universität wien

DIPLOMARBEIT

Titel der Diplomarbeit

„Mixed Integer Programming - Techniques for the Dial-a-Ride Problem“

Verfasserin

Elisabeth Pfliegl

angestrebter akademischer Grad

Magistra der Sozial- und Wirtschaftswissenschaften (Mag. rer. soc. oec.)

Wien, im November 2007

Studienkennzahl lt. Studienblatt	A 157
Studienrichtung lt. Studienblatt	Internationale Betriebswirtschaft
Betreuer	O.Univ.Prof. Dipl.-Ing. Dr. Richard F. Hartl

I would like to thank Prof. Richard F. Hartl and all members of the Chair of Production & Operartions Management for their mentoring, especially Sophie N. Parragh, Günther Füllerer, and Karl F. Dörner for guiding me as I worked on this thesis.

Thanks also to my family, especially to my father, for their support while I have been a student at the BWZ.

I dedicate this work to my fiancé, Gerhard.

Contents

List of Tables

List of Figures

Listings

Abstract

With a growing percentage of elderly or disabled people in our society, the number of people not being able to drive a car themselves, or even to go by bus, increases, too. Specific services are required to address mobility demands, and dial-a-ride systems have been developed to provide an appropriate answer. Customers are picked up from a service provider at specified locations (e.g. from their homes) and are carried to a destination (a doctor, for example). If required, also the return trip is carried out. The dial-a-ride problem (DARP) is first treated in the literature in the late 1960s. In this variant of the vehicle routing problem with pickup and delivery, a customer specifies a pick-up and a drop-off location, as well as desired times. Routes with minimum costs are constructed, starting and ending at a depot, under consideration of a number of constraints, so that overall transportation costs are minimized. What makes the DARP so special compared to other vehicle routing problems (VRP) is the circumstance that people, instead of goods, are transported, and therefore user convenience has to be considered [11].

Objective of this work is a survey on parameter settings in CPLEX[1]. Based on the implementation of a DARP model by Cordeau [2], five CPLEX parameters are tested to analyze if default settings can be improved. The model by Cordeau was chosen because of two advantages: firstly, it was published recently at that time, and secondly, no cuts need to be generated, therefore it is particularly suitable for impementation and parameter testing. In total, twelve test instances[2] with different size are used for testing. Every instance is first solved with default settings, and afterwards, every parameter value is tested on every instance, where all other parameters are left as default. Further, lower bounds for the objective function are added to support the solution process. These lower bounds were generated by Cordau [2] by running a tabu search heuristic for 1.000 iterations. Finally, the problem generated by CPLEX is solved by XPRESS[3]. Results showed that default

[1] CPLEX is an optimization tool, developed by ILOG

[2] Test instances are developed by Cordeau, `http://www.hec.ca/chairedistributique/data/darp`

[3] XPRESS is another optimization tool, developed by Dash Optimization

settings provide in general good results, but changing some of the tested parameters led to even better solutions.

1. Introduction

The dial-a-ride problem (DARP) belongs to the group of vehicle routing problems with pickups and deliveries. The most important aspect is that, instead of goods, people are transported, and therefore user (in)convenience has to be considered when designing routes. Inconvenience can be expressed in terms of maximum ride time of a user during a trip, waiting time, or the difference between actual and desired arrival time. Tighter constraints on ride times and pick-up or drop-off times improve customer convenience, but result usually in higher operating cost [11].
Like in most vehicle routing problems, a static case, where all requests are known in advance, and a dynamic mode, where new requests are placed during the day and new solutions are computed in real-rime, are distinguished. In reality, neither pure dynamic nor pure static versions exists, usually. Instead, a mixture of both can be observed, where some requests are known in advance, and some arrive later, or some already placed requests are cancelled. However, for route construction, it is reasonable to assume that all requests are known in advance, which facilitates the problem solution [5].

