfies degree constraints and any time dependent path given by $P = (i,j,w)$ and $T = (m,p)$, with $p \in H^m(i,j,w)$ is infeasible because we cannot depart from j before the earliest arrival instant to j when travelling arc (i,j) in time period m.

We now analyze the remaining part of the inequality. Suppose $z_j^\beta = 1$ for some $\beta > \beta_{\text{lim}}$. From constraints 4.34 we have that $z_j^b = 0$ for $b \neq \beta$ In particular, the latest possible arrival bucket at j when travelling arc (i,j) in time period m is γ_{ij}^m, and then $x_{ij}^m = 0$ because $\gamma_{ij}^m < \beta$ by definition of β_{lim}. With an analogous argument, $y_j^b = 0$ for $b \leq \beta$ and in particular

$$\max_{\substack{1 \leq w \leq n+1,\ w \neq j \\ p \in H^m(i,j,w)}} \{\bar{b}_{jw}^p\} \leq \beta_{\text{lim}} < \beta,$$

implying that x_{jw}^p for $p \in H^m(i,j,w)$ since we cannot depart from j to w in time period p. Therefore, $z_j^\beta = 1$ implies $x_{ij}^m = 0$ and $x_{jw}^p = 0$, for $w \neq j$ and $p \in H^m(i,j,w)$, which completes the proof. □

Finally, we present another family of inequalities that exploits information provided by the inclusion of buckets in the model. If travelling an arc (i, j) starting at i in a particular bucket b and arriving at j in bucket $b' = N_j(i, b)$, then during the lapse of time between the end of b and the beginning of b' there cannot exist activity. Figure 4.4 shows a graphical interpretation of these inequalities.

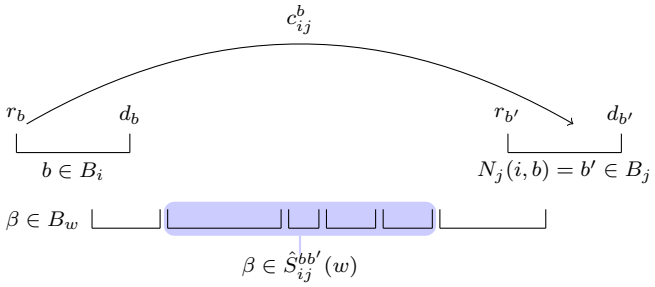


Figure 4.4.: Support for constraints (4.61).

Proposition 4.3.15. *Let $i, j, w \in V$, $i \neq j$, and buckets $b \in B_i$, $b' \in B_j$ such that $N_j(i, b) = \{b'\}$. Consider also the following set of buckets from w*

$$\hat{S}_{ij}^{bb'}(w) = \{\beta \in B_w : d_b < r_\beta \text{ and } d_\beta < r_{b'}\}.$$

Then, inequality

$$y_{ij}^b + \sum_{\beta \in \hat{S}_{ij}^{bb'}(w)} y_w^\beta \leq 1 \tag{4.61}$$

is satisfied by all canonical solutions.

Proof. First, if $y_{ij}^b = 0$, then the inequality is trivially satisfied. Otherwise, suppose $y_{ij}^b = 1$ in a canonical solution. From the proof of Proposition 4.2.2 we know that it holds that $r_b \leq t_i \leq d_b$ and $r_{b'} \leq s_j \leq t_j$, where t_i is the departure time from vertex i and s_j is the arrival time at vertex j. Since we are travelling directly from i to j, we consider the two possibilities for the position of w in the tour.

1. *Vertex w is before vertex i.* The sequences of vertices representing this scenario would be of the form $P = (0, \ldots, w, \ldots, i, j, \ldots, n+1)$. Then, the departure time from vertex w, say t_w, satisfies $t_w \leq t_i$. Let $\beta(w) \in B_w$ be the departure bucket from w in the solution -recall that it is a canonical solution-, i.e. $y_w^{\beta(w)} = 1$, we have that

$$r_{\beta(w)} \leq t_w \leq t_i \leq d_b,$$

which implies that $\beta(w) \notin \hat{S}_{ij}^{bb'}(w)$ and then $\sum_{\beta \in \hat{S}_{ij}^{bb'}(w)} y_w^\beta = 0$ by constraints (4.35).

2. *Vertex w is after vertex j.* Similarly to the previous item, the sequence of vertices representing this situation would be of the form $P = (0, \ldots, i, j, \ldots, w, \ldots, n+1)$. Then, $t_j \leq t_w$. Let again $\beta(w) \in B_w$ be the departure bucket from w in the solution. An analogous relation can be stated

$$r_{b'} \leq t_j \leq t_w \leq d_{\beta(w)},$$

which implies that $\beta(w) \notin \hat{S}_{ij}^{bb'}$ and therefore $\sum_{\beta \in \hat{S}_{ij}^{bb'}(w)} y_w^\beta = 0$.

Then, we proved that if $y_{ij}^b = 1$ the inequality is also valid, which completes the proof. □

Noticing the arguments used in the previous proof, we can strengthen the above inequalities by means of adding some extra variables y_{ij}^β. The idea is to include variables for buckets $\beta \in B_i$ that concern time instants earlier than B and arrive at j in a bucket greater than or equal to b'. This strengthened version is shown in the following proposition. The proof is omitted since it is very similar to the one from Proposition 4.3.15.

Proposition 4.3.16. *Let $i, j, w \in V$, $i \neq j$, and buckets $b \in B_i$, $b' \in B_j$ such that $N_j(i, b) = b'$ and $\hat{S}_{ij}^{bb'}(w)$ defined in Proposition 4.3.15. We further define $\hat{T}_{ij}^{bb'} \subseteq B_i$ as*

$$\hat{T}_{ij}^{bb'} = \{\beta \in B_i : \beta \leq b \text{ and } N_j(i, \beta) \geq b'\}$$

Then, inequality

$$\sum_{\beta \in \hat{T}_{ij}^{bb'}} y_{ij}^\beta + \sum_{\beta \in \hat{S}_{ij}^{bb'}(w)} y_w^\beta \leq 1 \tag{4.62}$$

is satisfied by all canonical solutions.

All families presented in this section have a polynomial number of members. For the separation algorithms, they can be easily enumerated to check whether there exists or not a violated member. In practice, these enumerations require a reasonable computational effort and, therefore, no extra developments are required.

4.4. B&C algorithm

In the following sections we describe in detail all the main parts of the B&C algorithm developed for the TDTSP-TW. As it was mentioned before, since the feasibility problem for the TDTSP-TW is $\mathcal{NP}$-Complete, we decide to develop several initial heuristics aiming to find a feasible solution before starting with the B&B enumeration. Later, we explain the preprocessing phase, applied to the original graph and therefore considered by all methods tested, in which we strengthen the time windows originally defined and remove arcs from the graph, based on the relations between time windows and time periods. Then, we proceed to describe how precedences are inferred, which involves the imposition of an extra condition to the buckets' definition. Finally, we provide details regarding the cutting plane algorithm and priorities established to select the branching variable.

4.4.1. Initial heuristic

In order to start the B&C algorithm with an upper bound, we develop three different initial heuristics aiming to find a feasible solution for the TDTSP-TW: a sorting heuristic, a nearest feasible neighbour heuristic and an insertion heuristic. A similar approach is considered in Ascheuer et al. [6] for the TSPTW. In reduced preliminary computational experiments, we noticed that counting with an initial upper bound improves indeed the computational times for all models considered. It is important to remark that the upper bounds provided by these heuristics are taken into account by all models since they are executed before starting the optimization of the corresponding MILP formulation.

Before explaining the heuristics, we first describe an algorithm used in two of them. As mentioned before, finding a feasible solution for the TDTSP-TW is an $\mathcal{NP}$-Complete problem. However, given a fixed sequence of vertices from V, deciding whether there exists or not a feasible assignment of time periods in each transition can be done by means of a simple greedy algorithm. The main idea is that, since waiting times at the departure from a vertex are allowed and the sequence of vertices is fixed, arriving at the earliest possible time to each vertex ensures that all possible departure time periods are considered. The sketch of the algorithm is shown in Algorithm 4.2.

Algorithm 4.2 Check Feasible Sequence

Input: $P = (v_1, \ldots, v_k)$ a sequence of vertices.

1. Define $s_{v_1} = r_{v_1}$

2. **for** $i = 1, \ldots, k-1$ **do**:

3. For each feasible time period m for arc (v_i, v_{i+1}) calculate the earliest departure time from v_i, $t_{v_i}^{(m)}$, and the earliest arrival time to v_{i+1} when travelling arc (v_i, v_{i+1}) in time period m, $s_{v_{i+1}}^{(m)}$.

4. Define $m_i = \arg\min_{m \text{ feasible}} \{s_{v_{i+1}}^{(m)}\}$.

5. Set $s_{v_{i+1}} = s_{v_{i+1}}^{m_i}$ and go back to 2

6. **end for**

We describe next in more detail the heuristics mentioned previously. Before starting with the resolution of the B&C algorithm we execute the three heuristics and set, if any, the one having minimum cost as the initial upper bound. In preliminary computational results, the insertion heuristic is the one that provided the best results, finding feasible solutions for the TDTSP-TW in more instances and with smaller objective value than the others.

Sorting heuristic

For this heuristic, we consider three different criteria based on the information provided by the time windows. The objective is to check whether an ordering of the vertices in V according to the information provided by the time windows results in a feasible solution for the TDTSP-TW.

We sort increasingly vertices in V according to the following values: release date r_i, due date d_i and the *mid point* of the time window, i.e. $(r_i + d_i)/2$. For these three sequences of vertices, we check the existence of a feasible assignment of time periods using Algorithm 4.2. We select the one with minimum cost.

Nearest feasible neighbour heuristic

Starting from vertex 0, we construct iteratively a time dependent path in a similar manner as in Algorithm 4.2. We select as the next vertex and time period to be included in the partial path, the i combination that minimizes the arrival time to the new vertex. For feasibility purposes, we consider as possible candidates those which, when added to the current subpath, do not prevent from being visited later, vertices not yet considered.

For this feasibility test, we consider a necessary condition which can be computed efficiently. Suppose v is the last vertex added to the partial path P and consider $w \notin P$. Let t_v be the earliest departure time from vertex v and s_w the earliest arrival time to vertex w during time period m. If a vertex $u \notin P$ satisfies $t_v < r_u, d_u < s_v$, then vertex u is not visited before v and cannot be visited after w. Under this situation, vertex w and time period m are not considered as candidates since travelling from v to w in time period m will exclude u to be visited and the sequence will not yield to a feasible solution.

The procedure stops either when all vertices in $V\backslash\{0, n+1\}$ have been added or when no feasible vertex is found to extend the current path.

Insertion heuristic

The objective of this heuristic is to construct a feasible solution by inserting vertices iteratively on a partial time dependent path. The outline of the heuristic is described as follows. For vertices not yet included, we consider a set of feasible predecessors and successors containing vertices from the partial path. For each possible combination, we try an insertion and, in case that at least one of them is feasible, we select the one with minimum cost and iterate again. It is important to remark that the heuristic constructs only feasible partial time dependent paths. Because of the time dependency factor, each transition considered in the partial paths has assigned a feasible time period. In addition, the construction of the successors' set depends on the predecessor selected for the new vertex to be added. This is because the time period chosen to travel the arc entering the new vertex influences the feasibility of its possible successors.

We proceed now to describe specifically the main parts of the heuristic. The initial path considered is $P = (0, n+1)$ and an artificial time period $T = (0)$. Since there may not exist a vertex connected to both, the initial and final depots, we will manage these two special vertices in a particular way: they will be always considered a predecessor and a successor, respectively, for new vertices to be added. However, in case no arc and time period between them exist, a cost of ∞ is assigned. Then, we will also manage the feasibility of a tour by considering the cost of the partial path constructed.

Consider a partial time dependent path $[P,T]$, with $P = (0, v_1, \ldots, v_k, n+1)$ and $T = (m_0, m_1, \ldots, m_k)$, and its corresponding earliest arrival and departure times, s_{v_i} and t_{v_i}. If arc $(0, v_1)$ is infeasible, given that we are assuming to have a complete digraph, vertex 0 is initially connected to all vertices. We can therefore use $c_{0v_1}^{m_0}$ to compute s_{v_1}. Let $v \in V \backslash P$ be a vertex not contained in the partial tour. To define the set of feasible predecessors of v, $pred(v)$, we consider vertices $u \in P$ and we use their departure times, t_u, as an estimated value to determine whether it can be considered a possible predecessor of w. Thus, we let $pred(v) = \{u \in P \ : \ s_v^u = \max\{t_u + c_{uv}^{m_u}, r_v\} \leq d_v\}$ to be the set of possible predecessors of v, where s_v^u represents the estimated arrival time to v given that we travel from $u \in P$ in time period m_u. For each vertex $u \in pred(v)$ we calculate the set of possible successors for v, denoted by $succ(v,u)$, as those vertices $w \in P$ for which there exists a feasible time period (v,w) departing from v in $s_v^u + p_v$. Specifically, $succ(v,u) = \{w \in P \ : \ \exists\ m_v$ feasible time period for $(v,w), s_v^u + p_v \leq T_{vw}^{m_v}\}$. For each combination of u, v, w, we attempt an insertion of v between u and w with transitions m_u for arc (u,v) and m_v for arc (v,w).

