COLUMN GENERATION TECHNIQUES
FOR
PICKUP AND DELIVERY PROBLEMS

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van
de Rector Magnificus, prof. dr. J.H. van Lint,
voor een commissie aangewezen door het College
van Dekanen in het openbaar te verdedigen op
dinsdag 8 november 1994 om 14.00 uur

door
Marc Sol
geboren te Temeuzen

druk: wibro dissertatiedrukkerij, heemerd.
Dit proefschrift is goedgekeurd
door de promotoren
prof.dr. J.K. Lenstra
en
prof.dr. M.W.P. Savelbergh
Preface

This thesis results from the research project 'Design and implementation of algorithms for the pickup and delivery problem,' which was financed by the Stichting voor de Technische Wetenschappen (Technology Foundation), and was carried out at Eindhoven University of Technology. The research focused on a pickup and delivery problem that we encountered at Van Gend & Loos BV in the Netherlands. The main result of the project is the algorithm DRIVE (Dynamic Routing of Independent VEHicles), which is a planning tool that can be embedded in a decision support system at Van Gend & Loos BV.

This project would never have come to a successful ending if I would not have received support from various people. I therefore start this thesis by thanking all people who, in one way or another, have contributed to the project.

I express my gratitude to the people at Van Gend & Loos BV who were directly involved in the development of DRIVE. P. Maier provided the managerial support and gave useful suggestions for improving the 'magic box.' B. Muusers provided me with the necessary test data and was a great help in evaluating the test results. Ton Steysiger indicated most of the side constraints that the routes should satisfy. I also thank Jeanine Hendriks of Logion, whose support was invaluable throughout the project. Furthermore, I thank W.A.M. Aspers, C.G.E. Boender,
M.T. de Jong, A. Philippo and M. Zijlstra, for their constructive remarks during the meetings of the users' committee of the project.

Many of my colleagues in Eindhoven contributed to a pleasant period as a Ph.D. student. In particular, I want to thank Marc Wennink for being an interested brainstorm-partner, Wietse Venema for teaching me many programming skills, Cor Hurkens for helping me with any problem I could think of, and Marjan van den Akker and Rob Vaessens for cooperating in solving the numerous LNMB exercises.

I thank my supervisor Jan Karel Lenstra for giving me the opportunity to start the research project and for teaching me many aspects of combinatorial optimization. I have benefited much from his writing and editing skills. I am greatly indebted to my advisor Martin Savelsbergh. I have learned a lot from his knowledge on vehicle routing problems and integer programming techniques. Working together has been a very nice experience for me.

I thank my parents, who provided all the opportunities for becoming the person I am at present. I always remember that, of all knowledge I have obtained during my life, the first was given to me by them.

Finally, I thank my wife Yvonne. She always supported and encouraged me during the period I was writing this thesis.

Marc Sol

## Contents

Introduction 1

1 The General Pickup and Delivery Problem 7
  1.1 Problem formulation 8
  1.2 Problem characteristics 11
    1.2.1 Transportation requests 11
    1.2.2 Time constraints 12
    1.2.3 Objective functions 13
  1.3 Solution approaches 15
    1.3.1 The static pickup and delivery problem 16
    1.3.2 The dynamic pickup and delivery problem 27
  1.4 Concluding remarks 30

2 The Resource Assignment Problem 31
  2.1 Problem formulation 32
  2.2 The set partitioning formulation 36
  2.3 Comparison of lower bounds 37
  2.4 A branch-and-price algorithm 39
    2.4.1 Column generation 40
    2.4.2 The pricing problem 42
Contents

2.4.3 Lower bounds ............................................. 44
2.4.4 Branching schemes ........................................ 46
2.4.5 Upper bounds ............................................. 51
2.5 Implementing a branch-and-price algorithm .................. 51
  2.5.1 Column management ...................................... 52
  2.5.2 Approximation algorithms for the pricing problem ....... 54
  2.5.3 Identical versus nonidentical resources ................ 55
2.6 Redundancy of columns ....................................... 56
  2.6.1 Identical resources ..................................... 57
  2.6.2 Nonidentical resources .................................. 60
  2.6.3 An alternative pricing problem .......................... 62
2.7 A general approximation algorithm for the resource assignment problem ............................................. 65

3 The Pickup and Delivery Problem with Time Windows .......... 67
  3.1 Problem formulation ......................................... 68
  3.2 The pricing problem ......................................... 69
    3.2.1 An optimization algorithm for the pricing problem ... 70
  3.3 Solving the linear program ................................... 74
    3.3.1 Improving the dynamic programming algorithm ........ 74
    3.3.2 A cheapest insertion algorithm ........................ 75
  3.4 An alternative branching scheme ............................ 77
  3.5 Computational experiments ................................... 78
    3.5.1 Test problems ......................................... 79
    3.5.2 Quality of the lower bound ............................. 81
    3.5.3 Performance of the optimization algorithms ............ 82
    3.5.4 Performance of the approximation algorithms .......... 83
  3.6 Concluding remarks .......................................... 86

4 DRIVE .......................................................... 89
  4.1 Problem characteristics ...................................... 90
    4.1.1 Requests .............................................. 91
    4.1.2 Vehicles ............................................... 91
    4.1.3 Costs ................................................ 92
    4.1.4 Problem size .......................................... 92
  4.2 Current methodology ......................................... 92
  4.3 Envisioned methodology ..................................... 93
  4.4 Handling the dynamics ....................................... 94
    4.4.1 Vehicles .............................................. 94
    4.4.2 Requests ............................................. 94
Introduction

Vehicle routing problems arise in situations in which a set of vehicles is available to serve a set of requests. These vehicles are, for example, trucks, ships, airplanes, or sometimes even people. Requests specify one or more locations that have to be visited by a single vehicle, and various types of side constraints that restrict the way in which these locations can be visited. Depending on the type of routing problem, the vehicle must perform some actions at these locations. Actions consist of, for example, picking up or delivering goods, repairing some equipment at the location, or visiting people. Serving a set of requests by a vehicle involves a certain cost. The objective in a routing problem is to assign the requests to the vehicles and, for each vehicle, to construct a route through the locations of the requests assigned to that vehicle, such that all side constraints are satisfied, and the total cost involved with executing the routes is minimal. When there is only one vehicle that must serve all requests, no assignment has to be performed. In this case the problem is referred to as a single-vehicle problem.

In general, routing problems are subject to various types of side constraints. The most common side constraints, which have been well studied in the past, are capacity constraints and time constraints. Capacity constraints arise when the actions that have to be performed at request locations consist of picking up or delivering some goods and each
vehicle has a capacity which limits the total amount of goods that it can carry at the same time. Time constraints arise when a request specifies time windows, i.e. time intervals in which service at the various locations must take place, or when the vehicles are available only during given time periods. When time constraints are present, the problem is often referred to as a vehicle routing and scheduling problem.

The costs involved with executing a set of routes can be of numerous types. In many situations these costs are dominated by the fixed cost of using a vehicle. In these cases, we obtain a vehicle routing problem in which the objective is to minimize the number of vehicles needed to serve all requests. Other major cost components are travel distance, because of fuel costs, and route duration, when drivers are paid per hour. In systems in which people are transported, some measure of client inconvenience can be incorporated into the costs, in order to obtain routes that are convenient to the passengers.

Vehicle routing and scheduling problems have been widely studied in the last several decades. There are two main reasons for this. The first is that the costs incurred in transportation systems are usually very high. Kearney [30] shows that physical distribution costs account for approximately 16% of the sales value of an item. An important part of these distribution costs are associated with operating vehicles and drivers. This means that by studying vehicle routing and scheduling problems, and applying the results in practice, we may be able to achieve large savings, even if only a small relative improvement is made. The second reason for studying vehicle routing and scheduling problems is that they are interesting from a mathematical point of view. Their intrinsic difficulty has been a challenge for researchers for more than 40 years [10]. For example, the basic single-vehicle routing problem, the traveling salesman problem (TSP), is \( \mathcal{NP} \)-hard. In the TSP each request specifies one location. The problem is to find the cheapest route for the vehicle, i.e. to find the order in which all requests must be served such that the total distance traveled is minimal. When each request also specifies a time window in which its location must be visited, it is even \( \mathcal{NP} \)-complete to decide whether a feasible route for the vehicle exists.

Because most vehicle routing and scheduling problems are \( \mathcal{NP} \)-hard, we cannot find efficient, i.e. polynomial time, optimization algorithms for such problems, unless \( \mathcal{P} = \mathcal{NP} \). Nonetheless a great deal of effort has been spent on the development of optimization algorithms for simplified, though still \( \mathcal{NP} \)-complete, routing problems. Polyhedral techniques have, for example, led to the solution of TSP instances with more
than 4000 requests. For multi-vehicle problems, such problem sizes are, at the moment, far beyond reach of optimization algorithms.

In summary, there is no hope to obtain optimal solutions for practical vehicle routing problems. These problems usually contain too many requests and are subject to many complex side constraints. Hence, most research has been devoted to approximation algorithms. Such algorithms try to find good solutions within a reasonable amount of time. Most progress has been made in the development of local search algorithms. In a local search algorithm, one first constructs some solution, and then iteratively changes this solution into a better solution by performing small modifications. For many single-vehicle routing problems these iterative improvement methods and their variants give satisfactory approximate solutions [31]. Recent results indicate that for multi-vehicle problems the performance of such methods is improving [23].

For multi-vehicle problems no satisfactory general solution approach has been developed yet. Many methods decompose the problem into a problem for assigning requests to vehicles, and single-vehicle routing problems for constructing routes for the individual vehicles. The assignment of requests to vehicles seems to be the most difficult aspect of the problem. First, the size of this problem is large compared to the size of a single-vehicle problem. Second, it is difficult to estimate the cost of assigning a request to a vehicle, because it depends on the other requests that are to be assigned to the vehicle. Another approach is to construct combinations of requests that are then to be served by one vehicle. In order for this approach to work, a good measure of closeness of requests is needed in order to form the combinations. When each request specifies only one location and no side constraints are present, the distance between locations might be a good measure of closeness. In more complicated situations such a measure is far less obvious to find.

In this thesis, we focus on routing and scheduling problems in which each request specifies two or more locations: at least one pickup location (the origin) and at least one delivery location (the destination). One vehicle must pick up the loads at the origins of a request and then, without transshipment at a depot, deliver these loads at the destinations. These routing problems are known as pickup and delivery problems. In these problems we have precedence constraints between locations, because, obviously, a pickup has to precede the corresponding delivery. Pickup and delivery problems arise in many practical situations. Dial-a-ride systems, for example, provide transportation of people. These services are primarily designed for elderly and handicapped people who are unable
to use a car or public transportation. People specify a single origin and a single destination and usually a desired pickup or delivery time. Physical distribution systems also give rise to various types of pickup and delivery problems. These problems can be divided into two categories. In full-truck-load problems, each load entirely fills a vehicle. Each load must be delivered directly after it has been picked up. In less-than-truck-load problems, various loads can be in a vehicle at the same time.

We have used a set partitioning model for the pickup and delivery problem. This model provides a good means to handle the assignment of requests to vehicles. In the set partitioning model, a large set of feasible routes is constructed for each vehicle. From these sets, one route must be chosen for each vehicle, such that each request is served on exactly one route. The set partitioning model led to the development of a branch-and-price algorithm that solves the problem to optimality. In this algorithm we solve the linear programming relaxation of the set partitioning formulation of the problem. Because the set of feasible routes for a vehicle is usually very large, we use column generation to solve this linear program. We then use branch-and-bound to find integral solutions. This approach consecutively subdivides the solution space into smaller subspaces. We designed a branching strategy that enables us to use column generation in each generated subset of the solution space. We have designed the branch-and-price algorithm such that it can easily be turned into a very effective approximation algorithm. In addition, almost all the ideas behind the branch-and-price algorithm can be easily generalized to a wide class of decision problems in which a set of tasks has to be assigned to a set of resources. We have therefore introduced the resource assignment problem.

The main motivation for our research has been the development of a planning tool for a practical pickup and delivery problem that we encountered at Van Geud & Loos BV in the Netherlands. This problem is a dynamic problem. When the vehicles start executing their routes, not all requests that have to be served are known. When new requests become available, they have to be incorporated into the current set of routes. For this purpose we have defined a reoptimization problem, whose solution corresponds to a new set of routes that contain all currently available requests. In order to solve this reoptimization problem, we developed and implemented the algorithm DRIVE (Dynamic Routing of Independent VEHicles), which is based on a branch-and-price algorithm. DRIVE has been designed in order to be used in a decision support system for Van Geud & Loos BV. We have not implemented the entire
decision support system, but our test results have shown that DRIVE may yield a cost decrease at Van Gend & Loos BV of about 5%.

This thesis is organized as follows. In Chapter 1, we present a general model for the pickup and delivery problem, we isolate and discuss some of the characteristics that differentiate pickup and delivery problems from traditional vehicle routing problems, and we give an overview of the literature on pickup and delivery problems. This chapter is based on the paper ‘The general pickup and delivery problem’ by M. Sol and M.W.P. Savelsbergh [51]. In Chapters 2 and 3, we describe the branch-and-price algorithm. The contents of these chapters are a substantial revision and extension of the paper ‘A branch-and-price algorithm for the pickup and delivery problem with time windows’ by M. Sol and M.W.P. Savelsbergh [52]. In Chapter 2, we present the general algorithm for the resource assignment problem. In Chapter 3, we present the aspects of the algorithm that are specific to the pickup and delivery problem with time windows. In Chapter 4, we describe the practical problem at Van Gend & Loos BV and the models and methods that we have designed for DRIVE.
The General Pickup and Delivery Problem

In the general pickup and delivery problem (GPDP) a set of routes has to be constructed in order to satisfy transportation requests. A fleet of vehicles is available to operate the routes. Each vehicle has a given capacity, a start location and an end location. Each transportation request specifies the size of the load to be transported, the locations where it is to be picked up (the origins) and the locations where it is to be delivered (the destinations). Each load has to be transported by one vehicle from its set of origins to its set of destinations without any transshipment at other locations.

Three well-known and extensively studied routing problems are special cases of the GPDP. In the pickup and delivery problem (PDP), each transportation request specifies a single origin and a single destination and all vehicles depart from and return to a central depot. The dial-a-ride problem (DARP) is a PDP in which the loads to be transported represent people. Therefore, we usually speak of clients or customers instead of transportation requests and all load sizes are equal to one. The vehicle routing problem (VRP) is a PDP in which either all the origins or all the destinations are located at the depot.

The GPDP is introduced in order to be able to deal with various
complicating characteristics found in many practical pickup and delivery problems, such as transportation requests specifying a set of origins associated with a single destination or a single origin associated with a set of destinations, vehicles with different start and end locations, and transportation requests evolving in real time.

Many practical pickup and delivery situations are demand responsive, i.e., new transportation requests become available in real-time and are immediately eligible for consideration. As a consequence, the set of routes has to be reoptimized at some point to include the new transportation requests. Observe that at the time of the reoptimization, vehicles are on the road and the notion of depot becomes void.

In this chapter we present a general model that can handle the practical complexities mentioned above, we isolate and discuss some of the characteristics that differentiate pickup and delivery problems from traditional vehicle routing problems, and we give an overview of the literature on pickup and delivery problems. In our survey of the literature, we focus primarily on deterministic models. For a discussion of issues concerning stochastic vehicle routing problems, the reader is referred to the survey paper by Dror, Laporte and Trudeau [17].

1.1 Problem formulation

Let \( N \) be the set of transportation requests. For each transportation request \( i \in N \), a load of size \( \bar{\eta}_i \in \mathbb{N} \) has to be transported from a set of origins \( N^+_i \) to a set of destinations \( N^-_i \). Each load is subdivided as follows:

\[
\bar{\eta}_i = \sum_{j \in N^+_i} \eta_j = - \sum_{j \in N^-_i} \eta_j,
\]

i.e., positive quantities for pickups and negative quantities for deliveries. Define \( N^+ := \cup_{i \in N} N^+_i \) as the set of all origins and \( N^- := \cup_{i \in N} N^-_i \) as the set of all destinations. Let \( V := N^+ \cup N^- \). Furthermore, let \( M \) be the set of vehicles. Each vehicle \( k \in M \) has a capacity \( Q_k \in \mathbb{N} \), a start location \( k^+ \), and an end location \( k^- \). Define \( M^+ := \{ k^+ | k \in M \} \) as the set of start locations and \( M^- := \{ k^- | k \in M \} \) as the set of end locations. Let \( W := M^+ \cup M^- \).

For all \( i, j \in V \cup W \) let \( d_{ij} \) denote the travel distance, \( t_{ij} \) the travel time, and \( c_{ij} \) the travel cost. Note that the dwell time at origins and destinations can be easily incorporated in the travel time and therefore will not be considered explicitly.

**Definition 1** A pickup and delivery route \( R_k \) for vehicle \( k \) is a directed route through a subset \( V_k \subset V \) such that:
1. $R_k$ starts in $k^+$.
2. $(N_i^+ \cup N_i^-) \cap V_k = \emptyset$ or $(N_i^+ \cup N_i^-) \cap V_k = N_i^+ \cup N_i^-$ for all $i \in N$.
3. If $N_i^+ \cup N_i^- \subseteq V_k$, then all locations in $N_i^+$ are visited before all locations in $N_i^-$.
4. Vehicle $k$ visits each location in $V_k$ exactly once.
5. The vehicle load never exceeds $Q_k$.
6. $R_k$ ends in $k^-$.

Definition 2 A pickup and delivery plan is a set of routes $\mathcal{R} = \{R_k | k \in M\}$ such that:
1. $R_k$ is a pickup and delivery route for vehicle $k$, for each $k \in M$.
2. $\{V_k | k \in M\}$ is a partition of $V$.

Define $f(\mathcal{R})$ as the price of plan $\mathcal{R}$ corresponding to a certain objective function $f$. We can now define the general pickup and delivery problem as the problem:

$$\min \{f(\mathcal{R}) | \mathcal{R} \text{ is a pickup and delivery plan.} \}$$

The special cases of the GPDP mentioned above can be characterized as follows:

The pickup and delivery problem

$|W| = 1$ and $|N_i^+| = |N_i^-| = 1$ for all $i \in N$. In this case we define $i^+$ as the unique element of $N_i^+$ and $i^-$ as the unique element of $N_i^-$. 

The dial-a-ride problem

$|W| = 1$ and $|N_i^+| = |N_i^-| = 1$ and $\bar{q}_i = 1$ for all $i \in N$.

The vehicle routing problem

$|W| = 1$, $|N_i^+| = |N_i^-| = 1$ for all $i \in N$, and $N^+ = W$ or $N^- = W$.

Although the literature merely covers pickup and delivery problems with $|N_i^+| = |N_i^-| = 1$, in many practical situations $|N_i^+| > 1$ or $|N_i^-| > 1$. 
In some of these situations, a transportation request with \(|N_i^+| > 1\) or \(|N_i^-| > 1\) can be decomposed into several, independent requests with a single pickup point and a single delivery point. However, in many other situations, a request with multiple pickup or delivery points has to be served by a single vehicle and therefore cannot be decomposed.

Although we are not aware of any real-life applications where requests occur with both \(|N_i^+| > 1\) and \(|N_i^-| > 1\), this case is part of the GPDP definition for the sake of symmetry.

To formulate the GPDP as a mathematical program, we introduce four types of variables: \(z_i^k (i \in N, k \in M)\) equal to 1 if transportation request \(i\) is assigned to vehicle \(k\) and 0 otherwise, \(x_{ij}^k \ (i, j) \in (V \times V) \cup \{(j, j^+)|j \in V\} \cup \{(j, j^-)|j \in V\}, k \in M)\) equal to 1 if vehicle \(k\) travels from location \(i\) to location \(j\) and 0 otherwise, \(D_i \ (i \in V \cup W)\), specifying the departure time at vertex \(i\), and \(y_j \ (i \in V \cup W)\), specifying the load of the vehicle arriving at vertex \(i\). Define \(q_k = 0\) for all \(k \in M\).

The problem is now to minimize \(f(x)\)

subject to

\[
\sum_{k \in M} z_i^k = 1 \quad (i \in N) \tag{1}
\]

\[
\sum_{j \in V} x_{ij}^k = \sum_{j \in V} x_{ji}^k = z_i^k \quad (i \in N, l \in N_i^+ \cup N_i^-, k \in M) \tag{2}
\]

\[
\sum_{j \in V} x_{ij}^k + x_{ji}^k = 1 \quad (k \in M) \tag{3}
\]

\[
\sum_{j \in V} x_{ij}^k + x_{ji}^k = 1 \quad (k \in M) \tag{4}
\]

\[
D_{p+} = 0 \quad (p \in N, q \in N_i^-) \tag{5}
\]

\[
x_{ij}^k = 1 \Rightarrow D_i + t_{ij} \leq D_j \quad (i, j \in V \cup W, k \in M) \tag{6}
\]

\[
y_i = 0 \quad (i \in N, l \in N_i^+ \cup N_i^-) \tag{7}
\]

\[
y_i = 0 \quad (i \in N, l \in N_i^+ \cup N_i^-) \tag{8}
\]

\[
z_i^k \in \{0, 1\} \quad (i \in N, k \in M) \tag{9}
\]

\[
z_i^k \in \{0, 1\} \quad (i \in N, k \in M) \tag{10}
\]

\[
D_i \geq 0 \quad (i \in V \cup W) \tag{11}
\]

\[
y_i \geq 0 \quad (i \in V \cup W) \tag{12}
\]

Constraint (1) ensures that each transportation request is assigned to exactly one vehicle. By constraint (2) a vehicle only enters or leaves a location \(i\) if it is an origin or a destination of a transportation request assigned to that vehicle. Constraints (3) and (4) make sure that each
vehicle starts and ends at the correct place. Constraints (5), (6), (7) and (13) together form the precedence constraints. Constraints (8), (9), (10) and (14) together form the capacity constraints.

1.2 Problem characteristics

1.2.1 Transportation requests

A very important characteristic of routing problems is the way in which transportation requests become available. In a static situation, all requests are known at the time the routes have to be constructed. In a dynamic situation, some of the requests are known at the time the routes have to be constructed and the other requests become available in real time during execution of the routes. Consequently, in a dynamic situation, when a new transportation request becomes available, at least one route has to be changed in order to serve this new request. Most vehicle routing problems are static, whereas most pickup and delivery problems are dynamic.

In practice, a dynamic problem is often solved as a sequence of static problems. This method is commonly referred to as an on-line method. In its simplest form, the current set of routes is updated each time a new request becomes available. However, it is usually possible and beneficial to buffer incoming requests and only update the current set of routes if the buffer size exceeds a certain preset value. Another important issue concerning on-line algorithms is how to incorporate information that may be known about the spatial or time distribution of future requests.

A depot is another important concept in routing problems. In the routing literature, the depot is usually the place where vehicles start and end their routes. Since most pickup and delivery problems are dynamic, often with a long planning horizon, the concept of a depot vanishes. Drivers sleep at the last location they visited or at the first location they have to visit the next day. Even for problems with a short planning horizon, such as a single day, where vehicles start and end at a central depot, a demand responsive situation leads to problems without depots. When new transportation requests become available and the current set of routes has to be updated, the vehicles are spread out over the planning area.

The general pickup and delivery model is well suited for dealing with the subproblems that occur in dynamic, demand responsive, routing problems.
1.2.2 Time constraints

Apart from the vehicle capacity constraints and the intrinsic precedence constraints related to pickup and delivery, side constraints related to time arise in almost every practical pickup and delivery situation. Although time constraints have become an integral part of models for vehicle routing problems (for recent surveys on the vehicle routing problem with time windows see Desrochers, Lenstra, Savelsbergh, and Soumis [12] and Solomon and Desrochers [53]), they play an even more prominent role in pickup and delivery problems. Among other reasons, because the most studied pickup and delivery problem is the dial-a-ride problem, which deals with the transportation of people who specify desired pickup or delivery times.

The presence of time constraints complicates the problem considerably. If there are no time constraints, finding a feasible pickup and delivery plan is trivial: arbitrarily assign transportation requests to vehicles, arbitrarily order the transportation requests assigned to a vehicle and process each transportation request separately. In the presence of time constraints the problem of finding a feasible pickup and delivery plan is \( \mathcal{NP} \)-hard. Consequently, it may be difficult to construct a feasible plan, especially when time constraints are restrictive. On the other hand, an optimization method may benefit from the presence of time constraints, since the solution space may be much smaller.

Time constraints related to transportation requests

For each \( i \in V \cup W \) a time window \([c_i, d_i]\) is introduced denoting the time interval in which service at location \( i \) must take place. Given a pickup and delivery plan and departure times of the vehicles, the time windows define for each \( i \in V \) the arrival time \( A_i \) and the departure time \( D_i \). Note that \( D_i = \max \{ A_i, c_i \} \). If \( A_i < c_i \), then the vehicle has to wait at location \( i \).

Next to the explicit time windows mentioned above there also exist implicit time windows. Implicit time windows originate from controlling customer inconvenience. In dial-a-ride systems, people usually specify either a desired delivery time \( \bar{A}_{i,-} \) or a desired pickup time \( \bar{D}_{i,+} \). Because people do not want to be late at their destination, in a pickup and delivery plan the actual delivery time \( A_{i,-} \) must satisfy \( A_{i,-} \leq \bar{A}_{i,-} \). Analogously, we require \( D_{i,+} \geq \bar{D}_{i,+} \). This defines half open time windows. To prevent clients from being served long before (after) their desired
delivery (pickup) time, we can either construct closed time windows or take an objective function that penalizes deviations from the desired service time. Closed time windows can, for example, be constructed by defining a maximum deviation from the desired pickup or delivery time.

Another problem in which customer inconvenience can be controlled by time windows emerges from a demand responsive, immediate request, dial-a-ride system. In such a system, clients that request service want to be picked up as soon as possible. If client $i$ has a pickup location that is geographically far away, this client will be scheduled last on a vehicle route. This situation will not change even when new clients request service. To prevent the client from suffering indefinite deferment, a closed pickup time window $[0, l_{+}]$ is defined where $l_{+}$ is an input to the system.

Other time constraints that originate from customer inconvenience restrictions in dial-a-ride systems are maximum ride time restrictions, i.e., a bound on the time a client is in the vehicle, and deadhead restrictions, i.e., no waiting time is allowed when a client is in the vehicle. Deadhead restrictions introduce the concept of schedule blocks, i.e., working periods between two successive slack periods.

Time constraints related to vehicles

Vehicles are usually not available all day. Drivers have to eat and sleep and vehicles are subjected to service plans. These constraints can be modeled as time windows for vehicles. Typically a vehicle has multiple time windows defining all the periods in which it is available.

1.2.3 Objective functions

A wide variety of objective functions is found in pickup and delivery problems. The most common ones are discussed below.

First, we present objective functions related to single-vehicle pickup and delivery problems.

Minimize duration

The duration of a route is the total time a vehicle needs to execute the route. Route duration includes travel times, waiting times, loading and unloading times, and break times.
Minimize completion time

The completion time of a route is the time that service at the last location is completed. In case the start time of the vehicle is fixed at time zero, the completion time coincides with the route duration.

Minimize travel time

The travel time of a route refers to the total time spent on actual traveling between different locations.

Minimize route length

The length of a route is the total distance traveled between different locations.

Minimize client inconvenience

In dial-a-ride systems, client inconvenience is measured in terms of pickup time deviation, i.e., the difference between the actual pickup time and the desired pickup time, delivery time deviation, i.e., the difference between the desired delivery time and the actual delivery time, and excess ride time, i.e., the difference between the realized ride time and the direct ride time. In demand responsive situations where clients request immediate service, i.e., as soon as possible, the difference between the time of pickup and the time of request placement may also contribute to the definition of client inconvenience. Different kinds of functions, linear as well as nonlinear, have been proposed to model client inconvenience.

Second, we present objective functions related to multi-vehicle pickup and delivery problems.

Minimize the number of vehicles

This function is almost always used in dial-a-ride systems, combined with one of the above functions to optimize the single-vehicle subproblems. Dial-a-ride systems are normally highly subsidized systems for the transportation of the elderly and handicapped. Therefore the objective is to minimize cost (mostly together with customer inconvenience). Because drivers and vehicles are the most expensive parts in a dial-a-ride system, minimizing the number of vehicles to serve all requests is usually the main objective.
Maximize profit

This function, which can use all of the above functions, can be used in a system where the dispatcher has the possibility of rejecting a transportation request when it is unfavorable to transport the corresponding load. Note that, for example, in a dial-a-ride system, rejecting a transportation request is not allowed. A model based on this objective function should not only incorporate the costs, but also the revenues associated with the transportation of loads.

For dynamic pickup and delivery problems it is not clear what kind of objective functions should be used. In a demand responsive environment pickup and delivery routes may be open ended. Therefore, objectives such as duration, completion time, and travel time have no clear meaning. Intuitively, an objective function for dynamic problems should emphasize decisions that affect the near future more than decisions regarding the remote future.

Note that if a dynamic problem is solved as a sequence of static problems, the objective function for the static subproblem does not necessarily have to be equal to the objective function for the dynamic problem. The objective function for the static subproblem may reflect some knowledge or anticipation of future requests.

1.3 Solution approaches

In this section, we review the literature on pickup and delivery problems. The survey is organized as follows. The class of pickup and delivery problems has been divided into static and dynamic problems, since their characteristics as well as their solution approaches differ considerably. Within either of these classes, we distinguish single-vehicle and multi-vehicle problems. Obviously, in the single-vehicle PDP all transportation requests are handled by the same vehicle, whereas in the multi-vehicle PDP the transportation requests have to be divided over the set of vehicles. Assigning transportation requests to vehicles in the PDP is much more difficult than assigning transportation requests to vehicles in the VRP. In the VRP, all the origins of transportation requests are located at the depot. Therefore, transportation requests with geographically close destinations are likely to be served by the same vehicle. In the PDP, geographically close destinations may have origins that are geographically far apart and we cannot conclude that they are
likely to be served by the same vehicle.

For each of the resulting subclasses, one or more papers are discussed. The level of detail depends on the originality, viability and importance of the described solution approach. A performance evaluation of the solution methods is only provided if such information is present in the corresponding paper. Furthermore, we do not cover iterative improvement methods in any detail. For these the interested reader is referred to Gendreau, Laporte and Potvin [23], and Kindervater and Savelsbergh [31].

1.3.1 The static pickup and delivery problem

The static single-vehicle pickup and delivery problem

The static single-vehicle pickup and delivery problem is probably the most studied variant of the PDP. First of all because the dial-a-ride problem belongs to this class, but also because it appears as a subproblem in multi-vehicle pickup and delivery problems. We discuss solution approaches for problems with and without time windows separately.

(A) The static 1-PDP without time windows

Optimization

Psaraftis [40] considers immediate request dial-a-ride problems. In these problems, every client requesting service wishes to be served as soon as possible. The objective is to minimize a weighted combination of the time needed to serve all clients and the total degree of ‘dissatisfaction’ clients experience until their delivery. Dissatisfaction is assumed to be a linear function of the time each client waits to be picked up and of the time he spends riding in the vehicle until his delivery. This leads to the following objective function:

$$\min w_1 T + w_2 \sum_{i \in N} (\alpha W_i + (2 - \alpha) R_i),$$

where $T$ denotes the route length, $W_i$ the waiting time of client $i$ from the departure time of the vehicle until the time of pickup, $R_i$ the riding time of client $i$ from pickup to delivery, and $0 \leq \alpha \leq 2$.

The problem is solved using a straightforward dynamic programming algorithm. The state space consists of vectors $(L, k_1, k_2, \ldots, k_n)$, where $L$ denotes the location currently being visited ($L = 0$ at the starting location, $L = i$ at the origin of client $i$ and $L = i + n$ at the destination of client $i$) and $k_i$ denotes the status of client $i$ ($k_i = 3$ if client $i$ has
not been picked up, \( k_i = 2 \) if client \( i \) has been picked up but has not been delivered and \( k_i = 1 \) if client \( i \) has been delivered). Starting with state vector \((0, 3, 3, \ldots, 3)\) the algorithm explores new states preserving feasibility of the partial routes constructed so far. The algorithm has a time complexity of \( O(n^2 3^n) \), where \( n = |N| \), and can solve problems with up to 10 clients, i.e., 20 locations.

Kalantari, Hill and Arora [29] propose a method based on the branch and bound algorithm for the TSP developed by Little, Murty, Sweeney and Karel [32]. The method handles precedence constraints by precluding in each branch all the arcs that violate the active precedence constraints. Problems with up to 18 clients were solved with this algorithm.

Fischetti and Toth [21] develop an additive bounding procedure that can be used in a branch-and-bound algorithm for the TSP with precedence constraints, i.e., a single-vehicle dial-a-ride problem without capacity constraints. The idea behind additive bounding is to use several bounds for a problem type additively rather than separately. Let \( L^{(k)} (k = 1, \ldots, r) \) be a procedure which, when applied to problem \( P \) with cost vector \( c \), returns a lower bound \( \delta^{(k)} \) and a residual cost vector \( c^{(k)} \), such that \( \delta^{(k)} + c^{(k)} x \leq cx \) for all feasible \( x \). Apply \( L^{(1)} \) to problem \( P \) with cost vector \( c \) and subsequently apply \( L^{(k)} \) to problem \( P \) with cost vector \( c^{(k-1)} \) (\( k = 2, \ldots, r \)). Then \( \sum_{k=1}^{r} \delta^{(k)} + c^{(k)} x \leq cx \) for all feasible \( x \). So \( \sum_{k=1}^{r} \delta^{(k)} \) is a lower bound for \( P \).