After an overview of relevant literature in Chapter 2, where variants of the Vehicle Routing Problem are discussed, the DARP is presented in more detail. The model of Cordeau is described in Section 3, constraints are presented, as well as a number of new inequalities developed by Cordau, and a heuristic to further decrease the problem size is discussed [2]. Implementation of the model as well as parts of the code are described in Section 4. Test instances, parameters and testing are subsequently presented. Results are depicted and discussed in Section 5, and conclusions follow in Section 6.

2. Vehicle Routing Problems

In growing globalization nearly everything has to be transported, either between production steps, between manufacturer and retailer, or to final users. But Vehicle Routing Problems (VRP) provide solutions for much more applications, like street cleaning or waste collection (arc routing problems), route-planning for salespeople (TSP), or transportation of elderly or handicapped persons (dial-a-ride systems) [17]. This first part gives an overview of VRP, a description of its main components, main objectives, and operational and possible additional constraints. Different VRP are presented, similarities as well as their differences, and finally the DARP will be discussed in more detail.

The term VRP comprises problems concerning the delivery or collection of goods between depots and customers, performed by a vehicle driving along a planned route. Routes are performed on a road network, which is described in a graph where streets are represented as arcs, and depots, customers, and road junctions are represented as vertices. This graph is then transformed into a complete graph, where for every customer pair i and j an arc representing the shortest path between these two customers, is included. Arcs can be directed or undirected, where directed arcs are used to show one-way streets. With all arcs travel times are associated, depending on the length of the arc and the vehicle traversing it. With every arc also travel costs are associated [17].
With each vehicle, a home depot is associated, where it starts and ends a route. The vehicle's capacity as well as additional information, e.g. information about loading or unloading devices, subdivisions in the storage space of the vehicle, special seats or equipment for the transportation of handicapped persons, etc. have to be considered. Sometimes, e.g. for heavy loads or especially big vehicles, it is useful to produce a graph consisting of a subset of arcs that can be traversed by the vehicle if it cannot use all arcs in the original graph (e.g. the truck is too high for an underbridge, or it is too wide for certain streets, the corresponding arcs are removed from the graph because they can't be used for transportation). Finally, for every vehicle operating costs are determined, either per time

unit or by distance travelled. These costs also include driver's wages, and all constraints concerning the driver are imposed on his/her vehicle. Such constraints include working periods and maximum ride times per day, breaks, etc.
With every customer a demand (delivery or collection of goods, pick-up or drop-off) is associated, and the service location is included as a vertex in the graph. In addition, service times (time windows), service duration for loading or unloading activities, and the subset of vehicles able to serve the customer (depending on special requirements like access limitations, special seatments, or loading/unloading requirements) are recorded [17].

Due to time or capacity reasons, it is sometimes not possible to satisfy all requests. To overcome such situations, it is possible to reduce the amounts delivered (or picked up), or to leave some customers unserved. In both cases, penalties can be associated with the lack of service, and priorities help to emphasize customers that are more important to serve. (For example, A-customers have to be served by all means, whereas requests of B- and C-customers can be fulfilled the next day). These penalties are included in the objective function, which can reflect different goals, for example minimization of global transportation cost (which is the usual objective), but also minimization of the number of vehicles, minimization of penalties, or a combination of those objectives. It has to be noted that minimization of penalties and therefore minimization of customers being unserved, will usually increase transportation costs, and vice versa [3, 8].

2.1. Variants of the VRP

The *capacitated VRP* (CVRP) consists of finding a collection of routes, starting and ending at the depot, so that every customer is visited exactly once, and the capacity of the vehicles performing the routes is not exceeded. The CVRP generalizes the Traveling Salesman Problem, which calls for the determination of a Hamiltonian cycle, and can therefore be seen as a CVRP with only one available vehicle [17].