An insertion of the type mentioned in the previous paragraph may produce alterations to vertices in P not directly involved, since u and w are not necessarily next to each other or even u before w. For this reason, depending on the situation, the insertion is performed in the following way:

1. *Vertex u appears before vertex w.* Consider the partial time dependent path $[P,T]$, where

$$\begin{aligned} P &= (0, \ldots, v_i, v_{i+1}, \ldots, v_j, v_{j+1}, \ldots, n+1), \text{ and} \\ T &= (m_0, \ldots, m_i, m_{i+1}, \ldots, m_j, m_{j+1}, \ldots) \end{aligned}$$

   and $v_i = u$ and $v_j = w$, with $i < j$. The resulting sequence of vertices is

$$P = (0, v_1, \ldots, v_i, v, v_j, \ldots, v_{i+1}, v_{j+1}, \ldots, n+1).$$

   where the subpath $(w = v_j, \ldots, v_{i+1})$ is reversed in comparison to the original one. As regards the time periods, for the subpath $(0, \ldots, v_i = u)$ the original assignment is feasible. For $(u = v_i, v, v_j = w)$, as mentioned before, we consider (m_u, m_v). Finally, for the final subpath $(w = v_j, \ldots, v_{i+1}, v_{j+1}, \ldots, n+1)$ we consider a slightly modified version of Algorithm 4.2 where we set s_{v_j} as the earliest arrival time to vertex v_j considering the starting subpath. If combining these two subpaths we obtain a (new) feasible partial time dependent path, then we establish that the insertion is feasible and therefore considered.

2. *Vertex u appears after vertex w.* Regard again the partial time dependent path $[P,T]$, where

$$P = (0, \ldots, v_{j-1}, v_j, v_{j+1}, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, n+1), \text{ and} \tag{4.63}$$
$$T = (m_0, \ldots, m_{j-1}, m_j, m_{j+1}, \ldots, m_{i-1}, m_i, m_{i+1}, \ldots) \tag{4.64}$$

   and $v_i = u$ and $v_j = w$, with $i > j$. The sequence of vertices is modified as follows:

$$P = (0, \ldots, v_{j-1}, v_{j+1}, \ldots, v_{i-1}, v_i, v, v_j, v_{i+1}, \ldots, n+1).$$

   This case is similar to the previous one, with some minor differences. First, note that the subpath starting at vertex 0 and ending in v_i is different from the original one, since v_j has

been removed from its position. Thus, for these subpaths we consider again Algorithm 4.2 to determine whether exists or not a feasible assignment of time periods to transitions. In case it exists, the original earliest departure from vertex $v_i = u$ might be different compared to the resulting one from the assignment, since the triangle inequality does not hold, and therefore we check whether the subpath $(u = v_i, v, v_j = w)$ is feasible to be travelled in time periods (m_u, m_v). If the answer is affirmative, then we apply Algorithm 4.2 again for the ending subpath $(v_j, v_{i+1}, \ldots, n+1)$. If the combination of all these subpaths and time periods is feasible, then we consider the new partial time dependent path as feasible.

Finally, it remains to describe which feasible insertions are performed and which vertices, not yet included in the partial path, are considered. Initially, we sort increasingly vertices $v \in V \backslash \{0, n+1\}$ according to the size of its time windows, i.e. $d_v - r_v$. In case of tie, the relative order among them is also increasing but with respect to the beginning of the time window, r_v. Let L be this list. We inspect elements in L one at a time, calculating the above mentioned sets of predecessors and successors and attempting to perform an insertion for every combination. Whenever at least one feasible insertion is found, we choose the one having the minimum cost, construct the new partial path, remove the actual vertex from L and repeat the process.

Algorithm 4.3 Insertion Heuristic

Input: $D = (V, A)$, time periods limits T_{ij}^m, time windows $[r_i, d_i]$.

1. Initialize $P = (0, n+1)$ and $T = (0)$.

2. Set $L = \{v_1, v_2, \ldots, v_n\}$ as the list not visited vertices, sorted increasingly according to the size of the vertices' time windows.

3. **for** $v \in L$ **do:**

4. Compute the list of feasible predecessors $pred(v)$.

5. For each $u \in pred(v)$, compute $succ(v, u)$.

6. For each combination of $u \in pred(v)$ and $w \in succ(v, u)$, attempt an insertion (u, v, w). If there is at least one feasible, choose the one with minimum cost. Update $[P, T]$ with the new sequence, remove v from L and start again from 3. Otherwise, continue with the next vertex in L.

7. **end for**

4.4.2. Preprocessing

One of the key elements when solving a vehicle routing problem with time windows is the preprocessing phase. In this section we provide some generalizations to the time dependent case of preprocessing procedures usually considered in the literature. However, it is important to mention that, since we do not assume that the triangle inequality holds, some of them are weaker versions of the original ones.

The processing stage involves mainly two complementary phases. The first one regards reducing the size of the time windows based on possible arrival and departures from all neighbours.

The second one involves removing arcs and time periods known not to be present in any feasible solution.

Tightening time windows

In order to reduce the size of the time windows, we generalize criteria used for the TSPTW (see, for example [6]) to increase the release and decrease the due date of certain vertices. These criteria are:

1. The earliest arrival time to vertex k from any possible preceding vertex i in time period m feasible for (i,k) is greater than r_k, i.e.,

$$r_k = \max\left\{r_k, \min_{\substack{(i,k)\in A \\ m \text{ feasible}}} \left\{ \max\{r_i + p_i, T_{ik}^{m-1} + 1\} + c_{ik}^m \right\}\right\}$$

2. Due to unnecessary waiting times at a successor of a vertex, the earliest possible time to start can be increased. Note that the formula does not eliminate feasible time periods by including T_{kj}^m, since in terms of the cost of the tour it may be more convenient to wait before entering a vertex to reduce the total cost. This is given by

$$r_k = \max\left\{r_k, \min\left\{d_k, \min_{\substack{(k,j)\in A \\ m \text{ feasible}}} \{T_{kj}^m, r_j - c_{kj}^m\} - p_k\right\}\right\}$$

3. If the due date d_k is greater than the latest arrival from possible predecessors, we can decrease d_k, However, due to the possibility of waiting at the departure we need to be sure that no feasible time period is excluded. For this reason, we name α to be the instant that is the maximum beginning of a feasible time period for an outgoing arc from k within the interval $[r_k + p_k, d_k + p_k]$, i.e., $\alpha = \max_{\substack{(k,j)\in A \\ m \text{ feasible}}} \{T_{kj}^{m-1} + 1\}$. The formula is shown below.

$$d_k = \min\left\{d_k, \max\left\{r_k, \max_{\substack{(i,k)\in A \\ m \text{ feasible}}} \{\min\{T_{ik}^m, d_i + p_i\} + c_{ik}^m\}, \alpha - p_k\right\}\right\}$$

4. Finally, we analyze the latest possible departure instant from k that is feasible for satisfying the time windows of its successors. If it is smaller than d_k, we can decrease the due date.

$$d_k = \min\left\{d_k, \max\left\{r_k, \max_{\substack{(k,j)\in A \\ m \text{ feasible}}} \{T_{kj}^{m-1} + 1 - p_k, d_j - c_{kj}^m - p_k\}\right\}\right\}$$

Removing infeasible arcs

Based on the information provided by time windows and time periods, before building any of the models, we can remove some arcs and time periods to reduce the size of the problem and eliminate infeasible possibilities. We next enumerate these situations.

Let $i, j \in V\backslash\{0, n+1\}$, $j \neq i$ and $1 \leq m \leq M$. We eliminate time period m for arc (i, j) if one of the following conditions holds:

- $T_{ij}^m < r_i + p_i$, i.e., the end of time period m for arc (i, j) is smaller than the beginning of i's time window. This can be done because in this situation the vehicle will not be able to travel arc (i, j) in time period m.
- $T_{ij}^{m-1} + 1 > d_i + p_i$, i.e., the beginning of the time period is greater than the last feasible departure time instant imposed by the time window.
- $\max\{T_{ij}^{m-1} + 1, r_i + p_i\} + c_{ij}^m > d_j$, i.e., $P = (i, j)$ and $T = (m)$ is an infeasible path of size one. Even if all time periods defined for arc (i, j) satisfy this condition it does not imply that vertex i must be placed before vertex j in every solution. This is because we are not assuming that the triangle inequality holds on the travel times, and therefore, there may exist a feasible path for travelling from i to j.
- We define $t_{\text{lb}} = \min\{T_{ij}^m, d_i + p_i\}$ and $t_{\text{ub}} = \max\{r_j, \max\{T_{ij}^{m-1} + 1, r_i + p_i\} + c_{ij}^m\} + p_j$. Time instants t_{lb} and t_{ub} represent the latest feasible departure instants from i and the earliest feasible departure instant from j when travelling arc (i, j) in time period m. If for some $w \in V$, $w \neq i, j$, w satisfies that $t_{\text{lb}} < r_w$ and $d_w < t_{\text{ub}}$ then, since travel times are non-negative, travelling arc (i, j) during time period m in a solution would imply that w is not visited, and therefore is infeasible.

For vertices representing the initial and final depot, i.e. vertices 0 and $n + 1$, we first define $d_{\min} = \min_{i \in V\backslash\{0,n+1\}}\{d_i\}$ and $r_{\max} = \max_{i \in V\backslash\{0,n+1\}}\{r_i\}$. Let $i \in V\backslash\{0, n+1\}$ and $1 \leq m \leq M$. We can remove time periods for arcs of the form $(0, i)$ and $(i, n+1)$, based on the following criteria:

- If $T_{0i}^{m-1} + 1 > d_{\min}$ then travelling arc $(0, i)$ in time period m is not feasible since at least one vertex would not be able to be part of the tour.
- Similarly to the previous case, if $T_{i0}^m < r_{\max}$ then travelling arc $(i, 0)$ in time period m would prevent at least one vertex from being part of the tour.
- If $T_{0i}^{m-1} + 1 + c_{0i}^m > d_i$, then arc $(0, i)$ in time period m is an infeasible path of size one.
- We define $t_{\text{ub}}^0 = \max\{r_i, T_{0i}^{m-1} + 1 + c_{0i}^m\} + p_i$. If for some $w \in V\backslash\{0, n+1\}$, $w \neq i$, holds that $d_w < t_{\text{ub}}^0$, then vertex w cannot be visited in any tour travelling from the initial depot 0 to vertex i in time period m and therefore is infeasible.
- We define $t_{\text{lb}}^{n+1} = \min\{d_i + p_i, T_{i0}^m\}$. If for some $w \in V\backslash\{0, n+1\}$, $w \neq i$, holds that $t_{\text{lb}}^{n+1} < r_w$, then time period m for arc $(0, i)$ can be removed since it cannot be present in any feasible solution by a similar argument to the previous case.

The procedure described above is applied to the graph and therefore available for all the models considered in this chapter. However, for the special case of the TDTBF, we can use the information provided by buckets to avoid some infeasible travels. Consider again $i, j \in V$ and $b \in B_i$. By definition of $N_j(i,b)$, if travelling from vertex i to vertex j starting at i in bucket b violates the deadline for j, then $N_j(i,b)$ is empty and therefore not considered as a part of a feasible solution.

We include another condition, using a similar idea to the one described in the general case that implies $N_j(i,b)$ to be defined as empty. Let $N_j(i,b) = b' \in B_j$. We further define $t^b_{\text{ub}} = \max\{r_b + c^b_{ij}, r_{b'}\} + p_j$. From Proposition 4.2.2, we know that any canonical solution satisfies $t_i \leq d_b$ and we can also deduce that $t^b_{\text{ub}} \leq s_j + p_j \leq t_j$. If for some $w \in V\backslash\{0, n+1\}$, $w \neq i, j$, it happens that $d_b < r_w$ and $d_w < t^b_{\text{ub}}$, travelling from vertex i to vertex j departing from i in bucket b produces vertex w not to be included in the tour. Then, we can set $N_j(i,b)$ empty and do not consider this situation as a possibility. An analogous condition can be established when $i = 0$ or $j = n+1$.

Identifying infeasible time instants

As we have already mentioned, buckets' definition allows us not to consider time instants which are known to be infeasible. These infeasible time instants may provide the TDTBF with more information. For this purpose, we apply a procedure to identify such instants so as to be further exploited in the bucket definition process.

Given a vertex $i \in V\backslash\{0, n+1\}$ and t a time instant in $[r_i, d_i + p_i]$. We characterize feasible time instants by checking if it is a *feasible arrival instant* or a *feasible departure instant*, using the following rules.

- $t \in [r_i + p_i, d_i + p_i]$ is a feasible departure instant if:
  1. t is the beginning of a feasible time period m of an outgoing arc from i, or
  2. $t - p_i$ is a feasible arrival instant for vertex i and there exists $j \in V$, $j \neq i$, such that $t \in (T^{m-1}_{ij}, T^m_{ij}]$ for some feasible time period m and the arrival time at vertex j is a feasible arrival instant (or arrives before r_j).
- $t \in [r_i, d_i]$ is a feasible arrival instant if:
  1. $t = r_i$ or
  2. there exists $k \in V$, $k \neq i$, such that if departing from k in a feasible departure time instant t_k, we arrive at i in t and there exists $t' \in [r_i + p_i, d_i + p_i]$, $t' \geq t + p_i$ that is a feasible departure time instant from i.

A time instant $t \in [r_i, d_i + p_i]$ is feasible if:

- $t \in [r_i, r_i + p_i)$ and t is a feasible arrival instant, or
- $t \in [r_i + p_i, d_i]$ and t is a feasible arrival or departure instant, or
- $t \in (d_i, d_i + p_i]$ and t is a feasible departure instant.

