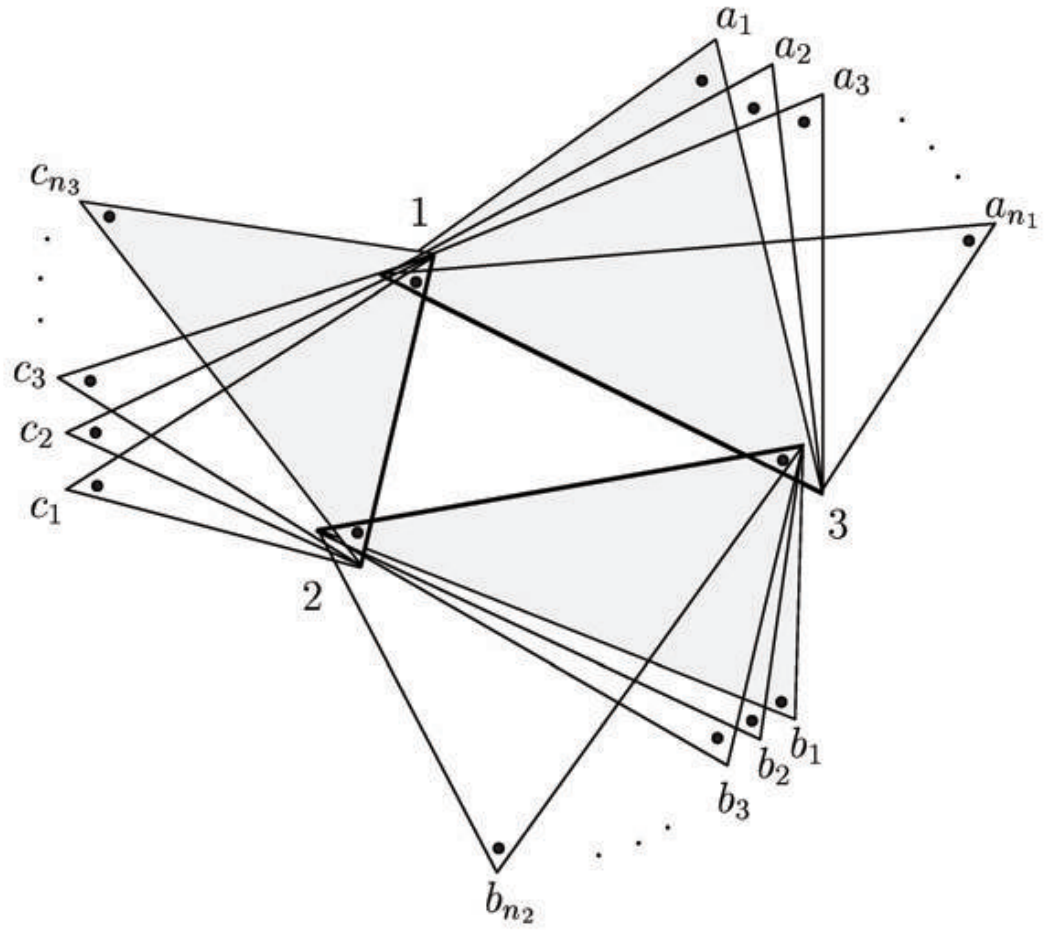
Luis M. Torres

Online Vehicle Routing

Set Partitioning Problems



Cuvillier Verlag Göttingen

Online Vehicle Routing

Set Partitioning Problems

vorgelegt von
Dipl.–Math. Luis M. Torres
Berlin

Der Fakultät II – Mathematik- und Naturwissenschaften
der Technischen Universität Berlin
zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften
Dr. rer. nat.

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Volker Mehrmann
1. Berichter:	Prof. Dr. Martin Grötschel
2. Berichter:	Dr. Sven O. Krumke

Tag der wissenschaftlichen Aussprache: 15. Dezember 2003

Berlin 2003

D 83

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.
1. Aufl. - Göttingen : Cuvillier, 2004
Zugl.: (TU) Berlin, Univ., Diss., 2003
ISBN 3-86537-127-2

Gedruckt mit Unterstützung des Deutschen Akademischen Austauschdienstes

Nonnenstieg 8, 37075 Göttingen
Telefon: 0551-54724-0
Telefax: 0551-54724-21
www.cuvillier.de

1. Auflage, 2004
Gedruckt auf säurefreiem Papier

ISBN 3-86537-127-2

ZUSAMMENFASSUNG

Diese Arbeit befaßt sich mit zwei verschiedenen Aspekten der kombinatorischen Optimierung: Online-Einsatzplanung von Fahrzeugen unter Echtzeit-Bedingungen und Mengenpartitionierungsprobleme. In der Online-Fahrzeugeinsatzplanung geht es darum, Routen zu berechnen, mit denen eine Flotte von Fahrzeugen eine gegebene Menge von Aufträgen optimal bedient. Das Wort "Online" weist auf die Tatsache hin, dass unmittelbare Entscheidungen auf Basis unvollständiger Information getroffen werden müssen. Darüber hinaus fordern die Echtzeit-Bedingungen garantierte Antworten innerhalb kurzer Zeit. Das Mengenpartitionierungsproblem, andererseits, ist ein grundlegendes Problem in der kombinatorischen Optimierung: Gegeben ein Untermengensystem $\mathcal{F} \subseteq 2^X$ einer Grundmenge X und eine Kostenfunktion $c : \mathcal{F} \to \mathbb{R}_+$, finde ein Subsystem $\mathcal{F}' \subseteq \mathcal{F}$ minimaler Kostensumme, dessen Mengen eine Partition von X bilden.

Die Probleme der Online-Fahrzeugeinsatzplanung, die wir hier betrachten, stammen aus einem Kooperationsprojektes des *Konrad-Zuse Zentrums für Informationstechnik* in Berlin mit dem *Allgemeinen Deutschen Automobil-Club* (ADAC), in dem ein Online-Dispatching-Verfahren für die Disposition der Pannenhilfefahrzeugen vom ADAC entwickelt wird. In dieser Arbeit wird die Aufstellung eines solchen Einsatzplans als Online-Optimierungsproblem modelliert und ein Lösungsalgorithmus dafür entworfen. Dieser verwendet einen Spaltengenerierungsansatz, um einen kostengüstigen Dispositionsplan zu berechnen, der alle zu einem bestimmten Zeitpunkt bekannten Aufträge mit den vorhandenen Hilfsfahrzeugen bedient. Mit denselben Methoden werden auch untere Schranken für die Kosten eines optimalen Plans ermittelt. Wir berichten über praktische Erfahrung mit dem Verfahren, das sich seit Herbst 2002 im Pilotbetrieb beim ADAC befindet. Auf der anderen Seite untersuchen wir das Online-Problem aus theoretischer Sicht. Für eine vereinfachte Problemstellung stellen wir einen kompetitiven Algorithmus vor und beweisen untere Schranken.

Ein weiterer theoretischer Beitrag dieser Arbeit ist die Untersuchung von Mengenpartitionierungsproblemen einer besonderen Form. Sie werden von unserem Algorithmus für die Aufstellung eines Fahrzeugeinsatzplans als Unterprobleme gelöst. Ihre Besonderheit besteht darin, dass die Kardinalität jeder Menge im gegebenen Mengensystem $\mathcal{F}$ von einer kleinen Konstante k begrenzt wird. Wir widmen uns diesem Problem aus der Perspektive der Polyedertheorie. Für das verwandte Mengenpackungsproblem erforschen wir, wie sich die Kardinalitätsbegrenzung der Untermengen auf die Struktur der Facetten vom Packungspolytop auswirkt.

Schlüsselbegriffe: Fahrzeugeinsatzplanung, Spaltengenerierung, Online-Optimierung, polyedrische Kombinatorik, Mengenpartitionierung

ABSTRACT

This thesis addresses two different subjects in combinatorial optimization: online vehicle dispatching in real-time, and set partitioning problems. Online vehicle dispatching is the task of computing an optimal schedule to serve a given set of requests with a fleet of vehicles. The word "online" means that immediate decisions have to be taken on the basis of incomplete information. Moreover, real-time conditions specify that an answer must be provided within a short time-frame. The set partitioning problem, on the other hand, is a fundamental problem in combinatorial optimization: Given a family $\mathcal{F} \subseteq 2^X$ of subsets from a ground set X, and a cost function $c : \mathcal{F} \to \mathbb{R}_+$, find a subsystem $\mathcal{F}' \subseteq \mathcal{F}$ whose sets form a partition of X and have the minimum sum of costs.

The instances of the online vehicle dispatching problem we discuss here appeared in the framework of a cooperation project between the *Konrad-Zuse Zentrum für Informationstechnik* in Berlin and the German Automobile Association (*Allgemeiner Deutscher Automobil-Club*, ADAC). The objective of this project has been to design an automated online dispatching system for operating the fleet of service vehicles at ADAC. In this thesis, we formulate the planning process as an online optimization problem and describe a solution algorithm for it. Given the set of waiting requests and available units at some stage in the planning, our algorithm uses a column generation approach to find a low cost feasible dispatch that covers all requests with the units. In the same way, lower bounds on the cost of an optimum dispatch are computed. An implementation of this algorithm has been in pilot operation at ADAC since fall 2002; we report here on the practical experiences made during this period. Besides, we also consider the online problem from a theoretical point of view. For a simplified version of it, a competitive online algorithm is presented, and lower bounds are proved.

Another theoretical issue we consider in this thesis regards set partitioning problems of a special type. They appear as subproblems in our solution strategy for the dispatching problem at ADAC, and are characterized by a particular feature: the cardinality of every set in $\mathcal{F}$ is bounded by a small positive constant k. We look at them from the perspective of polyhedral combinatorics. For the related set packing problem, we investigate the effects of restricting the subset cardinality on the facetial structure of the packing polytope.

Keywords: vehicle dispatching, vehicle routing, column generation, online optimization, polyhedral combinatorics, set partitioning

ACKNOWLEDGMENTS

I am greatly indebted to many people who supported me in several ways during these (almost) four years. First of all, I would like to thank my advisors Prof. Dr. Martin Grötschel and Dr. Sven O. Krumke for their direction, and for giving me the opportunity to work with freedom on the research subjects I like. Working with them at the *Konrad-Zuse Zentrum (ZIB)* in Berlin has definitely been both a fruitful and a nice experience. Sven and Jörg Rambau introduced the ADAC project to me, which was the starting point for this thesis.

My work was financially supported by a scholarship from the *German Academic Exchange Service* (DAAD). I thank all the people at DAAD and at the *Akademisches Auslandsamt* from the TU Berlin. They have been of great help for the various organizational issues related to my stay in Germany. In this regard, thanks also go to Bettina Kasse and Sylwia Markwardt at the secretariat in ZIB.

I want to thank Diana Poensgen and Tjark Vredeveld for the interest they showed on the ADAC-Problem. The discussions we had together formed the basis for Chapter 5. With Tjark we also had the opportunity to explore several other directions of research – in Berlin, Rome and Copenhagen. Diana read some chapters of this thesis and provided me with helpful corrections.

I am very grateful to Annegret Wagler for her reading and commentaries on the part concerning set partitioning, and to Ralf Borndörfer for his patience every time I jumped into his room with a new question. It was the reading of his thesis what initially woke my interest for set partitioning problems.

Many of the results presented here are joint work with the other members of the "ADAC project" team: Sven Krumke, Jörg Rambau, Benjamin Hiller, Inna Dischke and Stephan Westphal. Benjamin efficiently provided me with simulation data and results whenever I needed them, which was quite often during these last weeks.

Finally, thanks to all my friends in the Optimization Department at the Konrad-Zuse Zentrum. It has been a pleasure to be with you these years. Thanks in particular to my roommate Tobias Achterberg. And very, very special thanks to Marc Pfetsch, for introducing the *Kaffeepause* at ZIB.

Contents

Chapter 1

Introducing the ADAC-Problem

The German Automobile Association *ADAC* (*Allgemeiner Deutscher Automobil-Club*[1]) – the second largest such organization in the world, surpassed only by the American Automobile Association – maintains a heterogeneous fleet of over 1,600 service vehicles in order to help people whose cars break down on the road. Due to their color and the fact that they often bring needed help, people call these service vehicles affectively "yellow angels". In the sequel, we simply refer to them as *units*, for short. Figure 1.1 shows a picture of them. Every unit is equipped with more than 300 tools and a GPS system which allows to precisely locate its position at any time. Incoming help calls from anywhere in Germany are transferred to one of five help centers, where human dispatchers process them. Their task is to assign to each customer a unit capable of handling his *request*, and to predict the unit's estimated time of arrival at the customer's location. In addition to the ADAC fleet, about 5,000 units operated by service *contractors* can be employed to cover requests that otherwise could not be served in time.

Figure 1.1: The *yellow angels* fleet from ADAC. Every service unit is equipped with more that 300 tools which together weigh about 280 kg. The cumulative distance traveled by all units in one year exceeds 56 million kilometers. (Taken from ADAC's press office).

[1] Am Westpark 8, 81373 München, Germany, `http://www.adac.de`

Due to increasing operational costs and request volume, this dispatching system has come under stress. As part of a cooperation project between ADAC, the *Konrad-Zuse Zentrum für Informationstechnik Berlin*[2] (ZIB) and the software development company *Intergraph Public Safety*[3] (IPS), we have been working on the design of an automatic online dispatching system to relieve the human operators and to reduce costs while ensuring a specified level of service quality (measured for instance in the average wait times for the customers). Additionally, we seek to exploit the new optimization potential that arises when schedules for large areas are computed globally. (To keep the problem instances at sizes that are manageable for human dispatchers, the approach that has been taken before is to divide the geographic region covered by a help center into many small sectors).

A Working Day at ADAC

The dispatching problem at ADAC – which we simply call the *ADAC-Problem* from now on – is an *online optimization problem*: we are looking for a reasonable strategy to control a system which is dynamically changing. To explain this in more detail, let us follow a dispatcher at one of the ADAC's help centers through his typical working day. In Figure 1.2 we can see what his workplace looks like. (Actually, several dispatchers will take turns at one of these places in the course of a day).

Figure 1.2: An ADAC dispatcher at work. The computer system registers via GPS the exact position of the service units at any time. Information about emerging help requests is received by phone from the call centers. Orders are transmitted to the units over radio. (Taken from ADAC's press office).

We start early in the morning, say at 2 a.m. For simplicity, we assume no help requests have been reported yet and only a few units are currently active. Some thirty minutes later, the first call of the day is received. Somewhere in Berlin an unlucky driver, who wants to return home after a long party-night, cannot get his auto started. On the phone, the dispatcher gets information about the request: its

[2] Takustr. 7, 14195 Berlin, Germany, `http://www.zib.de`

[3] Huntsville, AL 35894 USA, `http://www.intergraph.com`

geographical position, the kind of failure, etc. Then he checks the available units that are technically equipped for attending this request. Depending on the type of the failure, the service might require application of specialized tools. (In the worst-case, it might be necessary to haul the damaged car to a garage). Fortunately for our driver, this is not the case, since every unit is capable of providing start help "on-site". The dispatcher assigns the request to a unit, estimates the arrival time at the customer's position, and tells him when he can expect the "yellow angel" to be there.

After that, nothing happens for the next half an hour. Then the computer system informs the dispatcher a new unit has logged in and is ready for taking orders. Every unit is equipped with a GPS, which allows the computer to automatically track its precise position at any time. Now the dispatcher is aware that he can include this unit in future plannings. A couple of minutes later, the unit assigned to the first request calls and reports it has successfully finished its assigned task. As the day approaches, more and more units log in, and at the same time the units who have worked in the night-shift log off and return to their depots.

It is about 7 a.m. People are on their way to work, the streets get crowded, the traffic increases – and with it the number of help requests. Every time, the cycle is the same: the call is taken, information about the request is collected, a new service dispatch is computed to assign the request to an adequate unit, the arrival time of the unit at the customer's site is predicted, and the customer is informed. What varies is the size (and hence the complexity) of the successive assignment problems that the dispatcher has to solve.

Between 9 a.m. and 4 p.m., the system is working at its maximum load. On some days, there may be situations where as many as 300 requests are waiting for service, and about 80 units are active; with new requests arising at a rate of about 4 per minute. Figure 1.3 (elaborated from recorded real-world data) shows how the picture typically looks like in practice. The small crosses indicate the positions of ADAC's units while the small circles are used to illustrate the positions of waiting requests. The figure contains 40 units and 195 requests, and yet it only shows a partial view of the whole situation: at that time, there were in total 364 waiting requests and 82 units distributed over an area covering approximately one fifth of Germany. Under this conditions, the task of computing a service dispatch gets very complicated. And yet it has to be done quickly – in less than 5 seconds, according to ADAC's specifications.

Two additional aspects increase the palette of options a dispatcher has to take into account. At first, there is the possibility to hire service contractors, which can take those requests that cannot be handled in time by the ADAC's own units. Secondly, it might be convenient to reassign requests: Suppose, for instance, that a unit u has been assigned a request r_1. While u is on the way to getting there, a second request r_2 arises very close to the unit's current location. Now the dispatcher has to consider the possibility of changing orders and telling u to serve r_2 first. This is called a *preemption*, and we shall say more about it in Chapter 7.

Every time a dispatch is computed, a *cost function* has to be minimized, which

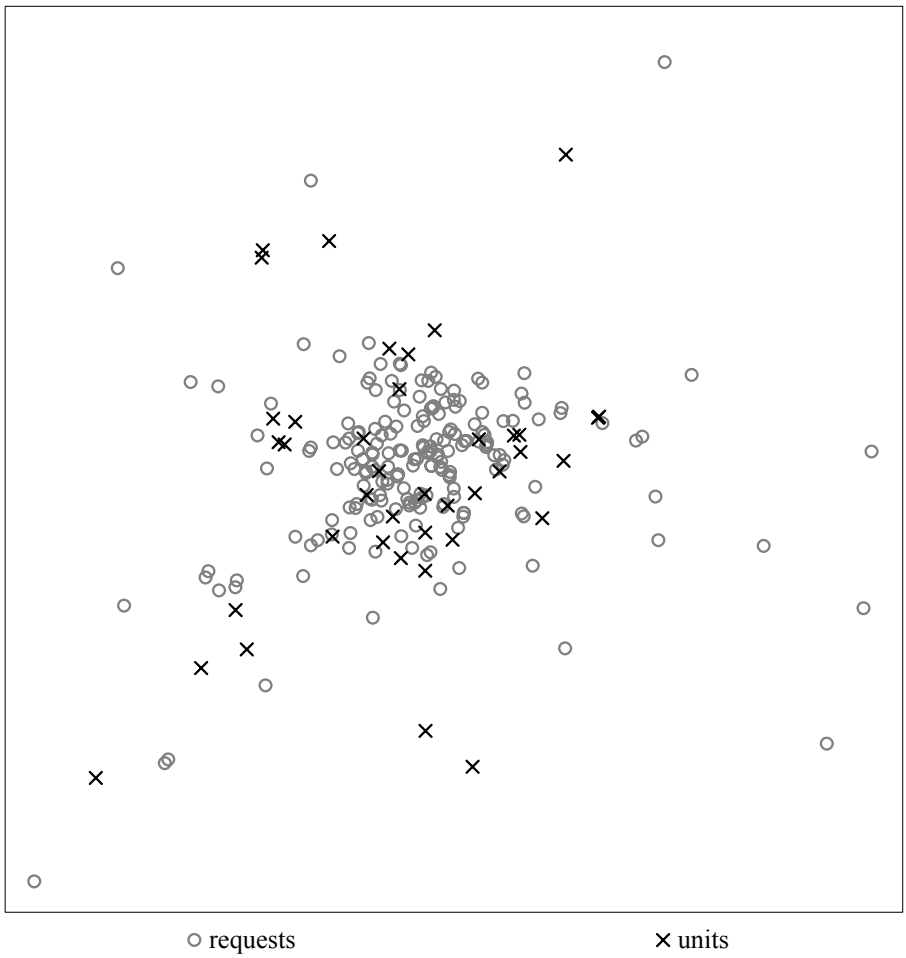


Figure 1.3: Partial view of a real-world snapshot in the dispatching process at ADAC. Crosses indicate the geographical positions of units, small circles the positions of help requests. The complete snapshot consisted of 364 requests and 82 units.

involves several components and is used as a measure of both the "real" operational costs of the system and some "fictive" costs that account for the quality of the service. (Again, a precise definition will be provided in Chapter 7, after introducing more background information on the problem).

After 7 p.m., the number of requests will eventually start to decrease. At some time, units that have been active during the day will finish their shifts and log off, and the units for the night-shift will log on again. Our working day at the ADAC's dispatching center ends here.

A first immediate observation when we look at the ADAC-Problem is that we

are dealing with a *discrete online problem*. To steer the system, we do not need to take actions continuously over time, but only whenever its state changes; and this can happen only as a consequence of one of the following *events*:

- a new request emerges,
- a unit finishes service of a request,
- a unit logs in,
- a unit finishes its shift and logs off.

Hence, we may model the dynamic planning process as a collection of *snapshots* in time. Every snapshot specifies a set of units, a set of requests, and a set of contractors; together with additional information concerning for instance their positions, their technical compatibilities (which requests may be assigned to which units/contractors), the time every request has been waiting in the system, etc. Our task is to compute on the basis of this information a minimum cost feasible service dispatch. Similar *vehicle routing* (or *vehicle scheduling*) problems constitute a stand-alone topic of investigation in Operations Research. In this case, we are dealing with a *real-time* routing problem, where the available time for finding a solution is strongly constrained. For the rest of this thesis, we shall use the term *Vehicle Dispatching Problem* (VDP) to refer to any of these snapshot problems. We will discuss an example in a moment.

A second issue to be addressed regards the embedding of a solution algorithm for the VDP into an *online strategy*. It might be a bad idea to change the schedule of the whole system at every new snapshot. Even if we were able to compute the optimum solution for each snapshot (which in high-load situations is unlikely to happen, due to the restricted running time), we have no guarantee that this would turn out to be the best choice in the end. Maybe it is better to incorporate incoming requests into the current plan and compute a new schedule only from time to time. We define the *Online Vehicle Dispatching Problem* (OLVDP) as the task of determining a good strategy for controlling the system in the long run. In particular, such a strategy has to determine how to integrate a new request into an existing dispatch, and when to globally recompute a new dispatch.

To conclude this initial description of the ADAC-Problem, remark that the input information we are dealing with can be classified into three groups according to its predictability. First, there are *deterministic* parameters which are known in advance with precision, like the shift-duration of the units, the unitary travel costs, etc. Then, there are some *stochastic* parameters that are not fixed beforehand, but may be estimated from information collected in the past, such as the travel time for getting to a certain position, the expected duration of service for a request, etc. Finally, information regarding the distribution of the requests in space and time is mainly of *uncertain* nature. Because of the high variance involved, this distribution cannot be predicted with reasonable exactitude from observations in the past.

The latter fact has some consequences on the design and evaluation of algorithms for the OLVDP. Instead of assuming requests will emerge adjusted to a specified probability distribution, we take a different approach, borrowed from game theory, and consider the OLVDP as a two-player game. On one side, there is the *online algorithm* that operates the dispatching system at ADAC according to a certain strategy. On the other side, there is an malicious *offline adversary* who decides when and where the next request will arise. The adversary wants to make evident to an observer (for instance to a customer), that the quality of the algorithm used at ADAC is poor. With this purpose, it issues the requests in such a way as to drive the online player into a trap and force it to take bad decisions. After that, the adversary solves the same problem instance itself, but knowing all requests in advance, and tells the observer how good the "correct" solution was to be. Conversely, the online algorithm must choose its strategy in such a manner, as to ensure the worst-case ratio between its own solution and that from the offline adversary will not be too bad. This technique is called *competitive analysis*, and we shall return to it in Chapter 3.

Solving the Snapshots

The VDP can be formulated mathematically as an integer program, and this will be the matter of further discussion in the next chapters. As a motivation, let us look at an example to introduce some relevant ideas and techniques. We cannot afford to solve here any of the practical instances appearing at ADAC, but we must restrict ourselves to something much smaller: assume at a certain stage there are three requests r_1, r_2 and r_3 standing in line, and two units of different types EWS, EV5 (these are real denominations) are currently active. Besides, a contractor TAX (for taxi) is available. The two first units may serve any of the requests, while TAX is only allowed to attend request r_2 due to technical constraints. Figure 1.4 shows how the requests and units are located.

The *operational costs* incurred by a vehicle during its duty depend both on the distance traveled and on the requests served. They are calculated in a different way for each type of vehicle, according to the formulas:

$$c_O := \begin{cases} 2.97\,t_D & \text{for EWS,} \\ 95 + t_S & \text{for EV5,} \\ 24 & \text{for TAX,} \end{cases}$$

where t_D and t_S represent the amount of time spent at driving and serving a request, respectively. In the figure, required travel times are shown on the edges of the graph, and required service times are depicted at the nodes. (Although the coefficients used here are fictive, cost structures like these are indeed present in the real-world instances at ADAC).

Costs are also incurred whenever some request r has to wait too long before being attended. At ADAC, such *lateness costs* are given by a function of the fol-

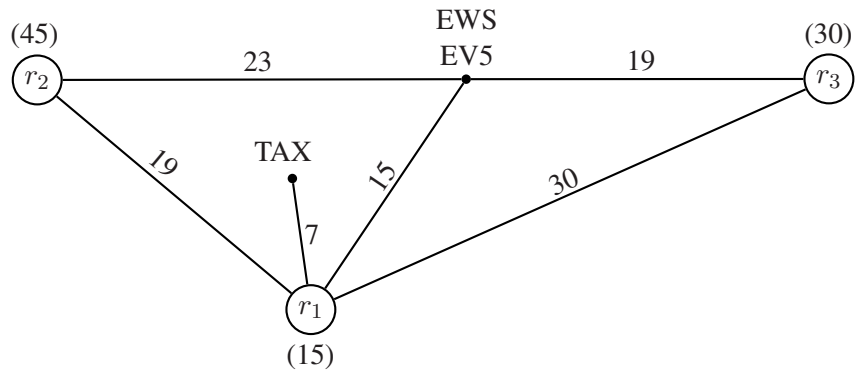


Figure 1.4: A small example of the VDP. Three requests r_1, r_2 and r_3 are waiting for service, two units of different types (EWS and EV5), and a contractor (TAX) are available. Weights on the edges represent travel times, node-weights expected service times.

lowing type:

$$c_L := \begin{cases} 0, & \text{for } t \leq 30, \\ 0.25\,(t-30) & \text{for } 30 < t \leq 60, \\ 3.5\,(t-60) & \text{for } t > 60, \end{cases}$$

where t specifies the time at which a unit arrives at r.

A feasible service tour T for a unit $u \in \{\text{EWS}, \text{EV5}\}$ is a route starting at the current position of u, visiting some requests in a specified order, and then returning to the initial position. Observe that this definition automatically fixes the times at which u reaches each of the requests and, hence, the corresponding lateness costs. The cost of T is defined to be the sum of all operational costs for u plus all lateness costs for the covered requests. T can be encoded by a $0/1$-vector of dimension five: the first three entries of this vector indicate which requests are served by the tour, and the last two entries indicate which unit drives the tour.[4] Figure 1.5 shows an example. A feasible contractor tour for TAX is also encoded as $0/1$-vector that indicates which requests are covered by it. Unlike a unit tour, however, this vector has no entry indicating which vehicle drives the tour, as we do not plan routes for the contractors.

A feasible dispatch is a set of service tours which partitions the requests and the units: each request must be covered exactly by one tour from the dispatch, and each unit must have exactly one tour to drive. The cost of a dispatch is defined as the sum of the costs of the tours contained in it. Thus, enumerating all 17 feasible tours, and assigning a binary variable $x_1, \ldots, x_{17}$ to each of them, our VDP can be

[4]The encoding of a tour as a binary vector does not account for the ordering of requests in it. This aspect must be addressed separately by the solution algorithm.

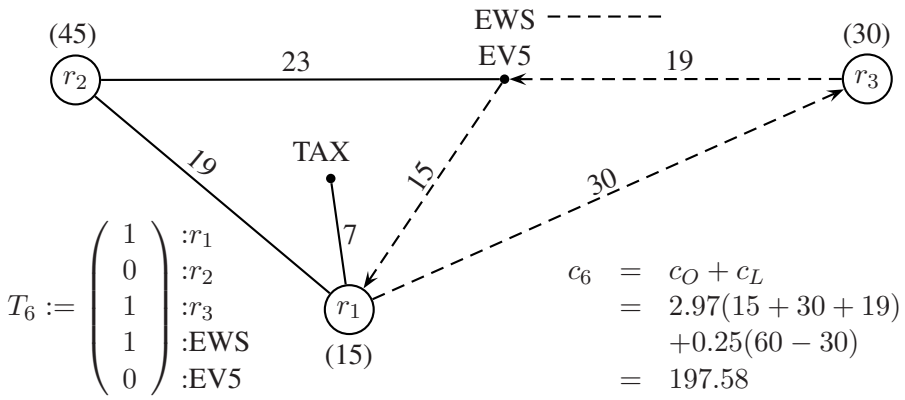


Figure 1.5: Encoding of a service tour as a $0/1$-vector. In tour T_6, EWS serves requests r_1 and r_3 in that order. T_6 is represented as an incidence vector and associated to a coefficient c_6 equal to its cost. To avoid ambiguity, we consider in this example only tours for which the ordering of the requests is optimal.

stated as the following integer program:

$$\min \sum_{j=1}^{17} c_j x_j$$

s.t.

$$\begin{array}{r} r_1: \\ r_2: \\ r_3: \\ \text{EWS}: \\ \text{EV5}: \end{array} \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix} \mathbf{x} = \mathbf{1},$$

$$x_j \in \{0, 1\}, \quad \forall\, 1 \leq j \leq 17,$$

where $\mathbf{x}^T = (x_1, \ldots, x_{17})$, c_j is the cost of the tour associated with the j-th column of the constraint matrix (computed from the formulas given before), and $\mathbf{1}$ is a vector full of ones, properly dimensioned. For simplicity, we have kept from explicitly writing the cost values down. Each binary variable indicates whether or not the corresponding tour is chosen to be in the dispatch. Columns 1 and 9 represent "empty" tours that cover no requests and have cost zero. They are used in the model in order to allow solutions where not all units participate. By inspection, the reader can check that a minimum cost dispatch is obtained setting $x_4 = x_{13} = 1$ and all other variables to zero; its cost is equal to 272.61 monetary units. This

means that EWS serves request r_3, and EV5 requests r_1 and r_2, while TAX does nothing.

Problems of this kind are known as *set partitioning problems*. They form a widely studied field in combinatorial optimization, where a lot of work has been invested both on the understanding of its theoretical properties and on the development of practical solution algorithms. Special methods, such as *column generation*, are required in order to address the huge instances that usually appear in connection with routing problems. (Just count how many feasible tours there could exist for 50 units and 100 requests). Again, all these aspects deserve better explanation and will be treated in more detail later.

Finally, we should mention here that general vehicle routing problems have an alternative formulation as nonlinear integer programs with a polynomial number of binary variables. Each variable indicates whether an arc is chosen to be in the route for a specified vehicle or not. We shall learn more about this model in Chapter 2. In the ADAC-Problem, however, the form of the lateness cost function makes a formulation over arc variables inapplicable.

Structure of this Thesis

In this thesis, our work on the ADAC-Problem and related topics is reported. Three threads of discussion will be pursued in a concurrent manner: The first one concerns the vehicle routing problem with time windows, for which several models, solution methods and applications are surveyed in Chapter 2. Many of the concepts introduced there will be picked up again many times in the subsequent chapters, ending up in Chapter 7, where an integer programming formulation and a solution algorithm for the VDP are considered.

The second thread evolves around online vehicle routing problems. It begins with a survey of them in Chapter 3, where at the same time the basic techniques of *competitive analysis* are developed. Chapter 5 then continues with some considerations on the competitivity of a simplified version of the ADAC-Problem , and Chapter 7 closes the topic with an aspect coming from the practical side: we evaluate online solution strategies for the OLVDP on the basis of simulations performed over previously recorded real-world data.

Finally, Chapters 4 and 6 are of more theoretical nature and can be regarded to a certain extent as a digression. They are devoted to set partitioning (and in particular set packing) problems which, as exposed above, appear within our solution scheme for the VDP. Due to specific features of the ADAC-Problem (for instance, the fact that in any good dispatch the service tours for the units are usually short) these set partitioning instances have a well determined form. Wondering if this could be exploited to further improve the performance of our solution algorithm, we were motivated to look into the polyhedral aspects of their structure in more detail. Although there does not seem to be any property which immediately leads to a breakthrough in efficiency, several interesting issues concerning the associated polyhedra were encountered, which we would like to share with the reader here.

Chapter 2

The Vehicle Routing Problem With Time Windows. A Survey.

2.1 Introduction

The vehicle routing problem with time windows (VRPTW) can be stated as the task of designing a minimum cost schedule for the delivery (resp. pick-up) of some goods to (resp. from) a set of customers, using a fleet of vehicles with limited capacities. Each customer has a certain demand and must be serviced exactly by one vehicle. Furthermore, the delivery (resp. pick-up) at a customer has to begin within a certain period of time called the customer's *time window*. (It is allowed, however, to arrive at a customer before the beginning of its time window and wait). All vehicles have to start and end their routes at specific locations called *depots*. The cost of a route involves usually a fixed component associated with the use of the vehicle and a variable one, which depends on the distance (or time) traveled.

The VRPTW arises in many practical situations. Time windows are a natural way to state problems that come up in certain environments where it is wished (or needed) to work on flexible time schedules. Specific examples are bank deliveries, postal deliveries, industrial garbage collection and routing and scheduling of school-buses. The problem itself is a generalization of the intensively investigated Vehicle Routing Problem VRP, which has been subject of many studies in the last three decades (Magnanti [1981], Assad et al. [1983], Laporte & Nobert [1987] and Laporte [1992] are survey papers). Also frequent in practical situations, but far less studied in theory, is the version of the problem that appears if so-called "soft time windows" are used. These may be violated at a cost given by some penalty function.

Desrosiers et al. [1995] have collected a comprehensive survey on the VRPTW and related problems. As far as possible, the notation and terms introduced there will be maintained throughout this chapter. We begin by stating their general non-linear formulation for the pick-up version of the problem. (The delivery version can be formulated in a very similar way).

Suppose the fleet consists of a set K of vehicles, from which at most v may be used simultaneously. Associated to each vehicle $k \in K$ is the following problem data:

a *capacity*	Q^k,
an *origin* depot	$o(k)$,
a *destination* depot	$d(k)$,
an *initial load*	$\ell_{o(k)}$,
a *final load*	$\ell_{d(k)}$,
a *start time window*	$[a_{o(k)}, b_{o(k)}]$, and,
a *return time window*	$[a_{d(k)}, b_{d(k)}]$.

In some instances, the origin and destination depots may correspond to the same physical location.

Similarly, let $N := \{1, \ldots, n\}$ be the set of customers. For every customer $i \in N$, two parameters are given:

a *demand*	ℓ_i, and,
a *time window*	$[a_i, b_i]$.

Define the directed graph $G = (V, A)$ as follows: The set

$$V := N \cup \{o(k), d(k) \ : \ k \in K\}$$

contains one node for each customer, plus nodes representing all (origin and destination) depots of the vehicles. We denote by $V^k \subseteq V$ the set $N \cup \{o(k), d(k)\}$. Given a vehicle $k \in K$ and an ordered pair of nodes $(i, j) \in V^k \times V^k$, let t_{ij}^k be the time required by k to travel from i to j, including any service time spent at i, in case $i \in N$. We say that (i, j) are compatible with respect to k if the time and capacity constraints allow this vehicle to attend j directly after i, i.e., if $a_i + t_{ij}^k \leq b_j$ and $\ell_i + \ell_j \leq Q^k$. For every $k \in K$, a set A^k of arcs is defined by

$$A^k := \{(i, j) \ : \ (i, j) \text{ is compatible with respect to } k\},$$

and $A := \cup_{k \in K} A^k$. Associated to each arc $(i, j) \in A^k$ is a weight c_{ij}^k equal to the travel cost incurred by k when moving from i to j. (Notice that service costs at customer i could also be considered here). Any fixed costs associated with the use of vehicle k are added to the weights $c_{o(k),i}^k$, $\forall i \in N$. Observe that G contains parallel arcs.

The mathematical programming formulation uses three types of variables: for every arc $(i, j) \in A^k$, a binary variable (called *flow* variable) X_{ij}^k indicates if (i, j) is used by the route scheduled for k or not. Besides, for any node $i \in V$ visited among this route, two other variables T_i^k and L_i^k reflect the time when service starts at i and the load of the vehicle *after* the pick-up. (For nodes not visited by the route, these variables hold arbitrary values). The VRPTW can then be formulated as follows:

$$\min \quad \sum_{k\in K}\sum_{(i,j)\in A^k} c_{ij}^k X_{ij}^k \tag{2.1}$$

subject to

$$\sum_{(i,j)\in A} X_{ij}^k = 1, \qquad \forall i \in N, \tag{2.2}$$

$$\sum_{k\in K}\sum_{(o(k),j)\in A^k} X_{o(k),j}^k \leq v, \tag{2.3}$$

$$\sum_{(o(k),j)\in A^k} X_{o(k),j}^k = 1, \qquad \forall k \in K, \tag{2.4}$$

$$\sum_{(i,j)\in A^k} X_{ij}^k - \sum_{(j,i)\in A^k} X_{ji}^k = 0, \quad \forall i \in N, \forall k \in K, \tag{2.5}$$

$$\sum_{(i,d(k))\in A^k} X_{i,d(k)}^k = 1, \qquad \forall k \in K, \tag{2.6}$$

$$X_{ij}^k(T_i^k + t_{ij}^k - T_j^k) \leq 0, \qquad \forall (i,j) \in A^k, \forall k \in K, \tag{2.7}$$

$$a_i \leq T_i^k \leq b_i, \qquad \forall i \in V^k, \forall k \in K, \tag{2.8}$$

$$X_{ij}^k(L_i^k + \ell_j - L_j^k) \leq 0, \qquad \forall (i,j) \in A^k, \forall k \in K, \tag{2.9}$$

$$\ell_i \leq L_i^k \leq Q^k, \qquad \forall i \in N \cup \{d(k)\}, \forall k \in K, \tag{2.10}$$

$$L_{o(k)}^k = \ell_{o(k)}, \qquad \forall k \in K, \tag{2.11}$$

$$X_{ij}^k \in \{0,1\}, \qquad \forall (i,j) \in A^k, \forall k \in K. \tag{2.12}$$

Due to the constraints (2.7) and (2.9) this model is nonlinear. The objective function (2.1) specifies the total cost of the pick-up schedule. Constraints (2.2) state that each customer has to be visited exactly once. Inequality (2.3) requires the number of vehicles used to be less or equal than v. Constraints (2.4) to (2.6) are flow constraints that characterize nonelementary directed paths[1] going from the origin to the destination depot of each vehicle. Subtours are eliminated by the other constraints, as we shall see later. Restrictions (2.7) - (2.8) and (2.9) - (2.11) specify the time windows and capacity requirements. Finally, (2.12) are the integrality constraints on the flow variables. Although many of these restrictions are redundant, they have been kept in the model for the sake of clarity.

Let's take a closer look at the restrictions (2.7). Consider any feasible solution to the model and define $\tilde{A}^k$ to be the set of all arcs used by a vehicle $k \in K$, i.e., let $\tilde{A}^k := \left\{ij \in A^k \,:\, X_{ij}^k = 1\right\}$. Constraints (2.7) require $\left(T_i^k\right)_{i\in V^k}$ to define a

[1] Given the directed graph $G(V, A)$, we denote by *nonelementary directed path from v_0 to v_k* a finite sequence $W = (v_0, v_1, \ldots, v_k)$ of nodes from V having the property $(v_i, v_{i+1}) \in A \; \forall\, 0 \leq i \leq k-1$. If the sequence contains no repetitions, we call it an *elementary directed path*.

potential on the subgraph $(V^k, \tilde{A}^k)$ with respect to the traveling times t_{ij}^k on the arcs. Since these times are all positive, $(V^k, \tilde{A}^k)$ has to be acyclic. Furthermore, combining this observation with the flow constraints (2.4) to (2.6), it follows that $(V^k, \tilde{A}^k)$ is a simple directed path from $o(k)$ to $d(k)$. Strict inequality may be obtained in (2.7) if a vehicle arrives too early at a customer and has to wait before starting the pick-up.

Making use of the fact that the flow variables X_{ij}^k are binary, it is possible to replace constraints (2.7) and (2.9) by the following linear inequalities:

$$T_i^k + t_{ij}^k - T_j^k \leq \left(1 - X_{ij}^k\right) M_{ij}^k, \forall (i,j) \in A^k, \forall k \in K \tag{2.13}$$

$$L_i^k + \ell_j - L_j^k \leq \left(1 - X_{ij}^k\right) Q^k, \quad \forall (i,j) \in A^k, \forall k \in K \tag{2.14}$$

where M_{ij}^k are sufficiently large constants. Whenever X_{ij}^k is equal to 0, these constraints become superfluous. On the other side, if $X_{ij}^k = 1$ they take exactly the same form as their nonlinear counterparts. Note that because of (2.8), it suffices to choose $M_{ij}^k := \max\ \{b_i + t_{ij}^k - a_j, 0\}$, $\forall (i,j) \in A^k$.

It is obvious to see that the VRPTW is an $\mathcal{NP}$-hard problem. The following simple construction, for example, gives a reduction from the ATSP: Given an instance of ATSP over the complete digraph $K = (V, A)$ with a nonnegative cost-function $\tilde{c}$ on A,

- choose the fleet size to be one,
- let $c := t := \tilde{c}$,
- choose one arbitrary vertex $v_0 \in V$ and "split" it into two vertices to represent the origin and destination depot of a vehicle,
- define on each node a (very wide) time window $[0, M]$, where M is the sum of the travel times over all arcs, and
- define the demands ℓ_i to be equal to 0 for each customer.

Solving the VRPTW over this new network gives an optimal solution for the ATSP. Moreover, Savelsbergh [1985] showed that even the task of finding a feasible solution for the case of just one uncapacitated vehicle is a $\mathcal{NP}$-complete problem itself .

Due to the enormous computational complexity of the VRPTW, it is not strange that solution methods for it were developed relatively late. In fact, Solomon [1987] was the first to generalize a number of known route construction heuristics for the VRP to incorporate time windows. Since then, but mainly during the last decade, some exact algorithms (or approximation methods of guaranteed quality) for the VRPTW have also been developed (see Kolen et al. [1987],

Desrochers et al. [1992], Halse [1992]). All of them rely on solving some relaxation of the model to get a lower bound that can then be used in a branch and bound scheme.

The next section starts by presenting a closer insight into some special cases of the VRPTW. A generalization of it, the so-called pick-up and delivery problem, is then described in Section 2.2.3. Section 2.3 focuses on the principal exact optimization approaches that have been developed for the VRPTW. A brief survey on some heuristics and meta-heuristics is finally given in Section 2.4.

2.2 Related Problems

2.2.1 Fixed Schedule Problems

Fixed schedule problems are special cases of the VRPTW, in which the service at a customer is not required to start within a time window, but rather at a certain precise moment. Obviously, this fact dramatically reduces the number of feasible vehicle routes and makes the time variables of the original model (see Section 2.1) superfluous, since they are forced to have constant values. In the following, we shall consider an even further simplified version of the problem and also drop capacity variables and constraints. The remaining linear model consists of the same objective (2.1) and constraints (2.2) - (2.6), and (2.12). The number of vehicles to be used might be fixed a priori, left free or be subject of minimization.

This kind of problem appears in several practical fields, as for example airline, rail, urban and school bus transportation. Desrochers et al. [1982] describe some real-world examples. In the case of airline transportation, flight legs (i.e., trips between two airports, with fixed schedule and required aircraft type) have to be concatenated to build routes for airplanes. In rail transportation, locomotives have to be assigned to haul train cars over a set of given trips. The problem in this case is a little more general, since a train may require more than one locomotive and, on the other side, an extra locomotive may be hauled in a train to cover other trips later. This situation can also be included in the model by making minimal changes in constraints (2.2). Finally, in school-bus and urban transportation, a set of trips with fixed start and end locations (in space and time) have to be attended using a fleet of buses.

The particular case when all vehicles start and end their routes at the same depot, or equivalently, when there are no restrictions about where a route has to start and end, is called the Single Depot Vehicle Scheduling Problem (SDVSP). Dantzig & Fulkerson [1954] were the first to formulate (and solve) the SDVSP as a minimum cost flow problem, showing thus its polynomial time complexity, which was explicitly stated later by Lenstra & Rinnooy Kan [1981]. Branco & Paixão [1987] realized that the structure of the problem closely resembles that of an assignment problem and developed a specialized algorithm for it.

On the other hand, Bertossi et al. [1987] showed that allowing two or more depots makes the problem $\mathcal{NP}$-hard. This version is then usually called the Mul-

tiple Depot Vehicle Scheduling Problem (MDVSP) and was formulated as an integer multi-commodity flow by Ribeiro & Soumis [1994]. Several heuristics have been proposed for the MDVSP. (see Bodin & Golden [1981], Assad et al. [1983] and Carraresi & Gallo [1984] for survey papers; and also Lamatsch [1992], Mesquita & Paixão [1992] and Dell'Amico et al. [1993], for newer heuristics not included there). The exact solution methods base upon computing lower bounds for the problem and using them in branch-and-bound enumeration schemes. We shall just mention two of them here.

The first one was proposed by Carpaneto et al. [1989]. Their lower bound comes from an additive scheme formerly introduced by Fischetti & Toth [1989], which basically combines an assignment bound with shortest paths bounds. The authors obtained an average integrality gap of 0.9% for a set of randomly generated problems involving up to 70 trips and 3 depots.

The second method was developed by Ribeiro & Soumis [1994] and is based on solving the linear relaxation of the model by a Dantzig-Wolfe decomposition procedure similar to the one which will be explained in Section 2.3.1. Again, the algorithm was tested only on randomly generated problems, but containing up to 250 trips and 6 depots, or 100 trips and 10 depots. An average gap of only 0.0008% was reported.

The difference between the integrality gap in the two methods was explained by Ribeiro & Soumis [1994], who proved the following result:

$$Z_{\text{ADD}} \leq Z_{\text{LP}} = Z_{\text{DW}} \leq Z_{\text{IP}},$$

where $Z_{\text{ADD}}, Z_{\text{LP}}, Z_{\text{DW}}$ and Z_{IP} are the values of the additive lower bound, the linear relaxation bound, the Dantzig-Wolfe bound and of the optimal solution, respectively. This means that the additive bound is not better than the LP bound. Even more, the authors showed that in the worst case, the ratio $Z_{\text{ADD}}/Z_{\text{LP}}$ can get arbitrarily small.

2.2.2 The Asymmetric Traveling Salesman Problem with Time Windows

Another special case of the VRPTW is obtained by relaxing from the original formulation (see Section 2.1) the capacity constraints (2.9) - (2.11), and fixing the number of available vehicles to one. In other words, we are looking for a route (i.e., a *tour*) that starts from the origin depot, visits all customers and returns to the destination depot, taking into account all time windows. This problem is called the asymmetric traveling salesman problem with time windows (ATSPTW). The word "asymmetric" is used to remark that we are working over a directed network. For the undirected case, the (symmetric) TSPTW is defined in an analogous way.

The ATSPTW comes up in a variety of real-world applications. It also constitutes an important element in some solution schemes for the VRPTW. The *cluster-first, route second* approach, for example, relies in heuristically assigning customers to vehicles at a first stage, and then solving one independent ATSPTW

for each vehicle to construct the routes. On the other side, the ATSPTW also appears as a stand-alone optimization problem in some situations, such as the control of automated manufacturing systems. Ascheuer et al. [1999], for instance, modeled in this way the problem of minimizing the unloaded travel time of a stacker crane within an automatic storage system in a warehouse. (See Ascheuer [1995] for a detailed description).

Desrochers et al. [1984] proved that the nonlinear formulation of the ATSPTW has an interesting property which is called *nonlinear integrality*. It states that if the problem is feasible, replacing the integrality constraints (2.12) on the flow variables by nonnegativity constraints does not alter the optimal solution value. To show this, suppose (X^*, T^*) is an optimal solution to the relaxed problem. If X^* is not integral, fix the values of T^* and and let $A^* := \{(i,j) \in A : (X^*)_{ij} > 0\}$ be the set of arcs used by the solution. (The vehicle index has been omitted for simplicity). Next, relax the time windows constraints (2.7) and (2.8), and solve the remaining problem restricted to the arcs of A^*. This is a min-cost flow problem over the network (V, A^*) with integral capacities, which is known to have an integral optimal solution $\bar{X}$. Since by construction X^* also defines a feasible flow in this network, it follows that the cost of $(\bar{X}, T^*)$ must be smaller than or equal to the cost of (X^*, T^*). On the other side, from (2.7) we have

$$\bar{X}_{ij} \neq 0 \Leftrightarrow X^*_{ij} \neq 0 \Leftrightarrow T_i + t_{ij} - T_j \leq 0$$

Thus, $(\bar{X}, T^*)$ is feasible for the original problem and therefore optimal. Remark, however, that integrality is no longer maintained in the linearized version of the model.

It is obvious that the ATSPTW, as a generalization of the ATSP, is $\mathcal{NP}$-complete. Savelsbergh [1985] even proved that finding a feasible solution constitutes itself an $\mathcal{NP}$-hard problem. He also proposed some tour improvement heuristics based on the idea of exchanging a certain number of arcs, which is quite a common approach for the classical ATSP.

Several authors have focused on developing exact enumeration algorithms for the ATSPTW. Besides of minimizing the *total cost* of the tour, the alternative version of minimizing the *total schedule time*, which consists of replacing the objective (2.1) by $T_d - T_o$, has also been considered.

In both cases, it has been observed that the running time required by every of these algorithms tends to increase rapidly as instances with wider time windows are considered. Therefore, many preprocessing techniques have been developed for "tightening up" time windows and removing unnecessary arcs iteratively. For example, if at a node $j \in V$,

$$b_j > \max \ \{b_i + t_{ij} : (i,j) \in A\}$$

then b_j may be decreased to the value of the right-hand side, since no feasible path can arrive at j at any time later. Reducing the time windows may in turn cause some arcs to become infeasible (we call an arc (i,j) infeasible if $a_i + t_{ij} > b_j$).

These arcs can be deleted, and the whole process can be repeated until no more improvements are possible. Langevin et al. [1993] derived some more sophisticated rules.

According to Desrosiers et al. [1995], a very simple branch-and-bound approach in which lower bounds are computed by relaxing time window constraints (2.7) - (2.8) and solving the remaining assignment problem, and branching is done on the X_{ij} variables, may provide a good algorithm in practice. The linear lower bound is at least as good as the assignment bound, but calculating it requires much more running time.

For the case of minimizing the total schedule time, and assuming that the traveling times are symmetric and satisfy the triangle inequality, Baker [1983] proposed a branch-and-bound algorithm where lower bounds are calculated by exploring a certain critical path in an acyclic network. This network is defined by precedence relationships that the time windows impose between the nodes. Branching is done under the assumption of further precedence relations.

State-space relaxation, a general relaxation method for a number of classical routing problems proposed by Christofides et al. [1981b] and which will discussed in Section 2.3.3, has also been used within a branch-and-bound scheme for the ATSPTW. (See Desrosiers et al. [1995] for an example in the case of minimizing the total schedule time). Moreover, under certain circumstances, the time windows may be tight enough as to reduce the number of feasible states and transitions sufficiently for direct dynamic programming methods to become viable. Dumas et al. [1995] further developed this idea in the context of minimizing the tour cost. They derived some tests which can be used to identify and eliminate states that have no chance of leading to an optimal solution. While their algorithm is still clearly exponential, they report nevertheless having solved large scale problems with up to 200 nodes (or up to 800 nodes if the density of the nodes in a geographical region is kept constant) to optimality without branching.

A quite different approach was proposed by Langevin et al. [1993], who extended a former formulation of Finke et al. [1984] for the classical TSP. In this model, the ATSPTW is presented as a two-commodity flow problem. One commodity Y has to be delivered at each node, while the same quantity of another commodity Z is picked-up. Thus, along any feasible tour, the total flow remains constant on each arc. The authors then associate flow quantities with traveling times as follows:

Minimize

$$\sum_{(i,j)\in A} c_{ij}(Y_{ij} + Z_{ij})/b_d + \sum_{i\in N\cup\{d\}} W_i \tag{2.15}$$

subject to

$$\sum_{j\in N\cup\{d\}} (Y_{ij} + Z_{ij})/b_d = 1, \qquad \forall i \in N \tag{2.16}$$

$$\sum_{j \in N \cup \{o\}} (Y_{ji} + Z_{ji})/b_d = 1, \qquad \forall i \in N \tag{2.17}$$

$$\sum_{i \in N} (Y_{o,i} + Z_{o,i})/b_d = 1, \tag{2.18}$$

$$\sum_{i \in N} (Y_{i,d} + Z_{i,d})/b_d = 1, \tag{2.19}$$

$$T_0 = \sum_{j \in N} Z_{o,j}, \tag{2.20}$$

$$T_i = \sum_{j \in N \cup \{d\}} Z_{ij}, \qquad \forall i \in N \tag{2.21}$$

$$T_i = \sum_{j \in N \cup \{o\}} \left((Z_{ji} + t_{ji})(Y_{ji} + Z_{ji})/b_d + W_i\right), \forall i \in N \tag{2.22}$$

$$T_d = \sum_{i \in N} \left((Z_{i,d} + t_{i,d})(Y_{i,d} + Z_{i,d})/b_d + W_d\right), \tag{2.23}$$

$$a_i \leq T_i \leq b_i, \qquad \forall i \in V \tag{2.24}$$

$$Y_{ij} \geq 0,\ Z_{ij} \geq 0, \qquad \forall (i,j) \in A \tag{2.25}$$

$$W_i \geq 0, \qquad \forall i \in N \cup \{d\} \tag{2.26}$$

$$(Y_{ij} + Z_{ij})/b_d \text{ binary}, \qquad \forall (i,j) \in A. \tag{2.27}$$

The objective function (2.15) combines traveling costs and waiting times W_i at the nodes. Constraints (2.16) - (2.19) are flow constraints that fix the total in- and outflow at each node to be equal to a parameter b_d. The integrality constraints (2.27) then require that each arc is either used at its full capacity b_d or not used at all. This prevents the flow from splitting. Constraints (2.20) - (2.23) then define the time variables in connection to the value of the commodity Z that is picked-up. The reader may think of it as if a certain amount of traveling, service and waiting time were "collected" at each node. Finally, (2.24) are the time windows and (2.25) - (2.26) the nonnegativity constraints. The parameter b_d imposes a bound on the maximal duration admitted for a tour, and it must therefore be set at a sufficiently large value in order to avoid eliminating feasible solutions.

To solve this model, a branch-and-bound approach may be used. Lower bounds can be computed by solving either the linear relaxation or the assignment problem obtained by dropping constraints (2.16) - (2.26). Again, the linear bound is at least as good as the assignment bound. A particular feature of this formulation is, however, that the linear relaxation involves only $O(n)$ constraints, and not $O(|A|)$ as it was the case before. Nevertheless, the performance of an algorithm based on it turned out to be rather modest for the practical instances investigated. The authors report having solved problems up to 60 nodes with a fixed starting time $T_0 = a_0 = b_0$.

Some well known related problems of the TSP have also been subject of re-

search in their constrained-time versions. They take into account special structures and alternative objective functions. Psaraftis et al. [1990] define an ordered set of points $N = \{1, 2, \ldots, n\}$ to be located in a shoreline if the distances c_{ij} between them satisfy the following conditions: For all $1 \leq i \leq k \leq j \leq n$,

$$c_{ii} = 0, \tag{2.28}$$
$$c_{ij} = c_{ji}, \tag{2.29}$$
$$c_{ij} \geq c_{ik}, \tag{2.30}$$
$$c_{ij} \geq c_{kj}, \tag{2.31}$$
$$c_{ij} \leq c_{ik} + c_{kj}. \tag{2.32}$$

In other words, c is a metric with two additional properties (2.30) and (2.31). The *shoreline* TSPTW, in its so-called path version, consists in finding the shortest path for a vehicle traveling at unit speed, which starts from node 1, visits all nodes within their time windows and finally arrives at n (the tour version includes returning back). Such problems arise, for example, in the routing and scheduling of cargo ships. The complexity of this problem is open, its classical counterpart (i.e., without time windows) is polynomially solvable. On the other side, if constraint (2.32) is forced to be fulfilled with equality, we arrive at the *straight-line* TSPTW. The authors gave an $O(n^2)$ algorithm for the path version of this problem, under the assumption that time constraints consist only of earliest pick-up times. Later, Tsitsiklis [1992] showed that when general time windows are present, or when service times are considered, the problem becomes $\mathcal{NP}$-complete.

Another related problem is the *traveling repairman problem with time windows* (TRPTW). It differs from the TSPTW only in the objective function: the aim here is to minimize the sum of completion times $\sum T_i$, or equivalently, the sum of flow times $\sum(T_i - a_i)$. Afrati et al. [1986] showed that the classical TRP can be solved in $O(n^2)$ time, and that the version where time windows consist of deadlines is $\mathcal{NP}$-complete but can be solved by a pseudo-polynomial algorithm. Later, Tsitsiklis [1992] proved that the case with general time windows is strongly $\mathcal{NP}$-complete.

Finally, let us mention that if the use of more than one vehicle is admitted, the problem is then called the *multiple* traveling salesman problem with time windows (m-ATSPTW). It is still a special case of the VRPTW, where the capacity constraints have been dropped, and the fleet of vehicles is homogeneous and shares the same depot. On the other side, the m-ATSPTW is a natural generalization of the fixed scheduling problems described in the last section. Consequently, its application areas are similar and cover aircraft, ship, school bus and urban bus scheduling. The solution methods are basically the same as those used for the VRPTW and which will be discussed in Section 2.3.

2.2.3 Pick-up and Delivery Problems

The formulation of the VRPTW at the beginning of this chapter was stated under the assumption that goods have to be picked-up from the customer and transported to the depot. The same model may still be applied for the problem of finding a minimum cost delivery schedule.

A slightly more general task is the so-called *simple back-hauling problem*. In this case, we admit both pick-ups and deliveries at the same time. The set N of customers consists of two subsets: the *line-haul* customers N^D, who are expecting to receive some goods from the depot, and *back-haul* customers N^P, who have to send some goods to the depot. A fundamental restriction is, however, that any vehicle must attend all deliveries before beginning with pick-ups. This problem may also be addressed by introducing a small change in the original model. All what is needed is to replace constraints (2.10) by

$$\ell_i \leq L_i^k \leq Q^k, \qquad \forall k \in K, \forall i \in N^D \tag{2.33}$$

$$Q^k + \ell_i \leq L_i^k \leq 2Q^k, \forall k \in K, \forall i \in N^P \cup \{d(k)\}\,. \tag{2.34}$$

At the delivery level, the restrictions (2.33) work exactly in the same manner as before. The load variable L_i^k indicates the total amount of goods delivered by vehicle k among the partial route that goes from its origin depot $o(k)$ to (and including) customer i. It must be larger than or equal to the demand ℓ_i at i, but may not exceed the capacity Q^k of the vehicle. Constraints (2.9) ensure that these variables are incremented adequately over the traversed arcs.

During the pick-up phase, however, the value L_i^k gives the total load collected by vehicle k up to the moment after having attended customer i, incremented by an offset of Q^k. This offset is used to introduce a precedence constraint between the nodes. Since the value of L_i^k must be smaller than Q^k for any $i \in N^D$, but larger than Q^k for any $i \in N^P$, and since, because of (2.9), this value can only increase after traversing an arc, once a node in N^P has been reached, no further delivery nodes can be visited. The restrictions (2.34) express the capacity windows for the pick-up nodes taking this offset into account. As the changes in the formulation are small, the same algorithms used for the VRPTW can be adapted to solve the simple back-hauling problem (see Gélinas et al. [1995]).

On the other side, if pick-up and delivery requests may be attended in any order, the capacity constraints require two variables: the first one records the maximum load on the vehicle up to the present, while the second one specifies the actual load after the last customer has been visited. The solution approaches needed are more complex. Halse [1992] describes some exact and approximate methods.

The situation gets more complicated if instead of two sets of (independent) pick-up and delivering requests, a single set of *transportation* requests is given, i.e., if each customer specifies a certain quantity of goods that has to be picked-up at one location and delivered at another, within established time windows. The task of planning an optimal set of routes in such an environment is called the *pick-up and*

delivery problem with time windows (PDPTW). As in the case of the VRPTW, the usual objectives to be minimized are both the fleet size and, for a fixed size, the operational costs. If instead of goods persons are transported, it is often also desirable to minimize the inconvenience created by pick-ups and deliveries made either sooner or later than wished by the customer . This version of the PDPTW is called the *dial-a-ride* problem and constitutes the core of research in this area.

The PDPTW can be formulated using a nonlinear model similar to the one introduced in Section 2.1. In this case, however, two nodes are defined for each customer, one for pick-up and one for delivery. To simplify the notation, these two nodes will be numbered as i and $n+i$, where n is the number of customers. Furthermore, N^P and N^D will denote the sets of all pick-up and delivery nodes, respectively, and $N := N^P \cup N^D$. The formulation has the following form:

Minimize

$$\sum_{k\in K}\sum_{(i,j)\in A^k} c_{ij}^k X_{ij}^k \tag{2.35}$$

subject to

$$\sum_{k\in K}\sum_{j\in N} X_{ij}^k = 1, \qquad \forall i \in N^P \tag{2.36}$$

$$\sum_{k\in K}\sum_{j\in N^P} X_{o(k),j}^k \le v, \tag{2.37}$$

$$\sum_{j\in N^P\cup\{d(k)\}} X_{o(k),j}^k = 1, \qquad \forall k \in K \tag{2.38}$$

$$\sum_{j\in N\cup\{o(k)\}} X_{ji}^k - \sum_{j\in N\cup\{d(k)\}} X_{ij}^k = 0, \forall i \in N, \forall k \in K \tag{2.39}$$

$$\sum_{i\in N^D\cup\{o(k)\}} X_{i,d(k)}^k = 1, \qquad \forall k \in K \tag{2.40}$$

$$X_{ij}^k(T_i^k + t_{ij}^k - T_j^k) \le 0, \qquad \forall (i,j) \in A^k, \forall k \in K \tag{2.41}$$

$$a_i \le T_i^k \le b_i, \qquad \forall i \in V^k, \forall k \in K \tag{2.42}$$

$$X_{ij}^k(L_i^k + \ell_j - L_j^k) \le 0, \qquad \forall (i,j) \in A^k, \forall k \in K \tag{2.43}$$

$$\ell_i \le L_i^k \le Q^k, \qquad \forall i \in N^P, \forall k \in K \tag{2.44}$$

$$0 \le L_{(n+i)}^k \le Q^k - \ell_i, \qquad \forall\, n+i \in N^D, \forall k \in K \tag{2.45}$$

$$L_{o(k)}^k = 0, \qquad \forall k \in K \tag{2.46}$$

$$T_i^k + t_{i,n+i}^k \le T_{n+i}^k, \qquad \forall i \in N^P, \forall k \in K \tag{2.47}$$

$$\sum_{j\in N} X_{ij}^k - \sum_{j\in N} X_{j,n+i}^k = 0, \qquad \forall i \in N^P, \forall k \in K \tag{2.48}$$

$$X_{ij}^k \ge 0, \qquad \forall (i,j) \in A^k, \forall k \in K \tag{2.49}$$

$$X_{ij}^k \text{ binary}, \qquad \forall (i,j) \in A^k, \forall k \in K \tag{2.50}$$

As in the case of the VRPTW, this formulation involves three type of variables: binary flow variables X_{ij}^k, time variables T_i^k and load variables L_i^k. Their meaning is exactly the same as before. The constraints of the former model are still present here: (2.36) specify that each customer has to be picked-up exactly once, (2.37) takes care of the number of vehicles used, (2.38) - (2.40) and (2.49) - (2.50) are multi-commodity flow constraints, and finally (2.41) - (2.42) and (2.43) - (2.46) are the time windows and capacity restrictions, respectively. Observe, however, that the capacity windows are defined differently at pick-up and delivery nodes, since the load values increase at the former, but decrease at the latter (the value of ℓ_i is negative for $i \in N^D$). There also two new kinds of constraints: the *coupling* constraints (2.48) specify that a customer must picked-up and delivered by the same vehicle, and the *precedence* constraints (2.47) guarantee that a pick-up has to be accomplished before the corresponding delivery.

As stated before, the dial-a-ride problem constitutes the main research subject in the area of the PDPTW. Early work was done already in the 70's to look for real-time solutions for the special case where only one vehicle is routed, called the 1-PDPTW. (See Wilson et al. [1971], Wilson & Weissberg [1976] and Wilson & Colvin [1977]). However, due to the size of practical problem instances, which frequently involve over 3,000 requests and relatively large numbers of requests per vehicle, most of the solution approaches are still based upon the use of heuristics, either as stand-alone methods or combined with exact optimization techniques. An example of the latter case is the so-called *cluster first, route second* algorithm, originally proposed by Sexton & Bodin [1986]. Basically, it consists in partitioning the set of requests into one cluster for each vehicle. In a second stage, a 1-PDPTW is solved for each of these clusters. The process and further refinements of it will be discussed with more detail in Section 2.4.

The 1-PDPTW can be seen as a TSPTW with additional capacity and precedence constraints. Psaraftis [1983] exploited this fact to design one of the first algorithms aiming at the minimization of total customer inconvenience. However, his dynamic programming approach had a computational complexity of $O(n^2 3^n)$ and was therefore restricted only to very small instances (at most 10 requests, to that time). Later, Sexton & Bodin [1985a,b] decoupled this problem into a routing master problem and a scheduling subproblem for a fixed route. Employing a heuristic version of Benders' decomposition, where only the subproblem is solved to optimality by an efficient network flow algorithm, they could treat real problems ranging from 7 to 20 customers in a matter of seconds on a UNIVAC 1100/81A computer. A similar approach was used by Sexton & Choi [1986] for the case when the objective function is a linear combination of the customer inconvenience and the total distance traveled. Desrosiers et al. [1986] considered minimizing only the travel distance and developed an optimal algorithm based on dynamic programming that was able to solve instances with 40 customers in less than 6 seconds on

a CYBER 173.

If several vehicles are considered, the exact optimization strategies are basically the same as for the VRPTW. Dumas et al. [1991] used a Dantzig-Wolfe decomposition schema (see Section 2.3.1) embedded into a branch-and-bound tree to solve two real life problems of 19 and 30 requests, as well as many randomly generated problems involving up to 55 requests. The real life instances were solved to optimality without branching, while in the other cases a gap between 0.6% and 3.2% on the total travel cost was obtained after the first cut. According to the authors, their algorithm is appropriate for instances of the PDPTW where the solutions require an average of at most 5 requests per vehicle.