The *distance constrained VRP* (DCVRP) is a variant of the VRP where the total distance travelled by a vehicle in a route is constrained by a maximum length T, or by a maximum travel time. It is possible to associate not only travel times with arcs, but also service times with vertices to include loading and unloading times. Service times can be included in travel times or can be associated with vertices being served. Objective of this variant

is the minimization of the total route length or the total travel duration, respectively [17].

An extension to the VRP is the *VRP with time windows* (VRPTW), where a time interval is associated with every customer, and servicing the customer is only possible within this time window. Again, capacity constraints are imposed and service durations are associated with every customer. Vehicles are allowed to wait if they arrive before the earliest time to start service (which is the beginning of the time window). Every customer has to be visited exactly once, every route starts and ends at the depot, vehicle capacity must not be exceeded, and service for every customer has to start within the time window. The objective is to minimize costs for a collection of routes satisfying these constraints [17].

A second extension of the VRP arises by *including backhauls* (VRPB). Customers are divided into linehaul customers, receiving products, and backhaul customers, where something has to be picked up. All linehaul customers are visited before the first backhaul customer, and a collection of circuits with minimum cost has to be found [17].

In reality, combinations of all problems described above are possible.

The *VRP with Pickup and Delivery* (VRPPD) calls for the determination of a set of routes with minimum cost, such that each circuit visits the depot and every customer is visited exactly once. Since in this problem a good is picked up from a customer and is delivered to another customer, both customers have to be served in the same circuit and therefore by the same vehicle, and the pickup customer has to be served before the delivery customer. Vehicle load is nonnegative and may never exceed vehicle capacity [17].

In the *VRP with simultaneous pickup and delivery* (VRPSPD), origins or destinations of demands are common (e.g. all quantities picked up are delivered to a single customer or the depot, as the collection of empty bottles, for example), and are not explicitly indicated [17].

Finally, the *Dial-a-Ride problem* (DARP) is a generalization of the VRP with Pickup and Delivery, where customers have to be picked up at a specified location and are dropped-off at another location, under consideration of time windows and ride time constraints [2,11]. The DARP will be discussed in detail in the next section.

In static as well as dynamic solution approaches, depots are assumed as start- and end-points in a route. Whereas this makes sense for the static case, in the dynamic version this is, if at all, only the case for the first schedule of the day. Later on, when routes are updated due to new requests coming in, vehicles are already somewhere on the route and "...the concept of a depot vanishes." ([14], p.19) Consequently, for the updated route, depots are not considered, and this has to be included in the model.

2.2. The dial-a-ride problem

The DARP resembles the VRP with pickup and delivery, but, due to the transportation of people instead of goods, some extensions are necessary. The most important difference to other VRP is the concept of customer inconvenience, and the introduction of service criteria to measure the level of inconvenience and to possibly decrease it. Since service quality is often opposed to cost minimization (usually, better service in terms of shorter waiting or ride times results in higher cost), both objectives have to be balanced against each other [8]. Although nobody likes to wait, customers would probably prefer to be served with deviation instead of not being served at all. A special difficulty in the DARP surely exists therein to serve as many customers as possible, but to choose time windows for pick-ups/drop-offs and maximum ride times tight enough to avoid unnecessarily long waiting times [1]. Pick-up and drop-off times are specified by users, and time-windows are constructed around them. Some approaches allow time windows only for the drop-off time at outbound requests and on pick-up times for inbound requests, respectively. Time-windows can be specified by users, or the user specifies only a time for pick-up or drop-off, and a window around this time is constructed by the operator. The compliance of maximum ride times is ensured by setting an upper bound to travel times of a passenger, which is either expressed as an absolute value (e.g. 30 minutes) or in a mathematical function based on the actual distance between pickup and delivery point [2].