The bounds used by Fischetti and Toth for the TSP with precedence constraints are based on the assignment relaxation, the shortest spanning 1-arborescence relaxation, disjunctions, and on variable decomposition, in that order.

**Approximation**

Stein [54] presents a probabilistic analysis of a simple approximation algorithm for the single-vehicle dial-a-ride problem without capacity constraints. The algorithm constructs a TSP tour through all origins and a TSP tour through all the destinations and then concatenates them.

Stein shows that if \( n \) origin-destination pairs are randomly chosen in a region of area \( a \), using a uniform distribution, there exists a constant \( b \) such that the length \( y'_n \) of the tour constructed by the algorithm satisfies

\[
\lim_{n \to \infty} \frac{y'_n}{\sqrt{n}} = 2b\sqrt{a} \quad \text{with probability 1,}
\]
where \( b \) is the constant that appears in the theorem of Beardwood, Halton and Hammersley [3] on the length of an optimal traveling salesman tour in the Euclidean plane. Recent experiments by Johnson [26] indicate that \( b \approx 0.773 \). Stein also proves that if \( n \) origin-destination pairs are randomly chosen in a region of area \( A \), using a uniform distribution, the length \( y_n^* \) of the optimal tour satisfies

\[
\lim_{n \to \infty} \frac{y_n^*}{\sqrt{n}} = 1.89b\sqrt{A} \text{ with probability 1.}
\]

This implies that the algorithm has an asymptotic performance bound of 1.86 with probability 1.

Psarafitis [42] presents a worst-case analysis of a simple two-phase approximation algorithm for the single-vehicle dial-a-ride problem in the plane. In the first phase, a TSP tour through all locations is constructed. In the second phase, this tour is traversed clock-wise, beginning at the start location of the vehicle and skipping any location that has been visited before, any origin where the pickup of a client would violate the capacity constraints, and any destination whose origin has not yet been visited, until a feasible dial-a-ride solution has been obtained.

Psarafitis shows that if there are no capacity constraints, the TSP tour has to be traversed at most twice. Furthermore, if in addition to the absence of capacity constraints the triangle inequality holds and Christofides' heuristic [7] is used to construct the TSP tour, the algorithm has a worst case ratio of 3.

Fiala Timlin and Pulleyblank [20] consider a slightly different problem. There are groups of customers, each group with its own priority. Customers have to be served in order of increasing priority. Fiala Timlin and Pulleyblank develop an algorithm based on insertion techniques and iterative improvement methods.

(B) The static 1-PDP with time windows

Optimization

Psarafitis [11] modifies the dynamic programming algorithm discussed above to solve the static single-vehicle dial-a-ride problems with time windows. The major difference between the new and the original algorithm is the use of forward instead of backward recursion. Time windows
are handled by the elimination of time infeasible states. The new algorithm still has a time complexity of $O(n^2 3^n)$.

Desrosiers, Dumas and Soumis [14] also present a dynamic programming approach. Their algorithm uses states $(S,i)$ with $S \subseteq V$ and $i \in V$. State $(S,i)$ is defined only if there exists a feasible path that visits all nodes in $S$ and ends in $i$. Elaborate state elimination criteria, based not only on $S$, but also on the state $(S,i)$, are used to reduce the state space. These elimination criteria are very effective when the time windows are tight and the vehicle capacity is small. The algorithm has been tested successfully on problems with up to 40 transportation requests. When capacity constraints are rather tight, then, despite the fact that a dynamic programming algorithm is exponential, the running time of this algorithm seems to increase only linearly with problem size.

**Approximation**

Sexton and Bodin [49, 50] consider the dial-a-ride problem with desired delivery times specified by the clients. The objective is to minimize client inconvenience, which is defined as a weighted combination of delivery time deviation and excess ride time.

The presented solution approach applies Benders decomposition to a mixed 0-1 nonlinear programming formulation, which separates the routing and scheduling component.

The concept of space-time separation indicators between tasks plays an important role in the algorithm. Such an indicator measures the travel time between locations at which the tasks are performed, i.e., their spatial separation, and the difference between the latest feasible times at which they can be performed, i.e., their temporal separation.

Let $\overline{A}_k$ be the desired delivery time of client $k$. The latest possible pickup time $\overline{D}_{k^+}$ is then defined as $\overline{D}_{k^+} := \overline{A}_k - t_{k^{+}}$. The space-time separation indicators between tasks $i$ and $j$ are now defined as follows:

$$
s_{i,j^+} = t_{i,j^+} + \overline{D}_{j^+} - \overline{D}_i
$$

$$
s_{i,j^-} = t_{i,j^-} + \overline{A}_j^- - \overline{D}_i
$$

$$
s_{i,j^+} = t_{i,j^+} + \overline{D}_{j^+} - \overline{A}_i
$$

$$
s_{i,j^-} = t_{i,j^-} + \overline{A}_j^- - \overline{A}_i
$$
The algorithm has been tested on real-life problem instances with up to 20 clients. The solutions produced by the algorithm were compared with the solutions used in practice and the results were encouraging.

Van der Bruggen, Lenstra and Schuur [6] develop a solution procedure based on the Lin-Kernighan variable-depth local search method for the TSP. They use the techniques introduced by Savelsbergh [46, 47] to handle time windows and precedence constraints efficiently. The algorithm has a construction and an improvement phase. An initial route is obtained by visiting the locations in order of increasing centers of their time window, i.e., \( (c_i + l_i)/2 \) for location \( i \), taking precedence and capacity constraints into account. The resulting route may be time infeasible. Using an objective function measuring total infeasibility, the route is made feasible by iterative improvement methods. The same procedures, with a different objective function, are then applied to find a better feasible route. The method has been tested on problem instances with up to 50 clients. For all these instances the optimal solutions were known and the values of the solutions produced by the method were always within one percent of the optimal value; in many the method produced the optimal solution.

The static multi-vehicle pickup and delivery problem

(A) The static m-PDP without time windows

**Approximation**

Cullen, Jarvis and Ratliff [8] propose an interactive approach for the multi-vehicle dial-a-ride problem with a homogeneous fleet, i.e., equal vehicle capacities. The problem is decomposed into a clustering part and a chaining part. Both parts are solved in an interactive setting, i.e., man and machine cooperate to obtain high quality solutions. The algorithmic approach in both parts is based on set partitioning and column generation.

A cluster consists of a seed arc and a set of clients assigned to this seed arc. The total number of clients assigned to a seed arc may not exceed the vehicle capacity \( Q \). Let \((u^+, u^-)\) denote the seed arc of the cluster and let \( S \) denote the set of clients assigned to this seed arc. The cost \( c \) of serving this cluster is approximated by \( c = 2 \sum_{i \in S} d_{u^+i} + d_{u^-i} - 2 \sum_{i \in S} d_{i, u^-} \), i.e., it is assumed that a vehicle serving a cluster starts in \( u^+ \), makes a round trip to each pickup location in the cluster,
travels to $u^-$ and makes a round trip to each delivery location in the cluster.

The clustering problem, i.e., the problem of constructing and selecting clusters to serve all the clients, can be formulated as a set partitioning problem. Let $J$ be the set of all possible clusters, i.e., seed arcs and assignments of clients to seed arcs. For each $j \in J$, let $c_j$ denote the approximate cost of serving the cluster, and for each $i \in N, j \in J$ let $a_{ij}$ be a binary constant indicating whether client $i$ is a member of cluster $j$ or not. Furthermore, introduce a binary decision variable $y_j$ to indicate whether a cluster is selected or not. The clustering problem is now to

minimize $\sum_{j \in J} c_j y_j$
subject to $\sum_{j \in J} a_{ij} y_j = 1$ for all $i \in N$, $y_j \in \{0, 1\}$ for all $j \in J$.

Because the set of all possible clusters is extremely large, a column generation scheme is used to solve the linear programming relaxation of this set partitioning problem.

The master problem is initialized with all columns corresponding to clusters consisting of a single client. The row prices ($\pi_1, \pi_2, \ldots, \pi_n$) are computed and used to define a subproblem to generate columns with negative reduced costs, i.e., clusters that correspond to attractive columns for the master problem. The master problem now heuristically tries to improve the current solution by using (some of) the new columns. Then new row prices are calculated and the subproblem is solved again.

The subproblem that has to be solved is a location-allocation problem. Let $c_{ij} = 2d_{u_j^+ w_i^+} + 2d_{w_i^- u_j^-}$ and $f_j = d_{u_j^+ u_j^-}$. For each $j \in J$ the binary variable $z_{ij}$ denotes whether cluster $j$ is used or not. For each $i \in N, j \in J$ the binary variable $z_{ij}$ denotes whether client $i$ is assigned to cluster $j$ or not. The subproblem has the following mathematical programming formulation:

minimize $\sum_{j \in J} \sum_{i \in N} (c_{ij} - \pi_i) z_{ij} + \sum_{j \in J} f_j x_j$
subject to $\sum_{i \in N} z_{ij} \leq Q x_j$ for all $j \in J$, $\sum_{j \in J} z_{ij} \leq 1$ for all $i \in N$, $z_{ij} \in \{0, 1\}$ for all $i \in N, j \in J$. 
\[ x_j \in \{0, 1\} \quad \text{for all } j \in J. \]

Because of the size of \( J \), the subproblem is still very difficult to solve. Therefore, it is solved approximately: Choose a set of clients to form the seed arcs. With these seed arcs fixed, solve the resulting assignment problem. With these assignments fixed, solve the resulting location problem. Continue alternating between the assignment problem and the location problem for some specified number of iterations, or until no further improvement in the objective function is found.

The clusters in the solution to the clustering problem, together with some promising clusters that did not appear in the final solution, are input to the chaining problem. In the chaining problem a subset of these clusters, partitioning the set of clients, is selected and linked to form pickup and delivery routes. In the set partitioning formulation of this problem the rows correspond to clients again, but columns now correspond to vehicle routes. The column generation procedure links a subset of the seed arcs of the selected clusters. Each linking of seed arcs is then translated into a column for the set partitioning problem by placing a 1 in row \( i \) if client \( i \) is part of one of the clusters in the linking.

Preliminary computational results, based on three problem instances with up to 100 points, show that the method is very effective. For the three instances used in the computational experiment the best known solution was either found or improved.

(B) The static m-PDP with time windows

Optimization

Dumas, Desrosiers and Soumis [19] present a set partitioning formulation for the static pickup and delivery problem with time windows and a column generation scheme to solve it to optimality. The approach is very robust in the sense that it can be adapted easily to handle different objective functions and variants with multiple depots and an inhomogeneous fleet of vehicles.

Let \( \Omega \) be the set of all feasible pickup and delivery routes. For each route \( r \in \Omega \), we denote the cost of the route by \( c_r \). For all \( i \in N \), \( r \in \Omega \) let \( a_{ir} \) be a binary constant indicating whether transportation request \( i \) is served on route \( r \) (\( a_{ir} = 1 \)) or not (\( a_{ir} = 0 \)). Furthermore, introduce a binary variable indicating whether route \( r \) is used in the optimal solution (\( x_r = 1 \)) or not (\( x_r = 0 \)). The static PDP can now be formulated as follows:
minimize \sum_{r \in \Omega} c_r x_r \\
subject to \sum_{r \in \Omega} a_{ir} x_r = 1 \quad \text{for all } i \in N, \\
\sum_{r \in \Omega} x_r = |M|, \\
x_r \in \{0, 1\} \quad \text{for all } r \in \Omega.

The cardinality of \( \Omega \) is much too large to allow an exhaustive enumeration. Therefore, a column generation method is used to solve the linear relaxation of the set partitioning problem. Columns, defining admissible routes, are generated as needed by solving a constrained shortest path problem on a perturbed distance matrix. The perturbation of the distance matrix depends on the dual variables of the set partitioning problem. The lower bound found in this way is usually excellent and constitutes a good starting point for a branch and bound approach to obtain an integral solution. The constrained shortest path problem is solved by dynamic programming. The dynamic programming algorithm employed is essentially the same as the one in Desrosiers, Dumas, and Soumis [14] (see Section 4.1.1), except for the fact that not all transportation requests have to be processed. With this approach problem instances with up to 55 clients and 22 vehicles have been solved.

Approximation

Dumas, Desrosiers and Soumis [18] develop an approximation algorithm for the dial-a-ride problem based on their optimization algorithm discussed above. The basic idea is to create route segments for small groups of clients, called mini-clusters. A minicluster is a segment of a route starting and ending with an empty vehicle. Each minicluster is then treated as a transportation request that entirely fills a vehicle. The optimization algorithm is now applied to this set of transportation requests. This reduces the number of rows in the setpartitioning matrix. The subproblem is now much easier to solve because each transportation request corresponds to a full truck load. Note that in the approach of Cullen, Jarvis and Ratliff [8] clusters cannot be identified with rows because they do not partition the set of clients.

Miniclusters are constructed simply by taking a known pickup and delivery plan, constructed using any existing algorithm, and cutting it into pieces, such that each piece starts and ends with an empty vehicle. In this setup, the approach can best be viewed as an improvement method.

Computational results are reported for problems with up to 31 vehi-
cles and 190 clients (resulting in problems with up to 85 miniclusters). Larger problems, with up to 53 vehicles and 880 clients (up to 282 miniclusters), are solved using a decomposition scheme. First, the problem is decomposed by dividing the planning period into successive time slices with 30 to 40 miniclusters per time slice. Second, the problems associated with the time slices are all solved separately. Finally, the solutions for the successive time slices are combined into one overall solution.

An alternative method to construct mini-clusters is presented by Desrosiers, Dumas, Soumis, Taillfer and Villeneuve [15]. A parallel insertion heuristic creates mini-clusters. The insertion criteria are based on the concept of neighboring requests. Two requests are considered to be neighbors if they satisfy all of the following temporal and spatial restrictions: the time intervals \([t_{i+}, t_{i-}]\) and \([t_{j+}, t_{j-}]\) overlap, \(t_{i+j} + t_{j+} - \leq \alpha t_{i+j} - \) or \(t_{i+j} + t_{i+j} - \leq \alpha t_{j+i} - \) for some constant \(\alpha\), the angle between the arcs \((i^+, i^-)\) and \((j^+, j^-)\) is less than some constant \(\beta\), and the difference in cost between serving \(i\) and \(j\) together and serving \(i\) and \(j\) separately is larger than some constant \(\gamma\).

Jaw, Odoni, Psaraftis and Wilson [27, 28] present two different approaches to the multi-vehicle dial-a-ride problem, in which clients that are to be picked up and delivered have the following types of service constraints: Each client \(i\) specifies either a desired pickup time \(D_{i+}\) or a desired delivery time \(D_{i-}\), and a maximum ride time \(T_i\). The objective is to minimize a combination of customer dissatisfaction and resource usage.

Jaw, Odoni, Psaraftis and Wilson [27] present a three-phase algorithm. The first phase, the grouping phase, decomposes the problem by dividing the time horizon into intervals and then assigning clients to groups according to the time interval into which their desired pickup or delivery time falls. The time intervals are chosen in such a way that it is possible to fully serve a client in two consecutive time intervals. The second phase, the clustering phase, partitions the set of clients in each time group into clusters and assigns a vehicle to each cluster. The number of clusters created is equal to the number of available vehicles. The third phase, the routing phase, constructs a route for each vehicle using a standard approximation algorithm for the single-vehicle dial-a-ride problem.

A more traditional insertion algorithm is presented in Jaw, Odoni, Psaraftis and Wilson [28]. Time windows on both the pickup time and delivery time of a client are defined based on a prescribed tolerance \(U\).
and the specified desired pickup or delivery time. If client \( i \) has specified a desired pickup time, the actual pickup time \( D_{i+} \) should fall within the time window \([\bar{D}_{i+}, \bar{D}_{i+} + U]\); if he has specified a desired delivery time, the actual delivery time \( A_{i-} \) should fall within the window \([A_{i-} - U, A_{i-}]\). Moreover, his actual travel time should not exceed his maximum ride time: \( A_{i-} - D_{i+} \leq \bar{T}_i \). Note that this information suffices to determine two time windows \([e_{i+}, l_{i+}]\) and \([e_{i-}, l_{i-}]\) for each customer \( i \). Finally, waiting time is not allowed when the vehicle is carrying passengers.

The selection criterion is simple: customers are selected for insertion in order of increasing \( e_{i+} \). The insertion criterion is as follows: among all feasible points of insertion of the customer into the vehicle schedules, choose the cheapest; if no feasible point exists, introduce an additional vehicle.

For the identification of feasible insertions, the notion of an active period is introduced. This is a period of time a vehicle is active between two successive periods of slack time. For a given route, let \( \bar{A}_1, \bar{A}_2, \ldots \) be the arrival times at the consecutive locations visited on the route. For each visit to an address \( i \) during an active period, we define the following possible forward and backward shifts:

\[
P_i^- = \min \{ \min_{j \leq i} \{ \bar{A}_j - e_j \}, \sigma \},
\]

\[
P_i^+ = \min_{j \leq i} \{ l_j - \bar{A}_j \},
\]

\[
S_i^- = \min_{j \geq i} \{ \bar{A}_j - e_j \},
\]

\[
S_i^+ = \min_{j \geq i} \{ l_j - \bar{A}_j \}, \sigma \},
\]

where \( \sigma \) and \( \sigma \) are the durations of the slack periods immediately preceding and following the active period in question. \( P_i^- (P_i^+) \) denotes the maximum amount of time by which every stop preceding but not including \( i \) can be advanced (delayed) without violating the time windows, and \( S_i^- (S_i^+) \) denotes the maximum amount of time by which every stop following but not including \( i \) can be advanced (delayed). These quantities thus indicate how much each segment of an active period can be displaced to accommodate an additional customer. Once it is established that some way of inserting the pickup and delivery of customer \( i \) satisfies the time window constraints, it must be ascertained that it satisfies the maximum travel time constraints.
Psaraftis [43] compares and tests these two approaches and concludes that the three-phase algorithm can be best used as a fast planning tool or as a device to produce good starting solutions in an operational situation, whereas the insertion algorithm can form the basis of an operational scheduling system that would assist the dispatcher in the actual execution of the schedule. These algorithms can handle fairly large problem instances effectively and efficiently. Solutions are reported for problem instances with up to 2600 customers and 20 simultaneously active vehicles.

Bodin and Sexton [5] extend their algorithm for the single-vehicle dial-a-ride problem with desired delivery times to a two-phase algorithm for the the multi-vehicle dial-a-ride problem with desired delivery times. In the first phase, the clients are assigned to vehicles and a single-vehicle dial-a-ride problem is solved to construct an initial feasible solution. In the second phase, the algorithm attempts to reassign clients with bad service to other vehicles in order to find a better solution. This swapping of clients is done in the following way: the algorithm passes repeatedly through the client list, each time reassigning clients with a client inconvenience larger than some threshold. This threshold depends on the pass the algorithm is on. The algorithm has been tested on problems with up to 85 clients and 7 vehicles.

The static full-truck-load pickup and delivery problem

The full-truck-load problem is the special case of the GPDP in which each load has to be transported directly from its origin to its destination, i.e., \(|N^+_i| = |N^-_i| = 1\) for all \(i \in N\), \(\bar{q}_i = 1\) for all \(i \in N\), and \(Q_k = 1\) for all \(k \in M\). In these problems, a transportation request is usually called a trip.

The full-truck-load problem is an interesting special case since it can be easily transformed into an asymmetric TSP in the single-vehicle case and an \(m\)-TSP in the multi-vehicle case. The set of vertices corresponds to the set of transportation requests, and the distance between two vertices \(i, j\) is taken to be \(d_{i-j}\).

We discuss two papers that address a version of the full-truck-load problem in which the number of physical locations is small with respect to the number of transportation requests, i.e., each physical location is origin or destination of many requests.

Ball, Golden, Assad and Bodin [1] present two different route first-
cluster second approximation procedures for the static full-truck-load problem. Both procedures construct one giant route servicing all transportation requests, and then divide this route into feasible vehicle routes. The first procedure constructs the giant route by solving a rural postman problem. The objective in this problem is to find a minimum cost cycle that traverses each arc in a given subset of arcs, in this case all arcs corresponding to traveling from an origin to a destination, at least once. The second procedure applies the above suggested transformation and constructs a giant route by solving an asymmetric TSP. On test problems with up to 800 requests and 45 vehicles the first procedure performed better than the second procedure.

Desrosiers, Laporte, Sauve, Soumis and Tailléfer [16] develop an optimization algorithm for a multi-depot full-truck-load problem in which the objective is to minimize the total distance traveled and there is a restriction on the length of the routes. The problem is transformed into an asymmetric TSP with a reduced set of subtour elimination constraints, prohibiting only subtours not including any depot, and distance constraints, prohibiting illegal subpaths of the Hamiltonian cycle. The transformed problem is solved by an algorithm of Laporte, Nobert and Tailléfer [33], which is a direct extension of the algorithm of Carpaneto and Toth [9] for the asymmetric TSP. Test problems with 15 physical locations, up to 104 requests, 1,2 or 3 depots and up to 18 vehicles per depot were solved with this algorithm.

1.3.2 The dynamic pickup and delivery problem

As in most combinatorial optimization problems, dynamic aspects of the pickup and delivery problem are not very well studied.

The dynamic single-vehicle pickup and delivery problem

Psaraftis [40] extends the dynamic programming algorithm described in Section 1.3.1 for the static immediate request dial-a-ride problem to the dynamic case. Indefinite deferral of customers, i.e., continuously reassigning service of a customer to the last position in the pickup and delivery sequence, is precluded with a special priority constraint.

The times at which requests for service are received define a natural order among the clients. In general, the position that a particular customer holds in the sequence of pickups will not be the same as his position in this natural order. The difference between these two positions
defines a *pickup position shift*. A *delivery position shift* can be similarly
defined as the difference between the position in the sequence of deliver-
ies and the position in the natural order. A priority constraint bounding
the two position shifts from above prevents indefinite deferment.

The dynamic problem is solved as a sequence of static problems.
Each time a new request for service is received, a slightly modified in-
stance of the static problem is solved to update the current route. Ob-
viously, all clients that have already been picked up and delivered can
be discarded and the new client has to be incorporated. The starting loca-
tion of the vehicle and the origins of the clients that have been picked
up but not yet delivered have to be set to the location of the vehicle at
the time of the update.

**The dynamic multi-vehicle pickup and delivery problem**

Psaraftis [44] develops an algorithm for the dynamic multi-vehicle prob-
lem in which the vehicles are in fact ships. In this case, the capacity of
the ports also has to be considered in order to avoid waiting times
when loads are to be picked up or delivered. The algorithm is based on
a rolling horizon principle. Let \( t_k \) be the ‘current time’, i.e., the time at
the \( k \)th iteration of the procedure. At time \( t_k \) the algorithm only con-
siders those known loads \( i \) whose earliest pickup time \( e_i \) falls between
\( t_k \) and \( t_k + L \), where \( L \) is the length of the rolling horizon, as a user input.

The algorithm then makes a tentative assignment of loads to eligible
ships. However, only loads with \( e_i \) within the interval \([t_k, t_k + \alpha L]\) for
some \( \alpha \in (0, 1) \) are considered for permanent assignment. In this way
the algorithm places less (but still some) emphasis on the less reliable
information about the future. Iteration \( k + 1 \) will move the ‘current
time’ to \( t_{k+1} \), which is equal to the time a significant input update has
to be made, or to the lowest \( e_i \) of all yet unassigned loads, whichever
of the two is the earliest.

The tentative assignment of loads to ships is calculated in two phases.
First the utility \( \pi_{ij} \) of assigning load \( i \) to ship \( j \) is calculated for each
load \( i \) and ship \( j \). This utility is a complicated function measuring the
assignment’s effect on (1) the delivery time of load \( i \), (2) the delivery
times of all other loads already assigned to ship \( j \), (3) the efficiency of
use of ship \( j \), and (4) the system’s port resources. In the second phase
the following assignment problem is solved:
maximize \[ \sum \sum u_{ij} x_{ij} \]
subject to
\[ \sum_{j} x_{ij} \leq K \]
for all \( j \),
\[ \sum_{i} x_{ij} \leq 1 \]
for all \( i \),
\[ x_{ij} \in \{0, 1\} \]
for all \( i, j \).

where \( K \) is a user specified integer denoting that no more than \( K \) loads are assigned to each ship.

The dynamic full-truck-load pickup and delivery problem

Powell [36, 37, 38, 39] and Frantzeskakis and Powell [22] consider a dynamic full-truck-load pickup and delivery problem in which transportation requests may be rejected. An algorithm is developed based on network flow representation of the problem. In this model the planning area is divided into regions \( R \) and the time axis with time horizon \( P \) is divided into days \( \{0, 1, 2, \ldots, P\} \). The network has node set \((R \times \{0, 1, 2, \ldots, P\}) \cup S \cup T\), where \( S \) represents a source and \( T \) represents a sink. The arc set consists of arcs representing loaded moves that are known at time \( t = 0 \), arcs representing empty moves, all arcs \(( (r, t), (r, t+1) ) \), where \( r \in R \) and \( 0 \leq t < P \), and all arcs \(( S, (r, 0) ) \) and \(( (r, P), T ) \), where \( r \in R \). Arcs representing a loaded move have capacity 1, arcs \(( S, (r, 0) ) \) have capacity equal to the number of vehicles available in region \( r \) on day 0. All other arcs have capacity \( \infty \). The profit of an arc representing a move is the net profit or cost of the corresponding move. The profit of all other arcs is 0. Note that a maximum profit flow in this network corresponds to an allocation of vehicles to transportation requests, but that it is not required to honor all transportation requests.

To anticipate future transportation requests, the network is extended with stochastic links. These stochastic links correspond to future uncertain trajectories of vehicles. They emanate from each node \((r, t)\) and end in \( T \). The profit of the \( k \)th stochastic link emanating from \((r, t)\) reflects the expected marginal value of another vehicle in region \( r \) on day \( t \) given that there are already \( k \) vehicles in region \( r \) on day \( t \). These expected marginal values of an extra vehicle are based on historical data. Each stochastic link has capacity 1. A maximum profit flow in this extended network not only represents a deterministic allocation of vehicles to loads known at \( t = 0 \), but also assigns vehicles to regions in order to be able to serve future transportation requests at minimal cost. The use
of a stochastic link from \((r,t)\) to \(T\) indicates that at time \(t\) the vehicle will be available in region \(r\) to serve some request that is not yet known. Because the vehicles are only allocated to known loads, the network has to be reoptimized a few times each day.

1.4 Concluding remarks

In this chapter, we have discussed various characteristics of pickup and delivery problems and have given an overview of the solution approaches that have been proposed. In the process, we have identified several important research topics.

Most real-life pickup and delivery problems that we are aware of are demand responsive. Currently, very little is known about on-line algorithms for dynamic pickup and delivery problems. Besides the obvious practical importance of such algorithms, it seems to be a fascinating and challenging research area as well.

Although the single-vehicle pickup and delivery problem is \(NP\)-hard, it can be solved very efficiently with dynamic programming as long as the number of transportation requests is relatively small, which is the case in many practical situations. Therefore, the main problem in solving multi-vehicle pickup and delivery problems is the assignment of transportation requests to vehicles. Consequently, pickup and delivery problems, as well as many other routing and scheduling problems, seem to be well suited for solution approaches based on set partitioning with column generation. Although this approach has already been explored successfully, it is likely to remain an active research area for the next decade.
The Resource Assignment Problem

In the resource assignment problem (RAP), a set of tasks has to be assigned to a set of resources such that each task is assigned to exactly one resource and all tasks that are assigned to a single resource can be executed together by that resource. Each assignment of tasks to resources incurs a certain cost. The goal is to find the assignment with minimal cost.

Two resources are called independent if the cost and feasibility of assigning a set of tasks to one resource does not influence the cost and feasibility of assigning another set of tasks to the other resource. We consider the class of resource assignment problems where all resources are mutually independent. Examples of resource assignment problems with independent resources are facility location problems, generalized assignment problems, vehicle routing problems and machine scheduling problems.

In this chapter, we present a branch-and-price algorithm for the RAP that is based on a set partitioning formulation of the problem. Branch-and-price algorithms [2] solve mixed integer programming formulations with huge numbers of variables. In branch-and-price algorithms, sets of columns are left out of the linear program because there are too many columns to handle efficiently and most of them have their associated variable equal to zero in an optimal solution anyway. In order to
check the optimality of a linear programming solution, a subproblem, called the pricing problem, is then solved in order to identify columns to enter the basis. If such columns are found, the linear program is reoptimized. Branching occurs when no columns price out to enter the basis and the linear programming solution is optimal and fractional. Branch-and-price, which is a generalization of branch-and-bound with linear programming relaxations, allows column generation to be applied throughout the branch-and-bound tree.

In recent years, set partitioning formulations have become very popular for many combinatorial optimization problems. There are two main reasons for this. First, for many problems alternative formulations are not known, or very hard to define. This is the case, for example, in crew pairing [1] and vehicle routing [11] problems. Second, for many problems where alternative formulations are known, the linear programming relaxation of the set partitioning formulation often yields a stronger bound. This is the case, for example, in cutting stock [55] and generalized assignment [48] problems.

2.1 Problem formulation

Let $N = \{1, 2, \ldots, n\}$ be the set of tasks and let $M = \{1, 2, \ldots, m\}$ be the set of resources. Define binary variables $z^k_i$ ($i \in N, k \in M$) indicating whether task $i$ is assigned to resource $k$ ($z^k_i = 1$) or not ($z^k_i = 0$). We call $z^k$ the characteristic vector of the set of tasks that is assigned to resource $k$. Let $Z_k \subseteq \{0, 1\}^N$ be the set of feasible assignments to resource $k$, i.e. $z^k \in Z_k$ if and only if $\{i \in N \mid z^k_i = 1\}$ can be executed together by resource $k$. For each $z^k \in Z_k$, define $c_k(z^k)$ as the cost of assigning $\{i \in N \mid z^k_i = 1\}$ to resource $k$.

The resource assignment problem can now be formulated as the problem to

\[
\begin{align*}
\text{minimize} & \quad \sum_{k \in M} c_k(z^k) \\
\text{subject to} & \quad \sum_{k \in M} z^k_i = 1 & \text{for all } i \in N \\
\quad & \quad z^k \in Z_k & \text{for all } k \in M \\
\quad & \quad z^k_i \in \{0, 1\} & \text{for all } k \in M, i \in N
\end{align*}
\]

For some, but not all, problem types, the constraint $z^k \in Z_k$ can be linearized by introducing, for some $n_k, m_k \in \mathbb{N}$, extra variables $\zeta^k \in \{0, 1\}$. The resource assignment problem can now be formulated as the problem to

\[
\begin{align*}
\text{minimize} & \quad \sum_{k \in M} c_k(z^k) \\
\text{subject to} & \quad \sum_{k \in M} z^k_i = 1 & \text{for all } i \in N \\
\quad & \quad z^k \in Z_k & \text{for all } k \in M \\
\quad & \quad z^k_i \in \{0, 1\} & \text{for all } k \in M, i \in N \\
\quad & \quad \zeta^k \in \{0, 1\} & \text{for all } k \in M \\
\quad & \quad \sum_{k \in M} \zeta^k_i = 1 & \text{for all } i \in N \\
\end{align*}
\]
two matrices \( A^k \in \mathbb{R}^{m_k \times n} \) and \( B^k \in \mathbb{R}^{m_k \times n_k} \), and a vector \( b^k \in \mathbb{R}^{m_k} \) such that:

\[
\mathbf{z}^k \in \mathbb{Z}_k \iff A^k \mathbf{z}^k + B^k \mathbf{\zeta}^k \leq b^k \quad \text{and} \quad \mathbf{\zeta}^k \in D^k,
\]

where \( D^k \subseteq \mathbb{R}^{n_k} \) imposes bounds or integrality restrictions on the variables \( \zeta_{j,}^k \) (\( j = 1, 2, \ldots, n_k \)). For those problem types, the cost function \( c_k(\mathbf{z}^k) \) can usually be linearized by introducing two vectors \( p^k \in \mathbb{R}^n \) and \( q^k \in \mathbb{R}^{n_k} \) such that

\[
c_k(\mathbf{z}^k) = p^k \mathbf{z}^k + q^k \mathbf{\zeta}^k.
\]

For some problem types, we only need the extra variables \( \mathbf{\zeta}^k \) in order to be able to linearize the cost function. We start by describing how some well-known problems fit into this framework.