We seek for infeasible time instants within the time window of each vertex by applying the rules defined above. The order in which vertices are examined may produce different results regarding new infeasible time instants. Therefore, the procedure is stopped after a limited number of iterations or when no new infeasible time instants are identified.

As opposed to the preprocessing described in the previous two sections, this preprocessing can be time consuming, specially for instances with large values of T and wide time windows. For this reason, we first use the procedures described in the previous two sections until no more changes are found, and then we preprocess time instants using the ideas described in this section.

4.4.3. Defining precedences

Time windows can induce precedences among vertices (as mentioned in Section 4.3.3). Including valid inequalities based on precedences as cutting planes in a B&C algorithm has been proven to be useful in the related TSPTW literature, as for example in Ascheuer et al. [6] and Dash et al. [19]. In these papers, precedences are defined assuming that the triangle inequality holds on the travel time, which is not the case of the problem under consideration in this chapter. However, we can also determine certain precedences based on the presence of time windows. Even when they are simpler and weaker than in the previous cases, in practice they produce considerable improvements in the LP relaxations as in the case of the TSPTW.

For vertices we can define precedences based on the relation between beginnings and ends of time windows. More specifically, consider $i, j \in V$, $i \neq j$, we can define $i \prec j$ if $d_i < r_j$ since travel times are non-negative. Regarding bucket-vertex precedences, we can establish similar relations, but it is necessary to impose an extra condition to the buckets' definition.

Let $i, j \in V$, $i \neq j$, and $b \in B_j$. We establish that $i \prec b$ if $d_i < r_b$. This condition is satisfied by all canonical solutions when buckets' definition satisfies conditions 1-4 from Definition 4.1. To check that in a feasible solution bucket $b \in B_j$ must appear always after any of the buckets in B_i, recall that $r_b \leq t_j$, with t_j as the earliest departure time from j in the solution. Then, arriving at i before the due date d_i is not possible since travel times are non-negative and therefore the time window of vertex i would be violated.

To determine when a bucket precedes a vertex, it would seem natural to establish that $i \prec b$ if $d_b < r_i$. However, this condition is not true if only conditions 1-4 from Definition 4.1 are considered for the buckets' definition. If a bucket $b \in B_j$ and a vertex $i \in V$ satisfy $d_b < r_i$ this does not imply that in any canonical solution b will be visited before any of the buckets in B_i. This is due to the, as called in Dash et al. [19], *negative waiting times*. Let $w_0, w_1 \in V$, $w_0 \neq w_1$, and $b_0 \in B_{w_0}$, $b_1 \in B_{w_1}$ such that $N_{w_1}(w_0, b_0) = b_1$. The negative waiting time is defined as $r_{b_0} + c^{b_0}_{w_0 w_1} - r_{b_1}$. If this value is large, it means that r_{b_1} is considerably smaller than $r_{b_0} + c^{b_0}_{w_0 w_1}$ Suppose now that we are travelling from w_1 to a vertex $v \neq w_0, w_1$ departing from w_1 at bucket b_1. The arrival bucket at v, say $b_v \in B_v$, may satisfy $d_v < r_{w_0}$ and it may appear after vertex w_0. For this reason, in order to be able to define the precedence relation $b \prec i$ as mentioned before, we define an extra condition for the buckets' definition, named *no return* condition, as shown below.

Definition 4.4 (No return condition)**.** *Let $i, k \in V$, $i \neq k$. If $r_k \in [r_i, d_i + p_i]$ and r_k is a valid time instant, then $r_{b_l^{(i)}} = r_k$ for some $b_l^{(i)} \in B_i$.*

The intuition behind this condition is to establish for all vertices a reference between the definition of the buckets and the earliest arrival and departure times of a feasible solution of the TDTSP-TW. Considering it will prevent the situation caused by negative waiting times. We prove some intermediate results before showing that precedence $b \prec i$ if $d_b < r_i$ is well defined.

Lemma 4.4.1. *Let $b \in B_i$, $b' \in B_j$ a pair of buckets such that it is feasible for b' to appear immediately after b in a canonical representation in terms of buckets of a feasible solution for the TDTSP-TW. If for some $k \in V$, $r_b \geq r_k$ and the buckets' definition satisfies conditions 1-4 from Definition 4.1 and condition from Definition 4.4, then $r_{b'} \geq r_k$ as well.*

Proof. We start by noticing that it is possible that i, j, and k are not required to be different. We divide the proof in two cases:

- $i = j$. In this case, we have that $b, b' \in B_i$ and since they are consecutive buckets in P_b, then b is the arrival bucket and b' is the departure bucket from i, satisfying constraints (4.36) that establishes a relation between arrival and departure buckets. These constraints require the departure bucket b' to satisfy at least $b' \geq b$, and then we have that $r_k \leq r_b \leq r_{b'}$.
- $i \neq j$. In this case, $b \in B_i$ is the departure bucket from i and $b' \in B_j$ is the arrival bucket to j, i.e. $N_j(i, b) = b'$. This implies that $r_k \leq r_b \leq r_b + c_{ij}^b \leq d_{b'}$ and $d_{b'-1} < r_b + c_{ij}^b \leq d_{b'}$. We analyze the following two possibilities:
  - $r_k \notin (d_{b'-1}, d_{b'}]$. From the relation deduced above, we know that $r_k \leq d_{b'-1} < r_{b'}$ by definition of buckets.
  - $r_k \in (d_{b'-1}, d_{b'}]$. By definition of buckets, the first feasible instant in $(d_{b'-1}, d_{b'}]$ is greater than or equal to $r_{b'}$. Then, if r_k is an infeasible instant not considered by b', then we can deduce that $r_k < r_{b'}$. Otherwise, if r_k is feasible, then $r_{b'} = r_k$ since the no-return holds and $r_k \in (d_{b'-1}, d_{b'}]$.

  Then, it follows that $r_k \leq r_{b'}$ if $i \neq j$.

These two cases cover all possibilities, and therefore the proof is completed. □

Based on this lemma, we now prove a more general result. The main idea is that, for any subpath of visited buckets in a canonical solution of the TDTBF, the beginning of the following bucket is at least the value of the maximum beginning of the time windows visited by the subpath.

Proposition 4.4.2. *Let $[P, T]$ be a feasible solution for the TDTSP-TW and $P_b = (b_0, b_1, \ldots, b_l)$ the sequence of buckets visited by the canonical representation of $[P, T]$. Let $\hat{P}_b = (b_s, \ldots, b_{s+k})$ be a subpath of P_b and $\hat{S} \subseteq V$ be the set of vertices visited by $\hat{P}_b$. If the buckets' definition satisfy conditions from definitions 4.1 and 4.4, then*

$$r_{b_{s+k}} \geq \max_{j \in \hat{S}} \{r_j\}. \tag{4.65}$$

Proof. We argue inductively on k. First, we consider $k = 1$, having $\hat{P}_b = (b_s, b_{s+1})$ with $b_s \in B_i$ and $b_{s+1} \in B_j$. From the buckets' definition, we know that $r_{b_s} \geq r_i$ and $r_{b_{s+1}} \geq r_j$. Then, by Lemma 4.4.1 we conclude that $r_{b_{s+1}} \geq \max\{r_i, r_j\}$.

Now suppose that the property holds for $\hat{P}'_b = (b_s, \ldots, b_{s+k-1})$ and let $\hat{S}' \subseteq V$ be the set of vertices visited by $\hat{P}'_b$. We show that it is also valid for $\hat{P}_b$. We suppose that $b_{s+k} \in B_w$, for some $w \in V$. By inductive hypothesis, we have that

$$r_{b_{s+k-1}} \geq \max_{j \in \hat{S}'}\{r_j\}.$$

We also know, by the definition of b_{s+k}, that $r_w \leq r_{b_{s+k}}$. Then, considering again Lemma 4.4.1, we have that

$$r_{b_{s+k}} \geq \max_{j \in \hat{S}'}\{r_j\}$$

and since $\hat{S} = \hat{S}' \cup \{w\}$, the property is also valid for $\hat{P}_b$. □

If buckets also satisfy the condition from Definition 4.4, from (4.65) we deduce that any bucket b which is visited after a particular vertex i satisfies that $r_i \leq r_b$. Therefore, if the upper limit of b satisfies $d_b < r_i$, then it cannot be present after any of the buckets in B_i and precedence $b \prec i$ is well defined.

4.4.4. Definition of buckets

As explained in Section 4.2.3, the buckets' definition is left open as long as it satisfies conditions 1-4 from Definition 4.1 and the no-return condition from Definition 4.4 (if the Bucket SOP inequalities are to be included as cutting planes). In Dash et al. [19] the authors propose a procedure to iteratively partition buckets aiming to increase the objective value of the LP relaxation, named *Iterative Bucket Refinement Heuristic* (IBR). Starting with an initial set of buckets defined for each vertex, the IBR solves the LP relaxation of the TDTBF (without considering SEC and TDIPEC) and, based on the fractional optimal solution, it partitions some particular buckets by analyzing the negative waiting times.

Specifically, following most of the notation from Dash et al. [19], a collection of buckets can be refined by means of Algorithm 4.4, obtaining as a result a set of improved buckets. Indeed, the complete procedure suggested by the authors executes this algorithm iteratively and every five iterations an improvement check is performed to determine whether to update or not the collection of buckets. If the upper integer part of the objective value of the new LP relaxation has been increased at least in one, the collection of buckets is updated and the procedure continues. Otherwise, the process is stopped and the last updated collection is considered for the B&C algorithm.

The results reported by Dash et al. regarding the IBR are really good, obtaining remarkable improvements in the LP relaxations as well as in the gaps after the cutting phase at the root node of the B&C enumeration tree. As a counterpart, LP relaxations become harder to solve and more time consuming, since the IBR produces a considerable increment in the number of variables and constraints in the model. From our perspective, an important factor for such

Algorithm 4.4 Bucket Refinement Heuristic (Dash et al. [19])

Input: Initial set of buckets $\mathcal{B}_0$.
Output: New set of buckets $\mathcal{B}_1$.

1. Solve the LP relaxation for the TDTBF generated from $\mathcal{B}_0$. Let $\bar{\boldsymbol{x}}$ be the optimal solution.

2. Define $\hat{B} = \{b \in \mathcal{B}_0 \ : \ \bar{z}_i^b > 0, \ b \in B_i\}$ as the set of bucket with non-zero arrival activity. For buckets $b \in \mathcal{B}_0$, $b \notin \hat{B}$, add them to $\mathcal{B}_1$.

3. **for** $b \in \hat{B}$ **do:**

4. For $t \in [r_b, d_b]$, $b \in B_i$, define $B^-(t) = \{(k, b') \ : \ b' \in B_k, N_i(k, b') = \{b\}, r_{b'} + c_{ki}^{b'} = t\}$ and decompose $\bar{z}_i^b$ into

$$\bar{z}_i^t = \sum_{(k,b') \in B^-(t)} \bar{y}_{ki}^{b'}, \text{ for } t = r_b + 1, \ldots, d_b$$

and $\bar{z}_i^{r_b} = \bar{z}_i^b - \sum_{t=r_b+1}^{d_b} \bar{z}_i^t$.

5. Choose the break point τ to minimize the function $\alpha(b, \tau)$ defined as

$$\alpha(b, \tau) = \sum_{t=r_b}^{\tau-1} (\tau - r_b)\bar{z}_i^t + \sum_{t=\tau}^{d_b} (t - \tau)\bar{z}_i^t.$$

6. Define new buckets $b^1 = [r_b, \tau - 1]$ and $b^2 = [\tau, d_b]$. Add b^1, b^2 to $\mathcal{B}_1$.

7. **end for**

success lies in the bucket preprocessing phase as well as in the effectiveness of the Bucket SOP inequalities when used as cutting planes.

For the TDTSP-TW, the bucket preprocessing is not as powerful and precedences inferred are weaker than in Dash et al. [19] since we do not assume that the triangle inequality holds, as mentioned in sections 4.4.2 and 4.4.3. In addition, the TDTBF presents some relevant differences compared to the TBF, in particular for the inclusion of variables y_i^b and constraints (4.36) to model the waiting times at the departure. For these reasons, we consider different alternatives for the buckets' definition in order to evaluate and analyze the tradeoff between the increase on the difficulty of solving the LP relaxation and the improvements in terms of gaps of the lower bounds obtained. Preprocessing regarding the tightening of time windows and the arcs removal described in Section 4.4.2 is considered in all cases. These alternatives are:

1. *Basic buckets definition (TDTBF-B):* We define buckets imposing only conditions in Definition 4.1. In this case, Simple Bucket SOP inequalities cannot be considered. Infeasible time instants identified in the preprocessing described in Section 4.4.2 is not included (i.e., the set of buckets for vertex i covers all time instants in $[r_i, d_i + p_i]$).

2. *Simple buckets definition (TDTBF-S):* Buckets' definition satisfy conditions from definitions 4.1 and 4.4. In this case, Simple Bucket SOP inequalities are included as cutting planes. As for TDTBF-B, preprocessing from Section 4.4.2 regarding infeasible time instants is not included in the buckets' definition.

3. *IBR-k:* For each vertex $i \in V$, we define the initial set of buckets as collections of consecutive feasible time instants based on the information provided by the preprocessing described in Section 4.4.2, imposing also buckets to satisfy conditions from definitions 4.1 and 4.4. For this collection, we apply the IBR heuristic with a maximum of k iterations and consider the result for the B&C algorithm. For the computational experiments, the possible values of k are $5, 10, 20$.

Each of these alternatives has different characteristics regarding the computational effort required to solve the LP relaxation and the quality of the lower bounds produced. Then, as we explain in the following section, different cut strategies are considered depending on the case.