2.3 Optimal Algorithms and Approximation Schemes

As stated in Section 2.1, all exact solution methods for the VRPTW employ branch-and-bound enumeration trees. A first, rather naive approach could be to use the lower bounds that come either from the linear or the network relaxation of the problem (i.e., the min-cost flow problem obtained by dropping constraints (2.5) and (2.7) - (2.11)). Desrochers et al. [1985] used the network relaxation to solve to optimality urban-bus scheduling problems involving up to 150 trips, either branching on the flow variables or by partitioning the time windows. However, as the time windows become wider, the size of the branch-and-bound tree grows too rapidly for this procedure to remain applicable. The linear lower bound does not help much either. Hence, alternative approaches are needed and some have been investigated in the last years. So far, the most promising ones are based either on Dantzig-Wolf decomposition or Lagrangian relaxation techniques. These will be described in the next two sections.

2.3.1 Danzig-Wolfe Decomposition

Dantzig-Wolfe decomposition is a classical procedure for solving linear problems that possess a certain block structure: relaxing a few constraints splits the problem into several smaller independent subproblems. The original method and its general aspects shall not be described here in detail, rather, the highlights of a specialized version for the VRPTW will be pointed out. Refer to Chvátal [1983] for further reading.

Let $x \in \mathbb{Q}^n$, $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $C \in \mathbb{Q}^{k \times n}$ and $d \in \mathbb{Q}^k$. Consider the following integer problem:

(IP)

Minimize

$$c^T x \tag{2.51}$$

subject to

$$Ax = b, \tag{2.52}$$

$$Cx = d, \tag{2.53}$$

$$x \geq 0, \tag{2.54}$$

$$x \text{ integer} . \tag{2.55}$$

Denote by P_{IP} and P_{LP} the polyhedra defined by the convex hull of the set of feasible solutions for IP and by the set of solutions of its linear relaxation LP, respectively. For the purposes of this discussion, it is possible to consider them as bounded polytopes. Moreover, define:

$$P_{\text{SP}} := \text{conv}\,\{x \in \mathbb{R}^n \,:\, Cx = d,\, x \geq 0,\, x \text{ integer}\}$$
$$P_{\text{MIP}} := \{x \in \mathbb{R}^n \,:\, Ax = b,\, x \in P_{\text{SP}}\}\,,$$

and call MIP the problem of minimizing $c^T x$ over P_{MIP}. Since $P_{\text{IP}} \subset P_{\text{MIP}} \subset P_{\text{LP}}$, it follows for the optimum values of the corresponding optimization problems that:

$$Z_{\text{LP}} \leq Z_{\text{MIP}} \leq Z_{\text{IP}}.$$

The basic idea of the Dantzig-Wolfe decomposition procedure consists of expressing the polytope P_{SP} as the convex combination of its vertices $v \in V$ and rewriting MIP, also known as the *master problem*, in the form:

(MIP)

Minimize

$$c^T x \tag{2.56}$$

subject to

$$Ax = b, \tag{2.57}$$

$$x = V\xi, \tag{2.58}$$

$$\mathbf{1}^T \xi = 1, \tag{2.59}$$

$$\xi \geq 0. \tag{2.60}$$

The usual convention to consider V both as a set of points and as a matrix containing the coordinates of these points as columns has also been adopted in this formulation. Replacing (2.58) in (2.56) and (2.57) yields now:

(MIP)
$$\text{Minimize} \quad c^T V\xi \tag{2.61}$$
$$\text{subject to} \quad AV\xi = b, \tag{2.62}$$
$$\mathbf{1}^T\xi = 1, \tag{2.63}$$
$$\xi \geq 0. \tag{2.64}$$

Obviously, the matrix V may contain so many columns that even the task of writing it down may become impossible from a practical point of view, not to mention solving MIP. Nevertheless, if the problem has a special structure, an approach called *column generation* can be used. It is based on the following observation: what the simplex algorithm needs in order to solve MIP is, besides from a starting basic solution which normally is not too hard to find, a way to decide at each iteration, if there exists some column with negative reduced cost outside the current basis.

Suppose some set $\hat{V} \subset V$ of columns has been generated (for example heuristically), and the modified version of MIP that consists of substituting V by $\hat{V}$ in (2.61) and (2.62), and restricting the variables accordingly has been solved. This problem is usually called the *restricted master problem*. Let $\lambda \in \mathbb{R}^m$ be the optimal dual multipliers associated to the equations (2.62) and $\beta \in \mathbb{R}$ the one for the equation (2.63). The problem now is to decide if there exists a column in $V \setminus \hat{V}$ whose reduced cost with regard to this multipliers is strictly negative. If there is not such a column, then the actual solution is optimal for MIP. The task of finding such a column can be accomplished by solving the following so-called *subproblem* SP:

(SP)
$$\text{Minimize} \quad c^T x - \lambda^T A x$$
$$\text{subject to} \quad x \in V.$$

Notice that SP consists of minimizing the linear function $(c^T - \lambda^T A)$ over the set of vertices of the integral polytope P_{SP}. We shall relax this problem a little and admit just any integral point of P_{SP}. Clearly, the optimal value of the objective function is the same in both cases. (Moreover, if P_{SP} is contained in some hypercube, as it is the case in many combinatorial optimization problems involving binary variables, then every integral point of P_{SP} is a vertex and both problems are equivalent). The subproblem takes now the form:

$$
\begin{aligned}
&\text{Minimize} \\
&\qquad c^T x - \lambda^T A x \\
\text{(SP)}\qquad &\text{subject to} \\
&\qquad Cx = d \\
&\qquad x \geq 0 \\
&\qquad x \text{ integer}
\end{aligned}
$$

Solving SP yields either a new column with negative reduced cost that can be used to create a new restricted master problem, or the proof that there is no such a column in V, which means that the current basic solution is optimal for MIP. In the first case, the new restricted master problem is re-solved and the whole process is repeated. Observe that it is possible, and usually also convenient, to add many columns with negative reduced costs at each iteration.

The quality of the bound Z_{MIP} obtained by Dantzig-Wolfe decomposition depends on the structure of the subproblem SP considered. If SP possesses the integrality property (i.e., if removing the integrality constraints does not alter the optimal solution value) then the polytopes P_{LP} and P_{MIP} are the same and Z_{MIP} coincides with the linear bound. On the other side, if SP does not have this property, part of the integrality gap between Z_{IP} and Z_{LP} might be explored by solving SP as an integer problem, provided this could be done (efficiently) employing some combinatorial approach. This is just the case with the VRPTW, where Z_{MIP} turns out to be significantly better than the linear bound.

Let IP be the linearized formulation of the VRPTW that was introduced in Section 2.1 and LP its linear relaxation. The objective function (2.1), the partitioning constraints (2.2) that require each customer to be served exactly once, and the constraints (2.3) that limit the maximal number of vehicles to be used are retained in the master problem MIP. All other constraints (including, of course, the integrality constraints) are moved to the subproblem SP. Observe that SP splits into $|K|$ disjoint subproblems SP(k), one for each vehicle $k \in K$. Moreover, a moment of thought reveals that SP(k) is the task of finding an elementary directed path from $o(k)$ to $d(k)$ that minimizes some arc-cost function and satisfies all time windows and capacity constraints. This is the so-called shortest path problem with resource constraints SPPRC which unfortunately is known to be (strongly) $\mathcal{NP}$-hard. Nevertheless, as we shall see in Section 2.3.4, if we relax the problem and allow paths to have cycles, it is possible to extend algorithms for the classical shortest path problem to the SPPRC, and it turns out that they work well for practical purposes.

For every vehicle $k \in K$, let Ω^k be the set of all feasible (i.e., satisfying time windows and capacity constraints) $[o(k), d(k)]$-paths. The (binary) flow variables in any solution of SP can be written in the form $(X_p^1, \ldots, X_p^{|K|})$, where X_p^k, $k \in K$ are (transposed) incidence vectors of paths $p^k \in \Omega^k$. On the other side, since SP splits into independent problems for each vehicle, any such a combination of feasible paths leads to a valid solution for the subproblem. The condition $X \in P_{\text{SP}}$

may therefore be expressed as:

$$
\begin{aligned}
X_{ij}^k &= \sum_{p\in\Omega^k} \xi_p^k x_{ijp}^k, && \forall ij \in A^k, \forall k \in K,\\
\sum_{p\in\Omega^k} \xi_p^k &= 1, && \forall k \in K,\\
\xi_p^k &\geq 0, && \forall p \in \Omega^k, \forall k \in K,
\end{aligned}
$$

where x_{ijp}^k, $ij \in A^k$ are the components of incidence vector X_p^k, and ξ_p^k are convex combination coefficients. With this substitution, the master problem MIP takes the form:

Minimize

$$\sum_{k\in K}\sum_{p\in\Omega^k} c_p^k \xi_p^k \tag{2.65}$$

subject to

$$\sum_{k\in K}\sum_{p\in\Omega^k} a_{ip}^k \xi_p^k = 1, \qquad \forall i \in N \tag{2.66}$$

$$\sum_{k\in K}\sum_{p\in\Omega^k} b_p^k \xi_p^k \leq v, \tag{2.67}$$

$$\sum_{p\in\Omega^k} \xi_p^k = 1, \qquad k \in K \tag{2.68}$$

$$\xi_p^k \geq 0, \qquad \forall k \in K, \forall p \in \Omega^k, \tag{2.69}$$

with parameters c_p^k, a_{ip}^k and b_p^k being defined as follows:

$$
\begin{aligned}
c_p^k &= \sum_{(i,j)\in A^k} c_{ij}^k x_{ijp}^k, && \forall p \in \Omega^k \forall k \in K,\\
a_{ip}^k &= \sum_{j\in N\cup\{d(k)\}} x_{ijp}^k, && \forall p \in \Omega^k, \forall k \in K, \forall i \in N,\\
b_p^k &= \sum_{j\in N} (x_{o(k),j,p}^k), && \forall p \in \Omega^k, \forall k \in K.
\end{aligned}
$$

The value of c_p^k gives the cost of a path $p \in \Omega^k$. a_{ij}^k is the number of times that customer i is visited among p (remember that paths are allowed to have cycles). Finally, coefficients b_p^k take only binary values due to the constraints (2.4). If b_p^k is equal to one, then path p visits some customer's node, otherwise it goes directly from the origin to the destination depot.

At each iteration, one instance of a restricted master problem is solved to optimality. The dual solution consists of multipliers $\alpha_i, i \in N$, β and $\gamma^k, k \in K$ for the constraints (2.66), (2.67) and (2.68), respectively. These multipliers are then used to redefine the arc-cost function as follows:

$$\bar{c}_{ij}^k = \begin{cases} c_{ij}^k - \alpha_i, & \text{if } i \in N, \\ c_{ij}^k - \beta - \gamma^k, & \text{if } i = o(k) \text{ and } j \in N, \\ c_{ij}^k - \gamma^k, & \text{if } i = o(k) \text{ and } j = d(k). \end{cases}$$

After that, new SPPRCs are solved an the whole process is repeated until the optimal value Z_{MIP} is found or a certain stop criterion is met. Z_{MIP} can then be used as a lower bound in a branch-and-bound algorithm. However, an additional complication comes here into play: after executing a branching decision and fixing some variable, the new optimal solution of the master problem may require a column that is still not present in the restricted problem. The reason for this is that the optimal dual prices, and therefore the reduced costs of the columns, are modified by the addition of a branching constraint. Thus, it might be necessary to execute a new round of column generation.

On the other side, fixing a path variable to zero alters the structure of the corresponding subproblem. As suggested by Hansen et al. [1991] and Maculan et al. [1992], instead of finding the shortest constrained path, it now consists of determining the second shortest path, then the third shortest path, and so on. A way to get around this obstacle has been proposed by Desrochers et al. [1992]: rather than fixing the path variables ξ_p^k, take the branching decisions on the original arc variables X_{ij}^k and transmit them to the subproblems. Thus, the structure of SP can be preserved among all the branch-and-bound tree.

The first attempt to apply Dantzig-Wolf decomposition for vehicle routing problems dates back to Appelgren [1969], who used it for solving a vessel scheduling problem with time windows. Later, Desrochers et al. [1984] applied it to instances of the m-TSPTW. Several 100-customer problems for the single depot VRPTW were solved by Desrochers et al. [1992] using the branching method mentioned above. The same idea has also been applied by Gélinas et al. [1995] in the context of the black-hauling problem.

For large problems, solving MIP to optimality may no longer be possible, not even using column generation, since the number of required iterations and the size of the reduced problems may become huge. In these cases, lower bounds on the value of Z_{MIP} can still be used to stop generating columns when no more substantial improvement is expected. Another approach is to employ *approximative* optimality criteria, i.e. to search for solutions that are "sufficiently" close to the optimum. Borndörfer et al. [2001] use this idea in their *adaptive* column generation scheme to solve very large problems (having up to 50,000 nodes) arising in the context of duty scheduling for urban transportation systems.

Finally, it should be mentioned here that the integrality gap in set partitioning problems that arise from Dantzig-Wolfe decomposition schemes have empirically

been observed to be very small, which explains the good behavior of branch-and-bound approaches. Bramel & Simchi-Levi [1997] investigated this fact further by employing probabilistic analysis in the particular case of the VRPTW when MIP is formulated as a set covering problem, the objective is the sum of route lengths and no maximal number of vehicles is specified. They showed that, for any distribution of customer locations and the other customer parameters (time windows, demands and service times), the relative integrality gap tends asymptotically to zero as the number of customers increases, i.e.,

$$\lim_{n\to\infty} \frac{1}{n} Z_{\text{MIP}} = \lim_{n\to\infty} \frac{1}{n} Z_{\text{IP}} \quad \text{(a.s.)}$$

The absolute integrality gap, on the other side, is at most of $O(n^{8/9})$.

2.3.2 Lagrangian Relaxation

Lagrangian relaxation may be applied to the VRPTW in many ways. Before discussing some of them, a brief general description of the method will be presented. Starting from the linear relaxation LP of the integer problem considered in the last section, and applying the Duality Theorem for linear programs twice, the following identity can derived:

$$\begin{aligned}
Z_{\text{LP}} &= \min_{x\in\mathbb{R}^n} \left\{ c^T x \,:\, Ax = b,\, Cx = d,\, x \geq 0 \right\} \\
&= \max_{\substack{\lambda \in \mathbb{R}^m, \\ \delta \in \mathbb{R}^r}} \left\{ \lambda^T b + \delta^T d \,:\, \lambda^T A + \delta^T C \leq c^T \right\} \\
&= \max_{\lambda\in\mathbb{R}^m} \left\{ \lambda^T b + \max_{\delta\in\mathbb{R}^r} \left\{ \delta^T d \,:\, \delta^T C \leq c^T - \lambda^T A \right\} \right\} \\
&= \max_{\lambda\in\mathbb{R}^m} \left\{ \lambda^T b + \min_{x\in\mathbb{R}^n} \left\{ (c^T - \lambda^T A)x \,:\, Cx = d,\, x \geq 0 \right\} \right\}
\end{aligned}$$

The *Lagrangian lower bound* of IP is obtained by adding an integrality condition on x in the inner minimization problem. This value will be denoted by Z_{L} and written as

$$Z_{\text{L}} = \max_{\lambda\in\mathbb{R}^m} F(\lambda),$$

where

$$\begin{aligned}
F(\lambda) &= \lambda^T b + \min_{x\in\mathbb{R}^n} \left\{ (c^T - \lambda^T A)x \,:\, Cx = d,\, x \geq 0, x \text{ integral} \right\} \\
&= \lambda^T b + \min_{x\in\mathbb{R}^n} \left\{ (c^T - \lambda^T A)x \,:\, x \in P_{\text{SP}} \right\}
\end{aligned}$$

As in the last section, the general problem decomposes into a main problem of maximizing $F(\lambda)$ over $\mathbb{R}^m$ and a subproblem of minimizing some linear function,

which in this case depends on λ, over the polytope P_{SP}. Notice also that $Z_{\text{LP}} \leq Z_{\text{L}}$ and that if the subproblem possesses the integrality property, both values are equal. On the other side, it is also straightforward to see that:

$$\begin{aligned} Z_{\text{L}} &= \max_{\lambda \in \mathbb{R}^m} \min_{x \in \mathbb{R}^n} \left\{ c^T x - \lambda^T (Ax - b) \,:\, x \in P_{\text{SP}} \right\} \\ &\leq \max_{\lambda \in \mathbb{R}^m} \min_{x \in \mathbb{R}^n} \left\{ c^T x - \lambda^T (Ax - b) \,:\, x \in P_{\text{SP}},\, Ax = b \right\} \\ &= \min_{x \in \mathbb{R}^n} \left\{ c^T x \,:\, x \in P_{\text{SP}}, Ax = b \right\} \\ &= Z_{\text{MIP}} \end{aligned} \tag{2.70}$$

Thus, $Z_{\text{LP}} \leq Z_{\text{L}} \leq Z_{\text{MIP}} \leq Z_{\text{IP}}$ and the Lagrangian bound is no better than the bound given by the Dantzig-Wolfe decomposition. Nevertheless, for many (large) practical applications, it is often convenient to use Z_{L} instead of Z_{MIP} because of efficiency reasons. To maximize $F(\lambda)$, several different techniques might be used, such as dual ascent methods, subgradient methods, bundle methods, etc.

There is also an interesting connection between Dantzig-Wolfe decomposition and Lagrangian relaxation. At each iteration of the column generation process, the Dantzig-Wolfe decomposition delivers a feasible primal solution for MIP. On the other side, the values of $F(\lambda)$ are equal to the objective values of feasible *dual* solutions of MIP, which can be obtained by fixing to λ the multipliers associated with the equations $Ax = b$ and maximizing over the remaining dual variables. (To prove this, imagine the condition $x \in P_{\text{SP}}$ written as a system of inequalities and construct the dual problem).

Using subproblems that possess the integrality property can be an advantage or a disadvantage, depending on the case. If the integrality gap $Z_{\text{IP}} - Z_{\text{LP}}$ is small, then the bounds obtained by approaches with and without integrality property will be similar, and of course we can expect the first ones to be easier to solve. On the contrary, as $Z_{\text{IP}} - Z_{\text{LP}}$ becomes larger, the quality of both bounds will split apart, since in the second case part of the integrality gap is explored during the solution of the subproblem. This fact can, in turn, affect the efficiency of a branch-and-bound algorithm dramatically.

One possible approach for the VRPTW could be to relax constraints (2.5), (2.13) and (2.14). Thus, the multi-commodity flow condition and the connection between flow, time and load variables disappear. SP then consists of three easy disjoint problems: a common min-cost flow problem on the flow variables, and trivial optimization problems over boxes for the time and load variables. This idea was analyzed by Desrosiers et al. [1988] for the m-TSPTW. However, since the subproblem possesses the integrality property, and since, as stated at the beginning of this section, the value of the linear relaxation lower bound was known to be too bad for a branch-and-bound algorithm, the authors discarded the idea and did not execute any numerical tests.

A more promising approach consists in relaxing constraints (2.2) and (2.3), which state that each customer has to be attended exactly once and that no more

than v vehicles may be used. As analyzed in the last section, the problem then decomposes into several constrained shortest path problems. Desrosiers et al. [1988] and Fisher et al. [1997] tested two versions of this method.

2.3.3 State-Space Relaxation

The concept of state-space arises in connection with dynamic programming, which is a general technique for optimum decision making due to Bellman [1957]. Virtually any combinatorial optimization problem might be modeled as a dynamic program by thinking of it as a sequential decision process. All relevant objects involved are considered as a system which is at all times in one of many possible states, each of them being described by the value of some variables, called *state variables*. From a given state, the system may move to another *successor* state at a certain cost. Both the set of successor states and the cost for reaching them are specified by some (usually recursive) rules called *transitions*. The optimization problem can then be stated as the task is to find a smallest-cost path of transitions between two prescribed initial and terminating states. This is usually accomplished by exploring the state-space exhaustively.

Unfortunately, apart from some few exceptions, this approach is not applicable for most combinatorial problems, since the number of states that have to be explored (the so-called *dimension* of the state-space) is simply too large. On the other side, in the few cases where the dimension of the state-space can be kept under control, dynamic programming algorithms have proven to be very fast.

This observation lead Christofides et al. [1981b] to propose a general relaxation procedure for many routing problems based on the idea of reducing the number of feasible states. Its approach is called state-space relaxation and consists in defining a mapping g from the original state-space S into a new state-space $\hat{S}$ of lower dimension. For every transition from a state S_1 to a state S_2 in S, a transition from $g(S_1)$ to $g(S_2)$ with the same cost is created. By solving the new smaller dynamic program, we get a lower bound on the original problem that can then be used by a branch-and-bound algorithm.

Following this idea, Kolen et al. [1987] developed a solution procedure for the special case of the VRPTW where an homogeneous vehicle fleet of capacity Q and sharing the same depot is considered. They constructed a branch-and-bound tree by fixing the flow variables at 0 or 1 in such an order that, at each node of the tree, the following holds:

- There is some set of fixed routes, each one starting and finishing at the depot.
- There is at most one partial route that starts from the depot but has not yet returned to it.
- There is a set of forbidden customers that may not be inserted as next in the partial route.

At each node of the tree, a state-space relaxation method is used to calculate a lower bound on all feasible extensions of the current incomplete solution. The condition that every customer has to be visited exactly once is dropped. Instead, all such extensions are considered, in which the total load of the routes is $\sum_{i \in N} \ell_i$ and the last customer visited is different for each route.

The dynamic program solved at the root node is defined as follows: every feasible state has the form (i, ℓ, k), $i \in N$, $0 \leq \ell \leq \sum_{i \in N} \ell_i$, $0 \leq k \leq v$ and corresponds to any situation where there are k complete routes covering a total load of ℓ and having different last customers. Those are required to be contained in the set $\{1, \ldots, i\}$. Two possible kinds of transitions are defined:

- From a state (i, ℓ, k) to a state $(i+1, \ell, k)$ at no cost.
- From a state (i, ℓ, k) to a state $(i+1, \ell+\ell', k+1)$ at a cost $c(i+1, \ell')$ defined as the minimal cost of a path from the origin to the destination depot that has $i+1$ as the last customer and covers a load of ℓ'. The value of such a path can be calculated by using the algorithms that will be described in Section 2.3.4.

The lower bound for the root node is calculated by determining the smallest-cost sequence of transitions from $(0, 0, 0)$ to any state of the set

$$\left\{ \left(n, \sum_{i \in N} \ell_i, k \right) : 1 \leq k \leq v \right\}$$

At the other nodes, if no partial route is present in the current solution, a smaller version of the same dynamic program is solved, considering only those customers who are still not covered. On the contrary, if there is such a partial route, a slight modification is needed. Let N^* be the set of unserved customers and v^* the number of remaining free vehicles. The dynamic program to be solved is defined on states of the form $(i, \ell, \tilde{\ell}, k)$, $i \in N^*$, $0 \leq \tilde{\ell} \leq \ell \leq \sum_{i \in N^*} \ell_i$, $0 \leq k \leq v^*$. Each of these states represents the situation when a total load of ℓ is covered by $k-1$ complete routes and a partial one, such that the load on this partial route is $\tilde{\ell}$ and the last customer visited by it is i. Two types of transitions are defined, depending if the current partial route is extended, or if it is closed and a new one is created. The initial state is determined according to the current solution values, while the admissible final states have the form:

$$\left(i, \sum_{i \in N^*} \ell_i, \tilde{\ell}, k \right),$$

where

$$i \in N^*$$
$$\sum_{i \in N^*} \ell_i - (k-1)Q \leq \tilde{\ell} \leq Q$$
$$1 \leq k \leq v^*$$

In the same article, the authors derive an analogous relaxation for solving the set partitioning master problem that appears in a Dantzig-Wolfe decomposition schema. Instead of dropping the integrality conditions, they relax the constraints requiring each customer to be attended exactly once, and replace them by one row aggregation constraint and $|N|$ constraints specifying that the last customers of the routes must be different. The resulting integral program is solved to optimality by dynamic programming. The lower bounds obtained are employed in the same branch-and-bound approach described before. A similar method was used by Bianco et al. [1989] in the context of urban crew scheduling.

2.3.4 The Shortest Path Problem with Resource Constraints

The shortest path problem with resource constraints (SPPRC) is a generalization of the classical shortest path problems known from basic graph theory. It appears as a subtask in almost all exact optimization approaches to routing problems. To formulate it, some preliminary definitions are needed. Let $D = (V, A)$ be a directed graph and $R = \{1, \dots s\}$ a finite set of so-called *resources*. In association with D and R, the following functions are defined:

- A *cost* function $c : A \longrightarrow \mathbb{Q}$ on the arcs of D.
- One *resource consumption* function $t^r : A \longrightarrow \mathbb{Q}$ for every resource $r \in R$.
- One *lower* (resp. *upper*) *limit* function $a^r : V \longrightarrow \mathbb{Q}$ (resp. $b^r : V \longrightarrow \mathbb{Q}$) for every resource $r \in R$. For each node $i \in V$, the interval $[a_i^r, b_i^r]$ will be called the r-th *resource window* of i.

Consider now a nonelementary directed path p in D as a sequence of nodes $i_0, i_1, \dots, i_H$ (possibly with repetitions). For any resource $r \in R$ and any node i_h among p, the *level* $T_{i_h}^r$ of r when reaching i_h is recursively defined as :

$$T_{i_h}^r = \max \left\{ T_{i_{h-1}}^r + t_{i_{h-1},i_h}^r,\ a_{i_h}^r \right\}$$

for $1 \leq h \leq H$. The initial level $T_{i_0}^k$ is arbitrarily chosen from the r-th resource window at the first node $[a_{i_0}^r; b_{i_0}^r]$. The path p will be called *feasible* if, for each resource $r \in R$, it is possible to choose this initial level in such a way that $a_{i_h}^r \leq T_{i_h}^r \leq b_{i_h}^r$ holds for every node i_h in the sequence. The SPPRC can then be stated as the task of finding a directed feasible path of minimal cost between two given nodes o and d.

The SPPRC appears in many contexts. For example, the shortest path problem with time windows (SPPTW) is the particular case when travel time is the only resource being considered. On the other side, the formulation of the VRPTW used in this chapter (see Section 2.1) leads to a SPPRC with two resources: time and vehicle's capacity. In more complex practical applications, the number of resources may increase further. (See Borndörfer et al. [2001] for models that involve more

than 10 resources to represent several side restrictions that appear in the context of duty scheduling for public transportation companies.)

Scaling by an appropriate factor, t^r, a^r and b^r may assumed to be integer-valued for all $r \in R$. It is then possible to reduce the SPPRC to a classical shortest path problem (SPP) on a new graph $\tilde{D} = (\tilde{V}, \tilde{A})$ defined as follows:

- Several copies $\hat{\imath}_1, \ldots, \hat{\imath}_{M(i)}$ of each node $i \in V$ are created. These copies correspond to all feasible combinations of allowed values for the resource levels at i. In the following, these combinations will be called *states*. Since a^r and b^r are integer-valued, there are "only" finitely many states. In fact,

$$M(i) = \sum_{r=1}^{s} (b_i^r - a_i^r + 1),\ \forall i \in V$$

- For each arc $ij \in A$, we introduce arcs with the same cost c_{ij} between all pairs $(\hat{\imath}_\kappa, \hat{\jmath}_\eta)$ of states associated to i and j that satisfy the following condition: starting from node i at state $\hat{\imath}_\kappa$, and traveling among ij, we reach node j at state $\hat{\jmath}_\eta$.

Given two nodes $o, d \in V$, it is straightforward to see that for every feasible (o, d)-path in D, there is a directed path with the same cost between certain states $\tilde{o}_\kappa$ and $\tilde{d}_\eta$ in $\tilde{D}$, and vice-versa. The usual algorithms for shortest path calculation could therefore be applied in $\tilde{D}$ to solve the SPPRC. However, there are two main difficulties that have to be considered.

The first one concerns the cost function. When the SPPRC is embedded as a subproblem in the calculation of a feasible path with minimal reduced cost for the VRPTW (see Section 2.3.1), the graph D will usually contain negative cost cycles. It is known that in such cases the (classical) SPP becomes $\mathcal{NP}$-hard. Fortunately, the additional resource constraints may help us out of this trouble. If at least one resource consumption function takes only positive values, (as it is usually the case with the travel time, for example) then it is easy to show that the transformed graph $\tilde{D}$ remains acyclic, so that Ford-Bellman's classical algorithm can still be employed and the results be translated back to D. However, as this is done, cycles may appear in the solution, since two different states in $\tilde{D}$ may correspond to the same node in D. Dror [1994], on the other side, showed that the task of finding an elementary constrained path in the case of just one resource is $\mathcal{NP}$-hard in the strong sense, which means that there are not even pseudopolynomial algorithms for it. Because of this reason, the algorithms for the VRPTW relax the subproblem and admit paths with cycles, though they might eliminate some of them "by hand".

The second difficulty regards the size of the auxiliary graph $\tilde{D}$. The transformation introduced above is by no means polynomial, since the number of nodes in $\tilde{D}$ depends on the *values* of the functions a^r and b^r which is exponentially larger than its encoding size. The algorithms presented here have therefore only a pseudopolynomial character. The set of possible states can grow rapidly out of control if we consider many resources and if the resource windows at the nodes are too

wide. Fortunately, for many practical applications concerning the VRPTW, the time windows will be narrow enough to allow this approach to be efficient.

A solution algorithm for the SPPRC will not create the graph $\tilde{D}$ explicitly. Instead, the states of a node are generated as they are needed and maintained as a set of so-called *labels*. More precisely, consider a feasible path p_κ in D that, starting from the origin o, reaches a node $i \in V$ consuming $T^r_{i,\kappa}$ units of resource $r \in R$ and at cost of $c_{i,\kappa}$. We shall associate to p_κ a label at i of the form $f(i,\kappa) = (c_{i,\kappa}, T^1_{i,\kappa}, \ldots, T^s_{i,\kappa})$. As in the classical algorithms for the SPP, these labels are created or adjusted during the solution process, until a certain optimality criterion is met. For simplicity, the index κ of a label will be omitted when there is no danger of confusion.

A simple, but important, observation stated at first by Aneja et al. [1983] helps to reduce drastically the number of labels that have to be taken into account. Given two labels $f(i,\kappa)$ and $f(i,\eta)$, we say that $f(i,\kappa)$ *dominates* $f(i,\eta)$ if the following holds:

$$
\begin{aligned}
& c_{i,\kappa} \leq c_{i,\eta}, \\
& T^r_{i,\kappa} \leq T^r_{i,\eta},\ \forall r \in R.
\end{aligned}
$$

A label is said to be *efficient* if there are no other labels that dominate it. A path using only efficient labels will be called an *efficient path*. It is straightforward to show that for every feasible path p_η between the origin o and any other node $i \in V$, there exists a feasible efficient (o,i)-path p_κ of at most the same cost. Therefore, when solving an instance of the SPPRC, only efficient labels have to be considered.

Denote by Q_i the set of all labels at a node i and let $\mathrm{EFF}(Q_i)$ be the subset of all efficient labels in Q_i. The three solution approaches for the SPPRC that will presented here rely on calculating $\mathrm{EFF}(Q_d)$ by dynamic programming. The shortest feasible path from o to d is then obtained from the label with the smaller cost in this set.

In the context of the SPPTW, Desrosiers et al. [1983] proposed their *label correcting* algorithm, which is in fact an adaptation of the classical Ford-Bellman-Moore algorithm (see Ford [1956], Bellman [1958], Moore [1957]). The version described here has slightly been modified to take additional resources into account.

Given a label $f(i,\kappa) = (c_{i,\kappa}, T^1_{i,\kappa}, \ldots, T^s_{i,\kappa})$ at a node i, and a node $j \in \Gamma^+(i)$ let $\mathrm{NEXT}\,(f(i,\kappa), j)$ be the function that returns the label $f(j,\eta)$ at j defined by:

$$
\begin{aligned}
& c_{j,\eta} = c_{i,\kappa} + c_{ij} \\
& T^r_{j,\eta} = \max\ \left\{T^r_{i,\kappa} + t^r_{ij},\ a^r\right\},\ \forall r \in R.
\end{aligned}
$$

If $f(j,\eta)$ is infeasible because of the violation of some resource window at j, then the return value of $\mathrm{NEXT}\,(f(i,\kappa), j)$ is set to be empty.

Algorithm 2.1 shows an outline of the label correcting method. Remark that the set Q_o is initialized using some valid label $f(o,\alpha)$. Normally, this label is fixed

{Initialization}
$Q_o \leftarrow \{(0, T^1_{o,\alpha}, \ldots, T^s_{o,\alpha})\}$;
$Q_i \leftarrow \{(\infty, a^1, \ldots, a^s)\}\ \forall i \in V \setminus \{o\}$;
$\mathcal{L} \leftarrow \{o\}$;
while $\mathcal{L} \neq \emptyset$ **do**
  {Selection of next node}
  Choose a node $i \in \mathcal{L}$; {Treatment of next node}
  for $j \in \Gamma^+(i)$ **do**
    $Q'_j \leftarrow \text{EFF}\,(\cup_\kappa \{\text{NEXT}\,(f(i,\kappa), j)\} \cup Q_j)$;
    if $Q'_j \neq Q_j$ **then**
      $Q_j \leftarrow Q'_j$;
      $\mathcal{L} \leftarrow \mathcal{L} \cup \{j\}$
    end if
  end for
  {Reduction of $\mathcal{L}$}
  $\mathcal{L} \leftarrow \mathcal{L} \setminus \{i\}$;
end while

Algorithm 2.1: A label correcting algorithm

by the problem instance. If this is not the case, the algorithm has to be run once for each possible initial state.

The algorithm maintains a list $\mathcal{L}$ of nodes that remain to be treated. Each time a node i is treated, all possible new labels for each of its successors $j \in \Gamma^+(i)$ are generated. Before adding any new labels to a set Q_j, however, a check is done to find out if they are not dominated by some older labels. For instances involving many resources, this check may require a lot of computational time. If (efficient) new labels are found, they are added to Q_j and j is inserted in the list $\mathcal{L}$. The execution finishes when the list gets empty.

For the remainder of this section, we shall restrict our attention to the case where at least one resource consumption function is either strictly positive or strictly negative; w.l.o.g., assume that t^1 is strictly positive. Since the value of $T^1_{i,\kappa}$ increases for each new label $f(i,\kappa)$ that is created, finiteness of the algorithm is then guaranteed by the upper bound limits b^1_i at the nodes. Obviously, the number of required iterations depends on the rule used for choosing the next node to be treated. FIFO and LIFO strategies have an exponential worst-case complexity. A better selecting criterion that ensures pseudopolynomiality has been discussed by Desrochers & Soumis [1988a] (see also Pape [1980] and Pallottino [1984]). Using this criterion, their implementation of the label correcting algorithm was able to solve instances of the SPPTW having 500 nodes, 50,000 arcs and 100 discrete time units in a few seconds.

In the case of the unconstrained SPP, if it is known a priori that the cost function is nonnegative, then this fact can be exploited to improve the efficiency of the label correcting method by specifying a certain order in which the nodes have to be

treated, so that no node needs to be considered more than once. The well-known Dijkstra's algorithm is based upon this idea. Since t^1 is strictly positive, it is natural to ask if a similar improvement may be attained here. This would require, however, to treat the labels independently of the nodes they belong to.

Desrochers & Soumis [1988a] proposed an approach in this sense. Their *label setting* algorithm provides a generalization of Dijkstra's algorithm for the case of the SPPTW. Again, the version presented in Algorithm 2.2 has been slightly modified to admit more than one constraining resource. Instead of maintaining a list $\mathcal{L}$ of nodes to be treated, this algorithm defines at each node a set P_i of *permanent* labels, i.e., labels that will not be changed anymore.

```
    {Initialization}
    Q_o ← {(0, T^1_{o,α}, ..., T^s_{o,α})};
    Q_i ← ∅, ∀i ∈ V \ {o};
    P_i ← ∅, ∀i ∈ V;
 5: while ∪_{i∈V}(Q_i \ P_i) ≠ ∅ do
      {Selection of next label}
      Choose a label f(i,κ) from ∪_{i∈V}(Q_i \ P_i) with T^1_{i,κ} minimal
      {Treatment of label f(i,κ)}
      for j ∈ Γ⁺(i) do
10:     Q_j ← EFF(Q_j ∪ NEXT(f(i,κ), j));
        P_i ← P_i ∪ f(i,κ)
      end for
    end while
```

Algorithm 2.2: A label setting algorithm

Finiteness is guaranteed by the same argument as in the label correcting algorithm. The key step in this method is the selection of the next label to be treated, since it involves comparing the value of $T^1_{i,\kappa}$ between all labels $f(i,\kappa)$ (of all nodes) which are still not marked as permanent. To accelerate the search, the idea of creating *buckets*, proposed by Denardo & Fox [1979] for the classical SPP, may be also applied here. A bucket is an unordered list that contains all labels whose values for the first resource lie within a specific interval. The h-th bucket, for example, would contain the labels $f(i,\kappa)$ for which $hw \leq T^1_{i,\kappa} < (h+1)w$, where w is a constant called the *width* of the buckets and is defined by:

$$w := \min_{ij \in A} t^1_{ij}$$

Thus, while a label within one bucket B is treated, all new labels created will belong to buckets that come after B. Instead of searching for the minimal among all untreated labels, we can then just process all the labels of the first bucket unorderly, then go to the second bucket, and so on. Of course, an extra computational cost has to be paid to insert the newly created labels in the correct buckets, but this is still more efficient than keeping the whole set of labels ordered. Using this approach,

the authors report having solved problems with up to 2,500 nodes and 250,000 arcs in a matter of seconds.

Both the label correcting and the label setting algorithms are so-called *reaching* approaches. They rely on extending feasible paths by examining at each iteration all successors of a certain node i, a task that implies updating the set of labels Q_j for all $j \in \Gamma^+(i)$. As the number s of resources increases, however, the computational cost of such operations grows rapidly, since determining $\text{EFF}\,(Q_j)$ implies comparing the level values $T^r_{j,\kappa}$ between *all* labels $f(j,\kappa) \in Q_j$ for *each* resource $r \in R$.

To reduce the frequency with which sets of labels are updated, Desrochers [1986] proposed a *pulling* approach for the SPPRC (see also Desrochers & Soumis [1988b] for the SPPTW). His method is described in Algorithm 2.3. The main idea is to treat a node j by examining the labels for all its predecessors $i \in \Gamma^-(j)$ and extending them as before. Since all new labels are created at j, only the set Q_j has to be updated during the iteration.

{Initialization}
$Q_o \leftarrow \left\{(0, T^1_{o,\alpha}, \ldots, T^s_{o,\alpha})\right\}$;
$\pi_o \leftarrow b^1_o$
$Q_i \leftarrow \emptyset, \forall i \in V \setminus \{o\}$;
$\pi_i \leftarrow a^1_i, \forall i \in V \setminus \{o\}$;
while $\exists j \in V$ with $\pi_j < b^1_j$ **do**
  {Selection of next node}
  Choose $j \in V$ with π_j minimal;
  {Treatment of node j}
  for $i \in \Gamma^-(j)$ **do**
    $Q_j \leftarrow Q_j \cup_\kappa \text{NEXT}\,(f(i,\kappa), j)$;
  end for
  $Q_j \leftarrow \text{EFF}\,(Q_j)$
  $\pi_j \leftarrow \min\ \left\{b^1_j, \pi_j + w\right\}$
end while

Algorithm 2.3: A pulling algorithm

As in the last case, w is the minimal value of the function t^1 over A. At each node i, a variable π_i is maintained to mark a subset of permanent labels. These are defined as the labels $f(i,\kappa)$ of Q_i for which the value concerning the first resource is less than π_i.

At each iteration, the node j with the smallest value of π_j is selected. By examining its predecessors, new labels are (possibly) created at j and Q_j is updated. Minimality of π_j guarantees that no further labels $f(j,\kappa)$ can be generated, for which $T^1_{j,\kappa} < \pi_j + w$ holds. The value of π_j is therefore increased by w and a new iteration starts. After finitely many steps, the values π_i will reach the upper bounds b^1_i for all nodes $i \in V$, and the algorithm will terminate. Notice that, during the iteration described above, it suffices to generate only such labels $f(j,\kappa)$ for which

$T^1_{j,\kappa}$ lies within the current "bucket" $[\pi_j; \pi_j + w)$.

An efficient implementation of the pulling process is described by Desrochers [1986]. He uses a primal-dual approach to identify permanent and nonpermanent labels. The computation of efficient subsets also takes advantage of specialized data structures to obtain a low complexity. He reports having solved problems with 1,000 nodes, 50,000 arcs and 5 resource constraints in less than a minute.

Several special cases of the SPPRC and the SPPTW have also been subject of research. Consider, for example, the case of minimizing the total path duration, while meeting the time windows at the nodes. This can be modeled as a SPPTW, where the labels (c_i, T_i) satisfy the relation $c_i = T_i - T_o$. It is easy to see that in this case, there can be only one efficient label at each node. The algorithms for the classical SPP may therefore be adapted to solve this problem.

On the other hand, if a path of maximal duration that fulfills the time window requirements is sought (a task called the *critical path problem*), the cost function can be seen as the negative value of the duration, and the same approach as before yields labels of the form (c_i, T_i), where $c_i = -(T_i - T_o)$. In this case, however, it is no longer possible to reduce the problem to a classical SPP. Anyway, the pseudopolynomial algorithms presented in this section may still be used.

As stated before, whenever cycles of negative cost are present, the problem of finding an elementary shortest path is $\mathcal{NP}$-hard in the strong sense. However, several authors have considered the relaxed problem of finding a shortest 2-cycle free path, i.e., a path without cycles containing just two nodes. Houck et al. [1980] and Christofides et al. [1981a] gave algorithms for the unconstrained case. These have been extended for the SPPTW by Kolen et al. [1987] and Desrochers et al. [1992]. The methods require duplicating the number of labels and will not be discussed in detail here.

Sometimes it is desirable that the value of the resource levels T^r_i among the nodes of a path lie as close as possible to the allowed lower or upper bounds. For example, in certain instances of the SPPTW, time windows may model intervals at which customers are waiting for service. Obviously, it is wished to reduce their waiting times by arriving as early as it is permitted. This problem may be modeled by introducing a nondecreasing penalty cost function on the arrival time. The algorithms presented in this section may still be used, provided the cost calculation is updated appropriately. On the other side, Salomon et al. [1997] found an application in the manufacturing context where a penalty function with nonpositive slope was required. The authors describe a transformation to deal with this case. Finally, when penalty functions with mixed positive and negative slopes are considered,Ioachim et al. [1994] showed that the algorithmic difficulties increase substantially.

2.4 Heuristics and Meta-Heuristics

Due to the huge computational complexity of time constrained routing problems, most real world applications are still solved using heuristics. Moreover, heuristics also often appear as subroutines within exact algorithms, for instance during the initialization phase, or to round fractional solutions obtained from a linear relaxation of the problem. The development and analysis of heuristics capable of coping with realistic problem sizes has in fact been one traditional focus of research in the area of the VRPTW. Several *route construction* and *route improvement* algorithms have been proposed during the last decades.

Route construction heuristics generate a feasible solution to the VRPTW by the iterative insertion of all customers. Sequential insertion methods create one route at a time, whereas parallel construction algorithms build several routes simultaneously. Both the next customer to be inserted, as well as the route and/or position where insertion shall occur are determined using various criteria. Approaches based on maximizing savings, minimizing additional cost and time, and nearest neighbor concepts have been proposed.

Solomon [1987] described a two-phase sequential algorithm in which the best insertion position for each unrouted customer is computed at first, and then this information is used to select the next customer to be inserted. This idea turned out to be very effective in the instances where it was tested. Roy et al. [1984a] and Roy et al. [1984b] developed a parallel insertion procedure for the dial-a-ride problem which constructs routes using certain proximity criteria. Similar approaches have been utilized by Jaw et al. [1986] and Madsen et al. [1995] to minimize a linear combination of customer inconvenience and travel costs.

The route improvement heuristics, on the other side, start from a given feasible solution and enhance it by a sequence of local modifications, usually until a so-called *local optimum* is reached. One classical idea in this context consists in replacing k arcs from the current solution with k other ones that are presently not used. This approach is called k-interchange and was inspired by early work from Lin & Kernighan [1973] and Lin [1965]. Based upon their ideas, improvement procedures for the VRPTW (see Russell [1977], Cook & Russell [1978] and Baker & Schaffer [1986]) and for the m-TSPTW (see Potvin et al. [1989]) have been developed. A main limitation of the method is, however, that large amount of processing time is required to examine all possible exchanges. To overcome this difficulty, the OR-opt procedure (Or [1976]) restricts the search only to those exchanges where one, two or three adjacent customers are moved to another position within the same route and inserted after another customer who meets a certain proximity criterion. This idea was applied successfully by Solomon et al. [1988] in an algorithm for the VRPTW. Savelsbergh [1992] addressed several additional issues concerning the efficiency of edge-exchanges for the case of minimizing route duration.

Another way of improving a solution was proposed by Thompson & Psaraftis [1993] based on their concept of cyclic η-transfers. These consist in moving η

customers from the route served by vehicle k to the route served by vehicle $\delta(k)$, where δ is a cyclic permutation on the set of vehicles. An OR-opt similar procedure can be used to reduce the number of exchanges to be examined,

Usually, heuristic algorithms for the VRPTW and related problems combine one or more route construction procedures that generate some initial schedules with local improvement methods that work on them. Potvin & Rousseau [1993], for example, integrated parallel insertion with OR-opt techniques. Similarly, in the "greedy randomized adaptive search procedure" (GRASP) of Kontoravdis & Bard [1995], vehicle routes are first constructed by a randomized greedy method and then improved via local search. On the other side, Russell [1995] has examined the possibility of using improvement criteria already at the route construction level to obtain what he calls "hybrid heuristics".

During the last decade, meta-heuristics have constituted a popular framework for the most local improvement methods. Much effort has been invested in developing and fine tuning specialized versions of general techniques like tabu search, simulated annealing, genetic algorithms, etc. We refer the reader to Gendreau et al. [1997] for a survey in this area. However, it must be observed here that the expectation generated around meta-heuristics has turned out to be exaggerated in many cases. While having achieved good results in some important practical instances, meta-heuristics have two fundamental drawbacks when compared with exact optimization approaches: they are unable to provide lower bounds or performance guarantees, and their algorithmic behavior usually depends on the value of some "meta-parameters" that have to be calibrated on a trial-and-error basis. It is by no means certain that the best suited set of values for some problem instances will still perform good on different ones. Of course, exact approaches have on their side the disadvantage of being limited to relative small problem sizes, but even this could change in the near future (see Borndörfer et al. [2001] for an example).

The most challenging routing problems arise in the context of dial-a-ride applications. The additional precedence and coupling constraints (see Section 2.2.3) dramatically increase the complexity of the route generation. In their *cluster first, route second* approach, Sexton & Bodin [1986] were the first to suggest a nowadays classical procedure for solving large practical instances. They proposed to partition the set of requests in so-called *clusters*, each of them associated to a vehicle. The routing within these clusters is accomplished at a second phase, and consists of solving independent single vehicle PDPTWs. A similar idea was followed by Koskosidis et al. [1992] to develop an iterative optimization-based heuristic for the VRPTW. The problem is decomposed into an assignment phase and a series of TSPTW. At each new iteration, the assignment is recalculated by solving a capacitated clustering problem.

A main limitation of the former methodology lies in the difficulty to construct a good clustering without relying on routing information. Dumas et al. [1989] proposed an improvement based on moving part of the clustering process into the routing process. Instead of creating one cluster for each vehicle, they devised heuristic algorithms that group together customers who can efficiently be served by a seg-

ment of a route to form what they call *mini-clusters*. In the second phase, a routing problem is solved on these mini-clusters. This is still a multi-vehicle problem, but some of the complicated local constraints like coupling and precedence are already satisfied a priori. In fact, a moment of thought reveals that what remains is a m-TSPTW for an heterogeneous vehicle fleet, which the authors solved using the methods discussed in Section 2.3. To build the mini-clusters, they employed a sequential insertion heuristic. Later, Desrosiers et al. [1991] replaced it by parallel insertion techniques and, finally, Desrosiers, Dumas, Ioachim & Solomon [1991] employed an exact optimization method based on set partitioning and column generation.

A similar approach was used for the optimization of Berlin's Telebus transportation system for handicapped people, a dial-a-ride problem involving instances with about 1,500 requests and 100 vehicles. (See Borndörfer [1998] for a detailed description of the project). The structure of the problem allowed the enumeration of all possible mini-clusters - about 100,000 in the most cases - in a couple of minutes. For the solution of the corresponding huge partitioning problem, the author devised a specialized code that was able, in all test instances, to deliver clusterings within the 1% optimality gap in less than three minutes on a Sun UltraSparc 1 / 170E. Using the best mini-clustering found, vehicle routes were generated heuristically and a new set partitioning problem was then solved to obtain the final schedule.

Chapter 3

Online Vehicle Routing. A Survey.

3.1 Introduction

One typical question asked by foreign students visiting Berlin concerns their mobility. Berlin's public transportation company, the *Berliner Verkehrsbetriebe* (BVG), offers a variety of tickets and fares for traveling in and around Berlin. Their prices range from €1.20 for a short-trip ticket (valid for 3 or fewer subway stations) to €72.50 for a monthly ticket covering all three fare zones. Alternatively, the two "canonical" transportation possibilities are also available here: to lend a bike from a friend, or to walk. However, especially during the winter months – and this is true not only for ecuadorian students – the two latter options may require a considerably high level of physical fitness and mental courage.

Now consider the situation of a student that has just arrived for a stay of, say, one month. He is wondering which is the right strategy for him. Should he buy a monthly ticket directly? Or is it better to try to go by bike? And what about a mixed choice (e.g., bike for the route to work, BVG for going out at night)? The particular feature that makes this decision so difficult is the fact that it has to be taken on the basis of *incomplete input information*. For instance, the student has no way to know in advance how the weather conditions during the two months will be, and if he will be able to travel by bike all the time, or if he will often be forced to pay fares for single trips, spending in the end more than what a monthly ticket would have costed. It is obvious that any decision he takes may later turn out to be a wrong one. However, the student does not want to give up. He admits that luck plays an important role, but he also realizes that some strategies seem to be more promising than others; and hence he is looking for a criterion to evaluate them.

Similar problems often occur in practical applications that operate in a real-time environment. During the last decade, there have been several approaches to model and study them from a mathematical point of view, as well as to establish a formal background for their analysis. This effort has given birth to a new branch in

operations research called *online optimization*. In plain words, online optimization problems are characterized by the fact that their input data is not explicitly given to a solution algorithm from the beginning. On the contrary, the solution process has a dynamic nature, with the *online algorithm* required to take immediate decisions on one side, and new input information arising on the other side.

As common to any new research area, the theoretical framework for online optimization is still "under construction", and many (even basic) issues have still not been settled in a definitive manner. For instance, there is no clear agreement on all fundamental concepts, and universally accepted ways of proceeding are also missing. This includes such questions as for example determining the quality of an online algorithm. No unified idea on what should be called a "good" online algorithm exists, but rather diverse criteria have been proposed that depend on the specific applications. Nevertheless, pioneer work is producing its first fruits, and increasingly many interesting results and practical experiences are being collected. Comprehensive reviews can be found in Borodin & El-Yaniv [1998] and Fiat & Woeginger [1998]. Krumke [2002] discusses various issues concerning the topic of competitive analysis, its limitations and alternative models. Finally, Krumke & Rambau [2002] give an introductory survey in the form of lecture notes.

The aim of this chapter is to focus on online optimization problems related to vehicle routing. Section 3.2 will introduce some notation and preliminary concepts, while the main problems and results in the area of online transportation, as well as some practical applications, are addressed in Section 3.3.

3.2 Basic Concepts and Notation

3.2.1 Online Problems and Online Algorithms

There are a couple of ways – called *online paradigms* – in which an online optimization problem and an online algorithm may be presented. The most common two are the sequence model and the time stamp model.

In the *sequence model*, the problem data is given by a sequence of so-called *requests*, which can be seen as "elementary pieces" of information. An online algorithm receives these requests one after another and has to compute a sequence of answers to the problem. Each new request requires the *immediate* calculation of a new answer, using only the information contained in those requests that are already known to the algorithm.

Alternatively, requests in the *time stamp model* are not ordered within a sequence, but are rather labeled by certain *release times* at which they become known to the online algorithm. The algorithm itself must give answers for the problem at certain moments in time, using only the information that has already been released. The principal change with respect to the sequence model lies in the fact that here the online algorithm may "collect" a couple of requests before taking an action.

Independently of the online paradigm used, most online problems can be stated as *request-answer games*, as shown by Ben-David et al. [1994]:

3.2.1 Definition (Request-Answer Game) *A* request-answer game $(R, \mathcal{A}, \mathcal{C})$ *consists of a request set* R*, a sequence* $\mathcal{A} = A_1, A_2, \ldots$ *of nonempty answer sets and a sequence* $\mathcal{C} = C_1, C_2, \ldots$ *of cost functions of the form*

$$C_i : R^i \times A_1 \times \cdots \times A_i \to \mathbb{R}_+ \cup \{+\infty\},$$

where $R^i = R \times R \times \cdots \times R$ *denotes the* i*-th Cartesian product of* R *with itself, i.e.,* R^i *contains all sequences of length* i *made from elements of* R*.*

The input for any online algorithm consists in a sequence $(r_1, r_2, \ldots)$ of requests from R. After the request r_i has been issued, the algorithm is required to choose one answer from the set A_i. Associated to each sequence of answers is an (accumulated) cost, given by a function C_i. The objective is to choose answers in such a way that the accumulated cost is minimized. The key point is that C_i depends not only on the current answer, but also on *all previous* requests and answers. Hence, decisions taken at some stage influence the cost for future stages.

An online algorithm which chooses the answers in some prescribed way, based only on the information revealed by previously issued requests, is called a *deterministic online algorithm*. Deterministic algorithms can be considered as functions:

3.2.2 Definition (Deterministic Online Algorithm) *A* deterministic online algorithm ALG *for a request-answer game* $(R, \mathcal{A}, \mathcal{C})$ *is a sequence* $(f_1, f_2, \ldots)$ *of functions having the form:*

$$f_i : R^i \to A_i, \, \forall i \in \mathbb{N}.$$

Each function f_i maps a sequence R^i of previously issued requests onto an answer from the current feasible answer set A_i. Hence, given a deterministic online algorithm ALG, a finite input sequence σ of requests automatically fixes a finite sequence of answers. Moreover, the pair (ALG, σ) determines a value for the accumulated cost:

3.2.3 Definition (Cost) *Given an* input sequence $\sigma \in R^n$*, with* $\sigma := r_1, \ldots, r_n$*, the value*

$$\mathsf{ALG}(\sigma) := C_n(\sigma, f_1(r_1), \ldots, f_n(r_1, \ldots, r_n))$$

is called the cost *of* ALG *with respect to* σ*.*

The example described at the beginning of this chapter – which we call the *BVG-ticket problem* in the sequel, for short – may be also modeled as a request-answer game. For simplicity, we assume that only three alternatives are relevant for the student each time he needs to travel to some destination: to buy a single-trip ticket for €1, to buy the monthly ticket for €30, or to get there by bike. (The real fares are different, but their relation is approximately the same). Moreover, whenever the weather conditions allow him to travel by bike, he will do it, as there are no (monetary) costs involved in that action. Thus, the only interesting requests are those *bad-weather requests* which require him to choose between the two tickets.

The BVG-ticket problem is the request-answer game $(R, \mathcal{A}, \mathcal{C})$, where

- $R = \{r\}$ is a set with only one element, which represents a bad-weather request. Any input sequence $\sigma := (r_1, \ldots, r_n)$ is made from repetitions of r.

- The answer sets in $\mathcal{A}$ contain the alternatives available to the student each time a bad-weather request is issued; each of them is equal to

$$A := \{\text{buy-single}, \text{buy-monthly}, \text{use-monthly}\}.$$

  However, we must take care of preventing the student from choosing the answer "use-monthly" as long as he has not yet selected "buy-monthly". This is done buy introducing penalizations in the cost function, as indicated below.

- For any $i \in \mathbb{N}$, the cost function C_i gives the total amount of money that the student has spent for transportation during the first i requests (including the i-th one). As stated before, C_i depends both on the requests issued so far and on the sequence of decisions $f_1(r_1), \ldots, f_i(r_1, \ldots, r_i)$ taken by the student. Moreover, the cost is defined to be equal to $+\infty$ whenever the students chooses "use-monthly" without having chosen "buy-monthly" before.

  Let $B_i(f_1, \ldots, f_i)$ be equal to smallest index in $\{1, \ldots, i\}$ such that the answer returned by the function $f_{B_i}(r_1, \ldots, r_{B_i})$ is equal to "buy-monthly". If no such index exists, then $B_i := +\infty$. Similarly, let U_i be the index corresponding to first answer in $f_1(r_1), \ldots, f_i(r_1, \ldots, r_i)$ which is equal to "use-monthly", with $U_i := +\infty$ if such an answer has not yet been chosen. Then C_i can be computed according to the formula:

$$C_i(r_1, \ldots, r_i, f_1, \ldots, f_i) = \begin{cases} |S_i| + 30\,|M_i|\,, & \text{if } B_i < U_i, \\ +\infty, & \text{otherwise.} \end{cases}$$

  Here,

$$S_i := \{f_j \,:\, f_j(r_1, \ldots, r_i) = \text{buy-single}, \, \forall 1 \leq j \leq i\}$$
$$M_i := \{f_j \,:\, f_j(r_1, \ldots, r_i) = \text{buy-monthly}, \, \forall 1 \leq j \leq i\}$$

  are used to count how many single-trip and monthly tickets the student has purchased, respectively.

A deterministic online algorithm for the BVG-ticket problem is defined by fixing the sequence of functions $f_1, f_2, \ldots$ to select answers from A according to the known requests. For instance, two trivial examples are the algorithms ATFIRST and NEVER given by

ATFIRST:

$$f_i = \begin{cases} \text{buy-monthly}, & \text{if } i = 1, \\ \text{use-monthly}, & \text{otherwise.} \end{cases}$$

NEVER:

$$f_i = \text{buy-single}, \, \forall i \in \mathbb{N}.$$

The algorithm ATFIRST buys the monthly ticket the first time a bad-weather request arises, while NEVER takes the opposite strategy and purchases only single-trip tickets, no matter how many times the student has to travel.

To judge the quality of an online algorithm, new, adequate reference parameters are required. A moment of thought reveals that traditional approaches like worst-case and average-case analysis are not of much help here. In fact, for the most online problem instances, it is always possible to choose the input sequence in such a way as to force the algorithm's performance get arbitrarily bad. On the other side, average-case results would require statistical information about the distribution of the input data, which is normally not available.

Sleator & Tarjan [1985] proposed their technique of *competitive analysis* for comparing the quality of various paging algorithms that manage cache memory in computers. This method has by now become a standard tool to evaluate online algorithms. It basically consists in comparing the output of an online algorithm with that of an *adversary*. An adversary is another algorithm for the same problem which has a certain degree of freedom to *choose* the input sequence in such a way as to let its own solution be better than the one of the online algorithm. The most common adversary is the *offline* adversary, which is allowed to choose *any* arbitrary input sequence, and which constructs its solution knowing this whole sequence in advance.

3.2.4 Definition (Competitiveness) *A deterministic online algorithm* ALG *is said to be* c-competitive *if there exist two constants* $c, b \in \mathbb{R}_+$ *such that, for any input sequence* σ*, the following holds:*

$$\mathit{ALG}(\sigma) \leq c \cdot \mathit{OPT}(\sigma) + b, \tag{3.1}$$

where $\mathit{OPT}(\sigma)$ *is the optimal solution value of the offline problem instance obtained when* σ *is given in advance.*

The constant c *is called the* competitive ratio *of* ALG*. If* $b = 0$*,* ALG *is said to be* strictly competitive*.*

Two remarks have to be made at this point. First, all optimization problems we consider here are minimization problems involving nonnegative cost functions. Therefore, in inequality (3.1) we assumed $0 \leq \mathsf{OPT}(\sigma) \leq \mathsf{ALG}(\sigma)$. On the other hand, some authors define competitiveness in a more ample sense and allow the competitive ratio to be a *function* $c : \mathbb{N} \to \mathbb{R}_+$ on the *length* $n = |\sigma|$ of the input sequence. We shall adopt the restricted definition from above for the rest of this thesis. Moreover, as long as we do not explicitly say the contrary, when we talk about competitiveness, we refer to *strict* competitiveness.

A moment of thought reveals that the algorithm ATFIRST for the BVG-ticket problem is 30-competitive. In the worst-case, the input sequence σ consists of only one bad-weather request, and the student buys a monthly ticket for it. Thus, $\mathsf{ATFIRST}(\sigma) = 30$, while $\mathsf{OPT}(\sigma) = 1$. In contrast, the competitive ratio for the algorithm NEVER can get arbitrarily large as the length of the input sequence increases. (Remark that it is possible to have several bad-weather requests per day).

Another competitive algorithm for this problem is REVIEW, which can be described as follows: After the i-th request has been revealed, the student takes the same decision it would have taken in the beginning, if he had known in advance that at least i bad-weather requests were to be expected. Hence, REVIEW buys the monthly ticket as soon as the 30th request is issued. The cost incurred by REVIEW depends on the length $n = |\sigma|$ of the input sequence:

$$\mathsf{REVIEW}(\sigma) = \begin{cases} n, & \text{if } n \leq 29, \\ 59, & \text{if } n \geq 30. \end{cases}$$

The reader can easily verify that the offline optimum value also depends only in the length of the input sequence, and that it is given by

$$\mathsf{OPT}(\sigma) = \begin{cases} n, & \text{if } n \leq 29, \\ 30, & \text{if } n \geq 30. \end{cases} \tag{3.2}$$

Hence, taking the largest ratio over all $n \in \mathbb{N}$ we obtain for the competitive ratio of REVIEW

$$\frac{\mathsf{REVIEW}(\sigma)}{\mathsf{OPT}(\sigma)} \geq \frac{59}{30} = 1 + \frac{29}{30},$$

which is much better than the ratio of ATFIRST. Now it is natural to ask if there exists some other online algorithm for which this ratio is further improved.

One subject of research in online optimization consists precisely in finding lower bounds on the best competitive ratios that can be achieved for certain classes of problems. This can be interpreted as the "dual" task to designing competitive algorithms: we want to determine the unavoidable additional cost that any online algorithm has to pay solely because it does not have knowledge about requests in the future.

In our case, note that any deterministic online algorithm for the BVG-ticket problem can be characterized by a value $\ell \in \mathbb{N} \cup \{+\infty\}$ which indicates how many requests have been issued when it decides to buy the monthly ticket. Thus, for $\ell = 1, 30, +\infty$ we obtain the algorithms ATFIRST, REVIEW, and NEVER, respectively. Since we have already proven that NEVER is not competitive, we assume in the following $\ell \neq +\infty$.

Denote by ALG_ℓ the deterministic algorithm corresponding to a fixed value $\ell \in \mathbb{N}$, and consider the input sequence σ_ℓ consisting of ℓ bad-weather requests. We have

$$\mathsf{ALG}_\ell(\sigma_\ell) = (\ell - 1) + 30,$$

as ALG_ℓ decides to buy the monthly ticket when the last request emerges. Combining this observation with (3.2),

$$\frac{\mathsf{ALG}_\ell(\sigma_\ell)}{\mathsf{OPT}(\sigma_\ell)} = \begin{cases} 1 + \dfrac{29}{\ell}, & \text{if } \ell \leq 29, \\ 1 + \dfrac{\ell - 1}{30}, & \text{otherwise.} \end{cases}$$

Clearly, this is a lower bound on the competitive ratio of ALG_ℓ. Moreover, for any deterministic algorithm ALG we can construct in this way an input sequence σ such that

$$\frac{\mathsf{ALG}(\sigma)}{\mathsf{OPT}(\sigma)} \geq \inf_{\ell \in \mathbb{N}} \frac{\mathsf{ALG}_\ell(\sigma_\ell)}{\mathsf{OPT}(\sigma_\ell)} = 1 + \frac{29}{30}.$$

It follows that REVIEW indeed achieves the best possible competitive ratio a deterministic online algorithm for this problem can have.

One major drawback of competitive analysis lies in the fact that it often constitutes a too pessimistic approach. In absence of any further restrictions, the ability to "see in the future" usually makes of the offline adversary such a powerful opponent that no online algorithm has a chance against it. To work around this limitation, a couple of ideas have been proposed. Basically, they all rely on weakening the position of the offline adversary in some way. For example, in the *resource augmentation* approach, the online algorithm is allowed to use more resources for serving the requests than its adversary, with the aim of compensating its lack of information. It is also possible to narrow the set of admitted input sequences in order to avoid those "degenerate cases" that produce bad competitive ratios, but are not likely to take place in practice. This was succesfully tried by Hauptmeier et al. [1999, 2000] in the context of the online dial-a-ride problem, for which the authors defined the notion of *reasonable load* (see Section 3.3.2). In the same manner, Blom et al. [2000] introduced a *fair adversary* for the online traveling salesman problem, and used it to improve the analysis for instances defined on the real line. (We shall come back to it later).

Other researchers (see e.g. Koutsoupias & Papadimitriou [1994]) have considered the possibility of requiring the adversary to belong to a given class of algorithms. They call this approach *comparative analysis*. In the context of job sequencing, Becchetti et al. [2003] improve the analysis of the Multi-Level Feedback Algorithm by applying *smoothening techniques* to flatten peaks in the competitive ratio caused by isolated bad input sequences. For a survey on extensions and alternative models to competitive analysis, the reader is referred to Fiat & Woeginger [1998].

3.2.2 Randomized Algorithms

From the point of view of competitive analysis, a key difficulty that online algorithms face regards their dependence on the input sequence. An algorithm that obtains good solutions for some sequences may perform very bad on other ones. Since competitive ratios are in this sense "worst-case" results, they commonly do not provide useful information when applied to practical situations. An alternative approach could be to look for "average-case" values, but, as anticipated before, this would require statistical knowledge on the distribution of the input data, which is normally either not present or insufficient. (Not to mention the additional technical issues that would have to be addressed).

A different idea consists in searching for average-case results in another sense, namely, by letting the online algorithm take its decisions according to a specified probability distribution. Thus, strategies that perform bad on a given input sequence may be averaged with better ones. Moreover, since the probability distribution used by the algorithm is known, it is possible to obtain *expected* values regarding the quality of the solution. An online algorithm based on this approach is called a *randomized algorithm*. In fact, one can think of a randomized algorithm as a collection of deterministic algorithms, combined together using a prescribed probability distribution.

3.2.5 Definition (Randomized Online Algorithm)
A randomized online algorithm RALG *for a request-answer game* $(R, \mathcal{A}, \mathcal{C})$ *is a probability distribution over the set* $\mathcal{D}$ *of all deterministic online algorithms for* $(R, \mathcal{A}, \mathcal{C})$.