The generalized assignment problem

In the generalized assignment problem (GAP) a set of tasks \( N \) has to be assigned to a set of machines \( M \). When task \( i \) is assigned to machine \( k \), it uses an amount \( a_{ki} \) of the machine capacity \( b_k \) and inures a cost \( c_{ki} \). The GAP can therefore be formulated as a linear RAP by taking \( n_k = 0 \) and \( m_k = 1 \) for all \( k \in M \). We now have

\[
\mathbf{z}^k \in \mathbb{Z}_k \iff \sum_{i \in N} a_{ki} z_{ik}^k \leq b_k.
\]

and for all \( \mathbf{z}^k \in \mathbb{Z}_k \)

\[
c_k(\mathbf{z}^k) = \sum_{i \in N} c_{ki} z_{ik}^k.
\]

The facility location problem

In the facility location problem (FLP) the tasks represent clients that demand some type of service, and the resources represent facilities that provide this service. The facilities can be opened at various locations. Let \( M \) be the set of possible facility locations. When a facility is opened at location \( k \in M \), and client \( i \in N \) is assigned to \( k \), this client uses an amount \( a_{ki} \) of the facility’s capacity \( b_k \). Opening a facility in \( k \) incurs a cost \( f_k \). Assigning client \( i \) to location \( k \) yields a cost of \( c_{ki} \). This problem can be formulated as a linear RAP by taking \( n_k = m_k = 1 \) for
all \( k \in M \). The variable \( \zeta^k \) is a binary variable equal to 1 if and only if facility \( k \) is opened. We now have

\[
 z^k \in Z_k \iff \sum_{i \in N} a_{ki} z_i^k - b_{ki} \zeta^k \leq 0 \text{ and } \zeta^k \in \{0,1\},
\]

and for all \( z^k \in Z_k \)

\[
 c_k(z^k) = \sum_{i \in N} c_{ki} z_i^k + f_k \zeta^k.
\]

The bin packing problem

In the bin packing problem (BPP) the tasks represent items and the resources represent bins. Each item \( i \in N \) has a size \( a_i \) and each bin has a capacity \( b \). The number of available bins satisfies \( |M| = |N| \), i.e. \( |M| \) is not restrictive. The goal is to minimize the number of bins needed to contain all items. Because all bins are identical, we have \( Z_k = Z \) for all \( k \in M \). This problem can be formulated as a linear RAP by taking \( n_k = m_k = 1 \) for all \( k \in M \). The variable \( \zeta^k \) is a binary variable equal to 1 if and only if bin \( k \) is used. For the BPP we now have

\[
 z^k \in Z \iff \sum_{i \in N} a_i z_i^k \leq b \zeta^k,
\]

and for all \( z^k \in Z \)

\[
 c_k(z^k) = \zeta^k.
\]

The bin packing problem is also known as the binary cutting stock problem. In this problem we have a set of stock rolls of length \( b \), and a set of requested lengths \( a_i \) \((i \in N)\), that have to be cut out of the rolls. The objective in this problem is to minimize the total waste, which is equivalent to minimizing the number of rolls needed to produce all lengths \( a_i \).

The graph coloring problem

In the graph coloring problem (GCP) a graph \( G = (N, E) \) is given with vertices \( N \) and edge set \( E \). Furthermore we have a set \( M \) of colors, with \( |M| = |N| \). The GCP is the problem of finding the minimum number of colors needed to color all nodes \( i \in N \) such that no adjacent nodes get the same color. This problem can be formulated as a RAP by representing the vertices as tasks and the colors as resources. Because
all colors are interchangeable, we have $Z_k = Z$ for all colors $k$. The GCP can be formulated as a linear RAP by taking $n_k = 1$ and $m_k = |E|$ for all $k \in M$. The variable $\zeta^k$ is a binary variable equal to 1 if and only if color $k$ is used. We now have

$$z^k \in Z \iff z_i^k + z_j^k \leq \zeta^k \quad \text{for all } \{i, j\} \in E,$$

and for all $z^k \in Z$

$$c_k(z^k) = \zeta^k.$$

The vehicle routing problem

In the vehicle routing problem (VRP) the tasks represent clients and the resources represent vehicles. A set $M$ of identical vehicles of capacity $b$ is stationed at a depot $0$. Each client $i \in N$ has a demand $a_i$ which has to be delivered by one of the vehicles. A trip for a vehicle is a tour that starts at the depot, visits a set of clients with a total demand of at most $b$ and then returns to the depot. Let $V = N \cup \{0\}$. For all $i, j \in V$, let $d_{ij}$ be the distance between locations $i$ and $j$. The goal in the VRP is to construct a set of trips such that each client is served and the total distance traveled is minimized. Because all vehicles are identical, we have $Z_k = Z$.

In order to give a linear description of $Z$ and the cost function, we take $n_k = |V|^2 + |V| + 1$ and $m_k = n(n+1)+n+3$. We partition the extra variables into three types. $x_{ij}$ ($i, j \in V$) equal to 1 if the vehicle travels from location $i$ to location $j$ and 0 otherwise, $t_i$ ($i \in V$) indicating the arrival time at location $i$ given that the vehicle departs from the depot at time 0 and all travel times are equal to one, and a binary variable $y$, equal to 1 if and only if the vehicle is used. The variable $y$ suffices to describe $Z$, the others are needed to linearize the cost function. We now have

$$z^k \in Z \iff \sum_{i \in N} a_i z_i^k \leq by,$$

and

$$c_k(z^k) = \sum_{i \in V} \sum_{j \in V} d_{ij} x_{ij},$$

where the variables $x$, $t$ and $y$ satisfy the following set of constraints:
\[ \sum_{j \in N} x_{0j} = \sum_{j \in N} x_{j0} = y, \]
\[ \sum_{j \in V} x_{ij} = \sum_{j \in V} x_{ji} = z_i \quad \text{for all } i \in N, \]
\[ t_0 = 0, \]
\[ t_i - t_j + (n + 1)x_{ij} \leq n \quad \text{for all } i, j \in N, \]
\[ \sum_{i \in N} a_i x_i \leq by, \]
\[ x_{ij} \in \{0,1\} \quad \text{for all } i, j \in V, \]
\[ y \in \{0,1\}. \]

### 2.2 The set partitioning formulation

To formulate the resource assignment problem as a set partitioning problem, we define, for all \( r \subseteq N \), \( \delta_r \) as the characteristic vector of \( r \). For each resource \( k \in M \) we define \( \Omega_k \subseteq 2^N \) as the collection of all sets that can be assigned to resource \( k \), i.e.

\[ \Omega_k := \{ r \subseteq N \mid \delta_r \in Z_k \}. \]

For each \( k \in M, r \in \Omega_k \), we define the cost

\[ c^k_r := c_k(\delta_r), \]

and introduce binary variables \( x^k_r \) \((k \in M, r \in \Omega_k)\) equal to 1 if subset \( r \in \Omega_k \) is used and 0 otherwise. The resource assignment problem can now be formulated as follows:

\[ \text{minimize} \sum_{k \in M, r \in \Omega_k} c^k_r x^k_r \]

subject to
\[ \sum_{k \in M, r \in \Omega_k} \delta_r x^k_r = 1 \quad \text{for all } i \in N, \]
\[ \sum_{k \in M, r \in \Omega_k} x^k_r \leq 1 \quad \text{for all } k \in M, \]
\[ x^k_r \in \{0,1\} \quad \text{for all } k \in M, r \in \Omega_k. \]

The constraints \( \sum_{k \in M} \sum_{r \in \Omega_k} \delta_r x^k_r = 1 \) are referred to as the **partitioning constraints**. The constraints \( \sum_{r \in \Omega_k} x^k_r \leq 1 \) are referred to as the **convexity constraints**. We denote this formulation by \( P \) and its linear programming relaxation by \( LP \).

When all resources are identical, we have \( \Omega_k = \Omega \) \((k \in M)\) and \( c^k_r = c_r \) \((r \in \Omega)\). In this case the set partitioning formulation can be reformulated by introducing binary variables \( x_r \) \((r \in \Omega)\) equal to 1 if subset \( r \in \Omega \) is used and 0 otherwise. The RAP with identical resources is the problem to
minimize \( \sum_{r \in \Omega} c_r x_r \)
subject to \( \sum_{r \in \Omega} \delta_r x_r = 1 \) for all \( i \in N \),
\( \sum_{r \in \Omega} x_r \leq |M| \),
\( x_r \in \{0,1\} \) for all \( r \in \Omega \).

We denote this formulation by \( P_0 \) and its linear programming relaxation by \( LP_0 \). When the size of \( M \) is not restrictive, the constraint \( \sum_{r \in \Omega} x_r \leq |M| \) can be deleted. In this case the set partitioning formulation is a pure set partitioning problem.

In the remainder we primarily focus on the case where the resources are not identical. Most ideas and results that are presented also hold when all resources are identical. When a result only holds for nonidentical resources, this will be explicitly stated. Note that, when all resources are identical, we can always consider them to be nonidentical.

### 2.3 Comparison of lower bounds

In this section, we show why the lower bound \( Z_{LP} \) given by the set partitioning formulation is usually stronger than the lower bound given by the linear programming relaxation of the traditional formulation. Suppose that we have a RAP that can be formulated as a linear integer programming problem as is described in Section 2.1. Let \( LP \) be the linear programming relaxation of this formulation. Without loss of generality we assume that the restriction \( \zeta^k \in D^k \) is relaxed to \( \zeta^j \geq 0 \) \( (j = 1,2,\ldots,n_k) \). Furthermore we assume that \( b^k \geq 0 \) for all \( k \in M \). Now \( LP \) is the problem to

minimize \( \sum_{k \in M} (r^k z^k + q^k \zeta^k) \)
subject to \( \sum_{k \in M} z^k_i = 1 \) for all \( i \in N \),
\( A^k z^k + B^k \zeta^k \leq b^k \) for all \( k \in M \),
\( \zeta^j \geq 0 \) for all \( k \in M, j = 1,2,\ldots,n_k \),
\( z^k_i \geq 0 \) for all \( k \in M, i \in N \).

We perform a Dantzig-Wolfe decomposition on \( LP \) in order to get an alternative formulation that is equivalent to \( LP \). This alternative formulation provides a good means to compare the values \( Z_{LP} \) and \( Z_{LP} \). In order to obtain a Dantzig-Wolfe decomposition of a problem, we select some constraints of the problem. The set of selected constraints defines
a polyhedral set. Since any point in a polyhedral set can be represented as a convex combination of a finite number of extreme points, we can reformulate the problem by substituting this extreme point representation. We perform a Dantzig Wolfe decomposition on problem $\overline{LP}$, by reformulating it in terms of the constraints $A^k z^k + B^k \zeta^k \leq b^k$ for all $k \in M$. For each $k \in M$, we define

$$X^k = \{(z, \zeta) \in M^*_+ \times M^*_+ \mid A^k z^k + B^k \zeta^k \leq b^k\}.$$ 

For ease of presentation we assume that this polyhedron is bounded. By the representation theorem of polyhedra we know that we can write each element of $X^k$ as a convex combination of the extreme points of $X^k$. Let

$$\{(z^k_j, \zeta^k_j) \mid j \in \Gamma_k\}$$

be the set of extreme points of $X^k$ and let $(z, \zeta) \in X^k$. Then there is a vector $\lambda \in M^*_+$ satisfying

$$z = \sum_{j \in \Gamma_k} \lambda_j z^k_j, \quad \zeta = \sum_{j \in \Gamma_k} \lambda_j \zeta^k_j, \quad \text{and} \quad \sum_{j \in \Gamma_k} \lambda_j = 1.$$ 

By performing this substitution in $\overline{LP}$, we get the formulation $\overline{LP}_\Gamma$, which is equivalent to $\overline{LP}$:

minimize $\sum_{k \in M} \sum_{j \in \Gamma_k} \lambda_j^k (p^k z^k_j + q^k \zeta^k_j)$
subject to $\sum_{k \in M} \sum_{j \in \Gamma_k} \lambda_j^k z^k_j = 1$ for all $i \in N$,
$\sum_{j \in \Gamma_k} \lambda_j^k = 1$ for all $k \in M$,
$\lambda_j^k \geq 0$ for all $k \in M, j \in \Gamma_k$.

Because $b^k \geq 0$, we have that $(0, 0)$ is an extreme point of $X^k$ for all $k \in M$. Let $\Gamma'_k$ be the index set of extreme points of $X^k$ excluding $(0, 0)$. When we define $\hat{e}^k_j = p^k z^k_j + q^k \zeta^k_j$ for all $k \in M, j \in \Gamma'_k$, we get formulation $\overline{LP}'$, which is still equivalent to $\overline{LP}$:

minimize $\sum_{k \in M} \sum_{j \in \Gamma'_k} \lambda_j^k \hat{e}^k_j$
subject to $\sum_{k \in M} \sum_{j \in \Gamma'_k} \lambda_j^k z^k_j = 1$ for all $i \in N$,
$\sum_{j \in \Gamma'_k} \lambda_j^k \leq 1$ for all $k \in M$,
$\lambda_j^k \geq 0$ for all $k \in M, j \in \Gamma'_k$. 

Formulation $\overline{LP}_{\text{rel}}^r$ differs from $LP$ because the values $z^k_{ji}$ may be fractional, where the values $\delta_{ri}$ are integral. Let $k \in M$ and $r \in \Omega_k$. Because $r$ is a feasible set of tasks for resource $k$, there is some $\zeta_r^k \in D^k$, such that $(\delta_r, \zeta_r^k) \in X^k$. This implies that there is some vector $\hat{\lambda} \in \mathbb{R}_+^{\mid \Omega_k^r \mid}$ satisfying

$$\delta_r = \sum_{j \in \Omega_k^r} \hat{\lambda}_j z^k_{ji}, \quad \zeta_r^k = \sum_{j \in \Omega_k^r} \hat{\lambda}_j^2 \zeta_r^k, \text{ and } \sum_{j \in \Omega_k^r} \hat{\lambda}_j = 1.$$ 

We can conclude that $\overline{LP}_{\text{rel}}^r$, and therefore $\overline{LP}$, is a relaxation of $LP$, and hence $Z_{LP} \geq Z_{\overline{LP}}$.

This result has been mentioned by Vance, Barnhart, Johnson, and Nemhauser [55] for the binary cutting stock problem, and by Savelsbergh [48] for the generalized assignment problem, but takes back to the earlier work of Geoffrion [24]. For the bin packing problem, which is identical to the binary cutting stock problem, we give an example of how large the difference between $Z_{LP}$ and $Z_{\overline{LP}}$ can be. For this problem, $\overline{LP}$ is the problem to minimize

$$\sum_{k \in M} \zeta^k$$

subject to

$$\sum_{k \in M} z^k_i = 1 \quad \text{for all } i \in N,$$

$$\sum_{i \in N} a_i z^k_i \leq b \zeta^k \quad \text{for all } k \in M,$$

$$\zeta^k \geq 0 \quad \text{for all } k \in M,$$

$$z^k_i \geq 0 \quad \text{for all } k \in M, i \in N.$$ 

Now let $l$ be some positive integer. Consider an instance of the BPP with $b = 2l - 1$ and $a_i = l$ for all $i \in N$. Because $\Omega_k = \{i\mid i \in N\}$, for all $k \in M$, we see that $Z_{LP} = Z_{LP} = |N|$. However, by taking $z^k_{ik} = 1$ if $i = k$, $z^k_{ik} = 0$ otherwise, and $\zeta^k = \frac{l}{2l-1}$ if $k \leq |N|$, $\zeta^k = 0$ otherwise, we obtain a feasible solution to $\overline{LP}$. This indicates that $Z_{\overline{LP}} \leq \frac{l}{2l-1} |N|$, which approaches $\frac{1}{2} Z_{LP}$ for large values of $l$.

### 2.4 A branch-and-price algorithm

In this section, we present an algorithm for the RAP based on the set partitioning formulation $P$ as presented in Section 2.2. An algorithm based on formulation $P$ must be able to handle the large number of variables that arises due to the size of the sets $\Omega_k$. In the past this
problem has been handled by first generating a large but manageable set of columns, and then solving the set partitioning problem on this incomplete set of columns. In this way no optimal solutions can be guaranteed.

We have developed a branch-and-price algorithm for the RAP, in which we use a column generation scheme to solve LP in order to handle the large number of variables. Instead of explicitly enumerating all feasible subsets in order to find a variable that prices out to enter the basis, in a column generation approach the nonbasic variable with the smallest negative reduced cost is found by solving an optimization problem, called the pricing problem. In this way, we generate the feasible subsets on the fly as needed, and only a small fraction of all feasible subsets is used to solve LP.

We have developed a special branching strategy to solve P. Such a strategy is necessary to ensure that the pricing problem can be adjusted so that at no node of the branch-and-bound tree infeasible columns are generated.

2.4.1 Column generation

Suppose that for each resource $k \in M$ a set $\Omega_k' \subseteq \Omega_k$ of feasible subsets is explicitly known. The restricted master problem LP' is defined as follows:

\[
\text{minimize} \quad \sum_{k \in M} \sum_{r \in \Omega_k'} \delta_{r} x_{r}^k
\]

\[
\text{subject to} \quad \sum_{k \in M} \sum_{r \in \Omega_k'} \delta_{r} x_{r}^k = 1 \quad \text{for all } i \in N,
\]

\[
\sum_{k \in M} x_{r}^k \leq 1 \quad \text{for all } k \in M,
\]

\[
x_{r}^k \geq 0 \quad \text{for all } k \in M, r \in \Omega_k'.
\]

Suppose that LP' has a feasible solution $x$ and let $(u, v)$ be the associated dual solution, where the dual variables $u_i$ $(i \in N)$ are associated with the partitioning constraints and the dual variables $v_k$ $(k \in M)$ are associated with the convexity constraints. From linear programming duality we know that $x$ is optimal with respect to LP if and only if for each $k \in M$ and for each $r \in \Omega_k$ the reduced cost $d_{r}^k$ is nonnegative, i.e.,

\[d_{r}^k = c_{r}^k - \sum_{i \in N} \delta_{i,r} u_i - v_k \geq 0 \quad \text{for all } k \in M, r \in \Omega_k.'
Testing the optimality of \( x \) with respect to \( LP \) can thus be done by solving the pricing problem:

\[
\text{minimize} \{ c_k^k - \sum_{i \in N} \delta_{r_i} u_i - v_k \mid k \in M, r \in \Omega_k \}.
\]

Let \( z \) denote the value of the solution to the pricing problem and let \( k_z \) and \( r_z \) denote the corresponding resource and subset. If \( z \geq 0 \), then \( x \) is also optimal with respect to \( LP \), otherwise \( r_z \) defines a column that can enter the basis and has to be added to \( \Omega'_{k_z} \). This yields the following column generation scheme:

1. Find initial sets \( \Omega'_k \) containing a feasible solution \( x \).
2. Solve the restricted master problem \( LP' \).
3. Solve the pricing problem. If \( z \geq 0 \) then stop, otherwise set \( \Omega'_{k_z} \leftarrow \Omega'_{k_z} \cup \{ r_z \} \) and go to Step 2.

Due to the presence of the convexity constraints, it is nontrivial to find initial sets \( \Omega'_k \subseteq \Omega_k \) containing a feasible solution to \( LP \). However, if they exist, such sets can always be found using a two-phase method similar in spirit to the two-phase method incorporated in simplex algorithms to find an initial basic feasible solution. Define \( LP_+ \) as the problem to minimize

\[
\sum_{k \in M} \sum_{r \in \Omega_k} c_{rk}^k x_{rk} + \sum_{i \in N} py_i,
\]

subject to

\[
\sum_{k \in M} \sum_{r \in \Omega_k} \delta_{r_i} x_{rk} + y_i = 1 \quad \text{for all } i \in N,
\]
\[
\sum_{r \in \Omega_k} x_{rk}^k \leq 1 \quad \text{for all } k \in M,
\]
\[
x_{rk}^k \geq 0 \quad \text{for all } k \in M, r \in \Omega_k,
\]
\[
y_i \geq 0 \quad \text{for all } i \in N,
\]

where \( y_i \ (i \in N) \) is an artificial variable and \( p > \max_{r \in \Omega_k} c_{rk}^k \) is an appropriate penalty cost. Problem \( LP_+ \) can be solved by the above column generation scheme by initializing \( \Omega'_k = \emptyset \) for each \( k \in M \).

The artificial variables \( y_i \) are not deleted when they have all become nonbasic, i.e., when a feasible solution to problem \( LP \) has been found. Because of their high cost, these variables will stay nonbasic and will not interfere with the optimization process. However, during the branching process, the sets \( \Omega'_k \) are restricted by the branching scheme, possibly yielding an initial infeasible \( LP \) in a node. In that case, the artificial variables will reappear in the basis, and the first phase is automatically started in order to find sets \( \Omega'_k \) that do contain a feasible solution for the linear program associated with this node.
2.4.2 The pricing problem

The pricing problem decomposes into several independent problems, one for each resource. Let \( S_k \) be the problem of minimizing \( \{c_r^k - \sum_{i \in N} \delta_{ir} u_i - v_k \mid r \in \Omega_k \} \). When all resources are identical these problems are identical, such that the pricing problem is the problem of minimizing \( \{c_r - \sum_{i \in N} \delta_{ir} u_i - v \mid r \in \Omega \} \). When the RAP can be formulated as a linear program as described in Section 2.1, we can formulate pricing problem \( S_k \) as the problem to

\[
\begin{align*}
\text{minimize} & \quad (p^k - u)z + q^k \zeta - v_k \\
\text{subject to} & \quad A^k z + B^k \zeta \leq b^k, \\
& \quad \zeta \in D^k, \\
& \quad z_i \in \{0, 1\} \quad \text{for all } i \in N.
\end{align*}
\]

The pricing problem can still be a difficult combinatorial optimization problem. We briefly discuss these pricing problems for the examples given in Section 2.1.

The generalized assignment problem

Savelsbergh [48] uses a column generation approach for the generalized assignment problem. For this problem, the reduced cost of \( r \in \Omega_k \) is

\[
d_r^k = \sum_{i \in r} (c_{ki} - u_i) - v_k.
\]

Consider the pricing problem for machine \( k \). Let \( z_i (i \in N) \) be a binary variable equal to 1 if and only if \( i \) is assigned to \( k \). The pricing problem for machine \( k \) can now be formulated as the problem to

\[
\begin{align*}
\text{minimize} & \quad \sum_{i \in N} (c_{ki} - u_i) z_i - v_k \\
\text{subject to} & \quad \sum_{i \in N} a_{ki} z_i \leq b_k, \\
& \quad z_i \in \{0, 1\} \quad \text{for all } i \in N,
\end{align*}
\]

which is a 0–1 knapsack problem. It is \( \mathcal{NP} \)-hard, but relatively easy to solve in practice.

The facility location problem

For the facility location problem, the reduced cost of \( r \in \Omega_k \) is

\[
d_r^k = \sum_{i \in r} (c_{ki} - u_i) + (f_k - v_k).
\]
Note that \( r \neq \emptyset \) implies that the facility is opened. As in the generalized assignment problem, the pricing problem for the facility location problem is a 0–1 knapsack problem.

The bin packing problem

A column generation approach has been used for the bin packing problem by Gilmore and Gomory [25], and Vance, Barnhart, Johnson and Nemhauser [55]. For this problem, the reduced cost of \( r \in \Omega \) is

\[
d_r = 1 - \sum_{i \in r} u_i.
\]

Let \( z_i \ (i \in N) \) be a binary variable equal to 1 if and only if \( i \) is chosen in the subset. Then the pricing problem can be formulated as the problem to

maximize \[\sum_{i \in N} u_i z_i\]
subject to \[\sum_{i \in N} a_i z_i \leq b\]
\[z_i \in \{0, 1\}\]
for all \( i \in N \),

which is again a 0–1 knapsack problem.

The graph coloring problem

Mehrotra and Trick [34], use a column generation approach for graph coloring problems. For the GCP, we have the same reduced cost as in the bin packing problem. For all \( r \in \Omega \) we have \( d_r = 1 - \sum_{i \in r} u_i \). Let \( z_i \ (i \in N) \) be a binary variable equal to 1 if and only if \( i \) is chosen in the subset. Then the pricing problem can be formulated as the problem to

maximize \[\sum_{i \in N} u_i z_i\]
subject to \[z_i + z_j \leq 1\]
for all \( \{i, j\} \in E \),
\[z_i \in \{0, 1\}\]
for all \( i \in N \),

which is a maximum weight independent set problem. It is \( NP \)-hard.

The vehicle routing problem

Column generation methods have been widely studied for the vehicle routing problem and related problems. A survey on column generation techniques for these problems is presented by Desrosiers, Dumas,
Solomon and Soumis [13]. For the vehicle routing problem, the reduced cost of \( r \in \Omega \) is
\[
d_r = c_r^k - \sum_{i \in r} u_i.
\]

When we define alternative travel distances \( d'_{ij} \) by \( d'_{ij} = d_{ij} - u_i \) if \( i \neq 0 \), and \( d'_{ij} = d_{ij} \) otherwise, then the reduced cost can be interpreted as the cost of the cheapest tour through \( r \cup \{0\} \) according to these travel distances \( d'_{ij} \). Note that \( d'_{ij} \) can be negative. The pricing problem is thus the problem of finding a minimum cost cycle that contains the depot 0 and visits each client from a set of clients with total demand at most \( b \) exactly once.

2.4.3 Lower bounds

Two types of lower bounds are important in a branch-and-price algorithm. The first type is a lower bound on \( Z_P \), the optimal solution value to the integer programming problem. A high quality lower bound on \( Z_P \) helps to keep the search tree small. Such a lower bound is provided by \( Z_{LP} \). The second type is a lower bound on \( Z_{LP} \) during the column generation process. If \( P \) is an integer programming problem with integer cost coefficients, then the bound provided by the LP relaxation is \([Z_{LP}]\). Consequently, column generation can be terminated as soon as \([Z_{LP}] = [Z_{LP}]\). We need a lower bound on \( Z_{LP} \) in order to check this criterion. Furthermore, such a lower bound is needed, because the column generation process suffers from a tailing off effect when \( LP \) has almost been solved to optimality. A lower bound on \( Z_{LP} \) provides a criterion to decide whether the process can be terminated, even when columns with negative reduced cost are still available. Note that during the column generation process, we have \( Z_{LP} \geq Z_{LP} \). We do not yet have a lower bound on \( Z_{LP} \).

In this section we present an improved lower bound on \( Z_P \), that can be used for resource assignment problems with a specific cost structure, and we present a lower bound on \( Z_{LP} \).

An improved lower bound on \( Z_P \):

Dumas, Desrosiers and Soumis [19] present an improvement of the lower bound \( Z_{LP} \) which can be generalized to resource assignment problems in which the primary objective is to minimize the number of resources
needed to execute all tasks. This objective is usually modeled by using
the following cost structure:

\[ c^k_i = F + \gamma^k_i, \]

where the fixed cost \( F \) of using a resource is much larger than any of the
variable costs \( \gamma^k_i \). Now suppose that we have such a cost structure and
suppose that the optimal solution \( x \) to \( LP \) corresponds to a nonintegral
number of resources. Let

\[ m = \left\lfloor \sum_{k \in M} \sum_{r \in \Omega_k} x^k_r \right\rfloor. \]

Then the constraint

\[ \sum_{k \in M} \sum_{r \in \Omega_k} x^k_r \geq m \]

is a valid inequality that may be added to \( LP \). When \( LP \) has been solved
to optimality, and we have added this constraint, new columns may price
out favorably to enter the basis. Although the addition of the constraint
modifies the reduced cost of all columns, it does not complicate the
pricing problems, because the dual value of this constraint appears as a
constant in their objective functions.

A lower bound on \( Z_{LP} \)

If \( P \) is an integer programming problem with integer cost coefficients,
then \( [Z_{LP}] \) is a lower bound on \( Z_P \). Therefore, we can stop generating
new columns as soon as \( [Z_{LP'}] = [Z_{LP}] \). Although, during the col-
umn generation process, we do not know \( Z_{LP'} \), we can use this stopping
criterion if we have a lower bound \( LB \) on \( Z_{LP} \) satisfying

\[ [Z_{LP'}] = [LB]. \]

When we use this stopping criterion, we also avoid the tailing off effect
that the column generation process usually suffers from. When \( LP \) has
almost been solved to optimality, during many iterations, only a small
number of columns is found that have almost no effect on the LP value.
Even when the optimal LP value has been reached, it often takes many
iterations to prove that the LP solution is optimal.

In this section we present a lower bound on \( Z_{LP} \). This bound can
be calculated from the current LP value \( Z_{LP'} \) and the optimal solutions
to the pricing problems.
Suppose that \((u, v)\) is an optimal solution to the dual of \(LP'\), and let, for each \(k \in M\), \(r_k\) be an optimal solution to pricing problem \(S_k\), i.e.
\[
d^k_{r_k} = e^k_{r_k} - \sum_{i \in r_k} u_i - v_k \leq e^k_r - \sum_{i \in r} u_i - v_k \quad \text{for all} \quad r \in \Omega_k
\]
Now let \(x\) be any feasible solution to \(LP\). Then we have
\[
\sum_{k \in M} \sum_{r \in \Omega_k} c^k_r x^k_r \\
\geq \sum_{k \in M} \sum_{r \in \Omega_k} (d^k_{r_k} + \sum_{i \in r} u_i + v_k)x^k_r \\
= \sum_{k \in M} (d^k_{r_k} + v_k)(\sum_{r \in \Omega_k} x^k_r) + \sum_{k \in M} \sum_{r \in \Omega_k} \sum_{i \in N} \delta_{r_i} x^k_r u_i \\
\geq \sum_{k \in M} (d^k_{r_k} + u_k) + \sum_{i \in N} u_i \\
= Z_{LP'} + \sum_{k \in M} d^k_{r_k},
\]
where the last inequality follows from \(d^k_{r_k} + v_k \leq 0\), and \(\sum_{r \in \Omega_k} x^k_r \leq 1\).
We conclude that
\[LB := Z_{LP'} + \sum_{k \in M} d^k_{r_k}\]
is a lower bound on \(Z_{LP'}\).

### 2.4.4 Branching schemes

In order to obtain integral solutions, we need a branching scheme that excludes the current fractional solution, validly partitions the solution space of the problem, and does not complicate the pricing problem too much. The third requirement almost always excludes the standard branching rules based on variable fixing. Fixing a variable to 1 does not complicate the pricing problem, because it just reduces the size of the problem. However, fixing a variable to 0 corresponds to forbidding a certain solution to the pricing problem. Deeper down the search tree this implies that a set of solutions to the pricing problem must be excluded, which is in general very complicated, if not impossible [2, 14, 19, 48, 55].
In order to develop branching schemes that satisfy the above requirements, we have to exploit the structure of the resource assignment problem. One way to do this is to calculate the original variables $z^k$ and $\zeta^k$ from the fractional solution $x$ of LP. For all $k \in M, i \in N$, and $j \in \{1, 2, \ldots, n_k\}$, we can calculate the original variables $z^k_i$ and $\zeta^k_j$ as follows:

$$(z^k_i, \zeta^k_j) = \sum_{r \in \mathcal{R}} z_r^k(\delta, \tau, \zeta^k_j).$$

If one of these variables is fractional and should be integral, it is often possible to branch on this variable, such that the branching information can be incorporated in the pricing problems. For a specific problem type it is sometimes possible to develop a branching scheme that is based on some of the variables $\zeta^k_j$. However, because all resource assignment problems contain the common structure of an assignment problem, we describe branching schemes that partition the solution space with respect to assignment decisions.

We present two such branching schemes. They are both special cases of a branching scheme proposed by Ryan and Foster [45] for pure set partitioning problems, which is based on the following proposition.

**Proposition 1** Let $A$ be a $0-1$ matrix of size $m \times n$ and let $x$ be a fractional basic solution to $Ax = 1, x \geq 0$. Then there exist two rows $p$ and $q$ such that $0 < \sum_{j=1}^{n} A_{pj}A_{qi}x_j < 1$.

The set of integral solutions to $Ax = 1$ can now be divided into two subsets, characterized by $x_j = 0$ if $A_{pj} + A_{qi} = 1$, and $x_j = 0$ if $A_{pj} + A_{qi} \neq 1$, respectively. In the first subset $p$ and $q$ must be covered by the same column. In the second subset $p$ and $q$ must be covered by two distinct columns.

In our set partitioning model for the resource assignment problem, we have two sets of rows, corresponding to the partitioning constraints for the tasks and the convexity constraints for the resources. When both $p$ and $q$ are partitioning rows, the Ryan and Foster branching scheme implies the creation of a subset in which tasks $p$ and $q$ must be executed by the same resource, and another subset in which both tasks must be executed by two different resources. When $p$ is a set partitioning row and $q$ is the convexity constraint of resource $k$, the first subset consists of solutions where task $p$ is executed by resource $k$, while in the second subset task $p$ may not be executed by resource $k$. One can easily see that it is impossible that both $p$ and $q$ correspond to convexity constraints.
Let $x$ be the current fractional solution to LP. Now define for each $i \in N$ and $k \in M$ the assignment value $z_i^k = \sum_{r \in \Omega_k} \delta_{ir} x_r^k$, indicating what fraction of task $i$ is executed by resource $k$ in the current LP solution, and define for all $i, j \in N$ the combination value $y_{ij} = \sum_{k \in M} \sum_{r \in \Omega_k} \delta_{ir} \delta_{jr} x_r^k$, indicating what fraction of the resources executing $i$ also executes $j$. We have the following two lemmas.

**Lemma 1** Let $x$ be an optimal solution of LP, and let $z_i^k = \sum_{r \in \Omega_k} \delta_{ir} x_r^k$ ($k \in M, i \in N$). Then $x$ is integral if and only if $z$ is integral.

**Proof**
If $x$ is integral, then trivially $z$ is integral. Now suppose $z$ is integral. Let $k \in M$ and $i \in N$ satisfy $z_i^k = 1$. Because $\sum_{r \in \Omega_k} \delta_{ir} x_r^k = 1$ and $\sum_{r \in \Omega_k} x_r^k \leq 1$, we have that $\delta_{ir} = 1$ for all $r \in \Omega_k$ with $x_r^k > 0$. Therefore all subsets $r \in \Omega_k$ with $x_r^k > 0$ are identical. Since all subsets in $\Omega_k$ are distinct, this implies that $x$ is an integral solution.

**Lemma 2** Let $x$ be an optimal solution of LP and let, for all $i, j \in N$, $y_{ij} = \sum_{k \in M} \sum_{r \in \Omega_k} \delta_{ir} \delta_{jr} x_r^k$. If $y$ is integral, then $x$ is a convex combination of integral solutions of LP.