4.4.5. Cutting planes

In Dash et al. [19] the authors perform a comparison showing that cuts using bucket information produce better results than traditional cuts. In particular, they compare the effectiveness between using Bucket SOP inequalities (simple π_B, σ_B, $(\pi, \sigma)_B$-inequalities and a bucket version of TOURs) and traditional cuts not using this information (SOPs and TOURs). Based on these results, we decided to follow a similar approach and for the TDTBF we consider the strengthened version using bucket information from the formulation.

The strategy adopted for the definition of buckets generates a formulation with particular characteristics that have to be considered for the development of a B&C algorithm. For instance, a formulation having a large number of buckets for each vertex's time window might produce tight LP relaxations at the cost of having a considerable increment on the number of variables and, consequently, on the LP resolution times. For this reason, we perform preliminary computational experiments over a restricted set of instances to evaluate different cutting strategies for each definition of buckets proposed in the previous section.

For TDTBF-B we only consider SEC and TDTOURs. The latter are only separated when no violated SEC has been detected. Bucket SOP inequalities cannot be included in this case since the no-return condition from Definition 4.4 is not satisfied by the definition of the buckets.

Regarding TDTBF-S and IBR-5 we consider a similar approach. The idea is to be aggressive with cuts only at the root node and consider inequalities with less time consuming separations in the internal nodes. The main difference between the cutting at the root node and the internal ones lies on the Bucket SOP inequalities. When the LP relaxations are relatively easy to calculate and in general the B&C tree tends to enumerate a large number of nodes, the separation procedures for these inequalities tend to require a considerable amount of time. For the root node, inequalities are considered in the following order:

1. Simple π_B and σ_B-inequalities. If no violated cut has been found, look for a simple $(\pi, \sigma)_B$-inequality.
2. The bucket version of the strengthened TDTOURs (4.51).
3. Inequalities (4.59), (4.60) and (4.62).

For the internal nodes the approach is similar. The only differences are that in step 1 we consider SEC instead of the Bucket SOP inequalities and that steps 2 and 3 are swapped. We set a maximum of 20 iterations for the root node and perform only one round at the internal nodes. However, if reaching these limits we obtain an integer feasible solution, we still have to check that the solution is feasible (i.e., it does not contain subtours and the earliest departure and arrival times are feasible).

As regards IBR-10 and IBR-20, we consider a more aggressive approach regarding the cutting planes. The cutting plane algorithm at the root is the same as for TDTBF-S and IBR-5. In the internal nodes, we replace step 1 by

1. Look for a violated SEC. If no violated cut has been found, look for a simple π_B-inequality. Again, if no violated cut has been found, look for a simple σ_B-inequality.

Similarly to TDTBF-S and IBR-5, the order of steps 2 and 3 are swapped and the number of rounds is the same. With respect to the number of iterations of the cutting plane algorithm, the settings are exactly the same as for TDTBF-S and IBR-5.

In all cases, SEC are separated considering Padberg and Rinaldi [49] procedure. For TDTBF-S and IBR-k, whenever a violated inequality (4.51) is found we check whether it can be strengthened to a TOUR constraint using Algorithm 4.2.

4.4.6. Branching

By analyzing the constraints defining the TDTBF, we can easily appreciate that integrality conditions for some variables can be relaxed. In particular, if variables z_i^b and y_i^b are defined as continuous, its integrality is implied by the variables y_{ij}^b and the assignment constraints (4.38) and (4.37), respectively. A similar argument can be used for variables x_{ij}^m, considered constraints (4.39), but in preliminary computational experiments this did not prove to be a good strategy.

Another possibility is to define variables x_{ij}^m as binary, and let y_{ij}^b, z_i^b and y_i^b variables as continuous. As long as x_{ij}^m is binary, fractional values of bucket variables will indicate the activity of the corresponding bucket (recall that a time period m for an arc $(i,j) \in A$ is defined by a unique set of buckets $S_{ij}(m)$). In this case, bucket variables can be seen as a refinement to increase the objective value of the LP relaxation. However, based on some restricted preliminary computational experiments, we observed that defining y_{ij}^b as continuous variables increases the time required for solving the LP relaxations since some LP preprocessing cannot be applied. Then, we decided to define x_{ij}^m and y_{ij}^b as binary variables while z_i^b and y_i^b as continuous ones.

Regarding the branching strategy, the B&C algorithm considers CPLEX default setting. However, for variables defined as binary we establish a priority criterion for the branching variable selection which showed to improve the overall computational times in preliminary computational experiments. As mentioned in the previous paragraph, the outcome of the algorithm can be expressed only in terms of variables x_{ij}^m. Then, we set for them a higher priority than to y_{ij}^b variables, forcing the branching variable selection to consider first x_{ij}^m variables.

4.5. Computational results

We conducted experiments to compare the different models and algorithms presented in this chapter. Algorithms were coded in `C++` using CPLEX 12.2 [16] callable library and tests were run on a workstation with an Intel i3-350 CPU, 4 Gb of RAM and running Ubuntu Linux 10.04.

In order to evaluate the performance of the algorithms, we consider the following sets of instances based on approaches from the related literature:

- *Group 1:* TDTSP-TW instances from Albiach et al. [3] for 1 minute discretization. In the original instances, the number of vertices is $n = 10, 20, \ldots, 60$ and the maximum values for the size of the time windows are $20, 40, 60$, with five instances for each value (named pnwX, with n the number of vertices). The number of time periods defined for these instances is eight ($M = 8$). Time dependent travel times are calculated multiplying the (integer) distance between two vertices by a coefficient assigned to each time period, as described in Table 4.1 and rounding the result. We consider also instances with time windows of maximum size 80 and 120, obtained doubling the size of the ones with 40 and 60, respectively (named pnwXA). We further extend these instances considering sixteen time periods constructed by dividing each of the originals by half and defining two time periods. The definition of each of them is shown in Table 4.1. In all cases, time-dependent travel costs are set as the travel time for the corresponding time period, i.e. $c_{ij}^m = \theta_{ij}^m$, and the time periods changes are the same for each arc ($T_{ij}^m = T^m$).

- *Group 2:* In a similar fashion as with Albiach et al. instances, we consider TSPTW instances from Dumas et al. [20] for $n = 20, 40, 60, 80, 100, 150$ and the width of the time windows are $w = 20, 40, 60, 80, 100$. Since some combinations of n and w are not available, we further consider some of the extended versions of these instances from Gendreau et al. [26] for $n = 80, w = 100$ and $n = 100, w = 80, 100$, producing a total amount of 140 instances. This dataset contains five instances for each combination of n and w. Eight time periods are considered, defined as in the case of Albiach et al. instances. The definition of the time periods follows the pattern of Table 4.1 for $M = 8$, rescaling proportionally the limits for each time period considering the particular value of T in the instance[2].

As regards the models and algorithms, we consider the following alternatives:

- MD: Refers the modified version of Malandraki and Daskin [42] model from Stecco et al. [57] presented in Section 4.1.1. This model is solved by means of CPLEX 12 default algorithm for MILPs.

- SCM: Refers to the formulation from Stecco et al. [57] presented in Section 4.1.1. As well as MD, this model is solved considering CPLEX 12 default algorithm for MILPs.

- TDIPF: Considers the time-dependent infeasible path formulation from Section 4.2.2. The B&C algorithm considers only SEC and TDTOURs. All CPLEX cuts and heuristics are disabled.

[2]This data set was obtained from the personal webpage of the second author of [48]. We observed that in these instances w stands for the average width of the time windows instead of the maximum.

- TDTBF: Refers to the formulation presented in Section 4.2.3. We report results for the B&C algorithm for versions described in Section 4.2.2, (TDTBF-B, TDTBF-S and IBR-k, with $k = 5, 10, 20$), each of them with its corresponding cutting plane algorithm as described in Section 4.4.5. In the five cases, all CPLEX cuts and heuristics are disabled.

$M = 8$			$M = 16$					
TP	Limits	Coef.	TP	Limits	Coef.	TP	Limits	Coef.
1	[0,99]	1.00	1	[0.49]	0.900	9	[330,384]	0.825
2	[100,219]	0.50	2	[50,99]	1.000	10	[385,439]	1.000
3	[220,279]	0.75	3	[100,159]	0.750	11	[440,469]	0.750
4	[280,329]	0.65	4	[160,219]	0.500	12	[470,499]	0.500
5	[330,439]	1.00	5	[220,249]	0.625	13	[500,569]	0.625
6	[440,499]	0.50	6	[250,279]	0.750	14	[570,639]	0.750
7	[500,639]	0.75	7	[280,304]	0.700	15	[640,679]	0.875
8	[640,720]	1.00	8	[305,329]	0.650	16	[680,720]	1.000

Table 4.1.: Definition of time periods for Albiach et al. instances for $M = 8, 16$.

For all methods we report the average percentage gap of the LP relaxation (%lpG), the average computational times[3] (in seconds) for the corresponding B&C algorithm and the number of tree nodes explored. In addition, for formulations TDIPF and the ones based on the TDTBF, we also report the average final gap at the root node (%rG) after the cutting phase. Percentage gaps are calculated as $100 * (BESTUB - LB)/BESTUB$, where $BESTUB$ is the objective value of the best solution achieved considering all the algorithms. For the computational times, a cell filled with $(***)$ means that the algorithm cannot solve any of the corresponding instances to optimality within 1800 seconds. A number between parenthesis represents the number of instances that are solved to optimality within the time limit. The averages of computational times and number of tree nodes explored are calculated over all the instances solved by the corresponding algorithm within the time limit.

4.5.1. Evaluation of models

We begin showing a comparison of the different models presented in this chapter, which in early stages of this research led us to pay special attention to the TDTBF. In tables 4.2 and 4.3 we show the results for MD, SCM, TDIPF and TDTBF-B over instances in Group 1 for $M = 8$. Instances with less than 30 vertices are not reported since they are easily solved by all algorithms and do not provide useful information.

In Table 4.2 we present the average gaps of the lower bounds produced by each formulation. As expected, in all cases the gaps tend to increase as the times windows become wider and, to a lesser extent, when the number of vertices grows. With respect to the linear relaxations, results for TDIPF represent the objective value of the underlying assignment problem. Both MD and SCM produce, in general, similar lower bounds, although in some of them the former provides slightly better results. From the table under consideration we can also appreciate that LP

[3]For IBR-k algorithms, the times required by the refinement heuristic are not included in the values reported. We analyze them separately in Section 4.5.3.

relaxations produced by the TDTBF are the best ones, obtaining in several cases improvements of 2-3% more than the other algorithms. For TDIPF and TDTBF-B, the cutting plane algorithm provides reasonable results reducing the gap at the root node after the cutting plane algorithm. These improvements have a greater impact on TDTBF-B than in TDIPF.

Inst.	n	w	MD	SCM	TDIPF		TDTBF-B	
			%lpLB	%lpLB	%lpLB	%rLB	%lpLB	%rLB
p30w20X		20	0.46	0.46	0.46	0.00	0.46	0.00
p30w40X		40	6.03	6.08	6.08	0.27	4.35	0.18
p30w60X	30	60	6.32	6.35	6.35	0.82	5.33	0.45
p30w40XA		80	10.75	10.85	10.85	3.04	10.25	2.91
p30w60XA		120	13.37	13.44	13.44	4.49	12.84	4.46
p40w20X		20	4.13	4.17	4.17	0.06	4.08	0.06
p40w40X		40	7.30	7.36	7.36	1.75	6.13	1.48
p40w60X	40	60	14.20	14.30	14.30	5.77	13.64	5.42
p40w40XA		80	12.51	12.57	12.57	4.14	11.20	3.03
p40w60XA		120	16.13	16.30	16.30	5.05	14.01	3.66
p50w20X		20	3.69	3.71	3.71	1.19	3.54	1.10
p50w40X		40	8.05	8.09	8.09	1.54	7.42	1.11
p50w60X	50	60	13.05	13.18	13.21	4.36	11.30	3.44
p50w40XA		80	19.02	19.11	19.11	6.89	16.98	4.93
p50w60XA		120	19.44	19.61	19.65	8.95	16.11	6.68
p60w20X		20	5.76	5.79	5.79	1.05	5.53	1.10
p60w40X		40	9.65	9.70	9.70	2.59	8.68	2.15
p60w60X	60	60	10.24	10.33	10.33	3.08	9.57	2.46
p60w40XA		80	19.02	19.13	19.13	8.95	16.38	6.71
p60w60XA		120	18.09	18.20	18.25	7.91	15.91	5.68

Table 4.2.: Average % GAPs on Albiach et al. [3] instances with $M = 8$.

The results for the B&C algorithms are presented in Table 4.3. Instances having less than 40 vertices or with $w \leq 60$ are solved by all algorithms. However, SCM shows greater overall resolution times compared to the other algorithms.

The most interesting results are the ones for $n = 50, 60$ and $w = 80, 120$. For all algorithms, the running time as well as the number of nodes explored increase considerably. Both SCM and TDIPF cannot solve many of these instances within the time limit imposed. The best results are produced by TDTBF-B, which is the only one capable of solving all instances in the set in less time than the others. The performance of MD is reasonable, not being able to solve only three of the instances.

The performance of TDTBF-B can be linked with the gaps reported in Table 4.2. Even when considering only SEC and TDTOURs as cuts, the gaps obtained at the root node help in reducing the size of the B&C tree. Furthermore, for $n = 60$ and $w = 80, 120$ it is able to cover a large number of nodes since the LP relaxations can be solved in a reasonable time. In several cases, even when solving less instances than TDTBF-B the number of nodes explored by the other methods is higher.