Observe that the cost $\mathsf{RALG}(\sigma)$ of the output from a randomized algorithm RALG for a given input sequence σ is a random variable. Furthermore, since the decisions taken by RALG in response to σ are no longer predetermined, it is necessary to introduce some modifications in the basic framework of competitive analysis. An important step consists in defining which will be the position of the adversary, i.e. what kind of information from the online algorithm will be available to it. Three distinct models have been considered here:

- An adversary like the one used in the last section, which chooses σ *a priori* and processes it offline is called an *oblivious adversary*.
- In contrast, an *adaptive online adversary* has the freedom to choose each new request r_i after knowing the answer given by the algorithm to request r_{i-1}. The term *online* refers to the way in which the adversary itself has to process the input sequence.
- Finally, the *adaptive offline adversary* is allowed to construct the input sequence based on the answers provided by the online algorithm and to process it offline.

Taking these changes into account, competitiveness is defined in the first model as follows:

3.2.6 Definition (Competitiveness Against an Oblivious Adversary) *A randomized online algorithm* RALG *for a request-answer game* $(R, \mathcal{A}, \mathcal{C})$ *is said to be* c*-competitive against an oblivious adversary if, for any input sequence* σ*,*

$$\mathbb{E}\left[\mathsf{RALG}(\sigma)\right] \leq c \cdot \mathsf{OPT}(\sigma),$$

where $\mathbb{E}\left[\mathsf{RALG}(\sigma)\right]$ *denotes the expectation of the random variable* $\mathsf{RALG}(\sigma)$.

By considering deterministic algorithms as special randomized algorithms with Dirac probability measures, it is straightforward to see that the above definition is

in fact a generalization of Definition 3.2.4. In the case of the adaptive adversaries, it has to be taken into account that their solution values are also random variables:

3.2.7 Definition (Competitiveness Against an Adaptive Adversary) *A randomized online algorithm* RALG *for a request-answer game* $(R, \mathcal{A}, \mathcal{C})$ *is said to be* c*-competitive against an (online or offline) adaptive adversary if, for any input sequence* σ*, the following holds:*

$$\mathbb{E}\left[\mathit{RALG}(\sigma) - c\,\mathit{ADV}(\sigma)\right] \leq 0,$$

where $\mathit{RALG}(\sigma)$ *and* $\mathit{ADV}(\sigma)$ *are random variables that reflect the objective values of the algorithm and its adversary, respectively.*

Common sense suggests that the adaptive adversaries are stronger than the oblivious one. In fact, the following holds for any randomized algorithm RALG (see e.g. Borodin & El-Yaniv [1998]):

$$c_{\mathrm{OBL}} \leq c_{\mathrm{ADON}} \leq c_{\mathrm{ADOFF}},$$

where c_{OBL}, c_{ADON} and c_{ADOFF} are the competitive ratios of RALG against the oblivious, the adaptive online and the adaptive offline adversary, respectively. Moreover, Ben-David et al. [1994] showed the following two results:

3.2.8 Theorem (Ben-David et al. [1994])
Let $(R, \mathcal{A}, \mathcal{C})$ *be a request-answer game:*

1. *If there exists a* c*-competitive randomized online algorithm for* $(R, \mathcal{A}, \mathcal{C})$ *against the adaptive online adversary, then there exists also a* c^2*-competitive deterministic algorithm for* $(R, \mathcal{A}, \mathcal{C})$*.*
2. *If there exists a* c*-competitive randomized online algorithm for* $(R, \mathcal{A}, \mathcal{C})$ *against the adaptive offline adversary, then there exists also a* c*-competitive deterministic algorithm for* $(R, \mathcal{A}, \mathcal{C})$*.*

In particular, the second part of the theorem states that the adaptive offline adversary is so strong that the effects of randomization on the improvement of competitiveness disappear.

3.3 Online Transportation Problems

In this section, online versions of two common optimization problems in the area of transportation will be considered. Basically, they deal with the two main tasks that appear in vehicle routing: *matching*, i.e., assigning requests to vehicles, and *routing* itself, i.e. finding the best path for a vehicle to attend some given requests. Due to the complexity of these problems, the competitive ratios that can be proved theoretically are still far away from been satisfactory for practical purposes, except for some special cases. Nevertheless, the ideas derived from online optimization have been applied successfully to many real-world instances, some of which will be described here. (See Ascheuer et al. [1999] for more examples).

3.3.1 The k-Server Problem

Given a set X of *points*, a nonnegative function $d : X \times X \rightarrow \mathbb{R}_0^+$ is called a *metric* on X if and only if d satisfies the following properties:

(i) $d(x, y) = 0 \Leftrightarrow x = y, \quad \forall x, y \in X$

(ii) $d(x, y) = d(y, x) \quad \forall x, y \in X$

(iii) $d(x, y) \leq d(x, z) + d(z, y) \quad \forall x, y, z \in X$

The pair (X, d) is then called a *metric space*.

Let (X, d) be a metric space and $S = \{s_1, \ldots, s_k\}$ a set of so-called *servers* which are initially positioned on some points of X. An instance of the k-server problem is given by a sequence $\sigma = r_1, \ldots, r_n$ of n points of X, called *requests*. An online algorithm receives this sequence over time and has to *attend* the requests in the same order as they appear, where "to attend a request" simply means to move one server to it. The cost function to be minimized is the sum of the distances traveled by all the servers. This problem is a natural online version of the well-known transportation problem and has been extensively discussed in previous work (see for example Borodin & El-Yaniv [1998] and Hochbaum [1997]). This section will present a brief summary of the main ideas and results.

The k-server problem can be seen as a generalization of the *paging problem*, one of the first online problems that was studied. Computer systems often make use of two different kinds of memory devices for storing the data and program code needed during their operation. The first one, the *principal* memory, has a relatively large capacity, but accessing information from it takes more time than what would be desired. On the other side, the *cache* memory is faster, but its capacity is much more limited due to technical and/or production cost reasons. Given a sequence of (online) incoming memory access requests, the paging problem basically consists of deciding which information to maintain in the cache so that the number of times that principal memory has to be accessed is minimized. (The term "paging" comes from the fact that information is copied to and from memory in blocks of fixed size called pages). If k denotes the number of pages that can be stored in the cache simultaneously, a moment of thought reveals that this task can be modeled as a k-server problem on the metric space defined by the set N of all pages from the principal memory and by the discrete metric $d : N \rightarrow \{0, 1\}, d(x, y) = 1, \forall x \neq y$.

Several online algorithms for the paging problem have been considered. Some of them, including the very simple strategy of assigning pages in the cache according to a FIFO (first-in-first-out) scheme, turned out to be k-competitive. At the same time, it has been shown that k is a lower bound on the competitive ratio of any deterministic algorithm. Therefore, from the point of view of competitive analysis, "best possible" algorithms are known, at least for the deterministic case. (There is a randomized algorithm that beats the deterministic lower bound when compared against an oblivious adversary. Refer again to Borodin & El-Yaniv [1998] for a detailed discussion on this topic).

A natural question to ask is if the above results may be extended to the general k-server problem. If $|X| \geq k+1$, given any deterministic algorithm ALG, an input sequence σ can be constructed which enforces ALG to move a server each time a new request is released. Using such a sequence, it is possible to prove that k is still a valid lower bound on the competitive ratio for the general case. On the other hand, the question about the existence of a k-competitive deterministic online algorithm – the *k-server conjecture*, as Manasse et al. [1988] called it – remains one of the biggest open problems in online optimization.

The first step towards a (positive) resolution of this conjecture was taken by Fiat et al. [1990], who gave a $\mathcal{O}\left((k!)^3\right)$-competitive algorithm. Five years later, Koutsoupias & Papadimitriou [1995] proved that the previously proposed *work function algorithm* (WFA) is $(2k-1)$-competitive. As this chapter is written, this is still the best ratio that has been achieved in general. Additionally, the conjecture has been proved for some particular cases. For instance, Bartal & Koutsoupias [2000] show that the WFA is k-competitive when X is a line, when $|X| = k+2$, and for the symmetric weighted cache problem.

We explain in the sequel how the WFA works. With this purpose, a few preliminary definitions and some new notation have to be introduced here:

3.3.1 Definition (Configuration. Distance between Configurations) *Given an instance of the k-server problem over a metric space (X, d):*

1. *A multiset C containing k points of X is called a* configuration. *C will be used to represent the positions of the servers at a certain moment in time. The set of all possible configurations is denoted by $\mathcal{C}$.*

2. *For two configurations $C_1, C_2 \in \mathcal{C}$, the distance $D(C_1, C_2)$ between them is the value of a minimum-cost perfect matching in the bipartite graph $G = (C_1 \cup C_2, C_1 \times C_2)$ with respect to the cost function given by the metric d.*

3.3.2 Definition (Work Function) *Let C_0 be the initial configuration for an instance of the k-server problem (i.e., the initial position of the servers) and σ a fixed input sequence. The function*

$$w_\sigma : \mathcal{C} \to \mathbb{R}^+$$

defined pointwise by requiring $w_\sigma(C)$ to be the optimal solution value for the offline problem of processing all requests in σ (in the same order as they appear), starting from configuration C_0 and ending up in configuration C, is called the work function *associated to σ. In particular, $w_\emptyset(C)$ denotes the minimal cost for switching from configuration C_0 to configuration C, i.e.,*

$$w_\emptyset(C) = D(C_0, C)$$

For simplicity, the following notation will be used in the remainder of this section:

- Given a configuration C, a point $x \in C$ and a point $y \in X$, $C - x + y$ denotes the new configuration obtained by dropping x and adding y to C, i.e.
$$C - x + y := C \setminus \{x\} \cup \{y\}\,.$$

- σ will be used to denote the request sequence $r_1, \ldots, r_n$, and σ_i will refer to the subsequence that contains the first i requests, where $0 \leq i \leq n$. By σ_0 we just mean an empty input sequence.

The values of the work function w_σ may be determined via dynamic programming. It is easy to prove that, for all $i \in \{1, \ldots n\}$ and any $C \in \mathcal{C}$,

$$w_{\sigma_i}(C) = \min \left\{ w_{\sigma_{i-1}}(\hat{C}) + D(\hat{C}, C) \,:\, \hat{C} \in \mathcal{C},\, r_i \in \hat{C} \right\}.$$

In fact, consider an optimal solution and let C^* be the configuration of the system immediately after processing request r_i.The cost for getting to this configuration cannot be smaller than $w_{\sigma_{i-1}}(C^*)$. (Remind that σ_i has one request more than σ_{i-1}). It is also not larger, because if the system reaches configuration C^* after processing σ_{i-1} then request r_i can be processed at no cost, since $r_i \in C^*$. Therefore, $w_{\sigma_i}(C) = w_{\sigma_i}(C^*) + D(C^*, C) = w_{\sigma_{i-1}}(C^*) + D(C^*, C)$.

On the other side, observe that $w_\sigma(C_1) \leq w_\sigma(C_2) + D(C_1, C_2)$ holds for any two configurations $C_1, C_2 \in \mathcal{C}$. This fact can be used to show (see for instance Borodin & El-Yaniv [1998]) that there always exists a configuration of the form $\hat{C} := C - x + r_i$ that minimizes the right-hand side above, i.e.,

$$w_{\sigma_i}(C) = \min \left\{ w_{\sigma_{i-1}}(C - x + r_i) + d(x, r_i) \,:\, x \in C \right\}. \tag{3.3}$$

We are now in position to discuss the WFA. Algorithm 3.1 outlines the basic ideas of it. Each time a new request r_i arises, the algorithm moves *exactly one* server x_i, namely, the one who minimizes the right-hand side of (3.3). By doing so, WFA generates a sequence of configurations $C_0, \ldots, C_n$, where C_0 is the start configuration and C_n is the configuration after having processed the entire input sequence. Since the cost associated to such a move is $d(x_i, r_i)$, it follows from (3.3) that

$$\mathsf{WFA}(\sigma) = \sum_{i=1}^{n} d(x_i, r_i) = \sum_{i=1}^{n} \left(w_{\sigma_i}(C_{i-1}) - w_{\sigma_{i-1}}(C_i) \right).$$

Adding and subtracting $w_{\sigma_{i-1}}(C_{i-1})$ to each term of the last sum and taking into account that, since $r_i \in C_i$, the equality $w_{\sigma_{i-1}}(C_i) = w_{\sigma_i}(C_i)$ holds, one can work out that,

$$\mathsf{WFA}(\sigma) = \sum_{i=1}^{n} \left(w_{\sigma_i}(C_{i-1}) - w_{\sigma_{i-1}}(C_{i-1}) \right) - w_\sigma(C_n). \tag{3.4}$$

{Initialization}
$i \leftarrow 1$;
$C_0 \leftarrow$ initial configuration;
{Main loop}
loop
  wait until a new request r_i arrives;
  choose $x_i \in C_{i-1}$ that minimizes the expression

$$w_{\sigma_{i-1}}(C_{i-1} - x_i + r_i) + d(x_i, r_i);$$

  $C_i \leftarrow C_{i-1} - x_i + r_i$;
  $i \leftarrow i + 1$;
end loop

Algorithm 3.1: Work Function Algorithm WFA

Consider now the optimal offline solution. A direct consequence from Definition 3.3.2 is that

$$\mathsf{OPT}(\sigma) = \min \ \{w_\sigma(C) \ : \ C \in \mathcal{C}\} \leq w_\sigma(C_n) \tag{3.5}$$

The expression

$$\max \ \left\{w_{\sigma_i}(C) - w_{\sigma_{i-1}}(C) \ : \ C \in \mathcal{C}\right\}$$

is called the *extended cost* EC_i associated to the request r_i. It can be shown that (again, refer to Koutsoupias & Papadimitriou [1995]):

$$\sum_{i=1}^{n} \mathrm{EC}_i \leq 2k\,\mathsf{OPT}(\sigma) + b, \tag{3.6}$$

where b is a constant that does not depend on the input sequence. Replacing (3.5) and (3.6) in (3.4), it follows finally that

$$\mathsf{WFA}(\sigma) \leq (2k-1)\,\mathsf{OPT}(\sigma) + b,$$

which demonstrates the competitive ratio of WFA. The proof of (3.6) is intricate and will not be given here. The reader is rather referred to the original article for more information.

As a last remark, let us mention that Kalyanasundaram & Pruhs [1998] consider a slightly different version of the k-server problem, where the servers "disappear from the map" after attending a request. More precisely, instead of a set of servers, a set of *server sites* with fixed locations and capacities is given. Each time a request is issued, the online algorithm has to choose one from these sites to attend it, and the maximal number of requests a site can attend its given by its capacity. As in the k-server problem, it is known that k is a lower bound on the competitive ratio of any deterministic algorithm for this *online transportation problem*. To

derive positive results, the authors follow the "resource augmentation" paradigm: the online algorithm is compared against an adversary which is allowed to use only half of the capacity of each server site. Under this assumption, they show that the competitive ratio of a greedy algorithm which always assigns a request to the next available server site is still bounded from below by $\min\ \{k, \log C\}$, where C is the sum of the capacities of the k server sites. Next, they show how a simple modification of this greedy strategy gives an algorithm of constant competitive ratio.

3.3.2 The Online Dial-a-Ride Problem

As mentioned in Chapter 2, dial-a-ride problems constitute one important class of vehicle routing problems. The aim of this section is to consider some online versions of them, which will be stated using the time stamp model described in Section 3.2.1.

A metric space (X, d) is called "connected and smooth" if, for every pair of points $(x, y) \in X \times X$, there exists a function $p : [0, 1] \to X$ such that, for all $\tau \in [0, 1]$, the following properties hold:

(i) $p(\tau) \in X$,

(ii) $d(x, p(\tau)) = \tau d(x, y)$,

(iii) $d(y, p(\tau)) = (1 - \tau)\, d(x, y)$.

In particular, $p(0) = x$ and $p(1) = y$. The image set $p([0, 1])$ is said to be the *shortest path* between x and y. Examples of connected and smooth metric spaces are the Euclidean spaces $\mathbb{R}^n$ and the metric space induced by an edge-weighted undirected graph. (See Ausiello et al. [2001] for detailed explanations and more examples).

An instance of the online dial-a-ride problem (OLDARP) over a connected and smooth metric space (X, d) with a distinguished *origin* point $o \in X$ is given by a set σ of n requests having the following form:

$$\sigma := \left\{ (t_i, a_i, b_i) \,:\, \forall 1 \leq i \leq n,\ t_i \in \mathbb{R}_0^+;\ a_i, b_i \in X \right\}.$$

Each request $r_i \in \sigma$ represents a petition for transport of one unit of load from a *source point* a_i to a *target point* b_i. This petition becomes known to the system at a specific *release time* $t_i \geq 0$. All requests have to be attended using a single server of unit capacity and speed which is initially located on the origin o and which has to return there at the end. The task is to design a transportation schedule for this server which minimizes a certain cost function that will be explained later. Of course, no request may be attended before its release time. Furthermore, *preemption* is not allowed, which means that once the load corresponding to a request r_i has been picked-up, it may be unloaded only at the target point b_i. A feasible schedule for the server is called a *transportation plan*.

Notice that there is a small formal change with respect to the previous sections of this chapter: the input σ is now considered as a *set* of requests, not as a sequence. The set notation is more convenient for stating some definitions that will be needed later on. Besides, in the time stamp model, there is no need to require a specific ordering of the requests, since they are already labeled by their release times. To simplify the notation, however, we shall still assume that the requests in σ are indexed in such a way that these release times form a nondecreasing sequence $t_1 \leq \ldots \leq t_n$.

According to the time stamp model, an online algorithm ALG for the OLDARP has to work in *real-time*. This means that, at any time t, ALG must be able to give an immediate answer to the question of what to do at next with the server. Furthermore, this answer may depend only on the information contained in those requests whose release times are smaller than or equal to t. The offline adversary, on its side, knows the whole set σ in advance, but its solution must still take the release times of the requests into account.

Like in the offline case, several objective functions have been considered for minimization in the OLDARP:

- The *makespan* or *completion time* $C_{\max}$. This is the time at which the server has attended all requests in σ and returned to the origin.
- The *maximal flow time* $F_{\max}$. The flow time F_i of a request r_i is the difference $T_i - t_i$ between the time T_i when r_i has been attended and the time t_i when it was released. $F_{\max}$ is the maximal flow time obtained for a request in σ during the execution of transportation plan.
- The *average flow time*
$$F_{\text{avg}} := \frac{1}{n} \sum_{i=1}^{n} F_i.$$
- The *maximal wait time* $W_{\max}$. The wait time W_i of request r_i is defined by $W_i = F_i - d(a_i, b_i)$ and corresponds to the time elapsed between the moment when r_i is released and the moment when the server reaches its source a_i and starts to attend it.
- The *average wait time*
$$W_{\text{avg}} := \frac{1}{n} \sum_{i=1}^{n} W_i.$$

To take care of these different objectives, it is necessary to extend our original notation a bit. For the rest of this section, $\mathsf{ALG}(\sigma, f)$ will be used to denote the value of a solution obtained by an algorithm ALG in response to an input set σ, *with respect to the objective function* f, where $f \in \{C_{\max}, F_{\max}, F_{\text{avg}}, W_{\max}, W_{\text{avg}}\}$. Similarly, $\mathsf{OPT}(\sigma, f)$ will stand for the optimal value regarding the input set σ *and* the objective function f.

A first fact to notice is that minimizing the average flow (resp. wait) time is equivalent to minimizing the sum of all flow (resp. wait) times. Moreover, since flow and wait times are dependent on each other, it is reasonable to ask if there is some kind of relation between the corresponding optimization problems. In fact, if $N = \sum_{i=1}^{n} d(a_i, b_i)$ denotes the (unavoidable) time that any solution to the OLDARP has to spend transporting the requests (the so-called "loaded time"), it is straightforward to show that, for any online algorithm ALG and any input set σ,

$$\frac{\mathsf{ALG}(\sigma, W_{\text{avg}})}{\mathsf{OPT}(\sigma, W_{\text{avg}})} = \frac{n\mathsf{ALG}(\sigma, F_{\text{avg}}) - N}{n\mathsf{OPT}(\sigma, F_{\text{avg}}) - N} \geq \frac{\mathsf{ALG}(\sigma, F_{\text{avg}})}{\mathsf{OPT}(\sigma, F_{\text{avg}})}. \tag{3.7}$$

The last inequality implies that any c-competitive algorithm for the OLDARP with respect to the average wait time is automatically c-competitive when the average flow time is considered. The other way around, any lower bound on the competitive ratio regarding F_{avg} is also a valid lower bound for the case of W_{avg}. With a little more effort, an analogous relation between the maximal times can be proved.

Ascheuer et al. [2000] considered the problem of optimizing the completion time $C_{\max}$ and studied three online strategies: REPLAN, IGNORE, and the more elaborated SMARTSTART. The first one, REPLAN, consists basically of calculating a new transportation schedule each time a request arises. The server interrupts the execution of its former plan (maybe after having finished the current transport, to take care of the nonpreemption constraint), and then computes a new one to attend all remaining requests starting from its actual position and finishing in the origin. On the other hand, the main idea of IGNORE is to avoid interrupting a commenced plan. All requests that arise while the server is "busy" executing a transportation schedule are just *ignored* and collected. When the execution of the whole plan has finished, a new plan for all collected requests is calculated, and the cycle starts again. The authors showed that both REPLAN and IGNORE are $\frac{5}{2}$-competitive.

This competitive ratio is outperformed by the SMARTSTART strategy, whose main steps are illustrated in Algorithm 3.2. Basically, SMARTSTART resembles the IGNORE strategy, but is designed to avoid a worst-case situation that occurs when a request arises just after the server has started the execution of a long transportation plan.

The behavior of the algorithm at a certain moment in time depends on what is called the current *state* of the server. There are three possible states:

Idle No unattended requests are present, and the server is located in the origin and "waiting for orders". If one ore more new requests arise, they are added to a list $\mathcal{L}$ of *pending requests* and a control routine (called "work-or-sleep" oracle by the authors) is invoked. This routine computes an optimal transportation plan S for all requests in $\mathcal{L}$. (In fact, the term "optimal" needs further discussion. We shall return to this aspect later). Depending on the time $t(S)$ required for the execution of S, and on the current time t, the control routine may then set the server either into working (if $t + t(S) \leq \theta t$) or

{Initialization}
$\mathcal{L} \leftarrow \emptyset$;
goto Idle state;

{**Control routine**}
if $\mathcal{L} = \emptyset$ **then**
  goto Idle state;
end if
compute an optimal schedule S to serve all requests in $\mathcal{L}$;
$t(S) \leftarrow$ time needed to execute schedule S;
$t \leftarrow$ current time;
if $t + t(S) \leq \theta t$ **then**
  $\mathcal{L} \leftarrow \emptyset$;
  goto Working state;
else
  $t_{\text{wake}} \leftarrow \frac{t(S)}{\theta - 1}$;
  goto Sleeping state;
end if

{**Idle state**}
while no new requests arise **do**
  {keep waiting until new requests are issued ...}
end while
add new requests to $\mathcal{L}$;
goto Control routine;

{**Working state**}
start serving plan S;
while plan S is not completed **do**
  {if new requests are issued while working, add them to $\mathcal{L}$}
  if new requests arise **then**
    add new requests to $\mathcal{L}$;
  end if
end while
goto Control routine;

{**Sleeping state**}
while current time $< t_{\text{wake}}$ **do**
  {if new requests are issued while sleeping, add them to $\mathcal{L}$}
  if new requests arise **then**
    add new requests to $\mathcal{L}$;
  end if
end while
goto Control routine;

Algorithm 3.2: SMARTSTART for the OLDARP

into sleeping (if $t + t(S) > \theta t$) state. The factor θ involved in this decision is a constant calibration parameter larger than one.

Working The server is executing a transportation plan. All incoming requests are "ignored" and added to $\mathcal{L}$. When the server completes the plan, the control routine is invoked to decide what to do next.

Sleeping When the server enters this state at some time t, it sets a "wake-up clock" to time

$$t_{\text{wake}} := \frac{t(S)}{\theta - 1},$$

which is the earliest time at which the control routine would have decided to start working on plan S. Until t_{wake}, the server will just sit on the origin, ignoring all new incoming requests and adding them to $\mathcal{L}$. At that time, the control routine is invoked again.

By case distinction on the state of the server when the last request arises, the authors proved that, for any input sequence σ,

$$\frac{\textsf{SMARTSTART}(\sigma, C_{\max})}{\textsf{OPT}(\sigma, C_{\max})} \leq \max \left\{\theta, 1 + \frac{1}{\theta - 1}, \frac{\theta}{2} + 1\right\}.$$

In particular, choosing $\theta = 2$, a 2-competitive strategy is obtained. On the other side, Ausiello et al. [1994, 2001] had proved that 2 is a lower bound on the competitive ratio of any deterministic algorithm for this problem. This means that SMARTSTART achieves the best competitiveness, at least when considering instances over general metric spaces. (There are specific spaces like the real line, where the general construction used to derive the lower bound does not apply).

One important aspect that still has to be discussed concerns the offline subproblems solved by the control routine of SMARTSTART each time it is called. Notice that throughout this chapter, no consideration has been made about computational complexity issues when analyzing online algorithms. This was because in all cases the actions taken by an algorithm in response to a new incoming request were rather trivial. In the case of SMARTSTART, however, this is no longer true. As stated before, computing a transportation plan for the unattended requests is equivalent to solving an instance of a dial-a-ride problem, a task known to be $\mathcal{NP}$-hard. On the other hand, both for technical reasons related to the complexity of the analysis, as well as for practical requirements arising from the real-time environment in which the algorithm usually has to work, the transportation plan must be calculated "instantaneously" (i.e., within a time negligible when compared to $C_{\max}$). It makes therefore little sense to solve the subproblems to optimality. Instead, heuristics and approximation algorithms are commonly used.

Yet precisely this latter fact gives rise to a new question concerning competitiveness. Namely, the competitive ratio for SMARTSTART mentioned above was derived under the assumption that the optimal schedule was computed for all offline subproblems. Fortunately, it is possible to generalize the results to the case

when a ρ-approximation scheme is employed, i.e., when the transport plans calculated are at most ρ times longer than the optimal ones. For this situation, the authors proved the following:

3.3.3 Theorem (Ascheuer et al. [2000]) *For all real numbers $\theta \geq \rho$ with $\theta > 1$, the algorithm SMARTSTART based on a ρ-approximation scheme for constructing the partial transportation schedules has a competitive ratio equal to:*

$$\max \left\{ \theta, \rho \left(1 + \frac{1}{\theta - 1} \right), \frac{\theta}{2} + \rho \right\}.$$

The best choice for θ is $\frac{1}{2}(1+\sqrt{1+8\rho})$ and results in a competitive ratio of $\frac{1}{4}(4\rho+1+\sqrt{1+8\rho})$.

Frederickson et al. [1978] present a $\frac{9}{5}$-approximation algorithm for the offline subproblems that appear in the OLDARP, which are also known as *Stacker Crane problems*. Embedding this algorithm in SMARTSTART, a competitive ratio of $\frac{41+\sqrt{385}}{20}$ (≈ 3.03107) is achieved. Better results can be obtained if the underlying metric space has some special structure. Frederickson & Guan [1993], for instance, give a $\frac{5}{4}$-approximation scheme for the Stacker Crane problem on spaces arising from trees. Moreover, for metric spaces arising from a simple path, the polynomial time solvability of the problem has been proved.

A particular case of the OLDARP that has been subject of many studies is the *Online Traveling Salesman Problem* (OLTSP). It appears when $a_i = b_i$ holds for all requests $r_i \in \sigma$, and one can easily see that the related offline subproblems are in fact instances of the TSP. (Remark that under our assumptions, $C_{\max}$ is equal to the length of a tour visiting all requests). Therefore, from the last theorem, it follows that using the classical $\frac{3}{2}$-approximation scheme from Christofides [1976], SMARTSTART could reach a competitive ratio of $\frac{7+\sqrt{13}}{4}$ (≈ 2.6514) for this problem. Again, better results are known for special metric spaces.

Ausiello et al. [2001] proved a lower bound of $\frac{9+\sqrt{17}}{8}$ (≈ 1.64039) for the OLTSP on the real line and described an algorithm with a competitive ratio of $\frac{7}{4}$ (=1.75). Later, Lipmann [1999] found an algorithm whose competitiveness matches the lower bound. Finally, Blom et al. [2000] refined the former analysis by establishing two distinct categories for the online algorithms considered. They called an algorithm to be *zealous* if it does not let the server "wait" in an idle state and "collect" incoming requests. IGNORE and SMARTSTART are therefore examples of nonzealous algorithms. The authors showed that $\frac{7}{4}$ is a lower bound on the competitive ratio of any zealous algorithm for the OLTSP on the real line. As it turned out, the algorithm of Ausiello et al. [2001] is zealous and therefore best possible with regard to competitiveness, at least within its class. Moreover, the "bad" sequence used to derive the lower bound reveals a conceptual limitation of competitive analysis: it occurs in a situation of exaggerated inequity between the online algorithm and the offline adversary. Namely, the adversary moves the server far away from the previously released requests towards the point where the next request is going to take place. Motivated by this observation, the authors defined

an adversary to be *fair* if it is not allowed to move the server outside the convex hull of the previously released requests. Against a fair adversary, the lower bounds decrease to $\frac{8}{5}$ for zealous algorithms and $\frac{(5+\sqrt{57})}{8}$ for the general case. This opens a new gap for possible further improvements. However, it has not been possible to arrive at better positive results, except for the case when the metric space is restricted to $\mathbb{R}_0^+$.

The other way around, Krumke [2002] considers some generalizations of the OLDARP. For example, he shows that if the capacity of the server is allowed to be larger than one, the competitive ratio of REPLAN degrades to $\frac{7}{2}$, while those of IGNORE and SMARTSTART remain unchanged. Moreover, these latter ratios are still preserved if k distinct servers with different capacities $C_1, \ldots, C_k \in \mathbb{N}$ are admitted. Results are also described for the related *nonclosed makespan* problem, where the server is not required to return to the origin after having completed the transportation schedule. The competitive ratios of the three online strategies are in this case worse than the ones of their "closed" counterparts.

Notice that, since the OLDARP was defined on a metric space, all distances have to satisfy the symmetric property: for any $x, y \in X$, $d(x, y) = d(y, x)$. While working on a project to optimize the control of stacker cranes within automatic storage systems at a production plant of *Siemens Nixdorf Informationssysteme AG*, Ascheuer [1995] studied a similar problem where symmetry was no longer present. This new problem turned out to be by far more complicated and, up to the present, no competitive online algorithm is known for it. Nevertheless, an analysis based on simulation and a-posteriori comparison of the results produced by several heuristics made it possible to design a strategy that reduced the time needed for unloaded moves of the crane by approximately 30%.

Regarding the optimization of flow and wait times, the situation is somehow more complicated from the point of view of the competitive analysis, and results come rather from the negative side. Hauptmeier et al. [1999, 2000] give the following two examples that lead to lower bounds on the competitive ratio of any deterministic algorithm ALG for the problems related to F_{avg} and $F_{\max}$, respectively. For simplicity, both examples have been defined on the metric space $X = \mathbb{R}$, but they can be extended to any general metric space.

Consider at first the case of optimizing F_{avg}. The adversary starts by sending a request $r_1 = (0, \frac{1}{2}, o)$ and then waits until ALG decides to attend it at some time t. (Remember that, since ALG is deterministic, the value of t can be computed in advance). Just after that, at time $t + \varepsilon$, $n - 1$ new requests of the form $(t + \varepsilon, \frac{\varepsilon}{2}, o)$ are released. Since no preemption is allowed, ALG has to deliver the load of the first request before attending the other "small" ones. Thus, the following holds for

the sum $n\mathsf{ALG}(\sigma, F_{\text{avg}})$ of the flow times,

$$\begin{aligned} n\mathsf{ALG}(\sigma, F_{\text{avg}}) &\geq (t+1) + \sum_{i=1}^{n-1}(t+1+i\varepsilon-(t+\varepsilon)) \\ &= t+1+n-1+\varepsilon\sum_{i=1}^{n-1}(i-1) \\ &= t+1+n-1+\frac{\varepsilon}{2}(n^2-3n+2). \end{aligned} \tag{3.8}$$

The optimal solution, on the other hand, depends on the value of t. Suppose that $t < 1$. In this case, the offline adversary starts by moving the server to position $\frac{\varepsilon}{2}$ and waits there until the small requests arise. (There is enough time for this movement, since the requests are released at time $t+\varepsilon$). Then, it attends these requests and leaves r_1 for the end. Doing the corresponding calculations, it follows:

$$\begin{aligned} n\mathsf{OPT}(\sigma, F_{\text{avg}}) &= \sum_{i=1}^{n-1}\left(t+\frac{2i+1}{2}\varepsilon-t-\varepsilon\right)+t+\varepsilon+\varepsilon(n-1)+1-\frac{\varepsilon}{2} \\ &= \frac{\varepsilon}{2}\sum_{i=1}^{n-1}(2i-1)+t+1+\frac{\varepsilon}{2}+\varepsilon(n-1) \\ &= t+1+\frac{\varepsilon}{2}n^2. \end{aligned} \tag{3.9}$$

Combining (3.8) and (3.9), a bound on the competitive ratio is derived. In effect, for any $\delta > 0$ is it possible to find a sufficiently small value of ε, such that

$$\frac{\mathsf{ALG}(\sigma, F_{\text{avg}})}{\mathsf{OPT}(\sigma, F_{\text{avg}})} = \frac{t+1+n-1+\frac{\varepsilon}{2}(n^2-3n+2)}{t+1+\frac{\varepsilon}{2}n^2} \geq \frac{t+1+n-1}{t+1} - \delta.$$

Finally, since $0 \leq t < 1$,

$$\sup_{\sigma}\frac{\mathsf{ALG}(\sigma, F_{\text{avg}})}{\mathsf{OPT}(\sigma, F_{\text{avg}})} \geq \frac{t+1+n-1}{t+1} = 1+\frac{n-1}{t+1} \geq \frac{n-1}{2}.$$

On the other side, if $t \geq 1$, the offline adversary has enough time to process the first request and return to the origin before the small requests are released. In this case, it is easier to prove (in an analogous way) that $n+1$ is a valid lower bound on the competitive ratio.[1]

The last results show that the gap between the best online and the offline solutions will increase linearly with the cardinality of the input sequence. In particular, no deterministic algorithm with a constant competitive ratio exists. Randomization does not help much, either. For the online machine scheduling problem, Vestjens

[1] The analysis is greatly simplified if the small requests have the form $(t+\varepsilon, 0, o)$. However, the bound we have derived above remains valid even if we consider nonstrictly competitiveness, i.e., if we allow an additive constant in (3.1) on page 49.

[1997] proved a lower bound of $\Omega(\sqrt{n})$ on the competitive ratio of any randomized algorithm against the oblivious adversary. His construction can be translated to the setting of the F_{avg} − OLDARP.

Regarding the optimization of the maximal flow times, the situation is even worse. Consider the following example: the input set σ contains only one request of the form $(1, x, x + \frac{\varepsilon}{2})$, where $\varepsilon > 0$ and $x \in \{0, 1\}$ is fixed depending on the position y of the server at time $t = 1$. (Remark, again, that since ALG is deterministic, y is known a priori). If $y \leq \frac{1}{2}$, then $x = 1$, otherwise $x = 0$. It is easy to see that the solution found by ALG will have a flow time strictly larger than $\frac{1}{2}$. On the other side, the optimal solution consists in moving the server to position x during the first second of time, and achieves therefore a value of ε for the flow time. From this, it follows that:

$$\frac{\mathsf{ALG}(\sigma, F_{\max})}{\mathsf{OPT}(\sigma, F_{\max})} > \frac{1}{2\varepsilon}.$$

Since ε can be chosen arbitrarily small, there is no constant c that bounds the quotient above. Therefore, no deterministic algorithm can be competitive even for instances consisting only of one request! Optimizing over the maximal flow time seems therefore to be one kind of problem where "pure" competitive analysis reaches one of its main limitations: the offline adversary is so powerful that comparing any algorithm against it does not provide any information. This fact is usually denoted by saying that the *triviality barrier* has been reached.

In virtue of inequality (3.7) and its counterpart for the maximal times, these two lower bounds can be extended to the optimization of W_{avg} and $W_{\max}$, respectively. The following theorem summarizes the results:

3.3.4 Theorem (Hauptmeier et al. [1999, 2000]) *1. Any c-competitive deterministic online algorithm ALG for the* OLDARP *with objective functions F_{avg} or W_{avg} has a competitive ratio $c \geq \frac{n-1}{2}$, where n is the maximal length admitted for an input set.*

2. Any c-competitive randomized online algorithm RALG for the OLDARP *with objective functions F_{avg} or W_{avg} has a competitive ratio $c \in \Omega(\sqrt{n})$ against the oblivious adversary, where n is the maximal length admitted for an input set.*

3. There is no c-competitive deterministic online algorithm for the OLDARP *with objective functions $F_{\max}$ or $W_{\max}$.*

These strong lower bounds raise again the question mentioned in Section 3.2.1. Namely, if there is a reasonable way to limit the power of the offline adversary such that some positive results can be derived, at least for the problem instances that are relevant in practice. The authors proposed the idea of restricting the allowed input sets in order to exclude those weird cases where the algorithms present an extremely poor performance level. To describe their approach more precisely, a few definitions are needed:

3.3.5 Definition (Release-Span) *Given a request set σ, its release-span $\partial(\sigma)$ is the time elapsed between the moments at which the earliest and the latest request in σ were released, i.e.,*

$$\partial(\sigma) := t_n - t_1.$$

3.3.6 Definition (Offline Version of a Request Set) *Given a request set σ, its offline version σ^{offline} is the new request set obtained by setting the release times of all requests in σ to be equal to zero, i.e.,*

$$\sigma^{\text{offline}} := \{(0, a_i, b_i) \,:\, (t_i, a_i, b_i) \in \sigma\}\,.$$

In other words, σ^{offline} defines a relaxation of the associated offline dial-a-ride problem, which is obtained by dropping the time windows constraints. Using the last two concepts, the notion of load bound is at next introduced:

3.3.7 Definition (Load Bound) *A weakly monotone increasing real function is called a* load bound *on a request set σ if, for any $\partial \in \mathbb{R}$ and any $S \subseteq \sigma$ with $\partial(S) \leq \partial$, the following holds:*

$$\mathit{OPT}(S^{\text{offline}}, C_{\max}) \leq f(\partial).$$

Notice that a (tight) load bound provides a kind of stability measure for the system: it compares the frequency at which requests are released with the time it takes to process them. Common sense suggests that, for the system to be stable, requests should not arise "faster" than they can be attended. In fact, there is a similar stability assumption well known in queuing theory that inspired the authors to introduce this concept. A first idea could be to look at such input sets for which the identity function is a load bound. However, observe that, for any input set σ and any valid load bound f on it,

$$f(0) \geq \max\ \{\mathsf{OPT}(\{r_i\}, C_{\max}) \,:\, r_i \in \sigma\}\,.$$

Hence, requiring f to be the identity function is equivalent to allowing σ to contain only requests that can be served in no time. Such input sets are not of much interest. A second approach consists in admitting a larger class of load bounds:

3.3.8 Definition (Reasonable Load) *A load bound f is called (Δ, ρ)-reasonable for some $\Delta, \rho \in \mathbb{R}$ if, for all $\partial \geq \Delta$,*

$$\rho f(\partial) \leq \partial.$$

An input sequence σ is called (Δ, ρ)-reasonable if it has a (Δ, ρ)-reasonable load bound.

This criterion turned out to be effective enough to obtain some positive results. The authors proved that, for problem instances under reasonable load, the strategy IGNORE yields to solutions of bounded maximal and average flow times. More precisely,

3.3.9 Theorem (Hauptmeier et al. [1999, 2000]) *Let $\Delta > 0$ and $\rho \geq 1$. Consider an instance of the* OLDARP *given by a (Δ, ρ)-reasonable input set σ. The following holds for an implementation of the online strategy* IGNORE *that uses a ρ-approximation algorithm to solve the offline subproblems:*

(i) $\mathsf{IGNORE}(\sigma, F_{\max}) \leq 2\Delta$,

(ii) $\mathsf{IGNORE}(\sigma, F_{\text{avg}}) \leq 2\Delta$.

In fact, what the authors really proved was the upper bound for $\mathsf{IGNORE}(\sigma, F_{\max})$. The other bound follows from the trivial observation that the average cannot be greater than the maximum. The SMARTSTART strategy is also competitive under reasonable load:

3.3.10 Theorem (Krumke [2002]) *Let $\Delta > 0$ and $\rho \geq 1$. Consider an instance of the* OLDARP *given by a (Δ, ρ)-reasonable input set σ. For an implementation of* SMARTSTART *which uses a ρ-approximation algorithm to compute an optimal schedule in its control routine (see Algorithm 3.2), the following holds:*

$$\mathsf{SMARTSTART}(\sigma, F_{\max}) \leq \max\left\{\frac{\theta}{\theta - 1}\Delta,\, 2\Delta\right\}.$$

In particular, choosing $\theta = 2$ yields an upper bound of 2Δ for the maximum flow time of any request.

Again, from the last theorem it follows that SMARTSTART is also competitive for the minimization of average flow times. On the other hand, Hauptmeier et al. [2000] show a "disastrous example" for REPLAN, where the algorithm produces a schedule with unbounded maximal flow time for an instance of the OLDARP under reasonable load. Basically, what this example puts in evidence is the fact that continuous replanning without any other restrictions may cause individual requests to be postponed indefinitely. This situation was also observed by Grötschel et al. [1999] during simulation studies of various online strategies for instances of the OLDARP that arise within the elevator subsystem of a fully automated pallet transportation system in a large distribution center of the office supply provider Herlitz AG.

Finally, let us point out that the strategy of restricting the admissible input sequences has also been successfully employed by Krumke et al. [2002] in the context of the online traveling salesman problem with objective $F_{\max}$ on the real line: They call an adversary to be *non-abusive* if it is only allowed to move the server in a direction where unserved requests are located. Then they present an online algorithm with constant competitive ratio against any non-abusive adversary.

Chapter 4

A Tutorial on Set Partitioning

4.1 Introduction

As exposed in Chapter 2, a common solution approach for the VRPTW concerns the so-called Dantzig-Wolfe decomposition methods, which basically consist in splitting the problem into a master problem (MP) and a subproblem (SP). The latter addresses the issue of generating "good" individual feasible tours for the vehicles, and can be stated as a shortest path problem with resource constraints. On the other hand, MP deals with the task of putting all these tours together to assemble a valid routing schedule in such a way that a certain cost function is minimized, all customers are visited exactly once, and each vehicle of the fleet has exactly one tour to drive (which might be an "empty" tour that does not attend any customers and incurs no expenses).

The aim of this chapter is to give some highlights on the theoretical background behind MP. Observe that the problem can be stated in the following equivalent way: Let N be the set of all customers and K the set of all vehicles. We are given a family $\mathcal{E}$ of subsets of $N \cup K$, each of them containing exactly one element from K. To every such subset $E \in \mathcal{E}$ is associated a nonnegative cost $c(E)$. The task is to find a minimum cost subfamily $\mathcal{E}_1 \subseteq \mathcal{E}$ that constitutes a partition of $N \cup K$, where the cost $c(\mathcal{E}_1)$ of $\mathcal{E}_1$ is defined as the sum of the costs of the sets it contains, i.e.,

$$c(\mathcal{E}_1) := \sum_{E \in \mathcal{E}_1} c(E). \tag{4.1}$$

Problems of this kind are called *set partitioning problems (SPP)*, and constitute a fundamental and broadly studied topic in the field of combinatorial optimization. The general SPP can be formulated as follows: omitting the differentiation between customers and vehicles, consider a weighted *hypergraph* $\mathcal{H} = (X, \mathcal{E})$, consisting of a set X of *nodes*, a family $\mathcal{E}$ of subsets from X, known as *hyperedges*, and an integer-valued weight function w defined over $\mathcal{E}$. We are interested in finding a *perfect matching* in $\mathcal{H}$, i.e., a set of hyperedges $\mathcal{E}_1 \subseteq \mathcal{E}$ which forms a partition of X, and has the minimum weight in the sense of (4.1).

Two other well-known problems are related to SPP: the *set covering problem (SCP)* and the *set packing problem (SSP)*. The former is obtained by relaxing the condition that $\mathcal{E}_1$ must be a partition of X and instead requiring each element of the ground set to appear *at least* in one of the members from $\mathcal{E}_1$, i.e., requiring $\mathcal{E}_1$ to be an *edge cover of minimum weight*. Conversely, SSP asks for a *maximum weight matching* in $\mathcal{H}$: a set $\mathcal{E}_1$ of *disjoint* hyperedges having the largest possible sum of weights.[1] In general, set partitioning, set covering and set packing are known to be $\mathcal{NP}$-hard. (See for instance Garey & Johnson [1979]).

All three problems can be formulated as integer programs. In fact, suppose the elements of X are numbered from 1 through m, and let $\mathcal{E} = \{E_1, \ldots, E_n\}$. Moreover, let $A \in \{0,1\}^{m \times n}$ be a $0/1$-matrix whose columns are the incidence vectors of the sets in $\mathcal{E}$. Associating to each $E_i \in \mathcal{E}$ a binary variable x_i that indicates whether the set is chosen to be in $\mathcal{E}_1$ or not, we can state the problems as follows:

$$\begin{array}{lll}
\text{(SPP)} \ \min \ w^T x & \text{(SCP)} \ \min \ w^T x & \text{(SSP)} \ \max \ w^T x \\
\text{s.t.} & \text{s.t.} & \text{s.t.} \\
Ax = \mathbf{1}, & Ax \geq \mathbf{1}, & Ax \leq \mathbf{1}, \\
x \in \{0,1\}^n, & x \in \{0,1\}^n, & x \in \{0,1\}^n,
\end{array}$$

where $w^T := (w(E_1), \ldots, w(E_n))$, and $\mathbf{1} \in \mathbb{R}^n$ is a vector having all components equal to one. Throughout this chapter we assume that A has *no empty rows or columns*, since these imply either redundancy, unboundedness or infeasibility. As a consequence, in the packing and covering problems it suffices to consider only the cases where the weight vector w is positive.

A usual approach for studying these problems from the viewpoint of polyhedral combinatorics consists in looking at the polytopes defined by the convex hulls of their sets of feasible solutions. The *set partitioning polytope* $P_I^=(A)$, the *set covering polytope* $Q_I(A)$, and the *set packing polytope* $P_I(A)$ are given by:

$$\begin{aligned}
P_I^=(A) &:= \operatorname{conv}\{x \in \{0,1\}^n \,:\, Ax = \mathbf{1}\}, \\
Q_I(A) &:= \operatorname{conv}\{x \in \{0,1\}^n \,:\, Ax \geq \mathbf{1}\}, \\
P_I(A) &:= \operatorname{conv}\{x \in \{0,1\}^n \,:\, Ax \leq \mathbf{1}\}.
\end{aligned}$$

Associated to these polytopes are their *fractional relaxations*, the solution sets of the linear problems obtained from SPP, SCP and SSP by dropping the integrality

[1] Alternatively, one can consider the *dual* hypergraph $\mathcal{H}^* = (\mathcal{E}, \mathcal{E}^*)$ of $\mathcal{H}$, which has $\mathcal{E}$ as its set of nodes and whose hyperedges are the sets of the form $\{E \in \mathcal{E} : i \in E\}$ for all $i \in X$. In this case, SCP (resp. SSP) is the problem of finding a minimum weight vertex covering (resp. a maximum weight vertex packing) in $\mathcal{H}^*$.

constraints on the variables:

$$P^{=}(A) := \left\{x \in \mathbb{R}^n_+ \,:\, Ax = \mathbf{1}\right\},$$
$$Q(A) := \left\{x \in \mathbb{R}^n_+ \,:\, Ax \geq \mathbf{1}\right\},$$
$$P(A) := \left\{x \in \mathbb{R}^n_+ \,:\, Ax \leq \mathbf{1}\right\}.$$

The fractional covering polyhedron is unbounded, but it can be shown that all of its vertices lie in the unit cube $[0,1]^n$. Remark that SPP, SCP and SSP may be stated equivalently as linear optimization programs over $P_I^{=}(A)$, $Q_I(A)$ and $P_I(A)$, respectively, since the fundamental theorem in linear programming guarantees that whenever such a program has a (bounded) optimal solution, it must also have a *basic* optimal solution, i.e., one which is attained at a vertex of its feasibility polyhedron. Unfortunately, an explicit description of $P_I^{=}(A)$, $Q_I(A)$, and $P_I(A)$ in terms of linear inequalities is in general so large that it cannot be used directly to solve practical problem instances. However, the knowledge gained by examining the facetial structure of these polytopes can still be exploited for the design of solution algorithms (e.g., in the form of good cutting planes).

From the definition above, it follows that $P_I^{=}(A) = P_I(A) \cap Q_I(A)$, which implies that all linear inequalities valid for either the packing or the covering polytopes are also valid for the partitioning polytope. (In fact, observe that $P_I^{=}(A)$ is actually a face of each of the other polytopes). For this reason, a by now established approach for studying the structure of $P_I^{=}(A)$ is to consider the packing and covering polytopes separately.

In the next section, some known results regarding SSP and the structure of the packing polytope will be presented. It turns out that set packing can be reformulated as the problem of finding a maximum weight stable set in the so-called *conflict graph* of A (defined in that section). Certain structures in this graph give raise to valid – and sometimes facet defining – inequalities for $P_I(A)$. Section 4.3 deals with the SCP. As we shall see, set covering is equivalent to the very general *maximum independence system problem* described there. Not surprisingly, far less results concerning the structure of $Q_I(A)$ are available than for the packing case. A satisfying formulation of the problem in graph theoretic terms is also missing.

Set partitioning, set covering and set packing have been focus of research within the academic community for more than thirty years by now, and are still subject of ongoing work. Theoretical issues, as well as algorithmic aspects and practical applications have been addressed in numerous publications. Among them, we suggest the following survey articles: Garfinkel & Nemhauser [1972, Chapter 8], Balas & Padberg [1976], Padberg [1977, 1979], Grötschel et al. [1988, Chapter 9] (in connection with the stable set problem and perfect graphs), Borndörfer [1998, Chapter 1], and Schrijver [2003, Chapter 64, 77 & 82] (the last two chapters within the framework of hypergraph theory). Regarding algorithms and applications, some good references are Hoffman & Padberg [1993] (related to the scheduling of airline crews), Desrosiers et al. [1991], and Borndörfer [1998, Chapter 3 & 4] (in the context of vehicle routing for public transportation systems).

Before continuing, this is a good point to introduce a careful disclaimer: this chapter is *not* intended to be a comprehensive survey in set partitioning, but rather to present a few basic concepts that will be needed later in Chapter 6, and to draw the interest of the reader towards the main current research topics in this field. For a more detailed exposition of any of the subjects, refer to the survey works mentioned above. In fact, much of the material treated here has been taken from Borndörfer [1998].

4.2 Set Packing

Consider at first the set packing problem SSP. We have just seen that it can be formulated either as maximum weight matching or as a maximum weight vertex packing problem in a hypergraph. Edmonds [1962] suggested for the first time that set packing has also an interesting interpretation in graph theoretic terms. Namely, given the constraint matrix $A \in \{0,1\}^{m \times n}$, the *intersection or conflict graph* $\mathbb{G}(A) = (\mathbb{V}, \mathbb{E})$ of A is defined as follows:

- $\mathbb{V} = \{1, \ldots, n\}$,
- $\mathbb{E} = \{ij : i, j \in \mathbb{V},\ \text{supp}(A_{\cdot i}) \cap \text{supp}(A_{\cdot j}) \neq \emptyset\}$.

In other words, $\mathbb{G}(A)$ has one node for each column of A, and two nodes are joined by an edge if and only if there exists at least one row of A where the corresponding columns have both entries equal to one. Observe that the rows of A are incidence vectors of *cliques* (i.e., sets of mutually adjacent nodes) in $\mathbb{G}(A)$, an that every edge of $\mathbb{G}(A)$ has both of its end nodes contained at least in one of these cliques. Moreover, note that different $0/1$-matrices may lead to the same conflict graph, as there are various ways to cover the edges of a graph by cliques.

It is straightforward to notice that any set of columns from A related to a feasible solution of SSP (i.e., the columns corresponding to variables that have value one in the solution) reveals a *stable set* in $\mathbb{G}(A)$ – a set of nodes in the graph which are pairwise nonadjacent. Conversely, any stable set in $\mathbb{G}(A)$ can be used to obtain a feasible packing by setting the corresponding variables to 1, as no two of them appear in a row of A simultaneously. Thus, SSP can be restated as the problem of finding a maximum weight stable set in $\mathbb{G}(A)$.

Applying the same principle described in the last section, define the *stable set polytope* $\text{STAB}(G) \subset \mathbb{R}^V$ associated to an undirected graph $G(V, E)$ as the convex hull of all incidence vectors of stable sets from G. Due to the observations above, it immediately follows that $P_I(A) = \text{STAB}(\mathbb{G}(A))$ holds for any $0/1$-matrix A. Moreover, given an undirected graph G, we can always find a $0/1$-matrix A (for instance the edge-node incidence matrix) with $G = \mathbb{G}(A)$, and hence by studying the stable set polytope one can obtain polyhedral results for the set packing problem, and vice-versa. Therefore, as long as there is no risk of confusion, we shall from now on consider $P_I(A)$ or $\text{STAB}(G)$ as two different ways of denoting the same mathematical object. We list at next some its basic properties:

4.2.1 Lemma (Basic Facts About the Packing Polytope) *Let $A \in \{0,1\}^{m \times n}$ be a $0/1$-matrix (with no empty rows or columns), and let $P_I(A)$ be the packing polytope associated to A.*

(i) $P_I(A)$ is full dimensional, i.e., $\dim P_I(A) = n$.

(ii) $P_I(A)$ is down monotone*: For all $x_1, x_2 \in \mathbb{R}^n_+$, with $x_1 \leq x_2$, $x_2 \in P_I(A) \Rightarrow x_1 \in P_I(A)$.*

(iii) All nonnegativity constraints induce facets of $P_I(A)$.

(iv) All nontrivial facets of $P_I(A)$ have only nonnegative coefficients (when put in the form $a^T x \leq \alpha$).

(v) Let B be a submatrix obtained by selecting an (arbitrary) set of columns from A. Any valid inequality for $P_I(B)$ is also valid for $P_I(A)$.

Proof. For (i) just remark that the zero vector and all unit vectors $e_1, \ldots, e_n$ belong to $P_I(A)$. This at the same time proves (iii), as there are always at least n affinely independent vectors that satisfy a nonnegativity constraint with equality.

Down monotonicity is better explained using the stable set formulation. Remark that by dropping a node i from a stable set $S \subseteq V$ we obtain a new stable set $S \setminus i$. Now let $x \in P_I(A)$ be a convex combination of incidence vectors of some stable sets $S_1, \ldots, S_k$ in $\mathbb{G}(A)$. For any $i \in \{1, \ldots, n\}$, the vector x^i obtained from x by setting the i-th coordinate to zero is also contained in $P_I(A)$, as it is the convex combination of (the incidence vectors of) the stable sets $S_1 \setminus i, \ldots, S_k \setminus i$. But then, since $P_I(A)$ is convex, any nonnegative vector obtained from x by *decreasing* the value of its i-th coordinate is also contained in $P_I(A)$. Iterating this argument over all coordinates, the claim follows.

Claim (iv) follows from (iii). Suppose a nontrivial facet has some negative coefficient a_i, and let x be a point on this facet for which the i-th coordinate x_i is strictly positive (the reader can check that such a point must exist). Moreover, let x^* be obtained from x by changing x_i to zero. Due to down monotonicity, x^* belongs to $P_I(A)$, while at the same time it violates the facet inequality.

Finally, property (v) is also a consequence from down monotonicity. Notice that $P_I(B)$ is affine equivalent to the polytope $P := P_I(A) \cap \{x_j = 0 : j \in N\}$, where N is the set of indices of the columns from A that were *not* chosen to be in B. Now assume there is a point $x \in P_I(A)$ which violates an inequality valid for $P_I(B)$. Due to (iii), the point x^* obtained from x by setting all variables with indices in N to zero is also contained in $P_I(A)$. Moreover, x^* is contained in P and hence it reveals a point of $P_I(B)$ violating the inequality, a contradiction. □

The stable set problem is closely related to three other graph theoretic problems: the minimum clique-covering, the maximum weight clique, and the minimum vertex-coloring problems. These constitute a stand-alone research field that has attracted much attention in the past decades. (See, for instance, Schrijver [2003, Chapter 64] for an updated survey). One key issue here concerns the study

of those graphs for which the stable set polytope has only integral vertices (or, equivalently, of those matrices for which the set packing polytope coincides with its fractional relaxation). We enter then the realm of *perfect graphs* and *perfect matrices* . It turns out that perfect graphs have a lot of interesting properties and implications, both from the graph theoretical and from the polyhedral point of view. During the last months, news about perfect graphs made it to the top lines as Berge's 40-year-old *strong perfect graph conjecture* , one of the seminal questions in this field, was proved by Chudnovsky et al. [2002]. We shall look at this topic with some more detail in the next section.

A second direction of research addresses the case – by far more frequent in practice – when the packing polytope is not integral. Here, Lemma 4.2.1 (v) opens the way to the "polyhedral study through graph classification" approach initiated by Padberg [1973a]. According to this lemma, any *valid* inequality for the stable set polytope STAB(G') of a graph G' remains valid for the stable set polytope of every graph G that contains G' as a node induced subgraph. Thus, one can study the facetial structure of stable set polytopes related to simple classes of graphs, and then apply the gained knowledge for solving larger instances of the set packing problem. Moreover, Padberg investigated how *facet defining* inequalities from STAB(G') can be transferred (or *lifted*, to put it in his own words) to facets of STAB(G), and presented some classes of graphs that *produce* certain inequalities. Padberg's work marked the start of a race within the research community, with more and more articles presenting new facets of the set packing polytope and new classes of *facet defining* graphs. In Section 4.2.2 we will take a look at some classical examples.

4.2.1 Perfect Matrices and Perfect Graphs

Consider once again the IP formulation of set packing presented in Section 4.1. Due to linear programming duality, we have

$$\begin{array}{lcllcllcl}
\max & w^T x & \leq & \max & w^T x & = & \min & y^T \mathbf{1} & \leq & \min & y^T \mathbf{1} \\
\text{s.t.} & & & \text{s.t.} & & & \text{s.t.} & & & \text{s.t.} & \\
& Ax \leq \mathbf{1}, & & & Ax \leq \mathbf{1}, & & & y^T A \geq w^T, & & & y^T A \geq w^T, \\
& x \geq 0, & & & x \geq 0, & & & y \geq 0, & & & y \geq 0, \\
& x \text{ integral}, & & & & & & & & & y \text{ integral}.
\end{array} \tag{4.2}$$

Notice that the left-most integer program does not require the variables to be binary, but just positive integers. However, this is still a valid formulation for SSP, since the missing condition $x \leq \mathbf{1}$ is implied by the packing inequalities $Ax \leq 1$ together with the nonnegativity constraints. (Remark that A has no empty columns). Moreover, observe that the second problem consists in maximizing $w^T x$ over the fractional set packing polytope $P(A)$.

Now remark that, as pointed out in the last section, $P_I(A)$ is not altered if A is replaced by another $0/1$-matrix having the same conflict graph. Therefore, we may assume w.l.o.g. that A contains as rows all incidence vectors of inclusion-maximal cliques in $\mathbb{G}(A)$.

We have seen that the packing problem can be stated as the maximum weight stable set problem in $\mathbb{G}(A)$. It is therefore natural to ask, if its "dual" integer problem (the right-most IP above) also has a graph theoretical meaning. Indeed, we can formulate it as follows: Given the undirected graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, and the positive (integral) weight function $w : \mathbb{V} \to \mathbb{Z}$, find an assignment y of nonnegative integral values to all maximal cliques in $\mathbb{G}$, such that the weight w_i of any node $i \in \mathbb{V}$ is less or equal than the sum of the values corresponding to all cliques which contain i, and such that the sum $\sum y$ over all cliques of the graph is minimized.

The last problem is a weighted version of the *clique covering* problem, which asks for a minimum-size family of cliques that covers all nodes of a graph. In fact, setting $w = \mathbf{1}$ in (4.2), we obtain exactly the IP formulations for the maximum cardinality stable set problem, on the left-most side, and for the clique covering problem, on the right-most side. The optimum solution values for these problems are usually denoted by $\alpha(\mathbb{G})$ and $\bar{\chi}(\mathbb{G})$, respectively. (See Definition 4.2.4 below).

In general, (4.2) establishes that $\alpha(G) \leq \bar{\chi}(G)$ holds for any undirected graph G, which has an easy combinatorial explanation, as each node of a stable set must be covered by a different clique. In the late fifties, Berge started to study some classes of graphs for which this relation is satisfied with equality. In the following years, his work – and most important, his conjectures – led to the definition of *perfect graphs*, and gave birth to a new research thread inside graph theory. It goes beyond the purpose of this thesis to treat perfect graphs in detail here. Nor will we give an exact description of how Berge's famous conjectures historically developed. The reader interested in one of these topics is referred, for instance, to Ramírez Alfonsín & Reed [2001] and Berge [1996, 1997], respectively. We restrict ourselves to a brief exposition of the main ideas that are connected with the set packing polytope. Let us start with three basic definitions:

4.2.2 Definition (Node-Induced Subgraph) *Given an undirected graph $G(V, E)$, any graph $H(V(H), E(H))$ satisfying $V(H) \subseteq V$ and*

$$E(H) = \{ij \in E \,:\, i, j \in V(H)\}$$

is called a node-induced subgraph *of G. We shall denote this by writing $H \lhd G$.*

4.2.3 Definition (Clique Covering, Node Coloring) *Let $G(V, E)$ be an undirected graph. A family $\mathcal{Q}$ (resp. $\mathcal{S}$) of node-induced cliques (resp. stable sets) from G such that any node $i \in V$ is contained at least in one $Q \in \mathcal{Q}$ (resp. $S \in \mathcal{S}$) is called a* clique covering *(resp.* weighted node coloring[2]*) of G.*

[2]Usually, the term *node coloring* is used to denote a *partition* of V into stable sets, i.e., a coloring of the nodes in such a way that no edge joins two nodes of the same color.

4.2.4 Definition (Four Basic Quantities) *Let G be an undirected graph. We define the following four quantities related to G:*

(i) The independence number *or* stability number *of G:*

$$\alpha(G) := \max \{|S| : S \lhd G, S \text{ stable set}\}$$

(ii) The size of a maximum clique *in G:*

$$\omega(G) := \max \{|Q| : Q \lhd G, Q \text{ clique}\}$$

(iii) The clique covering number *of G:*

$$\bar{\chi}(G) := \min \{|\mathcal{Q}| : \mathcal{Q} \text{ clique covering of } G\}$$

(iv) The chromatic number *of G:*

$$\chi(G) := \min \{|\mathcal{S}| : \mathcal{S} \text{ node coloring of } G\}$$

Observe that $\chi(G)$ is the minimum number of colors required to paint the nodes of G in such a way that no edge joins two nodes of the same color. This explains the name *chromatic number*. Also note that, by definition, $\omega(G) = \alpha(\bar{G})$, and $\chi(G) = \bar{\chi}(\bar{G})$, where $\bar{G}$ denotes the complement of G. Hence, (4.2) applied on $\bar{G}$ implies that $\omega(G) \leq \chi(G)$, which again is no surprising result: it is easy to see that every node of a clique has to be painted with a different color. However, these four quantities are related in a stronger way, and this has deeper implications for the polytopes described above. To see how, let us first introduce the concept of perfect graphs, and formulate Berge's conjectures.

4.2.5 Definition (Perfect Graphs)
A graph G is called χ-perfect *if $\omega(G') = \chi(G')$ holds for every $G' \lhd G$. Similarly, it is called* $\bar{\chi}$-perfect *if $\alpha(G') = \bar{\chi}(G')$ holds for every $G' \lhd G$. Finally, G is said to be* perfect *if it is both χ-perfect and $\bar{\chi}$-perfect.*

As stated before, the work on perfect graphs was initiated by a seminal conjecture of Berge [1961], which later became known as the *Strong Perfect Graph Conjecture*, and was proved only recently by Chudnovsky et al. [2002]:

4.2.6 Theorem (Strong Perfect Graph Theorem) *An undirected graph G is perfect if and only if G does neither contain the odd hole C_{2k+1}, nor the odd antihole $\bar{C}_{2k+1}$ as a node induced subgraph, for all $k \geq 2$.*

Here, the term odd hole means an odd cycle without chords, and an odd antihole is the complement of an odd hole. (See page 85 for a formal definition). We shall say more about the proof of this theorem later. Of more importance for our discussion on the set packing polytope is the *weak perfect graph conjecture*, formulated again by Berge [1967] and proved some years later by Lovász [1971] and Fulkerson [1973] (recently, a new shorter proof has also been published by Gasparian [1996]):

4.2.7 Theorem (Weak Perfect Graph Theorem) *A graph G is perfect if and only it is χ-perfect if and only if it is $\bar{\chi}$-perfect.*

Due to the observations made above, an equivalent formulation for this theorem could read: "G is $\bar{\chi}$-perfect if and only if its complement $\bar{G}$ is $\bar{\chi}$-perfect" (or a similar statement concerning χ-perfection). Note that if a $0/1$-matrix A contains as rows the incidence vectors of *all* cliques in $\mathbb{G}(A)$, then requiring $\mathbb{G}(A)$ to be $\bar{\chi}$-perfect is by definition the same as requiring A to satisfy all relations in (4.2) with equality for any $0/1$-weight vector w. Such kind of results are called *combinatorial min-max theorems*. (The order of the terms "min" and "max" is important here, and will become clear in a moment). Now let B be a $0/1$-matrix whose rows are incidence vectors of all stable sets in $\mathbb{G}(A)$. Since $\mathbb{G}(B) = \overline{\mathbb{G}}(A)$, Theorem 4.2.7 implies that whenever a combinatorial min-max result of the form (4.2) holds for A and all $0/1$-weight vectors w, then automatically a second *companion* theorem of the same type is obtained for B.

Before the weak conjecture was enunciated, several celebrated theorems in combinatorics had been noticed to appear in pairs, for instance the both earliest and most famous König-Egerváry Theorem (see König [1986, Theorem XIV 13/14, page 249]) and König's Edge Coloring Theorem for bipartite graphs (see König [1986, Theorem XI 15, page 187]). In an attempt at proving the conjecture, Fulkerson [1971, 1972] developed his theory of blocking and anti-blocking polyhedra and provided a proper framework for explaining this duality. We present the basic notions at next.

4.2.8 Definition (Anti-Blocker of a Packing Polytope) *Consider a nonnegative (not necessarily $0/1$) matrix A without empty columns, and define the fractional packing polytope $P(A)$ as on page 71. The polyhedron*

$$P^*(A) := \left\{ y \in \mathbb{R}^n_+ \,:\, x^T y \leq \mathbf{1},\, \forall x \in P(A) \right\}$$

is called the anti-blocker *of $P(A)$.*[3]

Remark that if B is the nonnegative matrix whose rows are the vertices of $P(A)$ then

$$P^*(A) = P(B) \tag{4.3}$$

[3] At present, and in a broader sense, some authors also refer to $P^*(A)$ as the *polar* polyhedron of $P(A)$. (See, e.g., Ziegler [1998, page 59])

as trivially $P^*(A) \subseteq P(B)$ and any linear inequality defining $P^*(A)$ may be expressed as a convex combination from inequalities of $P(B)$. Moreover, it can be shown that if A has no empty columns then the same holds for B, and $P^*(B) = P(A)$. Hence, the anti-blocking relation is a duality relation on the class of packing polyhedra. The matrices A and B are called an *anti-blocking pair*.