**Proof**
Suppose that $y$ is integral and let $y_{ij} = 1$. Then $\sum_{k \in M} \sum_{r \in \Omega_k} \delta_{ir} \delta_{jr} x_r^k = 1 = \sum_{k \in M} \sum_{r \in \Omega_k} \delta_{ir} x_r^k = \sum_{k \in M} \sum_{r \in \Omega_k} \delta_{jr} x_r^k$. This implies that $\delta_{ir} = \delta_{jr}$ for all $k \in M$ and $r \in \Omega_k$ with $x_r^k > 0$. Let $\Omega_k = \{r \in \Omega_k | x_r^k > 0\}$, and $Q = \{\xi \in \Omega_k | \sum_{k \in M} \sum_{r \in \Omega_k} \delta_{ir} \xi_r = 1 \ (i \in N), \sum_{r \in \Omega_k} \xi_r x_r^k \leq m_k \ (k \in M), \xi_r \geq 0\}$. Observe that the restriction of $x$ to $\Omega_k (k \in M)$, say $\bar{x}$, is an element of $Q$. Furthermore, the constraint matrix $A$ defining $Q$ is totally unimodular, since, after deletion of duplicate rows, each column of $A$ contains exactly two 1 entries: one in the partitioning rows and one in the convexity rows. Consequently, $Q$ is an integral polyhedron and $\bar{x}$, and thus $x$, are convex combinations of integral solutions of LP.

**Corollary 1** Let $x$ be an optimal extremal solution of LP. Define for each $(i, j \in N)$, $y_{ij} = \sum_{k \in M} \sum_{r \in \Omega_k} \delta_{ir} \delta_{jr} x_r^k$. Then $x$ is integral if and only if $y$ is integral.

Note that in Lemma 1, Lemma 2 and Corollary 1 we can replace LP by the restricted master problem LP'. This ensures that the branching
schemes that are based on these results are also valid when $LP$ is not solved to optimality.

The two branching schemes we propose can be summarized as follows.

**Scheme 1:**

When $x$ is fractional, there exist a task $i \in N$ and a resource $\hat{k} \in M$ with $0 < z^k_i < 1$. Create two subsets characterized by $z^k_i = 0$ and $z^k_i = 1$ respectively.

**Scheme 2:**

When $x$ is fractional, there exist two tasks $i, j \in N$ such that $0 < y_{ij} < 1$. Create two subsets characterized by $y_{ij} = 0$ and $y_{ij} = 1$ respectively.

We can only use scheme 1 when we use set partitioning formulation $P$, i.e. when all resources are nonidentical. When all resources are identical, we must either choose scheme 2, which is then equivalent to the Ryan and Foster branching rule, or use formulation $P$. The latter solution increases problem size, but it allows for the usage of scheme 1. Scheme 1 has the advantage that it does not complicate the pricing problem in the nodes of the search tree. The restriction $z^k_i = 0$ is easily satisfied by ignoring task $i$ when solving pricing problem $S_{\hat{k}}$. It is less trivial to see that the restriction $z^k_i = 1$, i.e. requiring that task $i$ is executed by resource $\hat{k}$ does not complicate the pricing problem either. In fact, any algorithm for the pricing problem $S_{\hat{k}}$ can be used, as is stated by the following theorem.

**Theorem 1** Let $S_{\hat{k}}(i)$ be the problem that arises from $S_{\hat{k}}$ by adding the constraint $z^k_i = 1$, and let $A$ be any algorithm that solves $S_{\hat{k}}$. Then $A$ also solves $S_{\hat{k}}(i)$.

**Proof**

The idea behind the proof is to modify the dual variables such that they still form an optimal dual solution, but all sets $r \in \Omega_{\hat{k}}$ with $i \notin r$ will have $d^k_r \geq 0$. 
Note that we can force task $i$ to be executed by resource $k$ by adding the following branching constraint to the master problem:

$$
\sum_{r \in \Omega_k} (1 - \delta_{ir}) x_r^k + \sum_{i \in M \setminus \{k\}} \sum_{r \in \Omega_k} \delta_{ir} x_r^k \leq 0.
$$

Let $\bar{w}$ be the dual variable associated with this branching constraint. When the restricted master problem has been solved, the reduced costs of the columns in the restricted master problem satisfy

$$
d^k_r = c^k_r - \sum_{i \in r} u_i - v_k - \delta_{ir} w \geq 0 \quad \text{if } r \in \Omega'_k, k \in M \setminus \{\hat{k}\},
$$

and

$$
d^k_i = c^k_i + \sum_{i \in r} u_i - v_k - (1 - \delta_{ir}) w \geq 0 \quad \text{if } r \in \Omega'_k.
$$

Let $\eta > 0$ and consider the following alternative dual solution: $\bar{\pi} := \pi + \eta e_j, \bar{v} := v - \eta x^k$, and $\bar{w} := w - \eta$, where $e_j$ denotes the $j$th unit vector. This alternative dual solution is feasible, since the modified reduced costs $\overline{d}_r^k$ of the columns in the current restricted master problem satisfy

$$
\overline{d}_r^k = c^k_r - \sum_{i \in r} \bar{u}_i - \bar{v}_r - \delta_{ir} \bar{w} - c^k_r + \sum_{i \in r} u_i - v_k - \delta_{ir} w = d_r^k \geq 0
$$

if $r \in \Omega'_k, k \in M \setminus \{\hat{k}\}$,

$$
\overline{d}_r^k = c^k_i + \sum_{i \in r} \bar{u}_i - \bar{v}_r - (1 - \delta_{ir}) \bar{w} = c^k_i - \sum_{i \in r} u_i - v_k - (1 - \delta_{ir}) w = d_r^k \geq 0
$$

if $r \in \Omega'_k$ and $i \in r$, and

$$
\overline{d}_r^k = c^k_i - \sum_{i \in r} u_i - \bar{v}_r - (1 - \delta_{ir}) \bar{w} = c^k_i - \sum_{i \in r} u_i - v_k - (1 - \delta_{ir}) w + 2\eta = d_r^k + 2\eta \geq 0
$$

if $r \in \Omega'_k$ and $i \notin r$. Clearly, the alternative dual solution is also optimal, because $\sum_{r} \bar{\pi}_r + \sum_{k} \bar{v}_r = \sum_{r} u_r + \sum_{k} v_k$. It should be obvious that by
choosing \( \eta \) large enough, we can ensure that \( d_i^r \geq 0 \) for all \( r \in \Omega_k \) with \( i \neq r \).

\( \square \)

For branching scheme 2 there is no such a general result known. For some specific resource assignment problems, however, it is not difficult to adjust an algorithm for the pricing problem, such that it only produces solutions that satisfy all combination restrictions.

2.4.5 Upper bounds

In order to keep the search tree small, we need good lower and upper bounds. To obtain good upper bounds we developed a primal heuristic that tries to construct a feasible solution in each node of the search tree, starting from the current fractional solution and, if successful, tries to improve this solution.

Our construction algorithm is based on the assignment values defined in the previous section. Let \( x \) be the fractional solution to the LP. Then we define for each task \( i \in N \) and each resource \( k \in M \) the fractional assignment value \( z_i^k = \sum_{r \in \Omega_k} \delta_{ir} x_r^k \). The following algorithm now tries to construct a feasible solution:

1. \( N_0 \leftarrow N \).
   
   Sort the pairs \((k, i) \in M \times N\) such that \( z_i^{k_1} \geq z_i^{k_2} \geq z_i^{k_3} \geq \ldots \)
   
   \( t = 1 \).
   
   For each resource \( k \in M \) set \( r_k = \emptyset \).
2. If \( i_t \notin N_0 \) or \( c_{r_k \cup \{i_t\}} = \infty \), then go to 4.
3. Set \( r_{k_t} = r_k \cup \{i_t\} \), and remove \( i_t \) from \( N_0 \).
4. \( t = t + 1 \).
   
   If \( N_0 \neq \emptyset \) and \( t \leq |M||N| \), then go to 2, otherwise stop.

If a solution is found in this way, we subject it to two local search algorithms. They consider two resources and try to decrease the total cost by moving tasks from one resource to the other, or by exchanging two tasks between the resources.

2.5 Implementing a branch-and-price algorithm

In this section we discuss some issues concerning the implementation of a branch-and-price algorithm. The ideas we present extend the basic
column generation scheme by introducing a column pool and approximation algorithms for the pricing problem.

2.5.1 Column management

Observe that any column \( r \in \Omega_k \) with negative reduced cost is a candidate to enter the basis and may therefore be added to the restricted master problem. Consequently, it is not necessary to solve the pricing problem to optimality as long as a column with negative reduced cost can be found. Furthermore, if more than one column with negative reduced cost is found, these columns may be added simultaneously to the restricted master problem.

Based on these observations, we propose a column management system that uses approximation as well as optimization algorithms for the pricing problem. The approximation algorithms try to generate many subsets with negative reduced cost very fast. These subsets are then stored in a column pool. Rather than solving the pricing problem at every iteration, we first search the column pool for columns with negative reduced cost. If such columns exist, one or more of these are selected and added to the restricted master problem. Otherwise, we clean the column pool and invoke the pricing algorithms in order to try to refill the pool. Note that each time the restricted master problem is reoptimized the dual variables change. Therefore, the reduced costs of the columns in the pool have to be updated after every reoptimization. Cleaning the pool consists of removing all columns with reduced cost larger than some threshold \( D_{\max} \geq 0 \). A positive threshold value can be useful, because the reduced costs change after every reoptimization, possibly to a negative value. If the pricing problem is not solved to optimality, \( LP \) cannot be solved to optimality and so \( P \) cannot be solved to optimality either. Therefore, as soon as the approximation algorithms fail to find columns with negative reduced cost, an optimization algorithm must be used to solve the pricing problem to either prove optimality or to find new columns. The column generation scheme we propose now looks as follows:

1. Find initial sets \( \Omega_k \) containing a feasible solution \( x \).
2. Set the column pool equal to \( \emptyset \).
3. Solve the restricted master problem \( LP' \).
4. If the column pool contains columns with negative reduced cost, select some of these columns, add them to the restricted master problem and go to Step 3.

5. Delete columns with reduced cost larger than $D_{\text{max}}$ from the column pool and start the approximation algorithms for the pricing problem. If these are successful, add the generated columns to the pool and go to Step 4.

6. Solve the pricing problems $S_k \ (k \in M)$ to optimality. Let $z_k$ be the optimal solution value of $S_k$. If $[Z_{LP}] = [Z_{LP'} + \sum_{k \in M} z_k]$, then stop, otherwise add the generated columns to the pool and go to Step 4.

There are several ways to choose columns from the column pool to add to the restricted master problem. The first possibility is to select the column with the minimum reduced cost. In this way, the linear program will not grow very rapidly, but a linear program has to be solved for each added column. A second possibility is to select all columns with negative reduced costs from the pool. This will reduce the number of linear programs that have to be solved, but the linear programs will become very large. We have chosen a more adaptive greedy selection scheme that selects partial solutions to problem $P$. More precisely, we select a set of columns with negative reduced costs that correspond to a set of subsets satisfying the following requirements:

- Each task is in at most one subset.
- For each resource at most one subset is chosen.

The set of columns is constructed by successively choosing a column with minimum negative reduced cost such that the two requirements are still satisfied. The selection stops when no more such columns are available in the column pool.

The above column selection mechanism is motivated by two observations. First, adding columns corresponding to partial solutions to $P$ increases the chance of encountering integral solutions during the solution of the master problem. Second, it prevents the addition of similar columns, which could happen if the columns would be selected merely based on their reduced cost and the dual variables are far from being optimal.
2.5.2 Approximation algorithms for the pricing problem

As pointed out in the previous section, it is not necessary to solve the pricing problem by an optimization algorithm as long as columns with negative reduced cost can be found by an approximation algorithm. Because the pricing problem may be very hard to solve, we discuss some methods to find good approximate solutions to this problem. Note that to prove optimality of an LP solution we need to solve the pricing problem to optimality.

Our approximation algorithms for the pricing problem rely on the ability to calculate the marginal assignment cost \( l_k^r(j) \), which is the cost of adding task \( j \) to subset \( r \) for resource \( k \), i.e.

\[
l_k^r(j) = \begin{cases} c_k^{r \cup \{j\}} - c_k^r & \text{if } r \cup \{j\} \in \Omega_k, \\ \infty & \text{otherwise.} \end{cases}
\]

When these marginal assignment costs can be calculated in polynomial time, the approximation algorithms for the pricing problem are polynomial. For some problem types this marginal assignment cost can be easily calculated. For the generalised assignment problem, for example, \( c_k^r \) can be calculated in time \( O(|r|) \). For other problem types it may be \( \mathcal{NP} \)-hard to calculate \( l_k^r(j) \) or \( c_k^r \). For the vehicle routing problem, calculating \( l_k^r(j) \) means calculating a minimum cost traveling salesman tour through all clients \( i \in r \cup \{j\} \). In order to be able to use the approximation algorithms efficiently, we have to use some approximation of \( l_k^r(j) \) in this case.

Construction algorithms

Construction algorithms build subsets from scratch. Define the reduced marginal assignment cost \( D_k^r(j) = l_k^r(j) - u_j \). The construction algorithms initialize a set \( r \) and then repeatedly try to decrease the reduced cost of the set by adding a task \( j \not\in r \), with \( D_k^r(j) < 0 \). If \( D_k^r(j) \geq 0 \) for all tasks not in the set, and \( d_k^r < 0 \), then \( r \) is added to the column pool. When we use branching scheme 1, we can initialize \( r \) as the set of tasks that have been assigned to resource \( k \) in the current node of the search tree.

Improvement algorithms

Improvement algorithms modify existing subsets. Note that the current LP solution provides us with a set of subsets \( r \) with \( d_k^r = 0 \) (at least all
the subsets associated with basic variables). When such a set is used as a starting point for a local search algorithm, we expect to find a set with negative reduced cost very fast. The following two algorithms start with a set \( r \) with \( d_r^x = 0 \) and then try to decrease the reduced cost of the set by deleting tasks from the set and replacing them by other tasks. The first algorithm performs profitable swaps until no such swap can be found. The second algorithm performs a variable-depth search.

The first algorithm works as follows:

1. Let \( d_r^{x_{(i \setminus \{i_0\})\cup\{j_0\}}} = \min \{d_r^{x_{(i \setminus \{i\})\cup\{j\}}} \mid i, j \in N, \delta_{ir} = 1, \delta_{jr} = 0\} \).
2. If \( d_r^{x_{(i \setminus \{i_0\})\cup\{j_0\}}} < d_r \), then set \( r \leftarrow (r \setminus \{i_0\}) \cup \{j_0\} \) and go to Step 1.
3. If \( d_r^x < 0 \), then add \( r \) to the column pool.

At each iteration of the variable-depth search algorithm the best swap is performed, even if this increases the reduced cost. The algorithm maintains a set \( F \subset N \) of tasks that were deleted from the set in a previous iteration. These tasks are not allowed to reenter the subset. The best set found over all iterations is added to the column pool if it has negative reduced cost:

1. \( d_1 = \infty \), \( F = \emptyset \).
2. Let \( d_r^{x_{(i \setminus \{i_0\})\cup\{j_0\}}} = \min \{d_r^{x_{(i \setminus \{i\})\cup\{j\}}} \mid i, j \in N, \delta_{ir} = 1, \delta_{jr} = 0, j \notin F\} \).
3. If \( d_r^{x_{(i \setminus \{i_0\})\cup\{j_0\}}} = \infty \), then go to Step 6, otherwise set \( r \leftarrow (r \setminus \{i_0\}) \cup \{j_0\} \) and \( F \leftarrow F \cup \{i_0\} \).
4. If \( d_r < d_1 \), then set \( d_1 = d_r \) and \( r_1 = r_0 \).
5. Go to Step 2.
6. If \( d_1 < 0 \), then add \( r_1 \) to the column pool.

### 2.5.3 Identical versus nonidentical resources

We indicated in Section 2.2 that for a RAP with identical resources, we have the option of reformulating that problem as a RAP with non-identical resources. In this section we discuss the pros and cons of this reformulation. So, suppose we have a RAP with \( m \) identical resources.
When we use formulation $P_0$, we have $|\Omega|$ variables and one pricing problem. When we use formulation $P$, we have $m|\Omega|$ variables. So the transformation from $P_0$ to $P$ is pseudopolynomial. Furthermore, by using formulation $P$, we obtain $m$ pricing problems. In the root node these pricing problems are identical, but in the other nodes of the search tree they may become different because of the branching constraints. These observations indicate that, when we use formulation $P_0$, we will be able to solve the linear programs faster than when we use formulation $P$.

On the other hand, when we use formulation $P$, we can use branching scheme 1 in order to obtain integral solutions. This branching rule cannot be used with formulation $P_0$. Branching scheme 1 has the advantage that it does not complicate the pricing problems in the nodes of the search tree. To our knowledge, such a general branching rule is not yet known for formulation $P_0$. Formulation $P_0$ requires a new branching rule for each new problem type.

### 2.6 Redundancy of columns

Though both topics are often studied separately, there is a strong relationship between cutting plane and column generation methods. In fact, when we add a column (variable) to a problem, we add a constraint to its dual. The pricing problem, which identifies columns with negative reduced cost, is actually a separation problem in the dual: a negative reduced cost column corresponds to a violated constraint in the dual.

In most of the recent studies on cutting plane methods for integer programming problems, people have tried to identify nonredundant valid inequalities (facets) for the problem studied, and then constructed a separation algorithm that, given a fractional solution to the LP, tries to isolate one of these constraints that is violated by the LP solution. This research inspired us to study redundancy of columns in a set partitioning problem. We call a column redundant, if it corresponds to a redundant constraint in the dual problem. A redundant column can never be part of an optimal solution to $LP$. When we are able to avoid generating redundant columns, we might speed up the process of solving $LP$. We should note, however, that the final goal of a branch-and-price algorithm is to solve an integer program. It is very well possible that an optimal integral solution contains redundant columns.

In order to simplify the presentation of the results on redundancy, we
assume that the sets $\Omega_k$ and the costs $c^k_r$ satisfy a property that is very common in resource assignment problems. We will call it the subcolumn property.

**Definition 3** Let $k \in M$. The set $\Omega_k$ and the cost vector $c^k$ satisfy the subcolumn property if for all $r \in \Omega_k$, $l \in r$, the set $r \setminus \{l\} \in \Omega_k$ and $c^k_{r\setminus\{l\}} \leq c^k_r - 1$.

If the costs only satisfy $c^k_{r\setminus\{l\}} \leq c^k_r$, the subcolumn property can be satisfied by modifying the column costs by $c^k_r' := c^k_r + |r|$. This modification adds a constant term $|N|$ to $\mathbb{Z}_{LP}$ and so it does not change the problem.

Under the subcolumn property, we can formulate the resource assignment problem as a set covering problem, because each optimal solution of such a problem will correspond to a partitioning of the task set $N$. In this section we use the set covering formulation.

### 2.6.1 Identical resources

We consider the RAP in which the number of resources is not restrictive. The linear relaxation of the set covering formulation of this problem is the problem to

\[
\begin{align*}
\text{minimize} & \quad \sum_{r \in \Omega} c_r x_r \\
\text{subject to} & \quad \sum_{r \in \Omega} b_{ir} x_r \geq 1 \quad \text{for all } i \in N, \\
& \quad x_r \geq 0 \quad \text{for all } r \in \Omega.
\end{align*}
\]

The dual polyhedron corresponding to this formulation is

\[
D := \{ u \in \mathbb{R}_+^n \mid \sum_{i \in r} u_i \leq c_r \ (r \in \Omega) \}.
\]

Define for each $r \in \Omega$ the face $F_r$ of $D$ as

\[
F_r := \{ u \in D \mid \sum_{i \in r} u_i = c_r \}.
\]

and for each $\tau \in \Omega$ the set

\[
U_\tau := \{ u \in \mathbb{R}_+^{|\tau|} \mid \sum_{i \in r \subseteq \tau} u_i \leq c_r \quad (r \subseteq \tau), \quad \sum_{i \in \tau} u_i = c_\tau \}.
\]

In order to identify nonredundant columns, we first prove two lemmas.

**Lemma 3** Let $\Omega$ satisfy the subcolumn property and let $\tau \in \Omega$. Then $F_\tau$ is a facet of $D$ if and only if $\dim(F_\tau) = |\tau| - 1$. 

Proof

Without loss of generality let $\pi = \{1, 2, \ldots, R\}$. First suppose that $\dim(U_\pi) = |\pi| - 1 = R - 1$. Then there are $R$ affine independent elements $\hat{u}_{i}^{(j)} \in U_\pi$, $(j \in \pi)$. Now for each $j \in \{1, 2, \ldots, R\}$ and $i \in \{1, 2, \ldots, n\}$, define $u_{i}^{(j)} := u_{i}^{(j)}$ if $i \leq R$ and $u_{i}^{(j)} := 0$ otherwise. For each $j \in \{R + 1, R + 2, \ldots, n\}$ and $i \in \{1, 2, \ldots, n\}$, define $u_{i}^{(j)} := \hat{u}_{i}^{(1)}$ if $i \leq R$, $u_{i}^{(j)} := 1$ if $i = j$ and $u_{i}^{(j)} := 0$ otherwise.

We first show that $u^{(j)} \in D$ for all $j \in \mathcal{N}$. First, let $j \leq R$ and $r \in \Omega$. Then $\sum_{i \in r} u_{i}^{(j)} = \sum_{i \in r \cap \pi} \hat{u}_{i}^{(j)} \leq c_{r \cap \pi} \leq c_r$, where the last inequality follows from the subcolumn property. Second, suppose $R < j \leq n$ and $r \in \Omega$. Analogously, we find for $r \subseteq \pi$, that $\sum_{i \in r} u_{i}^{(j)} = \sum_{i \in r \cap \pi} \hat{u}_{i}^{(1)} \leq c_r$, and for $r \not\subseteq \pi$, that $\sum_{i \in r} u_{i}^{(j)} = \sum_{i \in r \cap \pi} \hat{u}_{i}^{(1)} + 1 \leq c_{r \cap \pi} + 1 \leq c_r$.

One easily verifies that the vectors $u^{(j)}$ are affine independent elements of $F_\pi$. This finishes the first part of the proof.

Now suppose that $F_\pi$ is a facet of $D$. Let $\overline{U}_\pi$ be the set

$$\{u \in \mathbb{R}_{+}^n \mid \sum_{i \in r} u_{i} \leq c_r \ (r \subseteq \pi), \ \sum_{i \in \pi} u_{i} = c_\pi, \ u_{i} = 0 \ (i \not\in \pi)\}.$$ 

Then $\{u \in F_\pi \mid u_{i} = 0 \ (i \not\in \pi)\} \subseteq \overline{U}_\pi$, so we can conclude that $\dim(U_\pi) = \dim(\overline{U}_\pi) \geq \dim(F_\pi) - |N \setminus \pi| = n - 1 - (n - |\pi|) = |\pi| - 1$. □

Lemma 4 Let $\Omega$ satisfy the subcolumn property and let $\pi \subseteq \Omega$. Then $\dim(U_\pi) = |\pi| - 1$ if and only if $\sum_{r \subseteq \pi} \hat{\mu}_r c_r > c_\pi$ for each $\hat{\mu} \in \{\mu \in \mathbb{R}_{+}^n \mid \sum_{r \subseteq \pi} \mu_r c_r \geq 1 \ (i \in \pi), \ \mu_{i} = 0\}$.

Proof

We have the following equivalences:

$$\dim(U_\pi) = |\pi| - 1$$

$$\iff$$

$$\exists \ v \in \mathbb{R}_{+}^n, \exists \ v_0 > 0 : \sum_{i \in r} u_{i} \leq c_r - \varepsilon \ (r \subseteq \pi), \ \sum_{i \in \pi} u_{i} = c_\pi$$

$$\iff$$

$$\max\{\varepsilon \mid \sum_{i \in r} u_{i} \leq c_r - \varepsilon \ (r \subseteq \pi), \ \sum_{i \in \pi} u_{i} = c_\pi, \ u_{i} \geq 0 \ (i \in \pi)\} > 0$$
\[
\min \left\{ \sum_{r \in \mathbb{F}} c_r \lambda_r \mid \sum_{r \in \mathbb{F}} \delta_{ir} \lambda_r \geq 0 \ (i \in \mathbb{F}), \ \sum_{r \in \mathbb{F}} \lambda_r \geq 1, \right. \\
\left. \lambda_r \geq 0 \ (r \neq \mathbb{F}), \ \lambda_{\mathbb{F}} \in \mathbb{R} \right\} > 0
\]

Now suppose that \( \lambda \) satisfies \( \sum_{r \in \mathbb{F}} \delta_{ir} \lambda_r \geq 0 \ (i \in \mathbb{F}), \ \sum_{r \in \mathbb{F}} \lambda_r \geq 1, \lambda_r \geq 0 \ (r \neq \mathbb{F}), \) and \( \sum_{r \in \mathbb{F}} c_r \lambda_r \leq 0. \) We conclude that \( \lambda_{\mathbb{F}} < 0. \) By defining \( \mu_r := -\lambda_r / \lambda_{\mathbb{F}} \), we find that \( \mu \) satisfies \( \sum_{r \in \mathbb{F}} \delta_{ir} \mu_r \geq 1 \ (i \in \mathbb{F}), \) and \( \sum_{r \in \mathbb{F}} c_r \mu_r \leq c_r. \)

From these lemmas the following theorem directly follows.

**Theorem 2** Let \( \Omega \) satisfy the subcolumn property and let \( \mathbb{F} \in \Omega \). Then \( \mathbb{F} \) is a nonredundant column if and only if \( \sum_{r \in \mathbb{F}} c_r \mu_r > c_r \) for each \( \mu \in \{ \mu \in \mathbb{R}_+^{2^{|\mathbb{F}|}} \mid \sum_{r \in \mathbb{F}} \delta_{ir} \mu_r \geq 1 \ (i \in \mathbb{F}), \ \mu_{\mathbb{F}} = 0 \}. \)

In Section 2.1 we gave three examples of problems with identical resources: the BPP, the GCP and the VRP. One easily verifies that the BPP and the GCP satisfy the subcolumn property. The VRP satisfies the subcolumn property when the travel distances satisfy the triangle inequality. We first show that all columns are nonredundant for the BPP and the GCP. Then we give an example of a VRP instance with redundant columns.

**Example**

In the bin packing problem, all columns have cost \( c_r = 1. \) Suppose that \( \mathbb{F} \in \Omega \) is redundant. Then there is a \( \mu \in \mathbb{R}_+^{2^{|\mathbb{F}|}} \), satisfying \( \sum_{r \in \mathbb{F}} \delta_{ir} \mu_r \geq 1 \) for all \( i \in \mathbb{F}, \ \mu_{\mathbb{F}} = 0, \) and \( c_r = 1 \geq \sum_{r \in \mathbb{F}} c_r \mu_r = \sum_{r \in \mathbb{F}} \mu_r. \) Now let \( l \in \mathbb{F}. \)

We find that \( 1 \geq \sum_{r \in \mathbb{F}} \mu_r \geq \sum_{r \in \mathbb{F}} \delta_{ir} \mu_r \geq 1, \) so \( \delta_{ir} = 1 \) for all \( r \in \mathbb{F} \) with \( \mu_r > 0. \) So \( \mu_r = 0 \) for all \( r \in \mathbb{F}, \) and \( \mu_{\mathbb{F}} \geq 1, \) which is a contradiction. We conclude that \( \mathbb{F} \) is not redundant.

We easily verify that this result generally holds when the costs satisfy \( c_r = 1 \ (r \in \Omega). \)

**Example**

Consider an instance of the vehicle routing problem with time windows (VRPTW). This is a vehicle routing problem in which each client specifies a time window in which it must be visited. Now suppose that
The Resource Assignment Problem

\( \tau = \{1, 2, 3, 4\} \), the time windows of clients 1, 2, 3 and 4 are \([0, 1]\), \([0, 5]\), \([0, 3]\), and \([0, 7]\) respectively, and the travel times between locations satisfy \( t_{10} = 1 \) (i.e., \( i \in \tau \)), \( t_{12} = t_{34} = \varepsilon \) and \( t_{13} = 2 \), otherwise, where \( \varepsilon \) satisfies \( 0 < \varepsilon < 1 \). Let \( c_r := A(R_r) + |r| \) where \( R_r \) is the trip corresponding to \( r \), and \( A(R_r) \) is the arrival time at the depot of trip \( R_r \). Observe that \( \Omega \) and the cost function \( c \) satisfy the subcolumn property. We now have

\[
c_{\{1,2,3,4\}} = A(0 \rightarrow 1 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 0) + 4 = 8 + 4 = 12,
\]

\[
c_{\{1,2\}} = A(0 \rightarrow 1 \rightarrow 2 \rightarrow 0) + 2 = 2 + \varepsilon + 2 = 4 + \varepsilon,
\]

and

\[
c_{\{3,4\}} = A(0 \rightarrow 3 \rightarrow 4 \rightarrow 0) + 2 = 2 + \varepsilon + 2 = 4 + \varepsilon.
\]

So \( c_{\{1,2\}} + c_{\{3,4\}} < c_{\{1,2,3,4\}} \), and \( \{1, 2, 3, 4\} \) is a redundant column.  

2.6.2 Nonidentical resources

If \( \mu \) satisfies \( \sum_{r \in \mathcal{R}} \delta_{ir} \mu_r \geq 1 \) for all \( i \in \tau \), and \( \mu_{\tau} = 0 \), then there is some \( i \in \tau \) and some \( r \in \mathcal{R} \), satisfying \( \mu_r > 0 \) and \( i \not\in \mathcal{R} \). Therefore we find that \( \sum_{r \in \mathcal{R}} \mu_r \geq \mu_r + \sum_{r \in \mathcal{R}} \delta_{ir} \mu_r \geq \mu_r + 1 > 1 \). Therefore, the test for redundancy as described above for identical resources is not meaningful for nonidentical resources. In fact, we show in this subsection that, when all resources are distinct, all columns are nonredundant.

When all resources are distinct, the linear relaxation of the set covering formulation of the problem is the problem to

\[
\text{minimize} \quad \sum_{k \in M} \sum_{r \in \Omega_k} c_r^k x_r^k
\]

\text{subject to} \quad \sum_{k \in M} \sum_{r \in \Omega_k} \delta_{ir} x_r^k \geq 1 \quad \text{for all } i \in N,

\sum_{r \in \Omega_k} x_r^k \leq 1 \quad \text{for all } k \in M,

x_r^k \geq 0 \quad \text{for all } k \in M, r \in \Omega_k,

\]

with dual polyhedron

\[
D := \{ (u, v) \in M^u \times M^v \mid \sum_{i \in \mathcal{R}} u_i + v_k \leq c_r^k \ (k \in M, r \in \Omega_k) \}.
\]

Define for each \( k \in M, r \in \Omega_k \) the equality set

\[
F^k := \{ (u, v) \in D \mid \sum_{i \in \mathcal{R}} u_i + v_k = c_r^k \}.
\]
For $\vec{k} \in M$, $\vec{r} \in \Omega_{\vec{k}}$, let $H_{\vec{k}}^{\vec{r}} \subseteq R_{+}^{\left|\vec{r}\right|} \times R_{-}$ be the hyperplane $\sum_{i \in \vec{r}} u_{i} + v = c_{\vec{r}}^{\vec{k}}$, and define

$$U_{\vec{r}}^{\vec{k}} := \{(u, v) \in R_{+}^{\left|\vec{r}\right|} \times R_{-} \mid \sum_{i \in \vec{r}} u_{i} + v \leq c_{\vec{r}}^{\vec{k}} (r \subseteq \vec{r})\} \cap H_{\vec{r}}^{\vec{k}}.$$ 

In order to see that $U_{\vec{r}}^{\vec{k}} \neq \emptyset$, observe that

$$\max \{\sum_{i \in \vec{r}} u_{i} + v \mid \sum_{i \in \vec{r}} u_{i} + v \leq c_{\vec{r}}^{\vec{k}} (r \subseteq \vec{r})\}$$

$$= \min \{\sum_{r \subseteq \vec{r}} c_{\vec{r}}^{\vec{k}} \lambda_{r} \mid \sum_{r \subseteq \vec{r}} \delta_{r} \lambda_{r} \geq 1, \lambda_{r} \geq 0\}$$

$$= \frac{c_{\vec{r}}^{\vec{k}}}{C_{\vec{r}}},$$

which can be easily seen, because the minimization problem has only one solution: $\lambda_{\vec{r}} = 1$ and $\lambda_{r} = 0 (r \subseteq \vec{r}).$

**Theorem 3** Let $\Omega_{k}$ satisfy the subcolumn property for all $k \in M$, let $\vec{k} \in M$ and $\vec{r} \in \Omega_{\vec{k}}$. Then $F_{\vec{r}}^{\vec{k}}$ is a facet of $D$.