Concerning the computational times, a similar analysis can be done. TDTBF-B outperforms the other algorithms and, even when the averages are calculated over all the instances solved by the corresponding algorithm, computational times are smaller (except for SCM and TDIPF for $n = 60, w = 80, 120$, where they solve only a few of the instances and the times are smaller).

To summarize, TDTBF-B clearly appears as the best approach, solving more instances than the other methods with less computational effort. For this reason, in the next section we present more experiments with different approaches based on the TDTBF that incorporate the the specific developments from sections 4.3 and 4.4.

Inst.	n	w	MD		SCM		TDIPF		TDTBF-B	
			Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes
p30w20X		20	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00
p30w40X		40	0.02	1.60	0.07	6.40	0.01	2.40	0.01	1.60
p30w60X	30	60	0.03	17.20	0.15	105.20	0.01	7.40	0.01	4.40
p30w40XA		80	0.34	307.40	3.56	1487.60	0.11	141.40	0.14	126.80
p30w60XA		120	1.09	1088.80	16.81	7785.20	0.57	591.60	0.33	248.40
p40w20X		20	0.02	0.00	0.03	0.00	0.01	2.20	0.01	1.00
p40w40X		40	0.19	119.40	12.06	4647.60	0.13	83.00	0.16	95.20
p40w60X	40	60	4.10	4425.60	131.20	53491.80	2.55	1906.80	4.61	3066.00
p40w40XA		80	8.53	5643.80	296.48	42515.00	1.64	873.00	0.55	246.00
p40w60XA		120	7.47	3220.20	378.27	44765.40	4.82	1557.40	3.80	1103.60
p50w20X		20	0.02	0.00	0.07	1.60	0.03	9.80	0.04	12.40
p50w40X		40	0.39	285.40	2.09	691.80	0.19	94.40	0.16	46.60
p50w60X	50	60	0.98	871.80	23.38	5588.40	1.38	529.80	1.66	511.40
p50w40XA		80	37.04	18763.60	(3) 582.90	63817.00	48.23	13927.80	12.64	2747.20
p50w60XA		120	(4) 99.00	18735.25	(2) 727.34	50388.50	(3) 49.18	8111.67	22.56	2828.20
p60w20X		20	0.05	1.00	0.24	37.20	0.07	13.40	0.11	17.80
p60w40X		40	3.39	4341.40	52.20	8705.80	1.02	440.40	1.07	379.60
p60w60X	60	60	5.38	3069.00	24.12	3213.25	1.42	381.75	1.01	217.00
p60w40XA		80	(4) 173.47	56365.75	(1) 1238.54	99880.00	(3) 49.12	110273.20	128.98	16396.60
p60w60XA		120	(4) 771.08	124174.67	(2) 32.71	1746.00	(2) 1.72	216.00	570.29	22987.00

Table 4.3.: Average computational times (in seconds) and number of tree nodes explored on Albiach et al. [3] instances with $M = 8$.

4.5.2. Evaluation of new inequalities

In this section we present computational results to evaluate the inclusion of the new inequalities proposed in this chapter, i.e., the strengthened version of TDTOURs (4.51) and constraints (4.59), (4.60) and (4.62). For this purpose and based on the results from the previous section, we set an intermediate level of bucket refinement (5 iterations) and consider SEC instead of Bucket SOP constraints. There is no maximum number of cutting planes iterations and the buckets definition does not satisfy the *no-return* condition from Definition 4.4. We consider augmented instances in Group 1 for $M = 8$. In Table 4.4 we report the %rG, the overall

computational times (in seconds) and the number of nodes explored in the B&C tree for three different configurations of cuts:

1. *Conf-1*: SEC and TDTOURs,
2. *Conf-2*: SEC and the strengthened version of TDTOURS (4.51), and
3. *Conf-3*: SEC, constraints (4.51), (4.59), (4.60) and (4.62).

We remark that for Conf-2 and Conf-3, when a violated constraint (4.51) is identified, Algorithm 4.2 is used to check whether a TOUR constraint can be included.

Inst.	n	w	Conf-1			Conf-2			Conf-3		
			%rG	Time	Nodes	%rG	Time	Nodes	%rG	Time	Nodes
p30w40XA	30	80	2.34	0.22	59.60	2.34	0.22	67.40	2.08	0.24	46.60
p30w60XA		120	3.63	0.44	135.60	3.63	0.40	109.20	3.34	0.43	102.00
p40w40XA	40	80	2.57	1.15	205.80	2.57	0.97	152.80	2.36	1.38	173.40
p40w60XA		120	3.59	5.57	700.20	3.58	5.02	671.00	3.45	4.77	543.80
p50w40XA	50	80	4.63	10.08	1509.00	4.63	7.62	991.80	4.02	4.92	465.60
p50w60XA		120	5.90	35.81	1805.00	5.88	31.90	1650.80	5.66	19.34	939.40
p60w40XA	60	80	5.10	74.85	5322.00	5.10	39.54	3291.00	4.40	43.72	2217.40
p60w60XA		120	5.20	(4) 227.74	8775.25	5.20	278.13	7044.00	4.77	186.77	5071.25

Table 4.4.: Evaluation of cuts.

First we observe that Conf-1 is the only configuration not capable of solving all instances. In comparison, Conf-2 solves all instances, reduces the average computational times (147.46 seconds if considering only instances solved by both configurations) and also the number of nodes explored. Regarding %rG, we cannot observe major differences. This means that most of the improvements at the root node are provided by SEC, but constraints (4.51) and TOURs show to be useful in the internal nodes of the B&C tree.

Similar remarks can be done for the comparison between Conf-2 and Conf-3. The average computational times are considerably reduced in almost all cases, with the exception of $n = 60$ and $w = 80$, where they are comparable. This small difference mainly due to only one instance. The number of tree nodes explored is also reduced in almost all cases. As regards the %rG, we can appreciate some improvements in the gaps compared to Conf-2 (about 0.5% smaller). This is caused by the inclusion of constraints (4.59), (4.60) and (4.62) which, in conjunction with the strengthened TDTOURs, justify the reduction on the number of nodes explored.

4.5.3. TDTBF B&C algorithms

In this section we test the four different alternatives mentioned before based the TDTBF, that are: TDTBF-S and IBK-k, for $k = 5, 10, 20$. Results for TDTBF-B are also reported for comparison purposes, in order to have an idea of the improvements obtained in each case. Regarding the instances, from Group 1 we consider the augmented ones (i.e., $w = 80, 100$) for $M = 8, 16$ and all instances in Group 2. As seen in the previous section, TDTBF-B solves easily the other instances in Group 1 and the same occurs for the other versions considered in this section.

In Table 4.5 we present the results for the average percentage gaps of the LP relaxation and the cutting phase at the root node for the instances from Group 1 considered. We begin

comparing the results for TDTBF-B and TDTBF-S. The most relevant differences can be seen in the gaps after the cutting plane algorithm, where TDTBF-S reduces it in a 78 % for $M = 8$ and 74 % for $M = 16$ on average.The reason for this reduction is the inclusion of bucket-based inequalities, where the best improvements are mainly due to the Bucket SOP. With respect to the linear relaxations, recall that the only difference between TDTBF-B and TDTBF-S is that for the latter the buckets' definition also satisfy the no-return condition from Definition 4.4. The inclusion of these conditions produces in some cases small improvements in the LP relaxation of the TDTBF-S.

Regarding the approaches based on the IBR, we can observe how the LP relaxations are improved as the number of iterations increases. For IBR-5 we appreciate the most noticeable reduction, although from IBR-5 to IBR-10 and IBR-10 to IBR-20 there are some improvements too. Related to this, in some cases the IBR stops before reaching the limit of k iterations since no considerable improvement has been achieved. For IBR-10, the best reductions in %lpG are obtained with $n \geq 40$, since for smaller values of $n = 30$ the results are not so important. Similarly, for IBR-20, the best gains appear for $n = 50, 60$, which are near to 0.4 %.

Analogous observations can be made concerning the gaps after the cutting plane algorithm. Increasing the maximum number of iterations allowed to the IBR produces improvements but these become less evident. In particular, this effect can be appreciated better when considering the lower bounds in the root node. For instance, for $M = 16$, $n = 60$ and $w = 80$, the average improvement in %rG is 0.62% from TDTBF-S to IBR-5 and 0.33% from IBR-5 to IBR-10. In fact, the improvement in %rG is less than 1% from TDTBF-S to IBR-10 while for the LP relaxations the average improvement is close to the 6%. This behaviour may be due to the definition of the bucket-vertex precedence relations, because in all cases -except TDTBF-B- the cutting plane algorithm at the root node is the same. These relations are weaker than in the case of Dash et al. [19] since we are not assuming that the triangle inequality holds on the travel times.

The results for the B&C algorithm are presented in Table 4.6. First, we can observe that TDTBF-S and IBR-k algorithms are able to solve all instances for both $M = 8, 16$. We can observe an increment of the computational times as k increases. Considering that for almost all combinations of n and w on average the computational times required for solving these instances are less than a minute, this behaviour seems reasonable. For IBR-10 and IBR-20, the LP relaxations become harder to solve and the cutting plane algorithms of the internal nodes are more time consuming. This can be also appreciated on the average of nodes explored. In general, the greater the value of k, the smaller the number of enumerated nodes. However, for these instances TDTBF-S as well as IBR-k algorithms produce both reasonable results. Many of these effects and situations are more visible and its impact is clearer on instances from Group 2.

Finally, we analyze the TDTBF-B. The results for $M = 8$ are the same as in Table 4.3. For the case of $M = 16$, we can see that it is not capable of solving one of the instances for $n = 60$ and $w = 120$. Regarding the impact of increasing the number of time periods M, results are somehow mixed. With the exception of $n = 60$ and $w = 80$, average computational times for $M = 16$ are slightly greater than for $M = 8$. In addition to the unsolved instance, for $M = 16$ the number of nodes explored is increased in at least a 30% for $n = 40$ and $n = 50, w = 80$.

M	Inst.	n	w	TDTBF-B		TDTBF-S		IBR-5		IBR-10		IBR-20	
				%lpG	%rG	%lpG	%rG	%lpG	%rG	%lpG	%rG	%lpG	%rG
8	p30w40XA	30	80	10.25	2.91	10.13	0.36	8.25	0.24	7.39	0.24	7.31	0.24
	p30w60XA		120	12.84	4.46	12.84	0.56	9.91	0.43	9.08	0.43	8.89	0.39
	p40w40XA	40	80	11.20	3.03	11.20	0.75	8.75	0.56	8.04	0.51	7.77	0.51
	p40w60XA		120	14.01	3.66	14.01	0.92	12.38	0.88	11.05	0.74	10.99	0.73
	p50w40XA	50	80	16.98	4.93	16.71	1.23	12.75	0.96	11.44	0.42	11.20	0.39
	p50w60XA		120	16.11	6.68	16.00	1.26	12.73	0.91	12.16	0.70	11.79	0.70
	p60w40XA	60	80	16.38	6.71	15.93	2.04	12.01	1.11	11.03	0.91	10.63	0.91
	p60w60XA		120	15.91	5.68	15.59	1.57	11.70	1.28	11.18	1.00	10.70	0.91
16	p30w40XA	30	80	10.33	2.61	10.33	0.17	7.83	0.17	7.46	0.12	7.42	0.12
	p30w60XA		120	11.80	2.82	11.80	0.41	9.12	0.24	8.43	0.24	8.35	0.24
	p40w40XA	40	80	10.88	2.35	10.88	0.73	8.43	0.58	7.38	0.56	7.21	0.56
	p40w60XA		120	14.90	4.68	14.81	1.62	13.02	1.46	11.65	1.23	11.35	1.23
	p50w40XA	50	80	16.77	4.46	16.77	1.25	12.38	0.96	10.84	0.75	10.44	0.71
	p50w60XA		120	15.05	5.11	14.87	1.75	12.06	1.29	11.20	1.09	10.86	0.94
	p60w40XA	60	80	17.33	6.75	16.56	1.69	12.77	1.07	10.70	0.74	10.31	0.66
	p60w60XA		120	17.52	7.78	17.20	2.44	13.67	1.90	12.84	1.74	12.49	1.71

Table 4.5.: Average % GAPs on Albiach et al. [3] instances for TDTBF-based algorithms.

M	Inst.	n	w	TDTBF-B		TDTBF-S		IBR-5		IBR-10		IBR-20	
				Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes
8	p30w40XA	30	80	0.14	126.80	0.43	21.80	0.63	10.00	0.62	3.60	0.67	3.60
	p30w60XA		120	0.33	248.40	0.44	6.40	0.55	6.00	1.00	7.00	1.30	7.40
	p40w40XA	40	80	0.55	246.00	1.09	29.80	1.52	9.80	2.42	14.60	3.01	14.80
	p40w60XA		120	3.80	1103.60	2.66	79.80	5.63	109.60	6.39	75.40	8.46	79.00
	p50w40XA	50	80	12.64	2747.20	3.54	201.60	4.28	84.40	7.38	48.20	9.54	35.00
	p50w60XA		120	22.56	2828.20	8.04	114.00	12.30	110.20	20.38	44.80	39.00	49.40
	p60w40XA	60	80	128.98	16396.60	13.09	465.60	21.46	273.00	15.81	87.60	30.33	87.80
	p60w60XA		120	570.29	22987.00	20.79	220.75	35.69	123.25	50.13	88.75	142.15	79.25
16	p30w40XA	30	80	0.20	113.80	0.36	2.00	0.46	1.80	0.54	2.40	0.62	2.40
	p30w60XA		120	0.47	248.00	0.51	6.00	0.54	7.60	0.78	2.60	1.00	2.60
	p40w40XA	40	80	1.83	501.80	1.37	38.20	1.50	12.00	2.59	12.40	3.24	12.40
	p40w60XA		120	7.21	1457.40	3.22	99.00	6.21	83.40	5.18	23.40	11.48	20.00
	p50w40XA	50	80	14.26	3749.80	3.26	100.60	5.05	46.00	6.31	16.20	9.66	12.40
	p50w60XA		120	24.37	2352.80	9.76	183.80	17.28	145.60	17.35	25.60	28.62	35.00
	p60w40XA	60	80	97.39	14682.60	13.96	281.40	16.12	93.20	18.66	45.00	33.59	48.80
	p60w60XA		120	(4) 100.16	5995.33	74.65	884.25	63.29	319.00	58.71	159.75	63.14	135.00

Table 4.6.: Average computational times (in seconds) and number of tree nodes explored on Albiach et al. [3] instances for TDTBF-based algorithms.