Associated to anti-blocking polyhedra are the *min-max equality* and the *max-max inequality*:

4.2.9 Definition (Min-max Equality, max-max Inequality) *Let $A \in \mathbb{Q}_+^{r\times n}$, $B \in \mathbb{Q}_+^{s\times n}$ be two nonnegative matrices.*

(i) We say that the min-max equality *holds for the ordered pair (A, B) if*

$$\begin{array}{lcl} \min\ y^T\mathbf{1} & = & \max\limits_{1\le i\le s} B_{i\cdot}w \\ \text{s.t.} & & \\ y^TA \ge w^T, & & \\ \quad y \ge 0 & & \end{array} \tag{4.4}$$

holds for every positive (integral) vector $w \in \mathbb{Z}_+^n$.

(ii) Similarly, the max-max inequality *is said to hold for the (unordered) pair $\{A, B\}$ if*

$$\left(\max_{1\le i\le r} A_{i\cdot}l\right)\left(\max_{1\le i\le s} B_{i\cdot}w\right) \ge l^Tw$$

holds for every pair of nonnegative (integral) vectors $w, l \in \mathbb{Z}_+^n$.

Fulkerson also proved the following:

4.2.10 Theorem (Characterization of Anti-Blocking Polyhedra) *Let $A \in \mathbb{Q}_+^{r\times n}$ and $B \in \mathbb{Q}_+^{s\times n}$ be two nonnegative matrices, none of which has a zero column. The following four statements are equivalent:*

(i) $P(A)$ and $P(B)$ are a pair of anti-blocking polyhedra.

(ii) The min-max equality holds for (A, B).

(iii) The min-max equality holds for (B, A).

(iv) The max-max inequality holds for $\{A, B\}$ and $A_{i\cdot}B_{j\cdot}^T \le 1$ holds for all $1 \le i \le r$ and $1 \le j \le s$.

Of particular importance for the study of combinatorial min-max results are those pairs of anti-blocking matrices A, B which are both $0/1$-matrices. This implies that the polytopes $P(A)$ and $P(B)$ have only integral vertices. In fact, much more is true and this is maybe the main result of anti-blocking theory. The min-max equality is said to hold *strongly* for (A, B) if the linear program in (4.4) has an integer optimum solution for all integral nonnegative objectives $w \in \mathbb{Z}_+^n$. This is

exactly the case when (4.2) holds with equality for both matrices A and B, and all $w \in \mathbb{Z}^n_+$. Such kind of linear systems of inequalities are called *totally dual integral* (TDI). It is clear that a pair of anti-blocking $0/1$-matrices is a necessary condition for the min-max equality to hold strongly. Surprisingly, this is also sufficient, as showed in Fulkerson [1972]:

4.2.11 Theorem (Integrality of Anti-Blocking Polyhedra) *Let $A \in \{0,1\}^{r\times n}$ be a $0/1$-matrix without zero columns, and $B \in \mathbb{Q}^{s\times n}_+$ an anti-blocker of A. The following statements are equivalent:*

(i) All essential rows of B (i.e., all facets of $P(B)$) are $0/1$-vectors.

(ii) The polytope $P(A)$ is integral.

(iii) The polytope $P(B)$ is integral.

(iv) The min-max equality holds strongly for A, B.

(v) The min-max equality holds strongly for B, A.

(vi) The system (4.2) *is TDI.*

(vii) The system obtained by replacing in (4.2) *A by B is TDI.*

What is the relation between this last theorem and the weak perfect graph conjecture? Well, as pointed out earlier, observe that the weak conjecture closely resembles the equivalence "(vi) $\Leftrightarrow$ (vii)", except that in the former w is restricted to be a $0/1$-vector. In this sense, Fulkerson had proven a "weighted" version of the conjecture, which he called the *pluperfect* graph theorem.

The original conjecture remained open until Lovász [1971] provided the missing link in form of his *replication lemma*, which in our context says the following:

4.2.12 Theorem (Replication Lemma) *Let $A \in \mathbb{Q}^{r\times n}_+$ be a $0/1$-matrix without zero columns, the following two statements are equivalent:*

(i) The system (4.2) *holds with equality for all $w \in \{0,1\}^n$.*

(ii) The system (4.2) *is TDI.*

Figure 4.1 illustrates the key ideas we have presented in this section, and how they are put together to assemble a proof for the weak perfect graph theorem. Since integrality of the packing polyhedra is now known to be equivalent to perfection of the conflict graph, a $0/1$-matrix having the property $P_I(A) = P(A)$ is also called a *perfect matrix*, as it is exactly the clique-node incidence matrix of a perfect graph. Moreover, from the weak theorem it also follows that the stable set vs. node incidence matrices of perfect graphs are perfect matrices, too.

The proof of the strong perfect graph theorem by Chudnovsky et al. [2002] is very long, with the current manuscript (submitted for publication) comprising

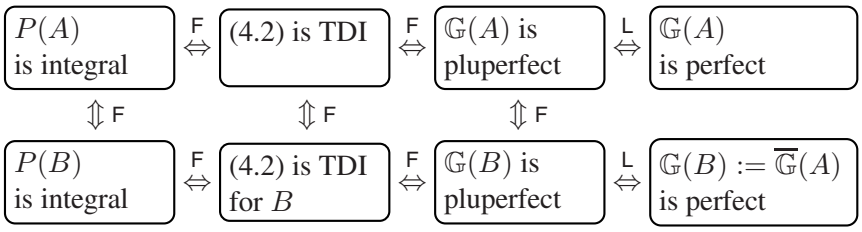


Figure 4.1: The weak perfect graph theorem. Summary of the results obtained by Fulkerson [1972] (F: see Theorem 4.2.11), and Lovász [1971] (L: see Theorem 4.2.12). The matrices A and B are assumed to contain as rows all incidence vectors of maximal cliques from their conflict graphs $\mathbb{G}(A)$ and $\mathbb{G}(B)$.

about 150 pages, and based upon some previous results obtained by other authors. In the sequel we give a very brief outline of the main steps. (A more extensive summary can be found in Chudnovsky et al. [2003]). A graph is called *Berge* if it does not contain odd holes or odd antiholes. Obviously, every perfect graph is Berge, and the "difficult direction" to prove in Theorem 4.2.6 is the converse one.

We need to introduce a short remark here. Observe that perfectness is by definition a *hereditary* property, i.e., if G is perfect then every $G' \lhd G$ is also perfect. A graph G is called *minimally imperfect* if G itself is not perfect, but all its node-induced subgraphs are; and it follows that every graph is either perfect or contains a minimally imperfect (node-induced) subgraph. Something similar can be said for matrices in terms of *contraction minors* (submatrices obtained by deletion of columns).

The strategy for proving the strong perfect graph theorem (originated by Conforti, Cornuéjols and Vušković) is to show that every Berge graph either belongs to some known classes of perfect graphs or has some structure that cannot appear in (minimally) imperfect graphs. Three of such structures were required for the proof: the 2-*join*, the *skew partition*, and the M-*join*. We will not define them here, but just mention that they are partitions of the set of nodes from G which satisfy certain properties. Cornuéjols & Cunningham [1985] had already proven that no minimally imperfect graph has a 2-join; and it follows from the work of Chvátal & Sbihi [1987] that no minimally imperfect graph has an M-join, either. Moreover, Chvátal [1985] conjectured that no minimally imperfect graph has a skew partition. This conjecture turned out to be true, but as a consequence of the perfect graph theorem. To prove the latter, Chudnovsky et al. [2002] showed a slightly different version of Chavátal's conjecture, restricted to so-called *even* (or *balanced*) skew partitions which comply with additional conditions. The authors demonstrated that no *minimum* imperfect Berge graph has an even skew partition, where a Berge graph G is said to be *minimum imperfect* if G is imperfect, and every Berge graph H with $|V(H)| < |V(G)|$ is perfect.

Finally, the authors proved the following main result which, when combined with the other ingredients described above, implies the strong perfect graph theo-

rem:

4.2.13 Theorem (Characterization of Berge Graphs) *Let G be a Berge graph. Then one of the following statements is true:*

(i) G belongs to one of six classes of perfect graphs (bipartite graphs, line graphs of bipartite graphs, double split graphs, and the classes made from their complements)

(ii) either G or $\bar{G}$ admits a 2-join

(iii) G admits a M-join

(iv) G admits an even skew partition

There are more interesting aspects regarding perfect graphs/matrices that we cannot present in this short "tour d'Horizon". One of them is for instance the *recognition problem*: Given a graph G (resp. a matrix $A \in \{0,1\}^{m \times n}$), decide whether G (resp. A) is perfect or not. It is easy to see that the recognition problem for matrices is in co-$\mathcal{NP}$, as it suffices to expose n linearly independent rows from A which produce a nonintegral vertex $x \in P(A)$. (Observe that testing membership for x requires $O(mn)$ time). The situation is much more complicated when the input is a graph, or equivalently, when A is only implicitly given by some oracle that decides whether a $0/1$-vector is a row of A or not. In these cases, we are interested in algorithms which are polynomial on n alone. It is of no use to construct the clique-node incidence matrix for G, since the number of rows of such a matrix will usually be an exponential function of n.

The approach followed has been to characterize perfect graphs/matrices in terms of *forbidden minors*. Padberg [1973b, 1976] studied several properties of minimally imperfect matrices, which can be used to conclude that the recognition problem for perfect graphs belongs to co-$\mathcal{NP}$, although this was at first stated (and proven in a different way) by Grötschel et al. [1984]. With the resolution of the strong conjecture, however, this result has become somehow "obsolete": now we know that the only forbidden minors for perfection (in graph theoretic terms) are odd holes and odd antiholes, and obviously one can test in polynomial time if a given graph is one of them. In fact, even more is true. Simultaneously, Chudnovsky & Seymour [2003] and Cornuéjols et al. [2003] devised two polynomial time algorithms for detecting odd holes in a graph which, together with the strong theorem, shows that the recognition problem can be solved in polynomial time. Both algorithms need a first phase that was developed jointly in Chudnovsky et al. [2003].

Another issue concerns the complexity of the maximum stable set problem. This problem is known to be $\mathcal{NP}$-hard on a general graph $G(V,E)$, in both its weighted and unweighted versions (see Garey & Johnson [1979]). Now consider the following relaxation of STAB(G), called the *clique-constrained stable set polytope*:

$$\text{QSTAB}(G) := \left\{x \in \mathbb{R}_+^V \,:\, x(Q) \leq 1,\ \forall Q \text{ clique from } G\right\}$$

where $x(Q) = \sum_{i \in Q} x_i$. Note that if A contains as rows all incidence vectors of maximal cliques in G, then $\text{QSTAB}(G) = P(A)$. Moreover, as in (4.3) we have $\text{QSTAB}(G) = \left\{y \in R_+^V : x^T y \leq 1, \forall x \in \text{STAB}(\bar{G})\right\} =: \text{STAB}^*(\bar{G})$, i.e., the clique-constrained stable set polytope of a graph is the anti-blocker of the stable set polytope of its complement. One can prove that there is a polynomial time transformation between the optimization problems over a packing polytope and its anti-blocker, and hence optimizing over $\text{QSTAB}(G)$ turns out to be $\mathcal{NP}$-hard in general, too.

For perfect graphs, Grötschel et al. [1984, 1988] obtained a surprising result. They showed that in this case the two problems can be solved in polynomial time (in fact, they are equivalent). Their proof is based on two ideas: the *orthonormal representations* previously introduced by Lovász [1979] (which later opened the way for the field of *semidefinite programming*), and the *polynomial time equivalence of separation and optimization*, proved by the same authors (see Grötschel et al. [1981, 1988]). Basically, a convex body $\text{TH}(G)$, in general not a polytope, is defined for any graph G as the solution set of infinitely many linear inequalities – the so-called *orthonormal* constraints. This body has the property

$$\text{STAB}(G) \subseteq \text{TH}(G) \subseteq \text{QSTAB}(G). \tag{4.5}$$

The key issue is that the *weak optimization problem* (see Grötschel et al. [1988] for a definition) can be solved over $\text{TH}(G)$ in polynomial time. Now observe that if G is perfect then $\text{STAB}(G) = \text{QSTAB}(G)$ and equality holds overall in (4.5), which implies the polynomial time solvability for the (weighted) stable set problem on G.

This is not the end of the story: being of theoretical importance, the last result does not deliver a "practically efficient" algorithm for solving the stable set problem on perfect graphs, because optimization over $\text{TH}(G)$ is carried out using the ellipsoid method. The search for a "pure" combinatorial algorithm remains in general open, although such algorithms have been proposed for some specific classes of perfect graphs. (See e.g. Grötschel et al. [1988, pp. 279–283] and the references there).

Finally, let us just mention that perfect graphs reveal miscellaneous connections to several other problems in combinatorics and information theory: they are related for instance to *graph entropy* (see Körner [1973], Cziszár et al. [1990] and Simonyi [2001]), to the *radio channel assignment problem* (see McDiarmid & Reed [2000] and McDiarmid [2001]), and also to the *Shannon capacity* of a graph (see Shannon [1956], Berge [1963] and Lovász [1979]). In fact, the latter was one of Berge's original motivations for looking at these graphs. Tucker [1972, 1973] also presents some applications in the context of optimization of municipal services. A lot of articles have been devoted to the study of structural properties and special classes of perfect graphs. Again, the interested reader is referred to Ramírez Alfonsín & Reed [2001]. We close this section here with a nice example of one of the various directions of research that have been pursued. Wagler

[2000] investigated the effects of deleting edges from (resp. adding edges to) perfect graphs. A graph G is said to be *critically* (resp. *anticritically*) perfect G itself is perfect, but any graph G' obtained from G by deleting (resp. adding) an edge is imperfect. Figure 4.2 shows the smallest graph which is both critically and anticritically perfect at the same time. Applying these concepts, Hougardy & Wagler [2003] proved that perfectness is an *elusive* graph property: to decide if a graph G is perfect, any algorithm has in the worst-case to look at all the entries of the adjacency matrix of G .

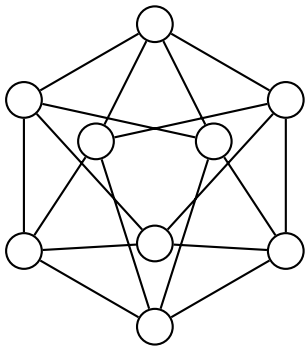


Figure 4.2: The smallest *critically* and *anticritically* perfect graph: either adding or deleting any edge destroys perfection. (Taken from Wagler [2000]).

4.2.2 Facet Defining Subgraphs

In general, set packing problem instances that appear in practical applications do not involve perfect matrices, or matrices having perfect conflict graphs. On the contrary, we usually have $\text{STAB}(\mathbb{G}(A)) \subset \text{QSTAB}(\mathbb{G}(A))$ and hence there must be facets of the stable set polytope which are not associated with cliques in $\mathbb{G}(A)$.

As pointed out earlier, Lemma 4.2.1 (v) presents a fundamental observation that can be exploited to study the structure of the stable set polytope: if $G' \lhd G$ then any valid inequality for $\text{STAB}(G')$ is also valid for $\text{STAB}(G)$. Motivated by this idea, Padberg [1973a] looked for the first time at certain classes of graphs that *define* new inequalities. We say that a graph G defines an inequality $a^T x \leq \alpha$ if the latter is essential for $\text{STAB}(G)$, i.e., if it forms a facet of the stable set polytope of G. By characterizing such facet defining graphs, we can hope to devise separation algorithms for the general set packing problem.[4]

Several classes of facet defining inequalities for the set packing polytope have

[4]Of special interest are facet defining graphs that give rise to an inequality for the first time. Padberg called them *facet producing*: A graph $G(V, E)$ is facet producing for a certain inequality $\gamma : a^T x \leq \alpha$ if G is facet defining for γ, but any subgraph of the form $G \setminus i$, for all $i \in V$, is not. Moreover, if the last statement is true for any arbitrary $G' \lhd G$, then G is said to be *strongly facet producing* for γ. We shall not treat facet producing graphs in this survey.

been described in this way. (See Borndörfer [1998] for an updated survey). We shall briefly consider a few classical examples on the next pages which, besides of illustrating basic proof techniques and properties, introduce some concepts that will be needed in Chapter 6.

In the same article, Padberg also addressed the more complicated issue of obtaining *facets* of the stable set polytope of a graph G from facets of the stable set polytope of some $G' \lhd G$. This led to development of *lifting techniques*. Other authors studied certain graph theoretic *composition procedures* that can be used to "glue together" graphs in such a way that their stable set polytopes are combined in a predictable manner, i.e., some (or in the best case all) of the facets of the stable set polytope corresponding to the composed graph can be determined from the facets of the stable sets polytopes of the pieces. A couple of elementary ideas regarding lifting techniques and composition procedures are presented at the end of this section.

Edge Inequalities

These are the most simple valid inequalities for the stable set polytope. Given a graph $G(V, E)$, edge inequalities have the form

$$x_i + x_j \leq 1$$

for all edges $ij \in E$. Edge inequalities are trivially valid for STAB(G) , and they constitute a special case of the clique inequalities treated below.

The fractional packing polytope $P(A)$ where A is the matrix of edge-node incidences from $\mathbb{G}(A)$ is called the *edge relaxation* of the packing polytope $P_I(A)$. For bipartite graphs, this edge relaxation indeed coincides with $P_I(A)$. Padberg [1973a] and Nemhauser & Trotter, Jr. [1973] (among others) studied several properties of the edge relaxation polytope of general graphs. It can be shown, for example, that its vertices are half integral, i.e., their coordinates are equal to 0, $\frac{1}{2}$ or 1. Moreover, given a fractional optimal solution x^* for the set packing problem over the edge relaxation polytope, there exists an optimal integer solution $\tilde{x}$ such that $\tilde{x_i} = x_i^*$ holds for every integer component of x^*. Unfortunately, in most practical applications x^* will have only a few coordinates different from $\frac{1}{2}$. Pulleyblank [1979] gave a probabilistic justification for this fact.

Clique Inequalities

Associated to every clique Q in a graph G is an also trivially valid *clique inequality* for STAB(G):

$$\sum_{i \in Q} x_i \leq 1. \tag{4.6}$$

If Q is maximal with respect to inclusion, it is easy to show (and was at first noticed by Fulkerson [1971] and Padberg [1973a]) that the clique inequality as-

sociated to Q is facet defining for STAB(G). Just consider the following stable sets:

$$S_i := \{i\}, \quad \forall i \in Q$$
$$S_j := \{i(j), j\}, \quad \forall j \in V \setminus Q$$

where $i(j)$ is any node of Q which is not adjacent to j. (At least one such a node exists for every $j \notin Q$, since Q is inclusion-maximal). The n incidence vectors of these stable sets are affinely independent and they all satisfy (4.6) with equality.

As pointed out on page 81, the polytope QSTAB(G) defined by all clique and nonnegativity inequalities is called the *clique-constrained stable set polytope* of G and is in general a relaxation of the stable set polytope. The optimization problem over QSTAB(G) turns out to be equivalent to the optimization problem over STAB$(\bar{G})$ and is therefore $\mathcal{NP}$-complete. Moreover, due to the polynomial time equivalence of separation and optimization, the separation problem for clique inequalities is also $\mathcal{NP}$-complete (see Grötschel et al. [1988, Section 9.2, page 283]).

On the other hand, clique inequalities belong to a larger class of polynomially separable *orthonormal representation constraints* (again, refer to Grötschel et al. [1988, Section 9.3, page 285], and Lovász [1979]). For perfect graphs, we have seen on page 82 that STAB(G) = QSTAB(G) holds, and hence all facets of the stable set polytope are given by clique constraints. Moreover, these constraints can then (at least theoretically) be separated in polynomial time.

Odd Cycle, Odd Hole and Odd Antihole Inequalities

An *odd cycle* is a graph $C(V(C), E(C))$ having an odd number of nodes, say $V(C) = \{0, \ldots, 2k\}$, and *at least* all edges of the form $\{ij : j = i + 1\}$, with the sum taken modulo $2k + 1$. Any other kind of edge is called a *chord* in the cycle. An *odd hole* is a chordless odd cycle, and an *odd antihole* is the complement of an odd hole. Figure 4.3 shows examples of these three classes of graphs.

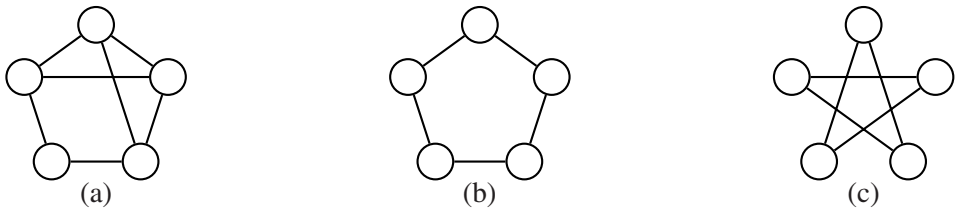


Figure 4.3: (a) An odd cycle, (b) an odd hole, and (c) an odd antihole.

If C is either an odd cycle or an odd hole, it is straightforward to verify that the *odd hole inequality*

$$\sum_{i \in V(C)} x_i \leq \left\lfloor \frac{V(C)}{2} \right\rfloor \tag{4.7}$$

is valid for STAB(C), and that it does not follow from the edge inequalities (just take a point having all coordinates equal to $\frac{1}{2}$). Moreover, Padberg [1973a] proved that (4.7) defines a facet of the stable set polytope if and only if C is an odd hole. If C appears as a node-induced subgraph from a larger graph G, then in general the odd cycle constraint for C will not define a facet of STAB(G), but it can be used to obtain one (or more) facets via the lifting techniques that will be presented later.

Grötschel et al. [1988, Lemma 9.1.11] have shown that odd cycle constraints can be separated in polynomial time. Chvátal [1975] suggested the term t-perfect (from *trou*, the French word for hole) to refer to those graphs whose stable set polytopes are completely described by the system of odd hole, edge, and nonnegativity inequalities. It follows from the equivalence between separation and optimization that the stable set problem can be solved in polynomial time for the class of t-perfect graphs. *Series parallel* graphs – graphs that do not contain a subdivision of K_4 as a node-induced subgraph – are one well-known example of t-perfect graphs. (See Chvátal [1975], Boulala & Uhry [1979], and Mahjoub [1988] for more details).

Associated to an *odd antihole* $\bar{C}$ is the following *inequality*:

$$\sum_{i \in V(\bar{C})} x_i \leq 2. \tag{4.8}$$

Again, it is easy to verify that (4.8) is valid for STAB$(\bar{C})$, and that it is nonredundant with respect to the edge constraints. Odd antihole inequalities were at first studied by Nemhauser & Trotter, Jr. [1973], who also showed that they are facet defining. No combinatorial separation algorithm has been proposed for this class of constraints, but (similarly as happens with clique inequalities) they have been proved to belong to a larger class of *matrix inequalities* that can be separated in polynomial time using the techniques quadratic and semidefinite programming (see Lovász & Schrijver [1991]).

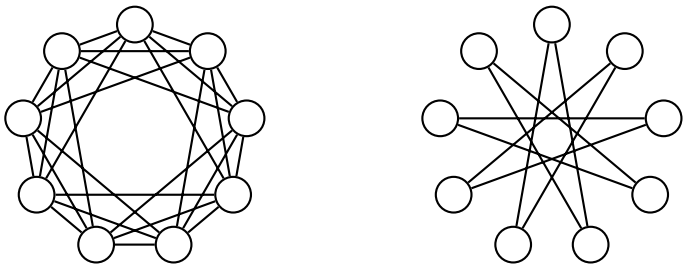
Antiweb and Web Inequalities

An *antiweb* $C(n,k)$ is a graph having the node set $V(C) = \{0,\ldots,n-1\}$, and the set of edges

$$E(C) := \{ij \,:\, i,j \in V(C),\ (j-i \mod n) < k\}\,.$$

A *web* $\bar{C}(n,k)$ is the complement of an antiweb. In the following, $2 \leq k \leq \left\lfloor \frac{n}{2} \right\rfloor$ will be assumed, as otherwise antiwebs and webs degenerate into either cliques or stable sets. Figure 4.4 shows two examples.

Observe that odd holes are exactly the antiwebs $C(2k+1,2)$, and odd antiholes are the corresponding webs $\bar{C}(2k+1,2)$. Trotter, Jr. [1975] proved that odd antiholes are also antiwebs of the form $C(2k+1,k)$ (and hence odd holes are also the webs $\bar{C}(2k+1,k)$). Moreover, the author showed that holes and antiholes are the only structures which are both webs and antiwebs at the same time.



(a) (b)

Figure 4.4: (a) The antiweb $C(9,4)$, and (b) the web $\bar{C}(9,4)$.

Given an antiweb $C(n,k)$, the *antiweb inequality*

$$\sum_{i \in V(C)} x_i \leq \left\lfloor \frac{n}{k} \right\rfloor \tag{4.9}$$

is valid for the stable set polytope STAB(C). Similarly, the *web inequality*

$$\sum_{i \in V(\bar{C})} x_i \leq k \tag{4.10}$$

is satisfied by all points of the stable set polytope STAB$(\bar{C})$ of a web $\bar{C}(n,k)$.

Trotter, Jr. [1975] proved that (4.9) and (4.10) are facet defining for their corresponding polytopes if and only if k and n are relatively prime (and k is restricted as above). Wagler [2002] showed that the stable set polytope of an antiweb is completely described by inequalities having only $0/1$-coefficients on the left-hand side. Such inequalities are called *rank inequalities*, and graphs with the mentioned property are said to be *rank-perfect*.

No polynomial time separation algorithm is known for antiweb inequalities, for web inequalities, or for some superclass containing them.

Lifting Techniques and Composition Procedures

On some occasions, when studying the stable set polytope of some graph G, we may be able to detect a facet defining node-induced subgraph $G' \lhd G$ (belonging, for instance, to one of the classes exposed in the previous paragraphs). Associated to G' we obtain a facet defining inequality for STAB(G'), which is at the same time a *valid* inequality for STAB(G). If G' is a maximal clique in G, we even know that the corresponding clique constraint is also facet defining for STAB(G). However, this is not true in general for the other structures.

Padberg [1973a] introduced a method called *sequential lifting* that allows to obtain facets of STAB(G) from odd hole inequalities related to node-induced odd holes $C \lhd G$. Later, Nemhauser & Trotter, Jr. [1973] applied this method to arbitrary facets of packing and covering polyhedra, and Zemel [1978] further extended

it to general $0/1$ polytopes. The following theorem formulates Padberg's technique in the context of the stable set problem.

4.2.14 Theorem (Sequential Lifting) *Let $G(V,E)$ be an undirected graph and $G' \lhd G$. Let W be a sequence containing all nodes from $V \setminus V(G')$ ordered in some (arbitrary) way, say $W = (w_1, \ldots, w_k)$. Assume $a^T x \leq \alpha$ is a facet defining inequality for STAB(G'), and determine the numbers $\beta_1, \ldots, \beta_k$ according to the recursion:*

$$G_i := G_{i-1} + w_i - N(w_i)$$
$$\beta_i := \alpha - \max_{x \in STAB(G_i)} \left(a^T x + \sum_{j=1}^{i-1} \beta_i x_i \right) \tag{4.11}$$

for all $i \in \{1, \ldots, k\}$. Here, $G_{i-1} + w_i - N(w_i)$ denotes the subgraph spanned by all the nodes from G_{i-1} which are nonadjacent to w_i, plus w_i itself; and $G_0 := G'$.

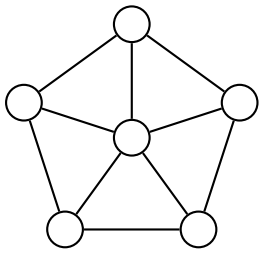
Then the inequality $a^T x + \sum_{i=1}^{k} \beta_i x_i \leq \alpha$ defines a facet of STAB(G).

W is called a *lifting sequence*, and the numbers $\beta_1, \ldots, \beta_k$ to be determined are the *lifting coefficients*. The word *sequential* denotes the fact that these coefficients are computed one after each other. Zemel [1978] also investigated the alternative of *simultaneous lifting*.

We will not discuss all issues related to sequential lifting here, but rather point out two basic ideas: first, observe that to calculate the lifting coefficients via (4.11) an instance of the stable set problem has to be solved at each step. From an algorithmical point of view, this seems at the first glance to make no sense, since in order to obtain *one* facet defining inequality for *one* instance of the SSP, we are required a solve a *collection* of several such problem instances. However, lifting techniques perform well in practice, and this is due to certain details. Remark, for example, that all lifting coefficients are integers between 0 and α, and hence (4.11) can be solved by enumeration in pseudo polynomial time (i.e, in a running time polynomial on both n and α). Besides, the successive instances of the lifting problem are closely related to each other, so chances are good that information can be reused when solving them. Finally, note that if we are not able to obtain the exact values of the lifting coefficients, we can still work with *lower bounds* for them. In this case, we will not get a facet for STAB(G) in the end, but we will still have "tightened up" our initial inequality.

The second aspect we want to highlight here is of conceptual nature. Since ordering is important when computing the lifting coefficients, in theory (and this is often also the case in practice) there are exponentially many facets of STAB(G) that can be lifted from a single facet defining subgraph, revealing that the facetial structure of a general set packing polytope might be extremely complex. Moreover, it is easy to use the lifting procedure to construct arbitrarily complicated facets involving intricate graph structures and large coefficients. On the other hand, one can also formulate this in a positive way: it might be possible to classify the wildness of facet defining inequalities appearing in set packing problems according to

a few simple core structures. In fact, many such inequalities resemble the ones presented in the previous sections, except for additional terms that can be seen as the results of sequential lifting. Figure 4.5 shows an example. The graph depicted there is known as a *wheel*, and associated to it is the *wheel inequality* (4.12), which is known to be facet defining. Notice that this constraint can be lifted from the odd hole inequality corresponding to the subgraph induced by the nodes on the "rim" of the wheel.



$$\sum_{i=0}^{2k} x_i + kx_{2k+1} \leq k \tag{4.12}$$

Figure 4.5: An odd wheel and the corresponding inequality. Odd wheel inequalities can be obtained from odd hole inequalities by sequential lifting.

Sequential lifting can be considered as one of several so-called *composition procedures*. In general, these procedures consist in taking two or more graphs and combining them into a new larger one by means of certain *graph theoretic operations* such that knowledge about the structure of the pieces can be transferred to the composed graph. Basically, three alternatives have been investigated:

Extensions. Given facet defining inequalities for the stable set polytope of a graph G, obtain facets for the polytope STAB(G'), where G' is constructed from G by addition of some simple structures. In the case of sequential lifting, these added structures are single nodes. Wolsey [1976] and Padberg [1977] explore other alternatives.

Substitutions. Given two graphs G_1 and G_2, a new graph G is defined by substituting certain subgraphs of G_1 by copies of G_2 and connecting them to the rest of the graph in some prescribed manner. Those substitutions are investigated for which facets (or complete linear descriptions) of STAB(G) may be obtained from facets of STAB(G_1) and STAB(G_2). A prominent example here is the substitution of nodes by cliques, which is also called the *replication* of a node and played an important role in the proof of the weak perfect graph conjecture by Lovász [1971] and Fulkerson [1972]. Later, Chvátal [1975] generalized this operation to the substitution of a fixed node by an arbitrary graph.

Joins. Given two graphs G_1 and G_2 which contain the isomorph node-induced subgraphs $G \lhd G_1$ and $G' \lhd G_2$, their join $G_1 \cup G_2$ is constructed

by identifying G and G'. Again, the idea is to consider joins where facets of STAB$(G_1 \cup G_2)$ may be obtained from facets of the stable set polytopes of G_1 and G_2. Two examples are the *clique identification* procedure of Chvátal [1975], and the *coedge identification* introduced by Barahona & Mahjoub [1994].

Several other composition procedures based on these three approaches have been described. For more information on them see Borndörfer [1998, page 35] and the references there.

4.3 Set Covering

We now focus our attention on the set covering problem, and discuss for it the same issues treated in the last section for set packing. Yet there are far fewer (and less elaborated) results to report here. This is mainly due to the fact that SCP is a very general optimization problem which subsumes many (hard) combinatorial problems – including set packing. Indeed, Laurent [1989] and Nobili & Sassano [1989] (among others) pointed out by means of the affine transformation $\tilde{x} = \mathbf{1} - x$ that SCP can be formulated as follows:

$$\begin{array}{lll} \text{(SCP)} \quad \min \; w^T x & \Leftrightarrow & w^T\mathbf{1} - \max \; w^T\tilde{x} \\ \text{s.t.} & & \text{s.t.} \\ Ax \geq \mathbf{1}, & & A_{i\cdot}\tilde{x} \leq |\text{supp}(A_{i\cdot})| - 1, \\ x \in \{0,1\}^n & & \forall 1 \leq i \leq m, \\ & & \tilde{x} \in \{0,1\}^n . \end{array} \tag{4.13}$$

Let us look at the combinatorial interpretation of the last integer program. Observe that any row $A_{i\cdot}$ whose support contains the support of some other row is redundant for SCP. Therefore, we may assume that this does never occur, i.e., that the rows of A are the incidence vectors of a *clutter* , to use the terminology introduced by Edmonds & Fulkerson [1970]. It is known that for a clutter $\mathcal{C}$ over some ground set X, the family

$$\mathcal{I} := \{I \subseteq X \,:\, |I \cap C| \leq |C| - 1, \forall C \in \mathcal{C}\}$$

is an *independence system*. Conversely, any independence system $\mathcal{I}$ can be defined in this form with respect to the clutter formed by its cycles. Hence, (4.13) just asks for a set of maximum weight belonging to a general independence system given through an independence oracle.[5] This is called the *maximum independence set problem* (ISP), and a moment of thought reveals that almost *any* combinatorial optimization problem can be put in this way, which in turn yields a set covering

[5] An *independence oracle* can be considered as a "subroutine" that gets as input any subset $I \subseteq X$, and answers the question whether I belongs to $\mathcal{I}$ or not. In our case, the oracle consists of the system of linear constraints in (4.13).

formulation for it. (The stable set problem, for instance, can be "translated" into the equivalent *minimum node covering problem*).

The set covering polytope has some basic properties analog to the ones listed on Lemma 4.2.1 for the packing polytope, although some further conditions have to be imposed on the $0/1$-matrix A:

4.3.1 Lemma (Basic Properties of the Covering Polytope) *Let $A \in \{0,1\}^{m \times n}$ be a $0/1$-matrix that has no empty columns and at least two nonzero entries on each row, and let $Q_I(A)$ be the covering polytope associated to A.*

(i) *$Q_I(A)$ is full dimensional, i.e., $\dim Q_I(A) = n$.*

(ii) *$Q_I(A)$ is* up monotone*: For all $x_1, x_2 \in \mathbb{R}^n_+$, with $x_1 \geq x_2$, $x_2 \in Q_I(A) \Rightarrow x_1 \in Q_I(A)$.*

(iii) *The upper bound constraints $x_i \leq 1$ induce facets of $Q_I(A)$.*

(iv) *A nonnegativity constraint $x_i \geq 0$ induces a facet of $Q_I(A)$ if and only if the minor A^i obtained from A by deleting column i has at least two nonzeros in each row.*

(v) *All facets of $Q_I(A)$ which are not induced by upper bound constraints have only nonnegative coefficients when put in the form $a^T x \geq \alpha$.*

(vi) *Let B be a submatrix obtained by deleting an (arbitrary) set of columns from A, and all the rows where at least one of these columns has a nonzero entry. Any valid inequality for $Q_I(B)$ is also valid for $Q_I(A)$.*

The matrix B in (vi) is called a *deletion* minor of A. This property establishes that valid inequalities for the polytopes associated to such deletion minors remain valid for $Q_I(A)$, and opens the way for studying facets of the covering polytope through the classification of matrix minors.

Unlike the case of set packing, however, SCP lacks a "good" interpretation in graph theoretic terms. Sassano [1989] and Cornuéjols & Sassano [1989] propose to consider A as the node-node adjacency matrix of a bipartite graph $G(V_1 \cup V_2, E)$, but the problem thus obtained - to find a minimum set of nodes in V_1 that covers all nodes in V_2, or vice-versa - does not possess any interesting structure that could help increase our understanding of $Q_I(A)$.

A $0/1$-matrix A that gives rises to an integral covering polyhedron (i.e., which has the property $Q(A) = Q_I(A) + \mathbb{R}^n_+$) is called *ideal*. Ideal matrices were introduced at first by Lehman [1979][6]. One can work out that if A is ideal, then every minor of A obtained by a sequence of contractions and deletions (a so-called *contraction-deletion minor*) is also ideal (see Seymour [1977]). Hence, it seems

[6] Although written in 1963, this paper was only published in 1979. Lehman called these matrices "W-L matrices", since he studied them in connection with his *width-length* inequality. (See the observations following Theorem 4.3.3). The term "ideal" was proposed by Cornuéjols & Novick [1989].

again reasonable to characterize ideality through forbidden minors, introducing the concept of *minimally nonideal matrices*, which only ideal contraction-deletion minors. However, in contrast to the minimally imperfect matrices discussed on page 80, these reveal a much more complex structure, as we shall see in the sequel.

For a *nonnegative* matrix $A \in \mathbb{R}_+^{m\times n}$, the (fractional) covering polyhedron $Q(A)$ is defined as on page 71, and its *blocker* $\mathrm{bl}(Q(A))$ is a second covering polyhedron given by

$$\mathrm{bl}(Q(A)) := \left\{ y \in \mathbb{R}_+^n \,:\, x^T y \geq \mathbf{1},\, \forall x \in Q(A) \right\}.$$

Fulkerson [1971] observed that $\mathrm{bl}(\mathrm{bl}(Q(A))) = Q(A)$ and hence the blocking relation pairs the members of the class of covering polyhedra. Similarly as for the anti-blocking case, the author characterized blocking pairs of matrices/polyhedra in terms of a *max-min equality* and a *min-min inequality*:

4.3.2 Definition (Max-min Equality, min-min Inequality) *Let $A \in \mathbb{Q}_+^{r\times n}$, $B \in \mathbb{Q}_+^{s\times n}$ be two nonnegative matrices.*

(i) We say that the max-min equality *holds for the ordered pair (A, B) if*

$$\begin{array}{lcl} \max\ y^T\mathbf{1} & = & \min\limits_{1\leq i\leq s} B_{i\cdot} w \\ \textit{s.t.} & & \\ y^T A \leq w^T, & & \\ \quad y \geq 0 & & \end{array}$$

holds for every positive (integral) vector $w \in \mathbb{Z}_+^n$.

(ii) Similarly, the min-min inequality *is said to hold for the (unordered) pair $\{A, B\}$ if*

$$\left(\min_{1\leq i\leq r} A_{i\cdot} l\right)\left(\min_{1\leq i\leq s} B_{i\cdot} w\right) \leq l^T w \tag{4.14}$$

holds for every pair of nonnegative (integral) vectors $w, l \in \mathbb{Z}_+^n$.

4.3.3 Theorem (Characterization of Blocking Polyhedra) *Let $A \in \mathbb{Q}_+^{r\times n}$ and $B \in \mathbb{Q}_+^{s\times n}$ be two nonnegative proper matrices. The following four statements are equivalent:*

(i) $Q(A)$ and $Q(B)$ are a pair of blocking polyhedra.

(ii) The max-min equality holds for (A, B).

(iii) The max-min equality holds for (B, A).

(iv) The min-min inequality holds for $\{A, B\}$ and $A_{i\cdot} B_{j\cdot}^T \geq 1$ holds for all $1 \leq i \leq r$ and $1 \leq j \leq s$.

Theorem 4.3.3 was motivated in part by two former results from combinatorial optimization, concerning the study of paths and cuts in a two-terminal network: (ii) and (iii) are analogues of the *max-flow min-cut equality* presented by Duffin [1962], while (iv) is the analogue of the *width-length inequality* proposed by Lehman [1979].

The main difference to anti-blocking theory appears when one considers blocking pairs of $0/1$-matrices and the corresponding strong version of the max-min equality, where y is required to be integral. In particular, two important observations have to be made here (see Fulkerson [1971] for a counter-example):

- The strong max-min equality for an ordered pair (A, B) does *not* imply the strong max-min equality for the pair (B, A).
- The integrality of a covering polyhedron $Q(A)$ does *not* imply that the corresponding system of constraints $Ax \geq 1$, $x \geq 0$ is TDI.

A weaker version of Theorem 4.2.11 was proven by Lehman [1979, 1981] for the covering case. Later, the same result was obtained by Padberg [1993] from a polyhedral point of view, and by Seymour [1990] within the framework of hypergraph theory. We state it below, slightly modified to fit in our current notation.

4.3.4 Theorem (Width-Length Property of Ideal Matrices)
Let A, B be two $0/1$-matrices. The following statements are equivalent:

(i) *A and B are a blocking pair.*

(ii) *The min-min inequality holds for $\{A, B\}$.*

(iii) *The min-min inequality holds for $\{A, B\}$ if the vectors w and l in (4.14) are allowed to have only coefficients from the set $\{0, 1, \infty\}$, except for at most one coefficient whose value is equal to the number of 1-coefficients minus one.*

For (iii) we adopt the convention $0 \cdot \infty = 0$. The extra coefficient not in $\{0, 1, \infty\}$ is required only to deal with a special case of minimally nonideal structures, the *degenerate projective planes*. And here we meet another difference to the perfect case. Lehman [1979] gave *three* infinite families of minimally nonideal matrices: the odd circulants $C(2k+1, 2)$, the integral part of their blockers, and the incidence matrices J_n of degenerate projective planes, which have the form

$$J_n = \begin{pmatrix} 0 & \mathbf{1}^T \\ \mathbf{1} & I_{n-1} \end{pmatrix},$$

where I_{n-1} is the identity matrix of order $n-1$. The two former classes can be seen as the counterparts of odd holes an odd anti-holes, but the third one has no equivalent. Moreover, researchers have compiled a large (but finite) list of "exception" matrices that are proven to be minimally nonideal, but do not belong to any of

the three Lehman's classes (see Cornuéjols & Novick [1989] and Lütolf & Margot [1962]). Besides, Cornuéjols & Novick [1989] give operations to construct further infinite families of minimally nonideal matrices from the ones known. This makes a characterization of ideal matrices in terms of forbidden minors extremely complicated. Still, they conjecture that such classification is possible for matrices with a sufficiently large number of columns if we restrict our attention to their *cores*[7].

As the reader may have already guessed, there is much more to say about ideal matrices than what can be covered here. We stop at this point and turn to the (in practice more common) nonideal case. In general, there will be facets of $Q_I(A)$ whose inequalities are not contained within the rows of A; they might even involve coefficients different from 0 and 1. Theorem 4.3.1 (vi) establishes that facets of the covering polytope still can be determined by local structures. Lacking of a satisfactory graph formulation, however, these structures must be expressed as submatrices, or, to be more precise, as *deletion minors* of A.

A matrix A' is said to *define* the inequality $\gamma : a^T x \geq \alpha$ if the latter is essential for $Q_I(A')$. Whenever we are able to identify such a *facet defining matrix* as a deletion minor of the constraint matrix A from some SCP instance, we get a new valid inequality that can be used to tighten up our relaxation of $Q_I(A)$. Moreover, *lifting procedures* to further sharpen γ (maybe until we obtain a facet of $Q_I(A)$) have also been studied, though the situation here is slightly more complicated as in the packing case, since both additional columns *and rows* have to be taken into account. (See Nemhauser & Trotter, Jr. [1973], Sassano [1989] and Nobili & Sassano [1989] for some examples.)

We present at next the two main classes of facet defining matrices that have been described (as far as we are aware).

Generalized Antiweb Inequalities

Given three natural numbers $n \geq t \geq q$, a *generalized antiweb* $\mathcal{AW}(n,t,q)$ is a $n\binom{t-1}{q-1} \times n$ $0/1$-matrix that has a row $\sum_{i \in Q} e_i^T$ for each q-element subset Q of any set of t consecutive column indices $\{j, \ldots, j+t-1\}$ (all additions taken modulo n). Figure 4.6 (a) shows an example. Associated to this matrix, Laurent [1989] and Sassano [1989] introduced the inequality:

$$\sum_{i=1}^{n} x_i \geq \left\lceil \frac{n(t-q+1)}{t} \right\rceil,$$

which is facet defining if and only if $n = t$ or t does not divide $n(q-1)$.

The generalized antiweb inequalities subsume three other classes of inequalities that had been previously proposed: the *generalized cliques*, obtained when

[7]The core of a matrix $A \in \{0,1\}^{m \times n}$ is a $n \times n$ regular submatrix A' with row and column sums equal to a constant α and having the property that any row of A which is not in A' is either the duplicate of some row in A', or has row sum strictly larger than α. Obviously, not all $0/1$-matrices possess a core, but whenever it exists, a core is unique up to permutations of rows.

$$\begin{pmatrix} 1 & 1 & . & . & . \\ 1 & . & 1 & . & . \\ . & 1 & 1 & . & . \\ . & 1 & . & 1 & . \\ . & . & 1 & 1 & . \\ . & . & 1 & . & 1 \\ . & . & . & 1 & 1 \\ 1 & . & . & 1 & . \\ 1 & . & . & . & 1 \\ . & 1 & . & . & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & . & . & . & 1 & . & . & . & . \\ 1 & . & . & . & . & 1 & . & . & . \\ . & 1 & . & . & . & 1 & . & . & . \\ . & 1 & . & . & . & . & 1 & . & . \\ . & . & 1 & . & . & . & 1 & . & . \\ . & . & 1 & . & . & . & . & 1 & . \\ . & . & . & 1 & . & . & . & 1 & . \\ . & . & . & 1 & . & . & . & . & 1 \\ . & . & . & . & 1 & . & . & . & 1 \end{pmatrix}$$

(a) (b)

Figure 4.6: Two examples of facet defining matrices:(a) The generalized antiweb $\mathcal{AW}(5,3,2)$, and (b) the generalized web $\mathcal{W}(9,4,2)$.

$n = t$ (see Nemhauser & Trotter, Jr. [1973], Sekiguchi [1985] and Euler et al. [1987]); the *generalized cycles*, that appear when $q = t$ and t does not divide n (see Sekiguchi [1985] and Euler et al. [1987]); and the *generalized antiholes*, for the case $n = qt + 1$ (see Euler et al. [1987]).

Remark that the antiwebs $\mathcal{AW}(2k+1,2,2)$ are exactly the circulant matrices $C(2k+1)$, the *odd holes*. These were further investigated by Cornuéjols & Sassano [1989]. Sassano [1989] and Nobili & Sassano [1989] also found two more classes of facet defining matrices that are constructed from the antiwebs $\mathcal{AW}(n,q,q)$ using a lifting and a composition operation, respectively.

Generalized Web Inequalities

An example of this second structure is depicted in Figure 4.6 (b). Generalized webs can be seen as the complements of generalized antiwebs: Given the natural numbers $n \geq t \geq q$, the *generalized web* $\mathcal{W}(n,t,q)$ is a $\binom{n}{q} - n\binom{t-1}{q-1} \times n$ 0/1-matrix that has a row $\sum_{i \in Q} e_i^T$ for each q-element subset Q not contained in any of the sets $\{j, \ldots, j+t-1\}$ of t consecutive column indices (all additions taken modulo n). In relation to it, Sassano [1989] defined the *generalized web inequality*:

$$\sum_{i=1}^{n} x_i \geq n - t.$$

Nobili & Sassano [1989] showed that this inequality is facet defining if and only if t does not divide n.

We close this section by mentioning a word about the *separation* problem for set covering inequalities. As one would expect, apart from the case of *odd hole inequalities*, only very few polynomial time algorithms are known, and all

of them are of heuristic nature: Nobili & Sassano [1992] introduced the concept of k-projections and used it to suggest a separation heuristic for *rank inequalities* (inequalities having only $0/1$-coefficients on the left side). Balas [1980] and Balas & Ho [1980] explored the use of *conditional cuts*, which are only valid under the assumption that a solution better than the best currently known one exists. Caprara & Fischetti [1996] and Schulz [1996] studied certain classes of $\left\{0, \frac{1}{2}\right\}$ Chvátal-Gomory cuts. Finally, Borndörfer [1998, page 80] applied his *set packing relaxation* to derive a separation heuristic for the new class of *aggregated cycle inequalities*.

Chapter 5

The ADAC-Problem. Competitive Analysis Results.

5.1 Introduction

Acknowledgement. The results from this chapter are joint work with Diana Poensgen[1] and Tjark Vredeveld[1].

In this chapter, some competitive analysis results for a simplified version of the *ADAC-Problem* introduced in Chapter 1 will be presented. As we shall see in the next section, this problem (even in the simplified version treated here) includes several standard online optimization problems as particular cases, like for example the online dial-a-ride problem. For some of these problems, families of input sequences are known which show that there exists no online algorithm with constant competitive ratio. Section 5.3 extends this result to the case of the ADAC-Problem and considers similar bad sequences and lower bounds. Following a common approach in online optimization, we iteratively introduce "reasonable" additional constraints into the model in order to get rid of these sequences. There is no general rule to decide if a constraint is reasonable or not. On the contrary, different criteria may be applied, leading to different results. As a guideline, we tried to keep the model as close as possible to the "real world" situation and allowed only constraints that we found justified in the context of our current application at ADAC. After adding a few restrictions, it is possible to come up with an online strategy which has a constant competitive ratio determined by some problem parameters. This is the subject of Section 5.4.

Let us start with a formal definition of the version of the problem we are considering here. Given are k homogeneous service vehicles of unit speed initially located at a depot $o \in X$, where X is a *connected and smooth* metric space (see Section 3.3.2). The input for any online algorithm consists of a set $\sigma = \{r_1, \ldots, r_n\}$

[1] Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustr. 7, 14195 Berlin, Germany, http://www.zib.de

of service requests[2], each of them containing the following information:

- a *position* $a(r_i) \in X$ where the request r_i takes place,
- a *release time* $\tau(r_i) \geq 0$ when r_i enters the system, i.e., when it becomes known to the online algorithm,
- a *deadline* $\theta(r_i)$, which is the maximum time at which r_i can be served without incurring lateness costs,
- a *duration* $\delta(r_i)$ for the service of r_i, and,
- a *service cost* $s(r_i)$. (Service costs are kept in the model only because are needed to get rid of some pathological input sequences that will be presented in the next sections. Evidently, there is no real room for optimization in them.)

The task is to design a *minimum cost* feasible *online* service schedule to *answer* all requests employing the k servers. There are three issues here that need further explanation. At first, to answer a service request r means to move a server to the request's position $a(r)$ and then hold it there for a time larger than or equal to the request's service duration $\delta(r)$. We assume that all servers are identical, i.e., that any request can be attended by any server. The second issue refers to the exact signification of "feasible online service schedule" in the sentence above. Following the time stamp model presented in section 3.2.1, we require from such a schedule that all decisions taken up to any time t must rely exclusively on information provided by requests having release times smaller than or equal to t. Moreover, any feasible schedule is required to start and end at a configuration where all servers are located at the depot.

The third key point is related to the definition of the cost of a schedule. In fact, the structure of the cost function is complicated, as it needs to take into account several heterogeneous components:

- *Travel costs*, which are proportional to the total distance traveled by the servers during the execution of a dispatch.
- The actual *service costs* of the requests.
- *Lateness costs*, which are nonmonetary costs that aim at ensuring a certain level of service quality from the viewpoint of the users of the system (i.e., the ones who issue the requests). If a request r is served at some time $t(r)$ after its deadline $\theta(r)$, then a cost proportional to the late time $(t(r) - \theta(r))$ has to be paid. (Here, $t(r)$ is defined as the time when a server *arrives* at r.)

[2]We shall follow a usual practice in the literature and treat σ both as set and as a *sequence*, abusing a little of the terminology.

- *Overtime costs*, used in the model to deal with another problem constraint arising from management decisions and labor agreements: the *shift-duration* of the servers. We introduce a global parameter T defined as the time by which all servers should have finished duty and returned to the depot. If a server i misses this deadline and finishes its schedule at some time t_i^* later, then it incurs a cost proportional to its overtime $t_i^* - T$.

To be able to put these diverse quantities together, we scale them by introducing constant coefficients as further problem parameters. In order to simplify notation, we do this scaling in such a way that the coefficient corresponding to the travel costs equals to one. Moreover, since service costs are constants, we also assume they have already been properly scaled and omit their coefficient. The remaining two coefficients will be denoted by c_{late} for the lateness costs, and c_{ot} for the overtime costs. The total cost of a dispatch is then given by the weighted sum of the travel and overtime costs of all servers, plus the service and lateness costs associated to all requests:

$$z = \sum_{i=1}^{k} D_i + \sum_{j=1}^{n} s(r_j) + \sum_{j=1}^{n} c_{\text{late}} \left(t(r_j) - \theta(r_j)\right)^+ + \sum_{i=1}^{k} c_{\text{ot}} \left(t_i^* - T\right)^+, \quad (5.1)$$

where D_i denotes the total distance traveled by server i, and x^+ stands for $\max\{x, 0\}$.

An alternative approach that we looked at (in fact, the first one we tried) consists in dropping the overtime component from (5.1) and replacing it by a "hard" constraint in the model that requires all servers to have completed their schedules by time T. However, as we shall see in Section 5.3.1, this modification involves certain technical difficulties: for any online algorithm, input sequences can be constructed such that the set of feasible online solutions is empty, while there exist some offline feasible schedules.

We are now ready to begin the analysis of the ADAC-Problem by establishing its connections to other online optimization problems.

5.2 Relation to Other Problems

The ADAC-Problem, as defined in the previous section, includes two well known online optimization problems as special cases: the k-server problem of Section 3.3.1, and a modified version of the online dial-a-ride problem from Section 3.3.2.

The most evident relation is the first one. Suppose service costs are ignored and $c_{\text{late}} = c_{\text{ot}} = 0$. The task is then to attend service requests arriving over time with k servers, minimizing the total distance traveled by them. The only difference to the "original" k-server problem is that the online algorithm is in our case not required to attend requests in the same order as they are released. At least for deterministic algorithms, however, this is not an important issue, since it is always possible to

enforce this property by choosing the release times for the requests appropriately. We tried to generalize known results (both lower bounds and algorithms) from the k-server problem to the ADAC-Problem, but we didn't obtain any interesting results.

On the other hand, if lateness or overtime costs dominate the other components of the cost function (5.1), the ADAC-Problem instances may be considered as slightly modified versions of online dial-a-ride problems with multiple servers, by considering service requests as transportation requests among an additional "service-time dimension". To explain this further, consider the following transformation: Given an instance of the ADAC-Problem over a connected and smooth metric space (X, d), the *service-time extension* (Y, d') of X is a new metric space defined by:

- $Y := \{(x, t) \,:\, x \in X,\, t \geq 0\}$
- $d'[(x_1, t_1), (x_2, t_2)] := d(x_1, x_2) + |t_1 - t_2|$

One can think of Y as being constructed by expanding X over time. The distance between two points in Y is the sum of the distance between their "projections" on X plus their separation on the service-time axis. It is straightforward to show that Y is indeed a metric space, and that it is connected and smooth.

An instance of the OLDARP is defined on Y from the input sequence σ of the ADAC-Problem in X by introducing, for every $r \in \sigma$, a transportation request $\hat{r}$ in Y with the following properties:

- the pick-up position of $\hat{r}$ is $(a(r), 0)$,
- the delivery position of $\hat{r}$ is $(a(r), \frac{1}{2}\delta(r))$,
- the release time of $\hat{r}$ is identical to the release time $\tau(r)$ of r.

Figure 5.1 illustrates the idea of this transformation. The objective of introducing an additional time axis is to account for the fact that a server is "immobilized" at a certain position $a(r)$ while attending a request r in the ADAC-Problem. This situation is modeled in the OLDARP by requiring the server to *transport* r along the service-time axis for a time equal to one half of the request's service duration $\delta(r)$. Since the start points of all transportation requests lie at the zero-time slice of Y, (i.e., have coordinates $(a, 0)$, with $a \in X$) the server has to return back to that time slice before picking-up the next request $\tilde{r}$. The way in which d' has been defined ensures that no matter which route the server chooses for doing so, the distance traveled cannot be smaller than $\delta(r) + d(r, r')$. Thus, the effect is the same as if the server had just waited at $a(r)$ for a time larger than or equal to $\delta(r)$.

Now consider what happens in the original ADAC-Problem if the cost coefficients are chosen in such a way that travel, service and overtime costs are negligible

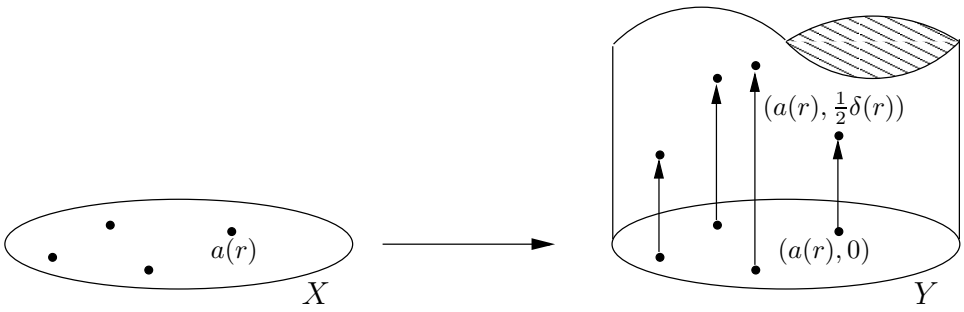


Figure 5.1: From the ADAC-Problem to the OLDARP

when compared to lateness costs. In this case, the cost function (5.1) may be approximated by

$$c_{\text{late}} \sum_{j=1}^{n} (t(r_j) - \theta(r_j))^+ .$$

Hence, the task is to minimize the sum of late times $(t(r) - \theta(r))^+$ for all requests $r \in \sigma$. In the setting of the OLDARP described above, this is equivalent to minimizing the total *surplus* of wait times. For each request r, this surplus is defined to be the quantity by which the wait time of r exceeds a maximal allowed value of $\theta(r) - \tau(r)$. To maintain the analogy with Section 3.3.2, let us rather consider the *average* surplus of wait times, and denote this new objective function with W^+_{avg}. Obviously, if $\theta(r) = \tau(r)$ holds for all requests, then $W^+_{\text{avg}} = W_{\text{avg}}$, and the problem turns into a "classical" OLDARP with average wait time as the objective function.

Unfortunately, the same does not hold in general, as shown by the example in figure 5.2 for a problem instance with one server. Throughout this section, black dots will be used to represent requests, little crosses for servers, and a white little circle for the depot. At time 0, a request r_1 with deadline 2 and duration $\frac{1}{2}$ is released at a point located two units away from the depot, as indicated in the figure. One time unit later, a second request r_2 with deadline 3 and duration 1 is issued exactly at the middle position between the depot and r_1. If we have to minimize the simple sum of wait times, then the optimal offline solution is to attend r_2 before r_1, and has a value of 3 (compared to $\frac{9}{2}$ for the other alternative). However, this is not the correct solution when minimizing the late times: its value of 1 exceeds the value of $\frac{1}{2}$ obtained if r_1 is served at first.

Nevertheless, there is a strong relation between optimization of late and wait times. In fact, if ALG is a c-competitive online algorithm (in the sense of definition 3.2.4) for the problem of minimizing wait times, then, for any input sequence $\sigma =$

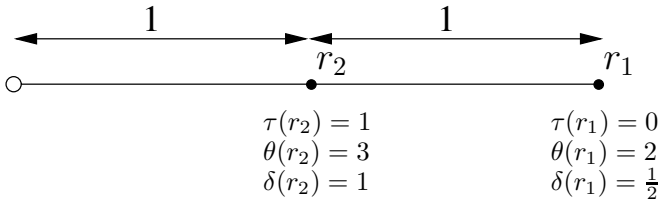


Figure 5.2: Wait times vs. late times. The black dots represent two requests r_1 and r_2 released at the positions shown with respect to the depot (the small white circle). The optimal offline solutions for the problems of minimizing wait and late times do not coincide.

$r_1, \ldots, r_n$, the following holds:

$$\begin{aligned} \mathsf{ALG}(\sigma, W^+_{\text{avg}}) \leq \mathsf{ALG}(\sigma, W_{\text{avg}}) &\leq c\,\mathsf{OPT}(\sigma, W_{\text{avg}}) + b \\ &\leq c\,\mathsf{OPT}(\sigma, W^+_{\text{avg}}) + \frac{c}{n}B(\sigma) + b, \end{aligned} \tag{5.2}$$

where $b, c \in \mathbb{R}$ and $B(\sigma) := \sum_{j=1}^{n}(\theta(r_j) - \tau(r_j))$. Furthermore, we keep the notation from Chapter 3 and denote by $\mathsf{ALG}(\sigma, f)$ the solution value obtained by an algorithm ALG when solving an instance of the OLDARP (or of the ADAC-Problem) with an input sequence σ and with regard to a cost function f. (For the ADAC-Problem, if we omit f, we just refer by default to the total cost function).

The first inequality comes from the fact that late times cannot be longer (by definition) than wait times. The second inequality just states that ALG is c-competitive for the minimization of W_{avg}. The last inequality is justified by observing that the optimal offline solution for the problem of minimizing W^+_{avg} is at the same time a feasible (offline) solution for the problem of minimizing W_{avg}, with value less or equal than $\big(\mathsf{OPT}(\sigma, W^+_{\text{avg}}) + \frac{1}{n}B(\sigma)\big)$.

Inequality (5.2) does not mean that ALG is c-competitive for the minimization of late times, since the additive "constant" $B(\sigma)$ that links wait and late times depends on the input sequence. However, this relation poses the question if, and under which conditions, it is possible to translate known competitive results from the average-wait-time OLDARP to the ADAC-Problem. As we shall see in the next sections, the answer is yes. At the same time, we know that minimizing average wait times in the general setting of the OLDARP is hopeless from the point of view of competitive analysis. Therefore, some restrictions have to be taken in order to achieve positive results for the ADAC-Problem.

A similar situation occurs if (5.1) is dominated by overtime costs. In this case, the ADAC-Problem can be transformed into an OLDARP where the objective is to minimize the total *excess of makespan* C^+_{sum}. Here, the excess of makespan $(t^*_i - T)^+$ for a server i is defined as the amount by which the completion time of its schedule surpasses the global return time T. As with late times, there is a connection between this problem of minimizing C^+_{sum}, and the "standard" OLDARP that aims at minimizing the global makespan $C_{\max}$ (the time at which the last

server has completed its schedule and returned to the depot). Moreover, competitive results for the latter do in this case imply results for the former.

Suppose ALG is a c-competitive online algorithm for the makespan OLDARP, and let $\sigma = r_1, \ldots, r_n$ be an input sequence. Then, with the same arguments used for (5.2),

$$\begin{aligned}\mathsf{ALG}(\sigma, C^+_{\mathrm{sum}}) \leq k\,\mathsf{ALG}(\sigma, C_{\max}) &\leq k\,(c\,\mathsf{OPT}(\sigma, C_{\max}) + b)\\ &\leq kc\,\mathsf{OPT}(\sigma, C^+_{\mathrm{sum}}) + kcT + kb,\end{aligned}$$

where the constant k appears due to the fact that C^+_{sum} is the sum of overtimes for *all* k servers, while $C_{\max}$ is the return time of only the *last* server. Again, to prove the third inequality, note that an optimal offline solution for the minimization of C^+_{sum} yields a feasible solution for the minimization of $C_{\max}$, with value not larger than $\mathsf{OPT}(\sigma, C^+_{\mathrm{sum}}) + T$. Since the additive constant $kcT + kb$ does in this case not depend on σ, it follows that ALG is a kc-competitive algorithm for the problem of minimizing overtimes (although not necessarily strictly competitive).

Finally, let us point out that the OLDARP with objective function W^+_{avg} (resp. C^+_{sum}) is indeed more general than the ADAC-Problem restricted to the minimization of late (resp. overtime) costs: the latter one requires all transportation requests to have starting positions only at points of Y with time coordinate equal to zero.

5.3 Lower Bounds and Problem Restriction

Considering the general character of the ADAC-Problem, the odds of finding a competitive algorithm for it are not good. Therefore, in this section we take the opposite approach and derive lower bounds on the competitive ratios of deterministic algorithms. Alternately, we restrict the set of allowed input sequences, with the aim of arriving at some scenario where we can obtain some positive results. Our search was guided by the relations between the ADAC-Problem and the "standard" optimization problems described in the last section.

5.3.1 No Overtime Allowed

As pointed out in the introduction, our original formulation of the ADAC-Problem did not consider overtime costs in the objective function. Instead, all servers were required to finish their schedule by time T. However, this restriction has a bad side-effect: given any deterministic online algorithm ALG, it is possible to construct an input sequence σ for which ALG even fails to find a feasible solution, which is, on the other hand, available to the offline adversary.

Figure 5.3 shows an example of such a sequence. It consists only of one request r, released at time $\frac{T}{2}$ at a point $a \in X$ whose distance to the nearest server is at least $\varepsilon > 0$, and whose distance to the depot is at most $\frac{T}{2}$. Since X is connected and smooth, it is always possible to choose a sufficiently small ε for which such a

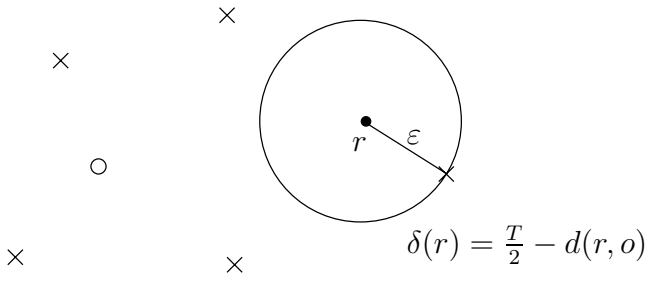


Figure 5.3: A bad example if no overtimes are allowed. A request r is issued at a distance of ε from the nearest server. Due to its duration, r has to be attended immediately after having been released to allow the server return to the depot on time.

point exists. The service duration of the request is

$$\delta(r) = \frac{T}{2} - d(a, o).$$

Now remark that ALG has to move a server at least for ε time units before reaching r. Hence, it is easy to see that this server will not be able to return to the depot by time T.

On the other hand, an offline adversary ADV starts moving one server towards a at time 0, while leaving all other servers at the depot. This server has already reached a by the time r is released, since $d(a, o)$ was chosen to be less or equal than $\frac{T}{2}$. ADV starts serving r immediately, and then heads the server back towards the depot, to arrive exactly at the deadline T.

5.3.2 Degenerated Service Costs

The example of the last subsection motivated us to change the problem formulation and allow penalized overtimes, resulting in the cost structure shown in (5.1). Let us consider now a second "degenerated" situation that may arise if the service costs can be chosen arbitrarily.