**Proof**

We will define $n + m$ affine independent elements of $F_{\vec{r}}^{\vec{k}}$. Let $(\vec{u}, \vec{v}) \in U_{\vec{r}}^{\vec{k}}$ and $\vec{C} := 1 + \max\{c_{\vec{r}}^{\vec{k}} \mid k \in M, r \in \Omega_{k}\}$. Let $(\vec{u}, \vec{v}) \in R_{+}^{n_{r}} \times R_{-}^{m_{r}}$ satisfy $\vec{u}_{i} = \vec{u}_{i}$ if $i \in \vec{r}$, $\vec{u}_{i} = 0$ otherwise, and $\vec{v}_{k} = \vec{v}_{k}$ if $k = \vec{k}$, $\vec{v}_{k} = \vec{v} - \vec{C}$ otherwise. We define the following four sets of vectors in $R_{+}^{n_{r}} \times R_{-}^{m_{r}}$:

$$A := \{(\vec{u}_{i} + e_{i}^{(a)} \vec{C}, \vec{v} - J^{(a)} \vec{C}) \mid i \in \vec{r}\} = \{(u, v)^{(a)} \mid j \in \vec{r}\}$$

$$B := \{(\vec{u}_{i} + e_{i}^{(a)} \vec{C}, \vec{v} - J^{(a)} \vec{C}) + c_{\vec{k}}^{\vec{k}} \} \mid i \notin \vec{r}\} = \{(u, v)^{(a)} \mid j \in N \setminus \vec{r}\}$$

$$\vec{I} := \{(\vec{u}, \vec{v})\} = \{(u, v)^{(a)}\}$$

$$\Delta := \{(\vec{u} - e_{k}^{(a)} \vec{C} \mid k \in M, k \neq \vec{k}\} = \{(u, v)^{(a)} \mid k \in M, k \neq \vec{k}\}$$

where $e_{j}^{(a)}$ is the $j$th unit vector in $R_{+}$ and $J^{(a)}$ is the all-one vector in $R^{2}$. One easily verifies that all vectors $(u, v)$ in $A \cup B \cup I \cup \Delta$ satisfy $\sum_{i \in \vec{r}} u_{i} + v_{\vec{k}} = c_{\vec{r}}^{\vec{k}}$. In order to show that $A \cup B \cup I \cup \Delta \subseteq D$, distinguish three cases:
1. If $k \neq \overline{k}$, $r \in \Omega_k$, and $(u, v) \in A \cup B \cup \Gamma \cup \Delta$, then $\sum_{i \in r} u_i + v_k \leq \sum_{i \in r \cap \overline{\Gamma}} u_i + 1 + \delta^{-\hat{c}} \leq \epsilon_r^{\overline{c}} + 1 - \hat{c} \leq c_r^{}.$

2. If $r \in \Omega_{\overline{k}}$ and $(u, v) \in A \cup \Gamma \cup \Delta$ or $(u, v) \in B$ and $r \subseteq \overline{r}$, then $\sum_{i \in r} u_i + v_k = \sum_{i \in r \cap \overline{\Gamma}} u_i + v_k \leq \epsilon_r^{\overline{c}} \leq c_r^{}.$

3. If $r \in \Omega_{\overline{k}}$ and $(u, v) \in B$ and $r \not\subseteq \overline{r}$, then $\sum_{i \in r} u_i + v_k \leq \sum_{i \in r \cap \overline{\Gamma}} u_i + 1 + v_k \leq \epsilon_r^{\overline{c}} + 1 \leq c_r^{}$, because $|r| > |r \cap \overline{r}|$.

Now suppose that $(U, V) := \sum_{j \in F} \alpha_j(u, v)^{(j)} + \sum_{y \in N \setminus \overline{F}} \beta_j(u, v)^{(y)} + \gamma(u, v)^{(v)} + \sum_{k \in M \setminus \{\overline{k}\}} \delta_k(u, v)^{(k)} = (0, 0)$, and $\sum_{j \in F} \alpha_j + \sum_{y \in N \setminus \overline{F}} \beta_j + \gamma + \sum_{k \in M \setminus \{\overline{k}\}} \delta_k = 0$. Then for $j \in N \setminus \overline{r}$, we find $0 = U_j = \alpha_j$. For $j \in F$, we find $0 = V_j = \beta_j(\sum \alpha_j + \gamma + \sum \delta_k) + \alpha_j \hat{c} = \alpha_j \hat{c}$. Therefore $\alpha_j = 0$ and $\gamma + \sum \delta_k = 0$. For $k \in M, k \neq \overline{k}$, we find $0 = V_k = (\hat{c} - \hat{c})(\sum \delta_k) - \hat{c} \delta_k = -\hat{c} \delta_k$. We conclude that $\alpha = 0, \beta = 0, \gamma = 0$ and $\delta = 0$, such that the vectors in $A \cup B \cup \Gamma \cup \Delta$ are affine independent. $\Box$

2.6.3 An alternative pricing problem

The pricing problem has been introduced to test for optimality of the current solution to the restricted master problem, and if this solution is not optimal, to identify a profitable column. The traditional way to achieve these goals is by minimizing the reduced cost $d_r$ over all feasible columns. This pricing objective is based on the standard pricing rules used in simplex algorithms for solving linear programs. Among all nonbasic variables, the one with minimal reduced cost is selected. If its reduced cost is negative the variable is put into the basis.

In the last several years people have been studying alternative pricing rules for the simplex algorithm. One of the results of this research has been the development of various alternative pricing rules, such as, the steepest edge rule and the descent rule. These seem to work very well for set partitioning problems.

For the RAP with identical resources, the following example shows that the traditional pricing problem may have an optimal solution that corresponds to a redundant column.
Example

Let \( N = \{1, 2, 3\} \), \( \Omega = 2^N \setminus \emptyset \), and let the costs satisfy \( c_r = 4 \) if \( |r| = 1 \), \( c_r = 5 \) if \( |r| = 2 \), and \( c_N = 8 \). Note that \( \frac{1}{2} c_{\{1,2\}} + \frac{1}{2} c_{\{1,3\}} + \frac{1}{2} c_{\{2,3\}} = 7.5 < c_N \), so the set \( N \) corresponds to a redundant column. Now suppose that we start the column generation process with a restricted master problem with column set \( \Omega' = \{\{1\}, \{2\}, \{3\}\} \). An optimal dual solution to this problem is \( u_i = 4 \) \( (i \in N) \). We now find \( d_r = 5 - 2 \cdot 4 = -3 \) if \( |r| = 2 \), and \( d_N = 8 - 3 \cdot 4 = -4 \). Minimizing \( d_r \) now yields the redundant column \( N \). The other columns are only generated in the next iteration. Note that, for an optimal integral solution, the column \( N \) is needed.

We present an alternative objective function for the pricing problem and prove that optimal solutions with respect to this objective are not redundant. In order to be able to prove this result, we first prove two lemmas.

Lemma 5 If \( \Omega \) satisfies the subcolumn property and \( \varpi \in \Omega \) is redundant, then there exists a \( \mu \in \mathbb{R}^{2^{|\Omega|}}_+ \), with \( \mu_{\varpi} = 0 \) and \( \sum_{r \subseteq \varpi} c_r \mu_r \leq c_{\varpi} \), that satisfies \( \sum_{r \subseteq \varpi} \delta_{ir} \mu_r = 1 \) for all \( i \in \varpi \).

Proof

Let \( Q \) be the problem to minimize \( \{ \sum_{r \subseteq \varpi} c_r \mu_r \mid \sum_{r \subseteq \varpi} \delta_{ir} \mu_r \geq 1, \mu \in \mathbb{R}^{2^{|\Omega|}}_+ \} \), with \( \mu_{\varpi} = 0 \) and let \( w \in \mathbb{R}^{2^{|\Omega|}}_+ \) be the dual variables of the set covering constraints in \( Q \). Now let \( \mu^* \) be an optimal solution to \( Q \), let \( w^* \) be the optimal dual values, and suppose that \( c_r - \sum_{i \in r} w^*_i \geq 0 \) for each \( r \subseteq \varpi \).

Because \( \varpi \) is redundant, we have \( \sum_{r \subseteq \varpi} c_r \mu^*_r \leq c_{\varpi} \). Now suppose that, \( \sum_{r \subseteq \varpi} \delta_{ir} \mu^*_r > 1 \), for some \( i \in \varpi \). From the complementary slackness relations we know that \( w^*_i = 0 \). Now let \( r \subseteq \varpi \) satisfy \( l \in r \) and \( \mu^*_r > 0 \).

Because \( \mu^*_r > 0 \), the reduced cost of \( r \) satisfies \( c_r - \sum_{i \in r} w^*_i = 0 \). We now find \( c_{r \setminus \{l\}} - \sum_{i \in r \setminus \{l\}} w^*_i = c_{r \setminus \{l\}} - \sum_{i \in r \setminus \{l\}} w^*_i < c_r - \sum_{i \in r} w^*_i = 0 \), which is a contradiction.

Lemma 6 If \( \Omega \) satisfies the subcolumn property, \( \varpi \in \Omega \), and \( \mu \in \mathbb{R}^{2^{|\Omega|}}_+ \) satisfies \( \sum_{r \subseteq \varpi} \delta_{ir} \mu_r = 1 \) for all \( i \in \varpi \), then \( \sum_{r \subseteq \varpi} |r| \mu_r = |\varpi| \).

Proof

\[ |\varpi| = \sum_{i \in \varpi} 1 = \sum_{i \in \varpi} \sum_{r \subseteq \varpi} \delta_{ir} \mu_r = \sum_{r \subseteq \varpi} |r| \mu_r. \]
We now have all the results to identify an objective function for the pricing problem that avoids the generation of redundant columns.

**Theorem 4** Suppose \( \Omega \) satisfies the subcolumn property, \( r \in \Omega \) is redundant and \( \mu \in \mathbb{R}^{|\Omega|}_+ \) satisfies \( \sum_{r \in \Omega} \delta_{ir} \mu_r = 1 \) for all \( i \in r \), and \( \sum_{r \in \Omega} \epsilon_r \mu_r < c_r \). Then \( r \) cannot be an optimal solution to the problem to minimize

\[
\left\{ \frac{c_r - \sum_{i \in r} u_i}{|r|} \mid r \in \Omega \right\}.
\]

**Proof**

If \( r \subset \Omega \), and \( (c_r - \sum_{i \in \Omega} u_i)/|r| \geq (c_r - \sum_{i \in \Omega} u_i)/|\Omega| \), then \( c_r|r| - c_r|r| \leq |r| \sum_{i \in r} u_i - |\Omega| \sum_{i \in \Omega} u_i \). Now suppose that \( (c_r - \sum_{i \in \Omega} u_i)/|r| \geq (c_r - \sum_{i \in \Omega} u_i)/|\Omega| \), for each \( r \subset \Omega \) with \( \mu_r > 0 \). Then

\[
c_r = \sum_{r \in \Omega} \mu_r c_r
\]

\[
= \frac{1}{|\Omega|} \left( \sum_{r \in \Omega} \mu_r |r| - |r| \sum_{r \in \Omega} \mu_r c_r \right)
\]

\[
= \frac{1}{|\Omega|} \sum_{r \in \Omega} \mu_r (c_r |r| - |r| \epsilon_r)
\]

\[
\leq \frac{1}{|\Omega|} \sum_{r \in \Omega} \mu_r \left( |r| \sum_{i \in r} u_i - |r| \sum_{i \in \Omega} u_i \right)
\]

\[
= \frac{1}{|\Omega|} \left( \sum_{i \in \Omega} \sum_{r \in \Omega} \mu_r |r| - |r| \sum_{r \in \Omega} \mu_r \sum_{i \in \Omega} u_i \right)
\]

\[
= \sum_{i \in \Omega} u_i - \sum_{r \in \Omega} \mu_r \sum_{i \in \Omega} \delta_{ir} u_i
\]

\[
= \sum_{i \in \Omega} u_i - \sum_{i \in \Omega} u_i \sum_{r \in \Omega} \mu_r \delta_{ir}
\]

\[
= 0,
\]

which yields \( c_r \leq \sum_{r \in \Omega} \mu_r c_r \), a contradiction. \( \Box \)
Example

For the example at the beginning of this section, we find in the first iteration \((c_r - \sum_{i \in r} u_i)/|r| = -\frac{2}{3} < -\frac{3}{4} = (c_N - \sum_{i \in N} u_i)/|N|\) if \(|r| = 2\). After having added the columns \(\{1, 2\}\), \(\{1, 3\}\) and \(\{2, 3\}\), we find \(u_1 + u_2 + u_3 = 7\frac{1}{2}\), so column \(N\) will not be generated.

At first sight Theorem 4 seems to introduce a problem. Suppose \(r\) is a redundant column that is part of the optimal integral solution. How are we going to generate this column? The answer to this problem is that Theorem 4 can only be used in the root node of the search tree. Even if the subcolumn property holds in the root, we cannot guarantee that it holds in the other nodes. Suppose for example that we use branching scheme 2 and that we require \(i\) and \(j\) to be executed by the same resource. If \(r \in \Omega\) and \(\{i, j\} \subseteq r\), then \(r \setminus \{i\}\) and \(r \setminus \{j\}\) are not feasible. So when we use \((c_r - \sum_{i \in r} u_i)/|r|\) as an objective for the pricing problem in the nodes other than the root, we can generate redundant columns.

2.7 A general approximation algorithm for the resource assignment problem

The branch-and-price algorithm that we presented in this chapter can be easily turned into an approximation algorithm that works for any type of resource assignment problem. In this section we discuss how such an algorithm can be designed.

Regardless of whether the resources are identical or not, we use set partitioning model \(P\), i.e., we consider the resources as being nonidentical. In this way, we cover the most general version of the RAP and we can use the general branching scheme 1. Because each specific problem type demands its own optimization algorithm for the pricing problem, we do not solve the pricing problems to optimality, i.e., we use the column generation scheme of Section 2.5.1, but we delete Step 6. This causes the overall algorithm to be an approximation algorithm because we cannot guarantee that \(LP\) is solved to optimality. Instead of solving the pricing problems to optimality, we use the approximation algorithms as described in Section 2.5.2. Both these approximation algorithms, and the primal heuristic of Section 2.4.5 require an algorithm that calculates the marginal assignment costs \(I_r^k(j)\). Such an algorithm has to be developed for each specific problem type.
We propose to implement a general approximation algorithm for the RAP as a software system in which only one procedure has to be implemented by the user for each new application. This procedure must, given a set \( r \), a resource \( k \) and a task \( j \not\in r \), calculate the marginal assignment cost \( I^k_r(j) \). The software system itself will take care of all the details of the branch-and-price algorithm.

Compared to the effort needed to develop and implement an entire algorithm for a specific resource assignment problem, the effort needed to implement a marginal assignment cost algorithm for a problem is only small.

In the next chapter, we study the performance of the general approximation algorithm for the pickup and delivery problem with time windows. This is a resource assignment problem with nonidentical resources, for which the pricing problems are extremely difficult to solve to optimality. Our computational results indicate that the general approximation algorithm performs very well for this particular problem type.
3

The Pickup and Delivery Problem with Time Windows

In this chapter, we apply the branch-and-price algorithm for the resource assignment problem of the previous chapter to the general pickup and delivery problem with time windows. We assume in this chapter that each transportation request specifies a single origin and a single destination. This problem is easily interpreted as a RAP by taking the transportation requests as tasks and the vehicles as resources. Because all vehicles have their own home location and can have different capacities, we have a RAP with nonidentical resources.

In order to apply the branch-and-price algorithm, we need an optimization algorithm for the pricing problem, and an algorithm that calculates the marginal assignment costs $l^k_j$.

Dumas, Desrosiers and Soumis [19] have designed and implemented a branch-and-price algorithm for the pickup and delivery problem with time windows that differs from ours in various aspects. They apply branching rules that focus on routing rather than assignment decisions, and they do not use our column management system.

In this chapter, we discuss their optimization algorithm for the pricing problem, which is a dynamic programming algorithm, and their branching rule. We use the dynamic programming algorithm for the
pricing problem in our branch-and-price algorithm and perform some computational experiments. These experiments indicate that our algorithm can solve moderately sized problems to optimality, and that the general approximation algorithm for the RAP performs very well for this particular problem type.

3.1 Problem formulation

Let $N$ be the set of transportation requests. For each transportation request $i \in N$, a load of size $q_i \in N$ has to be transported from origin $i^+$ to destination $i^-$. Define $N^+:={i^+ | i \in N}$ as the set of origins and $N^- := \{i^- | i \in N\}$ as the set of destinations. For each request $i \in N$ the pickup time window is denoted by $[e_i, l_i]$ and the delivery time window by $[e_i, l_i]$. Furthermore, let $M$ be the set of available vehicles. Each vehicle $k \in M$ has a capacity $Q_k \in N$, is available in the interval $[e_k, l_k]$, and is stationed at depot $k^+$. Define $M^+ := \{k^+ | k \in M\}$ as the set of depots. Let $V := N^+ \cup N^- \cup M^+$.

For all $i, j \in V, k \in M$ let $d_{ij}$ denote the travel distance, $t_{ij}$ the travel time, and $c_{ij}^k$ the travel cost.

Now let $\Omega_k$ be the set of all feasible pickup and delivery routes for vehicle $k$ and let $c_r^k$ be the cost of route $r$. We will consider pickup and delivery problems where the primary objective is to minimize the number of vehicles needed to serve all transportation requests and the secondary objective is to minimize the total distance traveled. This is accomplished by taking the route costs

$$c_r^k = F + L_r^k,$$

where $L_r^k$ denotes the length of route $r$ for vehicle $k$ and where $F > |N| \max_{k \in M, r \in \Omega_k} L_r^k$ is a large constant. This cost structure can be achieved by defining the travel costs $c_{ij}^k = d_{ij}$ ($i \neq k^+$), and $c_{k^+j}^k = F + d_{k^+j}$ ($j \neq k^+$). Furthermore let

$$\delta_{ir}^k := \begin{cases} 
1 & \text{if } i \in N \text{ is served on route } r \in \Omega_k, \\
0 & \text{otherwise}.
\end{cases}$$

The pickup and delivery problem can now be formulated as a set partitioning problem as follows:

minimize $\sum_{k \in M} \sum_{r \in \Omega_k} c_r^k x_r^k$.
subject to \[
\sum_{k \in M} \sum_{r \in \Omega_k} s_{ir}^k x_{k}^r = 1 \quad \text{for all } i \in N, \\
\sum_{r \in \Omega_k} x_{k}^r \leq 1 \quad \text{for all } k \in M, \\
x_{k}^r \in \{0,1\} \quad \text{for all } k \in M, r \in \Omega_k.
\]

As in Chapter 2, we denote this formulation by \( P \) and its linear programming relaxation by \( LP \).

### 3.2 The pricing problem

The pricing problem decomposes into several independent problems, one for each vehicle. Let \((u, v)\) be an optimal solution to the dual of the restricted master problem \(LP'\) at some iteration of the column generation process. The pricing problem \(S_k\) for vehicle \(k\) is the problem to minimize

\[
\min \left\{ c_k^r - \sum_{i \in N} s_{ir}^k u_i - v_k \mid r \in \Omega_k \right\},
\]

i.e., the problem of finding a minimum cost route for vehicle \(k\), using a modified cost structure, that serves a subset of the transportation requests. To be more precise, introduce four types of variables: \(z_i\) (\(i \in N\)) equal to 1 if transportation request \(i\) is served by the vehicle and 0 otherwise, \(x_{ij}\) (\((i, j) \in V \times V\)) equal to 1 if the vehicle travels from location \(i\) to location \(j\) and 0 otherwise, \(D_i\) (\(i \in V\)), specifying the departure time at location \(i\), and \(y_i\) (\(i \in V\)), specifying the load of the vehicle arriving at location \(i\).

If the route \(r\) defined by vector \(x\) is a feasible pickup and delivery route for vehicle type \(k\), its reduced cost can be expressed as follows:

\[
d_k^r = \sum_{i \in V} \sum_{j \in V} c_{ij}^k x_{ij} - \sum_{i \in N} u_i z_i - v_k.
\]

When we define \(c_{i+j}^k := c_{i+j}^k - u_i\), \(c_{k+i}^j := c_{k+i}^j - v_k\) and \(c_{ij}^k := c_{ij}^k\) if \(i \notin N^+ \cup \{k^+\}\), then \(d_k^r = \sum_{i \in V} \sum_{j \in V} c_{ij}^k x_{ij}\). Note that, when the original costs \(c_{ij}\) satisfy the triangle inequality, then also the modified costs \(c_{ij}^k\) satisfy the triangle inequality. Define \(q_{k^+} = 0\). The pricing problem \(S_k\) is to minimize

\[
\sum_{i \in V} \sum_{j \in V} c_{ij}^k x_{ij}
\]

subject to

\[
\sum_{j \in V} x_{i+j} = \sum_{j \in V} x_{ji} = z_i \quad \text{for all } i \in N, \tag{1}
\]
The Pickup and Delivery Problem with Time Windows

\[
\sum_{j \in V} x_{i-j} = \sum_{j \in V} x_{j-i} = z_i \quad \text{for all } i \in N, \\
\sum_{j \in V} x_{k+j} = 1, \\
\sum_{i \in V} x_{i+k} = 1, \\
D_{i+} \leq D_{i-} \quad \text{for all } i \in N, \\
x_{ij} = 1 \Rightarrow D_i + t_{ij} \leq D_j \quad \text{for all } i, j \in V, \\
y_{k+} = 0, \\
x_{ij} = 1 \Rightarrow y_{ij} = y_i + y_j \quad \text{for all } i, j \in V, \\
x_{ij} \in \{0, 1\} \quad \text{for all } i, j \in V, \\
z_i \in \{0, 1\} \quad \text{for all } i \in N, \\
\epsilon_i \leq D_i \leq t_i \quad \text{for all } i \in V, \\
0 \leq y_i \leq Q_k \quad \text{for all } i \in V.
\]

By constraints (1) and (2) a vehicle visits either both the origin and the destination of a request or it visits neither of them. Constraints (3) and (4) make sure that the vehicle starts and ends at its depot. Constraints (5), (6), and (11) together form the time and precedence constraints. Constraints (7), (8), and (12) together form the capacity constraints.

3.2.1 An optimization algorithm for the pricing problem

Let \( k^- \) be a copy of the starting location \( k^+ \), representing the end location for vehicle \( k \). Problem \( S_k \) can be viewed as a shortest path problem from \( k^+ \) to \( k^- \) with precedence constraints, capacity constraints and time windows, on the perturbed distance matrix \( c' \). Dumas, Desrosiers and Soumis [19] propose a dynamic programming algorithm, which uses labeling techniques to handle the precedence, capacity and time constraints, to solve this shortest path problem. Because the width of the time windows greatly influences the running time of such an algorithm, the first step in solving the pricing problem for vehicle \( k \), is to perform the following time window reduction for each \( i \in N \):

\[
\begin{align*}
\epsilon_{i,-} &= \min\{\epsilon_{i,-}, t_{k+} - t_{i-k^+}\} \\
\epsilon_{i,+} &= \min\{\epsilon_{i,+}, t_{i-} - t_{i+k^+}\} \\
\epsilon_{i,+} &= \max\{\epsilon_{i,+}, \epsilon_{i,-} + t_{i+k^+}\} \\
\epsilon_{i,-} &= \max\{\epsilon_{i,-}, \epsilon_{i,+} + t_{i-k^+}\}
\end{align*}
\]

This is a preprocessing step that only has to be executed once for each vehicle. The dynamic programming algorithm is based on a network...
representation of the problem. This network consists of a node set \( V_k = \{ k^+ \} \cup N^+ \cup N^- \cup \{ k^- \} \) and an arc set \( A \subseteq V_k \times V_k \) that contains all arcs that are not a priori infeasible. A priori infeasible arcs are characterized as follows:

1. For all \( i \in N \), the arcs \((k^+, i^-), (i^+, k^+)\) and \((i^-, i^+)\) are not in \( A \).

2. For all \( i, j \in N \). If \( q_i + q_j > Q_k \), then \((i^+, j^+), (j^+, i^+), (i^+, j^-), (j^+, i^-), (i^-, j^-), \) and \((j^-, i^-)\) are not in \( A \).

3. For all \( i, j \in V_k \). If \( e_i + t_{ij} > l_j \), then \((i, j) \notin A \).

4. When the travel times \( t_{ij} \) satisfy the triangle inequality, then we have for each \( i, j \in N \):

   - If the path \( j^+ \rightarrow i^+ \rightarrow j^- \rightarrow i^- \) is time infeasible with \( D_{j+} = e_{j+} \), then \((i^+, j^-) \notin A \).
   - If the path \( i^+ \rightarrow i^- \rightarrow j^+ \rightarrow j^- \) is time infeasible with \( D_{j+} = e_{j+} \), then \((i^-, j^+) \notin A \). (Note that this cannot happen after time window reduction and test 3.)
   - If both path \( i^+ \rightarrow j^+ \rightarrow i^- \rightarrow j^- \), and path \( i^+ \rightarrow j^+ \rightarrow j^- \rightarrow i^- \) are time infeasible with \( D_{j+} = e_{j+} \), then \((i^+, j^-) \notin A \).
   - If both path \( i^+ \rightarrow j^+ \rightarrow i^- \rightarrow j^- \) with \( D_{j+} = e_{j+} \), and path \( j^+ \rightarrow i^+ \rightarrow i^- \rightarrow j^- \) with \( D_{j+} = e_{j+} \) are time infeasible, then \((i^-, j^-) \notin A \).

A labeling algorithm for the pricing problem should maintain, at each node \( n \in V_k \), a set of labels \((S^+, S^-, T, Z)_n\) corresponding to the various paths that end in node \( n \). \( S^+ \subseteq N \) is the set of requests that are picked up on the path, \( S^- \subseteq N \) is the set of requests that are delivered on the path, \( T \) is the departure time from \( n \) and \( Z \) is the cost of the path at departure from \( n \). The size of such a state space is exponential, due to the presence of the sets \( S^+ \) and \( S^- \) in the labels.

Initially there is only one label available: \((\emptyset, \emptyset, e_{k+}, -v_k)_{k^+}\). At each iteration of the algorithm, labels are evaluated by extending the corresponding path with one stop in all possible ways. The following extensions of label \((S^+, S^-, T, Z)_n\) (\( n \neq k^- \)) are possible:

1. For all \( i \in N \setminus S^+ \), with \((u, i^+) \in A \), \( T + t_{ni^+} \leq l_i \) and \( q_i + \sum_{j \in S^+} \chi S^- q_j \leq Q_k \), create a new label \((S^+ \cup \{ i \}, S^-, \max\{e_i, T + t_{ni^+}\}, Z + e_{ni^+} - u_i)_{i^+}\). 

2. For all $i \in S^+ \setminus S^-$, with $(n, i^-) \in A$, and $T + t_{ni^-} \leq l_{i^-}$, create a new label $(S^+, S^-, \{i\}, \max\{c_{i^-}, T + t_{ni^-}\}, Z + c_{ni^-})_{i^-}$.

3. If $S^+ = S^-$, and $T + t_{nk^-} \leq l_{k^-}$, then create the label $(S^+, S^-, \max\{c_{k^-}, T + t_{nk^-}\}, Z + c_{nk^-})_{k^-}$. This label represents a feasible solution to the pricing problem.

The algorithm stops when no more labels $(S^+, S^-, T, Z)_n$ with $n \neq k^-$ are available. The optimal solution to the pricing problem is now found by inspecting the set of labels at node $k^-$ and choosing the one with minimal cost.

In order to decrease the number of labels at each iteration and thus speed up the algorithm, the following label elimination criteria can be used:

Feasibility: When the travel times $t_{ij}$ satisfy the triangle inequality, and there is no feasible path that starts in node $n$ at time $T$, then visits all $i^-$ for $i \in S^+ \setminus S^-$, and ends in $k^-$, satisfying all time windows, then label $(S^+, S^-, T, Z)_n$ can be deleted.

Dominance: If two labels $(S^+, S^-, T_1, Z_1)_n$ and $(S^+, S^-, T_2, Z_2)_n$ are such that $T_1 \leq T_2$ and $Z_1 \leq Z_2$, then label $(S^+, S^-, T_2, Z_2)_n$ can be deleted.

When the time windows are tight, these elimination criteria are very effective.

Note that checking the feasibility criterion implies solving a traveling salesman problem with time windows. This becomes very time consuming when $|S^+ \setminus S^-|$ is large. Dumas, Desrosiers and Soumis [19] therefore propose only to test subsets of $S^+ \setminus S^-$ of one and two elements. So, if for some $i \in S^+ \setminus S^-$ the path $n \rightarrow i^- \rightarrow k^-$ with $D_n = T$ is time infeasible, then label $(S^+, S^-, T, Z)_n$ can be deleted. Furthermore, if for some $i, j \in S^+ \setminus S^-$ both path $n \rightarrow i^- \rightarrow j^- \rightarrow k^-$ and path $n \rightarrow j^- \rightarrow i^- \rightarrow k^-$ are time infeasible with $D_n = T$, then label $(S^+, S^-, T, Z)_n$ can be deleted.

There are various strategies to search the state space. One possibility is to have two sets of labels at every node of the network. One set contains all labels that have been generated in the previous iteration, and that are evaluated in the current iteration. The other subset contains all new labels that have been generated at the current iteration. As soon as all labels in the first set of a node have been evaluated, this set can be removed. In this way, at iteration $j$, each label that is created
corresponds to a path with \( j \) stops. Another possibility of searching the state space is by sequentially evaluating the set of labels at the various nodes. After a set has been evaluated, it can be removed. In this way, the sets contain labels that correspond to paths of a varying number of stops. At each iteration, this method provides the opportunity to select that node \( n \in V_k \) for evaluation, that contains the most promising labels.

The state space of the algorithm of Dumas, Desrosiers and Soumis [19], consists of labels \( (R(S^+), T, Z)_n \), where \( R(S^+) = S^+ \setminus S^- \). By storing \( R(S^+) \) instead of \( S^+ \) and \( S^- \) in each label, the state space is reduced, but the constraint \( z_i \in \{0,1\} \) is relaxed to \( z_i \in \mathbb{N} \), allowing the pricing algorithm to produce a route that serves requests more than once. This relaxation provides a second dominance criterion for label elimination, which can be stated as follows.

**Dominance 2:** When the travel costs \( c_{ij} \) satisfy the triangle inequality and two labels \( (R(S^+_1), T_1, Z_1)_n \) and \( (R(S^+_2), T_2, Z_2)_n \) are such that \( R(S^+_1) \subset R(S^+_2), T_1 \leq T_2 \) and \( Z_1 \leq Z_2 \), then label \( (R(S^+_2), T_2, Z_2)_n \) can be deleted.

Note that, even when negative cost cycles are present in the network, we cannot have unbounded solutions by allowing \( z_i > 1 \) when all time windows are finite. When the time windows are tight, it is unlikely that solutions exist with \( z_i > 1 \) for some \( i \in N \).

Although routes that serve some requests more than once can never appear in an integral solution of \( P \), they might decrease \( Z_{LP} \), the optimal value of \( LP \), thus inducing a weaker lower bound than would be found with the restriction \( z_i \in \{0,1\} \). The following example shows that this relaxation may have a great impact on \( Z_{LP} \).

**Example**

Consider the problem where all vehicles are identical, so \( \Omega_k = \Omega \) for all \( k \in M \). Define \( r_j^{(1)} \) as the route that starts at the common depot, serves request \( j \) \( I \) times and then returns to the depot. Suppose that \( \Omega = \{r_j^{(1)} \mid j \in N\} \). The only integral solution uses \( |N| \) vehicles and has a total cost \( Z_{IP} = \sum_{i \in N} c_{i(1)} + |N|F + \sum_{i \in N} L_{i(1)} \). Clearly \( Z_{LP} = Z_{IP} \).

When all routes \( r_j^{(2)} \) are feasible, one easily verifies that, by relaxing \( z_i \in \{0,1\} \) to \( z_i \in \{0,1,2\} \) in the pricing problem, the optimal value of
$LP$ becomes $\sum_{i \in N} \frac{1}{2} c_{i,j}^{(2)} = \frac{1}{2} |N| F + \frac{1}{2} \sum_{i \in N} I_{i,j}^{(2)} \leq \frac{1}{2} (|N| + 1) F$, which is about $\frac{1}{2} \cdot Z_{LP}$.

In our implementation of the labeling algorithm we therefore use the full labels $(S^+, S^-, T, Z)$, allowing us to satisfy the constraint $z_i \in \{0, 1\}$.

### 3.3 Solving the linear program

As pointed out in the previous chapter, it is not necessary to solve the pricing problem by an optimization algorithm as long as columns with negative reduced cost can be found by an approximation algorithm. Because the labeling algorithm is very time consuming, we discuss various strategies to solve the linear program efficiently.

#### 3.3.1 Improving the dynamic programming algorithm

Dumas, Desrosiers and Soumis [19] propose to speed up the dynamic programming algorithm in earlier iterations of the column generation process by working on a reduced network. Network reduction is achieved by deleting nodes corresponding to requests with a low dual value and by deleting arcs with relatively high cost. Obviously, this reduces the state space of the dynamic program and thus the computation times, but it no longer guarantees that the optimal solution is found. If no more profitable routes can be found in the reduced network, it is enlarged and the dynamic programming algorithm is started again. Note that this approach guarantees that in the end the pricing problem is solved to optimality.

Observe that the dynamic programming algorithm may encounter many columns with negative reduced costs before it identifies the one with the smallest reduced cost. Obviously all these columns could be stored in the column pool. However, since the number of columns with negative reduced cost in the first iterations of the column generation process is huge, this would lead to an unmanageable column pool. Furthermore, only a few of the columns generated in the first iterations of the column generation scheme will be actually added to the restricted master problem. Therefore, in our implementation, we have put an upper bound on the number of columns that the algorithm can create in one run for each vehicle. The algorithm will stop as soon as the upper bound is reached, which reduces computation times drastically. Note
that this approach also guarantees that in the end the pricing problem is solved to optimality.

3.3.2 A cheapest insertion algorithm.

We have used the approximation algorithms based on construction and improvement strategies as described in the previous chapter. In order for these algorithms to work, we must be able to calculate the cost $I^k_r(j)$ of adding transportation request $j$ to route $r$.

Since computing the true marginal assignment cost involves the solution of a single-vehicle pickup and delivery problem with time windows, i.e., a traveling salesman problem with time windows, precedence constraints and capacity constraints, we have chosen to work with an approximation of $I^k_r(j)$, namely the cheapest insertion cost. This is the minimal detour needed to add request $j$ to a route $r$, such that the order in which the locations in the original route $r$ are visited does not change.