Transportation is fulfilled with a fleet of vehicles. To keep costs low, it is possible to satisfy demand with a core fleet, and use extra vehicles like taxis if necessary. This will also be favoured if the service is run by public authorities, whereas in private companies, requests that cannot be satisfied, will be turned down frequently. In both cases, the different types of vehicles being available have to be considered. Vehicles can be equipped for ambulatory

passengers, or can be adequate to transport only wheelchairs, or both (e.g. with seats that can be removed). This heterogeneity of vehicles can be introduced to the model by adding an additional index for vehicles as done by Cordeau [2, 8].

Quality of service criteria
In passenger transportation so called service-criteria have to be considered. These include customer ride time, and delay, which is the difference between actual and desired delivery times [5]. These criteria can be integrated in the model either by including penalties in the objective function, or as side constraints. In the first case, violations are possible, and this is often called a "soft" constraint. In the latter case, where ride times are included as side constraints, ride times have to be maintained and no violation is possible (what is therefore called a "hard" constraint) [17].

In summary, the DARP consists of designing a number of m routes for as many vehicles, such that

- every route starts and ends at the depot,
- every customer is served by a single vehicle,
- the drop-off location of a customer is visited after the pick-up location (precedence),
- service starts within the time interval specified by user or operator (time window),
- the ride time of a user does not exceed the maximum ride time,
- the vehicle-capacity is never exceeded, and
- the overall routing cost is minimized;

Besides the minimization of routing cost, several alternative objectives can be found in the literature. These are

- minimization of total travel time
- minimization of the number of vehicles
- minimization of customer inconvenience or

- maximization of service quality (with respect to waiting time and deviation from promised service times)
- maximization of user respond rate (accepting as many requests as possible), given that the number of vehicles is fixed

According to Cordeau and Laporte [3], three decisions have to be made when constructing a solution for the DARP, which are

1. *Determination of clusters*
   Customers with similar time windows and locations in the same vicinity are grouped together. When every customer is assigned to a cluster, every cluster is in turn assigned to a vehicle.
2. *Sequencing within clusters*
   To form a route, customers within a cluster are sequenced.
3. *Scheduling*
   For every customer in a route, pick-up, drop-off, and ride times are fixed. These schedules focus on minimization of route duration under consideration of time windows and maximum ride times.

These decisions can be performed successively, or simultaneously.

To reduce route duration, Savelsbergh [13] first proposed in 1992 the computation of a forward time slack, representing the maximum possible delay at each vertex. This is also done by Cordeau [2] in the solution approach presented in Section 3. The forward time slack is calculated as

$$F_i = min_{i \leq j \leq q}\{ \sum_{i<p\leq j} W_p + min\{l_j - B_j, L - P_j\}\} \tag{2.1}$$

where W_i is the waiting time at node i, l_i is the end of the time window at node i, and is therefore the latest possible time to begin service, and B_i is the actual beginning-time of service at node i. L states the maximum ride time allowed for a customer. P_i denotes the ride time of a user if node i is the drop-off node of this user ($i \in D$), and $P_i = -\infty$ otherwise. The forward time slack is therefore the minimum out of the sum of waiting

times plus the biggest possible delay. The delay is bounded by the size of the time window on the one hand, and by maximum ride time on the other hand.

2.2.1. Solution approaches

The DARP can be subdivided into a static and a dynamic problem, in single- or multi-vehicle models, and in exact, heuristic, and metaheuristic solution approaches. Of course, multi-vehicle approaches can always be applied to solve single vehicle problems, but latter ones are often used in cluster first, route second approaches, and clusters are solved with single-vehicle algorithms that usually succeed in terms of solution time [5, 11].

An early exact solution approach was presented by Psaraftis in 1980 for the static DARP, followed by an updated version for the dynamic variant in 1983. For the single vehicle, many-to-many DARP Psaraftis first clustered customers in the same neighborhood, and then a cluster was assigned to a vehicle, implying that as many clusters were formed as vehicles were available. The degree of dissatisfaction of customers was measured in terms of waiting and ride time. Instances with up to 9 customers have been solved to optimality. The relatively small number of requests is due to the exponential computing time of the exact algorithm. More recently, a three-index formulation was presented by Cordeau ([2]), this approach is described in Section 3. Another two-index formulation was published this year by Ropke et al. [5, 12].