Let K be some large positive number, and choose ε such that $0 < 2K\varepsilon < T$. Figure 5.4 shows the construction of this example. At time $t = 0$, k requests are released at the depot, with deadlines equal to zero, service costs equal to ε and service durations of $\frac{T}{2}$. (In the figure, the requests have been drawn *near* the depot, for clarity). The way the input sequence σ is continued, depends on what the online algorithm ALG has done by time $t' := K\varepsilon$.

> **Case I:** Not all servers are busy at time t'
> This means that there is at least one request for which service has yet not started at this time. (Remark that due to the choice of ε, it is not possible that service has finished for any request by t'). Hence, ALG has to pay at least

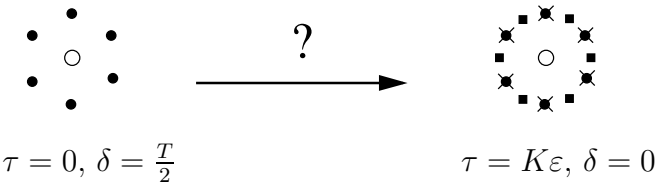


Figure 5.4: A degenerated case if service costs are not restricted. At time 0, k requests with duration $\frac{T}{2}$ are released at the depot. If all servers are busy after $K\varepsilon$ time units, a second wave of short requests (small squares) is issued, otherwise the input sequence is terminated.

the late costs associated to this request. In this case, the input sequence σ is ended here, and

$$\mathsf{ALG}(\sigma) \geq \mathsf{ALG}(\sigma, c \cdot W^{+}_{\mathrm{sum}}) \geq K\varepsilon c_{\mathrm{late}},$$

where $c \cdot W^{+}_{\mathrm{sum}} := kc_{\mathrm{late}}W^{+}_{\mathrm{avg}}$ denotes the sum of late costs for all requests. The offline adversary ADV, meanwhile, may attend all requests immediately after they have been released at time zero, and incurs therefore zero late costs. Moreover, since it does not have to pay any travel or overtime costs either, its total cost reduces to the sum s_{sum} of service costs:

$$\mathsf{ADV}(\sigma) = \mathsf{ADV}(\sigma, s_{\mathrm{sum}}) = k\varepsilon.$$

It then follows for the competitive ratio of ALG,

$$\frac{\mathsf{ALG}(\sigma)}{\mathsf{OPT}(\sigma)} \geq \frac{Kc_{\mathrm{late}}}{k}. \tag{5.3}$$

Case II: All servers are busy at time t'
If this happens, k new requests are released at this time, all of them located at the depot and with deadline t', duration equal to zero, and service cost of ε. Since each of these requests has to wait at least until time $\frac{T}{2}$ before being attended,

$$\mathsf{ALG}(\sigma) \geq \mathsf{ALG}(\sigma, c \cdot W^{+}_{\mathrm{sum}}) \geq k\left(\frac{T}{2} - K\varepsilon\right) c_{\mathrm{late}}.$$

On the other hand, ADV waits until time t' and serves the "small" requests first. This involves a late cost of $K\varepsilon c_{\mathrm{late}}$ for each of the requests released at time zero. It is straightforward to see that, together with service costs, these are the only costs incurred by ADV. Hence,

$$\mathsf{ADV}(\sigma) = kK\varepsilon c_{\mathrm{late}} + 2k\varepsilon.$$

Thus, the competitive ratio is at least

$$\frac{\mathsf{ALG}(\sigma)}{\mathsf{OPT}(\sigma)} \geq \frac{k\left(\frac{T}{2} - K\varepsilon\right) c_{\text{late}}}{kK\varepsilon c_{\text{late}} + 2k\varepsilon} = \frac{\frac{T}{2}c_{\text{late}} + 2\varepsilon}{K\varepsilon c_{\text{late}} + 2\varepsilon} - 1 \geq \frac{Tc_{\text{late}}}{\varepsilon(2Kc_{\text{late}} + 4)} - 1. \tag{5.4}$$

Therefore, it is possible to force the competitive ratio to be larger than any positive constant, by choosing the values of K and ε adequately: choose K large enough in (5.3) and, for this fixed K, choose ε small enough in (5.4). This implies that there is no competitive deterministic algorithm for the general ADAC-Problem with objective function as given by (5.1).

Observe that the construction of this lower bound relies on a somehow weird asymmetry: all requests have the same service costs despite of the fact that the first ones have a much longer duration. (If not, the ratio in the second case cannot be chosen arbitrarily bad). In practice, however, service costs are usually correlated (up to a certain degree) to service times, i.e., to the durations requests. We decided therefore to refine our model and admit only such input sequences where the service cost of any request is equal to its duration, multiplied with a *service coefficient* c_{svc}. In other words, the objective function (5.1) is now replaced by:

$$z = \sum_{i=1}^{k} D_i + \sum_{j=1}^{n} c_{\text{svc}}\delta(r_j) + \sum_{j=1}^{n} c_{\text{late}}\left(t(r_j) - \theta(r_j)\right)^+ + \sum_{i=1}^{k} c_{\text{ot}}\left(t_i^* - T\right)^+. \tag{5.5}$$

5.3.3 Arbitrarily Small Durations

Another family of input sequences for which the competitive ratio of any deterministic online algorithm is unbounded can be constructed if requests with arbitrarily small durations are allowed. Let N be a positive integer and $R = \frac{T}{4}$. Choose a set $A = \{a_0, \ldots a_k\}$ of $k+1$ different points of X, all of them located within a circle of radius R around the depot, and let

$$\varepsilon = \min_{a_i, a_j \in A} \frac{d(a_i, a_j)}{2}.$$

The existence of this set of points follows from the fact that X is connected and smooth. Observe that the balls $B(a_i, \varepsilon)$, $0 \leq i \leq k$ do not intersect each other. Moreover, since there are only k servers, notice that at any time there is always a ball with no server in it, i.e., there is a point in A whose distance to the nearest server is at least ε.

Given a deterministic online algorithm ALG, the input sequence σ consists of N requests released at time $t = R$ at such a point $a^* \in A$ at least ε length units away from ALG's next server, as indicated in figure 5.5. All requests have the same deadline R, and the same duration $\hat{\delta}$, with $N\hat{\delta} \leq \frac{T}{2}$.

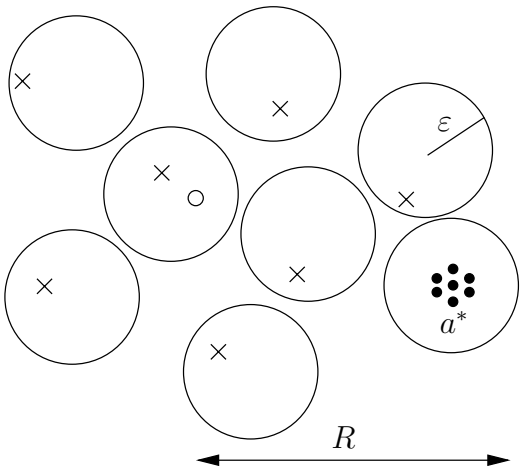


Figure 5.5: Requests having arbitrarily small durations do also constitute a problem. Given $k+1$ non-intersecting balls, N very short requests are released at the center a^* of the ball which does not contain a server.

No matter what ALG does, the algorithm cannot start processing any request before time $R+\varepsilon$, since it first has to move a server to a^*. Therefore, it incurs some unavoidable late costs of at least $\varepsilon\, c_{\mathrm{late}}$ for each request, and we conclude

$$\mathsf{ALG}(\sigma) \geq \mathsf{ALG}(\sigma, c \cdot W^{+}_{\mathrm{sum}}) \geq N\varepsilon c_{\mathrm{late}}. \tag{5.6}$$

In fact, the costs for ALG are higher, but bound (5.6) suffices for our purposes.

A (not necessarily optimal) offline adversary ADV, on the other hand, could start moving a server to a^* already at time zero. Because all points of A are located within R length units of the depot, there is enough time for this server to reach a^* by time t. Thus, ADV can start processing the requests (one after the other) immediately after they have been released. There is still lateness involved, since latter requests have to wait until the earlier ones have been served. However, the total late cost is in this case

$$\mathsf{ADV}(\sigma, c \cdot W^{+}_{\mathrm{sum}}) = c_{\mathrm{late}} \sum_{j=2}^{N} (j-1)\hat{\delta} = \frac{N(N-1)}{2} c_{\mathrm{late}} \hat{\delta}.$$

Furthermore, a moment of thought reveals that ADV does not incur any overtime costs, due to the choice of R and $\hat{\delta}$. As for travel and service costs, the former are bounded by $2d(a^*, o) \leq 2R$, whereas the latter are given by the cost definition (5.5). Hence,

$$\mathsf{ADV}(\sigma) \leq 2R + N c_{\mathrm{svc}} \hat{\delta} + \frac{N(N-1)}{2} c_{\mathrm{late}} \hat{\delta}. \tag{5.7}$$

Finally, from (5.6) and (5.7) it follows that, if $\hat{\delta}$ is chosen to be sufficiently

small, then

$$\frac{\mathsf{ALG}(\sigma)}{\mathsf{OPT}(\sigma)} \geq \frac{N\varepsilon c_{\text{late}}}{2R} - 1.$$

Since the number N of requests can be chosen freely, and is in particular independent from the values of ε and R, this quotient can become arbitrarily large. It follows that there is no deterministic online algorithm which can achieve a constant competitive ratio for the second version of the ADAC-Problem with cost function (5.5). This result incited us to introduce a further restriction: from now on, we shall allow only such input sequences where all requests have durations larger than or equal to a fixed value (given by a new parameter), the *minimum request duration* δ^*.

Before continuing with the next lower bound, let us remark that the results of this section can be extended for the case of a non-abusive and fair adversary, in the sense defined by Krumke et al. [2002] and Blom et al. [2000], respectively. It suffices, for instance, to issue a first "wave" of requests at time zero, one at each point of A, all having deadlines of $2R + \hat{\delta}$ and durations of $\hat{\delta}$. This increases the costs for the offline adversary, but it is easy to show that still a ratio of $O(N)$ can be obtained by choosing $\hat{\delta}$ sufficiently small:

$$\frac{\mathsf{ALG}(\sigma)}{\mathsf{OPT}(\sigma)} \geq \frac{N\varepsilon c_{\text{late}}}{(2k+1)R + (k+1+N)c_{\text{svc}}\hat{\delta} + \frac{N(N-1)}{2}c_{\text{late}}\hat{\delta}} \geq \frac{N\varepsilon c_{\text{late}}}{(2k+1)R} - 1.$$

5.3.4 Unbounded Metric Space

If the metric space X is unbounded, it is also possible to obtain input sequences that induce arbitrarily bad competitive ratios. Given a positive integer N, choose a point $a_k \in X$ such that $d(a_k, o) = 2kN^2$. Then, along the shortest path between o and a_k, choose $k-1$ further points $a_1, \ldots, a_{k-1}$ such that $d(a_j, o) = 2jN^2$, for $j = 1, \ldots, k-1$. As in the last examples, the existence of such points is guaranteed by the fact that X is connected and smooth. We denote by A the set $\{a_1, \ldots, a_k\}$. Notice that this set satisfies two basic properties:

$$\min_{a,a' \in A} d(a, a') = 2N^2 \tag{5.8}$$

$$R := \max_{a \in A} d(a, o) = 2kN^2 \tag{5.9}$$

The input sequence σ starts with k requests, released at time zero, one at each of the points in A, as shown in Figure 5.6. All requests have deadlines equal to R and durations of N^2. Let t_0 be the earliest time when an online algorithm ALG starts the service of (at least) one of these requests. We distinguish between two cases:

Case I: $t_0 \leq N^{2.5}$
In this case, observe that due to (5.8) there is a point $a^* \in A$ whose distance to the next *free* server is at least N^2: there are at most $k-1$ free servers (at

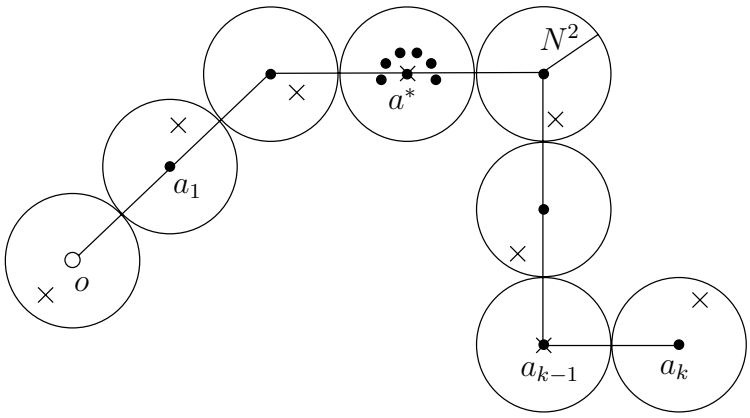


Figure 5.6: In an unbounded metric space, k requests of duration N^2 are given at points in distance of $2N^2$ length units of each other. As soon as a server starts processing a request, N new short requests are released at a point a^* at least N^2 units away from the nearest free server.

least one is busy processing a request), and there are k non-intersecting balls $B(a_i, N^2)$ with radii N^2 and centers at the points in A. At time $t = t_0$, N new requests of duration 1 and deadline $t_0 + 1$ are released at a^*.

To attend any of these new requests, ALG has either to move a free server to a^*, or to wait that a "busy" server on a^* finishes its current duty. No matter which strategy it chooses, each of the N new events has to wait at least for N^2 time units before being served. Hence,

$$\mathsf{ALG}(\sigma) \geq \mathsf{ALG}(\sigma, c \cdot W^+_{\mathrm{sum}}) \geq c_{\mathrm{late}}\, N^3. \tag{5.10}$$

An offline adversary ADV would move the servers to all points in A, but would not start service of the request in a^* immediately. Instead, it would just keep that server waiting there until the "small" requests arise, and it would process them at first. Thus, the large request will have to wait for $t_0 + N$ time units, while the total wait time of all small requests is like in the last example $\frac{1}{2}N(N-1)$. Therefore,

$$\mathsf{ADV}(\sigma, c \cdot W^+_{\mathrm{sum}}) \leq \left(t_0 + N + \frac{N(N-1)}{2}\right) c_{\mathrm{late}} \leq 2c_{\mathrm{late}}\, N^{2.5}.$$

Similarly, the overtime costs $c \cdot C^+_{\mathrm{sum}}$ for ADV can be bounded as follows: the $k-1$ servers which attended the requests located at $A \setminus \{a^*\}$ will have returned to the depot at latest by time $2R + N^2$, the server at a^* at latest by

time $t_0 + N + N^2 + R$. Adding up these quantities together,

$$\begin{aligned}\mathsf{ADV}(\sigma, c \cdot C^+_{\mathrm{sum}}) \leq & (k-1)\left(2R + N^2 - T\right)^+ c_{\mathrm{ot}} \\ & + \left(t_0 + N + N^2 + R - T\right)^+ c_{\mathrm{ot}} \\ \leq & \left[(2k-1)R + kN^2 + N + N^{2.5}\right] c_{\mathrm{ot}} \\ \leq & \left[(2k-1)R + (k+2)N^{2.5}\right] c_{\mathrm{ot}} \\ \leq & (4k^2 - k + 2)\, c_{\mathrm{ot}}\, N^{2.5} \\ \leq & 6k^2 c_{\mathrm{ot}}\, N^{2.5},\end{aligned}$$

where the last but one inequality follows from (5.9).

Travel and service costs are straightforward to bound. Thus, putting all pieces together,

$$\begin{aligned}\mathsf{ADV}(\sigma) &\leq 2kR + \left(kN^2 + N\right) c_{\mathrm{svc}} + 2N^{2.5} c_{\mathrm{late}} + 6k^2 N^{2.5} c_{\mathrm{ot}} \\ &\leq \left[4k^2 + (k+1)c_{\mathrm{svc}} + 2c_{\mathrm{late}} + 6k^2 c_{\mathrm{ot}}\right] N^{2.5}. \qquad (5.11)\end{aligned}$$

Again, in the last inequality we made use of (5.9) to substitute R.

From (5.10) and (5.11), we obtain the following ratio for the solution values:

$$\frac{\mathsf{ALG}(\sigma)}{\mathsf{OPT}(\sigma)} \geq \frac{c_{\mathrm{late}}}{4k^2 + (k+1)c_{\mathrm{svc}} + 2c_{\mathrm{late}} + 6k^2 c_{\mathrm{ot}}} \sqrt{N}.$$

Case II: $t_0 > N^{2.5}$

If ALG takes that long to start attending the first request, then the input sequence is simply ended. As every request has to wait for service at least until t_0, it follows that

$$\mathsf{ALG}(\sigma) \geq \mathsf{ALG}(\sigma, c \cdot W^+_{\mathrm{sum}}) \geq k\,(t_0 - R)^+\, c_{\mathrm{late}} \geq k\left(N^{2.5} - 2kN^2\right)\, c_{\mathrm{late}}.$$

The offline adversary does not lose any time, but moves the servers immediately to the requests, and starts processing them. Due to the choice of the deadlines, ADV doesn't incur any late costs. Moreover, it can be seen easily that

$$\begin{aligned}\mathsf{ADV}(\sigma) &\leq 2kR + kN^2 c_{\mathrm{svc}} + k\left(2R + N^2 - T\right)^+ c_{\mathrm{ot}} \\ &\leq \left[4k^2 + kc_{\mathrm{svc}} + k(4k+1)c_{\mathrm{ot}}\right] N^2.\end{aligned}$$

Finally, combining the last two observations, it follows for this case

$$\frac{\mathsf{ALG}(\sigma)}{\mathsf{OPT}(\sigma)} \geq \frac{c_{\mathrm{late}}}{4k + c_{\mathrm{svc}} + (4k+1)c_{\mathrm{ot}}} \left(\sqrt{N} - 2k\right).$$

As N may become arbitrarily large, the ratio between the solutions of ALG and the offline adversary cannot be bounded by any constant. Observe that the adversary in this example is also fair and non-abusive.

5.3.5 Heavy Load

We present now a last example for which the performance of any deterministic online algorithm can get arbitrarily bad. Our input sequence does not include requests with arbitrarily short durations. Nor does it require the metric space to be unbounded. Its idea is to force the system to enter an *overloaded* state. We just describe the example here, leaving the formal definition of "heavy load" (as opposed to reasonable load) for the next section.

Let N be a positive integer. At time zero, a request r_0 is released at the depot with duration N and deadline 0. Let t_0 be the time when the online algorithm ALG starts serving this request. Again, we consider two cases.

Case I: $t_0 \leq N^{1.5}$
In this case, the input sequence σ is continued in the following manner: at times $t_i := t_0 + i - 1$, for $i = 1, \ldots, N$, "waves" of k requests each are released at the depot, like in figure 5.7. All requests have durations of 1 and deadlines equal to their release times. [3]

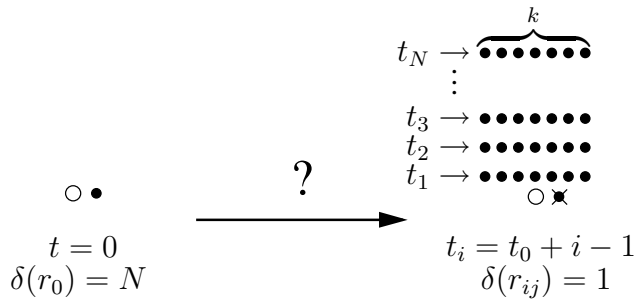


Figure 5.7: An example of heavy load. While one server is busy attending a long request, waves of k requests each are released one after the other. The online algorithm does not have enough resources to service them on time and accumulates late costs.

Observe that while these waves of requests are released, ALG has only $k-1$ free servers, since one server is busy processing r_0. However, with each new wave a total "workload" arises, which requires k servers to be processed in time. Thus, no matter which decisions ALG takes, the algorithm just doesn't have enough available resources to keep serving the requests at the same rate they come in. As a result, some requests will have to be postponed, incurring increasing late costs.

To estimate these late costs, remark that by time t_0+N a total of Nk requests have been released, but ALG cannot have processed more than $N(k-1)$. Thus, there are at least N pending requests. From these, at most k have been released at time $t_N = t_0 + N - 1$, i.e., have been waiting for service for one

[3] We assume w.l.o.g. that the minimum request duration δ^* is larger than one. If not, all times in the example must be scaled up appropriately.

time unit. Another k requests have been waiting for at least two time units, and so on. Adding these figures up, we obtain

$$\begin{aligned}\mathsf{ALG}(\sigma) &\geq \mathsf{ALG}(\sigma, c \cdot W^{+}_{\mathrm{sum}}) \geq c_{\mathrm{late}}\, k \sum_{j=1}^{\lfloor \frac{N}{k} \rfloor} j \\ &\geq c_{\mathrm{late}}\, k \frac{\lfloor \frac{N}{k} \rfloor \left(\lfloor \frac{N}{k} \rfloor + 1 \right)}{2} \geq \frac{N^2 - Nk}{2k}\, c_{\mathrm{late}}. \end{aligned} \tag{5.12}$$

On the other hand, an offline adversary ADV can process the N waves of requests first, leaving the "long" request for the end. Thus, it has enough servers to attend the requests as they arise and incurs late costs only for r_0, which has to wait for a total time of $t_0 + N$. Adding up the service and overtime costs, we obtain the following result:

$$\begin{aligned}\mathsf{ADV}(\sigma) \leq & N(k+1)c_{\mathrm{svc}} + (t_0 + N)\, c_{\mathrm{late}} \\ & + \left[(k-1)\, (t_0 + N - T)^{+} + (t_0 + 2N - T)^{+} \right] c_{\mathrm{ot}} \\ \leq & N^{1.5} \left[(k+1)c_{\mathrm{svc}} + 2c_{\mathrm{late}} + (2k+1)c_{\mathrm{ot}} \right]. \end{aligned} \tag{5.13}$$

Combining (5.12) and (5.13), we finally obtain for sufficiently large N:

$$\frac{\mathsf{ALG}(\sigma)}{\mathsf{OPT}(\sigma)} \geq \frac{c_{\mathrm{late}}}{2k \left[(k+1)c_{\mathrm{svc}} + 2c_{\mathrm{late}} + (2k+1)c_{\mathrm{ot}} \right]} \left(\sqrt{N} - 1 \right).$$

Case II: $t_0 > N^{1.5}$
If, on the other hand, ALG takes too long to attend r_0, then the input sequence is ended after this request. The costs for the online algorithm are thus

$$\mathsf{ALG}(\sigma) \geq \mathsf{ALG}(\sigma, c \cdot W^{+}_{\mathrm{sum}}) \geq N^{1.5} c_{\mathrm{late}}. \tag{5.14}$$

The offline adversary ADV attends the request immediately after it is released and incurs no late costs. Hence,

$$\mathsf{ADV}(\sigma) \leq N c_{\mathrm{svc}} + (N - T)^{+}\, c_{\mathrm{ot}} \leq N(c_{\mathrm{svc}} + c_{\mathrm{ot}}). \tag{5.15}$$

From (5.14) and (5.15) it follows:

$$\frac{\mathsf{ALG}(\sigma)}{\mathsf{OPT}(\sigma)} \geq \frac{c_{\mathrm{late}}}{c_{\mathrm{svc}} + c_{\mathrm{ot}}} \sqrt{N}.$$

Thus, the ratio between the solution values is in both cases bounded from below by a quantity proportional to $\sqrt{N}$ and will become arbitrarily large as N increases. Observe that all requests from this example (in the first case) are released within a time interval of length $t_0 + N - 1$, but require at least $t_0 + N$ time units to be processed. This gives an intuitive idea of what is meant by saying that the system is under "heavy load". In the next section, we look back on the concepts from Section 3.3.2, introduce a more precise formal framework to deal with system load, and use it to specify a last restriction on the input sequence.

5.4 A Competitive Deterministic Algorithm

After having reviewed some bad examples for the ADAC-Problem, we now introduce an online deterministic algorithm that has constant competitive ratio for all input sequences that satisfy certain conditions. Besides of requiring the service costs to be proportional to the service times, and all service times to be larger than or equal to a certain (positive) value δ^*, we will further consider only such problem instances where the system is under *reasonable load*. This last condition also implies that the underlying metric space X must be bounded.

The concept of reasonable load (introduced by Hauptmeier et al. [2000]) was discussed in Section 3.3.2, in the context of the online dial-a-ride problem. Roughly speaking, the idea is to avoid such input sequences where requests arrive faster than they can be served. Let us at first recall the key formal definitions that are relevant for the purpose of this section (some slight changes have been introduced to adapt them to the notation and particular requirements of our current problem). For simplicity, we consider an input sequence σ both as a sequence and as a set of requests.

5.4.1 Definition (Release-Span) *For any request set $S \subseteq \sigma$, its* release-span $\partial(S)$ *is the time between the moments at which the earliest and the latest request in S are released, i.e.,*

$$\partial(S) := \max_{r \in S} \tau(r) - \min_{r \in S} \tau(r).$$

5.4.2 Definition (Offline Makespan) *Given a request set $S \subseteq \sigma$, its* offline makespan $C_{\max}^{\text{offline}}(S)$ *is the minimal total duration of a (feasible) service plan to attend all requests in S, if their release times are all changed to zero. This plan is required to start and end at a configuration where all servers are located at the depot.*

Common sense suggests that the "load level" of a system is determined by a relation between the former two quantities. This idea is made formal in the following definitions:

5.4.3 Definition (Load Bound) *A weakly monotone increasing real function f is called a* load bound *on a request set σ if, for any $\partial \in \mathbb{R}$ and any $S \subseteq \sigma$ with $\partial(S) \leq \partial$, the following holds:*

$$C_{\max}^{\text{offline}}(S) \leq f(\partial).$$

5.4.4 Definition (Reasonable Load) *A load bound f is called* (Δ, ρ)-reasonable *for some $\Delta, \rho \in \mathbb{R}$ if, for all $\partial \geq \Delta$,*

$$\rho f(\partial) \leq \partial.$$

An input sequence σ is called (Δ, ρ)-reasonable if it has a (Δ, ρ)-reasonable load bound.

Observe that, if an input sequence σ is (Δ, ρ)-reasonable, then there cannot be any subsequence $S \subseteq \sigma$ with $\partial(S) \leq \Delta$ but $C_{\max}^{\text{offline}}(S) > \Delta/\rho$. In particular, no subsequence consisting of only one element may have an arbitrarily large offline

makespan. This implies, among other things, that requests must lie within a certain range from the depot, i.e., that the metric space X must be bounded.

The main objective of this section is to prove that the following online algorithm IGNORE achieves a constant competitive ratio over all (Δ, ρ)-reasonable problem instances that comply to the other restrictions stated at the beginning (service costs proportional to service times, and service times bounded from below by some constant δ^*):

IGNORE

Keep all servers at the depot until the first set S_0 of requests arises (which may be a singleton). Compute for all requests in S_0 a service plan of makespan at most $\rho\, C^{\text{offline}}_{\max}(S_0)$ (including the time needed to return the servers to the depot at the end). Start execution of this plan immediately.

Let S_{i+1} ($i \geq 0$) be the set of requests that arise while the service schedule for request set S_i is being executed. These requests are simply ignored *for some time. When execution of the plan for S_i finishes,* IGNORE *tests if there are pending requests (i.e., if $S_{i+1} \neq \emptyset$). As long as this is true, a new schedule is computed and the cycle starts again. When there are no pending requests anymore, the algorithm returns to its initial state (just keep servers at the depot and "wait for orders").*

Before proceeding with the analysis, let us state a small side-remark. The reason for introducing the constant ρ in the definition of reasonable load is to allow the use of approximation algorithms when the intermediate service schedules are computed. Observe that computing such an schedule implies solving an (offline) instance of the dial-a-ride problem, which is $\mathcal{NP}$-hard itself. Hence, for larger problems it might not be possible to find an optimal solution within a reasonable computation time. Our competitive analysis results, however, are based on the assumption that these intermediate computations demand no (significant) time.

Let $\ell_0, \ldots, \ell_{m-1}$ be the points in time at which IGNORE computes a new service schedule and ℓ_m the time at which the execution of the last schedule has finished. Under the assumption of reasonable load, it is possible to show that the time required to carry out each of these partial schedules is bounded by a constant. More precisely,

$$\ell_{i+1} - \ell_i \leq \Delta, \text{ for } i = 0, \ldots, m-1. \tag{5.16}$$

Indeed, by construction we know that $\partial(S_0) = 0$. Therefore, given any load bound f on σ,

$$\ell_1 - \ell_0 \leq \rho\, C^{\text{offline}}_{\max}(S_0) \leq \rho f(0) \leq \rho f(\Delta) \leq \Delta,$$

where the first inequality follows from the definition of the algorithm, the second and the third inequalities from the fact that f is a load bound, and the last inequality is the reasonable load assumption. By induction, we then have for $i = 1, \ldots, m-1$ that

$$\ell_{i+1} - \ell_i \leq \rho\, C_{\max}^{\textit{offline}}(S_i) \leq \rho\, f(\ell_i - \ell_{i-1}) \leq \rho f(\Delta) \leq \Delta.$$

Equation (5.16) has a key implication for the competitive analysis: since, for $i = 1, \ldots, m-1$, all requests in S_i arrive in the time interval $(\ell_{i-1}, \ell_i]$ and are served in the interval $(\ell_i, \ell_{i+1}]$, the delay of any request $r \in \sigma$ is at most 2Δ. (For the requests in S_0 the delay is even no larger than Δ). Hence, the total lateness costs incurred by IGNORE are bounded by

$$\textsf{IGNORE}(\sigma, c \cdot W_{\text{sum}}^{+}) \leq 2n\Delta\, c_{\text{late}} \leq \frac{2k\ell_m \Delta c_{\text{late}}}{\delta^*}, \tag{5.17}$$

where $n = |\sigma|$. Considering that all requests have service times of at least δ^*, n is bounded by the amount of "total workload" that the k servers are able to carry out until time ℓ_m, as reflected in the second inequality.

In a similar way, the total distance D_{sum} traveled by the servers before returning home at time ℓ_m cannot be longer than

$$\textsf{IGNORE}(\sigma, D_{\text{sum}}) \leq k\ell_m. \tag{5.18}$$

Finally, the overtime costs for IGNORE are also obviously bounded by

$$\textsf{IGNORE}(\sigma, c \cdot C_{\text{sum}}^{+}) \leq k\,(\ell_m - T)^+\, c_{\text{ot}}. \tag{5.19}$$

On the other hand, observe that, since there are requests issued after time ℓ_{m-2}, the offline adversary must keep at least one server working until that time. Thus,

$$\textsf{OPT}(\sigma, c \cdot C_{\text{sum}}^{+}) \geq (\ell_{m-2} - T)^+\, c_{\text{ot}} \geq (\ell_m - 2\Delta - T)^+\, c_{\text{ot}}.$$

Ignoring travel and lateness costs, the total costs for the offline adversary can be (roughly) bounded from below as follows:

$$\begin{aligned} \textsf{OPT}(\sigma) &\geq S + (\ell_m - 2\Delta - T)^+\, c_{\text{ot}} \\ &\geq n c_{\text{svc}} \delta^* + (\ell_m - 2\Delta - T)^+\, c_{\text{ot}} \\ &\geq c_{\text{svc}} \delta^* + (\ell_m - 2\Delta - T)^+\, c_{\text{ot}}, \end{aligned} \tag{5.20}$$

where $S = \sum_{i=1}^{n} c_{\text{svc}} \delta(r_i)$ is the sum of all service costs. Recall that these costs were required to be proportional to the service times. Although this bound is very weak, it turns out to be sufficient for the goals of our analysis.

Combining the results (5.17) – (5.20), it is finally possible to derive a competitive ratio for IGNORE. Let us consider two possible situations:

Case I: $\ell_{\mathbf{m}} \leq \mathbf{T} + \mathbf{2\Delta}$
In this case, from (5.19) it follows that the overtime costs incurred by IGNORE are bounded by $2k\Delta\, c_{\mathrm{ot}}$ and hence,

$$\begin{aligned}\frac{\mathsf{IGNORE}(\sigma)}{\mathsf{OPT}(\sigma)} &\leq \frac{\frac{2k\ell_m\Delta\, c_{\mathrm{late}}}{\delta^*} + k\ell_m + 2k\Delta\, c_{\mathrm{ot}} + S}{S} \\ &\leq 1 + \frac{\frac{2k\ell_m\Delta\, c_{\mathrm{late}}}{\delta^*} + k\ell_m + 2k\Delta\, c_{\mathrm{ot}}}{c_{\mathrm{svc}}\delta^*} \\ &\leq 1 + k\,\frac{2(T+2\Delta)\Delta\, c_{\mathrm{late}} + (T+2\Delta)\delta^* + 2\Delta\, c_{\mathrm{ot}}\delta^*}{c_{\mathrm{svc}}(\delta^*)^2},\end{aligned}$$

which gives a constant bound for any fixed combination of the problem parameters.

Case II: $\ell_{\mathbf{m}} > \mathbf{T} + \mathbf{2\Delta}$
If the total completion time of IGNORE exceeds this critical value, on the other hand, then (5.20) guarantees that there will be unavoidable overtime costs for the offline adversary. Thus,

$$\begin{aligned}\frac{\mathsf{IGNORE}(\sigma)}{\mathsf{OPT}(\sigma)} \leq& \frac{\frac{2k\ell_m\Delta\, c_{\mathrm{late}}}{\delta^*} + k\ell_m + k(\ell_m - T)c_{\mathrm{ot}} + S}{(\ell_m - 2\Delta - T)c_{\mathrm{ot}} + S} \\ =& \frac{2k\ell_m\Delta\, c_{\mathrm{late}} + k\delta^*\ell_m + k\delta^*(\ell_m - T)c_{\mathrm{ot}} + S\delta^*}{(\ell_m - 2\Delta - T)\,\delta^* c_{\mathrm{ot}} + S\delta^*} \\ =& \frac{(\ell_m - 2\Delta - T)(2k\Delta\, c_{\mathrm{late}} + k\delta^* + k\delta^* c_{\mathrm{ot}})}{(\ell_m - 2\Delta - T)\delta^* c_{\mathrm{ot}} + S\delta^*} \\ &+ \frac{(2\Delta + T)(2k\Delta\, c_{\mathrm{late}} + k\delta^*) + 2\Delta k\delta^* c_{\mathrm{ot}} + S\delta^*}{(\ell_m - 2\Delta - T)\,\delta^* c_{\mathrm{ot}} + S\delta^*} \\ <& \frac{2k\Delta\, c_{\mathrm{late}} + k\delta^* + k\delta^* c_{\mathrm{ot}}}{\delta^* c_{\mathrm{ot}}} \\ &+ \frac{(2\Delta + T)(2k\Delta\, c_{\mathrm{late}} + k\delta^*) + 2\Delta k\delta^* c_{\mathrm{ot}} + S\delta^*}{(\ell_m - 2\Delta - T)\,\delta^* c_{\mathrm{ot}} + S\delta^*}\end{aligned}$$

$$\begin{aligned}\leq& \frac{2k\Delta\, c_{\mathrm{late}} + k\delta^* + k\delta^* c_{\mathrm{ot}}}{\delta^* c_{\mathrm{ot}}} \\ &+ \frac{(2\Delta + T)(2k\Delta\, c_{\mathrm{late}} + k\delta^*) + 2\Delta k\delta^* c_{\mathrm{ot}} + c_{\mathrm{svc}}(\delta^*)^2}{c_{\mathrm{svc}}(\delta^*)^2},\end{aligned}$$

and the right-hand side of the last inequality is again a function of the problem parameters which does not depend on the input sequence σ.

We summarize the results of this section in the following theorem:

5.4.5 Theorem *IGNORE is K-competitive for all ADAC-Problem instances of (Δ, ρ)-reasonable load for which service costs are proportional to durations of requests, and all durations are bounded from below by a positive value δ^*. Moreover, K is a constant whose value if determined by the problem parameters $c_{\mathrm{svc}}, c_{\mathrm{late}}, c_{\mathrm{ot}}$, T, k, δ^* and Δ.*

Chapter 6

Set Packing with Small Subsets

6.1 Introduction

The objective of this chapter is to study a special kind of set packing problems, and their associated polytopes. Given a $0/1$-matrix $A \in \{0,1\}^{m \times n}$ and a nonnegative cost vector $c \in \mathbb{Q}_+^n$, the set partitioning problem associated to A was defined in Chapter 4 as the integer program

$$\text{(SPP)} \quad \begin{cases} \min \ c^T x \\ \text{s.t.} \\ \quad Ax = \mathbf{1}, \\ \quad x \in \{0,1\}^n . \end{cases}$$

If in addition every column of A has at most κ entries equal to one (i.e., if $\text{supp}(A_{\cdot j}) \leq \kappa$, $\forall j = 1, \ldots, n$), then we call SPP a *κ-set partitioning problem*. This notation is better understood by looking at the problem formulation in set theoretic terms: we are given a family $\mathcal{E}$ of n subsets of a ground set V, with $|V| = m$. Each subset $S_j \in \mathcal{E}$ (which corresponds to the support of column $A_{\cdot j}$) has cardinality smaller than or equal to κ, and has a cost $c_j \in \mathbb{Q}_+$ associated to it. The task is to find a subfamily $\mathcal{E}^* \subseteq \mathcal{E}$ such that the sets in $\mathcal{E}^*$ form a partition of V, and the sum of their costs is minimal among all subfamilies from $\mathcal{E}$ satisfying this condition. The expression "with small subsets" is used to emphasize that we are interested in those cases where κ is a small integer constant. We define the κ-set packing and κ-set covering problems in a similar way.

If κ is equal to one, the problem is trivial and can be solved by a greedy algorithm. For the case when $\kappa = 2$ and all subsets in $\mathcal{E}$ are uniform, i.e., they contain *exactly* two elements, the κ-set partitioning problem can be reduced to the problem of finding a perfect matching of minimum cost in a graph G, whose nodes are the elements of V and whose (weighted) edges are the 2-subsets in $\mathcal{E}$ with their corresponding costs.

This reduction is also possible for the nonuniform case, although some extra work has to be invested in dealing with the singleton subsets in $\mathcal{E}$ (empty subsets

can be discarded from the input, since they are not needed in any optimal solution, as the cost vector is nonnegative). Suppose there are s such singletons, and introduce s "artificial" nodes in G. Now each singleton is represented by a weighted edge that connects the element it contains with a uniquely determined artificial node. Moreover, we add a set $\mathcal{E}'$ of edges with cost equal to zero connecting every pair of artificial nodes.

Remark that the parity of m and s determines the parity of the number of singletons that are *not* used in a feasible solution of the set partitioning problem. This is exactly the quantity of artificial nodes that will *not* be covered by the edges associated to the singletons if we translate the solution to a matching in G. In fact, if $m+s$ is even, a moment of thought reveals that the number of uncovered artificial nodes is even, too. If this happens, it is straightforward to extend any solution of the set partitioning problem to a perfect matching in G having the same cost, by just taking an appropriate set of edges from $\mathcal{E}'$. On the other hand, if $m + s$ is odd, then we add to G an additional node and connect it with all artificial nodes using edges of cost zero. Again, any solution to the set partitioning problem reveals a perfect matching on G, and vice-versa.

Surprisingly, if $\kappa = 3$ and all subsets in $\mathcal{E}$ contain exactly three elements, then the κ-set partitioning problem becomes $\mathcal{NP}$-complete, a result that can be proved using a reduction from 3-SAT (see for instance Garey & Johnson [1979]). The problem has in this case no (satisfactory) equivalent formulation in graph theoretic terms, though it can still be stated as a matching problem in a hypergraph.

Moreover, while it is easy to show that all vertices of fractional relaxations of 2-set partitioning polytopes are half integral, it follows from the work of Chung et al. [1988] that the vertices of fractional polytopes related to 3-set covering problems may have arbitrary (rational) coordinates.

As discussed in Section 4.1, a usual approach when studying set partitioning problems from the viewpoint of polyhedral theory consists in looking at the packing and covering polytopes separately. In this chapter, we will focus exclusively on polytopes associated to κ-set packing problems, as there was not enough time in our schedule to examine the κ-covering polytopes as well. Our aim is to investigate how the structure of the packing polytopes is affected when the cardinality of the subsets is constrained. Most of the time, we will concentrate on the 3-set packing problem, for which we introduce an alternative formulation in the next section. Within the framework of this formulation, we then study in Sections 6.3 to 6.5 three "standard" classes of valid/facet defining linear inequalities for the packing polytope: clique, antiweb and web inequalities. In Section 6.6 we extend some results to the case of general κ-set packing problems, and state further questions for future research.

This research was at first motivated by set partitioning problems that appeared when solving instances of the VRPTW in a project for automating the online dispatching of service vehicles at ADAC (see Section 7.2 for the details). We solve this routing problem using a tour-based set partitioning model. Due to particular problem constraints, the feasible tours that come in question for any service

vehicle are strongly limited in the number of requests they can cover, so that the corresponding set partitioning instances involve matrices having a limited number of nonzero entries per column. However, there is yet a gap between the real-world requirements, and the current state of our theoretical research: while most of the results presented here concern the 3-set packing environment, the average tour length in a typical dispatch at ADAC lies between 4 and 6, and tours containing up to ten requests have also been found in some cases.

6.2 A Combinatorial Formulation: Packing Triangles

As anticipated, we shall at first focus our attention on the polytopes associated to the 3-set packing problem. We state here a new formulation for this problem as the task of "packing triangles". Before doing so, however, we need to briefly introduce some basic definitions and observations.

Let $A \in \{0,1\}^{m \times n}$ be the constraint matrix associated to a 3-set packing problem, i.e., A has the property $\text{supp}(A_{\cdot j}) \leq 3$, $\forall j \in \{1,\ldots,n\}$. Keeping the notation introduced in Chapter 4, we define the corresponding set packing polytope $P_I(A)$, and its fractional relaxation $P(A)$ as:

$$P_I(A) = \text{conv}\left\{x \in \{0,1\}^n \,:\, Ax \leq \mathbf{1}\right\}$$
$$P(A) = \left\{x \in \mathbb{R}^n_+ \,:\, Ax \leq \mathbf{1}\right\}.$$

From now on, we shall assume that A has no empty rows or columns. In this case, it is easy to show that both polytopes are contained in the unit hypercube, from which it follows that all inequalities of the form $x_j \leq 1$, $j \in \{1,\ldots,n\}$ are valid for them. Hence, if some column $A_{\cdot j}$ has only one or two nonzero entries, by adding sufficient (maybe duplicated) inequalities of the form $x_j \leq 1$, it is possible to obtain a new matrix A' such that $\left|\text{supp}(A'_{\cdot j})\right| = 3$ and both packing polytopes of A and A' are the same. Proceeding in this way, we prove the following.

6.2.1 Lemma *Any* 3*-set packing problem can be reduced to a* uniform 3*-set packing problem, in which all subsets contain* exactly *three elements.*

We shall therefore restrict our attention to uniform problems. Moreover, we will work with the set theoretic formulation presented in the introduction, which has in this case a nice combinatorial interpretation. Let $V = \{1,\ldots,m\}$ be a set of *points* and call any subset of V having cardinality equal to three a *triangle* in V. Define

$$\mathcal{E} = \{\text{supp}(A_{\cdot j}) \,:\, 1 \leq j \leq n\}$$

as the set of triangles in V associated to the columns of A. To each of these triangles, a nonnegative weight is assigned by the cost function c. The task is to find a "packing of triangles" of maximum weight, i.e., a set $\mathcal{E}^* \subseteq \mathcal{E}$ of triangles such that no point is contained in more than one triangle of $\mathcal{E}^*$, and the sum of the

weights of all triangles in $\mathcal{E}^*$ is maximal. Figure 6.1 shows an example. We call A the *point-triangle* incidence matrix.

Analogously, the linear relaxation of the set packing problem can be stated in this setting as the problem of assigning a positive quantity x_e to each triangle $e \in \mathcal{E}$, such that for each $v \in V$, the sum of the values corresponding to triangles that contain v is less or equal to one, while the total sum over all triangles in $\mathcal{E}$ is as large as possible. Notice that each point in V corresponds to a row of A and, therefore, to a linear constraint from the description of $P(A)$. Hence, we shall refer to these constraints as *point constraints*.

Finally, let us recall one basic concept from Section 4.2. Given a point-triangle incidence matrix $A \in \{0,1\}^{m \times n}$ associated with a triangle packing problem, we define the *conflict* or *column intersection graph* $\mathbb{G}(A) = (\mathbb{V}(A), \mathbb{E}(A))$ of A as follows:

$$\mathbb{V}(A) = \{1, \ldots, n\},$$
$$\mathbb{E}(A) = \{\{i, j\} \,:\, \text{supp}(A_{\cdot i}) \cap \text{supp}(A_{\cdot j}) \neq \emptyset\}.$$

As pointed out in the last chapter, one can obtain results concerning the structure of $P_I(A)$ by looking into the structure of $\mathbb{G}(A)$. Our approach in the next sections will be to exploit the triangle packing formulation presented here in order study three "standard" classes of subgraphs from $\mathbb{G}(A)$ that are known to give rise to valid (and some times facet defining) inequalities for $P_I(A)$: cliques, antiwebs and webs. We start by considering how cliques of triangles look like.

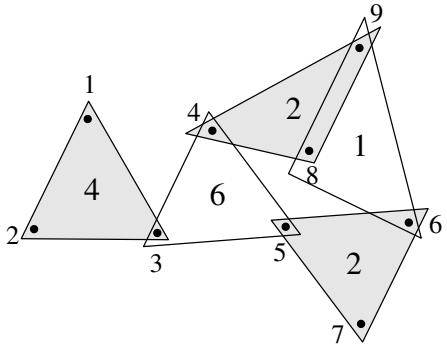
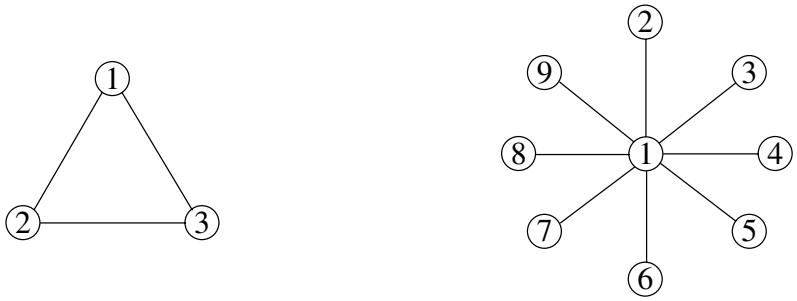


Figure 6.1: Packing triangles. Given a set $V = \{1, \ldots, 9\}$ of points, and a set $\mathcal{E}$ of weighted triangles (i.e., subsets of V having cardinality 3), find a subset of $\mathcal{E}$ of maximal weight sum, such that each point appears at most in one triangle. Here, the numbers inside the triangles indicate the weights, and the shadowed triangles represent the optimal solution.

6.3 Clique Inequalities

We consider in this section the simplest (and also most frequent) class of inequalities appearing in the linear description of any packing polytope: the clique inequalities. We derive some properties that "cliques of triangles" must have, and use them to prove that clique inequalities can be separated in polynomial time for our problem. (In general, the separation problem for clique inequalities is known to be $\mathcal{NP}$-complete, although it can be solved in polynomial time for the superclass of *orthonormal constraints*. For further details refer to Section 4.2.2, page 84).

The motivation for studying cliques of triangles came from a simple observation concerning the corresponding structures in the (uniform) 2-set packing problem. In that case, the set of rows of A can be associated to a set V of points, and its set of columns to a set $\mathcal{E}$ of edges between points of V.[1] A clique in the conflict graph $\mathbb{G}(A)$ corresponds in this setting to a set of edges from G, any two of which share a common point. (Repeated columns can be trivially dropped from A). A moment of thought reveals that there are only two possible classes of such structures, which are shown in Figure 6.2: the complete graph K_3 on three points, and the stars.



(a) (b)

Figure 6.2: Cliques of edges. The only possible structures corresponding to sets of pairwise incident edges are (a) the K_3 and (b) the stars.

Cliques associated to K_3 structures are by definition limited in their size, as they contain exactly three edges. Therefore, it is possible to solve the separation problem for their corresponding inequalities in polynomial time by enumeration of all 3-subsets of V. On the other hand, cliques arising from stars may be arbitrarily large in size, but the associated inequalities are already present in the linear description of the fractional packing polytope as point constraints: since there is a point $v \in V$ (the center of the star) contained in all edges of the clique, the row of A associated to v will contain an entry equal to one for each column corresponding to an edge of the clique.

[1] For the sake of clarity, we avoid calling the elements of V "nodes", and reserve this term exclusively for the conflict graph $\mathbb{G}(A)$ of A.

Unfortunately, this very simple structure (cliques either limited in size or associated to point inequalities) is no longer present when we turn to the problem of packing triangles. Figure 6.3 shows a counterexample. Suppose $V = \{1, 2, 3\} \cup A \cup B \cup C$, with $A = \{a_1, \ldots, a_{n_1}\}$, $B = \{b_1, \ldots, b_{n_2}\}$ and $C = \{c_1, \ldots, c_{n_3}\}$. Define $\mathcal{Q}$ as the union of the set of triangles $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{E}_3$, where

$$\begin{aligned}
\mathcal{E}_1 &= \{\{1, 2, a_i\} \,:\, 1 \leq i \leq n_1\} \\
\mathcal{E}_2 &= \{\{2, 3, b_i\} \,:\, 1 \leq i \leq n_2\} \\
\mathcal{E}_3 &= \{\{1, 3, c_i\} \,:\, 1 \leq i \leq n_3\}\,.
\end{aligned}$$

$\mathcal{Q}$ is a clique, since any pair of triangles shares at least one common point from the set $\{1, 2, 3\}$. Associated to this clique is the inequality

$$\sum_{E \in \mathcal{Q}} x_E \leq 1. \tag{6.1}$$

Now, choose one triangle from each of the sets $\mathcal{E}_1$, $\mathcal{E}_2$, and $\mathcal{E}_3$, and assign to them a value of $\frac{1}{2}$. Assign a value of 0 to all other triangles in $\mathcal{Q}$. One can easily check that this solution satisfies all point inequalities, while violating (6.1) since $\sum_{E \in \mathcal{Q}} x_E = \frac{3}{2} > 1$. Using this construction, it is possible to obtain cliques of arbitrarily large sizes whose associated inequalities are not contained among the point inequalities.

However, cliques belonging to this class still have a well-defined specific structure that makes it possible to recognize (and separate) them efficiently. Observe that there is a set $S = \{1, 2, 3\}$ of three points with the property that all triangles from $\mathcal{Q}$ contain two points of S. On the other hand, to intersect all triangles in $\mathcal{Q}$, any other triangle must contain at least two elements from S. Thus, by looking at all 3-subsets of the point set V, it is then possible to enumerate all such cliques in $O(m^3 n)$.

The main result of this section is that all cliques of triangles either belong to the class presented above, or are related to inequalities "almost contained" within the point inequalities. More precisely, we are going to prove the following theorem:

6.3.1 Theorem (Cliques of Triangles) *Let $\mathcal{Q} \subseteq \mathcal{E}$ be a maximal clique of triangles. Then one of the two following statements hold:*

(i) There exists a set $S \subseteq V$, with $|S| = 3$ such that

$$\mathcal{Q} = \{E \in \mathcal{E} \,:\, |S \cap E| \geq 2\}\,,$$

(ii) there exists a point $v \in V$ with the property

$$|\{E \in \mathcal{Q} \,:\, v \notin E\}| \leq 5.$$

Observe that if (ii) holds, then the clique inequality associated to $\mathcal{Q}$ can be obtained from the point inequality of v by extending it to include *at most* five additional variables. Hence, by trying out all possible combinations, these inequalities can be enumerated in $O(mn^5)$. It follows as an immediate corollary:

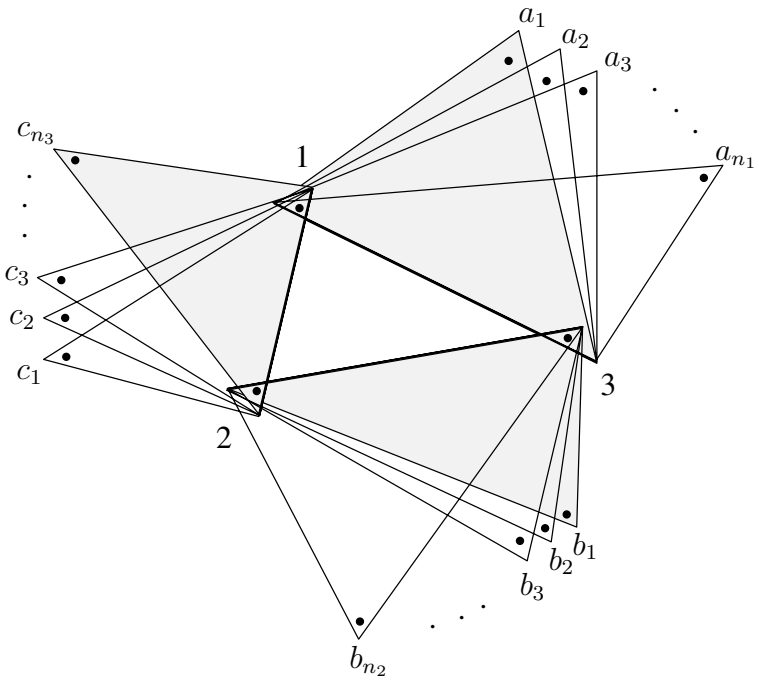


Figure 6.3: A class of arbitrarily large cliques of triangles that give rise to fractional solutions. If the variables associated with the shadowed triangles are given a value of $\frac{1}{2}$, and all other variables are set to 0, then the point inequalities are satisfied, but the sum of the values among all triangles in the clique is strictly larger than 1.

6.3.2 Corollary *Clique inequalities can be separated in polynomial time for the* 3*-set packing problem.*

Before proving Theorem 6.3.1, we need to introduce a few definitions and lemmas. Throughout the rest of this section, we assume w.l.o.g. that the triangle $E_0 := \{1, 2, 3\}$ belongs to the clique $\mathcal{Q}$ and we classify all other triangles of $\mathcal{Q}$ into six classes:

$$\mathcal{Q}_i := \{E \in \mathcal{Q} : E \cap E_0 = \{i\}\}, \ \forall i \in \{1, 2, 3\},$$
$$\mathcal{Q}_{\{i,j\}} := \{E \in \mathcal{Q} : E \cap E_0 = \{i, j\}\}, \ \forall i, j \in \{1, 2, 3\}, \ i \neq j.$$

To simplify the notation, we shall from now on refer to the class $\mathcal{Q}_{\{i,j\}}$ simply as either $\mathcal{Q}_{ij}$ or $\mathcal{Q}_{ji}$, where the order of the indexes is not important. Since any triangle of the clique must intersect E_0 either in one or in two points,

$$\mathcal{Q} = E_0 \uplus \mathcal{Q}_1 \uplus \mathcal{Q}_2 \uplus \mathcal{Q}_3 \uplus \mathcal{Q}_{12} \uplus \mathcal{Q}_{13} \uplus \mathcal{Q}_{23}.$$

We call the first three classes *single* classes, to reflect the fact that they intersect E_0 in only one point. Similarly, we shall refer to the other three classes as *double*

classes. A single class $\mathcal{Q}_i$ and a double class $\mathcal{Q}_{jk}$ are said to be *opposite* to each other if $i \notin \{j,k\}$. Otherwise, the classes are said to be *adjacent*.

A first immediate observation is the following:

6.3.3 Lemma *For any pair of opposite classes $\mathcal{Q}_i$ and $\mathcal{Q}_{jk}$, with $i,j,k \in \{1,2,3\}$, the following holds:*

(i) If $|\mathcal{Q}_{jk}| = 1$ then there is a point $v \in V$ such that

$$\bigcap_{E \in \mathcal{Q}_i \cup \mathcal{Q}_{jk}} E = \{v\}\,.$$

(ii) If $|\mathcal{Q}_{jk}| = 2$ then $|\mathcal{Q}_i| \leq 1$.

(iii) If $|\mathcal{Q}_{jk}| \geq 3$ then $\mathcal{Q}_i = \emptyset$.

Proof. For the first case, suppose $\mathcal{Q}_{jk} = \{\{j,k,v\}\}$. Since every triangle of $\mathcal{Q}_i$ has to intersect the triangle in $\mathcal{Q}_{jk}$, and since by definition no triangle in $\mathcal{Q}_i$ contains either j or k, then all these triangles *must* contain v.

On the other hand, if $\mathcal{Q}_{jk} = \{\{j,k,v_0\},\{j,k,v_1\}\}$, then all triangles in $\mathcal{Q}_i$ must contain both v_0 and v_1 in order to intersect the two triangles in $\mathcal{Q}_{jk}$. But this leaves us only with one possible candidate, namely $\{i,v_0,v_1\}$. Therefore, $|\mathcal{Q}_i| \leq 1$ in this case.

Finally, a similar argument shows that if $\mathcal{Q}_{jk}$ contains three or more triangles, it is not possible for a triangle in $\mathcal{Q}_i$ to intersect all of them simultaneously. Hence, $\mathcal{Q}_i = \emptyset$ must hold in this case. □

The following lemma can be seen as an extension of (i) to include not only a pair of opposite classes, but also an additional single class.

6.3.4 Lemma *Let $\mathcal{Q}_i$ and $\mathcal{Q}_{jk}$ be a pair of opposite classes, with $i,j,k \in \{1,2,3\}$, and let $l \in \{j,k\}$. If $|\mathcal{Q}_i| \geq 3$, $|\mathcal{Q}_l| \geq 2$ and $|\mathcal{Q}_{jk}| > 0$, then there exists a point $v \in V$ with the property*

$$\bigcap_{E \in \mathcal{Q}'} E = \{v\}$$

where $\mathcal{Q}' := \mathcal{Q}_i \cup \mathcal{Q}_l \cup \mathcal{Q}_{jk} \cup \mathcal{Q}_{rs}$, and $\mathcal{Q}_{rs}$ is the double class opposite to $\mathcal{Q}_l$.

Proof. Observe that, since the opposite classes $\mathcal{Q}_i$ and $\mathcal{Q}_{jk}$ satisfy $|\mathcal{Q}_i| \geq 3$ and $|\mathcal{Q}_{jk}| > 0$, Lemma 6.3.3 (i) implies that $|\mathcal{Q}_{jk}| = 1$, and that there exists a *unique* point $v \in V$ contained in all triangles from both classes. Hence,

$$\{\{i,v,x_1\},\{i,v,x_2\},\{i,v,x_3\}\} \subseteq \mathcal{Q}_i$$

where x_1, x_2, x_3 are three different points. Now, to intersect all these triangles, any triangle in $\mathcal{Q}_l$ must also contain v. (Again, remark that by construction $l \notin \{v,x_1,x_2,x_3\}$) Since $|\mathcal{Q}_l| \geq 2$, we have

$$\{\{l,v,y_1\},\{l,v,y_2\}\} \subseteq \mathcal{Q}_l$$

with $y_1, y_2 \in V$, $y_1 \neq y_2$.

Finally, Lemma 6.3.3 also implies that the opposite double class $\mathcal{Q}_{rs}$ of $\mathcal{Q}_l$ is either empty or contains only one triangle. In the latter case, the only candidate that intersects $\{l, v, y_1\}$ and $\{l, v, y_2\}$ simultaneously is $\{r, s, v\}$. Thus, v is contained in any triangle from $\mathcal{Q}_{rs}$, and the statement follows. □

Next, we prove an analogous result that takes into account all three single classes and one double class.

6.3.5 Lemma *Let $\mathcal{Q}_i$ and $\mathcal{Q}_{jk}$ be a pair of opposite classes, with $i, j, k \in \{1, 2, 3\}$. If $|\mathcal{Q}_i| \geq 2$, $|\mathcal{Q}_j| \geq 1$, $|\mathcal{Q}_k| \geq 1$ and $|\mathcal{Q}_{jk}| > 0$, then there exists a point $v \in V$ with the property*

$$|\{E \in \mathcal{Q} : v \notin E\}| \leq 5.$$

Proof. From Lemma 6.3.3 it follows that $\mathcal{Q}_{jk}$ contains exactly one triangle, say $\mathcal{Q}_{jk} = \{\{j, k, v\}\}$. Moreover, v must be contained in all triangles from $\mathcal{Q}_i$ and hence,

$$\{\{i, v, x_1\}, \{i, v, x_2\}\} \subseteq \mathcal{Q}_i$$

where $x_1, x_2 \in V$, $x_1 \neq x_2$. We distinguish between two different cases.

Case I: All single classes contain at least two triangles
If this happens, Lemma 6.3.3 (i) implies that the cardinality of every double class is less or equal to one. Moreover, since every triangle in $\mathcal{Q}_j$ (resp. in $\mathcal{Q}_k$) intersects all triangles from $\mathcal{Q}_i$, there can be at most one triangle in this class that does not contain v, namely, $\{j, x_1, x_2\}$ (resp. $\{k, x_1, x_2\}$). Therefore, there are at most four triangles in the union of the six classes, and hence at most five (including E_0) triangles in $\mathcal{Q}$, which do not contain v.

Case II: At least one single class consists of only one triangle
In this case, assume w.l.o.g. that $|\mathcal{Q}_j| = 1$, and let E_1 be the unique triangle belonging to this class. Due to Lemma 6.3.3 (iii), each of the double classes $\mathcal{Q}_{ik}$ and $\mathcal{Q}_{ij}$ may contain at most two triangles. Now suppose $v \in E_1$. There can be at most one triangle in the opposite class $\mathcal{Q}_{ik}$ which does not contain v, intersecting E_1 in its other "free" point. Hence, there are at most four triangles in the six classes (one in $\mathcal{Q}_k$, two in $\mathcal{Q}_{ij}$ and one in $\mathcal{Q}_{ik}$) which do not contain v, and the claim follows.

On the other hand, assume E_1 does not contain v. Then $E_1 = \{j, x_1, x_2\}$, since it must intersect all triangles from $\mathcal{Q}_i$. At the same time, this means that $\mathcal{Q}_i$ may only contain the two triangles indicated above. Furthermore, the number of triangles in the classes $\mathcal{Q}_{ik}$ and $\mathcal{Q}_k$ is also limited due to the restriction that they have to intersect E_1. In fact,

$$\mathcal{Q}_{ik} \subseteq \{\{i, k, x_1\}, \{i, k, x_2\}\}, \text{ and,}$$
$$\mathcal{Q}_k \subseteq \{\{k, v, x_1\}, \{k, v, x_2\}, \{k, x_1, x_2\}\}.$$

If $\mathcal{Q}_k$ does not contain $\{k, x_1, x_2\}$, then there are again at most four triangles in the six classes which do not contain v: two in $\mathcal{Q}_{ik}$, one in $\mathcal{Q}_j$ and at most one in $\mathcal{Q}_{ij}$ (if $|\mathcal{Q}_k| = 1$). If, on the contrary, $\{k, x_1, x_2\} \in \mathcal{Q}_k$, then the opposite class $\mathcal{Q}_{ij}$ may contain only the triangles $\{i, j, x_1\}$ and $\{i, j, x_2\}$. In this case, either x_1 or x_2 is contained in all but at most four triangles of the six classes.

□

In the previous three lemmas, we have always made assumptions on the cardinality of a double class. The following result considers the case when only two single classes are known to have sizes bounded from below.

6.3.6 Lemma *Let $\mathcal{Q}_i$ and $\mathcal{Q}_j$ be a pair of single classes, with $i, j \in \{1, 2, 3\}$, $i \neq j$. If $|\mathcal{Q}_i| \geq 4$ and $|\mathcal{Q}_j| \geq 3$, then there exists a point v with the property*

$$\bigcap_{E \in \mathcal{Q}'} = \{v\}$$

where $\mathcal{Q}' := \mathcal{Q}_i \cup \mathcal{Q}_j \cup \mathcal{Q}_{ik} \cup \mathcal{Q}_{jk}$, and $\mathcal{Q}_{ik}$ (resp. $\mathcal{Q}_{jk}$) is the opposite class to $\mathcal{Q}_j$ (resp. $\mathcal{Q}_i$).

Proof. Consider the following four triangles from $\mathcal{Q}_i$ and three triangles from $\mathcal{Q}_j$.

$$\{\{i, x_1, y_1\}, \{i, x_2, y_2\}, \{i, x_3, y_3\}, \{i, x_4, y_4\}\} \subseteq \mathcal{Q}_i$$
$$\{\{j, r_1, s_1\}, \{j, r_2, s_2\}, \{j, r_3, s_3\}\} \subseteq \mathcal{Q}_j$$

Dropping i and j from all triangles, we can consider the resulting 2-subsets as edges from a graph on the remaining points. We group these edges into two classes, depending whether they arise from triangles belonging to $\mathcal{Q}_i$ or to $\mathcal{Q}_j$. Let us call these classes "dashed" ($\mathcal{Q}_i$) and "solid" ($\mathcal{Q}_j$). Since any triangle from $\mathcal{Q}_i$ must intersect all triangles from $\mathcal{Q}_j$ in points different than i and j, any dashed line must intersect all solid lines. (And similarly, any solid line has to intersect all dashed lines.) Notice that it is possible to have parallel edges belonging to distinct classes. Figure 6.4 shows all possible arrangements of the three solid edges (up to isomorphisms). Let us consider these five cases separately.

Case I: Three pairwise disjoint dashed edges
In this case, it is not possible to intersect all three solid edges with a single dashed edge. Thus, this configuration may not appear.

Case II: Two connected components
Figure 6.4 (b) illustrates this configuration. The only two possible ways to place a dashed edge so that it intersects all solid edges are $\{r_1, r_3\}$ and $\{r_1, s_3\}$. This contradicts the condition $|\mathcal{Q}_i| \geq 4$.

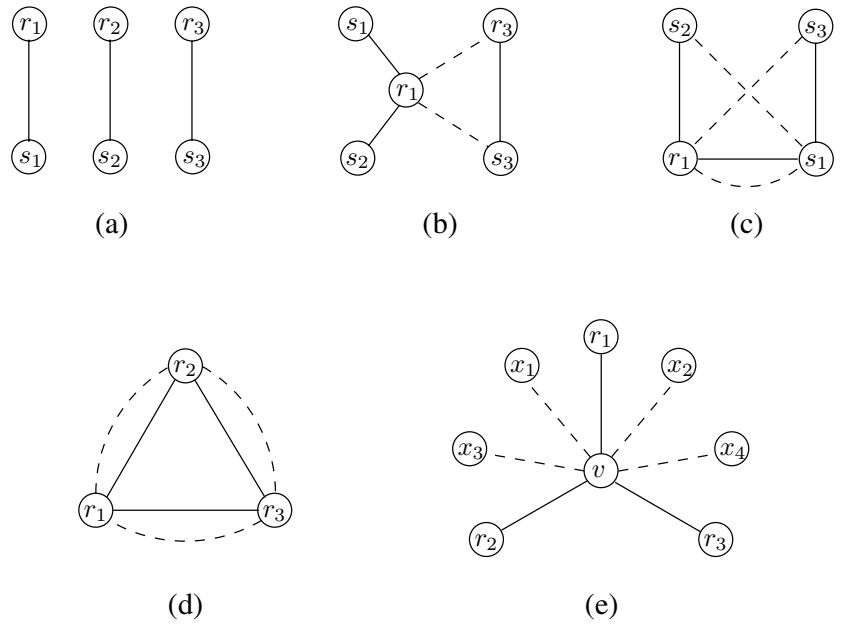


Figure 6.4: Sets of intersecting edges. The pictures show all graphs whose edges can be divided into two sets $\mathcal{E}_1$ (solid lines) and $\mathcal{E}_2$ (dashed lines), such that $|\mathcal{E}_1| = 3$, $|\mathcal{E}_2|$ is maximal and $e \cap e' \neq \emptyset$, for any $e \in \mathcal{E}_1$ and $e' \in \mathcal{E}_2$. Five cases are distinguished, according to the subgraph spanned by the edges from $\mathcal{E}_1$: (a) pairwise disjoint edges, (b) two connected components, (c) a simple path, (d) a triangle, and (e) a star. In the last case, $\mathcal{E}_2$ may be arbitrarily large.