Suppose that we are given a route $r \in \Omega_k$ that consists of $n+1$ stops. Stop 0 and stop $n$ are at location $k^+$, the other stops are at the origin or destination of some request. Furthermore, we are given a request $j \in N_r$ with $k^+_j = 0$. We want to find the cheapest insertion of request $j$ into route $r$, i.e., we want to find $i^*_0$ and $i^*_1$, such that $0 \leq i^*_0 \leq i^*_1 < n$ and such that the insertion of $j^+$ directly after $i^*_0$ and $j^-$ directly after $i^*_1$, is feasible and gives the minimal detour over all feasible insertions. Note that, if $i^*_0 = i^*_1$, then $j^-$ will directly succeed $j^+$.

The number of candidate pairs $(i_0, i_1)$ is $O(n^2)$, therefore, an algorithm that enumerates all candidate pairs will have a time complexity of $\Omega(n^2)$. The one we present here achieves this bound.

When $(i_0, i_1)$ is a feasible insertion pair, its cost can be calculated as

$$C(i_0, i_1) = d_{i_0 j^+} + d_{j^+ j^-} + d_{j^- j^+_{i_0+1}} - d_{i_0 j^-}$$

if $i_0 = i_1$, and

$$C(i_0, i_1) = d_{i_0 j^+} + d_{j^+ j^-} + d_{j^-_{i_0+1} j^+} + d_{j^- j^-_{i_1+1}} - d_{i_1 j^-}$$

if $i_0 < i_1$.

For $1 \leq i \leq n$, let $[e_i, l_i]$ denote the time window of stop $i$. If $i < n$, this is the time window of the origin or destination of some request. If $i = n$, this is the time window within which the vehicle must return home.

We first calculate $\lambda_i$ ($1 \leq i \leq n$), which is the latest possible time the vehicle may arrive at stop $i$ in order to ensure feasibility, with respect
\[ \lambda_n = l_n \]
for \( i = n - 1 \) downto \( 1 \) do
\[ \lambda_i \leftarrow \min\{l_i, \lambda_{i+1} - l_{i+1}\} \]
od
\( i_0 = n - 1 \)
while \( i_0 \geq 0 \) do
\[ A^+ \leftarrow D_{i_0} + l_{i_0} \]
if \( A^+ \leq l_{j+} \quad \text{and} \quad \bar{q}_{i_0} + q_i \leq Q_k \) then
\[ A^- \leftarrow \max\{e_{j+}, A^+\} + t_{j+i} \]
if \( A^- \leq l_{j-} \) then
\[ \bar{A}_{i_0+1} \leftarrow \max\{e_{j-}, A^-\} + t_{j-i_0+1} \]
if \( \bar{A}_{i_0+1} \leq \lambda_{i_0+1} \) then
Update(\( i_0, i_0 \))
fi
\fi
\[ i_1 = i_0 + 1 \]
\[ \bar{A}_{i_1} = \max\{e_{j+}, A^+\} + t_{j+i_1} \]
if \( \bar{A}_{i_1} \leq \lambda_{i_1} \) then
while \( i_1 < n \quad \text{and} \quad \bar{q}_{i_1} + q_i \leq Q_k \) do
\[ A^- \leftarrow \max\{e_{i_1}, \bar{A}_{i_1}\} + t_{i_1-j} \]
\[ \bar{A}_{i_1+1} \leftarrow \max\{e_{j-}, A^-\} + t_{j-i_1+1} \]
if \( A^- \leq l_{j-} \quad \text{and} \quad \bar{A}_{i_1+1} \leq \lambda_{i_1+i_1} \) then
Update(\( i_0, i_1 \))
fi
\[ \bar{A}_{i_1} = \max\{e_{i_1}, \bar{A}_{i_1}\} + t_{i_1+i_1+1} \]
\[ i_1 = i_1 + 1 \]
\fi
\fi
\[ i_0 = i_0 - 1 \]
od

Figure 3.1: The cheapest insertion algorithm
to the time windows of the remaining part of the route \((i, i+1, \ldots, n)\). These values are calculated in linear time with help of the following recurrence relation:

\[
\lambda_n = l_n
\]

\[
\lambda_i = \min \{ l_i, \lambda_{i+1} - t_{i,i+1} \} \quad \text{for} \quad i = n-1, n-2, \ldots, 1
\]

The set of all candidate pairs \((i_0, i_1)\) is enumerated in the following way: \(i_0\) consecutively takes on values \(n-1, n-2, \ldots, 0\), and for each value of \(i_0\), we consecutively take \(i_1 = i_0, i_0 + 1, \ldots, n-1\).

In Figure 3.1, we present the algorithm. In this algorithm \(A_i, D_i\) and \(\mathcal{Z}_i\) denote the arrival and departure time and the vehicle load at stop \(i\) in the original route \(r\). The arrival times at \(j^+\) and \(j^-\) are denoted by \(A^+\) and \(A^-\) respectively. Furthermore \(\overline{A}_i\) denotes the arrival time at stop \(i\) after having inserted request \(j\) according to the current pair \((i_0, i_1)\).

### 3.4 An alternative branching scheme

As pointed out in the previous chapter, it is sometimes possible to design a branching scheme for a branch-and-price algorithm based on the specific structure of the tasks and the resources.

Dumas, Desrosiers and Soumis [19] present a branching scheme that focuses on routing decisions, rather than on assignment decisions. This branching scheme is based on the one proposed for the asymmetric traveling salesman problem by Carpaneto and Toth [9]. Let \(x^*_r > 0\) be a fractional variable in the current LP solution. Suppose that the corresponding route \(r\) serves \(n\) requests \(\{i_1, i_2, \ldots, i_n\} \subset N\) in such a way that \(i_p\) is picked up before \(i_q\) if \(p < q\). Binary order variables \(O_{ij} \quad (i, j \in N^+ \cup M^+)\) are introduced, where \(O_{ij}\) is equal to 1 if no requests are picked up between locations \(i\) and \(j\), and 0 otherwise. The current subset of solutions is now divided into \(n+2\) subsets, as follows. Define \(i^+_0 = i^+_{n+1} = k^+\). The first subset is characterized by the constraints \(O_{i^+_0 i^+_1} = O_{i^+_1 i^+_2} = \ldots = O_{i^+_n i^+_{n+1}} = 1\). For each \(j \in \{0, 1, \ldots, n\}\) another subset is characterized by \(O_{i^+_0 i^+_j} = O_{i^+_j i^+_2} = \ldots = O_{i^+_n i^+_{j+1}} = 1\) and \(O_{i^+_j i^+_{j+2}} = 0\). The dynamic programming algorithm to solve the pricing problem can be easily modified such that the routes found by the algorithm satisfy the constraints on the order variables.
3.5 Computational experiments

Our goal has been to develop a high-quality approximation algorithm for the PDPTW that can solve fairly large instances in an acceptable amount of computation time. We believe that, with the appropriate choices, our branch-and-price algorithm satisfies these requirements, and the results of the various computational experiments that we have conducted support that claim.

To be able to determine the impact of the various choices that are inherently available in our branch-and-price algorithm, e.g., whether to use approximation or optimization algorithms for the pricing problem, we have implemented several versions of our algorithm. We start our description of these algorithms with a discussion of the implementation issues that are common to all versions.

The cheapest insertion algorithm that we developed for the approximation algorithms for the pricing problem is also used to construct a starting solution and a first set of columns for the set partitioning matrix.

The bound on the number of columns that the dynamic programming algorithm can generate for each vehicle in a single execution was set to 200. We experimented with upper bounds of 50, 200 and 1000 columns per vehicle, but the computation times hardly varied. Because the upper bound of 200 columns per vehicle produced slightly better results, this value was chosen. The threshold value used when the column pool is cleaned up has been set to 10.

All versions of the algorithm use branching scheme 1, i.e., they branch on assignment decisions of transportation requests to vehicles. The branching pair \((k, i)\) is chosen such that \(z^k_i = \max\{z^k_i \mid z^k_i < 1, i \in N, k \in M\}\). The search tree is explored according to a best bound search. We did not implement scheme 2, because the search trees turned out to be fairly small and the effect of a different branching scheme would be minimal.

The first two versions of our algorithm, \(O_1\) and \(O_2\), are optimization algorithms. Both of them use the column generation scheme of Section 3.2, but \(O_1\) only uses the dynamic programming algorithm for column generation, whereas \(O_2\) solves the pricing problem approximately as long as this gives profitable routes.

The next two versions of our algorithm, \(A_1\) and \(A_2\), are approximation algorithms. Both of them use the column generation scheme of Section 3.2, but \(A_1\) never uses the dynamic programming algorithm.
and therefore never solves the pricing problem to optimality, whereas $A_2$ solves the pricing problem optimally in the root when the approximation algorithms fail to produce profitable columns. Note that in the root $A_2$ is equivalent to $O_2$ and solves the LP to optimality. Therefore it provides a valid lower bound on the optimal solution value. In none of the other nodes $A_1$ or $A_2$ guarantee that the LPs are solved to optimality. This implies that for these nodes the lower bounds are not always valid, possibly resulting in the deletion of the node containing the optimal solution.

For comparison purposes, we have also included the more traditional approximation algorithm, $A_3$, that circumvents the complications introduced by branching by only generating columns at the root node and then performing standard branching based on variable dichotomy with the given set of columns.

All versions have been implemented using MINTO, a Mixed INTeger Optimizer [35]. MINTO is a software system that solves mixed-integer linear programs by a branch-and-bound algorithm with linear programming relaxations. It also provides automatic constraint classification, preprocessing, primal heuristics and constraint generation. Moreover, the user can enrich the basic algorithm by providing a variety of specialized application routines that can customize MINTO to achieve maximum efficiency for a problem class. All our computational experiments have been conducted with MINTO 1.6/CPLEX 2.1 and have been run on an IBM/RS6000 model 550.

### 3.5.1 Test problems

As the size of the set of all feasible solutions to an instance of the pickup and delivery problem with time windows strongly depends on the number of transportation requests that can be in a vehicle at the same time and the width of the pickup and delivery time windows, and this size may have a strong relationship with the ability of our algorithms to solve
The Pickup and Delivery Problem with Time Windows

<table>
<thead>
<tr>
<th>Problem</th>
<th>$Z_{opt}$</th>
<th>$Z_{LP}$</th>
<th>gap</th>
<th>Problem</th>
<th>$Z_{opt}$</th>
<th>$Z_{LP}$</th>
<th>gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>A30.1</td>
<td>104291</td>
<td>104263.0</td>
<td>0.05</td>
<td>B30.1</td>
<td>124655</td>
<td>124655.0</td>
<td>0</td>
</tr>
<tr>
<td>A30.2</td>
<td>114451</td>
<td>114451.0</td>
<td>0</td>
<td>B30.2</td>
<td>94341</td>
<td>94332.0</td>
<td>0.01</td>
</tr>
<tr>
<td>A30.3</td>
<td>135415</td>
<td>135415.0</td>
<td>0</td>
<td>B30.3</td>
<td>124882</td>
<td>124872.0</td>
<td>0.01</td>
</tr>
<tr>
<td>A30.4</td>
<td>111549</td>
<td>111549.0</td>
<td>0</td>
<td>B30.4</td>
<td>104431</td>
<td>104431.0</td>
<td>0</td>
</tr>
<tr>
<td>A30.5</td>
<td>114555</td>
<td>114555.0</td>
<td>0</td>
<td>B30.5</td>
<td>94454</td>
<td>94454.0</td>
<td>0.01</td>
</tr>
<tr>
<td>A30.6</td>
<td>114967</td>
<td>114938.8</td>
<td>0.08</td>
<td>B30.6</td>
<td>104294</td>
<td>104246.0</td>
<td>0.02</td>
</tr>
<tr>
<td>A30.7</td>
<td>104247</td>
<td>104247.0</td>
<td>0</td>
<td>B30.7</td>
<td>104263</td>
<td>104262.0</td>
<td>0.01</td>
</tr>
<tr>
<td>A30.8</td>
<td>115154</td>
<td>115146.0</td>
<td>0.08</td>
<td>B30.8</td>
<td>135232</td>
<td>135242.0</td>
<td>0.01</td>
</tr>
<tr>
<td>A30.9</td>
<td>114883</td>
<td>114883.0</td>
<td>0</td>
<td>B30.9</td>
<td>144666</td>
<td>144666.0</td>
<td>0</td>
</tr>
<tr>
<td>A30.10</td>
<td>115208</td>
<td>115208.0</td>
<td>0</td>
<td>B30.10</td>
<td>114685</td>
<td>114685.0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 3.3: Optimal solutions and integrality gaps for problem classes A30 and B30

<table>
<thead>
<tr>
<th>Problem</th>
<th>$Z_{opt}$</th>
<th>$Z_{LP}$</th>
<th>gap</th>
<th>Problem</th>
<th>$Z_{opt}$</th>
<th>$Z_{LP}$</th>
<th>gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>C30.1</td>
<td>84664</td>
<td>84044.0</td>
<td>1.29</td>
<td>D30.1</td>
<td>94660</td>
<td>94460.5</td>
<td>0.44</td>
</tr>
<tr>
<td>C30.2</td>
<td>84674</td>
<td>84074.0</td>
<td>0</td>
<td>D30.2</td>
<td>73855</td>
<td>73852.0</td>
<td>0.03</td>
</tr>
<tr>
<td>C30.3</td>
<td>84011</td>
<td>83061.7</td>
<td>1.18</td>
<td>D30.3</td>
<td>84737</td>
<td>84737.0</td>
<td>0</td>
</tr>
<tr>
<td>C30.4</td>
<td>74205</td>
<td>73168.7</td>
<td>0.86</td>
<td>D30.4</td>
<td>84765</td>
<td>84764.0</td>
<td>0.01</td>
</tr>
<tr>
<td>C30.5</td>
<td>84426</td>
<td>84391.2</td>
<td>0.79</td>
<td>D30.5</td>
<td>84590</td>
<td>84408.5</td>
<td>2.06</td>
</tr>
<tr>
<td>C30.6</td>
<td>84547</td>
<td>84500.5</td>
<td>0.05</td>
<td>D30.6</td>
<td>84257</td>
<td>84217.5</td>
<td>0.22</td>
</tr>
<tr>
<td>C30.7</td>
<td>84279</td>
<td>84187.4</td>
<td>1.13</td>
<td>D30.7</td>
<td>84302</td>
<td>84278.5</td>
<td>0.55</td>
</tr>
<tr>
<td>C30.8</td>
<td>84778</td>
<td>84767.0</td>
<td>0.03</td>
<td>D30.8</td>
<td>74293</td>
<td>74007.0</td>
<td>2.87</td>
</tr>
<tr>
<td>C30.9</td>
<td>84246</td>
<td>84231.2</td>
<td>0.35</td>
<td>D30.9</td>
<td>84110</td>
<td>84110.0</td>
<td>0</td>
</tr>
<tr>
<td>C30.10</td>
<td>84239</td>
<td>84215.2</td>
<td>0.40</td>
<td>D30.10</td>
<td>74499</td>
<td>74402.5</td>
<td>1.14</td>
</tr>
</tbody>
</table>

Figure 3.4: Optimal solutions and integrality gaps for problem classes C30 and D30

instances, we have developed a random problem generator that allows us to vary these instance characteristics.

Instances are constructed as follows. Generate a set of 100 points randomly within a square of size 200 × 200. The distance between two points is the Euclidean distance. The travel time between two points is equal to the distance between these points. Origins ($i^+$), destinations ($i^-$) and vehicle home locations ($k^+$) are now chosen from this set of points. The load of a request is selected from an interval $[q_{min}, q_{max}]$. The capacity of all vehicles is equal to $Q$. The time windows of the requests are constructed in the following way. The planning period has length $L = 600$. Each window has width $W$. For each request $i$ choose $t_i$ randomly within the interval $[0, t_i^{max}]$, where $t_i^{max} = L - t_{i+} - t_{i-}$. The time windows for request $i$ are now calculated as $[t_{i-}, t_{i+} + W]$ for the
<table>
<thead>
<tr>
<th>Problem</th>
<th>( Q_1 ) CPU</th>
<th>Generated</th>
<th>Added</th>
<th>( Q_2 ) CPU</th>
<th>Generated</th>
<th>Added</th>
<th>( \Delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A30.1</td>
<td>197</td>
<td>9342</td>
<td>216</td>
<td>145</td>
<td>1613</td>
<td>204</td>
<td>0.74</td>
</tr>
<tr>
<td>A30.2</td>
<td>32</td>
<td>5747</td>
<td>204</td>
<td>26</td>
<td>885</td>
<td>204</td>
<td>0.81</td>
</tr>
<tr>
<td>A30.3</td>
<td>17</td>
<td>2006</td>
<td>133</td>
<td>17</td>
<td>962</td>
<td>190</td>
<td>1.00</td>
</tr>
<tr>
<td>A30.4</td>
<td>54</td>
<td>4324</td>
<td>180</td>
<td>31</td>
<td>933</td>
<td>221</td>
<td>0.57</td>
</tr>
<tr>
<td>A30.5</td>
<td>51</td>
<td>8392</td>
<td>190</td>
<td>30</td>
<td>791</td>
<td>183</td>
<td>0.59</td>
</tr>
<tr>
<td>A30.6</td>
<td>28</td>
<td>3400</td>
<td>182</td>
<td>22</td>
<td>944</td>
<td>202</td>
<td>0.79</td>
</tr>
<tr>
<td>A30.7</td>
<td>51</td>
<td>5496</td>
<td>193</td>
<td>33</td>
<td>897</td>
<td>190</td>
<td>0.65</td>
</tr>
<tr>
<td>A30.8</td>
<td>18</td>
<td>3282</td>
<td>227</td>
<td>18</td>
<td>862</td>
<td>224</td>
<td>1.00</td>
</tr>
<tr>
<td>A30.9</td>
<td>23</td>
<td>3640</td>
<td>198</td>
<td>15</td>
<td>1033</td>
<td>215</td>
<td>0.65</td>
</tr>
<tr>
<td>A30.10</td>
<td>28</td>
<td>4215</td>
<td>215</td>
<td>21</td>
<td>922</td>
<td>193</td>
<td>0.75</td>
</tr>
<tr>
<td>B30.1</td>
<td>64</td>
<td>3474</td>
<td>162</td>
<td>21</td>
<td>571</td>
<td>165</td>
<td>0.62</td>
</tr>
<tr>
<td>B30.2</td>
<td>89</td>
<td>7097</td>
<td>218</td>
<td>53</td>
<td>1103</td>
<td>235</td>
<td>0.60</td>
</tr>
<tr>
<td>B30.3</td>
<td>36</td>
<td>3447</td>
<td>161</td>
<td>28</td>
<td>600</td>
<td>170</td>
<td>0.83</td>
</tr>
<tr>
<td>B30.4</td>
<td>61</td>
<td>7600</td>
<td>188</td>
<td>32</td>
<td>536</td>
<td>214</td>
<td>0.54</td>
</tr>
<tr>
<td>B30.5</td>
<td>162</td>
<td>9640</td>
<td>217</td>
<td>50</td>
<td>938</td>
<td>214</td>
<td>0.56</td>
</tr>
<tr>
<td>B30.6</td>
<td>57</td>
<td>5460</td>
<td>191</td>
<td>33</td>
<td>960</td>
<td>226</td>
<td>0.58</td>
</tr>
<tr>
<td>B30.7</td>
<td>60</td>
<td>5383</td>
<td>217</td>
<td>40</td>
<td>975</td>
<td>220</td>
<td>0.67</td>
</tr>
<tr>
<td>B30.8</td>
<td>15</td>
<td>2015</td>
<td>183</td>
<td>12</td>
<td>800</td>
<td>171</td>
<td>0.80</td>
</tr>
<tr>
<td>B30.9</td>
<td>28</td>
<td>4502</td>
<td>165</td>
<td>25</td>
<td>807</td>
<td>213</td>
<td>0.59</td>
</tr>
<tr>
<td>B30.10</td>
<td>33</td>
<td>3770</td>
<td>105</td>
<td>38</td>
<td>933</td>
<td>193</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Figure 3.5: Performance of optimization algorithms for \( Z_{LP} \)

pickup and \([c_i + t_{i+1} - c_j + t_{j+1} + W]\) for the delivery. A time unit can be interpreted as a minute. In this way the length of the diagonal of the square corresponds to approximately half a planning period. We choose the number of available vehicles \(|M| = |N|/2\).

As indicated earlier, the objective is to minimize the number of vehicles used and the total distance traveled. We have taken the fixed cost \( F \) to be \( F = 10000 \).

Figure 3.2 lists the problem classes that we have used in our experiments in order of anticipated difficulty. We have randomly generated 10 instances in each problem class.

### 3.5.2 Quality of the lower bound

We first consider the quality of the lower bound \( Z_{LP} \) obtained in the root of the branch and bound tree. Note that this includes the addition of the constraint \( \sum_{k \in M} \sum_{r \in Q_k} \bar{x}_{r}^k \geq \bar{m} \) (see Section 3.4.3). For all instances, the optimal number of vehicles equals \( \bar{m} \). We therefore focus on the quality of the lower bound with respect to the total distance traveled. Figures 3.3 and 3.4 show the linear programming bound at the root \( (Z_{LP}) \), the value of the optimal solution \( (Z_{OPT}) \), and the integrality gap
Figure 3.6: Performance of optimization algorithms for $Z_{L, P}$

with respect to distance traveled only: $(Z_{OPT} - Z_{LP})/(Z_{OPT} - mF)$. For 17 out of 40 instances this gap equals 0, indicating that the problem was solved without any branching, and only for 7 out of 40 instances the gap exceeds 1%.

### 3.5.3 Performance of the optimization algorithms

To analyze the effect of using approximation algorithms in the column generation scheme, we have evaluated the root, i.e., solved the linear program by both algorithm $O_1$ and $O_2$ for all instances in the problem classes A30, B30, C30 and D30. Figures 3.5 and 3.6 show the CPU time, the number of columns generated, and the number of columns added by $O_1$ and $O_2$. The number of columns generated is the total number of columns that have been stored in the column pool during the solution process. The last column shows the ratio $\text{CPU}(O_2)/\text{CPU}(O_1)$.

Algorithm $O_2$ clearly outperforms $O_1$. Over all 40 instances, we have observed an average decrease in computation time of 35% when using the approximation algorithms for the pricing problem. For problem class D30 the average computation time was almost halved. The number of columns in the optimal master problem never becomes very large and
Figure 3.7: Performance of optimization algorithm \( O_2 \)

this number does not differ drastically for \( O_1 \) and \( O_2 \). There is however a big difference in the number of generated columns. This is due to the fact that the dynamic programming algorithm stores all columns with negative reduced cost it encounters (up to 200 per vehicle per execution) in the column pool. Although this results in a larger poolsize for \( O_1 \) than for \( O_2 \), the larger poolsize does not cause the differences in computation times. The differences in computation times can be fully attributed to the fact that \( O_1 \) only uses optimization algorithms to solve the pricing problem whereas \( O_2 \) also uses approximation algorithms.

Based on the above observations, we have chosen algorithm \( O_2 \) to solve all the problem instances to optimality. Figure 3.7 shows the total CPU time and the number of nodes evaluated in the search tree.

### 3.5.4 Performance of the approximation algorithms

Figures 3.8 and 3.9 show the CPU time, the number of evaluated nodes, and the relative error \((Z_{\text{BEST}} - Z_{\text{OPT}})/(Z_{\text{OPT}} - \bar{m}F)\) for the approximation algorithms \( A_1, A_2, \) and \( A_3 \). When a problem is solved in the root, \( A_2 \) and \( A_3 \) are equivalent to \( O_2 \). This event is indicated by an asterisk (*) in the last column.

**Quality**

Algorithm \( A_2 \) clearly outperforms the others with respect to quality of the solutions. It solves 36 out of 40 problems to optimality. For 17 of these problems this is due to the fact that no branching was required and for 19 problems \( A_2 \) found the optimal solution even though branching
<table>
<thead>
<tr>
<th>Problem</th>
<th>$A_1$ CPU</th>
<th>nodes</th>
<th>error</th>
<th>$A_2$ CPU</th>
<th>nodes</th>
<th>error</th>
<th>$A_3$ CPU</th>
<th>nodes</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>A30.1</td>
<td>158</td>
<td>14</td>
<td>0</td>
<td>187</td>
<td>27</td>
<td>0</td>
<td>149</td>
<td>11</td>
<td>0.09</td>
</tr>
<tr>
<td>A30.2</td>
<td>23</td>
<td>4</td>
<td>1.37</td>
<td>26</td>
<td>1</td>
<td>0</td>
<td>26</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>A30.3</td>
<td>7</td>
<td>1</td>
<td>1.70</td>
<td>17</td>
<td>1</td>
<td>0</td>
<td>17</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>A30.4</td>
<td>13</td>
<td>1</td>
<td>1.07</td>
<td>31</td>
<td>1</td>
<td>0</td>
<td>31</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>A30.5</td>
<td>16</td>
<td>1</td>
<td>0.04</td>
<td>30</td>
<td>1</td>
<td>0</td>
<td>30</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>A30.6</td>
<td>58</td>
<td>29</td>
<td>0</td>
<td>66</td>
<td>41</td>
<td>0</td>
<td>23</td>
<td>17</td>
<td>0.46</td>
</tr>
<tr>
<td>A30.7</td>
<td>11</td>
<td>1</td>
<td>4.87</td>
<td>33</td>
<td>1</td>
<td>0</td>
<td>33</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>A30.8</td>
<td>15</td>
<td>2</td>
<td>0.35</td>
<td>19</td>
<td>3</td>
<td>0</td>
<td>19</td>
<td>5</td>
<td>0.10</td>
</tr>
<tr>
<td>A30.9</td>
<td>13</td>
<td>1</td>
<td>1.00</td>
<td>15</td>
<td>1</td>
<td>0</td>
<td>15</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>A30.10</td>
<td>17</td>
<td>3</td>
<td>0.31</td>
<td>21</td>
<td>1</td>
<td>0</td>
<td>21</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B30.1</td>
<td>16</td>
<td>1</td>
<td>2.77</td>
<td>21</td>
<td>1</td>
<td>0</td>
<td>21</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B30.2</td>
<td>16</td>
<td>1</td>
<td>0.25</td>
<td>61</td>
<td>5</td>
<td>0</td>
<td>54</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>B30.3</td>
<td>10</td>
<td>1</td>
<td>2.03</td>
<td>30</td>
<td>5</td>
<td>0.20</td>
<td>26</td>
<td>5</td>
<td>0.20</td>
</tr>
<tr>
<td>B30.4</td>
<td>13</td>
<td>1</td>
<td>0.14</td>
<td>32</td>
<td>1</td>
<td>0</td>
<td>32</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B30.5</td>
<td>50</td>
<td>11</td>
<td>0.07</td>
<td>108</td>
<td>13</td>
<td>0</td>
<td>50</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>B30.6</td>
<td>15</td>
<td>1</td>
<td>2.18</td>
<td>33</td>
<td>1</td>
<td>0</td>
<td>33</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B30.7</td>
<td>20</td>
<td>1</td>
<td>2.05</td>
<td>43</td>
<td>3</td>
<td>0</td>
<td>41</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>B30.8</td>
<td>8</td>
<td>1</td>
<td>0.75</td>
<td>12</td>
<td>1</td>
<td>0</td>
<td>12</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B30.9</td>
<td>12</td>
<td>3</td>
<td>1.50</td>
<td>25</td>
<td>7</td>
<td>0</td>
<td>25</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B30.10</td>
<td>9</td>
<td>1</td>
<td>3.71</td>
<td>38</td>
<td>1</td>
<td>0</td>
<td>36</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 3.8: Performance of approximation algorithms for the problem classes A30 and B30

was required and the pricing problems were only solved to optimality in the root node. For the four problems that were not solved to optimality, the relative error was at most 0.33%.

By comparing the results of $A_2$ and $A_3$, we conclude that it pays off to use column generation during branch and bound. However, as the relative errors of $A_3$ are still small, it is clear that creating a good set of columns in the root is the most important issue in finding good approximate solutions.

Speed

Algorithm $A_1$, which never solves the pricing problem to optimality, outperforms the others with respect to speed. For all problems the optimal number of vehicles was obtained, and though only 9 out of 40 problems are solved to optimality, the average relative error over all 40 problems is only 1.07%. These observations indicate that $A_1$ might be a good algorithm for practical situations where problem sizes are bigger and time and capacity constraints are less restrictive. In fact, in such situations the other algorithms cannot be used because of the
### Table

<table>
<thead>
<tr>
<th>Problem</th>
<th>$A_1$ CPU</th>
<th>nodes</th>
<th>error</th>
<th>$A_2$ CPU</th>
<th>nodes</th>
<th>error</th>
<th>$A_3$ CPU</th>
<th>nodes</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>C30.1</td>
<td>264</td>
<td>41</td>
<td>0.30</td>
<td>250</td>
<td>69</td>
<td>0</td>
<td>114</td>
<td>9</td>
<td>0.75</td>
</tr>
<tr>
<td>C30.2</td>
<td>33</td>
<td>2</td>
<td>0</td>
<td>240</td>
<td>1</td>
<td>0</td>
<td>240</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C30.3</td>
<td>27</td>
<td>1</td>
<td>2.14</td>
<td>380</td>
<td>25</td>
<td>0</td>
<td>321</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>C30.4</td>
<td>222</td>
<td>53</td>
<td>1.07</td>
<td>525</td>
<td>29</td>
<td>0</td>
<td>482</td>
<td>21</td>
<td>0.19</td>
</tr>
<tr>
<td>C30.5</td>
<td>64</td>
<td>9</td>
<td>0.34</td>
<td>424</td>
<td>67</td>
<td>0.18</td>
<td>285</td>
<td>25</td>
<td>0.54</td>
</tr>
<tr>
<td>C30.6</td>
<td>75</td>
<td>21</td>
<td>0</td>
<td>209</td>
<td>71</td>
<td>0</td>
<td>102</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>C30.7</td>
<td>327</td>
<td>102</td>
<td>0</td>
<td>804</td>
<td>127</td>
<td>0</td>
<td>505</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>C30.8</td>
<td>63</td>
<td>11</td>
<td>0.17</td>
<td>218</td>
<td>4</td>
<td>0</td>
<td>216</td>
<td>31</td>
<td>0.30</td>
</tr>
<tr>
<td>C30.9</td>
<td>48</td>
<td>6</td>
<td>0.31</td>
<td>387</td>
<td>19</td>
<td>0.02</td>
<td>309</td>
<td>9</td>
<td>0.12</td>
</tr>
<tr>
<td>C30.10</td>
<td>103</td>
<td>15</td>
<td>0</td>
<td>756</td>
<td>21</td>
<td>0</td>
<td>722</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>D30.1</td>
<td>36</td>
<td>3</td>
<td>0.85</td>
<td>257</td>
<td>9</td>
<td>0</td>
<td>258</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>D30.2</td>
<td>68</td>
<td>12</td>
<td>0.60</td>
<td>1214</td>
<td>21</td>
<td>0</td>
<td>1190</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>D30.3</td>
<td>22</td>
<td>2</td>
<td>2.05</td>
<td>296</td>
<td>1</td>
<td>0</td>
<td>206</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>D30.4</td>
<td>23</td>
<td>1</td>
<td>3.44</td>
<td>191</td>
<td>1</td>
<td>0</td>
<td>101</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>D30.5</td>
<td>1540</td>
<td>556</td>
<td>0</td>
<td>1281</td>
<td>465</td>
<td>0</td>
<td>166</td>
<td>107</td>
<td>0.09</td>
</tr>
<tr>
<td>D30.6</td>
<td>22</td>
<td>1</td>
<td>2.37</td>
<td>306</td>
<td>57</td>
<td>3.33</td>
<td>215</td>
<td>23</td>
<td>0.68</td>
</tr>
<tr>
<td>D30.7</td>
<td>26</td>
<td>1</td>
<td>0.03</td>
<td>243</td>
<td>13</td>
<td>0</td>
<td>230</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>D30.8</td>
<td>984</td>
<td>277</td>
<td>0</td>
<td>725</td>
<td>31</td>
<td>0</td>
<td>658</td>
<td>25</td>
<td>0.63</td>
</tr>
<tr>
<td>D30.9</td>
<td>38</td>
<td>1</td>
<td>2.85</td>
<td>614</td>
<td>1</td>
<td>0</td>
<td>614</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>D30.10</td>
<td>120</td>
<td>14</td>
<td>0</td>
<td>406</td>
<td>101</td>
<td>0</td>
<td>167</td>
<td>11</td>
<td>0.60</td>
</tr>
</tbody>
</table>

Figure 3.9: Performance of approximation algorithms for the problem classes C30 and D30

<table>
<thead>
<tr>
<th>Class</th>
<th>N</th>
<th>M</th>
<th>g&lt;sub&gt;min&lt;/sub&gt;</th>
<th>g&lt;sub&gt;max&lt;/sub&gt;</th>
<th>Q</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>A50</td>
<td>50</td>
<td>25</td>
<td>5</td>
<td>15</td>
<td>15</td>
<td>60</td>
</tr>
<tr>
<td>B50</td>
<td>50</td>
<td>25</td>
<td>5</td>
<td>20</td>
<td>20</td>
<td>60</td>
</tr>
<tr>
<td>DAR30</td>
<td>30</td>
<td>15</td>
<td>1</td>
<td>5</td>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.10: Larger and less restricted problem classes

computation times of the dynamic programming algorithm.

The effectiveness of $A_1$ may be further improved by incorporating more sophisticated iterative improvement algorithms [23, 31].