Now we consider instances from Group 2. In Table 4.7 we show the results regarding the average gaps for LP relaxations and the lower bounds after the cutting plane algorithm at the root node. Results for the B&C algorithm are presented in Table 4.8.

These instances are indeed harder to solve than the ones from Group 1. From the results in Table 4.8, we can see that TDTBF-B starts failing in solving instances with $n = 40$ and that for $n = 60, w = 100$ it is not capable of solving any of the instances. Furthermore, compared to instances in Group 1 the average gaps for similar combinations of n and w are greater. The main difference between these sets is that, in Group 1, w stands for the maximum width of a time window while in Group 2 it stands for the average. One consequence of this difference is that, for instances in Group 2, some vertices have more feasible time periods, making the instance harder to solve by the algorithms. This is supported by Table 4.7, where gaps of the LP relaxations and the root node for similar combinations of n, w are higher for this group.

Regarding the performance of the algorithms, we first notice that TDTBF-B has a poor performance, specially for instances with $w \geq 60$. Indeed, for $w = 100$ it is only able to solve 10 of the 25, where 5 of them are for $n = 20$ and 4 for $n = 40$. Considering the 140 instances in the whole group, TDTBF-B fails in solving 34.

TDTBF-S and IBR-k are comparable in terms of number of instances solved and all variants seem to produce reasonable results. IBR-10 cannot solve 8 out of the 140 in the group, IBR-20 9, IBR-5 10 and TDTBF-B 11. For the gaps reported in Table 4.7, the results are somehow aligned with the ones from the previous group. By performing more iterations of the bucket refinement heuristic the gaps of the LP relaxations tend to decrease, with more significant relative improvements for $k = 5, 10$. A similar observation holds for the gaps at the root node after the cutting phase. As a counterpart, LP relaxations become harder to solve and more time consuming.

From Table 4.8, the number of nodes explored tends to decrease for larger values of k. In some cases, we observed that IBR-20 needs to explore more nodes than IBR-10. Furthermore, since k represents the maximum number of iterations of the IBR, sometimes IBR-20 stops at the 10th iteration and therefore the result is exactly the same as for IBR-10. This is the case, for example, of instance $n60w100.003$, which neither TDTBF-B, TDTBF-S nor IBR-5 can solve it. In general, when more than 10 iterations of the refinement heuristics are applied, computational times of IBR-20 tend to increase compared to the ones for IBR-10. The improvements in %lpG and %rG are not very significant and the number of variables increases considerably. For instance, in Table 4.7 we can see that for $n = 100, w = 100$ the average absolute improvement is of 0.56% for %lpG and 0.08 % for %rG comparing IBR-10 with IBR-20. Analyzing the model in each case, the average number of variables for IBR-20 is 97096.8 while for IBR-10 is 74219.2, which represents an increment of approximately the 30 %. As a reference, the average number of variables of TDTBF-S is 40815.4 and the improvement achieved by IBR-20 in %rG is 1.65%. The reasons for this effect could depend on the effectiveness of preprocessing and the definition of precedence relations. The fact that we do not assume that the triangle inequality holds on the travel times has a direct impact on these two aspects of the algorithm, obtaining as a result dense formulations which are difficult to solve.

Inst.	n	w	TDTBF-B		TDTBF-S		IBR-5		IBR-10		IBR-20	
			%lpG	%rG	%lpG	%rG	%lpG	%rG	%lpG	%rG	%lpG	%rG
n20w20X		20	1.54	0.76	1.40	0.76	0.40	0.18	0.40	0.18	0.40	0.18
n20w40X		40	7.72	3.33	7.28	0.93	4.36	0.25	3.93	0.13	3.78	0.13
n20w60X	30	60	13.08	4.93	10.43	1.29	6.90	0.62	6.34	0.47	6.17	0.46
n20w80X		80	19.21	10.55	18.84	4.88	15.01	3.22	13.99	3.02	13.66	2.68
n20w100X		100	16.85	4.15	16.09	0.58	11.81	0.37	10.59	0.36	9.62	0.22
n40w20X		20	4.19	1.45	3.37	0.75	1.61	0.15	1.54	0.15	1.52	0.15
n40w40X		40	10.53	2.57	9.53	0.42	7.36	0.27	6.49	0.24	6.43	0.24
n40w60X	40	60	16.18	7.84	15.69	4.28	13.10	3.01	11.52	2.09	11.24	2.06
n40w80X		80	22.04	11.92	21.48	5.37	18.07	2.91	16.48	1.70	15.61	1.34
n40w100X		100	21.02	11.52	20.48	3.87	18.62	3.22	16.47	2.86	15.91	2.70
n60w20X		20	3.82	1.20	2.93	0.75	2.24	0.52	2.01	0.49	1.96	0.41
n60w40X		40	9.17	2.54	8.39	0.71	6.51	0.57	5.10	0.38	4.88	0.28
n60w60X	60	60	15.13	6.29	12.55	2.41	10.14	1.24	9.69	1.12	9.67	1.09
n60w80X		80	20.92	9.28	18.94	1.11	16.13	0.63	15.55	0.56	14.89	0.47
n60w100X		100	25.56	13.68	24.67	4.84	20.67	3.21	19.57	2.62	19.05	2.43
n80w20X		20	5.75	1.21	5.07	0.35	2.92	0.27	2.40	0.24	2.39	0.24
n80w40X		40	9.80	4.29	9.28	1.07	7.30	0.55	6.79	0.50	6.42	0.49
n80w60X	80	60	15.30	5.34	13.85	1.26	11.55	1.08	10.96	0.84	10.56	0.80
n80w80X		80	20.13	7.51	19.01	1.64	16.21	1.20	14.80	0.89	14.23	0.77
n80w100X		100	22.16	12.87	20.82	3.43	18.20	2.79	16.46	2.49	15.81	1.91
n100w20X		20	6.14	2.42	5.44	0.83	3.09	0.31	2.46	0.25	2.42	0.25
n100w40X		40	10.47	2.79	9.61	0.66	7.57	0.48	6.50	0.38	6.15	0.35
n100w60X	100	60	14.36	5.51	12.74	1.77	10.14	1.28	9.28	1.05	8.89	0.88
n100w80X		80	21.42	10.08	20.34	2.00	17.25	1.63	16.17	0.76	15.49	0.70
n100w100X		100	23.55	10.76	22.79	3.85	19.71	2.87	18.15	2.18	17.59	2.10
n150w20X		20	6.83	2.47	6.22	0.46	4.38	0.17	3.65	0.10	3.56	0.10
n150w40X	150	40	10.85	3.79	9.94	0.86	7.61	0.46	6.79	0.30	6.42	0.28
n150w60X		60	14.92	5.23	13.02	1.43	11.31	1.06	9.94	0.76	9.35	0.61

Table 4.7.: Average % GAPs on Group 2 instances for TDTBF-based algorithms.

Inst.	n	w	TDTBF-B		TDTBF-S		IBR-5		IBR-10		IBR-20	
			Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes
n20w20X		20	0.00	1.00	0.04	0.40	0.02	0.00	0.02	0.00	0.02	0.00
n20w40X		40	0.04	63.40	0.14	7.20	0.15	2.60	0.14	1.80	0.14	1.80
n20w60X	30	60	0.16	109.20	0.25	39.40	0.18	5.80	0.24	3.20	0.26	3.80
n20w80X		80	3.70	1929.40	0.82	124.40	1.58	109.60	3.41	78.40	3.57	73.80
n20w100X		100	0.27	97.00	0.29	3.40	0.50	2.40	0.72	2.80	1.27	2.40
n40w20X		20	0.03	8.80	0.36	5.20	0.36	3.20	0.36	3.40	0.36	3.40
n40w40X		40	0.34	167.00	0.52	9.80	0.73	5.00	1.09	4.00	1.09	4.00
n40w60X	40	60	3.42	1642.00	2.59	169.40	5.38	114.40	9.29	59.80	7.05	45.20
n40w80X		80	(3) 26.54	5808.00	35.52	1186.40	15.38	319.40	18.64	154.60	63.09	242.60
n40w100X		100	(4) 65.83	9097.00	149.66	2484.60	230.17	1486.20	(4) 32.45	97.50	(4) 31.72	38.50
n60w20X		20	1.00	230.80	1.28	74.60	1.23	24.40	1.36	18.00	1.27	9.80
n60w40X		40	1.60	433.20	2.10	37.60	2.47	6.60	4.09	11.00	4.41	4.80
n60w60X	60	60	51.52	9696.00	5.70	147.20	8.10	65.20	13.94	61.00	13.97	39.20
n60w80X		80	(4) 152.97	11158.00	14.89	180.60	25.26	71.40	40.35	33.80	41.31	13.60
n60w100X		100	* * *	* * *	(4) 175.91	1791.25	(4) 109.16	627.00	365.93	826.00	623.44	825.60
n80w20X		20	0.58	76.00	1.75	7.60	2.25	6.60	2.76	4.80	2.76	4.80
n80w40X		40	(4) 26.46	4774.75	29.14	1252.00	8.03	86.40	15.00	55.40	17.82	62.80
n80w60X	80	60	(4) 27.91	3542.75	(4) 8.69	47.25	(4) 14.16	44.75	(4) 23.98	23.00	(4) 40.96	22.75
n80w80X		80	(3) 169.78	19243.67	149.64	2056.00	93.48	465.60	189.69	162.40	338.28	134.20
n80w100X		100	* * *	* * *	(4) 263.86	1117.75	(4) 811.15	607.25	(4) 580.69	323.25	(4) 1150.85	136.00
n100w20X		20	21.54	3338.60	6.58	121.60	5.62	14.20	6.80	7.20	6.79	7.20
n100w40X		40	31.26	3755.60	9.92	192.00	12.54	40.20	17.35	23.40	21.99	13.00
n100w60X	100	60	(3) 311.55	13029.33	(4) 56.87	514.25	310.93	1629.00	(4) 87.33	36.00	390.85	190.80
n100w80X		80	(2) 121.17	6057.50	(4) 397.46	3009.25	(3) 96.06	58.33	441.00	587.80	(4) 438.89	72.00
n100w100X		100	(1) 81.08	3402.00	(2) 132.67	322.50	(2) 210.26	76.00	(2) 418.59	33.50	(1) 530.32	6.00
n150w20X		20	283.84	20257.80	18.11	51.00	19.64	21.80	25.17	8.40	25.76	9.80
n150w40X	150	40	(2) 236.35	11203.50	(4) 218.72	976.50	(4) 370.57	1215.75	353.91	402.00	464.56	384.00
n150w60X		60	(1) 195.14	5258.00	(3) 218.33	1061.00	(4) 391.39	717.75	(4) 557.85	234.00	(4) 816.60	259.50

Table 4.8.: Average computational times (in seconds) and number of tree nodes explored on Group 2 instances for TDTBF-based algorithms.

In order to determine the algorithm that produces the best results, we can consider different alternatives. The table below summarizes some general results regarding TDTBF-S and IBR-k, for $k = 5, 10, 20$. For each method, we can observe the number of unsolved instances, the average computational times (in seconds) over all instances that it solves as well as considering only the instances solved by all algorithms.

	TDTBF-S	IBR-5	IBR-10	IBR-20
Unsolved instances	11	10	8	9
Avg. times (in seconds)				
all instances solved	58.55	86.26	102.44	157.23
instances solved by all algorithms	41.67	52.27	56.49	112.58

Considering the number of instances solved, IBR-10 appears as the best approach. Indeed, IBR-5, IBR-10 and IBR-20 are able to solve more instances than TDTBF-S, showing that the refinement heuristic is quite useful and produces good results in this regard.

In terms of average computational times, in both cases the best results are obtained by TDTBF-S. The differences are quite significant when considering all the instances solved by each algorithm, but tend to be less important when restricting to instances solved by all of them. This is reasonable since IBR-k algorithms solve more instances that TDTBF-S.

With respect to IBR-k algorithms, we can appreciate that the average times for IBR-20 are considerable higher than for IBR-5 and IBR-10. The difference remains almost the same when restricting to instances solved by all algorithms, specially compared with IBR-10. This is because IBR-10 and IBR-20 solve almost the same instances, and the difference lies mainly on the computational effort required to solve the LP relaxations. Computational times for IBR-5 and IBR-10 are comparable, specially when restricting to instances solved by all algorithms.