Case III: A simple path
This configuration is shown in Figure 6.4 (c) . At most three dashed edges can be placed so that they intersect all solid edges: $\{s_1, s_2\}$, $\{r_1, s_1\}$ and $\{r_1, s_3\}$. Again, this contradicts the assumption on the cardinality of $\mathcal{Q}_i$.

Case IV: A triangle
Figure 6.4 (d) depicts this configuration. Once more, a contradiction to $|\mathcal{Q}_i| \geq 4$ is obtained, since at most three dashed edges can be placed to intersect all solid edges, namely, the same triangle edges $\{r_1, r_2\}$, $\{r_2, r_3\}$ and $\{r_1, r_3\}$.

Case V: A star
This configuration, drawn in Figure 6.4 (e), is the only one which admits four (or more) dashed edges. All edges (solid and dashed) share a common point v. Notice that, since both dashed and solid edges form stars, any further edge obtained from $\mathcal{Q}_i$ or $\mathcal{Q}_j$ in the manner described above has to contain

v. Hence,

$$\bigcap_{E \in \mathcal{Q}_i \cup \mathcal{Q}_j} E = \{v\}.$$

Moreover, a moment of thought reveals that any triangle from the opposite classes $\mathcal{Q}_{jk}$ and $\mathcal{Q}_{ik}$ must also contain this point, and the claim follows.

□

The last lemma in this sequence presents an analogous result for the case when the cardinalities of all three single classes are known to be sufficiently large.

6.3.7 Lemma *Let $\mathcal{Q}_i$, $\mathcal{Q}_j$, and $\mathcal{Q}_k$ be the three single classes, with $i, j, k \in \{1, 2, 3\}$. If $|\mathcal{Q}_i| \geq 3$, $|\mathcal{Q}_j| \geq 2$ and $|\mathcal{Q}_k| \geq 2$ then there exists a point $v \in V$ such that*

$$|\{E \in \mathcal{Q}_i \cup \mathcal{Q}_j \cup \mathcal{Q}_k \,:\, v \notin E\}| \leq 3.$$

Proof. Following an approach similar to the one presented in the proof of Lemma 6.3.6, we drop the points i, j and k from all triangles and obtain three classes of edges. Denote by S, D and O (for "Solid", "Dashed" and "dOtted") the classes obtained from $\mathcal{Q}_i$, $\mathcal{Q}_j$ and $\mathcal{Q}_k$, respectively. Hence, $|S| \geq 3$, $|D| \geq 2$, $|O| \geq 2$, and any two edges belonging to different classes must intersect at least in one point.

As in the proof of the last lemma, let $\{r_1, s_1\}$, $\{r_2, s_2\}$, and $\{r_3, s_3\}$ be three edges from S, and look at all five possible configurations for them, which are shown in Figure 6.5. As in the previous proof, the first configuration (three pairwise disjoint edges) is uninteresting, since there is no way to intersect all three solid edges with another edge. Let us discuss the other four cases separately.

Case I: Two connected components
In this case, D and O may contain at most two edges, and there is only one feasible configuration for them, namely:

$$D = O = \{\{r_1, r_3\}, \{r_1, s_3\}\}$$

At the same time, any further edge from S must contain r_1 in order to intersect the edges from D and O. Hence, r_1 is contained in all but one of the sets from $S \cup D \cup O$ and, therefore, in all but one of the triangles from $\mathcal{Q}_i \cup \mathcal{Q}_j \cup \mathcal{Q}_k$.

Case II: A triangle
This configuration is depicted in Figure 6.5 (c). Let $S_1 \subseteq S$ be the set of three solid edges shown there. Observe that, to intersect all these edges, any edge from $D \cup O$ must have its two points in $\{r_1, r_2, r_3\}$. Thus, both classes consist of either two or three edges parallel to the sides of the triangle. Now, by adding the degrees of the points, we obtain

$$d(r_1) + d(r_2) + d(r_3) \geq 2\,(|S_1| + |D| + |O|) \geq 14.$$

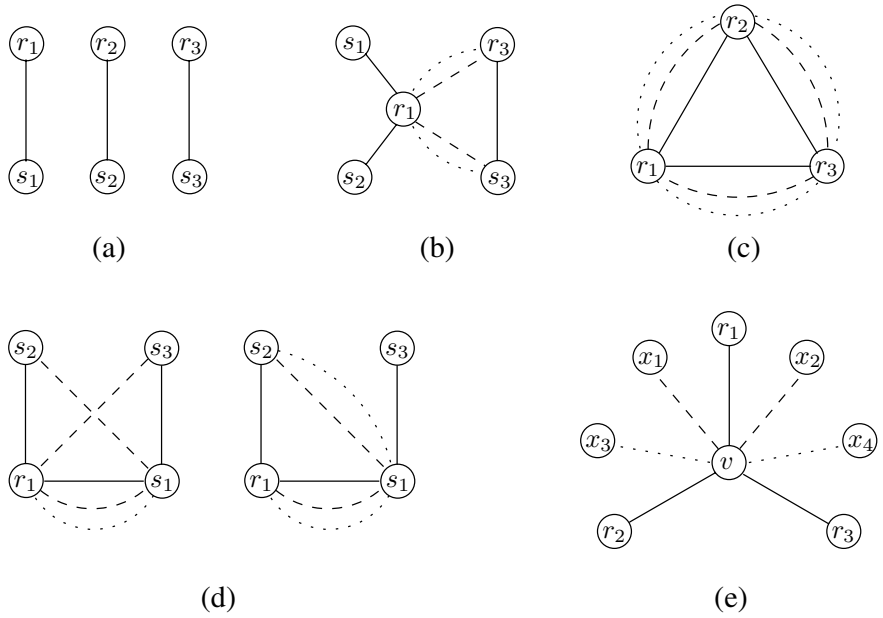


Figure 6.5: Sets of intersecting edges. The pictures show all graphs whose edges can be divided into three sets $\mathcal{E}_1$ (solid lines), $\mathcal{E}_2$ (dashed lines) and $\mathcal{E}_3$ (dotted lines), such that $|\mathcal{E}_1| = 3$, $|\mathcal{E}_2| \geq |\mathcal{E}_3|$, $\mathcal{E}_2$ and $\mathcal{E}_3$ are maximal, and $e \cap e' \neq \emptyset$ for any two edges e and e' belonging to different sets. Five main cases are distinguished, according to the subgraph spanned by the edges from $\mathcal{E}_1$: (a) pairwise disjoint edges, (b) two connected components, (c) a triangle, (d) a simple path, and (e) a star. In the last case, $\mathcal{E}_2$ and $\mathcal{E}_3$ may be arbitrarily large.

Consequently, there must be a point $v \in \{r_1, r_2, r_3\}$ such that $d(v) \geq 5$, i.e, such that v is contained in all but at most two of the edges from $S_1 \cup D \cup O$.

To complete the proof, we need to consider the remaining solid edges in $S \setminus S_1$. Notice that if this set is not empty, then there exists a point v' contained in all edges from $D \cup O$, and also in all edges from $S \setminus S_1$. But then, v' is contained in all but one of the triangles from $\mathcal{Q}_i \cup \mathcal{Q}_j \cup \mathcal{Q}_k$, and the claim follows.

Case III: A simple path
This situation is shown in Figure 6.5 (d). Since all edges in S have to be intersected, we have

$$D \subseteq \{\{s1, s2\}, \{r_1, s_1\}, \{r_1, s_3\}\}.$$

Now, suppose D contains both $\{s1, s2\}$ and $\{r_1, s_3\}$. It follows that $O \subseteq \{\{r_1, s_1\}\}$, as this is the only possibility to intersect all edges in S and D. But then $|O| < 2$, a contradiction.

On the other hand, either $\{s1, s2\}$ or $\{r_1, s_3\}$ *must* be contained in D, as $|D| \geq 2$. Assume $\{s1, s2\} \in D$. In this case, it is easy to see that $D = O$, and that both classes have cardinality equal to two. Moreover, any further edge from S has to contain s_1, and hence this point is contained in all but one of the edges from $S \cup D \cup O$.

Case IV: A star
If three solid edges form a star with point r_1 as its center, then all edges in $D \cup O$ have to contain r_1 in order to intersect them. Furthermore, apart for the one exceptional case treated below, all the other edges in S must also contain r_1, and the claim is proved.

The only exceptional case occurs when $|D| = |O| = 2$ and any edge in one class is parallel to some edge in the other class, i.e.,

$$D = O = \{\{r_1, y_1\}, \{r_1, y_2\}\}.$$

Hence S may contain the edge $\{y_1, y_2\}$. But, again, this is then the only edge in $S \cup D \cup O$ which does not contain the point r_1.

□

We are now in position to put all the pieces together and assemble a proof for the main theorem from this section.

Proof (of Theorem 6.3.1). Let $E_0 \in \mathcal{Q}$ and assume w.l.o.g. that $E_0 = \{1, 2, 3\}$. Moreover, consider the partition of $\mathcal{Q}$ into the seven classes defined on page 125. We distinguish four cases, according to the cardinalities of the double classes.

Case I: All double classes contain three or more triangles
In this case, from Lemma 6.3.3 (iii) it follows that all single classes must be empty. Therefore, all triangles of the clique contain at least two points in $\{1, 2, 3\}$. Conversely, any triangle $E \in \mathcal{E}$ having two points in this set intersects all triangles in $\mathcal{Q}$ and must therefore belong to the clique, as $\mathcal{Q}$ was assumed to be maximal. Thus, (i) is proved.

Case II: Exactly two double classes contain three or more triangles
Suppose w.l.o.g. these two classes are $\mathcal{Q}_{12}$ and $\mathcal{Q}_{13}$. Then their opposite classes $\mathcal{Q}_3$ and $\mathcal{Q}_2$ must be empty, according to Lemma 6.3.3 (iii). Hence, the only triangles which possibly do not contain the point 1 are the ones belonging to class $\mathcal{Q}_{23}$. Since $|\mathcal{Q}_{23}| \leq 2$, the claim (ii) follows.

Case III: Exactly one double class contains three or more triangles
Again, assume w.l.o.g that this class is $\mathcal{Q}_{12}$. It follows from Lemma 6.3.3 (iii) that $\mathcal{Q}_3 = \emptyset$. If one of the other two single classes, say $\mathcal{Q}_1$, contains no more than two triangles, then there is a point contained in all but at most 4 triangles from $\mathcal{Q}$ (in this case point 2), and claim (ii) follows. Thus, we

may suppose that both single classes $\mathcal{Q}_1$ and $\mathcal{Q}_2$ contain each three or more triangles.

Now assume at least one of the other double classes is not empty, say for example $\mathcal{Q}_{13}$. From Lemma 6.3.4 it then follows that there is a point v which is contained in all triangles from $\mathcal{Q}_1 \cup \mathcal{Q}_2 \cup \mathcal{Q}_{13} \cup \mathcal{Q}_{23}$. Observe that $v \notin \{1, 2, 3\}$, as it is contained in opposite classes. Hence, all triangles of the clique contain at least two points from $\{1, 2, v\}$, and claim (i) is proved.

Finally, if $|\mathcal{Q}_1| \geq 3$, $|\mathcal{Q}_2| \geq 3$ and $|\mathcal{Q}_{13}| = |\mathcal{Q}_{23}| = 0$, we consider two possible situations. If one (or both) of the single classes contains at least four triangles, applying Lemma 6.3.6 we find a $v \in V$ which is contained in all triangles from $\mathcal{Q}_1 \cup \mathcal{Q}_2$ and prove claim (i). On the other hand, if both $\mathcal{Q}_1$ and $\mathcal{Q}_2$ contain exactly three triangles, then there are only three triangles in $\mathcal{Q}$ which do not contain the point 1 (or the point 2), and claim (ii) follows.

Case IV: All double classes contain at most two triangles
For this last situation, let us consider three different subcases, according to the cardinalities of the single classes:

(a) **Two (or more) single classes contain at most one triangle.** Suppose these classes are $\mathcal{Q}_1$ and $\mathcal{Q}_2$. Then, $|\mathcal{Q}_1|+|\mathcal{Q}_2|+|\mathcal{Q}_{12}| \leq 1+1+2 = 4$, and at most four triangles do not contain the point 3, which proves (ii).

(b) **Exactly one single class contains at most one triangle.** Assume w.l.o.g that this class is $\mathcal{Q}_1$. If any of its adjacent double classes, say $\mathcal{Q}_{12}$, is not empty, then from Lemma 6.3.3 it follows that $|\mathcal{Q}_{12}| = 1$, as the opposite class $\mathcal{Q}_3$ contains at least two triangles. Say $\mathcal{Q}_{12} = \{\{1, 2, 4\}\}$ and observe that point 4 is contained in all of the triangles from $\mathcal{Q}_3$, and all but (maybe) one of the triangles from $\mathcal{Q}_2$. Now, either $|\mathcal{Q}_1| = 0$, in which case there are at most $|\mathcal{Q}_{13}|+1+|\mathcal{Q}_{23}|+1 \leq 5$ triangles (including E_0) that do not contain 4, or $|\mathcal{Q}_1| = 1$, and applying Lemma 6.3.5 we find a point $v \in V$ contained in all but at most five of the triangles from $\mathcal{Q}$. In both cases, claim (ii) is proved.

On the other hand, if both adjacent double classes $\mathcal{Q}_{12}$ and $\mathcal{Q}_{13}$ are empty, we look at the cardinalities of the single classes $\mathcal{Q}_2$ and $\mathcal{Q}_3$. If at least one of those, for instance $\mathcal{Q}_2$, contains no more than three triangles, then there is a point, in this case 3, which is contained in all but at most $|\mathcal{Q}_1| + |\mathcal{Q}_2| + |\mathcal{Q}_{12}| \leq 1 + 3 + 0 = 4$ triangles from the clique. If, on the contrary, $|\mathcal{Q}_2| \geq 4$ and $|\mathcal{Q}_3| \geq 4$, Lemma 6.3.6 shows that there is a point $v \in V$ contained in all triangles from $\mathcal{Q}_2 \cup \mathcal{Q}_3 \cup \mathcal{Q}_{12} \cup \mathcal{Q}_{13}$. Thus, v is contained in all but at most $|\mathcal{Q}_1| + |\mathcal{Q}_{23}| + 1 \leq 1 + 2 + 1 = 4$ triangles from the clique (including E_0), and claim (ii) follows.

(c) **All single classes contain two or more triangles.** In this configuration, if at least one of the double classes is not empty, then applying

Lemma 6.3.5 we get a point $v \in V$ which is contained in all but at most five triangles from $\mathcal{Q}$.

On the other hand, suppose $|\mathcal{Q}_{12}| = |\mathcal{Q}_{13}| = |\mathcal{Q}_{23}| = 0$. If there is a single class containing at least three triangles, Lemma 6.3.7 again guarantees the existence of a point v contained in all but at most four of the triangles (including E_0) from $\mathcal{Q}$. Otherwise, if $|\mathcal{Q}_1| = |\mathcal{Q}_2| = |\mathcal{Q}_3| = 2$, then there are only four triangles (those in $|\mathcal{Q}_2| + |\mathcal{Q}_3|$) which do not contain the point 1. In all cases, claim (ii) holds.

□

Before concluding this section, let us state a final remark. Theorem 6.3.1 is best possible in the sense that the right-hand side from inequality (ii) cannot be improved to a lower value. To see this, let $V = \{1, \ldots, 6\}$ and consider the following set $\mathcal{Q}$ consisting of 10 triangles in V:

$$\mathcal{Q} = \{\{1,2,3\}, \{2,3,4\}, \{1,4,5\}, \{1,4,6\}, \{2,5,6\}, \\ \{2,4,6\}, \{3,5,6\}, \{3,4,5\}, \{1,3,6\}, \{1,2,5\}\}$$

The reader can easily check that $\mathcal{Q}$ is a clique. Moreover, notice that each point appears exactly in $|\mathcal{Q}| - 5 = 5$ triangles from $\mathcal{Q}$, which means that (ii) holds with equality. At the same time, there is no pair of points appearing in all triangles, and hence, claim (i) does not hold for $\mathcal{Q}$.

6.4 About the Fractional Vertices Associated to Cliques

In the last section, we have shown that clique inequalities can be separated in polynomial time for the 3-set packing problem. Now we turn our attention to another issue regarding cliques of triangles. Namely, if $\mathcal{Q}$ is such a clique and A is the matrix of point-triangle incidences of $\mathcal{Q}$, we shall see that the nonintegral vertices of the fractional packing polytope $P(A)$ associated to A can be completely characterized.

The idea of looking for such a characterization is once again motivated by similar observations made for the cliques of edges presented in Figure 6.2 on page 123. As pointed out, there are only two possible structures associated with such cliques: the K_3 and the star. From these, only the former gives rise to a nonintegral vertex in $P(A)$, namely, the one obtained by setting all variables to $\frac{1}{2}$. On the other hand, the example in Figure 6.3 (on page 125) indicates that the situation is not so simple for cliques of triangles: there can be, for instance, arbitrarily large cliques which are not associated to a point inequality and, therefore, give rise to nonintegral vertices in $P(A)$.

Nevertheless, the last example still retains a simple structure: no matter how large such a clique $\mathcal{Q}$ is, all nonintegral vertices from $P(A)$ contain at most three nonzero coordinates. To see why, observe that only the inequalities corresponding to the points 1, 2 and 3 may be satisfied with equality by a fractional vertex

$x^* \in \mathbb{Q}^n$. Hence, at least $|\mathcal{Q}| - 3$ from the nonnegativity inequalities must also be satisfied by x^* with equality (as there must be a full-rank basis associated to this vertex), and thus at most 3 of its components may be different from 0.

In fact, even more is true: if x^* satisfies two point inequalities together with $|\mathcal{Q}| - 2$ nonnegativity inequalities with equality, one can check that the submatrix containing all these restrictions is not of full column rank. Therefore, the vertex needs to have *exactly* three fractional coordinates, and satisfy with equality all three inequalities corresponding to the points 1, 2 and 3. Moreover, the only way for doing so is assigning the value of $\frac{1}{2}$ to one triangle in each of the classes $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{E}_3$. (One can consider this structure in some sense as a generalization of the K_3 clique.)

As we shall see, the last example is not an exception. To the contrary, something similar holds in general for any clique of triangles. The work in this section consists of two parts. In a first step, we prove that the nonintegral vertices of fractional packing polytopes associated to cliques can have at most 7 coordinates different from zero. After that, we present a complete description, obtained via computer enumeration, of all the vertices having up to 7 fractional coordinates.

We start again with some preliminary definitions and lemmas. Observe that requiring any two triangles of a clique to intersect at least in one point is equivalent, in terms of the corresponding point-triangle incidence matrix A, to the condition:

$$\text{supp}(A_{\cdot j_1}) \cap \text{supp}(A_{\cdot j_2}) \neq \emptyset \tag{6.2}$$

for any two columns $A_{\cdot j_1}$ and $A_{\cdot j_2}$ of A.

6.4.1 Definition (Pseudo-basis of a Vertex) *Let $A = (a_{ij})$ be the point-triangle incidence matrix associated to a clique of triangles, where $1 \leq i \leq n$ and $1 \leq j \leq m$, . Let $x^* \in \mathbb{Q}^n$ be a vertex of the fractional packing polytope $P(A)$, and consider the following sets of column and row indices of A:*

$$\begin{aligned}
J &= \left\{ j \in [n] \,:\, x^*_j > 0 \right\}, \\
\tilde{I} &= \left\{ i \in [m] \,:\, \forall \ell \in [m],\ \ell \neq i,\ \textit{supp}(A_{i\cdot}) \cap J \not\subset \textit{supp}(A_{\ell\cdot}) \cap J \right\}, \\
I &= \left\{ i \in \tilde{I} \,:\, \forall \ell \in \tilde{I},\ \ell > i,\ \textit{supp}(A_{i\cdot}) \cap J \neq \textit{supp}(A_{\ell\cdot}) \cap J \right\},
\end{aligned}$$

where $[k]$ denotes the set $\{1, \ldots, k\}$.

The submatrix $A^* = (a_{ij})$, with $i \in I$ and $j \in J$ will be called the pseudo-basis *associated to x^*.*

In plain words, the pseudo-basis A^* associated to a vertex x^* is the submatrix obtained from A by dropping all columns that correspond to variables equal to zero, and then dropping from this new matrix any row whose support is contained in the support of another row. Duplicated rows are also removed. Pseudo-bases satisfy some elementary properties that we shall describe at next, as they will used later to prove the main result of this section.

6.4.2 Lemma *Let x^* be a nonintegral vertex of the fractional packing polytope $P(A)$ of a clique of triangles, and let A^* be the pseudo-basis associated to x^*. Then the following holds:*

(i) A^* *has full column rank,*

(ii) $\left|\mathit{supp}(A^*_{\cdot j})\right| \leq 3,\ \forall j \in J,$

(iii) $|\mathit{supp}(A^*_{i\cdot})| < |J|\,,\ \forall i \in I,$

(iv) $|\mathit{supp}(A^*_{i\cdot})| \geq 2,\ \forall i \in I,$

(v) $\left|\mathit{supp}(A^*_{\cdot j_1}) \cap \mathit{supp}(A^*_{\cdot j_2})\right| \geq 1,\ \forall j_1, j_2 \in J,$

where the index sets I and J are defined in 6.4.1.

Proof. To prove (i), remark at first that for any row $A_{i\cdot}$ of A corresponding to a inequality satisfied by x^* with equality, either $i \in I$ or $\hat{\imath} \in I$ must hold, where $A_{\hat{\imath}\cdot}$ is another row such that $\text{supp}(A_{\hat{\imath}\cdot}) \cap J = \text{supp}(A_{i\cdot}) \cap J$. If this is not the case, then by construction there exists a third row $A_{\ell\cdot}$ such that:

$$\text{supp}(A_{i\cdot}) \cap J \ \subset\ \text{supp}(A_{\ell\cdot}) \cap J,$$

and hence $A_{\ell\cdot}x^* > A_{i\cdot}x^* = 1$, as all variables with indices in J have values strictly larger than zero. But now, x^* violates the inequality associated to $A_{\ell\cdot}$, and the contradiction $x^* \notin P(A)$ follows. Therefore, A^* contains as a submatrix a basis of x^* in the linear programming sense, from which we conclude that A^* has full column rank.

The second property follows directly from the fact that A^* is a submatrix of a point-triangle incidence matrix A. For (iii), suppose there is a row containing $|J|$ ones (i.e., a row full of ones). In this case, $|I| = 1$ must hold by construction. But then, as A^* is of full column rank, it follows that $|J| = 1$, and thus x^* has only one nonzero coordinate, which in addition must be equal to one. This contradicts the assumption that x^* was a nonintegral vertex.

To prove (v), let $j, \hat{\jmath}$ be two indices in J and notice that (6.2) implies the existence of at least one row $A_{i\cdot}$ of A whose support contains both j and $\hat{\jmath}$. Moreover, either $i \in I$ or $\hat{\imath} \in I$ holds, where $A_{\hat{\imath}\cdot}$ is a row whose support contains (or is equal to) $\text{supp}(A_{i\cdot})$. In any case, j and $\hat{\jmath}$ are included in the support of some row of A^*, and the claim follows.

Statement (iv) says that every row of A^* contains at least two entries equal to one. Assume this is not true and let $A^*_{i\cdot}$ be a row with $\text{supp}(A^*_{i\cdot}) = \{j\}$. From the definition of I, it follows that no other row in A^* may contain j in its support. Now, either $|J| = 1$ and x^* is integral, or the column $A^*_{\cdot j}$ violates (v). □

The properties presented above can be used to limit the maximal number of columns a pseudo-basis may have, as stated in the following. Observe that this is at the same time the maximal number of nonzero coordinates in a vertex of the fractional packing polytope related to a clique.

6.4.3 Theorem (Maximal Number of Columns in a Pseudo-basis) *Let A^* be the pseudo-basis associated to a nonintegral vertex of the fractional packing polytope from a clique of triangles. Let J be the set of column indices of A^*, as defined in 6.4.1. Then,*

$$|J| \leq 7.$$

Proof. Suppose A^* is a pseudo-basis for which $J \geq 8$ holds. Interchanging rows and columns (as none of these operations affects the properties indicated in Lemma 6.4.2), we may assume w.l.o.g. that A^* has the following form:

$$A^* = \begin{array}{c} \begin{array}{cccccccc} j_1 & j_2 & \cdots & j_{n_1} & k_1 & k_2 & \cdots & k_{n_2} \\ \downarrow & \downarrow & & \downarrow & \downarrow & \downarrow & & \downarrow \end{array} \\ \left(\begin{array}{cccc|ccc} 1 & 1 & \cdots & 1 & 0 & 0 & \cdots & 0 \\ \hline & & & & & & & \\ & A^*_{11} & & & & A^*_{12} & & \\ & & & & & & & \\ \hline & & & & & & & \\ & O & & & & A^*_{22} & & \\ & & & & & & & \end{array}\right) \end{array} \begin{array}{l} \leftarrow i_0 \\ \leftarrow i_1 \\ \vdots \\ \leftarrow i_{m_1} \\ \leftarrow \ell_1 \\ \vdots \\ \leftarrow \ell_{m_2} \end{array}$$

where the first row $A^*_{i_0\cdot}$ satisfies

$$\left|\text{supp}(A^*_{i_0\cdot})\right| \geq \left|\text{supp}(A^*_{i\cdot})\right|, \forall i \in I. \tag{6.3}$$

The columns of A^* are divided into two groups, indexed by the sets $J_1 = \{j_1, \ldots, j_{n_1}\}$ and $J_2 = \{k_1, \ldots, k_{n_2}\}$. The columns in the first group are exactly those which contain a 1 in the first row. Observe that $J_2 \neq \emptyset$, since otherwise the first row violates condition 6.4.2 (iii). The rows are also distributed into three groups: $A^*_{i_0\cdot}$ as a group itself, and two other groups with index sets $I_1 = \{i_1, \ldots, i_{m_1}\}$ and $I_2 = \{\ell_1, \ldots, \ell_{m_2}\}$, with the rows in the last group being exactly those which have entries equal to zero for all columns in J_1. For simplicity in the notation, we shall refer to the four sets described here either as sets of indexes or sets of columns. We shall distinguish between two different cases, depending whether I_2 is empty or not.

Case I: $\mathbf{m_2 > 0}$
In this case, there is at least one row $A^*_{\ell_1\cdot}$ with its support completely contained in the set J_2. From Lemma 6.4.2 (iv), it then follows that there are (at least) two columns $A^*_{\cdot k_1}$ and $A^*_{\cdot k_2}$ with $k_1, k_2 \in J_2$ whose supports contain ℓ_1. Now observe that, for $t \in J_1 \cup \{k_1, k_2\}$,

$$|\text{supp}(A^*_{\cdot t}) \cap I_1| \leq 2,$$

since each column has at most three nonzero entries, all columns from J_1 have already a one in $A^*_{i_0\cdot}$, and the two columns in J_2 have a one in $A^*_{\ell_1\cdot}$. Moreover, property 6.4.2 (v) implies that, for any $j \in J_1$, $k \in \{k_1, k_2\}$

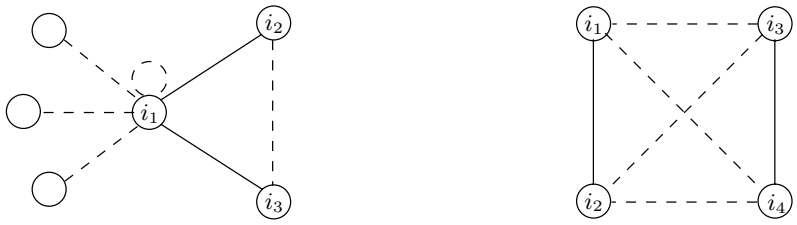
$$\text{supp}(A^*_{\cdot j}) \cap \text{supp}(A^*_{\cdot k}) \cap I_1 \neq \emptyset,$$

as I_1 contains all the rows where these two columns may intersect each other. Therefore, if either $\text{supp}(A^*_{\cdot k_1})$ or $\text{supp}(A^*_{\cdot k_2})$ contains a single row $i \in I_1$, then i must also be contained in the support of all columns from J_1, and we obtain

$$|\text{supp}(A^*_{i\cdot})| \geq n_1 + 1 > \left|\text{supp}(A^*_{i_0\cdot})\right|,$$

which is a contradiction to (6.3).

On the other hand, if both $A^*_{\cdot k_1}$ and $A^*_{\cdot k_2}$ contain exactly two nonzero entries in rows from I_1, we define the graph $G = (V, \mathcal{E})$, where $V = I_1$, $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2$, and $\mathcal{E}_1$ (resp. $\mathcal{E}_2$) are the edges obtained by intersecting the supports of the columns in J_1 (resp. in $\{k_1, k_2\}$) with I_1. Notice that there may be "edges" in $\mathcal{E}_1$ containing only one point, which we shall represent in the following as loops. Condition 6.4.2 (v) requires any two edges from $\mathcal{E}_1$ and $\mathcal{E}_2$ to intersect each other. Figure 6.6 illustrates the two only possible configurations, where the edges belonging to $\mathcal{E}_1$ are shown dashed, and the ones belonging to $\mathcal{E}_2$ solid. Again, the first configuration reveals the existence of a row $i_1 \in I_1$ that violates (6.3), as $\left|\text{supp}(A^*_{i_1\cdot})\right| \geq n_1 + 1$.



(a) (b)

Figure 6.6: Sets of intersecting edges. The pictures show all graphs whose edges can be divided into two sets $\mathcal{E}_1$ (dashed lines) and $\mathcal{E}_2$ (solid lines), such that $|\mathcal{E}_2| = 2$, $|\mathcal{E}_1|$ is maximal and $e \cap e' \neq \emptyset$, for any $e \in \mathcal{E}_1$ and $e' \in \mathcal{E}_2$. Two cases are distinguished, according to the subgraph spanned by the edges from $\mathcal{E}_2$: (a) a simple path, (b) two pairwise disjoint edges. In the first case, $\mathcal{E}_1$ may be arbitrarily large, but all of its edges (except maybe one) have to contain the node i_1.

Configuration (b) needs some more explanation. Observe that in this case the maximal number of edges in $\mathcal{E}_1$ (and, hence, of columns in J_1) is limited to four. (Parallel edges in $\mathcal{E}_1$ would imply duplicated columns in the full column rank matrix A^*). Moreover, if $|J_1| < 4$, it is not possible to add any more solid edges (not even parallel) without ending up with a point having a

degree larger than $|\mathcal{E}_1|$, which in turn leads to a contradiction to (6.3). Thus, $|J| = |J_1| + |J_2| < 4 + 2 = 6 < 7$ follows.

On the other hand, if $|J_1| = 4$ then at most two more solid edges can be added, while keeping the degrees of all points less or equal to 4, namely, one parallel to each of the solid edges already present. Now consider the column $A^*_{\cdot k_3}$ corresponding to one of these new edges, and suppose it is parallel to the edge obtained from $A^*_{\cdot k_1}$. Since $\text{supp}(A^*_{\cdot k_2}) \cap \text{supp}(A^*_{\cdot k_3}) \neq \emptyset$, it follows that $\ell_1 \in \text{supp}(A^*_{\cdot k_3})$. But then $\text{supp}(A^*_{\cdot k_1}) = \text{supp}(A^*_{\cdot k_3})$ and A^* has duplicated columns, which contradicts 6.4.2 (i).

Case II: $\mathbf{m_2 = 0}$
If I_2 is empty, we have $m = m_1 + 1$ and

$$\text{supp}(A^*_{i\cdot}) \cap J_1 \neq \emptyset \tag{6.4}$$

for any row $A^*_{i\cdot}$. Now consider the column $A^*_{\cdot k_1}$ and assume w.l.o.g. that $\text{supp}(A^*_{\cdot k_1}) = \{i_1, i_2, i_3\}$. Due to 6.4.2 (v), the support of any of the columns $A^*_{\cdot j}$, with $j \in J_1$, must contain at least one of these row indices, and consequently, it may contain *at most* one of the indices from $\{i_4, \ldots, i_{m_1}\}$. Together with (6.4), this implies that $m \leq n_1 + 4$. Moreover, since A^* has full column rank, it follows that

$$n_1 + n_2 \leq m \leq n_1 + 4 \quad \Leftrightarrow \quad n_2 \leq 4. \tag{6.5}$$

On the other hand, since $\text{supp}(A^*_{\cdot k}) \cap \text{supp}(A^*_{\cdot \hat{k}}) \neq \emptyset$ must hold for any $k, \hat{k} \in J_2$, and since no column has more than three nonzero entries, it follows that

$$m^* := \left| \bigcup_{k \in J_2} \text{supp}(A^*_{\cdot k}) \right| \leq 2n_2 + 1.$$

Observe that by construction, $m^* \leq m_1$, as all columns in J_2 have entries equal to zero in the first row. Now if $m^* < m_1$, then there must be at least one row (besides from $A^*_{i_0\cdot}$) whose support does not contain any indices from J_2 and is therefore included in (or equal to) $\text{supp}(A^*_{i_0\cdot})$, a contradiction to the assumption that A^* is a pseudo-basis. Hence, $m^* = m_1$ and

$$n_1 + n_2 \leq m = m_1 + 1 \leq 2n_2 + 2 \quad \Leftrightarrow \quad n_1 \leq n_2 + 2. \tag{6.6}$$

Let us consider all cases that satisfy the conditions (6.5) and (6.6). If $n_2 \leq 2$, then $n_1 \leq 4$ and $|J| = n_1 + n_2 \leq 6 < 8$. If $n_2 = 3$, the only possibility to obtain $|J| \geq 8$ is to choose $n_1 = 5$, which in turn implies $m = 8$. Since the three columns in J_2 have to cover $m_1 = m - 1 = 7$ rows, and due to Lemma 6.4.2 (v), there must be one row index i_1 contained in the supports of all three columns, and six row indices $i_2, \ldots, i_7$, each one contained exactly

in one of the supports. But then, the support of any column from J_1 has to contain the row index i_1, too. (Again, due to Lemma 6.4.2 (v)). It follows that A^* contains a row full of ones, contradicting Lemma 6.4.2 (iii).

Finally, if $n_2 = 4$, the only interesting cases arise when $n_1 \in \{4, 5, 6\}$. For $n_1 > 4$, the same argumentation as above leads to the conclusion that A^* has a row full of ones. If $n_1 = 4$, there is only one possible arrangement (up to exchangings of rows and columns) for the columns in J_1 that satisfies (6.4) and the requirement that $\text{supp}(A^*_{\cdot j}) \cap \{i_1, i_2, i_3\} \neq \emptyset$ for all $j \in J_1$, namely

$$\begin{array}{cccc} j_1 & j_2 & j_3 & j_4 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ \\ 1 & 1 & 1 & 1 \\ 1 & . & . & 1 \\ . & 1 & . & . \\ . & . & 1 & . \\ 1 & . & . & . \\ . & 1 & . & . \\ . & . & 1 & . \\ . & . & . & 1 \end{array} \begin{array}{l} \\ \\ \leftarrow i_0 \\ \leftarrow i_1 \\ \\ \\ \vdots \\ \\ \\ \leftarrow i_7 \end{array}$$

But then, to intersect these four columns, any column from J_2 must contain i_1. Thus,

$$\left|\text{supp}(A^*_{i_1 \cdot})\right| = 6 > 4 = \left|\text{supp}(A^*_{i_0 \cdot})\right|,$$

and a contradiction to (6.3) is obtained.

□

The following is an immediate consequence of Theorem 6.4.3:

6.4.4 Corollary *Let $\mathcal{Q}$ be a clique of triangles and A the corresponding point-triangle incidence matrix. Let $P(A)$ be the fractional packing polytope associated to A and x^* a vertex from $P(A)$. Then x^* has at most 7 nonzero coordinates.*

Actually, even more is true: the nonzero coordinates of fractional vertices may take only certain combinations of values. To prove this, we have enumerated all 0/1-matrices that may appear as (linear programming) bases in $P(A)$, and used them to calculate the corresponding vertices. As pointed out in the proof of Lemma 6.4.2 (i), any basis associated to a vertex $x^* \in P(A)$ is contained as a submatrix in the pseudo-basis defined by x^*. Therefore, most of the results that have been discussed in the previous pages for pseudo-bases can be trivially extended for bases. We summarize them here without proof, as they are in fact corollaries from Lemma 6.4.2 and Theorem 6.4.3.

6.4.5 Corollary *Let A be the point-triangle incidence matrix of a clique of triangles and x^* a fractional vertex from $P(A)$. Let B be a linear programming basis associated to x^*. Then the following holds:*

(i) B *is a square nonsingular matrix of size* n*, where* $2 \leq n \leq 7$*,*

(ii) $\left|supp(A^*_{\cdot j})\right| \leq 3$*,* $\forall j \in J$*,*

(iii) $|supp(A^*_{i\cdot})| < |J|$*,* $\forall i \in I$*,*

(iv) $|supp(A^*_{i\cdot})| \geq 2$*,* $\forall i \in I$*,*

where I *and* J *are the sets of rows and column indices of* B*.*

Observe that 6.4.2 (v) does not necessarily hold for a basis. However, it must be possible to extend any basis to a pseudo-basis by adding new rows until the supports of any two columns intersect each other, while keeping all the other properties.

Algorithm 6.1 shows our enumeration scheme. (Recall that $[n]$ is used to denote the set $\{1, \ldots, n\}$). At each main iteration, matrices of a fixed dimension $n \in \{2, \ldots, 7\}$ are considered. With this purpose, all $0/1$-vectors of dimension n containing up to three nonzero components are enumerated and stored into an array $\mathcal{S}$. At next, we enumerate all $n \times n$ matrices that can be formed by taking n-subsets from $\mathcal{S}$ as columns. Observe that, by construction, any such matrix B satisfies Lemma 6.4.5 (ii). We then check if B also satisfies the other properties. If so, we solve the linear system $Bx = \mathbf{1}$ to obtain the vertex candidate x^* and check if $0 < x^*_i < 1$ holds for all coordinates of x^*. If any of the tests fail, we discard B and continue with the next matrix.

Otherwise, B is a feasible basis associated to a clique if and only if it can be extended to a pseudo-basis. However, the computational effort for calculating such extensions is too high, and so we rather test two relaxed conditions and later examine the obtained matrices by hand. At first, we require the sum of any two coordinates x^*_i and x^*_j of x^* to be less or equal to 1, since otherwise the inequality associated to any row containing both i and j in its support would be violated. Then, we check if all columns having supports of cardinality equal to three intersect each other, since this cannot be changed by the addition of new rows.

If all tests are successfully completed, B is stored in the solution set $\mathcal{B}$, and the next matrix is considered. To keep $\mathcal{B}$ at a manageable size, we need to avoid the insertion of "redundant" matrices, i.e., matrices that can be obtained from other matrices in $\mathcal{B}$ by row and column exchanges. (The reader can compute how many such redundant matrices there are for one matrix of size 7). This is done by putting B in what we call a *standard form*: We consider each row as a number in binary representation. Notice that the value of these numbers can be altered by exchanging columns. We do this until we find the lexicographically smaller set of values. Then we sort these numbers by exchanging rows.

The output of this enumeration algorithm consisted of 12 matrices, from which three were discarded during the posterior manual checking, as it turned out that they cannot be extended to pseudo-bases. The remaining nine matrices, together with the corresponding fractional vertices, are listed in Appendix A. As a consequence, the following result is proved.

```
    {Initialization}
    B ← ∅;
 3: {Dimension is increased in the main loop}
    for n ← 2, 3, ..., 7 do
      {Generate the set S of all feasible columns}
 6:   S ← {x ∈ {0,1}^n : |supp(x)| ≤ 3};
      N ← |S|
      I ← {A ⊆ [N] : |A| = n};
 9:   {Test all n-subsets of S}
      for I ∈ I do
        {Get a new matrix B with the columns of S indexed by I}
12:     B ← {S_i ∈ S : i ∈ I};
        if |supp(B_ℓ.)| < 2 or |supp(B_ℓ.)| = n for some ℓ ∈ [n] then
          Skip B and continue with next matrix;
15:     end if
        if B is singular then
          Skip B and continue with next matrix;
18:     end if
        Solve linear system Bx = 1;
        if x_i ≤ 0 or x_i ≥ 1 for some i ∈ [n] then
21:       Skip B and continue with next matrix;
        end if
        if x_i + x_j > 1 for some i, j ∈ [n] then
24:       Skip B and continue with next matrix;
        end if
        if |supp(B_.j1)| = 3 and |supp(B_.j2)| = 3 and
         supp(B_.j1) ∩ supp(B_.j2) = ∅ for some j1, j2 ∈ [n] then
27:       Skip B and continue with next matrix;
        end if
        Put B in standard form;
30:     if B ∉ B then
          B ← B ∪ {B};
        end if
33:   end for
    end for
    Output all matrices from B;
```

Algorithm 6.1: Enumerating bases of cliques of triangles

6.4.6 Theorem (Classes of Vertices) *Let A be the point-triangle incidence matrix of a clique of triangles, and x^* a nonintegral vertex of the corresponding fractional packing polytope $P(A)$. Define $M(x^*)$ to be the multiset formed by all nonzero coordinates of x^*. Then x^* can be assigned to one of the following nine classes of vertices $X_1, \ldots, X_9$, according to the form of $M(x^*)$:*

$$X_1 : M(x^*) = \left\{\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right\}, \qquad X_2 : M(x^*) = \left\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\},$$
$$X_3 : M(x^*) = \left\{\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right\}, \qquad X_4 : M(x^*) = \left\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{2}{3}\right\},$$
$$X_5 : M(x^*) = \left\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\}, \qquad X_6 : M(x^*) = \left\{\frac{1}{5}, \frac{1}{5}, \frac{2}{5}, \frac{2}{5}, \frac{3}{5}\right\},$$
$$X_7 : M(x^*) = \left\{\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{2}, \frac{1}{2}\right\}, \qquad X_8 : M(x^*) = \left\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\},$$
$$X_9 : M(x^*) = \left\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\}.$$

Class X_1 contains the fractional vertices that appear in cliques similar as the one showed in Figure 6.3. We will not describe in detail the structures associated to the other classes, but just make some basic observations about them. Looking at the corresponding bases, one can check that vertices belonging to the classes $X_1, \ldots, X_7$ may appear in cliques having arbitrarily large sizes: in all these matrices it is always possible to choose a pair of rows that intersects all columns, which means that there is (at least) one pair of points that intersects all triangles. Now, any triangle formed from this pair of points and a new point can be used to extend the clique.

Class X_8 is related to a clique that can also be extended, but not arbitrarily. The minimal cardinality of a set of points that intersects all triangles of the clique is 3. From these sets of three points, some correspond to triangles already present in the clique, while others reveal new ones.

The structure related to class X_9 is more interesting. It is a clique $\mathcal{Q}$ consisting of seven triangles and seven points:

$$\mathcal{Q} = \{\{1,2,3\}, \{1,4,5\}, \{1,6,7\}, \{2,4,6\}, \{2,5,7\}, \{3,4,7\}, \{3,5,6\}\}.$$

In Figure 6.7 we show a schematic picture of $\mathcal{Q}$, obtained by drawing each triangle as a line. Surprisingly, a well-known mathematical object is revealed, which has been reported to appear within many different contexts: the *Fano plane*. We shall come to this fact later in Section 6.6.1.

It can be verified that $\mathcal{Q}$ satisfies the following properties:

(i) The clique cannot be extended. To intersect all triangles, a set containing at least three points is needed. Moreover, all possible such sets are already contained as triangles in $\mathcal{Q}$.

$$\begin{pmatrix} 1 & 1 & 1 & . & . & . & . \\ 1 & . & . & 1 & 1 & . & . \\ 1 & . & . & . & . & 1 & 1 \\ . & 1 & . & 1 & . & 1 & . \\ . & 1 & . & . & 1 & . & 1 \\ . & . & 1 & 1 & . & . & 1 \\ . & . & 1 & . & 1 & 1 & . \end{pmatrix}$$

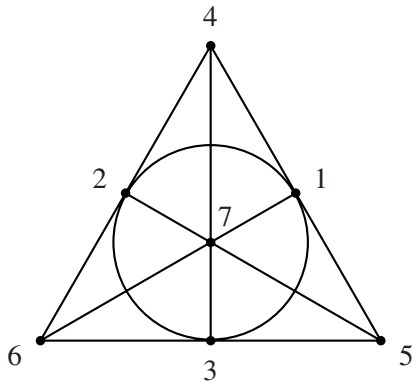


Figure 6.7: Clique of triangles associated to the vertex class X_9. The clique contains 7 triangles and 7 points, configured according to the point-triangle incidence matrix shown. Representing triangles by lines, the structure turns out to be isomorphic to the Fano plane.

(ii) The number of triangles is equal to the number of points.

(iii) The clique is "regular": Each point appears exactly in three triangles.

(iv) For any two points, there is exactly one triangle containing both of them.

(v) $\mathcal{Q}$ is "self-dual" in the sense of hypergraph duality described at next. Consider $\mathcal{Q} = (V, \mathcal{E})$ as a hypergraph. Its dual $\mathcal{Q}^*$ is defined as the new hypergraph whose set of points is $\mathcal{E}$, and whose hyperedges are all the sets of the form:

$$E' = \{E \in \mathcal{E} \,:\, v \in E\},$$

where $v \in V$. Now from the last two observations it follows that $\mathcal{Q}^*$ is also a clique of triangles. Moreover, $\mathcal{Q}^*$ is *isomorph* to $\mathcal{Q}$.

Finally, let us point out that there are some similarities between this clique and the clique of edges K_3 showed in Figure 6.2. Namely, if in the last list we replace the words "triangle" by "edge" and "three" by "two", then the same properties hold for the K_3 structure. Besides, the fractional vertex associated to a K_3 contains 3 coordinates, each one equal to $\frac{1}{2}$, while vertices of the class X_9 contain 7 coordinates, each one equal to $\frac{1}{3}$. What these numbers mean will be discussed in Section 6.6.1. Both structures are also extremal in the sense that they give raise to vertices of the fractional packing polytope having the largest possible number of nonzero coordinates.

6.5 Antiwebs and Webs

In this section, we consider two other structures in the conflict graph $\mathbb{G}(A)$ associated to a triangle packing problem that give rise to valid linear inequalities for

the packing polytope $P_I(A)$: webs and antiwebs. We refer to Section 4.2.2 for details about these inequalities, and review here just some definitions needed in the following.

An antiweb $C(n,k)$, with $2 \leq k \leq \left\lfloor \frac{n}{2} \right\rfloor$, is the conflict graph of a circulant matrix. Its set of nodes is $\{0,\ldots,n-1\}$, and two nodes i, j are connected with an edge if $(j-i \mod n) < k$ holds. A web $\bar{C}(n,k)$ is the complement of an antiweb. In particular, odd holes are the antiwebs obtained for $k=2$, and odd antiholes the corresponding webs. Figure 4.4 on page 87 shows two examples of these structures. Associated to webs and antiwebs are the following valid inequalities for the set packing polytope:

$$\sum_{i \in C(n,k)} x_i \leq \left\lfloor \frac{n}{k} \right\rfloor \qquad\qquad \sum_{i \in \bar{C}(n,k)} x_i \leq k.$$

Now remark that a circulant matrix $C(n,k)$ has exactly k entries equal to one in each row and in each column. Thus, it is obvious that odd holes and antiwebs of the form $C(n,3)$ may appear in conflict graphs stemming from triangle packing problems. However, they are not the only possible ones. Figure 6.8 shows a point-triangle incidence matrix whose conflict graph is the antiweb $C(9,4)$. In general, let A be the point-triangle incidence matrix of a family $\mathcal{E}$ of n triangles having the form

$$E_i = \{i, i-2, i-3\},$$

where $i \in \{0,\ldots,n-1\}$, and all operations are taken modulo n. One can check that, for any $E_i, E_j \in \mathcal{E}$, $E_i \cap E_j \neq \emptyset$ if and only if $|i-j| < 4 \mod n$. Hence, $\mathbb{G}(A) = C(n,4)$, which proves that any antiweb of this form may appear in the conflict graph related to a triangle packing problem. The next two results discard the existence of further structures.

$$\begin{array}{c} \begin{array}{ccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \left(\begin{array}{ccccccccc} 1 & . & 1 & 1 & . & . & . & . & . \\ . & 1 & . & 1 & 1 & . & . & . & . \\ . & . & 1 & . & 1 & 1 & . & . & . \\ . & . & . & 1 & . & 1 & 1 & . & . \\ . & . & . & . & 1 & . & 1 & 1 & . \\ . & . & . & . & . & 1 & . & 1 & 1 \\ 1 & . & . & . & . & . & 1 & . & 1 \\ 1 & 1 & . & . & . & . & . & 1 & . \\ . & 1 & 1 & . & . & . & . & . & 1 \end{array} \right) \end{array}$$

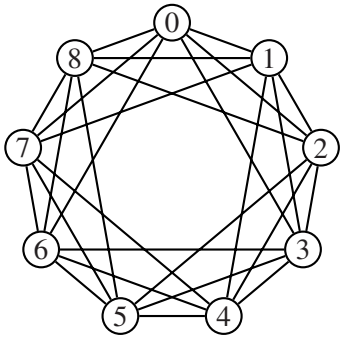


Figure 6.8: The antiweb $C(9,4)$ can be obtained as the conflict graph associated to a triangle packing problem.

6.5.1 Theorem (Antiwebs of Triangles) *Let A be the point-triangle incidence matrix associated to a triangle packing problem. The conflict graph $\mathbb{G}(A)$ of A cannot contain any antiweb of the form $C(n,5)$ as a node induced subgraph.*

Proof. Assume $A \in \{0,1\}^{m\times n}$ is a point-triangle incidence matrix for which $C(n^*,5)$, with $n^* \leq n$, appears as a subgraph from $\mathbb{G}(A)$. Let B be the submatrix of A containing the columns associated to the nodes of the antiweb and all rows where this set of columns has at least two nonzero entries. Furthermore, suppose w.l.o.g. that the columns of B have the indices $0,\ldots,n^*-1$. Observe that the supports of the rows of B correspond to cliques from the antiweb, and hence $|\text{supp}(B_{i\cdot})| \leq 5$ holds for any row $B_{i\cdot}$ of B.

We will use a counting argument to derive a contradiction. With this purpose, let us group the edges of $C(n^*,5)$ into four *types* $\Upsilon_1,\ldots,\Upsilon_4$, where $jk \in \Upsilon_\ell$ if and only if $|j-k| = \ell \mod n^*$. In the same manner, all cliques of $C(n^*,5)$ are divided into 15 *classes*, according to the relative position of their nodes on the antiweb. Thus, all cliques containing five nodes are assigned to the same class Π_5, while the cliques of size four are arranged into the following four classes:

$$\begin{aligned}
\Pi_{41} &= \{\{j,j+1,j+2,j+3\} \,:\, 0 \leq j \leq n^*-1\}\,,\\
\Pi_{42} &= \{\{j,j+1,j+2,j+4\} \,:\, 0 \leq j \leq n^*-1\}\,,\\
\Pi_{43} &= \{\{j,j+1,j+3,j+4\} \,:\, 0 \leq j \leq n^*-1\}\,,\\
\Pi_{44} &= \{\{j,j+2,j+3,j+4\} \,:\, 0 \leq j \leq n^*-1\}\,,
\end{aligned}$$

where all operations are taken modulo n^*. Analogously, six classes $\Pi_{31},\ldots,\Pi_{36}$ containing the cliques of size three and four classes $\Pi_{21},\ldots,\Pi_{24}$ with the cliques of size two (which are exactly the edge types) are defined. The number of edges of type Υ_ℓ contained in a clique from the class Π_h is denoted by $\alpha_\ell(\Pi_h)$. These values are presented in Table 6.1

Now define x_h to be the number of rows of B that correspond to cliques from the class Π_h. Since B contains at most $3n^*$ nonzero elements (at most three in each column), it follows that

$$5x_5 + 4(x_{41}+\cdots+x_{44}) + 3(x_{31}+\cdots+x_{36}) + 2(x_{21}+\cdots+x_{24}) \leq 3n^*. \quad (6.7)$$

On the other hand, notice that the number of edges of a certain type Υ_ℓ that are contained in the union of all cliques associated to the rows of B is bounded from above by

$$\sum_{h\in H} x_h \alpha_\ell(\Pi_h).$$

where $H = \{5,41,\ldots,24\}$ is the index set of all clique classes. Since there are exactly n^* different edges of each type in $C(n^*,5)$, using the information of table

Class	Nodes	α_1	α_2	α_3	α_4
Π_5	$\{j, \ldots, j+4\}$	4	3	2	1
Π_{41}	$\{j, j+1, j+2, j+3\}$	3	2	1	-
Π_{42}	$\{j, j+1, j+2, j+4\}$	2	2	1	1
Π_{43}	$\{j, j+1, j+3, j+4\}$	2	1	2	1
Π_{44}	$\{j, j+2, j+3, j+4\}$	2	2	1	1
Π_{31}	$\{j, j+1, j+2\}$	2	1	-	-
Π_{32}	$\{j, j+1, j+3\}$	1	1	1	-
Π_{33}	$\{j, j+1, j+4\}$	1	-	1	1
Π_{34}	$\{j, j+2, j+3\}$	1	1	1	-
Π_{35}	$\{j, j+2, j+4\}$	-	2	-	1
Π_{36}	$\{j, j+3, j+4\}$	1	-	1	1
Π_{21}	$\{j, j+1\}$	1	-	-	-
Π_{22}	$\{j, j+2\}$	-	1	-	-
Π_{23}	$\{j, j+3\}$	-	-	1	-
Π_{24}	$\{j, j+4\}$	-	-	-	1

Table 6.1: Clique classes of $C(n^*, 5)$. For each class Π_h, the nodes contained in any clique from Π_h are shown, as well as the number of edges from the types $\Upsilon_1, \ldots, \Upsilon_4$. In all cases, $0 \le j \le n^* - 1$, and all operations are taken modulo n^*.

6.1 we obtain the following four inequalities:

$$\begin{aligned}
\Upsilon_1: \quad & 4x_5 + 3x_{41} + 2x_{42} + 2x_{43} + 2x_{44} + 2x_{31} + x_{32} + x_{33} \\
& + x_{34} + x_{36} + x_{21} \ge n^* \\
\Upsilon_2: \quad & 3x_5 + 2x_{41} + 2x_{42} + x_{43} + 2x_{44} + x_{31} + x_{32} + x_{34} \\
& + 2x_{35} + x_{22} \ge n^* \qquad (6.8) \\
\Upsilon_3: \quad & 2x_5 + x_{41} + x_{42} + 2x_{43} + x_{44} + x_{32} + x_{33} + x_{34} + x_{36} \\
& + x_{23} \ge n^* \qquad (6.9) \\
\Upsilon_4: \quad & x_5 + x_{42} + x_{43} + x_{44} + x_{33} + x_{35} + x_{36} + x_{24} \ge n^* \qquad (6.10)
\end{aligned}$$

Finally, from the linear combination $\frac{1}{2}$(6.8) + $\frac{3}{4}$(6.9) + 2 (6.10) of these last three inequalities, and comparing the result with (6.7), we obtain

$$\begin{aligned}
\frac{13}{4}n^* &\le 5x_5 + \frac{7}{4}x_{41} + \frac{15}{4}x_{42} + 4x_{43} + \frac{15}{4}x_{44} + \frac{1}{2}x_{31} + \frac{5}{4}x_{32} + \frac{11}{4}x_{33} \\
&\quad + \frac{5}{4}x_{34} + 3x_{35} + \frac{11}{4}x_{36} + \frac{1}{2}x_{22} + \frac{3}{4}x_{23} + 2x_{24} \\
&\le 5x_5 + 4(x_{41} + \cdots + x_{44}) + 3(x_{31} + \cdots + x_{36}) \\
&\quad + 2(x_{21} + \cdots + x_{24}) \\
&\le 3n^*
\end{aligned}$$

which is a contradiction. □

6.5.2 Corollary *If A is the point-triangle incidence matrix associated to a triangle packing problem, then the conflict graph $\mathbb{G}(A)$ of A does not contain any antiweb of the form $C(n,k)$, with $k \geq 5$, as a (node induced) subgraph.*

Proof. To prove this, observe that the antiweb $C(n,k)$ contains as a subgraph an antiweb of the form $C(n^*, k-1)$, with $n^* < n$: Just remove from the former all nodes having indices $i = 0 \mod k$. Hence, if for some fixed $k > 3$, and for all $n \geq 2(k-1)$, the antiweb $C(n,k-1)$ is not contained in $\mathbb{G}(A)$ as a node induced subgraph, then the same must hold for $C(n,k)$, and the result follows from Theorem 6.5.1 by induction on k. □

The situation regarding webs is a bit more complicated. We shall consider at first only webs of the form $\bar{C}(n,k)$ with $k > 3$, and treat the other cases separately, as they do not fit into the general proof due to certain specific structural properties.

Remark that a web $\bar{C}(n,k)$ is a regular graph, i.e., any node i has a constant degree δ given by:

$$\delta = n + 1 - 2k, \tag{6.11}$$

since there are exactly $2k-1$ nodes *not* connected to i, namely, the nodes $i-k+1 \mod n, \ldots, i+k-1 \mod n$. The next theorem proves that the value of δ is bounded for any web obtained from the conflict graph of a set of triangles.

6.5.3 Theorem (Degree of Webs) *Let A be the point-triangle incidence matrix from a triangle packing problem, and $\bar{C}(n,k)$ a web contained as a node induced subgraph in the conflict graph $\mathbb{G}(A)$ of A. If $k > 3$, then the degree of a node in $\bar{C}(n,k)$ is at most equal to three.*

Proof. Suppose there is a submatrix B of A such that $\mathbb{G}(B) = \bar{C}(n,k)$, with $k \geq 4$ and $\delta \geq 5$. Furthermore, let E_i be the triangle associated to node $i \in \mathbb{V}(B)$, for $0 \leq i \leq n-1$, and assume w.l.o.g that $E_0 = \{1,2,3\}$. Now consider the (consecutive) nodes $k, k+1, k+2, k+3$. All these are neighbors from node 0 in the web, which means that each of the four triangles $E_k, E_{k+1}, E_{k+2}, E_{k+3}$ must contain at least one point from the set $\{1,2,3\}$. On the other hand, since $k \geq 4$, the four nodes form a stable set, and hence no two triangles may contain a common point, a contradiction. □

The case when $\delta = 1$ is uninteresting, since then the web decomposes into disjoint cliques of size two. On the other hand, if the degree is equal to two then from (6.11) it follows that n is an odd integer, and that $k = \frac{n-1}{2}$. But, as pointed out in Section 4.2.2 (on page 86), this kind of webs are isomorphic to the odd holes $C(n,2)$, which have already been considered.

Only for $\delta = 3$ a new structure is obtained . Again, (6.11) implies in this case that n is an even integer number and $k = \frac{n-2}{2}$. To prove that these webs can indeed be constructed, consider the set of points $V = \left\{0, \ldots, \frac{3}{2}n - 1\right\}$, and define a set $\mathcal{E} = \{E_0, \ldots, E_{n-1}\}$ of n triangles on V by:

$$E_i = \left\{ i,\ \ i + \frac{n}{2} + 1 \mod n,\ \ n + \left(i + 1 \mod \frac{n}{2} \right) \right\},$$

where $i \in \{0, \ldots, n-1\}$. One can check that, for all $i, j \in \{0, \ldots, n-1\}$, $i \neq j$,

$$E_i \cap E_j \neq \emptyset \quad \Leftrightarrow \quad j = i + \frac{n}{2} + \ell \mod n,$$

where $\ell \in \{-1, 0, 1\}$. Thus, the conflict graph associated to (the point-triangle incidence matrix of) $\mathcal{E}$ is exactly $\bar{C}(n, \frac{n-2}{2})$. We summarize the last observations in the following.

6.5.4 Corollary *If $\bar{C}(n,k)$, with $k > 3$, is a web contained as a node induced subgraph in the conflict graph $\mathbb{G}(A)$ of a point-triangle incidence matrix A, then $\bar{C}$ is either (isomorphic to) an odd hole or has the form $\bar{C}(n, \frac{n-2}{2})$, with n even.*

Theorem 6.5.3 does not hold when $k = 3$. Figure 6.9 shows a counterexample: The web $\bar{C}(9,3)$ can be obtained as the conflict graph of a point-triangle incidence matrix, while from (6.11) it follows that the nodes in this web have degrees equal to 4. On the other hand, the next result states that higher degrees are not possible for this value of k.

$$\begin{array}{c} \begin{array}{ccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \left(\begin{array}{ccccccccc} 1 & . & . & . & 1 & . & . & . & . \\ . & 1 & . & . & . & 1 & . & . & . \\ . & . & 1 & . & . & . & 1 & . & . \\ . & . & . & 1 & . & . & . & 1 & . \\ . & . & . & . & 1 & . & . & . & 1 \\ 1 & . & . & . & . & 1 & . & . & . \\ . & 1 & . & . & . & . & 1 & . & . \\ . & . & 1 & . & . & . & . & 1 & . \\ . & . & . & 1 & . & . & . & . & 1 \\ 1 & . & . & 1 & . & . & 1 & . & . \\ . & 1 & . & . & 1 & . & . & 1 & . \\ . & . & 1 & . & . & 1 & . & . & 1 \end{array} \right) \end{array}$$

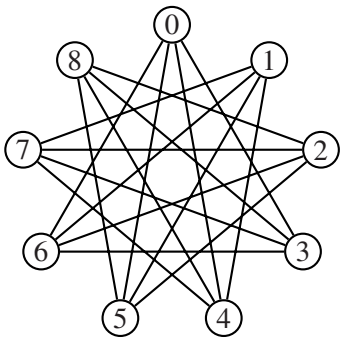


Figure 6.9: A point-triangle incidence matrix that has the web $\bar{C}(9,3)$ as its conflict graph. The degrees of the nodes in this web are all equal to four.

6.5.5 Theorem *Let A be a point-triangle incidence matrix and $\bar{C}(n,3)$ a web contained as a node induced subgraph in $\mathbb{G}(A)$. Then the degree of a node in $\bar{C}(n,3)$ is less or equal to four.*

Proof. Assume there is a web $\bar{C}(n,3)$ contained as a node induced subgraph in $\mathbb{G}(A)$, whose nodes have degrees equal to $\delta^* \geq 5$. Moreover, let $\{0, \ldots, n-1\}$ be the set of nodes of $\bar{C}$. From (6.11) it follows that $n \geq 10$. Now consider the subgraph induced by the eight consecutive nodes $0, \ldots, 7$, as shown in Figure 6.10, and let $E_0, \ldots, E_7$ be the corresponding triangles. Some of these triangles are displayed (as sets of points) near the nodes in the picture.

Since the nodes 0, 1 and 2 form an independent set, the triangles E_0, E_1 and E_2 must be disjoint, i.e., no two of them may contain the same point. Suppose w.l.o.g. that

$$E_0 = \{1, 2, 3\}\,, \quad E_1 = \{4, 5, 6\}\,, \quad E_2 = \{7, 8, 9\}\,.$$

Now observe that E_7 must intersect *all* of these three triangles. This is only possible if E_7 contains one point in common with each of them, say $E_7 = \{1, 4, 7\}$. On the other hand, E_3 must intersect E_7, but it cannot contain any point from $E_1 \cup E_2$, as these are its two "nearest" neighbors on the web, and $k = 3$. Hence, E_3 *must* contain the point 1.

Finally, since 1 is contained both in E_3 and in E_7, it follows that this point cannot be in any of the triangles E_4, E_5 and E_6. However, the nodes corresponding to these triangles form an independent set contained in the neighborhood from E_0, which means that each triangle must intersect E_0 in a different point, a contradiction.

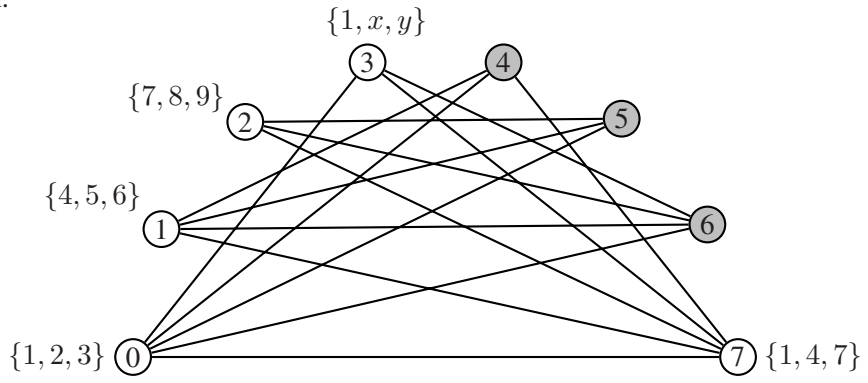


Figure 6.10: Any web $\bar{C}(n, 3)$ where the degree of the nodes is larger than or equal to five contains a subgraph which cannot be obtained as the conflict graph from a point-triangle incidence matrix.

□

The following is obtained as an immediate conclusion from the last theorem, by applying (6.11).

6.5.6 Corollary *The only webs of the form $\bar{C}(n, 3)$ that can be contained as node induced subgraphs in the conflict graph of a point-triangle incidence matrix are $\bar{C}(7, 3)$, $\bar{C}(8, 3)$ and $\bar{C}(9, 3)$. The web $\bar{C}(7, 3)$ is isomorphic to the odd hole $C(7, 2)$.*

At last, let us focus on the case when $k = 2$. If n is an odd number, then the web $\bar{C}(n, 2)$ is isomorphic to the antiweb $C(n, \frac{n-1}{2})$ and constitutes therefore no new structure. Moreover, Theorem 6.5.1, together with its succeeding corollary, implies that

$$\frac{n-1}{2} \leq 4 \quad \Leftrightarrow \quad n \leq 9.$$

Hence, the only webs of this form that might appear in conflict graphs associated to triangle packing problems are $\bar{C}(5,2)$, $\bar{C}(7,2)$ and $\bar{C}(9,2)$.

The case when $k = 2$ and n is an even number is not yet solved. While Figure 6.11 shows that the webs $\bar{C}(8,2)$ and $\bar{C}(10,2)$, containing nodes with degrees five and seven, respectively, may appear in conflict graphs of sets of triangles, for $n \geq 12$ the problem remains open. We have not encountered any triangle-node incidence matrix producing a $\bar{C}(12,2)$, nor have we found some structural feature in this web which leads to the conclusion that such a matrix does not exist.