**Difficult instances**

In a final experiment, we generated two sets of larger instances and one set of less restricted instances, and tested algorithms $A_1$ and $A_2$ on these sets. The characteristics of these classes are shown in Figure 3.10. The set DAR30 is a set of instances of the dial-a-ride problem, which is a well-known special case of the pickup and delivery problem in which loads represent people. In dial-a-ride problems the capacity restrictions are
Figure 3.11: Performance of algorithms $A1$ and $A2$ for the problem classes A50 and B50

fairly loose. Let $Z_{BEST}$ denote the best solution found by the algorithm. The relative error is then approximated by $(Z_{BEST} - [Z_{LP}])/(|Z_{LP} - mF|)$, which is an upper bound on the real relative error. We introduced an upper bound of 2500 on the number of nodes that may be evaluated in the search tree. When this bound causes the algorithm to stop, $Z_{BEST}$ is the best solution found so far.

The results for classes A50 and B50 are shown in Figure 3.11 and the results for the class DAR30 are shown in Figure 3.12. They indicate that the algorithms are indeed capable of providing a high quality solution in an acceptable amount of computation time.

### 3.6 Concluding remarks

We showed that the branch-and-price algorithm that we developed for the resource assignment problem provides a good basis to develop an algorithm that is capable of solving moderately sized pickup and delivery problems to optimality. Our computational results show that the general approximation algorithm for the RAP, as described in Section 2.7,
Figure 3.12: Performance of algorithms A1 and A2 for the problem class DAR30 performs very well for the pickup and delivery problem with time windows.

A more traditional use of set partitioning formulations for routing problems has been to generate a large set of columns heuristically, and then solve the resulting set partitioning problem over this set of columns. Our computational results, specifically those relating to algorithm A3 (see Section 3.5), indicate that linear programming based pricing provides a good way to generate a good set of columns. An additional advantage of this approach is that the resulting set partitioning problem may be solved by more sophisticated techniques such as branch-and-cut. As far as we know, there do not exist methods that successfully combine column generation and cut generation techniques in the context of set partitioning.
DRIVE

In this chapter, we discuss how the algorithm described in Chapters 2 and 3 can be used in a practical environment. For this purpose, we study a pickup and delivery problem that we encountered at Van Gend & Loos BV in the Netherlands.

Van Gend & Loos BV is a company that is part of the Nedloyd group, a Dutch organization providing world wide transportation services by land, air and sea. Van Gend & Loos BV is involved in road transportation of goods within the Benelux, which is the region consisting of the Netherlands, Belgium and Luxemburg. Van Gend & Loos BV is the largest company providing road transportation in the Benelux, with about 1400 vehicles, transporting 160,000 packages from thousands of senders to tens of thousands of addressees per day. The service of Van Gend & Loos BV can be roughly divided into two parts, the regular transportation system and the direct transportation system. In the regular transportation system, shipments, ranging from small packages to loads up to four pallets, are picked up at the sender and then delivered at the closest distribution center. During the night, the loads are transported from that distribution center to the distribution center closest to the destination of the load, from where they are delivered at their final destination during the next day. In the direct transportation system, shipments, ranging in size from four pallets to full truck loads,
are picked up by a vehicle at the sender and are delivered by that same vehicle at the destination. There is no transhipment at any distribution center. This direct transportation system is a practical example of a pickup and delivery situation.

The main goal of our research has been the design and implementation of DRIVE (Dynamic Routing of Independent Vehicles), a planning module that can be incorporated in a decision support system (DSS) for the direct transportation system at Van Gend & Loos BV. The problem that has to be solved with such a DSS can be modeled as a dynamic general pickup and delivery problem.

At an early stage of our research, we decided to develop a set partitioning based algorithm for the problem at Van Gend & Loos BV. The primary reasons for this decision were: A set partitioning approach focuses on the assignment of requests to vehicles, which is the most important and most difficult aspect of a multi-vehicle routing problem. A set partitioning approach is very flexible. When new restrictions are imposed on the routes, they can easily be incorporated, because they only affect the pricing problem. Set partitioning approaches have provided very promising results for various types of vehicle routing problems [8, 11, 19]. Furthermore, we believed that a set partitioning based branch-and-price algorithm, in general, contains many mathematically interesting aspects, which deserved more research. Our research resulted in the development of DRIVE, and the more general branch-and-price algorithm for the resource assignment problem. For ease of presentation, we describe DRIVE by starting with the branch-and-price algorithm of the previous chapters, and then adapting this algorithm to fit the pickup and delivery problem at Van Gend & Loos BV.

We have tested DRIVE on a set of real-life data that was provided by Van Gend & Loos BV. These tests have indicated that DRIVE provides a good basis for the development of a DSS for the Van Gend & Loos direct transportation system.

4.1 Problem characteristics

The vehicle routing problem at Van Gend & Loos BV can basically be modeled as a dynamic general pickup and delivery problem. In this section, we discuss the main characteristics of the problem.
4.1.1 Requests

All requests specify one pickup location and one or more delivery locations, which have to be visited in a predefined order. A single origin-destination pair is called a shipment. A request consisting of multiple delivery locations is said to consist of multiple shipments. These shipments have to be picked up at the same time by a single vehicle. All requests specify time windows for the pickup and all the deliveries. The policy of Van Gend & Loos BV is to serve all requests within their time windows.

Transportation requests are not all known in advance, but may become available in real time. This implies that at the time a new request becomes available, the set of routes that is currently being executed has to be modified. At the beginning of a day, about 60% of all requests that have to be served during that day are known.

4.1.2 Vehicles

The vehicles are not stationed at a central depot, but each vehicle has its own home location. There are various vehicle capacities. These capacities are 14, 16, 24 or 26 pallets. Vehicles are furthermore subdivided according to physical characteristics, because some clients have specific demands on the properties of a vehicle that serves their request.

A working period of a vehicle is a period of consecutive days in which the vehicle is in use. A vehicle starts and ends a working period at its home location with no load on board. Each night of a working period a driver sleeps at one of a set of sleeping locations, which includes his own home location. On Friday night a vehicle has to return home. At this time the vehicle does not have to be empty, meaning that its working period has not yet ended. On Monday morning the vehicles are again available at their home location. A driver must have a 45 minute lunch break between 11 am and 2 pm each day of a working period.

The vehicles are rented by Van Gend & Loos BV on a daily basis for working periods of unknown length. When the number of transportation requests decreases, some vehicles will be sent home in order to end their working period. A number of vehicles is rented permanently, i.e. they cannot be returned. When the number of transportation requests increases, new vehicles are rented. A new vehicle can only start its working period at the beginning of the next working day. This implies that, at the end of a day, when only part of the requests that have to be served
the next day are known, the vehicles for the next day have to be selected. Van Gend & Loos BV does not exchange one vehicle for another, but only returns vehicles when the total number of vehicles that is currently in use exceeds the number of vehicles that will be needed the next day.

4.1.3 Costs

The costs incurred by serving the requests are the drivers' pays, which include the rental costs of the vehicles. Each day of a working period a driver gets paid an amount proportional to the distance traveled that day unless this amount does not exceed some specified minimum, in which case this minimum is paid. Furthermore a driver gets a compensation for each night of a working period that he does not spend at his home location.

4.1.4 Problem size

The planning area, i.e. the region in which all locations are situated, is the Benelux. About 100 vehicles are used per day, of which 50 vehicles are rented permanently. The number of requests that has to be served per day is about 250 to 300. The total number of shipments per day is about 500.

4.2 Current methodology

A central dispatching office collects new requests and assigns them to vehicles. At every stop of a route the driver of a vehicle calls this central office for new instructions. At present, the office cannot directly contact the drivers, although it is anticipated that a more sophisticated communication system will be implemented in the future.

The planning is done by a team of five people. Two of them do the actual planning, the other three act as an interface between the planners and the drivers. All five people take turns on both jobs. The planning area is divided into two parts, each part covered by one planner. The planners basically use two sources of information: a request list with all transportation requests that have to be served within the next twenty hours, and a vehicle list with information about where and when the vehicles that are in use will become empty. On both lists, the items are geographically grouped. On the request list, this grouping is done by considering the origins of the requests. Within a group, the requests are
listed in order of nondecreasing earliest pickup time. On the vehicle list, vehicles within one group are listed in order of nondecreasing time when they become empty.

The planners use a three-phase approach. First they try to find combinations of requests that should be served by one vehicle. Requests are combined, based on proximity of origins and destinations in both space and time, and based on total load. In the second phase, these combinations are tentatively assigned to vehicles. Only when a driver calls for new instructions, these tentative assignments are made permanent.

During the first part of a day the planners only focus on the work that has to be done during that day. In the afternoon, the following day is also considered. At this time the planners must also decide how many vehicles they need for the next day. Though only a fraction of the requests of the next day is known at that time, usually there is some knowledge about the total amount of work that has to be done during the next day. Such knowledge is available as planner's expertise. The number of vehicles that will be used the next day is primarily based on the amount of work that has to be done during the first part of the morning, but is increased if the expected total amount of work for the next day makes this necessary.

4.3 Envisioned methodology

Because the pickup and delivery problem at Van Gend & Loos BV is a dynamic problem, we designed DRIVE to work in an iterative way. When DRIVE is invoked, it produces a plan, based on the current set of routes, the current set of known requests and some estimate on future workload. This plan should be seen as a base plan that a planner can modify. While evaluating the base plan, a planner must focus primarily on the short-term decisions proposed by DRIVE, i.e. the assignments of loads to vehicles, and the routing of the vehicles within, say, the next hour. As soon as the plan is accepted, these short-term decisions are made permanent. This means that the first parts of the routes in the plan will be executed as planned. The remaining parts may be changed when new requests become available. As soon as necessary, DRIVE is used again to produce a new plan that respects all permanent decisions.

In this way, all routes are divided into a head, which is the part that will be executed as planned, and a tail, which is the remaining part that may change in the future. In this environment a planner is
always busy preparing the tails of the routes. When a driver calls for new instructions, no calculations have to be performed because these instructions have been previously stored as the head of the driver’s route.

4.4 Handling the dynamics

**DRIVE** uses an iterative approach to the dynamic problem. At each iteration, a static general pickup and delivery problem is solved. The problem instance at iteration \( t \) depends on the solution of the problem at iteration \( t - 1 \). The problem that has to be solved at an iteration is called the reoptimization problem. In this section, we describe this problem when **DRIVE** is invoked at time \( \tau \).

4.4.1 Vehicles

Let \( M \) be the set of all available vehicles. Each vehicle \( k \in M \) has a capacity \( Q_k \in \mathbb{N} \) and a home location \( k^+ \). Define \( M_1 \subseteq M \) as the set of vehicles that are currently in use and define \( M_0 := M \setminus M_1 \). For all \( k \in M_1 \), let \( R_k \) be the current route of vehicle \( k \), i.e., the route of vehicle \( k \) in the solution to the reoptimization problem of the previous iteration. Let \( head_k \) be the part of \( R_k \) that has been made permanent and let \( tail_k \) be the remaining part of \( R_k \). The starting location of \( tail_k \) is denoted by \( k^+ \) and the arrival time at \( k^+ \) is denoted by \( \tau_k \). In general, \( \tau_k \geq \tau \), but when the dwell time at \( k^+ \) is large, we may have \( \tau_k < \tau \). In any case, the departure time from \( k^+ \) is larger than \( \tau \).

4.4.2 Requests

Each transportation request specifies a set of actions that have to be executed in a predefined order. The first action represents the pickup of the loads and the subsequent actions represent the deliveries of the loads. For each request \( i \), a load of size \( q_i \in \mathbb{N} \) has to be transported from the origin \( i^0 \) to the set of destinations \( N_i^- = \{ i^1, i^2, \ldots \} \). For each \( j \geq 1 \) destination \( i^j \) must be visited before \( i^{j+1} \). Each load is subdivided as follows: \( q_i = q_i^p = -\sum_{j \in N_i^-} q_j \), i.e., positive quantities for pickups and negative quantities for deliveries. For each \( j \in \{ i^0 \} \cup N_i^- \), we denote by \([c_j, l_j] \) the time window in which action \( j \) must start, and by \( \sigma_j \) the duration of action \( j \).

Sometimes, requests are known long before they can be served. It is not necessary to include such requests into the planning process as soon
as they become available. We therefore introduce a planning horizon \( H \geq 0 \), and we only consider those requests that can be picked up before time \( \tau + H \). The parameter \( H \) provides a means to control the size of the problem instance. Let \( \mathcal{N} \) be the set of all transportation requests that are known at time \( \tau \), that have not yet been completed, and that can be picked up before time \( \tau + H \). When some request is served on route \( R_k \) and will have been completed at time \( \tau_k \), i.e. before \( \text{tail}_k \) starts, then it is considered as being completed at time \( \tau \). When request \( i \in \mathcal{N} \) is served on route \( R_k \) and \( i^0 \) will have been visited before time \( \tau_k \), then \( i \) has been permanently assigned to vehicle \( k \). Let \( \mathcal{N}_1 \subseteq \mathcal{N} \) be the set of permanently assigned requests, and define \( \mathcal{N}_0 := \mathcal{N} \setminus \mathcal{N}_1 \).

Suppose that \( k \in \mathcal{M} \) is a vehicle that is currently in use. We define a virtual request \( i_k \) that represents the loads that are on board of the vehicle at \( k_0^+ \). The origin of \( i_k \) is \( k_0^+ \), and the destinations of \( i_k \) are the locations where the loads that are in the vehicle must be delivered. We enforce that vehicle \( k \) picks up \( i_k \) at time \( \tau_k \) by setting the pickup time window \( c_{i_k}^+ = \tau_k = \tau_k \). The set of virtual requests replaces the set of permanently assigned requests \( \mathcal{N}_1 \). For the virtual requests we relax the constraint that \( i_k \) must be visited before \( i_k^{j+1} \) for all \( j \geq 1 \), because \( i_k \) may consist of destinations of various different requests.

We now define \( \mathcal{N} = \mathcal{N}_0 \cup \{i_k \mid k \in \mathcal{M}_1\} \) as the set of requests that have to be served and we enforce that vehicle \( k \in \mathcal{M}_1 \) starts its new route by picking up request \( i_k \).

### 4.4.3 Route costs

The cost of a vehicle on a given day depends on the distance traveled and on the sleeping location of the driver on that day. Suppose vehicle \( k \) travels a distance of \( L > 0 \) on a given day. Let \( s \in \{0, 1\} \) indicate whether the driver sleeps at his home location \( k^+ \) at the end of the day (\( s = 0 \)) or somewhere else (\( s = 1 \)). The amount that has to be paid to the driver for that day is then equal to

\[
\pi_k \max\{L, \overline{L}_k\} + \mu s,
\]

where \( \pi_k \) is a price per unit of distance traveled, \( \overline{L}_k \) is the minimal traveling distance that has to be paid, and \( \mu \) is the compensation that has to be paid if a driver does not sleep at his home location.

Now let \( r \) be a feasible route for vehicle \( k \). In order to define the route costs \( c_r^k \), suppose that route \( r \) uses vehicle \( k \) on days \( 0, 1, \ldots, T_r \), if \( k \in \mathcal{M}_1 \), or on days \( 1, 2, \ldots, T \), if \( k \in \mathcal{M}_0 \), where day 0 denotes the
current day. Let $L_r^k(t)$ denote the distance traveled on day $t$. If $k \in M_0$, then let $s_r = 0$ if the driver sleeps at his home location on day 0 and $s_r = 1$ otherwise. We use the following route costs. If $k \in M_1$, then

$$c_r^k = \mu s_r + \pi_k \sum_{i=0}^{T_r} \left( \alpha \max \{ L_r^k(t), \bar{L}_r \} + (1 - \alpha) L_r^k(t) \right),$$

and if $k \in M_0$, then

$$c_r^k = F + \pi_k \sum_{i=1}^{T_r} \left( \alpha \max \{ L_r^k(t), \bar{L}_r \} + (1 - \alpha) L_r^k(t) \right),$$

for some $F \geq 0$ and $0 \leq \alpha \leq 1$. Note that we do not take into account the sleeping locations of days $t \geq 1$.

For a vehicle that is currently in use, the distance $L_r^k(0)$ will not exceed $\bar{L}_r$ during a large part of the current day. For all vehicles, $L_r^k(t)$ will never exceed $\bar{L}_r$, for $t \geq 1$. When we choose $\alpha = 1$, we therefore obtain a cost structure that, during the major part of the current day, satisfies $c_r^k \approx \mu s_r + (T_r + 1) \pi_k \bar{L}_r$ if $k \in M_1$, and $c_r^k \approx F + T_r \pi_k \bar{L}_r$ if $k \in M_0$, which is almost independent of $r$. By choosing $0 < \alpha < 1$, we reflect in some sense that we want good short-term decisions and still use some of the real cost structure in the model.

By taking the constant $F \gg \pi_k \alpha \bar{L}_r$ for all $k \in M_1$, we discourage exchanging a vehicle that is currently in use for a new vehicle. A new vehicle from $M_0$ is introduced only when a new request becomes available that cannot be served by a vehicle $k \in M_1$. The terms $\pi_k \alpha \max \{ L_r^k(t), \bar{L}_r \} \approx \pi_k \alpha \bar{L}_r$ ($t \geq 1$) add a fixed cost to the route cost $c_r^k$, for every day $t \geq 1$ that vehicle $k$ is in use when executing route $r$. Therefore, the main objective is to minimize the number of vehicles that will be used on the next day. We have defined the cost function in this way, because otherwise the number of vehicles that will be used by DRIVE becomes too large. This is due to the fact that the numbers $\bar{L}_r$ are much smaller than the average distance traveled per day per vehicle in practice. Let $\bar{L}$ be this average. Then we have $\bar{L}/\bar{L}_r \approx 4/3$. When we do not minimize the number of vehicles, we do not distinguish between a solution in which four vehicles are traveling a distance $\bar{L}_r$ each, and a solution in which three vehicles are traveling a distance $\bar{L}$ each and are, together, serving the same requests as the three vehicles in the first solution. The first solution, however, was unacceptable for Van Gend & Loos BV, because it would decrease the average income of the drivers, which would lead to an increase of the values $\pi_k$. 
4.4.4 The reoptimization problem

As indicated before, there are two main objectives in the planning process of each day. During the first part of a day, the planners focus on the work that has to be done during that day. In the afternoon, they start considering the work of the next day, and they determine which vehicles will be used then. We can easily accomplish that the work of the next day is being considered in the afternoon, by properly selecting the horizon $H$.

We define a reoptimization problem $P$, which can be used to obtain good short-term decisions, such as the assignment of requests to vehicles and the creation of routes for vehicles, as well as long-term decisions, such as selecting the vehicles for the next day.

Problem $P$ is based on the set partitioning model discussed in the previous chapters. For all $k \in M$, define $\Omega_k$ as the set of all feasible pickup and delivery routes for vehicle $k$. If $k \in M_1$ and $r \in \Omega_k$, then route $r$ starts with picking up request $i_k$ at time $\tau_k$. For each vehicle $k$ and each route $r \in \Omega_k$, define the constant $\gamma_r^k = Q_k$ if vehicle $k$ is in use the next day when executing route $r$, and $\gamma_r^k = 0$ otherwise. Note that if $k \in M_0$, then $\gamma_r^k = Q_k$ for all $r \in \Omega_k$. Problem $P$ is now defined as the problem to

\[
\text{minimize} \quad \sum_{k \in M} \sum_{r \in \Omega_k} c_r^k x_r^k
\]

subject to

\[
\sum_{k \in M} \sum_{r \in \Omega_k} b_r^k x_r^k = 1 \quad \text{for all } i \in N_0,
\]

\[
\sum_{r \in \Omega_k} x_r^k \leq 1 \quad \text{for all } k \in M_0,
\]

\[
\sum_{r \in \Omega_k} x_r^k = 1 \quad \text{for all } k \in M_1,
\]

\[
\sum_{k \in M} \sum_{r \in \Omega_k} \gamma_r^k x_r^k \geq Q,
\]

\[
x_r^k \in \{0, 1\} \quad \text{for all } k \in M, r \in \Omega_k,
\]

where $Q$ is some estimate of the total vehicle capacity that will be needed during the next day. The constraint $\sum_{k \in M} \sum_{r \in \Omega_k} \gamma_r^k x_r^k \geq Q$ forces a solution to use enough vehicles during the next day, such that it will be possible to add new requests.

During the first part of a day, $Q$ must be set at a low value, say $Q = 0$, such that the constraint $\sum_{k \in M} \sum_{r \in \Omega_k} \gamma_r^k x_r^k \geq Q$ is redundant. At this time, the optimal solutions to $P$ provide good short-term decisions. In the afternoon, we must select the vehicles for the next day. Because the objective function minimizes the number of vehicles, and at that
time only part of the requests that have to be served the next day are known, we must now increase the value of \( \bar{Q} \), because otherwise the optimal solutions to \( P \) would use only enough vehicles to serve the known requests.

We designed DRIVE to be embedded in an interactive decision support system. Interaction between planner and DRIVE is necessary, especially when the vehicles for the next day have to be selected. Suppose, at that time, we first set \( \bar{Q} = 0 \). Let \( \hat{x} \) be an optimal solution to the resulting problem \( P \), and define

\[
\hat{Q} = \sum_{k \in M} \sum_{r \in \Omega_{ak}} \gamma_{kr} x_{kr}^k.
\]

Then \( \hat{Q} \) provides a lower bound on the total vehicle capacity needed for the next day. In an interactive decision support system, \( \hat{Q} \) can be reported to the planner, who must then respond by providing \( \bar{Q} \). Another possibility is to set \( \bar{Q} = \beta \hat{Q} \) for some \( \beta \geq 1 \). The parameter \( \beta \) should be determined experimentally. Its value depends on many aspects. Suppose that, of all requests that have to be served on a day, a fraction \( 0 < f < 1 \) is usually available in the afternoon of the preceding day. When all requests \( j \) for the next day have their time windows \( [\epsilon_j, l_j] = [\Delta, 2\Delta] \), then service can take place at any time during the next day, so the total vehicle capacity needed for the next day is approximately \( \frac{1}{f} \hat{Q} \). We can set \( \beta = \frac{1}{f} \). On the other hand, when the time windows are much tighter, one can expect that most of the requests that have to be served during the first part of a day are already available in the afternoon of the preceding day. In this case \( \beta \) will be much smaller than \( \frac{1}{f} \).

### 4.5 A cheapest insertion algorithm

We want to construct approximate solutions to the reoptimization problems by applying the general approximation algorithm for the resource assignment problem as described in Chapter 2. In order to use this algorithm, we must be able to calculate the marginal assignment costs \( I^k_r(j) \). As in Chapter 3, we approximate \( I^k_r(j) \) by the cheapest insertion cost. In this section, we describe the cheapest insertion algorithm that we use in DRIVE. The cheapest insertion algorithm as described in Chapter 3 cannot be used directly, because the requests can specify multiple delivery locations, and because the routes contain lunch and night breaks.
4.5.1 Breaks

A driver must have two breaks every day of a working period: a night break and a lunch break. A night break is considered as belonging to the preceding day. Let $\Delta$ be the day length. So day $t$ is the time interval $(t\Delta,(t+1)\Delta)$. A break $b_t$ on day $t$ for vehicle $k \in M$ is characterized by a time window $[e_{b_t}, l_{b_t}] \subset (t\Delta,(t+1)\Delta)$ in which it must start, a duration $\sigma_{b_t}$, and a set $V^k_{b_t}$ of locations at which the break can take place.

Vehicle $k$ spends the night at $k^+$, or at a common sleeping location. Let $V_0$ be this set of common sleeping locations. If $b_t$ is the night break of vehicle $k$ on day $t$, then $V^k_{b_t} = \{k^+\}$ if day $t$ is a Friday, and $V^k_{b_t} = V_0 \cup \{k^+\}$ otherwise. The night break location depends on the predecessor and the successor in the route. Suppose we want a night break $b_t$ to take place between locations $u$ and $v$, and suppose that vehicle $k$ departs from $u$ at time $D_u$. The night break takes place at a location $w \in V^k_{b_t}$ for which $D_u + l_{uw} \leq l_{b_t}$ and that minimizes the detour $d_{uw} + d_{uw}$.

Lunch takes place immediately after arriving at some location, immediately before leaving a location, or at some location $j \in V_0 \cup \{k^+\}$ when traveling between two locations. If $b_t$ is the lunch break of vehicle $k$ on day $t$, then $V^k_{b_t}$ consists of $V_0 \cup \{k^+\}$ and all locations $j^i$ ($j \in N, i \geq 0$) that are visited by vehicle $k$ on day $t$. The lunch location depends on the predecessor and the successor in the route. Suppose we want lunch $b_t$ to take place between location $u$ and $v$, and suppose that vehicle $k$ departs from $u$ at time $D_u \leq l_{b_t}$. If $D_u \geq c_{b_t}$, lunch takes place at $u$. Otherwise, let $A_u = D_u + l_{uu}$. If $A_u \leq l_{b_t}$, then lunch takes place at $u$. Otherwise lunch takes place at a location $w \in V_0 \cup \{k^+\}$ for which $D_u + l_{uw} \leq l_{b_t}$ and that minimizes the detour $d_{uw} + d_{uw}$. If $D_u + l_{uw} > l_{b_t}$ for all $w \in V_0 \cup \{k^+\}$, then lunch takes place in $u$.

4.5.2 Route representation

Let $k$ be some vehicle and let $r \in \Omega_k$. Suppose that route $r$ ends at the end of day $j$. Inserting a new request into route $r$ may require that the vehicle is also in use on day $j+1$. When we add a day at the end of a route, we must also introduce a lunch break and a night break for that day. This complicates the insertion algorithm. We therefore assume that, for some $T \geq 1$, all required breaks on days $0, 1, \ldots, T$, are already present in the route, regardless of whether the vehicle will be used on those days or not. We do not allow inserting a request after the last night break. $T$ should be chosen large enough to ensure that this is not
Figure 4.1: A part of the search algorithm for a route with $n = 5$ and a request with $m = 2$

restrictive. In this way, for example, a route for a new vehicle $k \in M_0$ that serves no requests consists of $2T$ stops: $T$ lunch breaks and $T$ night breaks.

The locations of the break stops are not a priori determined. They depend on the predecessor and the successor stops in the route. This implies that the insertion of a request into route $r$ may lead to the relocation of some of these break stops.

4.5.3 The algorithm

Let $r$ be a route for vehicle $k$, consisting of $n+1$ stops. Let $s_i$ ($0 \leq i \leq n$) denote the location of the $i$th stop of $r$. Let $j \in N$ be a request consisting
of \( m + 1 \) actions: \( j^0, j^1, \ldots, j^m \), i.e., one pickup and \( m \) deliveries. We define an insertion vector as a vector \( p \in \{0, \ldots, n - 1\}^m \), satisfying \( p_0 \leq p_1 \leq \cdots \leq p_m \). Insertion vector \( p \) corresponds to inserting action \( j^i \) directly after stop \( p_i \) if \( p_i \neq p_{i-1} \), and directly after \( j^{i-1} \) otherwise. The algorithm searches the set of all candidate insertion vectors and identifies the cheapest among those that are feasible.

When we insert a new request into a route, according to some insertion vector, this yields a detour, which is defined as the difference between the length of the original route and the length of the new route. This detour is not the marginal assignment cost \( I^k_j \). However, our cheapest insertion algorithm searches for the insertion vector that yields the minimal detour. After this vector has been found, its actual cost is evaluated. This result is then used as an approximation for \( I^k_j \). The main reason for using this approximation is that we can design a much faster insertion algorithm when we minimize the detour. Furthermore, the insertion vector that minimizes the detour will very often also minimize the real cost increase.

For each insertion vector, we must check feasibility with respect to vehicle capacity and time windows. Checking feasibility with respect to vehicle capacity can be done as follows. For each \( 0 \leq i \leq n \), let \( y_i \) denote the vehicle load at departure from stop \( i \). Now suppose \( p_0 \in \{0, \ldots, n-1\} \) satisfies \( y_{p_0} + q_{p_0} \leq Q_k \). Then request \( j \) can be picked up directly after stop \( p_0 \). We now define upper bounds \( \hat{p}_i \) on the values \( p_i \) \((1 \leq i \leq m)\) that are valid for this value of \( p_0 \). If \( 1 \leq i \leq m \) and \( p_i \leq \hat{p}_i \), then the vehicle load at departure from stop \( i \) will be at least \( y_i + \sum_{h=0}^{i-1} q_{p_h} \). If this load exceeds the vehicle capacity \( Q_k \), then we must visit action \( j^i \) before stop \( i \). So, for all \( 1 \leq i \leq m \),

\[
\hat{p}_i = \min \{ l \mid p_0 < l \leq n - 1, \ y_l + \sum_{h=0}^{l-1} q_{p_h} > Q_k \} - 1
\]

is a valid upper bound on \( p_i \) if \( j^0 \) is visited directly after stop \( p_0 \).

We search the set of insertion vectors that are feasible with respect to vehicle capacity in the following way:

1. For \( i = 0, 1, \ldots, m \) set \( p_i = n - 1 \) and \( \hat{p}_i = n - 1 \).
2. \( n = 0 \).
3. If \( p \) is feasible with respect to time windows, and yields a shorter detour than the current best solution, then store \( p \).
4. \( n = \max \{ i \mid 0 \leq i \leq m, \ p_i < \hat{p}_i \} \).
4. If \( v = m \), then set \( p_m \leftarrow p_m + 1 \), and go to Step 2.

If \( 0 < v < m \), then set \( p_v \leftarrow p_v + 1 \), \( p_i \leftarrow p_v \) \( (v < i \leq m) \), and go to Step 2.

If \( v = 0 \), and \( p_0 = 0 \) or \( y_i + q_i \rho > Q_k \) for all \( 0 \leq i < p_0 \), then stop.

Otherwise, set \( p_0 \leftarrow \max \{i \mid 0 \leq i < p_0, \; y_i + q_i \rho \leq Q_k\} \), and \( p_i \leftarrow p_0 \) \( (1 \leq i \leq m) \). Calculate \( \hat{p}_i \) for all \( 1 \leq i \leq m \), and go to Step 2.

An example of this search strategy, with infinite vehicle capacity, is shown in Figure 4.1 for a route with \( n = 5 \) and a request with \( m = 2 \).

By searching the set of candidate insertion vectors in this way, we can efficiently check feasibility with respect to time windows and the detour of each vector. We now describe how we check feasibility at Step 2.

For each \( 0 \leq i \leq n \), let \( A_i, D_i, [c_i, l_i] \) and \( \sigma_i \) denote the arrival and departure time, the time window and the service time at stop \( i \). Analogously to the cheapest insertion algorithm in Chapter 3, we first define values \( \lambda_i \) \( (1 \leq i \leq n) \) as

\[
\lambda_i = l_i,
\]

and

\[
\lambda_i = \min \{l_i, \lambda_{i+1} - \sigma_i - l_{i, s_{i+1}}\}, \quad \text{for} \quad i = n - 1, n - 2, \ldots, 1.
\]

\( \lambda_i \) is the latest possible arrival time at location \( s_i \), such that the remaining part of the route \( (s_i, s_{i+1}, \ldots, s_n) \) is feasible with respect to time windows. Note that, when stop \( i \) corresponds to a break, then its location may change by inserting a new request. Therefore, arriving at stop \( i \) before time \( \lambda_i \) only guarantees that the part \( (s_i, s_{i+1}, \ldots, s_n) \) is feasible with respect to time windows if the location of stop \( i \) has not changed.

For \( 0 \leq i \leq n \), let \( \bar{s}_i, \bar{A}_i \), and \( \bar{D}_i \) denote the location, the arrival time and the departure time at stop \( i \) after inserting request \( j \) according to the current insertion vector \( p \), respectively. Note that, when stop \( i \) corresponds to a break, then location \( \bar{s}_i \) may differ from location \( s_i \). We call \( p \) feasible if and only if \( \bar{A}_i \leq l_i \), for all \( 1 \leq i \leq n \), and \( \bar{D}_i \leq l_{j'}, \) for all \( 0 \leq i \leq m \).

Suppose that the current values \( \bar{A}_i, \bar{D}_i \), and the locations \( \bar{s}_i \) \( (1 \leq i \leq n) \), and the values \( \bar{A}_i, \bar{D}_i \) \( (0 \leq i \leq m) \) correspond to the insertion vector directly preceding \( p \). Furthermore, suppose that for this predecessor insertion vector, \( \bar{A}_i \leq l_i \), for all \( 0 \leq i < p_v \), and that \( \bar{A}_i \leq l_{j'} \), for all \( i < v \).
In order to check feasibility of \( p \), we must calculate the new variables \( \overline{A}_i, \overline{D}_i, \overline{s}_i \) (\( 1 \leq i \leq n \)), and \( \overline{A}^i, \overline{D}^i \) (\( 0 \leq i \leq m \)). Note that we do not have to modify all of these values. We only have to evaluate the path \( \overline{s}_{p_v} \rightarrow j^v \rightarrow j^{v+1} \rightarrow \ldots \rightarrow j^n \rightarrow \overline{s}_{p_{v+1}} \), and, in some cases, the predecessor of \( \overline{s}_{p_v} \) and some successors of \( \overline{s}_{p_{v+1}} \). For this evaluation, we distinguish four steps. At each step we calculate some of the variables, and check whether the arrival times satisfy the time windows. If an infeasibility is observed, then the following steps are skipped. Furthermore, if \( \overline{A}_i > l_i \) for some \( i \in \{1, \ldots, n\} \), or \( \overline{A}^i > l^i \) for some \( i \in \{0, \ldots, m\} \), we can usually identify a set of infeasible successors of insertion vector \( p \). These successors can be skipped in the search process.