Another factor we consider is the computational times required by the IBR. For IBR-5, the effort required is not significant. For IBR-10 and IBR-20 the IBR requires more time, specially as the number of vertices increases and the time windows become wider. Considering all instances in Group 2, IBR-10 spends 6.47 seconds on average refining buckets and the maximum time is 55.56 seconds. For IBR-20, these values increase to 53.76 and 502.06 seconds, respectively. In both cases, more than 90% of the time is used to solve the LP relaxations. These results provide another reason to consider a maximum number of iterations for the IBR. It is important to remark that, for IBR-10, incorporating these times to the ones reported for the B&C algorithm does not change the analysis regarding the comparison with the other methods.

Based on these parameters, we analyze in more detail the results obtained by TDTBF-S and IBR-10. Considering the average computational times over instances solved by both algorithms, TDTBF-S requires 53.65 seconds and IBR-10 70.59 seconds. For the remaining instances, 6 of them cannot be solved by any of them, 2 can be solved only by TDTBF-B and 5 only by IBR-10. Due to this mix of unsolved instances, we inspect them carefully. The table below shows specific information for these instances, in particular the time required for the algorithm and the gap at the root node (%rG-S for TDTBF-S and %rG-10 for IBR-10). With the only exception of $n150w40.001$, the computational times of the corresponding algorithm are the best considering also IBR-5 and IBR-20.

TDTBF-S				IBR-10			
Instance	Time	%rG-S	%rG-10	Instance	Time	%rG-S	%rG-10
n40w100.002	642.15	4.71	3.83	n60w100.003	1216.83	9.10	5.24
n80w100.001	97.36	0.55	0.52	n80w100.002	629.18	2.45	1.03
				n100w80.001	601.74	2.01	1.27
				n150w40.001	772.07	1.17	0.57
				n150w60.005	1337.92	2.84	1.25

Based on the computational times, instances solved by IBR-10 seem more difficult than the ones solved by TDTBF-S. On average, IBR-10 requires 911.55 seconds for these instances, while TDTBF-S 369.75 seconds. Furthermore, for $n80w100.001$ the gap at the root node in TDTBF-S is in fact really close to the optimum, the improvement obtained by IBR-10 is not significant and the increment in the number of variables affects negatively its behaviour. Indeed, in several instances we observed that if %rG for TDTBF-S is below 1%, then refining the buckets is not convenient, producing in general higher computational times. On the other hand, in 3 of the 5 instances solved by IBR-10 we can appreciate a considerable reduction in the gap at the root node (above 1%) compared to TDTBF-S.

In Dash et al. [19], the IBR does not have any limit on the number of iterations to perform. The stopping criterion is the no achievement of a significant improvement in the objective value of the LP relaxation. In our case and for these instances, the number of variables tends to grow quickly and in many cases without reducing the gap considerably. Unfortunately, from the detailed results we were not able to obtain a clear pattern on whether it is convenient or not to continue refining the buckets. In several cases, obtaining significant improvements by the IBR in both %lpG and %rG reduces the overall computational times, although this is not always the case.

This observation leads us to conjecture that the IBR could be improved by including in the stopping criterion information regarding the improvements obtained after the cutting phase, paying special attention to the computational effort required. Executing the complete cutting plane algorithm may require a considerable amount of time, specially when the number of buckets defined tends to increase. However, we observed that at the root node the most significant improvements in the lower bound take place in the first iterations of the cutting plane algorithm, with smaller gains in the remaining rounds. This behaviour could be exploited by adding a cut probing phase during the execution of the IBR heuristic and use the resulting value to decide whether it is convenient or not to continue refining buckets.

In any case, based on the computational experiments presented in this section we can appreciate that for the TDTSP-TW the IBR represents an important factor to evaluate. Taking all this into account, we observe that IBR-10 represents a good tradeoff among the level of refinement, the effort spent on cutting planes and the overall computational times of the algorithms.

4.6. Conclusions

In this chapter we studied the Time Dependent Travelling Salesman Problem with Time Windows (TDTSP-TW) for the case where the variations are produced in the travel times and costs.

First, we generalized the well known *infeasible paths* in order to consider the time dependency factor (TDTBF). We adapted the formulation from Dash et al. [19] regarding the Time Bucket Formulation (TBF) and the valid inequalities the authors propose to the particularities of the time dependent case. We propose some strengthened versions of the Time Dependent Tournament constraints and new families of valid inequalities for the TDTSP-TW that incorporates bucket information.

One of the characteristics of the TDTSP-TW is that, due to the nature of the problem, the triangle inequality does not hold on the travel times. We adapted to our case several preprocessing procedures from the TSPTW. Upon this, we developed a B&C algorithm that considers the valid inequalities we derive as well as the Bucket SOP proposed in [19]. Computational experiments were conducted over instances generated from benchmark ones from the TSPTW, obtaining very good computational results that show that the overall approach is effective. We experimented with different time discretization of the buckets and provided a deeper insight on the behaviour of this formulation, noticing that it depends mainly on the combination of the definition of buckets, the cutting planes algorithm and the preprocessing phase.

Future research may involve different stages. Firstly, a more sophisticated preprocessing is worth investigating since it has a great impact on the TDTSP-TW in general, although not assuming that the triangle inequality holds represents a very limiting factor. Secondly, the iterative bucket refinement heuristic proposed by [19] could be reformulated to consider the particular case of the TDTSP-TW. One possibility is to include a more complex criterion to determine whether an improvement is significant or not. For example, some heuristic approach that estimates the improvements considering also information regarding the cutting plane algorithm. For this purpose, more experiments should be performed to determine these parameters experimentally.

5. Conclusions and Future Research

5.1. General conclusions

In this work we approach two different versions of the *Time Dependent Travelling Salesman Problem* using integer linear programming formulations. The TDTSP has many interesting applications in real-world problems, which motivated us to study some of its variations.

In the first part of this research, we studied the *position dependent* version of the TDTSP. We considered two models proposed in the literature and proved that the corresponding polytopes are equivalent. In order to improve the lower bound obtained by solving the corresponding LP relaxation, we performed a polyhedral study of the convex hull of the feasible solutions of the models. A great part of our work was dedicated to this aspect. We derived several families of valid inequalities and proved that five of them are facet defining. We call these inequalities *Lifted Time Dependent Cycle Inequalities*, which follow a similar pattern as the *lifted cycle inequalities* for the ATSP. Indeed, a theoretical framework was proposed which may allow to derive more facets of the polyhedra by means of applying a sequential maximum lifting over a determined set of variables.

Aiming to provide an exact algorithm for the TDTSP, we develop a special purpose *Branch and Cut* algorithm that incorporates the valid inequalities and facets derived as cutting planes. For this purpose, we proposed separation algorithms for the Lifted TDCI. Furthermore, we also developed a primal heuristic composed by a construction phase and an improvement phase. The algorithm was tested over benchmark TSP and TDP instances from the related literature. The results obtained were compared with the ones obtained using a general purpose commercial software and also with a specific exact algorithm for the TDP.

From a practical standpoint, we can conclude that our B&C algorithm produces very good results. The families of valid inequalities derived showed to be very effective when used as cutting planes, producing good quality lower bounds with small *gaps* at the root node of the B&C tree and requiring a reasonable computational effort for the separation procedures. In addition, the primal heuristic also turn out to be very useful for the algorithm, finding near optimal solutions at the very beginning of B&C tree. Indeed, the combination of these two factors plays a crucial role in the success of the overall approach, reducing considerably the number of nodes explored. Compared to the two other methods considered, our algorithm produces better results in terms of computational times. The specific algorithm for the TDP provides tight LP relaxations but the computational effort is much more time consuming. As regards the general purpose algorithm, we are able to solve more instances at less computational effort. This is due to the inclusion of specific information of the problem in our algorithm.

From a more general perspective, the position-dependent version of the TDTSP shows to be a very challenging problem. Compared to the *Travelling Salesman Problem*, the results we obtained as well as the ones reported in the related literature give evidence that this version of the TDTSP is much harder to solve than the TSP. In fact, while the for the TSP instances with hundreds of vertices can be solved in a few seconds, for the TDTSP it requires a considerable amount of time to solve instances with tens of vertices. This difference is reasonable, since the TDTSP is a generalization of the TSP.

During the second stage of this research we focused on the version of the TDTSP with time windows and time dependent travel times and costs. We proposed two formulations for the problem which are generalizations of models that have been proven to be very effective for the TSPTW. The idea behind the formulations is to manage the time dependence similarly to the time windows, using in particular the concept of *infeasible paths*. For this purpose, we generalized and adapted to the TDTSP-TW several approaches and results for the TSPTW in order to provide an exact algorithm. Based on computational experiments, one of the models, the TDTBF, produces the best results compared to the other formulation proposed and two adapted models from the TDTSP related literature.

The next was meant to study in more detail the TDTBF. Since the infeasibility of a path depends not only on the sequence of vertices but also in the time periods in which arcs are travelled, we proposed a time dependent version of the Tournament Constraints. Furthermore, we derived strengthened versions of these inequalities, one of them including specific information provided by the TDTBF. We also considered a few families of valid inequalities proposed in Dash et al. [19] regarding generalizations of inequalities for the PCATSP. Finally, we provided some other families of valid inequalities that exploit the structure of the TDTBF.

From a computational aspect, we developed a B&C algorithm that incorporates the valid inequalities considered with their corresponding separation procedures. We also implemented a preprocessing phase, initial heuristics and a procedure described in Dash et al. [19] for the definition of buckets for the TDTBF. In fact, the strategy adopted in this regard may produce significant changes in the behaviour of the algorithm. For this reason, we considered different alternatives and evaluated them over instances generated from benchmark instances from the TSPTW literature.

In our perspective, the results obtained are indeed very good. The approaches based on the TDTBF produce in general tight lower bounds, specially after the cutting phase of the root node. Valid inequalities proved to be very helpful, in particular the Bucket SOP inequalities. However, for the TDTSP-TW we encountered some differences in the behaviour of the TDTBF-based approaches when comparing to the TSPTW. We observed that the procedure for the bucket definition requires the adaptation to the particular case of the TDTSP-TW. The approaches evaluated in the computational experiments allowed us to obtain a clearer insight of this behaviour.

In general, studying this version of the problem gave us very interesting results. As regards the models and compared to previous approaches to similar problems, we proposed a different way of incorporating the time dependency factor, generalizing many results from the TSPTW. From the practical side, we did not assume that the triangle inequality holds on the travel times, which had an important effect on the preprocessing phase. This is also an important difference

regarding approaches to the TSPTW, where in general this assumption is made. Overall, we were able to solve instances with up to 150 vertices, fairly wide time windows and several time periods in a reasonable time.

5.2. Future research and open problems

Our work on the Time Dependent Travelling Salesman Problem leaves open several lines for future research.

Regarding the position dependent version of the TDTSP, the separation procedure of the Lifted TDCI could be improved. In addition, more families of facets and valid inequalities could be derived in order to include them in the B&C algorithm. In the computational experiments we observed that for instances with a large number of vertices the LP relaxations tend to become harder to solve due to the increase in the number of variables and constraints in the model. Tackling this aspect could lead to improvements in the overall computational times and therefore to a faster algorithm.

For the version with time windows and varying travel times, the TDTSP-TW, the preprocessing could be investigated in more detail. The impact of having an effective preprocessing may be quite significant. Since the triangle inequality does not hold, alternative methods could be considered to determine precedence relations that cannot be directly inferred using the limits of the time window. This might have an impact not only on the preprocessing phase, but also on the definition of stronger precedence relations. It would also be interesting to derive further families of valid inequalities that exploit this information.

As regards the TDTBF, we consider that more experiments and further developments on the iterative bucket refinement heuristic could be held. An alternative may be to include a probing phase to evaluate not only the objective value of the LP relaxation but also the impact of the cutting plane algorithm at the root node. Of course, computational times for this procedure must be taken into account. Considering the results obtained by the approaches based on the TDTBF, it would be interesting to try to adapt these ideas to other versions of the TDTSP, as could be the case without time windows and the one with time varying travel speeds.

Finally, developements of heuristic and meta-heuristic algorithms for the TDTSP would also be of practical interest. Some of the ideas and models presented in this thesis may be considered for mathematical programming-based heuristic approaches to solve different versions of the TDTSP, aiming to obtain good quality solutions by means of efficient and fast algorithms.