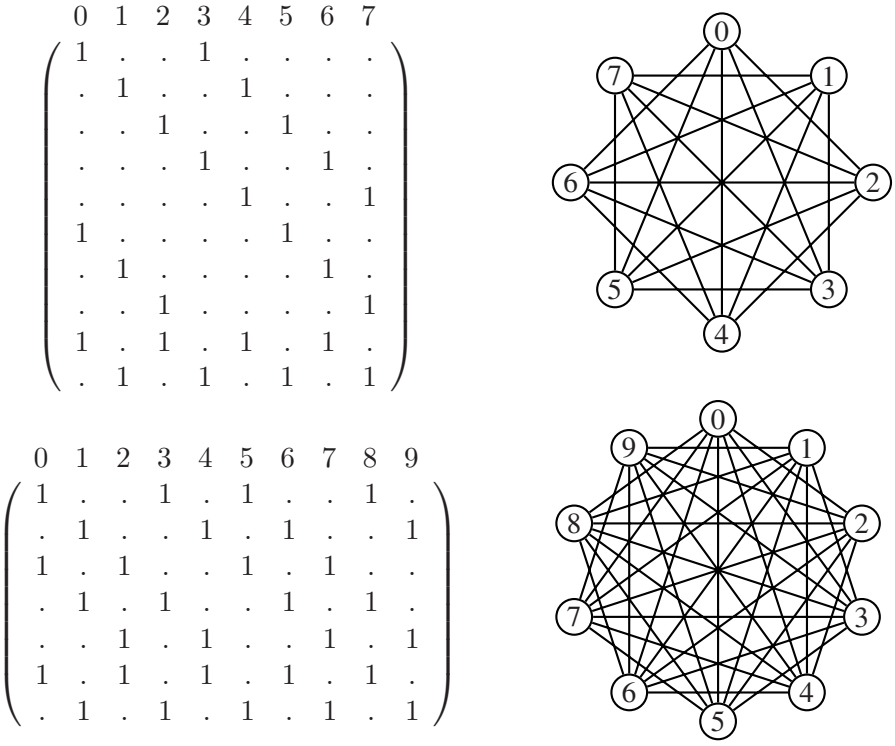


Figure 6.11: Point-triangle incidence matrices that give raise to the webs $\bar{C}(8,2)$ and $\bar{C}(10,2)$.

To conclude this section, we put all the pieces together and summarize the results that have been presented so far.

6.5.7 Theorem (Webs and Antiwebs of Triangles) *Let A be a point-triangle incidence matrix associated to a triangle packing problem. Then all webs and antiwebs that can be contained as node induced subgraphs in the conflict graph $\mathbb{G}(A)$ of A belong to one of the following classes:*

(i) Antiwebs of the form $C(n,k)$, with $k \leq 4$.

(ii) Webs of the form $\bar{C}(n, \frac{n-2}{2})$, with n an even number.

(iii) The webs $\bar{C}(8,3)$ and $\bar{C}(9,3)$.

(iv) The webs $\bar{C}(8,2)$, $\bar{C}(10,2)$.

(v) Webs of the form $\bar{C}(n,2)$, with $n \geq 12$ and even.

Moreover, except for (v)*, constructions are known which produce these structures.*

6.6 About the General κ-Set Packing Polytope

The aim of this section is to describe a few preliminary results, besides of stating some conjectures and open questions, regarding the problem, how the several polyhedral issues discussed so far in the setting of the triangle packing problem could be extended to the generic κ-set packing problem. The ideas presented here are not at the stage of "closed research", but rather intended to incite further work in this subject.

Throughout this section, we will deal with the following formulation of the κ-set packing problem: Given a $0/1$-matrix $A \in \{0,1\}^{m \times n}$ containing at most κ nonzero entries in each column, and a nonnegative cost vector $c \in \mathbb{Q}^n_+$, define the hypergraph $\mathcal{H}(V, \mathcal{E})$ as follows:

$$V := \{1, \ldots, m\}$$
$$\mathcal{E} := \{E_j \,:\, E_j = \text{supp}(A_{\cdot j}),\, 1 \leq j \leq n\}\,.$$

Associate with each hyperedge E_j a nonnegative weight equal to c_j. The task is to find a *maximum-weight matching* in $\mathcal{H}$, i.e., a set of disjoint hyperedges in $\mathcal{E}$ with the largest possible sum of weights. To keep the notation consistent with the previous sections, we shall use the term *points* to denote the elements of V. Observe that no hyperedge contains more than κ points. Moreover, it is straightforward to generalize Lemma 6.2.1 and assume that $\mathcal{H}$ is κ-uniform , i.e., that each hyperedge has cardinality *exactly* equal to κ. (See the remarks preceding this lemma on page 121). We denote such hyperedges simply as κ-hyperedges.

6.6.1 Cliques

A first aspect concerns the polynomial time separability of clique inequalities. For the triangle packing polytope, this was a consequence of Theorem 6.3.1, which states that any clique of n triangles either can be reduced by deleting points to the clique of edges K_3, or has a point i contained in at least $n-5$ triangles from the clique.

An equivalent statement for the general case would require any clique of n κ-hyperedges either to be reducible to a clique of $(\kappa-1)$-hyperedges by deletion of points, or to have a point contained in at least $n-q$ hyperedges, where q is a constant depending solely on κ. It is not clear if this statement holds for $\kappa > 3$,

but even if it held, it is not guaranteed that this would imply the existence of a polynomial time algorithm for separation of cliques.

In fact, such an implication is rather unlikely. Remark that for triangle packing, the polynomial time solvability of the separation problem followed from Theorem 6.3.1 only because all triangle cliques obtained from edge cliques (see Figure 6.3 on page 125) had a "nice" structure that made it possible to identify them. On the other hand, the construction described below reveals that almost *any* clique $\mathcal{Q}$ of triangles may be extended to a clique $\mathcal{Q}'$ of 4-hyperedges in such a way that every point appears in strictly less than $|\mathcal{Q}'| - q$ hyperedges, for all $q \in \mathbb{N}$.

Let $(V, \mathcal{Q})$ be a clique of triangles, and let $n := |\mathcal{Q}|$. Define the following 4-uniform hypergraph:

$$V' := V \uplus \{a_{11}, \ldots, a_{ns}\},$$
$$\mathcal{Q}' := \{E_i \cup \{a_{ij}\} \,:\, E_i \in \mathcal{Q},\ 1 \leq i \leq n,\ 1 \leq j \leq s\}.$$

Thus, each triangle E_i from $\mathcal{Q}$ is used to obtain s different 4-hyperedges in $\mathcal{Q}'$. Observe that $(V', \mathcal{Q}')$ is a clique, since any two hyperedges intersect in their "old" points from V. Moreover, for any point $v \in V$, the degree of v in $\mathcal{Q}'$ (defined as the number of 4-hyperedges from $\mathcal{Q}'$ containing v) is exactly equal to s times its degree in $\mathcal{Q}$. All points in $V' \setminus V$, on the other hand, have degrees equal to 1. Since $|\mathcal{Q}'| = s\,|\mathcal{Q}|$, if no point in V is contained in all triangles from $\mathcal{Q}$, then for any fixed $q \in \mathbb{N}$, it is possible to choose a sufficiently large value of s, such that all points in V' have degrees strictly smaller than $|\mathcal{Q}'| - q$.

Concerning the fractional packing polytope associated to a clique, we have seen in the previous sections that for $\kappa \leq 3$ there exist uniquely determined structures which are extremal in the sense that they produce vertices containing the largest possible number of nonzero coordinates. More precisely, they produce vertices containing exactly $n(\kappa)$ coordinates, all of them equal to $\frac{1}{\kappa}$, where $n(2) = 3$ and $n(3) = 7$. These structures are the K_3, for the edge packing problem, and the Fano plane (see Figure 6.7 on page 144), for the triangle packing problem. Both cliques also share several interesting properties listed on the same page. The next result states two sufficient conditions for a general clique of κ-hyperedges to constitute a similar structure.

6.6.1 Theorem *Let $\mathcal{Q}$ be a clique of κ-hyperedges and $A \in \{0,1\}^{m \times n}$ the corresponding point-hyperedge incidence matrix. If $m = n$ and*

$$|\mathit{supp}(A_{\cdot j}) \cap \mathit{supp}(A_{\cdot \hat{j}})| = 1, \quad \forall j, \hat{j} \in \{1, \ldots, n\},\ j \neq \hat{j}$$

then the following holds:

(i) Each point is contained in exactly κ hyperedges, i.e., the hypergraph is κ-regular.

(ii) $\mathcal{Q}$ consists of exactly $\kappa(\kappa - 1) + 1$ hyperedges, and the same number of points.

(iii) $P(A)$ *contains a fractional vertex* x^* *with all coordinates equal to* $\frac{1}{\kappa}$.

(iv) The dual hypergraph $\mathcal{Q}^*$ *of* $\mathcal{Q}$ *is again a clique of* κ*-hyperedges over a set of* $\kappa(\kappa-1)+1$ *points, and the intersection of any two* κ*-hyperedges of* $\mathcal{Q}^*$ *contains exactly one point.*

(v) $\mathcal{Q}$ *is maximal: it cannot be extended to a larger clique by adding new* κ*-hyperedges (assuming duplicates are not allowed).*

Proof. Consider the $n \times n$-square matrix defined by $B = A^T A$. Remark that the element b_{ij} of B is equal to $|\text{supp}(A_{\cdot i}) \cap \text{supp}(A_{\cdot j})|$. It is easy to check that

$$B = \begin{pmatrix} \kappa & 1 & \cdots & 1 \\ 1 & \kappa & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & \kappa \end{pmatrix} \quad \text{and} \quad B^{-1} = \begin{pmatrix} \alpha & -\beta & \cdots & -\beta \\ -\beta & \alpha & \cdots & -\beta \\ \vdots & \vdots & \ddots & \vdots \\ -\beta & -\beta & \cdots & \alpha \end{pmatrix}$$

where

$$\begin{cases} \alpha = (\kappa + n - 2)\beta \\ \beta = \dfrac{1}{(\kappa + n - 1)(\kappa - 1).} \end{cases}$$

Thus, A is nonsingular, as it is a square matrix and $\det B = \det A \det A^T \neq 0$. It follows that the linear system $Ax = \mathbf{1}$ (where $\mathbf{1}$ is a vector full of ones) has a unique solution equal to

$$x = A^{-1}\mathbf{1} = B^{-1}A^T\mathbf{1} = B^{-1}(\kappa\mathbf{1}) = \frac{\kappa}{\kappa + n - 1}\mathbf{1}. \tag{6.12}$$

Substituting this values in the system, we obtain

$$|\text{supp}(A_{i\cdot})| \frac{\kappa}{\kappa + n - 1} = 1 \quad \Leftrightarrow \quad |\text{supp}(A_{i\cdot})| = \frac{\kappa + n - 1}{\kappa},$$

for all $i \in \{1, \ldots, n\}$, which means that all rows from A contain the same number of ones. Since each column contains κ ones, each row *must* contain κ ones, too, and (i) is proved. Moreover,

$$\frac{\kappa + n - 1}{\kappa} = \kappa \quad \Leftrightarrow \quad n = \kappa(\kappa - 1) + 1,$$

as claimed in (ii). At the same time, plugging this value into (6.12), we obtain (iii).

To prove (iv), remark that the rows of A are the incidence vectors of the hyperedges in $\mathcal{Q}^*$. From (i) it follows that all hyperedges contain exactly κ points, i.e., that $\mathcal{Q}^*$ is κ-uniform. Furthermore, (ii) implies that $|\mathcal{Q}^*| = \kappa(\kappa - 1) + 1$, and that there are the same number of points and hyperedges. Now let $A_{i\cdot}$ and $A_{\hat{i}\cdot}$ be two rows of A and suppose their supports intersect in two (or more) columns, say j and $\hat{\jmath}$. This implies that the hyperedges of $\mathcal{Q}$ corresponding to these columns

contain both i and $\hat{\imath}$, and intersect therefore in more than one point, contradicting the condition of the theorem. So we may assume

$$|\text{supp}(A_{i\cdot}) \cap \text{supp}(A_{\hat{\imath}\cdot})| \leq 1.$$

Define the indicator variable $h(i, \hat{\imath}, j)$ to be equal to one if j is contained in the intersection of the supports from rows $A_{i\cdot}$ and $A_{\hat{\imath}\cdot}$, and zero otherwise. Then, from the last inequality it follows that

$$\begin{aligned} \sum_{i<\hat{\imath}} \sum_{j=1}^{n} h(i, \hat{\imath}, j) &= \sum_{i<\hat{\imath}} |\text{supp}(A_{i\cdot}) \cap \text{supp}(A_{\hat{\imath}\cdot})| \\ &\leq \binom{n}{2} \\ &= \binom{\kappa(\kappa-1)+1}{2}. \end{aligned} \tag{6.13}$$

On the other hand, exchanging the order of the terms in the sum, and taking into account that every hyperedge contains exactly $\binom{\kappa}{2}$ pairs of points,

$$\begin{aligned} \sum_{i<\hat{\imath}} \sum_{j=1}^{n} h(i, \hat{\imath}, j) &= \sum_{j=1}^{n} \sum_{i<\hat{\imath}} h(i, \hat{\imath}, j) \\ &= \sum_{j=1}^{n} \binom{\kappa}{2} = (\kappa(\kappa-1)+1)\binom{\kappa}{2} = \binom{\kappa(\kappa-1)+1}{2}, \end{aligned}$$

which implies that equality holds in (6.13) and hence every pair of rows must intersect exactly at one column.

Finally, it remains to show (v). Let $E' = \{i_1, \ldots, i_\ell\}$ be a set containing at most κ points, and define η_r to be the number of hyperedges of $\mathcal{Q}$ that are covered by the points $\{i_1, \ldots, i_r\}$, for $1 \leq r \leq \ell$. Since $|\text{supp}(A_{i\cdot})| = \kappa$ holds for any row $A_{i\cdot}$ of A, we have $\eta_1 = \kappa$. Now observe (iv) implies that any pair of points is contained exactly in one hyperedge of $\mathcal{Q}$, and denote by E_j the hyperedge containing i_1 and i_2. It follows that $\eta_2 = 2\kappa - 1$, as E_j is covered twice. Depending whether $i_3 \in E_j$ or not, η_3 may take two possible values. In the first case, $\eta_3 = 3\kappa - 2$, since E_j is covered three times. Otherwise, $\eta_3 = 3\kappa - 3$, because three different hyperedges are counted twice, namely, the ones containing the sets of points $\{i_1, i_2\}$, $\{i_1, i_3\}$ and $\{i_2, i_3\}$. (Notice that these are in fact three *different* hyperedges, as on the contrary there would be two hyperedges from $\mathcal{Q}$ intersecting in more than one point). Proceeding in the same manner, one can prove by induction on r that

$$\eta_r \leq r\kappa - (r-1), \ \forall\, 1 \leq r \leq \ell,$$

and that equality holds only if $\{i_1, \ldots, i_r\} \subseteq E_j$. Therefore, to intersect all $\kappa(\kappa - 1) + 1$ hyperedges from the clique we need at least κ points, and all of them must be contained in E_j, which implies $E' = E_j$. □

We have not yet found out if cliques satisfying the two sufficient conditions enunciated in the last theorem exist for all values of κ. However, they can be constructed in infinitely many cases, as shown below. Our proof will make use of the following well-known result from algebra (see for instance Birkhoff & Mac Lane [1996, p. 413]): for any prime number p and any positive integer a, it is always possible to define a *finite field* containing p^a elements. This field is usually denoted by $\mathrm{GF}(p^a)$.

6.6.2 Theorem *Let $\kappa = \nu + 1$, where $\nu = p^a$, p is any prime number, and a any positive integer. Let $\mathcal{Q}$ be the following set of κ-hyperedges:*

$$\mathcal{Q} = \Big\{E_0, \ldots, E_\nu\Big\} \cup \Big\{\hat{E}_{ij} \,:\, 0 \le i \le \nu - 1,\ 0 \le j \le \nu - 1\Big\},$$

with each κ-hyperedge defined by

$$E_i = \big\{\nu^2 + \nu\big\} \cup \big\{i\nu,\, i\nu + 1, \ldots,\, i\nu + \nu - 1\big\},$$
$$\hat{E}_{ij} = \big\{\nu^2 + i\big\} \cup \big\{j \oplus (i \otimes \ell) + \nu\ell \,:\, 0 \le \ell \le \nu - 1\big\},$$

where $\oplus$ and $\otimes$ denote the addition and multiplication in the field $GF(p^a)$, respectively.

Then $\mathcal{Q}$ is a clique that contains $\kappa(\kappa-1)+1$ hyperedges and the same number of points. Moreover, any two hyperedges of $\mathcal{Q}$ intersect in exactly one point.

Proof. By construction we have $|\mathcal{Q}| = \nu^2 + \nu + 1 = \kappa(\kappa - 1) + 1$, and all points are labeled by integers between 0 and $\nu^2 + \nu$. Moreover, observe that each of these integers appears at least once in some hyperedge, which means that there are also $\kappa(\kappa - 1) + 1$ points in $\mathcal{Q}$. It remains to show that any two hyperedges intersect in exactly one point.

For two distinct $E_i, E_j \in \mathcal{Q}$ we have $E_i \cap E_j = \big\{\nu^2 + \nu\big\}$ and the claim follows. In the same manner, observe that the hyperedges $E_0, \ldots, E_\nu$ partition the set $\big\{0, \ldots, \nu^2 + \nu - 1\big\}$ into $\nu + 1$ disjoint "intervals" of size ν, and that any hyperedge $\hat{E}_{ij}$ contains exactly one element from each of these intervals.

The interesting task is to prove that any two hyperedges of the form $\hat{E}_{ij}$ and $\hat{E}_{i\hat{\jmath}}$ intersect in exactly one element. We distinguish two cases:

Case I: $i = \hat{\imath}$
If this happens, $\hat{E}_{ij}$ and $\hat{E}_{i\hat{\jmath}}$ intersect only in their first element $\nu^2 + i$, as $j \neq \hat{\jmath}$ implies that $j \oplus (i \otimes \ell) \neq \hat{\jmath} \oplus (i \otimes \ell)$ for all $\ell \in \{0, \ldots, \nu - 1\}$.

Case II: $i \neq \hat{\imath}$
In this case, $\hat{E}_{ij}$ and $\hat{E}_{\hat{\imath}\hat{\jmath}}$ differ in their first elements. Moreover, notice that for each $0 \le \ell \le \nu - 1$, both hyperedges have exactly one element in the interval $\{\nu\ell, \ldots, \nu\ell + \nu - 1\}$. Hence, the intersection of the hyperedges is given by all values of ℓ that satisfy the equation:

$$j \oplus (i \otimes \ell) = \hat{\jmath} \oplus (\hat{\imath} \otimes \ell).$$

Now the field properties of GF(p^a) guarantee that there exists a unique solution to this equation. Namely,

$$\ell = (i \oplus \hat{i}^{-})^{\div} \otimes (j^{-} \oplus \hat{\jmath}),$$

where z^{-} and $z^{\div}$ are the additive and the multiplicative inverse of z in GF(p^a), respectively.

□

Figure 6.12 shows an example of this construction, obtained for $\kappa = 4$. One can check that each column (resp. each row) of the square 13×13 matrix A contains exactly κ entries equal to one, and that two different columns (resp. rows) have exactly one element in common.

$$A = \begin{pmatrix}
. & 1 & . & . & 1 & . & . & 1 & . & . & 1 & . & . \\
. & 1 & . & . & . & 1 & . & . & 1 & . & . & 1 & . \\
. & 1 & . & . & . & . & 1 & . & . & 1 & . & . & 1 \\
. & . & 1 & . & 1 & . & . & . & . & 1 & . & 1 & . \\
. & . & 1 & . & . & 1 & . & 1 & . & . & . & . & 1 \\
. & . & 1 & . & . & . & 1 & . & 1 & . & 1 & . & . \\
. & . & . & 1 & 1 & . & . & . & 1 & . & . & . & 1 \\
. & . & . & 1 & . & 1 & . & . & . & 1 & 1 & . & . \\
. & . & . & 1 & . & . & 1 & 1 & . & . & . & 1 & . \\
1 & . & . & . & 1 & 1 & 1 & . & . & . & . & . & . \\
1 & . & . & . & . & . & . & 1 & 1 & 1 & . & . & . \\
1 & . & . & . & . & . & . & . & . & . & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & . & . & . & . & . & . & . & . & .
\end{pmatrix}$$

Figure 6.12: Point-hyperedge incidence matrix of a clique having the extremal properties stated in Theorem 6.6.1, for $\kappa = 4$.

The extremal structures considered here can also be interpreted as *finite projective planes*. A finite projective plane of order n is a set of $n^2 + n + 1$ points and lines with the following properties:

- every point lies on $n + 1$ lines
- every line contains $n + 1$ points
- any two points determine a line
- any two lines determine a point

The second and forth properties imply the other two and are equivalent to the conditions of Theorem 6.6.1. The Fano plane is the projective plane of order 2, and in

general we are looking for finite projective planes of order $n = \kappa - 1$. One of the most important unsolved problems in combinatorics (see Bruck & Ryser [1949]) is the conjecture that these do only exist for n equal to a prime power p^a.

To finish this section, we collect here the main issues that remain to be solved with regard to cliques.

6.6.3 Open Questions *Let $\mathcal{H}$ be a κ-uniform hypergraph, where $\kappa \geq 2$ is any fixed integer. Let A be the point-hyperedge incidence matrix for $\mathcal{H}$.*

(i) Is it possible to separate clique inequalities for the packing polytope $P_I(A)$ in polynomial time?

(ii) Is the maximum number of nonintegral coordinates in a vertex of the fractional packing polytope $P(A)$ bounded by a function of κ? Moreover, is this maximum achieved by the cliques presented in Theorem 6.6.1?

(iii) Is it possible to construct such cliques when κ is not the successor of a prime power?

6.6.2 Antiwebs and Webs

Generalizing the observations established for antiwebs and webs of triangles to the case of κ-uniform hypergraphs turned out to be much more complicated than for cliques. The main reason is that most of the proofs are "enumerative" in nature, i.e., they rely on counting certain substructures such as edges or cliques, whose complexity simply explodes with increasing κ. Nevertheless, we can state at least two basic results.

The first one concerns the kind of antiwebs that might appear in the conflict graph $\mathbb{G}(A)$ associated to the point-hyperedge incidence matrix A. While it is obvious that any circulant of the form $C(n, k)$ with $k \leq \kappa$ may occur as a submatrix of A, the question remains about what happens for larger values of k. For instance, in Theorem 6.5.7 we showed that for the case of the triangle packing problem $k \leq 4$ must hold. Moreover, we have seen in Figure 6.8 an example where a $C(n, 4)$ is obtained from a matrix having three nonzero entries per column. For general κ-set packing problems, we have proved only the weaker upper bound presented at next.

6.6.4 Theorem *Let A be the point-hyperedge incidence matrix associated to a κ-set packing problem. Let $C(n, k)$ be an antiweb contained as a node induced subgraph in $\mathbb{G}(A)$. Then $k < 4\kappa$.*

Proof. Group the nodes of $\mathbb{V}$ into blocks $V_1, \ldots, V_r = V_0$ of k consecutive nodes (with maybe some nodes left at the end, which we just include into V_r). Consider two adjacent blocks V_i and V_{i+1}, as shown in Figure 6.13. By induction on k, one can prove that to produce the *crossing* edges, i.e., the edges joining nodes from V_i with nodes from V_{i+1}, at least

$$f(k) = 2 + 3 + \cdots + k = \frac{k(k+1)}{2} - 1$$

entries equal to one are needed in the matrix A. Moreover, since there is no clique containing a node in V_i, a node in V_{i+1}, and a node in $V \setminus (V_i \cup V_{i+1})$, it follows that for every pair of adjacent blocks, a different set of rows from A must be used to obtain the crossing edges. Now observe that there are $\lfloor \frac{n}{k} \rfloor$ pairs of adjacent blocks in the antiweb (remark that $V_0 = V_r$ and V_1 are adjacent). The total number $g(n,k)$ of entries equal to one required by the crossing edges is thus

$$g(n,k) = \left\lfloor \frac{n}{k} \right\rfloor \left(\frac{k(k+1)}{2} - 1 \right) \geq \left(\frac{n}{k} - 1 \right) \left(\frac{k^2+k-2}{2} \right) \geq \frac{n-k}{k} \frac{k^2}{2} > \frac{nk}{4},$$

as $2 \leq k < \frac{n}{2}$. Since the total number of nonzero entries available in A is equal to $n\kappa$, the claim $k < 4\kappa$ follows.

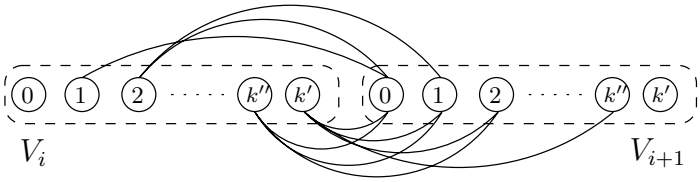


Figure 6.13: Crossing-edges between two consecutive blocks V_i and V_{i+1} of k nodes in a $C(n,k)$ antiweb. ($k'' = k-2$, $k' = k-1$).

□

Future work could improve this upper bound by carrying out a more detailed exploration of other structural features of $\mathbb{G}(A)$. However, classifying and counting edges and cliques, as done in the proof for Theorem 6.5.1, seems to be hopeless for larger values of κ.

Something similar occurs with respect to webs. While it is easy to extend Theorem 6.5.3 to the general κ-set packing problem, further results would require an in-depth analysis that goes beyond the scope of this research. So we stop our remarks at this point, and close the chapter with the following last observation.

6.6.5 Theorem *Let A be the point-hyperedge incidence matrix associated to a κ-set packing problem, and let $\bar{C}(n,k)$ be a web contained as a node induced subgraph in $\mathbb{G}(A)$, with $k > \kappa$. Then $n \leq \kappa + 2k - 1$.*

Proof. Notice that the conflict graph related to a matrix A having at most κ nonzero entries per column cannot contain the complete bipartite graph $K_{1,\kappa+1}$ as a node induced subgraph, since each edge of $K_{1,\kappa+1}$ is associated to a different row in A, and all rows contain one common column index.

Hence, if $\bar{C}(n,k)$ is a web contained as a subgraph in $\mathbb{G}(A)$, and if $k > \kappa$, then the degree δ of a node v in $\bar{C}$ cannot be larger than κ. Otherwise, v together with a set of $\kappa + 1$ consecutive neighbors of it would reveal a node induced $K_{1,\kappa+1}$ in

$\mathbb{G}(A)$. The claim is then proved by replacing this condition into equation (6.11). □

Chapter 7

The ADAC-Problem Revisited. A Solution Approach.

Acknowledgement. The results from this chapter are joint work with Benjamin Hiller[1], Sven O. Krumke[1] and Jörg Rambau[1].

Cooperation. This project was carried out at the Konrad-Zuse-Zentrum Berlin (ZIB)[1] in cooperation with the German Automobile Association (ADAC)[2] and Intergraph Public Safety[3].

7.1 Introduction

In the sequel we combine some of the concepts exposed throughout this thesis and discuss a solution strategy for the ADAC-Problem introduced in Chapter 1. As anticipated there, the ADAC-Problem can be decomposed into an online master problem OLVDP, and a real-time offline vehicle dispatching subproblem VDP. Let us briefly recall the principal aspects related to them:

The Online Master Problem OLVDP

The main task of the OLVDP is to find a good *global* strategy for processing the incoming help requests. Hereby, the following questions have to be addressed:

- How often should a new dispatch for the units be computed?
- Which (if any at all) of the previously scheduled requests may be re-scheduled in future plannings?

[1] Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustr. 7, 14195 Berlin, Germany, `http://www.zib.de`

[2] Allgemeiner Deutscher Automobilclub, Am Westpark 8, 81373 München, Germany, `http://www.adac.de`

[3] Intergraph Corporation, Huntsville, AL 35894 USA, `http://www.intergraph.com`

- Is it convenient to allow *preemption*? (i.e., changes in the next request assigned to a unit?)

The online character of the problem – and at the same time its main challenge – comes from the impossibility to predict if, where and when new requests in the (near) future will take place.

The (Real-Time) Vehicle Dispatching Subproblem VDP

Given a "snapshot" of the situation at some moment in time, the VDP consists in computing a minimum cost dispatch for serving all pending requests with the available units and (if required) contractors. The term "real-time" is used to underline ADAC's requirement that such a dispatch has to be returned in a very short time, usually in no more than 5 seconds for a system load of about 200 requests and 80 units.

In the next section, we present an integer programming formulation for the VDP, based on the Danzig-Wolfe decomposition approach from Section 2.3.1. Our solution algorithms for both the VDP and the OLVDP are described in Section 7.3. The former consists in a column generation procedure, where the pricing subproblem is solved by enumeration in a dynamically restricted search space. We use this scheme to obtain feasible solutions as well as valid lower bounds on the cost of a dispatch. Computational results are reported in Section 7.4.1.

On the other hand, as anticipated before, a theoretical treatment of the OLVDP from the point of view of competitive analysis seems to be out of reach. In Chapter 5 we presented some results concerning a simplified version of the problem, but this is still far away from the practical situation. Here we change our strategy and choose an approach based on *a posteriori* analysis via simulations. We compare how different online algorithms perform on problem instances made from recorded real-world data. In Section 7.4, we report some computational results regarding the two problems.

7.2 An IP Model for the VDP

The VDP can be modeled as a *multi-depot vehicle routing problem with soft time windows* (MDVRPSTW), where each guided vehicle is a depot of its own and the contractors maintain a depot with capacity equal to the number of requests. This problem is known to be NP-hard, since it contains the classic vehicle routing problem as a special case. (And hence also the TSP, as stated in Chapter 2).

Many algorithms, heuristic and exact (i.e., capable of providing optimality gaps), have been proposed for the related *vehicle routing problem with time windows*, where the time windows constitute a hard constraint that has has to be respected in any feasible dispatch (refer to Chapter 2 for a survey). In contrast,

the use of "soft" time windows, which may be violated at a certain cost, has been much less investigated. In our case, soft time windows are one of the "management constraints" specified by ADAC, and their purpose is to fix an adequate trade-off between quality of service and operational costs.

The IP formulation tackled by our algorithm relies on a tour-based, set partitioning model, similar to the one proposed (among others) by Desrosiers et al. [1995]. Such formulations have been successfully applied to other VRP instances involving hard time windows, capacity constraints, dial-a-ride precedence conditions, etc. To the best of our knowledge, however, none of the exact algorithms was ever reported to *predictably* meet strict real-time requirements in a large-scale real-world application, or, put in other words, to produce reasonable answers very early in the course of the optimization process.

An instance of the VDP consists of a set of units $\mathcal{U}$, a set of contractors $\mathcal{V}$, and a set of requests σ, comprising the following input data:

- For each unit $u \in \mathcal{U}$:

$a(u)$	its current position
$a_h(u)$	its home position
$t_{\text{on}}(u)$	its log-on time
$t_{\text{off}}(u)$	the time when its shift ends
$c_{\text{drv}}(u)$	its driving costs per time unit
$c_{\text{svc}}(u)$	its serving costs per time unit
$c_{\text{ot}}(u)$	its overtime costs per time unit
$F(u)$	a set of available capabilities

- For each contractor $v \in \mathcal{V}$:

$c_{\text{svc}}(v)$	a fixed cost for serving a request[4]
$\zeta(v)$	a delay before starting service
$F(v)$	a set of available capabilities

- For each request $r \in \sigma$:

$a(r)$	its position
$\tau(r)$	its release time
$\theta(r)$	its deadline
$\delta(r)$	its service duration
$F(r)$	a set of required capabilities

The soft time windows are expressed in form of a lateness cost function f that depends on the delay W^+ incurred when serving a request. We have considered several possible definitions for f, which were proposed during discussions with personnel from ADAC. Based upon empirical observations, we selected the one

[4] Actually, the service costs depend on the contractor-request pairing, but incorporating this into the computation of the cost coefficient for a contractor tour is trivial and does not affect the performance of the algorithm.

that produced the best effects in terms of setting a reasonable balance between average wait times and operational costs:

$$f(r,t) := c^1_{\text{late}} W^+(r,t) + c^2_{\text{late}} \left(W^+(r,t)\right)^2,$$

where $c^1_{\text{late}}, c^2_{\text{late}}$ are two nonnegative constants, and $W^+(r,t) := (t - \theta(r))^+$ is the *delay* or *late time* incurred if request r is served at time t (i.e., if a unit reaches r at that time). This quadratic lateness cost function has the advantage of giving higher priority to those requests that have been waiting longer, since the cost of each additional minute of delay is larger for them.

A feasible *service tour* T for a unit $u \in \mathcal{U}$ can be described by a sequence $(u, r_1, \ldots, r_k)$ of k distinct requests visited by u in that order, such that $F(r_i) \subseteq F(u)$ holds for all $i \in [k]$ (i.e., the unit is technically equipped to serve each of the requests in the tour).[5] It is assumed that u starts from its current position at time $t_{\text{on}}(u)$ and drives to its home position after having attended the last request in T. We will use the sequence (u) to denote the *go-home* tour, i.e., the tour in which u travels from its current position directly to its home position. A feasible *contractor tour* S is an (unordered) set $\{r_1, \ldots, r_k\}$ of requests that have been assigned to the best possible service contractor capable of serving them. If S is a singleton, we call it an *elementary contractor tour*.

A feasible solution of the VDP (which in the sequel we simply call a *dispatch*) is a set of service and elementary contractor tours[6] with the following two properties:

- each request appears exactly in one (either service or contractor) tour, and,
- each unit has exactly one service tour to drive.

Let us denote by $\mathcal{T}_u$ the set of all feasible service tours for unit u, and by $\mathcal{S}$ the set of all feasible contractor tours. Moreover, we define $\mathcal{T} := \cup_{u \in \mathcal{U}} \mathcal{T}_u$. To guarantee the existence of at least one feasible dispatch, we always assume that $\mathcal{T}$ contains all go-home tours and $\mathcal{S}$ contains all elementary contractor tours.

A service tour $T = (u, r_1, \ldots, r_k)$ fixes the time at which each of the requests $r_1, \ldots, r_k$ are visited by u. For $i \in [k]$, we call this time the *start-of-service time* for r_i in T, and we denote it by $t(T, r_i)$. Considering that the service of requests may not start before their release times, and that units are allowed to wait, $t(T, r_i)$ may be computed according to the following recursive formula:

$$t(T, r_i) = \begin{cases} \max\left\{t_{\text{on}}(u) + d_u(a(u), a(r_1)),\ \tau(r_1)\right\}, & \text{if } i = 1, \\ \max\left\{t(T, r_{i-1}) + \delta(r_{i-1}) + \right. & \\ \qquad \left. d_u(a(r_{i-1}), a(r_i)),\ \tau(r_i)\right\}, & \text{otherwise,} \end{cases}$$

[5] Recall that $[k]$ denotes the set $\{1, \ldots, k\}$.

[6] At the current stage of the project, we are only considering elementary contractor tours. Nonelementary tours can easily be added to the model and could be used in a future version to deal with more complex price structures involving for instance rebates based on the quantity of requests booked to a contractor.

where $d_u(x, y)$ is the time required by u to travel from position x to position y.

The cost c_T of a service tour like the one above is the sum of four terms: *driving*, *service*, *lateness* and *overtime* costs. These are defined by

$$\begin{aligned} c_T = {} & c_{\text{drv}}(u)\, d_u(a(u), r_1) + \sum_{i=2}^{k} c_{\text{drv}}(u)\, d_u(r_{i-1}, r_i) && (7.1)\\ & + c_{\text{drv}}(u)\, d_u(r_k, a_h(u)) && \text{(driving)}\\ & + \sum_{i=1}^{k} c_{\text{svc}}(u)\, \delta(r_i) && \text{(service)}\\ & + \sum_{i=1}^{k} f(r_i, t(T, r_i)) && \text{(lateness)}\\ & + c_{\text{ot}}(u)\ \left(t(T, r_k) + \delta(r_k) + d_u(r_k, a_h(u)) - t_{\text{off}}(u)\right)^+ . && \text{(overtime)} \end{aligned}$$

The cost of each contractor tour is explicitly given as part of the problem data. The cost of a dispatch is equal to the sum of the costs of all service and contractor tours that constitute it.

We are ready to formulate the VDP as an integer program. Using the approach discussed in Section 2.3.1, we define a binary "indicator" variable for each tour. For all $T \in \mathcal{T}$, we take $x_T \in \{0, 1\}$ with the following meaning: $x_T = 1$ if and only if the service tour T is chosen to be in the dispatch. A variable x_S is introduced for each contractor tour $S \in \mathcal{S}$ in a similar way. Moreover, for all $r \in \sigma$ and $X \in \mathcal{T} \cup \mathcal{S}$, let $a^r_X \in \{0, 1\}$ denote whether r is served in X or not. The problem can then be stated as follows:

$$\begin{aligned} \min \quad & \sum_{S \in \mathcal{S}} c_S x_S + \sum_{T \in \mathcal{T}} c_T x_T && && \text{(VDP)}\\ \text{subject to} \quad & && && \\ & \sum_{S \in \mathcal{S}} a^r_S x_S + \sum_{T \in \mathcal{T}} a^r_T x_T = 1 && \forall r \in \sigma, && (7.2)\\ & \sum_{T \in \mathcal{T}_u} x_T = 1 && \forall u \in \mathcal{U}, && (7.3)\\ & x_T \in \{0, 1\} && \forall T \in \mathcal{T}, && (7.4)\\ & x_S \in \{0, 1\} && \forall S \in \mathcal{S}. && (7.5) \end{aligned}$$

Constraints (7.2) specify that each request has to be covered *exactly* by one service or contractor tour. Constraints (7.3) state that every unit has to be assigned exactly one service tour to drive. Finally, (7.4) and (7.5) are integrality restrictions. Observe that, under our assumptions, this integer program has a trivial feasible solution consisting of all elementary contractor tours and all unit go-home tours.

7.3 Solution Approach

Now we present our solution algorithm ZIBDIP for the ADAC-Problem. Section 7.3.1 describes an approach based on column generation for solving the real-time dispatching subproblem VDP. In Section 7.3.2, we discuss some alternatives for embedding this algorithm into an online vehicle routing scheme for the OLVDP.

7.3.1 Solving the VDP

The IP model from the last section has one major drawback which makes a "direct" solution approach impracticable: for a typical problem instance involving about 80 units and more than 200 requests, there are so many feasible tours that only writing down the integer program explicitly would require huge amounts of memory space and computing time. Here is where the column generation approach comes into play. As described in Section 2.3.1, it is not necessary to know all columns of VDP in advance in order to find an optimal solution; all what one needs is a way to *generate* new columns with negative reduced costs dynamically, or to *prove* that no such columns exist.

Before looking at the details of the algorithm, let us first introduce a few definitions and a basic lemma. We shall use the term MLP (for *master linear program*) to denote the linear relaxation of VDP, obtained as usual by replacing restrictions (7.4) and (7.5) with nonnegativity constraints. Observe that this problem still has one variable (though no longer integer) for each feasible tour. The *reduced linear problem* RLP is defined by considering only a "small" subset $\widetilde{\mathcal{T}} \subseteq \mathcal{T}$ of feasible service tours, together with all the $|\sigma|$ elementary contractor tours. Initially, $\widetilde{\mathcal{T}}$ contains all go-home tours, as well as some heuristically constructed tours (see page 170). In the course of the algorithm, $\widetilde{\mathcal{T}}$ is enlarged as new tours are generated. At any given iteration, DRLP denotes the dual linear program of RLP. Observe that DRLP has one variable for each unit and each request. Following a standard convention (see for instance Chvátal [1983]) we call these variables the units' and requests' *prices*. Moreover, the *reduced cost* $\hat{c}_X$ of a (service or contractor) tour $X \in \mathcal{T} \cup \mathcal{S}$ with respect to some dual solution $\pi^* \in \mathbb{R}^m$, with $m = |\sigma| + |\mathcal{U}|$ is defined by

$$\hat{c}_X := \begin{cases} c_X - \sum_{r \in \sigma} a_X^r \pi_r - \pi_u, & \text{if } X \in \mathcal{T}_u \text{ for some } u \in \mathcal{U}, \\ c_X - \sum_{r \in \sigma} a_X^r \pi_r, & \text{otherwise.} \end{cases}$$

7.3.1 Lemma (Lower bound on an optimal dispatch) *At a certain stage of the column generation algorithm, consider the reduced linear problem* RLP *and its dual* DRLP. *Let* $\mathbf{x}^* \in \mathbb{R}^n_+$, *with* $n = |\widetilde{\mathcal{T}}| + |\mathcal{S}|$, *and* $\pi^* \in \mathbb{R}^m$, *with* $m = |\sigma| + |\mathcal{U}|$, *be their corresponding optimal solutions. Then the cost* c^{VDP} *of an optimal dispatch*

is bounded from below by:

$$c^{\text{VDP}} \geq c^{\text{MLP}} \geq c^{\text{RLP}} + \sum_{u \in \mathcal{U}} \min_{T \in \mathcal{T}_u} \hat{c}_T$$

where c^{MLP} *and* c^{RLP} *are the optimal solution values for* MLP *and* RLP*, respectively, and* $\hat{c}_T := c_T - \sum_{r \in \sigma} a_T^r \pi_r^* - \pi_u^*$ *is the reduced cost of service tour* T *with respect to* π^**.*

Proof. The first inequality is trivially valid, since MLP is just the linear relaxation of VDP. For proving the second inequality, let $\bar{\mathbf{x}} \in \mathbb{R}_+^N$, with $N = |\mathcal{T}| + |\mathcal{S}|$, be an optimal solution to MLP. Adding and subtracting the optimal solution values for RLP and DRLP (which are equal to each other, due to the Duality Theorem of linear programming) we obtain

$$\begin{aligned}
c^{\text{MLP}} &= \sum_{T \in \mathcal{T}} c_T \bar{x}_T + \sum_{S \in \mathcal{S}} c_S \bar{x}_S \\
&\quad + \sum_{T \in \widetilde{\mathcal{T}}} c_T x_T^* + \sum_{S \in \mathcal{S}} c_S x_S^* - \sum_{r \in \sigma} \pi_r^* - \sum_{u \in \mathcal{U}} \pi_u^* \\
&= \sum_{T \in \mathcal{T}} c_T \bar{x}_T + \sum_{S \in \mathcal{S}} c_S \bar{x}_S + \sum_{T \in \widetilde{\mathcal{T}}} c_T x_T^* + \sum_{S \in \mathcal{S}} c_S x_S^* \\
&\quad - \sum_{r \in \sigma} \pi_r^* \underbrace{\left(\sum_{u \in \mathcal{U}} \sum_{T \in \mathcal{T}_u} a_T^r \bar{x}_T + \sum_{S \in \mathcal{S}} a_S^r \bar{x}_S \right)}_{=1 \text{ by (7.2)}} - \sum_{u \in \mathcal{U}} \pi_u^* \underbrace{\left(\sum_{T \in \mathcal{T}_u} \bar{x}_T \right)}_{=1 \text{ by (7.3)}}.
\end{aligned}$$

Rearranging terms in this expression,

$$\begin{aligned}
c^{\text{MLP}} &= \sum_{T \in \widetilde{\mathcal{T}}} c_T x_T^* + \sum_{S \in \mathcal{S}} c_S x_S^* + \sum_{u \in \mathcal{U}} \left(\sum_{T \in \mathcal{T}_u} \bar{x}_T \left(c_T - \sum_{r \in \sigma} a_T^r \pi_r^* - \pi_u^* \right) \right) \\
&\quad + \sum_{S \in \mathcal{S}} \bar{x}_S \left(c_S - \sum_{r \in \sigma} a_S^r \pi_r^* \right) \\
&= c^{\text{RLP}} + \sum_{u \in \mathcal{U}} \sum_{T \in \mathcal{T}_u} \bar{x}_T \hat{c}_T + \sum_{S \in \mathcal{S}} \bar{x}_S \hat{c}_S \\
&\geq c^{\text{RLP}} + \sum_{u \in \mathcal{U}} \min_{T \in \mathcal{T}_u} \hat{c}_T.
\end{aligned}$$

The last inequality is the consequence of two facts:

- The reduced cost $\hat{c}_S$ of any contractor tour $S \in \mathcal{S}$ is nonnegative, as all such tours are already contained in RLP, and $\mathbf{x}^*$ is optimal.

- Since (7.3) requires $\sum_{T \in \mathcal{T}_u} \bar{x}_T$ to be equal to one for any unit u, we have replaced a weighted average of the reduced costs from the tours in $\mathcal{T}_u$ by the smallest such value.

□

The Top Level Algorithm

We are now in position to describe our solution algorithm ZIBDIP. Figure 7.1 shows an outline of it at the top level. The input of the algorithm is an instance of the VDP. During the initialization phase, a first reduced linear program is constructed which contains at least all (elementary) contractor tours and all unit go-home tours. Additional tours may be generated by diverse *start heuristics* described later in this section. The best feasible dispatch $\tilde{\mathbf{x}}$ found is stored to ensure the availability of a valid solution at an early stage in the program's execution, as demanded by the real-time requirements of the problem.

The initial RLP is solved to optimality by calling an LP solver[7]. This call returns a pair $(\mathbf{x}^*, \pi^*)$ of primal-dual optimal solutions for RLP and DRLP, respectively.

At each iteration of the main loop, new columns (i.e., service tours) having negative reduced costs with respect to π are generated by the pricing procedure AddNewColumns. Besides of the current dual solution, this procedure uses four parameters to control the column generation process: a *search degree* d, a *search depth* ℓ, an *acceptance threshold* a, and a *sorting criterion* $<$; we will learn more about their exact meaning later. For now, it suffices to say that d and ℓ determine how exhaustive the search for new tours will be. These parameters are adjusted between successive calls to the pricing module, within a nested-loop scheme. At each new iteration of the inner loop, the search depth is incremented. At each new iteration of the outer loop, the search depth is reset to its initial value, and the search degree is incremented. After a call to AddNewColumns, all columns found are added to RLP, and the problem is re-solved to obtain a new pair of primal-dual optimal solutions.

Whenever there has been enough progress on the value c^{RLP} of the current optimal solution for the reduced program, or whenever enough time has elapsed, RLP is solved as an integer program via a call to a third-party IP solver software, and the best available dispatch $\tilde{\mathbf{x}}$ is updated, in case the real-time-conditions force ZIBDIP to terminate in the near future. In order to reduce the impact caused by the intermediate calls to the IP solver on the performance of the column generation procedure, we require the solver to return the best solution found within a certain time-limit Δ. We let Δ increase between successive calls, since at later stages of execution the size of RLP becomes larger.

[7] In our prototype implementation, we have been using CPLEX 8.0 in its standard configuration as both LP and IP solver.

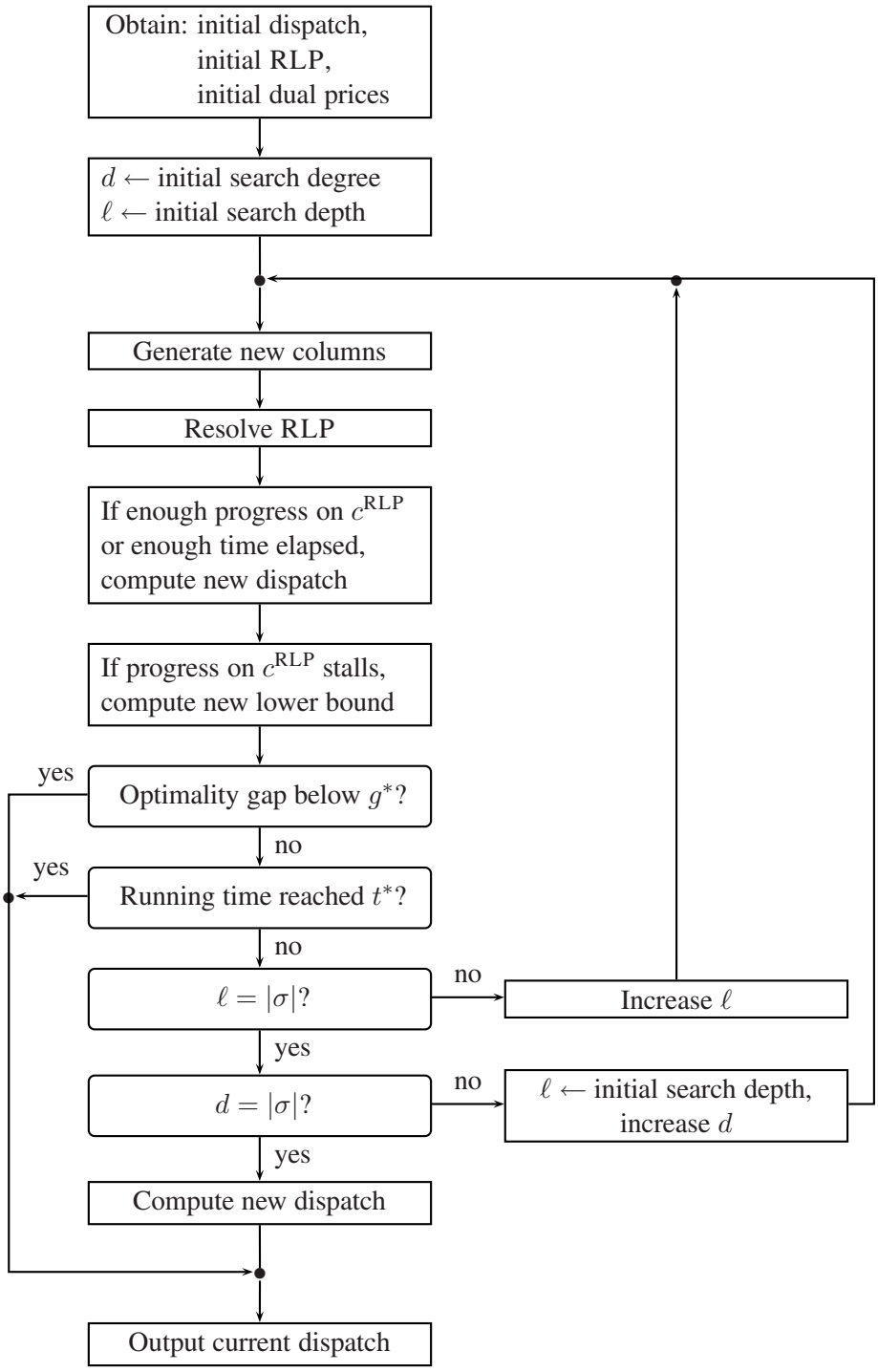


Figure 7.1: Algorithm ZIBDIP for the VDP. Top level view.

Applying Lemma 7.3.1 we could in principle compute a valid lower bound Ω on the cost of the optimum dispatch at any iteration during the column generation process. However, finding a service tour with minimum reduced cost for a unit is in general a task that demands excessive running time, specially in the cases of high system loads.

For this reason, we look for lower bounds only at late stages in the program, for instance when the progress on c^{RLP} stalls. We use two procedures for calculating lower bounds: the same column generation module AddNewColumns with a special configuration of the parameters, and a resource constrained shortest path algorithm RCLowerBound. We shall describe both of them in detail later.

The execution of ZIBDIP terminates when one of the three following *stopping criteria* is met:

- the running time reaches a certain limit t^*,
- a valid dispatch with value c^*, and a valid lower bound Ω are found for which the *optimality gap*
$$\frac{c^* - \Omega}{\Omega}$$
is less or equal to a certain value g^*,
- there are no more service tours with negative reduced costs.

Both t^* and g^* are provided as input parameters.

The last two criteria deserve some additional explanation. Observe that the absence of columns with negative reduced costs implies that $c^{\mathrm{RLP}} = c^{\mathrm{MLP}}$, but it does *not* imply that the optimal *integral* solution for MLP (in other words, the optimum dispatch) will consist solely of columns contained in RLP. In fact, as pointed out in Section 2.3.1, to solve an integer program to optimality via column generation a branch-and-price approach might be required, in which new columns have to be generated at each node of the branch-and-bound tree. Fortunately, in all practical instances we tested, there was no need of branching to achieve the prescribed optimality gaps (between 1% and 5%). On several occasions we were even able to find provably optimal dispatches.

After the column generation process has ended, RLP is solved as an integer program for a last time, and ZIBDIP outputs the best dispatch found as well as information on the optimality gap, if available.

Start Heuristics

On demand, our algorithm can employ heuristics during the initialization phase in order to enhance the quality of both the first available dispatch and the initial dual prices. This is specially useful for high-load instances of the VDP, where column generation from scratch may take too long to give a reasonable answer. We have planned to include several heuristics of both route construction and route

improvement kinds (see section 2.4 for an explanation of these terms). Here we describe the two of them which are currently implemented.

Best-Insertion. A feasible dispatch is built in a greedy manner. We start with all units having go-home tours assigned to them. Then we consider all requests sorted by increasing deadlines, which is the order in which they are usually given in our data sets. For each request $r \in \sigma$, we search for the best unit tour—together with the best position within this tour—where r can be inserted. If the cost for inserting r at this position is smaller than the cost of assigning r to the best contractor capable of serving it, the insertion is done. Otherwise, the corresponding elementary contractor tour is generated.

Node-2-OPT. This heuristic consists in executing successive *node-exchange* steps as long as this helps to improve the value of a given feasible dispatch. We define a node-exchange between two requests r_i and r_j as a swap of their current positions. Three possible cases have to be taken into account: both r_i and r_j might be covered by the same service tour; or, they might belong to different unit tours; or, one of them might be currently served by a contractor. Exchanges are taken whenever they lead to improvements in the solution value. After such a exchange has been executed, both r_i and r_j are tested to be in an (locally) optimal position, in a similar way as in the Best-Insertion procedure. The heuristic stops when no more improving steps can be taken.

Column Generation Procedure

Algorithm 7.1 shows an outline of AddNewColumns, our column generation procedure used to find new service tours with negative reduced costs. For each unit u, the search for such tours is done by enumeration in a branch-and-bound tree (called the *search tree*, for short). Nodes in this tree represent feasible tours for u. The root node corresponds to the go-home tour, and any other node represents the tour obtained by appending one new request *at the end* of the tour associated with its parent node.

The search tree for a unit u is traversed via depth-first-search in the subprocedure SearchTours. This procedure receives a partial tour T and tries to extend it. We denote by $\sigma(T)$ the set of requests that are covered by T. The elements in $\sigma \setminus \sigma(T)$ are sorted according to a *sorting criterion* $<$ and the first d of them (at most) are inserted into a priority queue Q. Both $<$ and d are passed as parameters to SearchTours.[8] At next, the $|Q|$ new tours of the form $\{(T, r) : r \in Q\}$ are considered. Whenever the reduced cost of such a tour T' lies below the given acceptance threshold a, T' is added to the answer-set $\mathcal{T}^*$. The process continues recursively until a given limit ℓ for the length of T (i.e., for the number of requests covered by the tour) is reached. To avoid exploring branches of the search tree unnecessarily, the procedure PruneBound (see below) computes a lower bound on the reduced cost of any tour that can be obtained by further extending T'.

[8] So far, we have tested the use of three such sorting criteria: the reduced cost of the new extended tour T', its completion time, and its primal cost.

Input Instance Σ of the VDP, $\Sigma := (\sigma, \mathcal{U}, \mathcal{V})$,
current dual solution π,
maximal search depth ℓ,
maximal search degree d,
acceptance threshold a,
sorting criterion for requests $<$

Output Set $\mathcal{T}^*$ of new feasible tours with negative red. costs

$\mathcal{T}^* \leftarrow \emptyset$
for $u \in \mathcal{U}$ **do**
  $T \leftarrow (u)$
  SearchTours($\mathcal{T}^*$, T, Σ, π, ℓ, d, a, $<$)
end for
return $\mathcal{T}^*$

SearchTours($\mathcal{T}^*$, T, Σ, π, ℓ, d, a, $<$)
if length$(T) = \ell$ **then**
  return
end if
$Q \leftarrow$ pqueue$(\sigma \setminus \sigma(T), <, d)$
$\{Q$ contains at most d requests to extend T, sorted according to $<\}$
while $Q \neq \emptyset$ **do**
  $r \leftarrow$ ExtractMin(Q)
  $T' \leftarrow (T, r)$
  if $\hat{c}_{T'} < a$ **then**
    $\mathcal{T}^* \leftarrow \mathcal{T}^* \cup \{T'\}$
  end if
  if PruneBound$(\Sigma, T', \pi, \ell) < a$ **then**
    SearchTours$(\mathcal{T}^*, T, \Sigma, \pi, \ell, d, a, <)$
  end if
end while
return

Algorithm 7.1: AddNewColumns

Input Instance Σ of the VDP, $\Sigma := (\sigma, \mathcal{U}, \mathcal{V})$,
current dual solution π,
maximal search depth ℓ,
feasible service tour T

Output A lower bound on $\min \{\hat{c}_{T'} : T' \in \mathcal{T}'\}$, where $\mathcal{T}'$ contains all tours of length at most ℓ that can be created by extending T.

$G \leftarrow \emptyset$
$\hat{\ell} \leftarrow \ell - \text{length}(T)$
for $r \in \sigma \setminus \sigma(T)$ **do**
  $\vartheta(T, r) \leftarrow$ maximal gain of serving r after T
  $G \leftarrow G \cup \{\vartheta(T, r)\}$
end for
$s \leftarrow$ sum of the largest $\hat{\ell}$ elements of G
return $\breve{c}_T - s$

Algorithm 7.2: PruneBound

The acceptance threshold parameter a is used to keep the number of generated columns under control: if too many columns were generated in the previous iteration, we double the (negative) value of a and thus "raise the standards" for accepting new columns in the current iteration. On the other side, if too few columns were generated, we halve the threshold.

Observe that the maximal search depth ℓ and the maximal search degree d strongly influence the running time required by a call to AddNewColumns, since they determine both the height and the breadth of the search trees. For instance, if $\ell = d = |\sigma|$ then we will explore via enumeration all tours in $\mathcal{T}$. On the contrary, if $\ell = 1$ then we are creating just one tour for each unit, in a greedy manner. This fact is exploited by our *dynamic pricing* strategy: In the beginning we consider only small search trees, and thus avoid generating too many columns, since we expect the dual price information to be too "unstable". At the same time, by doing so we enforce RLP to be re-solved often. As the iterations go on the dual solution π will eventually start to settle down. Giving larger values to the depth and degree parameters, we then allow a more exhaustive search to take place.

Pruning the Search Tree

PruneBound returns a lower bound on the reduced cost of any tour T' with length at most ℓ that can constructed by extending a given tour T. This value is computed in the way outlined in Algorithm 7.2.

Let $T = (u, r_1, \ldots, r_k)$ and $T' = (u, r_1, \ldots, r_k, \ldots, r_q)$, with $k < q \leq \ell$. Moreover, denote by

$$t^* := t(T, r_k) + \delta(r_k)$$

the time at which u has just finished serving the last request in T. Note that for all $k < i \leq q$, processing the request r_i at some time after t^* implies that u cannot return to its depot earlier than time $\hat{t}$, where

$$\hat{t} := t^* + d_u(a(r_k), a(r_i)) + \delta(r_i) + d_u(a(r_i), a_h(u)).$$

Thus, the insertion of r_i in T' causes at least the following *unavoidable additional cost* $e(r_i, t^*)$ to be incurred:

$$e(r_i, t^*) := c_{\text{svc}}(u)\,\delta(r_i) + f(r_i, t^*) \\ + c_{\text{ot}}(u)\ \min\left\{\left(\hat{t} - t_{\text{off}}(u)\right)^+,\ \delta(r_i)\right\}.$$

It now follows for the reduced cost $\hat{c}_{T'}$ of T',

$$\begin{aligned}\hat{c}_{T'} &= c_{T'} - \sum_{r\in\sigma(T')} \pi_r - \pi_u \\ &\geq \breve{c}_T - \sum_{i=k+1}^{q} (\pi_{r_i} - e(r_i, t^*)) \\ &\geq \breve{c}_T - \sum_{i=k+1}^{q} \vartheta(T, r_i), \end{aligned} \tag{7.6}$$

where $\breve{c}_T$ is the *reduced precost* of T, obtained by subtracting from $\hat{c}_T$ the driving costs for the return-home trip and the overtime costs; and

$$\vartheta(T, r_i) := (\pi_{r_i} - e(r_i, t^*))^+$$

is called the *maximal gain* of serving r_i after T.

Since $k < q \leq \ell$, (7.6) can be further extended:

$$\begin{aligned}\hat{c}_{T'} &\geq \breve{c}_T - \sum_{i=k+1}^{q} \vartheta(T, r_i) \\ &\geq \breve{c}_T - \sum_{i=1}^{\hat{\ell}} \vartheta^i \end{aligned}$$

where ϑ^i is the i-th largest value from the set $\{\vartheta(T, r) : r \in \sigma \setminus \sigma(T)\}$, and $\hat{\ell} := \ell - \text{length}(T)$. PruneBound computes and returns this last bound.

Lower Bounds

The key issue when computing a lower bound Ω on the cost of the optimal dispatch via Lemma 7.3.1 is to determine for each unit the minimum reduced cost of a feasible service tour. One way to do this consists in simply calling the same column generation procedure AddNewColumns with the parameter setting $\ell = d = |\sigma|$

and $a = 0$. This enforces the pricing module to enumerate all columns with negative reduced costs. Obviously, such an approach can be undertaken only when for some reason we expect the number of new columns to be small. In this case, it might be possible that the pruning criterion PruneBound is sharp enough to allow an efficient traversal of the whole search tree.

Alternatively, we may use the procedure RCLowerBound described in Algorithm 7.3, which consists in solving a *resource constrained shortest path* formulation of the pricing problem, following the strategy presented in Section 2.3.4. For each unit $u \in \mathcal{U}$, we define a digraph $D = (V, A)$ whose set of nodes

$$V := \{a(r) \,:\, r \in \sigma\} \cup \{a(u), a_h(u)\}$$

contains the positions of all requests in σ, together with the current and home positions of u. Any two nodes associated with requests are joined with arcs in both directions. Furthermore, there are arcs leaving from $a(u)$ to all other nodes in V, and arcs going from any node in V to $a_h(u)$.

The algorithm maintains a set of labels $\mathcal{L}_a$ for every $a \in V$. If $a = a(r_k)$ is associated to a request r_k, then each label $L_T \in \mathcal{L}_a$ represents what we call a *partial pseudotour* for u ending at r_k, i.e., a sequence of the form $T = (u, r_1, \ldots, r_k)$ that denotes the "service tour" where u starts from its current position at time $t_{\text{on}}(u)$ and visits requests $r_1, \ldots, r_k$ in that order. We call it a pseudotour because we do *not* require the visited requests to be different from each other. The word partial, on the other hand, means that u does not return to its home position after serving r_k.

To encode such a partial pseudotour T, L_T stores a triplet $(t_T, \breve{c}_T, \uparrow L_{T^-})$, where t_T is the time at which u has completed service of r_k, $\breve{c}_T$ is the reduced precost of T defined on page 174, and $\uparrow L_{T^-}$ is a pointer to the *predecessor* of L_T, i.e., to a label $L_{T^-} \in \mathcal{L}_{a(r_{k-1})}$ associated with the pseudotour $T^- = (u, r_1, \ldots, r_{k-1})$ that is obtained by dropping r_k from the end of T.

Checking the cost definition (7.1) one can see that both t_T and $\breve{c}_T$ can be computed from the information contained in the predecessor of L_T. Conversely, knowing the components of L_T, it is always possible to calculate the label of any pseudotour T^+ created from T by adding a new "request-node" at its end. We call this procedure to *extend* the label L_T.

The sets of labels for the two nodes $a(u)$ and $a_h(u)$ have a special interpretation. The first one contains only the *start label* $(t_{\text{on}}(u), -\pi_u, \cdot)$, which has no predecessor and is used with the sole purpose of forcing all pseudotours to start at position $a(u)$, at time $t_{\text{on}}(u)$, and to include the cost component regarding the unit's dual price. The labels at the home node $a_h(u)$ are associated with *complete* pseudotours, which include the return-home trip and the overtime costs. Observe that they may be obtained by extending other labels as described above.

Let $L_{T_1} = (t_{T_1}, \breve{c}_{T_1}, \uparrow L_{T_1^-})$ and $L_{T_2} = (t_{T_2}, \breve{c}_{T_2}, \uparrow L_{T_2^-})$ be two labels from a set $\mathcal{L}_a$, $a \in V$. L_{T_1} is said to be *dominated* by L_{T_2} if $t_{T_2} \leq t_{T_1}$ and $\breve{c}_{T_2} \leq \breve{c}_{T_1}$. In Section 2.3.4, we proved that dominated labels are unessential for finding a

Input Instance Σ of the VDP, $\Sigma := (\sigma, \mathcal{U}, \mathcal{V})$,
current dual solution π,
a unit $u \in \mathcal{U}$

Output A lower bound on $\min\{\hat{c}_T : T \in \mathcal{T}_u\}$

{Initialization:}
$L_0 \leftarrow (t_{\text{on}}(u), -\pi_u, \cdot)$;
{unprocessed labels are maintained in a priority queue Q, sorted according to their times:}
$Q \leftarrow \text{pqueue}(\{L_0\}, t_{\text{on}}(u))$
$\mathcal{L}_a \leftarrow \emptyset$, $\forall a \in V \setminus \{a(u)\}$
$\mathcal{L}_{a(u)} \leftarrow \{L_0\}$
{Main loop:}
while $Q \neq \emptyset$ **do**
  $L_{\min} \leftarrow \text{popmin}(Q)$
  $a^* \leftarrow$ node at which $L_{\min}$ is located
  {Label extension:}
  for $a^* a^+ \in A$ **do**
    extend $L_{\min}$ with a^+ to L_{T^+}
    $\mathcal{L}_{a^+} \leftarrow \neg\,\text{dom}(\mathcal{L}_{a^+} \cup \{L_{T^+}\})$
    if $L_{T^+} \in \mathcal{L}_{a^+}$ **then**
      $\text{push}(Q, L_{T^+}, t_{T^+})$
    end if
  end for
end while
{Determination of minimum reduced cost of a pseudotour:}
$\hat{c}_{T^*} \leftarrow \min\left\{\breve{c}_T : (t_T, \breve{c}_T, \uparrow L_{T^-}) \in \mathcal{L}_{a_h(u)}\right\}$
return $\hat{c}_{T^*}$

Algorithm 7.3: RCLowerBound

resource constrained shortest path and hence may be deleted without losing any information. However, for a reason that will be explained in a moment, we need to keep some of them. More precisely, we call L_{T_1} to be *strongly dominated* if one of the following is true:

- L_{T_1} is dominated by more than one label from $\mathcal{L}_a$, or,
- L_{T_1} is dominated by another label $L_{T_2} = (t_{T_2}, \breve{c}_{T_2}, \uparrow L_{T_2^-})$ such that their predecessors $L_{T_1^-}$ and $L_{T_2^-}$ belong to the same set $\mathcal{L}_{\hat{a}}$ (i.e., they are "located" at the same node).