1. If \( v = 0 \), or \( p_v \leq p_{v-1} + 1 \), or stop \( p_v - 1 \) is not a break, then go to Step 2.
   When \( v > 0 \) and \( p_v \geq p_{v-1} + 2 \), we have a situation that corresponds to steps C and F in Figure 4.1. Because stop \( p_v - 1 \) is a break whose successor changes, its location may have to be changed. Therefore, calculate the new location \( \overline{s}_{p_{v-1}} \), and the new values \( \overline{A}_{p_{v-1}} \) and \( \overline{D}_{p_{v-1}} \). There will always exist a feasible location \( \overline{s}_{p_{v-1}} \).

2. If stop \( p_v \) is a break, then calculate the new location \( \overline{s}_{p_v} \).
   Calculate \( \overline{A}_{p_v} \) and \( \overline{D}_{p_v} \).
   If \( v = 0 \), then \( \overline{A}_{p_v} \leq l_{p_v} \).
   If \( v > 0 \) and \( \overline{A}_{p_v} > l_{p_v} \), then all insertion vectors \( \overline{p} \), with \( \overline{p}_i = p_i \) if \( i < v \), and \( \overline{p}_v \geq p_v \) are infeasible.

3. Set
   \[
   \begin{align*}
   \overline{A}^v & = \overline{D}_{p_v} + t_{p_v, s^r}, \\
   \overline{D}^v & = \max\{\overline{A}^v, e_{s^r}\} + \sigma_{s^r},
   \end{align*}
   
   \text{and for } i = v + 1, v + 2, \ldots, m
   \begin{align*}
   \overline{A}^i & = \overline{D}^{i-1} + t_{j^{i-1}, j^i}, \\
   \overline{D}^i & = \max\{\overline{A}^i, e_{j^i}\} + \sigma_{j^i},
   \end{align*}
   
   \text{If } \overline{A}^i > l^i \text{ for some } v \leq i \leq m, \text{ then all insertion vectors } \overline{p}, \text{ with } \overline{p}_i = p_i \text{ if } i \leq v \text{ are infeasible.}
   \text{Set } i \leftarrow p_v + 1.
4. Calculate $s_i, A_i$ and $B_i$. If location $s_i$ differs from location $s_i$, then set $i ← i + 1$ and repeat Step 4.
If $A_i ≤ A_i$, then insertion vector $p$ is feasible. Otherwise $p$ is infeasible.

If $p$ is feasible, then the corresponding detour can be calculated analogously to the calculation of the arrival times $A_i$ and $B_i$.

4.6 Obtaining approximate solutions

We have used the general approximation algorithm for the resource assignment problem, as described in Section 2.7, as the basis for DRIVE. Because the reoptimization problems are large, and DRIVE must operate in a real-time environment, we modified various parts of the basic algorithm. In this section, we discuss these modifications.

DRIVE uses a heuristic construction algorithm in order to produce a starting solution very fast. This algorithm takes the solution of the previous reoptimization process as a starting point, and then sequentially assigns each new request $j$ to the vehicle $k ∈ M_i$ for which the marginal assignment cost $I^k_i(j)$ is minimal. If $I^k_i(j) = \infty$ for all $k ∈ M_i$, then a new vehicle $k ∈ M_0$ is introduced. After all new requests have been inserted into a route, we try to improve the solution by applying three types of improvement algorithms. The first algorithm reinserts each request into its route. When, after request $j$ has been inserted, new requests have been inserted into the same route, the current positions of the actions of $j$ may no longer be optimal. Reinserting request $j$ may therefore decrease the route cost. The other two algorithms take two routes $r_1 ∈ \Omega_{k_1}$ and $r_2 ∈ \Omega_{k_2}$, and try to find $r'_1 ∈ \Omega_{k_1}$ and $r'_2 ∈ \Omega_{k_2}$ such that $c_{r'_1} + c_{r'_2} < c_{r_1} + c_{r_2}$, either by moving requests from $r_1$ to $r_2$, or by exchanging requests between $r_1$ and $r_2$. If such routes $r'_1$ and $r'_2$ are found, then they are added to the column pool.

In a dynamic environment, a planning tool must be able to provide a good solution within a short amount of computation time. With an LP based branch-and-bound algorithm, we cannot guarantee that a good integral solution is found very fast, if we only use the branching phase to produce these integral solutions. Therefore DRIVE does not use branching, but it uses a primal heuristic in order to obtain integral solutions. This primal heuristic is basically the same algorithm as defined in Section 2.4.5. It is invoked, not only when the LP has been
solved to optimality in the root node, but each time when the column pool does not contain any more profitable columns, i.e., immediately before the pricing heuristics are invoked.

An advantage of using a set partitioning model in a dynamic environment is the fact that many of the columns that have been generated during the solution of the previous reoptimization problem are useful for the current reoptimization problem. Suppose that $r$ is some route for vehicle $k$ that has been generated in the previous reoptimization process, and suppose that $r$ serves requests $j \in N_r$. We now create a new route for vehicle $k$ as follows. Let $r'$ be the route for vehicle $k$ that only serves the virtual request $i_k$. Now sequentially insert all requests $j \in N_r$ that have not been permanently assigned to another vehicle, i.e., all requests $j \in N_r \cap N_0$, into route $r'$. The result is now added to the column pool. By storing all these columns at the beginning of the current reoptimization, we can considerably speed up the solution process.

### 4.7 Results

We have tested DRIVE by simulating a dynamic planning environment with real-life data. These data contained all requests that had been served by Van Gend & Loos BV in a given period. These simulations were based on a stand-alone methodology. Solutions presented by DRIVE were not modified by a planner, but were considered as being executed as proposed.

During the development of DRIVE, we repeatedly compared the results of the simulations to the results of the planners at Van Gend & Loos BV over the same period. These comparisons were needed for various reasons. First, we had to make sure that the data was complete and that we were aware of all constraints that a route should satisfy. Second, we had to compare global solution characteristics, such as the number of vehicles used and the average distance traveled per vehicle per day, in order to assure that these were acceptable to Van Gend & Loos BV. (Remember the discussion at the end of Section 4.4.3.) Finally, we compared the total distance traveled and the total cost over the entire planning period, in order to show that DRIVE is capable of providing good solutions.

DRIVE has been implemented using MINTO [35]. All computational experiments have been conducted with MINTO 1.6/CPLEX 2.1 and
have been run on a SUN SPARCstation ELC.

4.7.1 Organization of the tests

The test data covered a period of 14 working days, starting with a Thursday and ending with a Tuesday. The actual evaluation period covered 10 working days, starting with a Monday, and ending with a Friday. The two additional days at the beginning of the simulation were introduced in order to make sure that the vehicles are not empty on Monday morning, because that would not be realistic. The two additional days at the end of the simulation prevented DRIVE from deferring requests in order to obtain better results for the last days of the evaluation period. We only present results for the evaluation period.

Besides the usual request information, such as addresses of origins and destinations, time windows and load sizes, the test data also contained the times at which the requests became available, i.e., the times at which the requests were called in by the client. We used this data to simulate the process of clients calling in new requests.

The tests have been organized such that we have simulated the invocation of DRIVE once every hour, from 6:00 to 18:00, and once at midnight, for each day of the planning period. We assumed that the solution presented by DRIVE is then executed until DRIVE is invoked again. At 15:00 we let DRIVE select the vehicles for the next day. At this time we first solved the linear programming relaxation of $P$ with $\overline{Q} = 0$. Let $\hat{x}$ denote the solution to this linear program. We have then set $\overline{Q} = \beta \sum_{i \in M} \sum_{t \in T} \gamma^i_t \hat{x}^i_t$ with $\beta = 1.1$, and solved $P$.

In practice, DRIVE must be able to produce a solution within a reasonable amount of time, such that a planner has enough time to evaluate the proposed solution. Therefore we have put an upper bound on CPU time of 10 minutes for each time we invoked DRIVE, with an exception for the run at 15:00, where the upper bound was 20 minutes.

4.7.2 Characteristics of the reoptimization problems

Figure 4.2 shows the sizes of the reoptimization problems at times 0:00, 9:00, 12:00, 15:00 and 18:00 for each day of the two planning weeks. For each reoptimization problem, we list the number of active requests $|\overline{N}|$, the number of unassigned requests $|N_0|$, and the number of vehicles $|M_1|$ currently in use. The set of active requests $\overline{N}$ has been defined by taking the planning horizon $H$ equal to 19 hours. In this way we
<table>
<thead>
<tr>
<th>Week 1</th>
<th></th>
<th></th>
<th></th>
<th>Week 2</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>day</td>
<td>time</td>
<td></td>
<td>N</td>
<td></td>
<td></td>
<td>N₀</td>
<td></td>
</tr>
<tr>
<td>Mon</td>
<td>0:00</td>
<td>328</td>
<td>197</td>
<td>92</td>
<td>Mon</td>
<td>0:00</td>
<td>266</td>
</tr>
<tr>
<td></td>
<td>9:00</td>
<td>300</td>
<td>146</td>
<td>92</td>
<td></td>
<td>9:00</td>
<td>261</td>
</tr>
<tr>
<td></td>
<td>12:00</td>
<td>285</td>
<td>134</td>
<td>92</td>
<td></td>
<td>12:00</td>
<td>237</td>
</tr>
<tr>
<td></td>
<td>15:00</td>
<td>322</td>
<td>170</td>
<td>96</td>
<td></td>
<td>15:00</td>
<td>227</td>
</tr>
<tr>
<td></td>
<td>18:00</td>
<td>329</td>
<td>175</td>
<td>96</td>
<td></td>
<td>18:00</td>
<td>257</td>
</tr>
<tr>
<td>Tue</td>
<td>0:00</td>
<td>349</td>
<td>204</td>
<td>96</td>
<td>Tue</td>
<td>0:00</td>
<td>272</td>
</tr>
<tr>
<td></td>
<td>9:00</td>
<td>332</td>
<td>161</td>
<td>96</td>
<td></td>
<td>9:00</td>
<td>249</td>
</tr>
<tr>
<td></td>
<td>12:00</td>
<td>325</td>
<td>162</td>
<td>96</td>
<td></td>
<td>12:00</td>
<td>257</td>
</tr>
<tr>
<td></td>
<td>15:00</td>
<td>335</td>
<td>193</td>
<td>99</td>
<td></td>
<td>15:00</td>
<td>271</td>
</tr>
<tr>
<td></td>
<td>18:00</td>
<td>333</td>
<td>178</td>
<td>101</td>
<td></td>
<td>18:00</td>
<td>292</td>
</tr>
<tr>
<td>Wed</td>
<td>0:00</td>
<td>354</td>
<td>208</td>
<td>101</td>
<td>Wed</td>
<td>0:00</td>
<td>304</td>
</tr>
<tr>
<td></td>
<td>9:00</td>
<td>345</td>
<td>172</td>
<td>101</td>
<td></td>
<td>9:00</td>
<td>272</td>
</tr>
<tr>
<td></td>
<td>12:00</td>
<td>306</td>
<td>140</td>
<td>101</td>
<td></td>
<td>12:00</td>
<td>254</td>
</tr>
<tr>
<td></td>
<td>15:00</td>
<td>349</td>
<td>186</td>
<td>100</td>
<td></td>
<td>15:00</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>18:00</td>
<td>341</td>
<td>174</td>
<td>100</td>
<td></td>
<td>18:00</td>
<td>261</td>
</tr>
<tr>
<td>Thu</td>
<td>0:00</td>
<td>354</td>
<td>195</td>
<td>100</td>
<td>Thu</td>
<td>0:00</td>
<td>282</td>
</tr>
<tr>
<td></td>
<td>9:00</td>
<td>313</td>
<td>148</td>
<td>100</td>
<td></td>
<td>9:00</td>
<td>264</td>
</tr>
<tr>
<td></td>
<td>12:00</td>
<td>292</td>
<td>125</td>
<td>100</td>
<td></td>
<td>12:00</td>
<td>248</td>
</tr>
<tr>
<td></td>
<td>15:00</td>
<td>285</td>
<td>133</td>
<td>96</td>
<td></td>
<td>15:00</td>
<td>261</td>
</tr>
<tr>
<td></td>
<td>18:00</td>
<td>285</td>
<td>140</td>
<td>96</td>
<td></td>
<td>18:00</td>
<td>277</td>
</tr>
<tr>
<td>Fri</td>
<td>0:00</td>
<td>306</td>
<td>166</td>
<td>96</td>
<td>Fri</td>
<td>0:00</td>
<td>295</td>
</tr>
<tr>
<td></td>
<td>9:00</td>
<td>275</td>
<td>122</td>
<td>96</td>
<td></td>
<td>9:00</td>
<td>278</td>
</tr>
<tr>
<td></td>
<td>12:00</td>
<td>264</td>
<td>99</td>
<td>96</td>
<td></td>
<td>12:00</td>
<td>279</td>
</tr>
<tr>
<td></td>
<td>15:00</td>
<td>256</td>
<td>123</td>
<td>89</td>
<td></td>
<td>15:00</td>
<td>281</td>
</tr>
<tr>
<td></td>
<td>18:00</td>
<td>244</td>
<td>125</td>
<td>90</td>
<td></td>
<td>18:00</td>
<td>279</td>
</tr>
</tbody>
</table>

Figure 4.2: Characteristics of the reoptimization problems for the test set
consider requests with an earliest pickup time of 8:00 the next day, in the planning process starting at 13:00 this day. When we take $H$ much larger, DRIVE spends much CPU time in calculating assignments that will almost certainly have to be changed when new requests become available.

4.7.3 The contribution of column generation

From our test results we observed that it is sometimes difficult to construct a feasible solution to a reoptimization problem, i.e., a solution in which all requests $j \in N_0$ are served. Note that we cannot always serve a new request by introducing a new vehicle. A new vehicle can only start working at the beginning of the next working day, so it cannot serve a new request that has to be picked up during the current day. The construction algorithm that we used to produce a starting solution could not always insert all new requests. At the beginning of the column generation algorithm, all requests that could not be inserted in any route during the creation of the starting solution, get a very high dual value. Therefore the pricing heuristics will automatically try to construct routes that serve these requests.

In order to show the effect of the column generation algorithm, we tested DRIVE with all pricing heuristics deactivated. This did not completely deactivate column generation, because we allowed the relocation and exchange algorithms that were used for improving the starting solution, to store the routes they obtained into the column pool. At the beginning of the column generation phase the column pool therefore was not empty. This reduced form of column generation can sometimes improve the starting solution, but it cannot decrease the number of nonassigned requests. For the first planning week we observed that more than 5% of all requests could not be served by deactivating the pricing heuristics.

We therefore cannot compare the quality of the solutions obtained by DRIVE when the pricing heuristics are deactivated, to the solutions found when all pricing heuristics are active. We conclude that column generation is very important in order to obtain feasible solutions.

4.7.4 Quality of solutions

Figures 1.3 and 4.1 show the results of DRIVE compared to the results of the planners at Van Gend & Loos BV (VGL). For each day of the
<table>
<thead>
<tr>
<th></th>
<th>Nr. of vehicles</th>
<th>Travel distance</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VGL DRIVE</td>
<td>VGL DRIVE</td>
<td>VGL DRIVE</td>
</tr>
<tr>
<td>Mon</td>
<td>92 92</td>
<td>20.1 18.2</td>
<td>20.4 18.6</td>
</tr>
<tr>
<td>Tue</td>
<td>97 96</td>
<td>19.7 19.9</td>
<td>20.1 20.5</td>
</tr>
<tr>
<td>Wed</td>
<td>100 101</td>
<td>20.1 20.1</td>
<td>20.4 20.7</td>
</tr>
<tr>
<td>Thu</td>
<td>100 100</td>
<td>20.8 18.8</td>
<td>21.0 19.2</td>
</tr>
<tr>
<td>Fri</td>
<td>97 96</td>
<td>19.3 16.1</td>
<td>18.1 16.3</td>
</tr>
<tr>
<td>Total</td>
<td>486 485</td>
<td>100.0 93.1</td>
<td>100.0 95.3</td>
</tr>
</tbody>
</table>

Figure 4.3: Results for the first planning week

<table>
<thead>
<tr>
<th></th>
<th>Nr. of vehicles</th>
<th>Travel distance</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VGL DRIVE</td>
<td>VGL DRIVE</td>
<td>VGL DRIVE</td>
</tr>
<tr>
<td>Mon</td>
<td>80 90</td>
<td>21.1 18.1</td>
<td>21.3 19.1</td>
</tr>
<tr>
<td>Tue</td>
<td>77 87</td>
<td>17.5 17.5</td>
<td>18.0 18.5</td>
</tr>
<tr>
<td>Wed</td>
<td>85 90</td>
<td>21.0 21.1</td>
<td>21.3 21.3</td>
</tr>
<tr>
<td>Thu</td>
<td>84 86</td>
<td>19.1 17.0</td>
<td>19.4 17.9</td>
</tr>
<tr>
<td>Fri</td>
<td>87 92</td>
<td>21.3 20.0</td>
<td>20.0 19.5</td>
</tr>
<tr>
<td>Total</td>
<td>413 445</td>
<td>100.0 93.7</td>
<td>100.0 96.3</td>
</tr>
</tbody>
</table>

Figure 4.4: Results for the second planning week

planning period we show the number of vehicles used, the total distance traveled, and the total cost of the solutions. The travel distances and the costs have been scaled, such that the total travel distance and the total cost per week equal 100.0 for the solution of Van Gend & Loos BV.

We observe that for the first planning week, DRIVE obtained an improvement of 4.7% of the total cost compared to the planners at Van Gend & Loos BV. For the second week, this improvement was 3.7%. The number of vehicles used by DRIVE is not substantially smaller than the number used by the planners at Van Gend & Loos BV. For the second week, it is even 7.7% higher. Apparently, DRIVE obtains the cost decrease by constructing better assignments of requests to vehicles.

The characteristics in Figure 4.2 indicate that the first planning week is much busier than the second week. This is reflected in the difference between the solutions of DRIVE and the solutions of the planners. In a
busy period, there are so many active requests that the planners must reduce the planning horizon in order to keep the problem manageable. At this time, DRIVE provides much better solutions, because it is able to look further into the future. In a quiet period, a planner has more time to make decisions, so he can better evaluate the effect of various assignments. In such a period, the cost decrease that DRIVE obtains is smaller but can still be significant.

4.8 Concluding remarks

Although we tested DRIVE only in a simulated environment, we believe that the results indicate that DRIVE will provide a good basis for developing a decision support system at Van Gend & Loos BV. In some sense, the simulated environment in which we tested DRIVE was not realistic. In practice, it sometimes happens that a vehicle breaks down, or that a load is not yet available at an origin at its indicated earliest pickup time. Such situations, which increase total cost, did not occur in our simulations. On the other hand, our tests have indicated that DRIVE, when implemented as a stand-alone system, is capable of providing solutions that are better than those provided by the planners at Van Gend & Loos BV. When DRIVE is embedded in a DSS, its solutions serve as a starting point for these planners. We may therefore expect that the cost decrease that we observed when DRIVE is used as a stand-alone system, will be even more significant when DRIVE is embedded in a DSS.

During the development of DRIVE, we have identified several research topics on dynamic pickup and delivery problems that deserve more attention in the future. First, we believe that there is not yet a good insight in objective functions for dynamic routing problems. Usually there is some knowledge of a cost structure that can be used for cost evaluation after the routes have been executed, but it may be clear from our discussion in Section 4.4.3 that such a cost structure does not always provide a good objective function for a reoptimization problem. A second research topic is the development of techniques for predicting requests that are not yet available. If we would know the origin and load of such requests long before they are actually called in, this might help in producing better plans.
Bibliography


Bibliography


[34] A. Mehrotra and M. A. Trick, A column generation approach to graph coloring, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburg, PA (1993).


Samenvatting

In dit proefschrift bestuderen we het rechtstreeks-vervoerprobleem. Hierin is een verzameling wagens gegeven en een verzameling opdrachten die door de wagens moeten worden uitgevoerd. Elke opdracht bestaat uit het ophalen van een lading op een bepaalde locatie (de oorsprong), en het afleveren van die lading op een of meer andere locaties (de bestemmingen). Een wagen die een lading ophaalt, levert deze zelf ook af. Er treedt dus geen overslag op andere locaties op. Het doel is om alle opdrachten uit te voeren met behulp van de beschikbare wagens tegen minimale kosten.

Het rechtstreeks-vervoerprobleem komt men in de praktijk op verschillende plaatsen tegen. In het vrachtvervoer stellen klanten tegenwoordig vaak eisen aan ophaal- of aflevertijden waaraan met behulp van de standaard overslag-transportsystemen niet is te voldoen, waardoor transportsystemen gebaseerd op rechtstreeks vervoer steeds belangrijker worden. Ook het vervoer van mensen met behulp van taxi’s en taxibusjes is een situatie waarin men het rechtstreeks-vervoerprobleem tegenkomt. Vooral voor de dienstverlening aan ouderen en gehandicapten zijn vervoersystemen ontwikkeld waarin mensen thuis worden opgehaald en naar bijvoorbeeld familie, arts of theater worden gebracht en vice versa.

We onderscheiden rechtstreeks-vervoerproblemen waarin één voertuig alle opdrachten moet uitvoeren en problemen met verscheidene
voertuigen. Bij aanvang van ons onderzoek bleek dat er voor het probleem met één voertuig reeds zeer bevredigende oplossings- en benaderingsmethoden beschikbaar waren. Wij hebben ons daarom vooral gericht op het onderzoek naar methoden voor problemen met verschillende voertuigen. In deze problemen onderscheiden we enerzijds een toewijzingsprobleem, waarin wordt bepaald welke opdrachten door welke wagen worden uitgevoerd, en anderzijds routeringsproblemen, waarin, voor elke wagen afzonderlijk, een route wordt geconstrueerd door de eraan toegewezen opdrachten.

Het onderzoek naar het toewijzingsprobleem voor het rechtstreeks vervoer leidde tot de definitie van een algemene toewijzingsprobleem waarin taken moeten worden toegewezen aan middelen die deze taken kunnen uitvoeren. Dit algemene toewijzingsprobleem hebben we gedefinieerd als een setpartitieprobleem. Voor deze formulering genereert men eerst alle mogelijke verzamelingen van taken die aan één middel kunnen worden toegewezen. Vervolgens selecteert men hieruit voor elk middel één verzameling, zodanig dat elke taak in precies één verzameling aanwezig is en de totale kosten van de gekozen verzamelingen minimaal is. De verzameling toegelaten oplossingen voor een setpartitieprobleem wordt vaak gerepresenteerd met behulp van een matrix waarin elke kolom overeenkomt met een verzameling van taken die aan één middel kan worden toegewezen. Men noemt zo'n verzameling dan ook vaak een kolom.

Met behulp van het setpartitiemodel kan het algemene toewijzingsprobleem worden opgelost met een branch-and-price algoritme. Hierin staat de lineaire-programmeringsrelaxatie (LP-relaxatie) van het setpartitieprobleem centraal. Omdat het aantal variabelen in de setpartitieformulering erg groot is, wordt de LP-relaxatie opgelost met behulp van kolomgeneratie. Daarbij worden de mogelijke kolommen niet allemaal vooraf gegenereerd, maar worden de mogelijk interessante kolommen in de loop van het algoritme aan de formulerings toegevoegd. Deze kolommen worden gegenereerd door een pricing-probleem op te lossen. Om dit probleem exact op te lossen is inzicht nodig in de precieuze structuur van de taken en de middelen. We hebben daarom algemeen bruikbare pricing-heuristieken ontwikkeld, waarmee we echter niet kunnen garanderen dat de LP-relaxatie van het setpartitieprobleem optimaal wordt opgelost. Wanneer er geen kolommen te vinden zijn die moeten worden toegevoegd aan de formulering, zoeken we geheeltallige oplossingen met behulp van branch-and-bound. Een door ons ontwikkeld vertakkingschema maakt het mogelijk om in elke gegenereerde deelverzameling
van de oplossingsruimte kolomgeneratie te gebruiken. Op deze wijze ontstaat een benaderingsalgoritme voor een brede klasse van toewijzingsproblemen. Voor elk specifiek toewijzingsprobleem kan hiervan een optimaliseringsalgoritme worden gemaakt door het pricing-probleem optimaal op te lossen.

We hebben de hierboven beschreven branch-and-price algoritme getest op het rechtstreeks-vervoerprobleem met tijdvensters. Hierin geven de klanten aan tussen welke tijdstippen de verschillende locaties bezocht moeten worden. We hebben een verzameling testproblemen gegenereerd, en hebben die optimaal opgelost door een optimaliseringsalgoritme voor het pricing-probleem aan ons algoritme toe te voegen. Vervolgens hebben we de oplossingen voor deze problemen benaderd met behulp van het algemene benaderingsalgoritme. Door de oplossingen en rekentijden van de optimaliseringen en de benaderingen te vergelijken, kunnen we concluderen dat de benaderingsalgoritme in staat is om, in relatief weinig rekentijd, uitstekende oplossingen te vinden voor dit specifieke toewijzingsprobleem.

Het hoofddoel van het onderzoek was algoritmen te ontwikkelen voor een rechtstreeks-vervoerprobleem uit de praktijk. We hebben hier- toe samengewerkt met Van Gend & Loos BV, een Nederlands bedrijf dat onder andere rechtstreeks vervoer verzorgt in de Benelux. Het rechtstreeks-vervoerprobleem bij Van Gend & Loos BV is dynamisch. Wanneer de voertuigen hun routes beginnen uit te voeren, zijn nog niet alle opdrachten bekend, zodat de verzameling routes moet worden aangepast zodra er nieuwe opdrachten binnenkomen. Voor deze aanpassing hebben we een heroptimaliseringsprobleem gedefinieerd waarvan de oplossingen corresponderen met een nieuwe verzameling routes voor de wagens. Om goede oplossingen te vinden voor dit heroptimaliseringsprobleem, hebben we het algoritme DRIVE (Dynamic Routing of Independent VEHicles) ontwikkeld, dat gebaseerd is op de hierboven beschreven kolomgeneratie-methode. DRIVE is ontwikkeld om in een beslisssondersteunend systeem te worden ingebed, maar is alleen als stand-alone systeem getest, omdat de infrastructuur die nodig is om DRIVE aan de huidige informatiesystemen van Van Gend & Loos BV te koppelen nog niet aanwezig was. De testresultaten geven aan dat DRIVE, als stand-alone systeem, iets betere resultaten levert dan de planners bij Van Gend & Loos BV. We verwachten dat een interactief systeem, waarin DRIVE de planners assisteert, bij het rechtstreeks vervoer van Van Gend & Loos BV een kostenbesparing op zal leveren van ongeveer 5% en bovendien het werk van de planners zal vereenvoudigen.
Curriculum vitae

Marc Sol was born on October 21, 1966 in Terneuzen, the Netherlands. In 1985 he completed his secondary school education at the RSG Petrus Hondius in Terneuzen and moved to Eindhoven to start his studies for a mathematical engineer at Eindhoven University of Technology. During his studies, he specialized in discrete mathematics and finished his master's thesis on cyclic error correcting codes under supervision of Prof.dr. J.H. van Lint in 1990. After his graduation, he started as a Ph.D. student at Eindhoven University of Technology in the group on combinatorial optimization of Prof.dr. J.K. Lenstra. His project was concerned with a practical vehicle routing problem and was financed by the Stichting voor de Technische Wetenschappen (Technology Foundation). During the first two years of the project Marc attended the lectures of the Landelijk Netwerk Mathematische Beleidkunde, of which he obtained his diploma in 1993. At present Marc is working at ORTEC Consultants BV in Gouda.
Column generation techniques for pickup and delivery problems

van

Marc Sol
Zij $A \in \{0, 1\}^{m \times n}$ en $c \in \mathbb{R}^n_+$, zo dat

$$
\forall x \in \mathbb{R}^n_+ \quad \exists i \in \{1, 2, \ldots, m\} \quad s.t. \quad a_{i,j} x_j \geq c_j.
$$

Zij $P$ het probleem minimaliseren \{ex | Ax = 1, x \geq 0\}. Het polytoop van het duale probleem is $P^* = \{ u \in \mathbb{R}^m_+ | \sum_{j=1}^n a_{j,i} u_j \leq c_i \quad (1 \leq j \leq n) \}$. Veronderstelt dat $P$ wordt opgelost met behulp van een simplexalgorithm, en zij $d_j$ de gemiddelde kosten van variabele $x_j$ tijdens een iteratie van dit algoritme. Zij

$$
J = \text{argsort} \left\{ \frac{d_j}{\sum_{i=1}^m a_{i,j}} \right\} \quad (1 \leq j \leq n).
$$

Dan geldt

$$
\exists u \in \mathbb{R}^m_+ \quad \sum_{i=1}^m a_{i,J} u_i = c_{J} \quad \neq 0.
$$

Zie paragraaf 2.6.3 van dit proefschrift.

II

Zij $A \in \{0, 1\}^{m \times n}$ en $c \in \mathbb{R}^n_+$. Rushmer (1989) conclueert uit experimenten dat, voor het oplossen van min\{ex | Ax = 1, x \geq 0\} met behulp van een simplexalgorithm, de pivotregel waarbij de kolom $i$ is gegeven $\text{argsort} \left\{ \frac{d_j}{\sum_{i=1}^m a_{i,j}} \right\} \quad (1 \leq j \leq n)$ in de basis wordt geopend, tot veel minder simplexiteraties leidt dan de standaardpivotregel. Stelling 1 biedt een theorie voor de optimale basis van deze observatie.


III

Zij argument, $N = \{1, 2, \ldots, n\}$ en $c \in \mathbb{R}^n_+$. Beschouw het Handelsreiseprobleem op de puntenverzameling $N$ met afstandsmatrit $c$. Definieer voor elk punt $i \in N$

$$
x_i = \frac{1}{2} \left( c_{i,i} \right)^{\left( \sum_{j=1}^n (1)^{c_{j,i,j}} x_j + \sum_{j=1}^n (1)^{c_{j,i,j}} \text{mod} \right)}.
$$

Als $c_{i,i} = 1$ voor alle $i, j \in N$, dan is de tour $1, 2, 3, \ldots, n-1, n, 1$ optimaal.
IV

Zij \( T = (N, E) \) een ongerichte boom met puntenzameling \( N \) en kantenzameling \( E \). Zij \( n \in N \) en zij \( A \subseteq N \times N \) een preceedentrelatie op \( N \). Een rondgang \( r = i_1, i_2, \ldots, i_p, i_p = r \) door \( T \) heeft toegelaten als voor alle \( i < p \) en \( k \in N \) met \((h, i, j) \in A\) geldt dat \( h \in \{i_1, i_2, \ldots, i_{p-1}\} \). Definieer \( d_i \), als het aantal kanten in het pad van \( r \) naar \( i \) in \( T \), en \( A' = \{(i, j) \mid (i, j) \in E, d_i < d_j\} \). Er bestaat een toegelaten rondgang \( r = i_1, i_2, \ldots, i_p, i_p = r \) met \( \{i_1, i_2, \ldots, i_p\} = N \), dan en slechts dan als de gerichte graaf \((N, A \cup A')\) geen circuits bevat.

V

Gegeven is een onbeperkt aantal identieke parallele machines. Een verzameling taken met een gemeenschappelijke deadline \( T \) moet op deze machines worden uitgevoerd. Deze taken komen in real time beschikbaar. Dat wil zeggen dat, voor elke taak \( j \), zowel het beschikbaarheidsinterval \( r_j \) als de verwerkingstijd \( p_j \) al bekend wordt op tijd \( r_j \). Ook het totale aantal taken \( n \) is onbekend. De deadline \( T \) is vooraf bekend. Zij \( m \) het minimale aantal machines dat nodig is om alle taken uit te voeren indien \( n, r_j (1 \leq j \leq n) \) en \( p_j (1 \leq j \leq n) \) vooraf bekend worden zijn. Een online scheduling strategie is een mechanisme dat, in real time, taken toewijzet aan machines en zorgd voor nieuwe machines in gebruik nemen. Hierbij houdt een taak \( j \) niet direct op tijd \( r_j \) aan een machine te worden toegewezen. Als \( \sum_{j=1}^{n} p_j \) vooraf bekend is, \( r_j \in NT \) voor elke taak \( j \), en voor zekere \( k \in \mathbb{N} \) geldt dat \( p_j \in \{1, k\} \) voor elke taak \( j \), dan bestaat er een online scheduling strategie die \( m \) machines beheerst.

VI

Een algemene benaderingsmethode voor dynamische optimaliseringsproblemen is het sequentieel oplossen van een serie statische problemen. Het is echter genoeg om, gegeven de doelstellingfunctie van het dynamische probleem, een goede doelstellingfunctie voor de statische problemen te definiëren.

VII

Hoewel binnen de beslissingsproblemen worden bestudeerd die afkomstig zijn uit de logistiek praktijk, staan de toepassing van genereerde oplossingsmethoden in die praktijk nog in de kinderschoenen. Hiernaast kan genoemd worden wanneer er dergelijke methoden beschikbaar komen die toepasbaar zijn op een breedere klasse van problemen en die in korte tijd in een beslissingsondersteunend systeem kunnen worden ingebouwd.
VIII

Een weldegende oplossing methodé voor het juiste probleem is beter dan de beste oplossingsmethodé voor het verkeerde probleem.

IX

Het is opmerkelijk dat de indicatie 'gesorteerd' op een zak drop vaak aanduidt dat deze een aantal dropsoorten doet elkaars bevatten.

X

Het geeft te denken dat partijen, die onder andere zijn aangesteld om werknemers te beschermen tegen indringers, gedurende het grootste gedeelte van de tijd de werknemers zelf als indringers behandelen.

XI

Wanneer bij de verhuur van auto's dezelfde normen gehanteerd worden als bij de verhuur van huizen, zou men, zover een huurauto te kunnen gebruiken, deze eerst zelf een grote onderhoudsbicent moeten geven.