A. Glossary of acronyms

ATSP	Asymmetric Travelling Salesman Problem
B&B	Branch and Bound
B&C	Branch and Cut
B&P	Branch and Price
BC	Branch and Cut algorithm for Picard and Queyranne formulation
BC-R	Branch and Cut algorithm for the TDP from Méndez-Díaz et al.
GATSP	Graphical Asymmetric Travelling Salesman Problem
IBR	Iterative Bucket Refinement
IBR-k	Branch and Cut algorithm for the TDTBF, with buckets defined by the IBR heuristic performing a maximum of k iterations
ILP	Integer Linear Programming
LP	Linear Programming
LOP	Linear Ordering Problem
MD	Malandraki and Daskin Formulation
MILP	Mixed Integer Linear Programming
PCTSP	Precedence Constrained Travelling Salesman Problem
PQ	Picard and Queyranne Formulation
QAP	Quadratic Assignment Problem
SCM	Stecco, Cordeau and Moretti Formulation
SEC	Subtour Elimination Constraints
SOP	Sequential Ordering Polytope
STDSP	Sequence and Time Dependent Scheduling Problem
TBF	Time Bucket Formulations
TDCI	Time Dependent Cycle Inequalities
Lifted TDCI	Lifted Time Dependent Cycle Inequalities
TDIP	Time Dependent Infeasible Path
TDIPEC	Time Dependent Infeasible Path Elimination Constraints
TDIPF	Time Dependent Infeasible Path Formulation
TDP	Travelling Deliveryman Problem
TDTBF	Time Dependent Time Bucket Formulation
TDTBF-B	Branch and Cut algorithm for the TDTBF with a basic definition of buckets
TDTBF-S	Branch and Cut algorithm for the TDTBF with the simplest definition of buckets
TDTOUR	Time Dependent Tournament Constraint
TDTSP	Time Dependent Travelling Salesman Problem
TDTSP-TW	Time Dependent Travelling Salesman Problem with Time Windows
TDVRP	Time Dependent Vehicle Routing Problem
TIF	Time Index Formulation
TOUR	Tournament Constraint
TSP	Travelling Salesman Problem
TSPTW	Travelling Salesman Problem with Time Windows
VRP	Vehicle Routing Problem
VW	Vander Wiel and Sahinidis Formulation

B. Glossary of notation

$\mathbb{R}$	the set of real numbers
$\mathbb{Z}$	the set of integer numbers
$\mathbb{N}$	the set of natural numbers
$D = (V, A)$	complete directed graph
c_{ijk}	travel time when arc (i, j) is in position k of the tour
y_{ijk}	variable indicating if arc (i, j) is in position k of the tour
x_{ik}	variable indicating if vertex i is in position k of the tour
P_{PQ}	polytope defined by PQ formulation
P_{PW}	polytope defined by VW formulation
P^n_{TD}	overloaded notation to refer to P_{PQ} or P_{VW}
C, k	cycle travelled in consecutive positions starting in k
$P^n_{TD}(C, k)$	projected polytope induced by C, k
$\dim(P)$	dimension of polytope P
M	number of different time periods defined for arc (i, j)
T^m_{ij}	upper limit of time period m for arc (i, j)
c^m_{ij}	travel time for arc (i, j) starting at i in time period m
θ^m_{ij}	travel cost for arc (i, j) starting at i in time period m
$W_i = [r_i, d_i]$	time window of vertex i
p_i	processing time of vertex i
t_{v_i}	earliest departure time from vertex i given $[P, T]$
s_{v_i}	earliest arrival time at vertex i given $[P, T]$
S	subset of vertices
$\bar{S}$	complement of S
$[P, T]$	time dependent simple path
$[r_b, d_b]$	limits of bucket b
B_i	collection of buckets of vertex i
$\mathcal{B}$	collection of all buckets defined
$I_k(i, b)$	set of feasible departure buckets in B_k that arrive at i in bucket b
$N_i(k, b)$	arrival bucket in B_i when travelling arc (k, i) starting in bucket $b \in B_k$
$S_{ij}(m)$	set of buckets in B_i defining time period m for arc (i, j)
$\beta(i)$	departure bucket from i in a canonical solution
$\gamma(i)$	arrival bucket at i in a canonical solution
x^m_{ij}	variable indicating if travelling arc (i, j) starting at i in time period m
y^b_{ij}	variable indicating if travelling arc (i, j) starting at i in bucket b
y^b_i	variable indicating if departing from vertex i in bucket b
z^b_i	variable indicating if arriving to vertex i in bucket b

P_1^{TD}	polytope defined by TDIPF formulation
P_2^{TD}	polytope defined by TDTBF formulation
z_{v_i}	critical arrival time at v_i given $[P, T]$
y_{v_i}	critical departure time from v_i given $[P, T]$
$\prec$	precedence relation between vertices (overloaded also for buckets)
$\delta(S)$	set of arcs (i, j) with $i \in S$ and $j \in \bar{S}$
$\pi(S)$	set of vertices such that $i \prec j$, for some $j \in S$
$\sigma(S)$	set of vertices such that $i \prec j$, for some $i \in S$
$\pi(B)$	extension of precedence set to buckets
$\sigma(B)$	extension of precedence set to buckets

Bibliography

[1] H. Abeledo, R. Fukasawa, A. Pessoa, and E. Uchoa. The time dependent traveling salesman problem: Polyhedra and branch-cut-and-price algorithm. In P. Festa, editor, *Experimental Algorithms*, volume 6049 of *Lecture Notes in Computer Science*, pages 202–213. Springer Berlin / Heidelberg, 2010.

[2] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Oper. Res. Lett.*, 33(1):42–54, 2005.

[3] J. Albiach, J. M. Sanchis, and D. Soler. An asymmetric TSP with time windows and with time-dependent travel times and costs: An exact solution through a graph transformation. *European Journal of Operational Research*, 189(3):789–802, 2008.

[4] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, Princeton, NJ, USA, 2007.

[5] N. Ascheuer, M. Fischetti, and M. Grötschel. A Polyhedral Study of the Asymmetric Traveling Salesman Problem with Time Windows. *Networks*, 36(2):69–79, 2000.

[6] N. Ascheuer, M. Fischetti, and M. Grötschel. Solving the Asymmetric Travelling Salesman Problem with time windows by branch-and-cut. *Mathematical Programming, Series A*, 90(3):475–506, May 2001.

[7] E. Balas and M. Fischetti. Lifted cycle inequalities for the asymmetric traveling salesman problem. *Math. Oper. Res.*, 24(2):273–292, 1999.

[8] E. Balas, M. Fischetti, and W. R. Pulleyblank. The precedence-constrained asymmetric traveling salesman polytope. *Math. Program.*, 68:241–265, March 1995.

[9] E. Balas and M. Oosten. On the dimension of projected polyhedra. *Discrete Appl. Math.*, 87(1-3):1–9, 1998.

[10] J. E. Beasley. *Lagrangian relaxation*, pages 243–303. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[11] J. Bentner, G. Bauer, G. M. Obermair, I. Morgenstern, and J. Schneider. Optimization of the time-dependent traveling salesman problem with monte carlo methods. *Phys. Rev. E*, 64:036701, Aug 2001.

[12] L. Bianco, A. Mingozzi, and S. Ricardelli. The traveling salesman problem with cummulative costs. *Networks*, 23(2):81–91, 1993.

[13] L. Bigras, M. Gamache, and G. Savard. The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. *Discrete Optimization*, 5(4):685–699, 2008.

[14] CONCORDE. http://www.tsp.gatech.edu/concorde.html.

[15] J.-F. Cordeau, G. Ghiani, and E. Guerreiro. Properties and branch-and-cut algorithm for the time-dependent traveling salesman problem. *Working paper*, 2011.

[16] CPLEX. http://www.ilog.com/products/cplex.

[17] H. Crowder, E. L. Johnson, and M. W. Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31(5):803–834, 1983.

[18] G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson. Solution of a large-scale traveling-salesman problem. *Operations Research*, 3:393–410, 1954.

[19] S. Dash, O. Günlük, A. Lodi, and A. Tramontani. A time bucket formulation for the TSP with time windows. *Forthcoming in INFORMS Journal of Computing*, 2010.

[20] Y. Dumas, J. Desrosiers, E. Gelinas, and M. M. Solomon. An Optimal Algorithm for the Traveling Salesman Problem with Time Windows. *Operations Research*, 43(2):367–371, 1995.

[21] M. Fischetti. Facets of the asymmetric travelling salesman polytope. *Mathematics of Operations Research*, 16(1):42–56, 1991.

[22] M. Fischetti, G. Laporte, and S. Martello. The delivery man problem and cumulative matroids. *Oper. Res.*, 41(6):1055–1064, 1993.

[23] M. Fischetti and P. Toth. A polyhedral approach to the asymmetric traveling salesman problem. *Management Science*, 43(11):pp. 1520–1536, 1997.

[24] K. Fox. *Production scheduling on parallel lines with dependencies*. PhD thesis, John Hopkins University, 1973.

[25] K. R. Fox, B. Gavish, and S. C. Graves. An n-constraint formulation of the (time-dependent) traveling salesman problem. *Oper. Res.*, 28(4):1018–1021, 1980.

[26] M. Gendreau, A. Hertz, G. Laporte, and M. Stan. A generalized insertion heuristic for the traveling salesman problem with time windows. *Oper. Res.*, 46:330–335, March 1998.

[27] P. Gilmore and R. Gomory. A linear programming approach to the cutting stock problem. *Operations Research*, 9:849–859, 1961.

[28] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Society*, 64:275–278, 1958.

[29] L. Gouveia and S. Voss. A classification of formulations for the (time-dependent) traveling salesman problem. *European Journal of Operational Research*, 83(1):69–82, May 1995.

[30] M. Grötschel, M. Jünger, and G. Reinelt. A Cutting Plane Algorithm for the Linear Ordering Problem. *Operations Research*, 32(6):1195–1220, 1984.

[31] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *IIASA Collab. Proceedings Ser. CP-81-S1*, pages 511–546, 1981.

[32] M. Grötschel and M. W. Padberg. On the symmetric travelling salesman problem I: inequalities. *Mathematical Programming*, 16:265–280, 1979.

[33] M. Grötschel and M. W. Padberg. On the symmetric travelling salesman problem II: lifting theorems and facets. *Mathematical Programming*, 16:281–302, 1979.

[34] G. Gutin and A. Punnen. *The Traveling Salesman Problem and Its Variations*. Combinatorial Optimization. Kluwer Academic Publishers, 2002.

[35] A. Hill and W. Benton. Modelling intra-city time-dependent travel speeds for vehicle scheduling problems. *The Journal of the Operational Research Society*, 43(4):343–351, 1992.

[36] S. Ichoua, M. Gendreau, and J.-Y. Potvin. Vehicle dispatching with time-dependent travel times. *European Journal of Operational Research*, 144(2):379–396, 2003.

[37] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, December 1984.

[38] F. Li, B. Golden, and E. Wasil. Solving the time dependent traveling salesman problem. In B. Golden, S. Raghavan, E. Wasil, R. Sharda, and S. Voss, editors, *The Next Wave in Computing, Optimization, and Decision Technologies*, volume 29 of *Operations Research/Computer Science Interfaces Series*, pages 163–182. Springer US, 2005.

[39] S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269, 1965.

[40] A. Lucena. Time-dependent traveling salesman problem—the deliveryman case. *Networks*, 20(6):753–763, 1990.

[41] V. Mak and A. T. Ernst. New cutting-planes for the time- and/or precedence-constrained ATSP and directed VRP. *Mathematical Methods of Operations Research*, 66(1):69–98, 2007.

[42] C. Malandraki and M. S. Daskin. Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms. *Transportation Science*, 26(3):185–200, 1992.

[43] C. Malandraki and R. B. Dial. A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem. *European Journal of Operational Research*, 90(1):45 – 55, 1996.

[44] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, New York, 1990.

[45] I. Méndez-Díaz, P. Zabala, and A. Lucena. A new formulation for the traveling deliveryman problem. *Discrete Appl. Math.*, 156(17):3223–3237, 2008.

[46] M. C. Müller. Das dynamische travelling salesman problem. Master's thesis, Universität Kaiserslautern, Fachbereich Mathematik, August 1996.

[47] G. L. Nemhauser and L. A. Wolsey. *Integer and combinatorial optimization*. Wiley interscience series in discrete mathematics and optimization. Wiley, 1988.

[48] J. W. Ohlmann and B. W. Thomas. A compressed-annealing heuristic for the traveling salesman problem with time windows. *INFORMS J. on Computing*, 19:80–90, January 2007.

[49] M. Padberg and G. Rinaldi. An efficient algorithm for the minimum capacity cut problem. *Math. Program.*, 47:19–36, 1990.

[50] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Rev.*, 33:60–100, February 1991.

[51] M. W. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5:199–215, 1973.

[52] C. A. Persiani. *Algorithms for UAS insertion in civil air space*. PhD thesis, Faculty of Engineering, University of Bologna, 2011.

[53] J. Picard and M. Queyranne. The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations Res.*, 26(1):86–110, 1978.

[54] P. A. Rubin and G. L. Ragatz. Scheduling in a sequence dependent setup environment with genetic search. *Computers & OR*, 22(1):85–99, 1995.

[55] M. W. P. Savelsbergh. A Branch-and-Price Algorithm for the Generalized Assignment Problem. *Operations Research*, 45:831–841, 1997.

[56] D. Soler, J. Albiach, and E. Martínez. A way to optimally solve a time-dependent vehicle routing problem with time windows. *Oper. Res. Lett.*, 37(1):37–42, 2009.

[57] G. Stecco, J.-F. Cordeau, and E. Moretti. A branch-and-cut algorithm for a production scheduling problem with sequence-dependent and time-dependent setup times. *Comput. Oper. Res.*, 35(8):2635–2655, 2008.

[58] P. Toth and D. Vigo, editors. *The vehicle routing problem*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[59] C. A. van Eijl. A polyhedral approach to the delivery man problem. Memorandum COSOR 95-19, Eindhoven University of Technology, 1995.

[60] R. J. Vander Wiel and N. V. Sahinidis. Heuristic bounds and test problem generation for the time-dependent traveling salesman problem. *Transportation Sci.*, 29(2):167–183, 1995.

[61] R. J. Vander Wiel and N. V. Sahinidis. An exact solution approach for the time-dependent traveling-salesman problem. *Naval Res. Logist.*, 43(6):797–820, 1996.

[62] X. Wang and A. C. Regan. On the convergence of a new time window discretization method for the traveling salesman problem with time window constraints. *Computers & Industrial Engineering*, 56(1):161 – 164, 2009.

[63] I. R. Webb. Depth-first solutions for the deliveryman problem on tree-like networks: An evaluation using a permutation model. *Transportation Science*, 30(2):134–147, 1996.

[64] B. Y. Wu, Z.-N. Huang, and F.-J. Zhan. Exact algorithms for the minimum latency problem. *Inf. Process. Lett.*, 92:303–309, December 2004.