For any set $\mathcal{L}$ of labels, $\neg\,\mathrm{dom}(\mathcal{L})$ denotes a new set obtained by deleting from $\mathcal{L}$ all strongly dominated labels.[9]

We are now ready to describe how RCLowerBound works. In the beginning, all sets of labels $\mathcal{L}_a$ with $a \in V \setminus \{a(u)\}$ are initialized to be empty, while $\mathcal{L}_{a(u)}$ contains only the start label defined before. At each main iteration, the label $L_{\min}$ having the minimum time component is chosen, say $L_{\min} \in \mathcal{L}_{a^*}$. Then $L_{\min}$ is extended to generate new labels at all the nodes that can be reached from a^*, including in particular the home node $a_h(u)$. Whenever a label is inserted into a set, strongly dominance is tested and any redundant information is deleted. After this label extension phase has finished, $L_{\min}$ is marked as "processed", to avoid choosing it again, and a new cycle starts. The main loop terminates when there are no more unprocessed labels. All labels at the home node are then scanned, and the minimum value of the cost component is returned.

To prove that the stopping criterion for the main loop is reached at some stage, it suffices to show that after a certain number of iterations no new undominated labels can be created. In fact, suppose label $L_T = (t_T, \breve{c}_T, \uparrow L_{T^-})$ is obtained by extending $L_{T^-} = (t_{T^-}, \breve{c}_{T^-}, \uparrow L_{T^{--}})$, where $T^- = (u, r_1, \ldots, r_k)$ and $T = (u, r_1, \ldots, r_k, r_{k+1})$. Then by definition

$$t_T = \max\{t_{T^-} + d_u(a(r_k), a(r_{k+1})),\ \tau(r_{k+1})\} + \delta(r_{k+1})$$

which means that the time component strictly increases in each new label. The cost component may decrease during an "initial" phase. Yet at some point high times will translate into strictly increasing costs due to the lateness component f. After that, as both time and cost strictly increase, only a finite number of iterations can be executed before all newly generated labels are (strongly) dominated.

We need to state a final comment about the procedure. Observe that RCLowerBound does *not* exactly compute the bound from Lemma 7.3.1, but rather a relaxation from it, since we allow pseudotours to contain repeated requests. In Section 2.3.4, we showed that finding an *elementary* resource constrained shortest path, where all requests are required to be visited at most once, is an $\mathcal{NP}$-hard

[9] One can construct simple examples to show that $\neg\,\mathrm{dom}(\mathcal{L})$ is not unique in general. However, since this fact does not have any relevant consequences for the analysis of our algorithm, we shall not insist on it and rather define $\neg\,\mathrm{dom}(\mathcal{L})$ to be *any* of these sets.

problem. To improve the quality of our bound, however, we do forbid at least one (frequent) kind of these request repetitions: the *two-cycles* of the form r_i, r_j, r_i. This is done by preventing any label from being extended to the same node of its predecessor. However, doing so requires the dominance criterion to be modified, as otherwise feasible solutions may be lost. In plain words, we need to keep not only the "best", but also the "second best" labels for each node. This was the reason for introducing the idea of strongly dominance.

7.3.2 Online Strategies

As mentioned at the beginning of this chapter, we decided to keep away from carrying out a theoretical competitive analysis for the OLVDP, since we do not expect any significant results to be obtained from it. Unless the power of the offline adversary is restricted in some reasonable way, the situation for any online algorithm seems to be hopeless: there are simply too many ways in which it can be mislead. On the other hand, as soon as we start "refining" the model to better reflect the real-world problem instances, we rapidly reach the limit of what is technically possible to compute (at least within the time frame of our project).

Hence, we decided to take a more practical approach and evaluate online strategies on the basis of what we call *a posteriori competitive analysis*: we compare their performance on simulations executed using real-world data. The two main strategies we tested were the following:

> REPLAN. Each time a new request arises, re-solve the VDP considering all pending requests and all available units. (Units currently serving requests are marked as "busy" by setting their log-on times appropriately).
>
> IGNORE[x]. Independently of the arrival of new requests, dispatches are computed at fixed intervals of x time units, where x is a global parameter.

Both strategies use the ZIBDIP algorithm presented in the last section to solve the VDP instances. The test results are discussed in Section 7.4.2.

A phenomenon that was detected during the simulations (and also in the pilot operation of the system at some ADAC help centers) is the appearance of *preemptions*. We say that there is a preemption for a unit u whenever the next request assigned to u has to be changed (without being served) due to the successive recomputations of the service schedule. In most of the cases, this means that orders previously given to the unit have to be canceled. Although preemptions may just constitute an inherent part of the optimization process, if they occur too often they generate unnecessary operational costs. Besides, the idea of "aimlessly wandering units" makes the user of the system feel uncomfortable with it. A high number of preemptions also raises doubts about the quality of the solutions obtained for the intermediate VDP instances.

We have considered three different alternatives to handle the problem of preemptions:

- allow more running time for solving the VDP instances, with the hope that the obtained solutions will become more "stable",
- penalize preemptions in the cost function,
- forbid preemptions completely and check out how much optimization potential is sacrificed by doing this.

Again, the results obtained are reported in the next section.

7.4 Computational Results

7.4.1 Solution of VDP Instances

Figure 7.2 shows the results of an experiment conducted on a collection of snapshot instances belonging to a complete day (December 7th, 2002). It should be mentioned here that these are some of the hardest instances we have encountered since our algorithm started pilot operation at ADAC in October 2002. We recorded 1163 snapshots obtained during a simulation using the IGNORE[1min] strategy (i.e., dispatches computed at fixed intervals of one minute length, see last section).

We have run ZIBDIP on each of these instances with the following configuration:

- The column generation loop is executed until the gap between the lower bound Ω and the solution value of RLP is not larger than 2.5%, or until a time limit of 7,200 seconds (two hours) is reached.
- No intermediate dispatches are computed during column generation.
- After the column generation loop has ended, the IP solver is called to calculate a minimum cost dispatch using the columns in RLP. A time limit of ten minutes and a desired optimality gap of 2.5% (between the integer solution and the solution of RLP) are specified.

The tests were carried out on a two processor 800 MHz Pentium III machine, with 1.0 GB RAM and Linux as operating system.

We use the ratio between the number of waiting requests and the number of available units as a measure of the *load* of the system at any moment in time. Figure 7.2 shows two combined pictures: in the first one, the load of each snapshot along the day is plotted simultaneously with the *optimality gap*[10] obtained by our algorithm when solving it; in the second picture, the load is plotted together with the *running time* required by our algorithm for each snapshot. The horizontal dashed lines mark the aimed 5% optimality gap in the first picture, and the two-hours time limit in the second picture. (Some snapshots were actually run for a

[10]Remark that the optimality gap is taken between the best feasible dispatch found and the best valid lower bound, see page 170 for the definition.

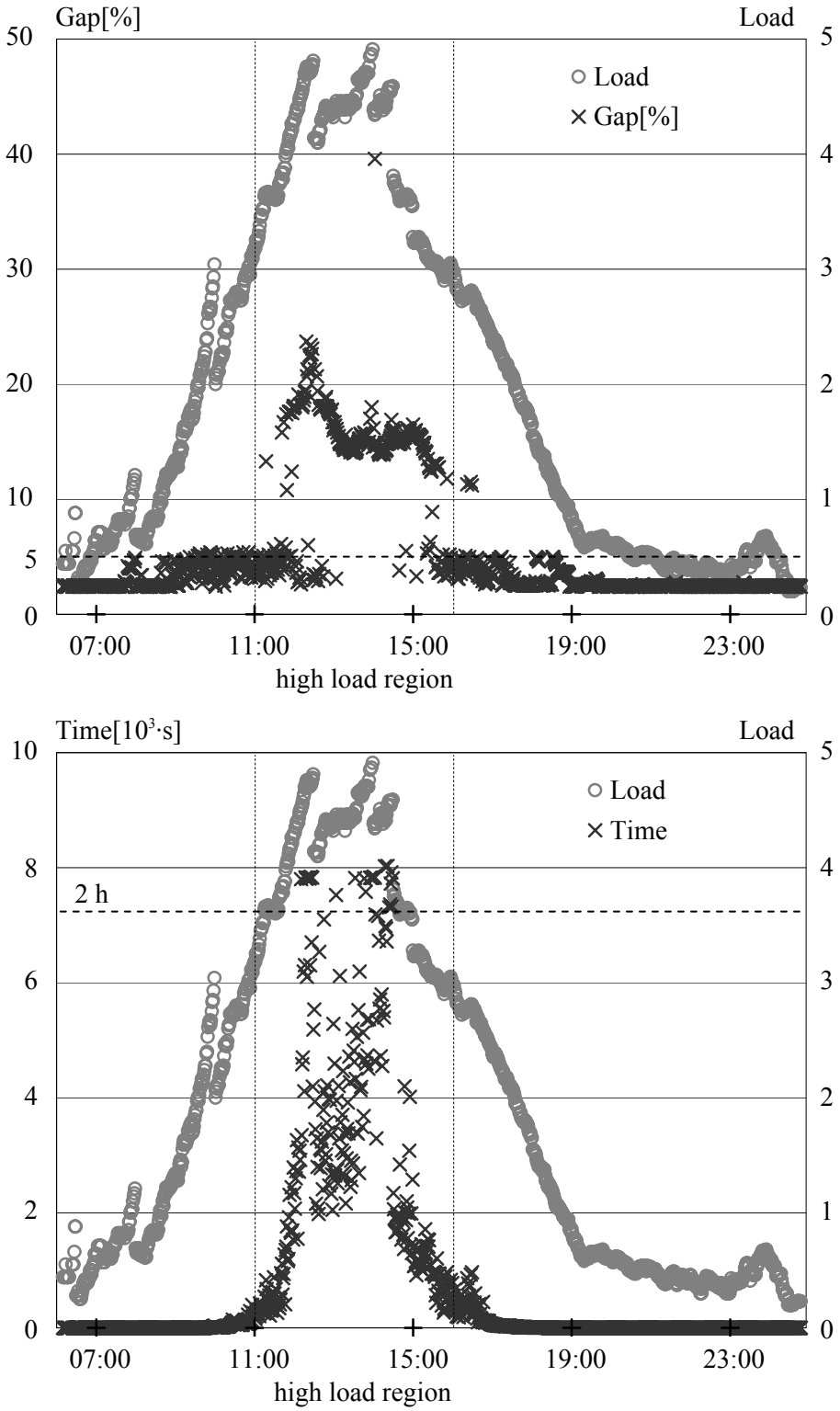


Figure 7.2: VDP snapshot instances solved during one "heavy" day. The horizontal axis represents the time in the day. The first picture shows the load of every snapshot together with the optimality gap obtained by ZIBDIP, the second picture combines the plots of load and required running time. (Input data was recorded by ADAC on December 7th, 2002).

little more than two hours, due to the way how our implementation controls the elapsed time).

Notice at first that the load increases during the day until about 14:00 and then it starts to drop back. There is a peak between 11:00 and 16:00, when the load ranges from 3 to 5 requests/unit. In the figure, this "high load region" has been marked out by enclosing it between the two vertical lines. In contrast, before 9:00 and after 19:00 the load is below 1 requests/unit, which means that there is at least one active unit per waiting request. Observe also that at some points across the day the load "jumps down" from one snapshot to the next. This behavior is caused by the sudden increase on the number of available units, either due to the beginning of a new shift, or to the intervention of the ADAC dispatchers (they might call for additional units to log in whenever they think this is necessary).

For the majority of the snapshots (724 of 1163), a solution with a proven optimality gap below 5% was obtained within one minute. For other 132 instances, this was still accomplished in the first ten minutes. The especially hard snapshots are the ones located in the high load region described above: 165 of them required more than half an hour to be solved to 20% optimality. For one instance, the gap was still about 40% when the time limit was reached. Moreover, there were 8 instances for which ZIBDIP was not able to find a valid lower bound within the time limit.

Notice also that the instance for which the 40% optimality gap was achieved represents an "isolated" case. For the instances both immediately before and immediately after it, the gap obtained was fairly below 20%. This fact illustrates another typical feature from our problem: slight changes in the input may lead to large leaps in the degree of difficulty of an instance.

Figure 7.3 depicts for each instance the optimality gap against the system load. Observe that for almost all instances having a load below 3, the desired 5% optimality gap is reached. If the system load is larger than 3, then our algorithm does not present a unique behavior: for some instances the attained optimality gap still lies below 5%, while for other instances it increases gradually from around 10% (when the load equals to 3) up to around 20% (when the load equals to 5).

To evaluate the performance of our algorithm in a real-time environment, we tested it on the same snapshot instances, but setting a time limit of 10 seconds for the computations. We recorded the *primal* solution values for the snapshots (lower bounds are usually not available within that time frame) and compared them against the ones obtained in the previous test. Figure 7.4 shows the results, plotted in the form of optimality gaps with respect to the lower bounds from the first test. For most snapshots, the gaps attained by the two configurations of the algorithm coincide, revealing that good solutions are found early in the optimization process. (This is even true for the "outsider" snapshot having an optimality gap of about 40%). Differences appear mainly on the borders of the high load region described above.

We also tested ZIBDIP on other real-world snapshots selected from representative problem instances recorded by ADAC. Table 7.1 describes their characteristics.

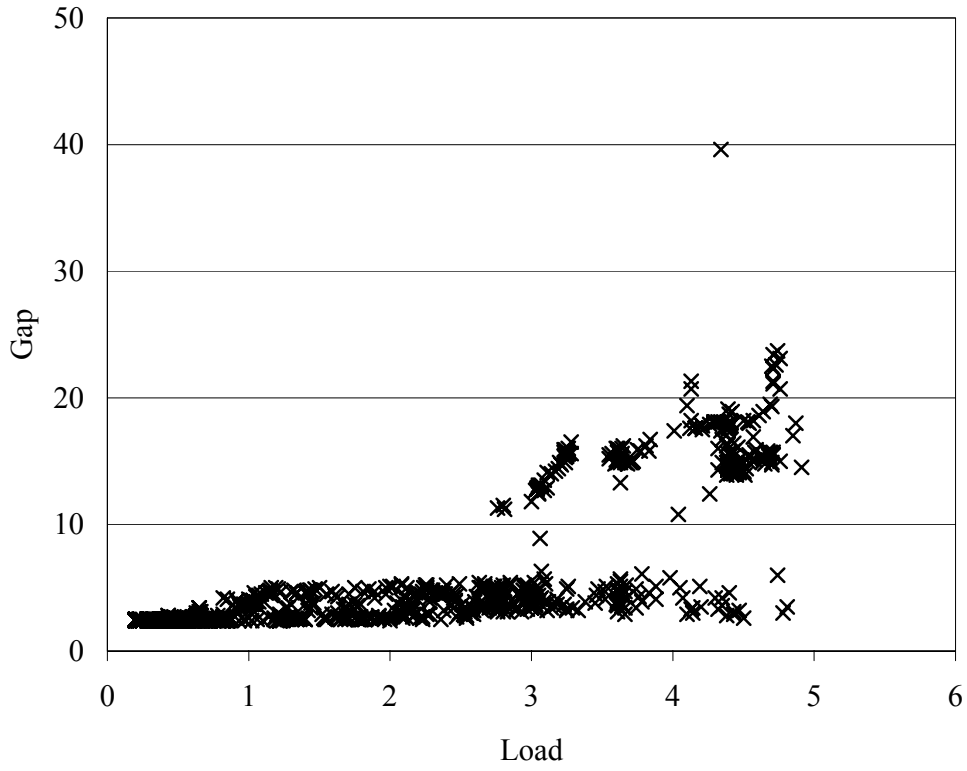


Figure 7.3: Optimality gap versus system load for the VDP instances from December 7th, 2002.

Each test consisted in the following:

- We let ZIBDIP run four times, providing time limits of 5s, 15s, 1min, and 2min. Moreover, the algorithm was configured to skip lower bound computations.
- In the last test, the lower bound module was activated and no time limit was specified. Instead, ZIBDIP was run until an optimality gap below 5% was achieved.

The results are reported in Table 7.2.

We have classified the snapshots into four groups, according to the difficulty they presented for our algorithm. The first eight snapshots are the easy ones: a primal solution within 5% off the optimum is usually found in the first five seconds, and one within 2% in the next ten seconds. In less than five minutes, a lower bound for proving this gap is also available. The next five snapshots form a second group. The solution in the first five seconds still lies 5% off the optimum, but it requires one or two minutes to improve it below 2%. The computation of the lower bound

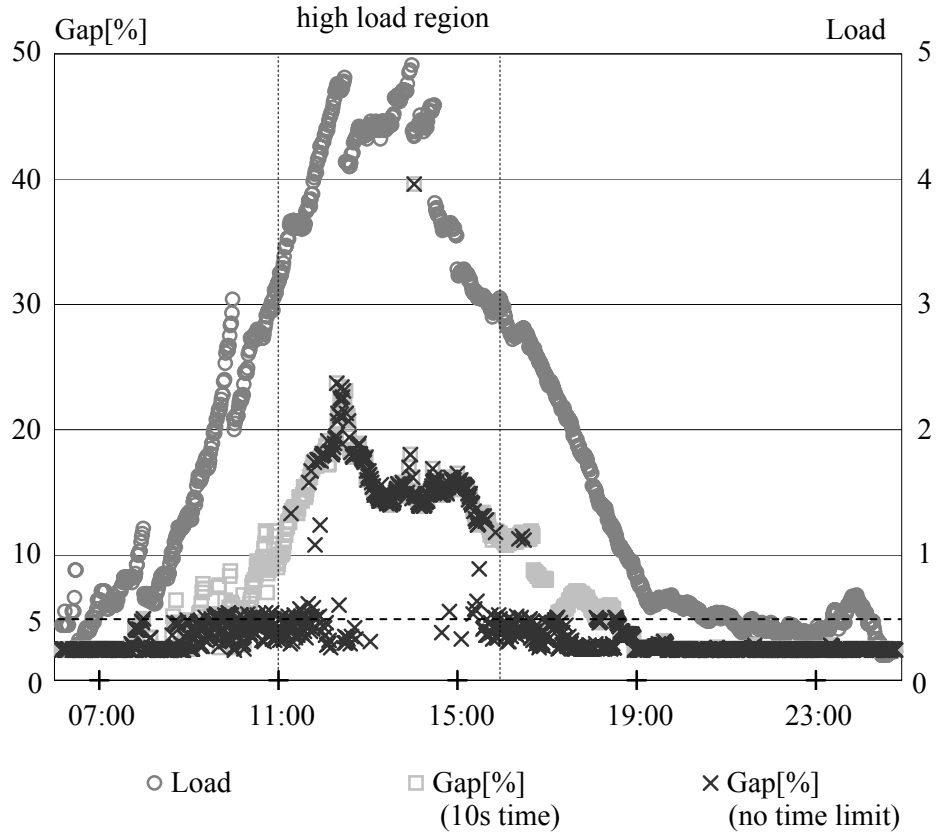


Figure 7.4: Optimality gaps of solutions obtained with ZIBDIP using 10s time limit and without time limit. (Lower bounds were computed only in the unlimited time case).

gets more complicated. Achieving 5% *proven* optimality may demand up to ten minutes. Most of the instances appearing in practice belong to these two classes.

The third group contains hard instances. Their main difficulty does not lie in the computation of a primal solution (this is still 5% apart from the optimum after the first five seconds), but on the computation of lower bounds and optimality gaps. When solving the snapshot `010115o.25`, the algorithm required more than 40 minutes to find a solution with proven 1.8% optimality gap. Meanwhile, a better (primal) solution was found in the first two minutes when the lower bound module was disabled. More than 38 minutes were invested for computing a good lower bound!

Finally, the last group contains especially difficult instances that stem from simulations based on the December 7th (2002) data. For all of them, solutions within 15% to 25% off the optimum are found in the first five seconds and cannot be improved in the next two hours. Again, the computation of a good lower bound is the task which requires the most running time.

Name	requests	units	load
010115m.14	77	23	3.35
010115m.12	75	23	3.26
021218.22	113	47	2.40
010115m.8	62	19	3.26
010115m.15	70	23	3.04
021218.21	115	47	2.45
010115o.14	174	80	2.18
010115o.12	162	80	2.03
010115m.9	68	19	3.58
010115o.17	193	80	2.41
010115o.15	180	80	2.25
010115o.13	160	80	2.00
010115o.22	214	80	2.68
010115m.10	75	19	3.95
010115o.20	202	80	2.53
010115o.25	226	80	2.83
021207.422	362	82	4.41
021207.445	364	82	4.44
021207.378	335	71	4.72
021207.494	418	92	4.54

Table 7.1: Characteristics of the VDP instances tested.

7.4.2 Simulation Tests for the OLVDP

We tested two online strategies on 17 real-world instances of low (7), medium (7), and high (3) system load. Each instance contains all requests appearing during a day, and has information about the availability of units. Table 7.3 lists the total number of requests and units in each case. No contractors were considered.

As anticipated in the last section, the two online strategies we tested are:

- REPLAN: Compute a schedule each time a new request arises or a unit finishes service of a request.
- IGNORE[1min]: Compute a new dispatch at fixed intervals of one minute length.

The two strategies use ZIBDIP as a subroutine to solve the intermediate dispatches. Moreover, in both cases the lower bound module was disabled and a time limit of 10 seconds was specified. The solution of every snapshot was "reused" as a start solution for the next one. We also investigated how the solution changes when preemption (see page 178) is penalized (at three different levels) or strictly forbidden.

The high-load instances were solved on a four processor 2.4 GHz Xeon PC, with 3.5 GB RAM. For the low- and medium-load instances we used a two processor 3 GHz Pentium 4 with 2 GB RAM. Both machines had Linux as operating system.

Tables 7.4 - 7.9 display the results obtained by both algorithms on the low,

Name	IP	gap	time	IP	gap	time	IP	gap	time	IP	gap	time	LB	gap	time
010115m.14	15254	1.3	4.7	15254	1.3	12.1	15254	1.3	45.0	15254	1.3	89.8	15338	15054	1.9	192
010115m.12	13852	1.4	4.7	13849	1.4	14.1	13849	1.4	58.1	13849	1.4	118.2	13917	13658	1.9	136
021218.22	45978	1.4	2.6	45874	1.2	12.4	45817	1.1	45.6	45817	1.1	90.4	45978	45329	1.4	282
010115m.8	11347	1.5	4.6	11301	1.1	10.2	11301	1.1	43.2	11301	1.1	88.1	11373	11183	1.7	89
010115m.15	14253	1.8	5.3	14253	1.8	14.5	14179	1.3	48.3	14161	1.1	92.7	14253	14002	1.8	132
021218.21	48493	2.0	4.1	48090	1.1	11.4	48051	1.0	44.0	48051	1.0	88.3	48406	47560	1.8	138
010115o.14	40780	3.0	4.6	40362	1.9	14.2	40306	1.8	58.2	40279	1.7	118.2	40478	39610	2.2	131
010115o.12	33952	4.5	4.7	32978	1.5	14.3	32904	1.3	58.2	32900	1.2	89.4	33011	32496	1.6	123
010115m.9	14112	3.2	4.4	13963	2.1	14.4	13895	1.6	58.2	13895	1.6	103.6	13937	13672	1.9	292
010115o.17	50090	3.9	4.7	49746	3.2	14.4	49213	2.1	58.2	49128	1.9	118.2	49322	48218	2.3	451
010115o.15	44865	4.5	4.6	44103	2.8	14.1	43811	2.1	58.1	43772	2.0	118.1	43843	42914	2.2	341
010115o.13	36674	4.6	4.8	35936	2.5	14.3	35613	1.6	58.2	35557	1.4	99.4	35672	35068	1.7	263
010115o.22	67589	2.5	2.6	67589	2.5	14.2	67435	2.3	58.4	67255	2.0	118.2	67424	65920	2.3	536
010115m.10	17739	3.8	3.1	17739	3.8	15.0	17422	1.9	59.1	17361	1.6	119.4	17422	17093	1.9	785
010115o.20	59804	4.7	2.6	58818	3.0	14.3	58385	2.2	58.2	58077	1.7	118.3	58354	57128	2.1	932
010115o.25	82839	5.1	2.6	82839	5.1	16.0	81975	4.0	58.8	80041	1.5	120.6	80240	78846	1.8	2522
021207.422	337351	14.8	3.8	337351	14.8	14.6	337351	14.8	62.2	337351	14.8	109.4	337351	293793	14.8	6119
021207.445	366005	14.3	3.5	366005	14.3	14.0	366005	14.3	62.1	366005	14.3	115.2	366005	320122	14.3	7820
021207.378	324000	23.5	3.7	324000	23.5	16.9	324000	23.5	68.2	324000	23.5	121.9	324000	262362	23.5	7842
021207.494	413602	15.0	4.1	413602	15.0	14.5	413602	15.0	51.5	413602	15.0	128.0	413602	359743	15.0	8033

Table 7.2: Results for tests on VDP instances.

Instance	Requests	Units	Conts
low_20021225	838	119	0
low_20021226	377	121	0
low_20021229	936	124	0
low_20030121	839	122	0
low_20030122	861	121	0
low_20030123	921	123	0
low_20030125	968	127	0
med_20021221	1252	131	0
med_20021222	1075	130	0
med_20021228	1023	129	0
med_20030118	1050	128	0
med_20030119	1029	122	0
med_20030120	1183	131	0
med_20030124	1087	127	0
high_20021206	1490	126	0
high_20021213	2566	146	0
high_20021214	1757	131	0

Table 7.3: Characteristics of the OLVDP instances tested.

medium and high load instances. For every instance, the following information was recorded:

- the *maximum load* of the system, defined as the maximum number of requests present in a snapshot divided by the average number of units in a snapshot;
- the *average load*, given by the sum of requests over all snapshots divided by the sum of units over all snapshots;
- the *maximum number of preempts* obtained for a single unit during the whole day;
- the *total number of preempts* for all units; and,
- the *total real cost* incurred during the day. (Excluding preempt penalizations).

The column PP indicates the level of penalization for preemption: 20, 40, or 100 monetary units per preempt. The sign ∞ means that preempts are strictly forbidden. (Which is *not* achieved by setting a huge penalization, but rather by changing the admission rules for tours).

In general, the two online strategies produce similar results, with a slight advantage in favor of REPLAN. This holds both for the cost and for the number

of preempts. However, there are also a few instances in which IGNORE[1min] performs better (see for example `low_20021229` with preemption forbidden).

The total number of preempts decreases (as expected) when the penalization is incremented. The maximum number of preempts for a unit, however, does not clearly follow the same pattern and may even increase (see e.g. instance `med_20030118`). This happens independently from the online strategy chosen.

The optimization potential lost when preempts are strictly forbidden ranges between 5% and 30%. Again, these figures are the same for both online strategies and for all levels of system load.

Instance		**Load**		**Preempts**		
	PP	max	avg	max	total	**Cost**
low_20021225	20	1.19	0.40	5	71	60174.46
	40	1.22	0.40	4	48	60203.58
	100	1.19	0.40	2	30	60652.76
	∞	0.75	0.19	0	0	73985.80
low_20021226	20	1.84	0.63	3	33	47189.27
	40	1.72	0.54	3	25	48005.38
	100	1.78	0.60	2	15	46463.36
	∞	1.34	0.48	0	0	61206.22
low_20021229	20	1.39	0.41	5	76	63247.77
	40	1.45	0.41	3	60	64508.03
	100	1.47	0.41	2	40	66795.55
	∞	0.92	0.21	0	0	73440.67
low_20030121	20	0.81	0.31	3	29	43155.53
	40	0.84	0.31	2	30	40976.23
	100	0.82	0.33	2	22	42453.33
	∞	0.48	0.15	0	0	47619.45
low_20030122	20	0.81	0.34	3	36	41813.03
	40	0.81	0.33	3	32	41156.02
	100	0.90	0.35	2	20	43292.27
	∞	0.47	0.17	0	0	43511.05
low_20030123	20	0.83	0.36	2	38	44730.64
	40	0.84	0.36	3	31	44426.07
	100	0.83	0.36	1	22	43774.12
	∞	0.53	0.16	0	0	45748.61
low_20030125	20	1.07	0.44	4	71	73269.70
	40	1.16	0.44	4	59	74274.07
	100	1.16	0.46	5	49	76862.91
	∞	0.85	0.26	0	0	99777.89

Table 7.4: Results for strategy IGNORE[1min] on instances of low load.

Instance		**Load**		**Preempts**		
	PP	max	avg	max	total	**Cost**
low_20021225	20	0.91	0.44	5	76	59082.95
	40	0.86	0.43	5	54	58393.62
	100	0.91	0.46	3	33	60625.56
	∞	0.56	0.22	0	0	73814.41
low_20021226	20	1.63	0.96	3	36	45664.31
	40	1.59	0.94	5	25	46006.57
	100	1.68	0.97	2	18	45115.80
	∞	1.46	0.70	0	0	58334.43
low_20021229	20	0.99	0.46	5	63	63399.90
	40	0.99	0.46	3	59	59654.65
	100	0.99	0.46	3	40	70327.14
	∞	0.66	0.23	0	0	79998.75
low_20030121	20	0.68	0.33	3	40	41946.36
	40	0.66	0.32	2	35	40911.23
	100	0.64	0.33	2	23	41291.67
	∞	0.42	0.16	0	0	46508.22
low_20030122	20	0.65	0.37	3	38	41521.09
	40	0.70	0.37	3	29	42957.41
	100	0.70	0.37	2	24	41838.11
	∞	0.48	0.20	0	0	42894.48
low_20030123	20	0.68	0.38	3	40	44259.35
	40	0.73	0.38	2	30	44513.01
	100	0.71	0.38	3	22	47764.95
	∞	0.46	0.18	0	0	46686.85
low_20030125	20	0.84	0.49	4	77	68260.71
	40	0.86	0.50	3	63	73287.95
	100	0.91	0.51	5	44	74336.53
	∞	0.75	0.32	0	0	98333.48

Table 7.5: Results for strategy REPLAN on instances of low load.

Instance		**Load**		**Preempts**		
	PP	max	avg	max	total	**Cost**
med_20021221	20	1.80	0.61	9	97	106419.10
	40	1.63	0.58	5	78	102754.00
	100	1.76	0.58	5	62	101153.30
	∞	1.27	0.40	0	0	125322.30
med_20021222	20	0.98	0.43	3	59	61273.00
	40	1.03	0.44	3	48	59660.05
	100	1.06	0.46	3	43	65876.80
	∞	0.67	0.23	0	0	71072.35
med_20021228	20	1.59	0.50	6	69	82811.43
	40	1.52	0.49	3	55	81443.26
	100	1.58	0.50	3	36	81537.02
	∞	1.12	0.30	0	0	88947.70
med_20030118	20	1.25	0.45	3	59	67692.81
	40	1.38	0.44	5	55	67016.35
	100	1.41	0.45	4	39	69127.66
	∞	0.94	0.25	0	0	80722.86
med_20030119	20	1.29	0.47	2	51	63757.81
	40	1.28	0.46	3	40	64499.95
	100	1.27	0.45	3	27	62914.14
	∞	0.85	0.26	0	0	75125.90
med_20030120	20	1.14	0.50	5	58	71809.81
	40	1.18	0.50	3	39	73583.92
	100	1.15	0.51	3	34	78678.86
	∞	0.83	0.30	0	0	83336.18
med_20030124	20	0.92	0.47	5	76	65084.40
	40	1.00	0.49	3	51	69630.53
	100	1.02	0.49	2	26	71652.59
	∞	0.67	0.28	0	0	78933.06

Table 7.6: Results for strategy IGNORE[1min] on instances of medium load.

Instance		Load		Preempts		
	PP	max	avg	max	total	**Cost**
med_20021221	20	1.33	0.69	5	91	99049.50
	40	1.30	0.68	4	74	95231.71
	100	1.42	0.74	6	62	112824.70
	∞	1.11	0.47	0	0	120951.70
med_20021222	20	0.83	0.47	3	68	60838.81
	40	0.78	0.47	3	52	61318.41
	100	0.88	0.50	3	40	64476.14
	∞	0.58	0.25	0	0	68577.20
med_20021228	20	1.27	0.62	5	70	81664.39
	40	1.20	0.60	5	49	78547.56
	100	1.25	0.62	3	35	82098.04
	∞	0.90	0.37	0	0	86380.92
med_20030118	20	1.10	0.54	3	67	67859.59
	40	1.13	0.53	5	66	67626.26
	100	1.04	0.54	4	41	70519.26
	∞	0.83	0.31	0	0	79688.72
med_20030119	20	0.99	0.52	3	62	62104.24
	40	0.96	0.51	3	41	62399.22
	100	0.99	0.51	3	32	63116.17
	∞	0.74	0.30	0	0	75990.65
med_20030120	20	0.95	0.55	5	73	76985.85
	40	0.99	0.54	3	46	74539.19
	100	0.97	0.55	2	32	77196.65
	∞	0.71	0.31	0	0	81422.37
med_20030124	20	0.83	0.52	5	68	66093.19
	40	0.83	0.51	4	52	65837.62
	100	0.86	0.53	4	35	70566.99
	∞	0.59	0.29	0	0	77281.46

Table 7.7: Results for strategy REPLAN on instances of medium load.

Instance		**Load**		**Preempts**		
	PP	max	avg	max	total	**Cost**
high_20021206	20	1.49	0.79	4	89	125411.30
	40	1.51	0.81	4	65	127017.40
	100	1.54	0.81	3	35	132941.50
	∞	1.45	0.61	0	0	171021.80
high_20021213	20	4.42	1.88	7	149	707088.00
	40	4.51	1.86	5	137	687735.20
	100	4.54	1.87	5	85	681813.80
	∞	4.34	1.75	0	0	819823.80
high_20021214	20	5.49	1.90	6	132	666345.80
	40	5.72	1.91	7	147	679517.50
	100	5.39	1.87	5	83	657351.70
	∞	5.32	1.76	0	0	773296.00

Table 7.8: Results for strategy IGNORE[1min] on instances of high load.

Instance		**Load**		**Preempts**		
	PP	max	avg	max	total	**Cost**
high_20021206	20	1.34	0.84	4	87	124576.10
	40	1.33	0.83	4	89	125476.10
	100	1.34	0.85	4	57	129623.20
	∞	1.23	0.66	0	0	178690.30
high_20021213	20	3.92	2.23	8	215	690317.30
	40	3.87	2.21	6	151	704327.70
	100	3.94	2.25	5	113	711134.20
	∞	3.76	2.05	0	0	774896.50
high_20021214	20	4.98	2.59	6	128	688662.40
	40	4.86	2.57	6	137	668457.30
	100	4.88	2.61	6	99	700186.30
	∞	4.83	2.48	0	0	761569.30

Table 7.9: Results for strategy REPLAN on instances of high load.

Appendix A

Fractional vertices associated to cliques of triangles

The following nine matrices $A_1, \ldots, A_9$ were obtained by computer enumeration. They characterize the fractional vertices in cliques of triangles completely. In fact, if A is the triangle-point incidence matrix of a clique of triangles and x_* is a fractional vertex of $P(A)$, then each submatrix A_* of A appearing as linear programming basis of x_* must be a equal to one of $A_1, \ldots, A_9$. (See Section 6.4).

$$A_1 = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \qquad x_*^T = \left(\tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}\right)$$

$$A_2 = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \qquad x_*^T = \left(\tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}\right)$$

$$A_3 = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \qquad x_*^T = \left(\tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}, \tfrac{2}{3}\right)$$

$$A_4 = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \qquad x_*^T = \left(\tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}\right)$$

$$A_5 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \qquad x_*^T = \left(\tfrac{1}{5}, \tfrac{1}{5}, \tfrac{3}{5}, \tfrac{2}{5}, \tfrac{2}{5}\right)$$

$$A_6 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \qquad x_*^T = \left(\tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{2}, \tfrac{1}{4}, \tfrac{1}{2}\right)$$

$$A_7 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} \qquad x_*^T = \left(\tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}\right)$$

$$A_8 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix} \qquad x_*^T = \left(\tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}\right)$$

$$A_9 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} \qquad x_*^T = \left(\tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}\right)$$

Bibliography

Aarts & Lenstra (Eds.) (1997). *Local search in combinatorial optimization*. John Wiley & Sons Ltd.

Afrati, Cosmadakis, Papadimitriou, Papageorgiou & Papakostantinou (1986). The complexity of the travelling repairman problem. *RAIRO, Inf. Théor. 20, 79–87*.

Aneja, Aggarwal & Nair (1983). Shortest chain subject to side constraints. *Networks 13, 296–302*.

Appelgren (1969). A column generation algorithm for a ship scheduling problem. *Transp. Sci. 3, 53–68*.

Ascheuer (1995). *Hamiltonian Path Problems in the On-line Optimization of Flexible Manufacturing Systems*. PhD thesis, Technische Universität Berlin, Germany. Available as Technical Report TR 96-03, Konrad-Zuse-Zentrum für Informationstechnik Berlin, at URL http://www.zib.de/ZIBbib/Publications.

Ascheuer, Fischetti & Grötschel (1999). Solving the Asymmetric Travelling Salesman Problem with Time Windows by Branch-and-Cut. Preprint SC 99-31, Konrad-Zuse-Zentrum Berlin. Available at `http://www.zib.de`.

Ascheuer, Grötschel, Krumke & Rambau (1999). Combinatorial Online Optimization. In P. Kall (Ed.), *Operations Research Proceedings 1998. Selected papers of the 4th annual international conference, OR98* pp. 21–37. Springer Verlag.

Ascheuer, Krumke & Rambau (2000). Online Dial-a-ride Problems: Minimizing the Completion Time. In Reichel & Tison [2000], pp. 639–650.

Assad, Ball, Bodin & Golden (1983). Routing and scheduling of vehicles and crews. *Computers & Operations Research 10(2), 63–211*.

Assad & Golden (1988). *Vehicle Routing: Methods and Studies*, volume 16 of *Studies in Management Science and Systems*. North-Holland.

Ausiello, Feuerstein, Leonardi, Stougie & Talamo (1994). Serving Requests With On-line Routing. In E. Schmidt & S. Skyum (Eds.), *Proceedings of the 4th Scandinavian Workshop on Algorithm Theory*, volume 824 of *Lecture Notes in Computer Science* pp. 37–48. Springer Verlag.

—— (2001). Algorithms for the On-line Traveling Salesman. *Algorithmica 29(4), 560–581*.

Baker (1983). An exact algorithm for the time–constrained traveling salesman problem. *Operations Research 31(5), 938–945*.

Baker & Schaffer (1986). Solution improvement heuristics for the vehicle routing and scheduling problem with time window constraints. *Am. J. Math. Manage. Sci. 6, 261–300*.

Balas (1980). Cutting Planes From Conditional Bounds: A New Approach to Set Covering. *Math. Prog. Study 12, 19–36*.

Balas & Ho (1980). Set Covering Algorithms Using Cutting Planes, Heuristics, and Subgradient Optimization: A Computational Study. *Math. Prog. Study 12, 37–60*.

Balas & Padberg (1976). Set Partitioning: A Survey. *SIAM Rev. 18, 710–760*.

Ball, Magnanti, Monma & Nemhauser (Eds.) (1995). *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*. Amsterdam: Elsevier Science B.V.

Barahona & Mahjoub (1994). Compositions of Graphs and Polyhedra II: Stable Sets. *SIAM J. on Disc. Math. 7, 359–371*.

Bartal & Koutsoupias (2000). On the Competitive Ratio of the Work Function Algorithm for the k-Server Problem. In Reichel & Tison [2000], pp. 605–613. Available at `citeseer.nj.nec.com/bartal00competitive.html`.

Becchetti, Leonardi, Marchetti-Spaccamela, Schäfer & Vredeveld (2003). Average case and smoothed competitive analysis for the multi-level feedback algorithm. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pp. 462–471., Cambridge, MA.

Bellman (1957). *Dynamic Programming*. Princenton, NJ: Princenton University Press.

—— (1958). On a routing problem. *Q. appl. Math. 16, 87–90*.

Ben-David, Borodin, Karp, Tardos & Wigderson (1994). On the Power of Randomization in On-Line Algorithms. *Algorithmica 11(1), 2–14*.

Berge (1961). Färbung von Graphen, deren sämtliche bzw. deren ungerade Kreise starr sind. *Wiss. Zeit. der Martin-Luther Universität Halle. Math. Natur. Reihe 10, 114–115*.

Berge (1963). Some classes of perfect graphs. In *Six papers on graph theory*, pp. 1–21., Calcutta. Research and Training School, Indian Statistical Institute.

Berge (1967). Some classes of perfect graphs. In F. Harary (Ed.), *Graph Theory and Theoretical Physics* pp. 155–165. New York: Academic Press.

—— (1996). The history of the perfect graphs. *Southeast Asian Bull. Math. 20(1), 5–10*.

—— (1997). Motivations and history of some of my conjectures. *Disc. Math. 165/166, 61–70*.

Bertossi, Carraresi & Gallo (1987). On Some Matching Problems Arising in Vehicle Scheduling Models. *Networks 17, 271–181*.

Bianco, Mingozzi, Ricciardelli & Spadoni (1989). Algorithms for the Crew Scheduling Problem based on the Set Partitioning Formulation. Technical

Report R.280, Instituto di Analisi dei Sistemi ed Informatica del CRN, Viale Manzoni 30, 00185 Roma.

Birkhoff & Mac Lane (1996). *A Survey of Modern Algebra* (5th ed.). New York: Macmillan.

Blom, Krumke, de Paepe & Stougie (2000). The Online-TSP Against Fair Adversaries. In Bongiovanni et al. [2000], pp. 137–149.

Bodin & Golden (1981). Classification in Vehicle Routing and Scheduling. *Networks 11, 97–108*.

Bongiovanni, Gambosi & Petreschi (Eds.) (2000). *Proceedings of the 4th Italian Conference on Algorithms and Complexity*, volume 1767 of *Lecture Notes in Computer Science*. Springer Verlag.

Borndörfer (1998). *Aspects of Set Packing, Partitioning and Covering*. Ph.D. dissertation, Tech. Univ. of Berlin, Berlin.

Borndörfer, Grötschel & Löbel (2001). Scheduling Duties by Adaptive Column Generation. ZIB-Report SC 01-02, Konrad-Zuse-Zentrum Berlin. Available at `http://www.zib.de`.

Borodin & El-Yaniv (1998). *Online computation and competitive analysis*. Cambridge University Press.

Boulala & Uhry (1979). Polytope des independants d'un graphe série-parallèle. *Disc. Math. 27, 225–243*.

Bramel & Simchi-Levi (1997). On the effectiveness of set covering formulations for the vehicle routing problem with time windows. *Op. Res. 45, 295–301*.

Branco & Paixão (1987). A Quasi-Assignment Algorithm for Bus Scheduling. *Networks 17, 249–269*.

Bruck & Ryser (1949). The Nonexistence of Certain Finite Projective Planes. *Canad. J. Math. 1, 88–93*.

Caprara & Fischetti (1996). $\{0, \frac{1}{2}\}$-Chvátal-Gomory Cuts. *Math. Prog. 74(3), 221–235*.

Carpaneto, Dell'Amico, Fischetti & Toth (1989). A Branch and Bound Algorithm for the Multiple Depot Vehicle Scheduling Problem. *Networks 19, 531–548*.

Carraresi & Gallo (1984). Network models for vehicle and crew scheduling. *European Journal of Operations Research 16, 139–151*.

Christofides (1976). Worst-case Analysis of a New Heuristic for the Traveling Salesman Problem. Tech. report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA.

Christofides, Mingozzi & Toth (1981a). Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Math. Prog. 20, 255–282*.

—— (1981b). State-space relaxation procedures for the computation of bounds to routing problems. *Networks 11, 145–164*.

Chudnovsky, Cornuéjols, Liu, Seymour & Vušković (2003). Cleaning for Bergeness. Manuscript. submitted for publication. Available at `http://www.math.princeton.edu/~mchudnov/`.

Chudnovsky, Robertson, Seymour & Thomas (2002). The strong perfect graph theorem. Manuscript. Available at `http://www.math.gatech.edu/~thomas/spgc.html`.

—— (2003). Progress on perfect graphs. *Math. Prog. Series B 97(1–2), 405–422*.

Chudnovsky & Seymour (2003). Recognizing Berge Graphs. Manuscript. submitted for publication. Available at `http://www.math.princeton.edu/~mchudnov/`.

Chung, Füredi, Garey & Graham (1988). On the fractional covering number of hypergraphs. *SIAM J. on Disc. Math. 1, 45–49*.

Chvátal (1975). On Certain Polytopes Associated with Graphs. *J. Comb. Theory 18, 138–154*.

Chvátal (1983). *Linear Programming*. W.H. Freeman and Company, New York.

Chvátal (1985). Star cutsets and perfect graphs. *J. Comb. Theory. Ser. B 39, 189–199*.

Chvátal & Sbihi (1987). Bull-free Berge graphs are perfect. *Graphs and Combin. 3, 127–139*.

Cook & Russell (1978). A Simulation and Statistical Analysis of Stochastic Vehicle Routing with Timing Constraints. *Decision Sci. 9(4), 673–687*.

Cook & Seymour (Eds.) (1990). volume 1 of *DIMACS Ser. in Disc. Math. and Theoretical Comp. Sci.* Am. Math. Soc., Providence, R.I.

Cornuéjols & Cunningham (1985). Compositions for perfect graphs. *Disc. Math. 55, 245–254*.

Cornuéjols, Liu & Vušković (2003). A Polynomial Algorithm for Recognizing Perfect Graphs. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pp. 20–27., Cambridge, MA.

Cornuéjols & Novick (1989). Ideal 0,1 Matrices. Technical Report Mgmt. Sci. Res. Rep. #537, Grad. School of Ind. Admin., Carnegie Mellon Univ., Pittsburgh, PA 15213, USA.

Cornuéjols & Sassano (1989). On the 0,1 Facets of the Set Covering Polytope. *Math. Prog. 43, 45–55*.

Cziszár, Körner, Lovász, Marton & Simonyi (1990). Entropy splitting for antiblocking corners and perfect graphs. *Combinatorica 10, 27–40*.

Dantzig & Fulkerson (1954). Minimizing the number of tankers to meet a fixed schedule. *Nav. Res. Logistics Q. 1, 217–222*.

Dell'Amico, Fischetti & Toth (1993). Heuristic Algorithms for the Multiple Depot Vehicle Scheduling Problem. *Mgmt. Sci. 39(1), 115–125*.

Denardo & Fox (1979). Shortest-route methods: 1. Reaching, pruning, and buckets. *Op. Res. 27, 161–186*.

Desrochers (1986). An algorithm for the shortest path problem with resource constraints. Publication 421A, Centre de recherche sur les transports, Montréal.

Desrochers, Desrosiers & Solomon (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations Reserach 40(2), 342–354*.

Desrochers, Desrosiers & Soumis (1982). Routes sur un réseau espace-temps. *Rech. Opér. 3, 28–44*.

—— (1984). Routing with time windows by column generation. *Networks 14, 545–565*.

—— (1985). Optimal urban bus routing with scheduling flexibilities. In Thoft (Ed.), *Lecture Notes in Control and Information Sciences*, pp. 155–165.

Desrochers & Rousseau (Eds.) (1992). *Computer-Aided Transit Scheduling*, Lecture Notes in Economics and Mathematical Systems. Springer Verlag.

Desrochers & Soumis (1988a). A generalized permanent labelling algorithm for the shortest path problem with time windows. *INFOR 26, 191–212*.

—— (1988b). A reoptimization algorithm for the shortest path problem with time windows. *Europ. J. on OR 35, 242–254*.

Desrosiers, Dumas, Ioachim & Solomon (1991). A Request Clustering Algorithm in Door-to-Door Transportation. Technical Report G-91-50, École des Hautes Études Commerciales de Montréal, Cahiers du GERAD.

Desrosiers, Dumas, Solomon & Soumis (1995). Time Constrained Routing and Scheduling. In Ball et al. [1995], chapter 2, pp. 35–139.

Desrosiers, Dumas & Soumis (1986). A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. *Am. J. Math. Manage. Sci. 6, 301–325*.

Desrosiers, Dumas, Soumis, Taillefer & Villeneuve (1991). An algorithm for mini-clustering in handicapped transport. Cahiers du GERAD G-91-02, École des Hautes Études Commerciales, Montréal.

Desrosiers, Pelletier & Soumis (1983). Plus court chemin avec contraintes d'horaires. *RAIRO, Inf. Théor. 17, 357–377*.

Desrosiers, Sauvé & Soumis (1988). Lagrangian relaxation methods for solving the minimum fleet size multiple traveling salesman problem. *Management Science 34, 1005–1022*.

Dror (1994). Note on the complexity of the shortest path models for column generation in VRPTW. *Op. Res. 42(5), 977–978*.

Duffin (1962). The extremal length of a network. *J. Math. Anal. Appl. 5, 200*.

Dumas, Desrosiers, Gelinas & Solomon (1995). An optimal algorithm for the traveling salesman problem with time windows. *Operations Research 43(2), 367–371*.

Dumas, Desrosiers & Soumis (1989). Large scale multi-vehicle dial-a-ride problems. Cahiers du GERAD G-89-30, École des Hautes Études Commerciales, Montréal.

—— (1991). The pickup and delivery problem with time windows. *Europ. J. on OR 54(1), 7–22*.

Edmonds (1962). Covers and Packings in a Family of Sets. *Bulletin of the Am. Math. Soc. 68, 29–32*.

Edmonds & Fulkerson (1970). Bottleneck extrema. *J. Comb. Theory 8, 299–306*.

Euler, Jünger & Reinelt (1987). Generalizations of Cliques, Odd Cycles and Anticycles and Their Relation to Independence System Polyhedra. *Math. of OR 12(3), 451–462*.

Fiat, Rabani & Ravid (1990). Competitive k–server algorithms. In *Proceedings of 22nd Symposium on Theory of Algorithms*.

Fiat & Woeginger (Eds.) (1998). *Online algorithms: The state of the art*, volume 1442 of *Lecture Notes in Computer Science*. Springer Verlag.

Finke, Claus & Gunn (1984). A two-commodity network flow approach to the travelling salesman problem. In of Oregon [1984], pp. 167–178.

Fischetti & Toth (1989). An additive bounding procedure for combinatorial optimization problems. *Operations Research 37(2), 319–328*.

Fisher, Jörnsten & Madsen (1997). Vehicle Routing With Time Windows: Two Optimization Algorithms. *Op. Res. 45(3), 488–492*.

Ford (1956). Network Flow Theory. Report P-923, The Rand Corporation, Santa Monica, Calif.

Frederickson & Guan (1993). Nonpreemptive ensemble motion planning on a tree. *Journal of algorithms 15(1), 29–60*.

Frederickson, Hecht & Kim (1978). Approximation algorithms for some routing problems. *SIAM J. on Computing 7(2), 178–193*.

Fulkerson (1971). Blocking and Anti-Blocking Pairs of Polyhedra. *Math. Prog. 1, 168–194*.

—— (1972). Anti-blocking Polyhedra. *J. Comb. Theory 12, 50–71*.

—— (1973). On the Perfect Graph Theorem, pp. 69–76. NY: Academic Press.

Garey & Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman and Company.

Garfinkel & Nemhauser (1972). *Integer Programming*. NY, London, Sydney, Toronto: John Wiley & Sons.

Gasparian (1996). Minimal imperfect graphs: A simple approach. *Combinatorica 16(2), 209–212*.

Gélinas, Desrochers, Desrosiers & Solomon (1995). A new branching strategy for time constrained routing problems with application to backhauling. *Ann. Oper. Res. 61, 91–109*.

Gendreau, Laporte & Potvin (1997). Vehicle routing: Modern heuristics. In Aarts & Lenstra [1997], pp. 311–336.

Grötschel, Hauptmeier, Krumke & Rambau (1999). Simulation Studies for the Online Dial-a-Ride Problem. Preprint SC 99-09, Konrad-Zuse-Zentrum Berlin, Berlin.

Grötschel, Lovász & Schrijver (1981). The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica 1, 169–197*.

—— (1984). Polynomial algorithms for perfect graphs. *Annals of Disc. Math. 21, 325–356*.

—— (1988). *Geometric Algorithms and Combinatorial Optimization*. Springer Verlag, Berlin.

Halse (1992). *Modeling and solving complex vehicle routing problems*. PhD thesis, IMSOR, Technical University of Denmark, Lyngby.

Hansen, Jaumard & Poggi de Aragão (1991). Un algorithme primal de programmation linéaire généralisée pour les programmes mixtes. *C. R. Acad. Sci. Paris 313, 557–560*.

Hauptmeier, Krumke & Rambau (1999). The Online Dial-a-Ride Problem under Reasonable Load. Preprint SC 99-08, Konrad-Zuse-Zentrum Berlin, Berlin.

—— (2000). The Online Dial-a-Ride Problem under Reasonable Load. In Bongiovanni et al. [2000], pp. 125–136.

Hochbaum (Ed.) (1997). *Approximation Algorithms for $\mathcal{NP}$-hard Problems*. PWS Publishing Company.

Hoffman & Padberg (1993). Solving Airline Crew-Scheduling Problems by Branch-And-Cut. *Mgmt. Sci. 39, 657–682*.

Houck, Picard, Queyranne & Vemuganti (1980). The travelling salesman problem as a constrained shortest path problem: Theory and computational experience. *Opsearch 17, 93–109*.

Hougardy & Wagler (2003). Perfectness is an Elusive Graph Property. ZIB-Report ZR 02-11, Konrad-Zuse-Zentrum Berlin. Available at `http://www.zib.de`.

Ioachim, Gélinas, Desrosiers & Soumis (1994). A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. Cahiers du GERAD G-94-24, École des Hautes Études Commerciales, Montréal.

Jaw, Odoni, Psaraftis & Wilson (1986). A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Res. 20B, 243–257*.

Kalyanasundaram & Pruhs (1998). The Online Transportation Problem. *SIAM J. on Disc. Math. 13(3), 370–383*.

Kolen, Rinnooy Kan & Trienekens (1987). Vehicle routing with time windows. *Operations Research 35(2), 266–272*.

König (1986). *Theorie der Endlichen und Unendlichen Graphen (German) [Theory of Finite and Infinite Graphs]*, volume 6 of *Teubner Archiv zur Mathematik*. BSB B.G. Teubner Verlagsgesellschaft Leipzig. Distributed by Springer Verlag Wien NY. Reprint.

Kontoravdis & Bard (1995). A GRASP for the Vehicle Routing Problem with Time Windows. *ORSA J. on Comp. 7(1), 10–23*.

Körner (1973). Coding of an information source having ambiguous alphabet and the entropy of graphs. In *Transactions of the 6th Prague conference on information theory*, pp. 411–425., Prague. Academia.

Koskosidis, Powell & Solomon (1992). An optimization-based heuristic for vehicle routing and scheduling with soft time window constraints. *Transp. Sci. 26(2), 69–85*.

Koutsoupias & Papadimitriou (1994). Beyond competitive analysis. In *Proceedings of the 35th Annual IEEE Symposium on the Foundations of Computer Science*, pp. 394–400.

Koutsoupias & Papadimitriou (1995). On the k-server Conjecture. *Journal of the ACM 42(5), 971–983*.

Krumke (2002). *Online Optimization - Competitive Analysis and Beyond*. Habilitationsschrift, Tech. Univ. of Berlin.

Krumke, Laura, Marchetti-Spaccamela, Lipmann, de Paepe, Poensgen & Stougie (2002). Non-Abusiveness Helps: An O(1)-Competitive Algorithm for Minimizing the Maximum Flow Time in the Online Traveling Salesman Problem. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization*, volume 2462 of *Lecture Notes in Computer Science*, pp. 200–214.

Krumke & Rambau (2002). *Online Optimierung*. Tech. Univ. of Berlin. Lecture notes.

Lamatsch (1992). An Approach to Vehicle Scheduling with Depot Capacity Constraints. In Desrochers & Rousseau [1992].

Langevin, Desrochers, Desrosiers, Gélinas & Soumis (1993). A two-commodity flow formulation for the traveling salesman and makespan problems with time windows. *Networks 23, 631–640*.

Laporte (1992). The vehicle routing problem: an overview of exact and approximate algorithms. *Europ. J. on OR 59, 345–358*.

Laporte & Nobert (1987). Exact algorithms for the vehicle routing problem. *ADM 31, 147–184*.

Laurent (1989). A Generalization of Antiwebs to Independence Systems and Their Canonical Facets. *Math. Prog. 45, 97–108*.

Lehman (1979). On the Width-Length Inequality. *Mathematical Programming 17, 403–417*.

—— (1981). The Width-Length Inequality and Degenerate Projective Planes. In Cook & Seymour [1990], pp. 101–105.

Lenstra & Rinnooy Kan (1981). Complexity of Vehicle Routing and Scheduling Problems. *Networks 11, 221–227*.

Lin (1965). Computer solutions of the traveling salesman problem. *Bell System Tech. J. 44, 2245–2269*.

Lin & Kernighan (1973). An efficient heuristic algorithm for the travelling salesman problem. *Operations Research 21, 498–516*.

Lipmann (1999). The Online Traveling Salesman Problem on the Line. Master's thesis, Department of Operations Research, University of Amsterdam, The Netherlands.

Lovász (1971). Normal Hypergraphs and the Perfect Graph Conjecture. *Disc. Math. 2, 253–267*.

Lovász (1979). On the Shannon capacity of a graph. *IEEE Transactions on Information Theory 25, 1–7*.

Lovász & Schrijver (1991). Cones of Matrices and Set-Functions and 0-1 Optimization. *SIAM J. on Opt. 1, 166–190*.

Lütolf & Margot (1962). A catalog of minimally nonideal matrices. *Math. Methods of OR 47(2), 221–241*.

Maculan, Michelon & Plateau (1992). Column-Generation in linear programming with bounding variable constraints and its application in integer programming. Working paper ES-268/92, Federal University of Rio de Janeiro, 21945-970 Rio de Janeiro.

Madsen, Ravn & Rygaard (1995). A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. *Ann. Oper. Res. 60, 193–208*.

Magnanti (1981). Combinatorial optimization and vehicle fleet planning: perspectives and prospects. *Networks 11, 179–214*.

Mahjoub (1988). On the stable set polytope of a series-parallel graph. *Math. Prog. Series A 40(1), 53–57*.

Manasse, McGeoch & Sleator (1988). Competitive algorithms for server problems. In *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing*, pp. 322–333.

McDiarmid (2001). Graph imperfection and channel assignment. In Ramírez Alfonsín & Reed [2001], pp. 215–231.

McDiarmid & Reed (2000). Channel assignment and weighted colouring. *Networks 36, 114–117*.

Mesquita & Paixão (1992). Multiple depot vehicle scheduling problem: A new heuristic based on quasi-assignment algorithms. In Desrochers & Rousseau [1992].

Moore (1957). The shortest path through a maze. In Proceedings [1957].

Nemhauser & Trotter, Jr. (1973). Properties of Vertex Packing and Independence System Polyhedra. *Math. Prog. 6, 48–61*.

Nobili & Sassano (1989). Facets and Lifting Procedures for the Set Covering Polytope. *Math. Prog. 45, 111–137*.

—— (1992). A separation routine for the set covering polytope. In Balas, Cornuéjols & Kannan (Eds.), *Integer Programming and Combinatorial Optimization.*, Proc. of the 2nd Int. IPCO Conf., pp. 201–219., Pittsburgh, USA. Carnegie Mellon University, University Printing and Publications.

Or (1976). *Travelling Salesman-Type Combinatorial Problems and Their Relation to the Logistics of Regional Blood Banking*. Ph.D. dissertation, Northwestern University, Evanston, Ill.

Padberg (1973a). On the Facial Structure of Set Packing Polyhedra. *Math. Prog. 5, 199–215*.

—— (1973b). Perfect Zero-One Matrices. *Math. Prog. 6, 180–196*.

—— (1976). Almost Integral Polyhedra Related to Certain Combinatorial Optimization Problems. *Lin. Algebra and Its Applications 15, 69–88*.

—— (1977). On the Complexity of Set Packing Polyhedra. *Annals of Disc. Math. 1, 421–434*.

—— (1979). Covering, Packing, and Knapsack Problems. *Annals of Disc. Math. 4, 265–287*.

—— (1993). Lehman's Forbidden Minor Characterization of Ideal 0-1 Matrices. *Disc. Math. 111(1–3), 409–420*.

Pallottino (1984). Shortest-path methods: Complexity, interrelations and new propositions. *Networks 14, 257–267*.

Pape (1980). Algorithm 562. Shortest path lengths. *ACM Trans. Math. Softw. 6, 450–455*.

Potvin, Lapalme & Rousseau (1989). A generalized K-opt exchange procedure for the MTSP. *INFOR 27, 474–481*.

Potvin & Rousseau (1993). A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *Europ. J. on OR 66(3), 331–340*.

Proceedings (1957). *Proc. Int. Symp. on the Theory of Switching, Part II*, Cambridge, Mass. Harvard Univerity Press.

Proskurowski, Farley & Chinn (Eds.) (1984). *The second West Coast conference on combinatorics, graph theory, and computing*, number 41 in Congressus Numerantium, Egene, Oregon. University of Oregon, Utilitas Mathematica Publishing Incorporated.

Psaraftis (1983). An exact algorithm for the single-vehicle many-to-many dial-a-ride problem with time windows. *Transp. Sci. 17, 351–357*.

Psaraftis, Solomon, Magnanti & Kim (1990). Routing and scheduling on a shoreline with release times. *Mgmt. Sci. 36, 212–223*.

Pulleyblank (1979). Minimum node covers and 2-bicritical graphs. *Math. Prog. 17, 91–103*.

Ramírez Alfonsín & Reed (Eds.) (2001). *Perfect graphs*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Chichester: John Wiley & Sons.

Reichel & Tison (Eds.) (2000). *Proceedings of the 17th Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *Lecture Notes in Computer Science*. Springer Verlag.

Ribeiro & Soumis (1994). A column generation approach to the multiple-depot vehicle scheduling problem. *Operations Research 42(2), 41–52*.

Roy, Rousseau, Lapalme & Ferland (1984a). Routing and Scheduling for the Transportation of Disabled Persons – The Algorithm. Working paper TP 5596E, Transport Canada, Transport Development Center, Montreal.

—— (1984b). Routing and Scheduling for the Transportation of Disabled Persons – The Tests. Working paper TP 5598E, Transport Canada, Transport Development Center, Montreal.

Russell (1977). An effective heuristic for the M-tour traveling salesman problem with some side conditions. *Op. Res. 25, 517–524*.

—— (1995). Hybrid heuristics for the vehicle routing problem with time windows. *Transp. Sci. 29(2), 156–166*.

Salomon, Solomon, Van Wassenhove, Luk N.; Dumas & Dauzère-Pérès (1997). Solving the discrete lotsizing and scheduling problem with sequence dependent set-up costs and set-up times using the travelling salesman problem with time windows. *Europ. J. on OR 100(3), 494–513*.

Sassano (1989). On the Facial Structure of the Set Covering Polytope. *Math. Prog. 44, 181–202*.

Savelsbergh (1985). Local search for routing problems with time windows. *Annals of Operations Research 4, 285–305*.

—— (1992). *Computer aided routing*. PhD thesis, Centre for Mathematics and Computer Sciene, P.O. Box 4079, 1009 AB Amsterdam, The Netherlands.

Schrijver (2003). *Combinatorial Optimization*, volume 24 of *Algorithms and Combinatorics*. Springer Verlag.

Schulz (1996). *Polytopes and Scheduling*. PhD thesis, Tech. Univ. of Berlin.

Sekiguchi (1985). A note on node packing polytopes on hypergraphs. *OR Letters 2, 243–247*.

Sexton & Bodin (1985a). Optimizing single vehicle many-to-many operations with desired delivery times. I: Scheduling. *Transp. Sci. 19, 378–410*.

—— (1985b). Optimizing single vehicle many-to-many operations with desired delivery times. II: Routing. *Transp. Sci. 19, 411–435*.

—— (1986). The multi-vehicle subscriber dial-a-ride problem. *TIMS Studies Manage. Sci. 26, 73–86*.

Sexton & Choi (1986). Pickup and delivery of partial loads with "soft" time windows. *Am. J. Math. Manage. Sci. 6, 369–398*.

Seymour (1977). The matroids with the max-flow min-cut property. *J. Comb. Theory. Ser. B 23, 189–222*.

Seymour (1990). On Lehman's Width-Length Characterization. In Cook & Seymour [1990], pp. 107–117.

Shannon (1956). The zero error capacity of a noisy channel. *I.R.E. Trans. Inform. Theory IT-2, 8–19*.

Simonyi (2001). Perfect graphs and graph entropy: An updated survey. In Ramírez Alfonsín & Reed [2001], pp. 293–328.

Sleator & Tarjan (1985). Amortized efficiency of list update and paging rules. *Communications of ACM 28, 202–208*.

Solomon, Baker & Schaffer (1988). Vehicle routing and scheduling problems with time window constraints: Efficient implementations of solution improvement procedures. In Assad & Golden [1988], pp. 85–105.

Solomon (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research 35(2), 254–265*.

Thompson & Psaraftis (1993). Cyclic transfer algorithms for multivehicle routing and scheduling problems. *Op. Res. 41(5), 935–946*.

Trotter, Jr. (1975). A Class of Facet Producing Graphs for Vertex Packing Polyhedra. *Discrete Mathematics 12, 373–388*.

Tsitsiklis (1992). Special cases of traveling salesman and repairman problems with time windows. *Networks 22, 263–282*.

Tucker (1972). The strong perfect graph conjecture and an application to a municipal routing problem. In *Graph Theory and App., Proc. Conf.*, volume 303 of *Lecture Notes in Math.*, pp. 297–303. Western Michigan Univ., Springer.

—— (1973). Perfect graphs and an application to optimizing municipal services. *SIAM Rev. 15, 585–590*.

Vestjens (1997). *On-line Scheduling*. Ph.D. thesis, Eindhoven University of Technology, Eindhoven, The Netherlands.

Wagler (2000). *Critical edges in perfect graphs*. Ph.D. dissertation, Tech. Univ. of Berlin, Berlin.

—— (2002). Antiwebs are Rank-Perfect. ZIB-Report SC 02-07, Konrad-Zuse-Zentrum Berlin. Available at `http://www.zib.de`.

Wilson & Colvin (1977). Computer control of the rochester dial-a-ride system. Report R77-31, Dept. of Civil Engineering, M.I.T., Boston, Mass.

Wilson, Sussman, Wang & Higonnet (1971). Scheduling algorithms for dial-a-ride systems. Urban Systems Laboratory Report USL TR-70-13, M.I.T., Boston, Mass.

Wilson & Weissberg (1976). Advanced dial-a-ride algorithms research project: Final report. Report R76-20, Dept. of Civil Engineering, M.I.T., Boston, Mass.

Wolsey (1976). Further Facet Generating Procedures for Vertex Packing Polytopes. *Math. Prog. 11, 158–163*.

Zemel (1978). Lifting the facets of zero-one polytopes. *Mathematical Programming 15, 268–277*.

Ziegler (1998). *Lectures on Polytopes* (Second ed.)., volume 152 of *Graduate Texts in Mathematics*. New York: Springer.

Index