Sexton and Bodin [15, 16] tackled the DARP by splitting it into two parts: the scheduling and the routing process. Customer inconvenience is expressed as excess ride time and deviation from desired delivery time, and has to be minimized. The addressed problem is again the single vehicle, many-to-many DARP.

Ioachim et al. [10] proposed a request clustering algorithm where mini-clusters are formed. A mini-cluster is a set of "... geographically and temporally cohesive transportation requests that can be feasibly served by the same vehicle" ([10], p.64). The authors also allude to the importance of the clustering phase, and the difficult decision of which constraints should be incorporated in this part of the solution process, as bad clustering decisions cannot be straightened out during the routing phase. The approach was applied

to instances with up to 2545 customers, resulting in reductions of both, total travel time, and vehicles used.

Gendreau et al. [7] proposed to precompute a number of different scenarios that can be used to speed up the solution process. This idea is seized up by Ho and Haugland [8], who proposed a procedure in 2004 consisting of a tabu search heuristic and a hybrid GRASP-tabu search heuristic to solve the probabilistic DARP. In this version of the DARP, requests are known but are required only with a certain propability, e.g. customers that always need to be transported from home to the hospital and back, but do not need the service every day. Routes are not scheduled new every day, but customers not being served on a day are simply removed from the route, and expected travel cost, instead of overall travel cost, is minimized.

As the solution of real life instances often fails due to computing time, Hunsaker and Savelsbergh [9] proposed an "efficient feasibility testing for the DARP", where in a first pass through a route load-, time window-, and waiting time constraints are assessed, a second pass in backward direction checks for violation of ride time constraints, and a third pass in forward direction through the route confirms attempts to adjust ride times during the second pass. This route check can be applied previously to other solution methods, or can be used to assess user requests being place dynamically.

3. The model

The model by Cordeau [2], that was implemented and used for parameter testing, is presented in the following, as well as parts of the code-file. A description of a heuristic to identify infeasible paths is given, and a number of inequalities used by Cordeau [2], that are added to the model prior to solving it, are presented. Due to readability reasons, variables in CPLEX have different names as in the model by Cordau. A list of variable names can be found in tables B.1 and B.2 in the appendix.

3.1. Notation

The problem is formulated on a directed, complete graph, where

P ... set of pickup vertices
D ... set of delivery vertices
N ... set of all nodes including both depots, indexed 0 and $2n + 1$
n ... number of pickup vertices, indexed $i = 1, ..., n$
and deliveries, indexed $i = n + 1, ..., 2n$
q_i ... demand/supply at vertex i; pickup vertices are associated with a positive value, delivery vertices with a negative value
$q_0 = q_{2n+1} = 0$
e_i ... earliest time to begin service at vertex i
l_i ... latest time to begin service at vertex i
d_i ... service duration at vertex i
c_{ij}^k ... cost to traverse arc (i, j) with vehicle k
t_{ij}^k ... travel time from vertex i to vertex j with vehicle k
K ... set of all vehicles
m ... number of vehicles, indexed $k = 1, ..., m$
Q ... capacity of vehicle

3. The model

T ... maximum route duration
L ... maximum ride time of a user
L_i ... ride time of user i

$$x_{ij}^k = \begin{cases} 1, & \text{if arc } (i,j) \text{ is traversed by vehicle } k \\ 0, & \text{otherwise} \end{cases}$$

B_i^k ... beginning of service of vehicle k at vertex i
Q_i^k ... load of vehicle k after visiting node i

$$\min \sum_{k\in K} \sum_{i\in N} \sum_{j\in N} c_{ij}^k x_{ij}^k \tag{3.1}$$

subject to:

$$\sum_{k\in K} \sum_{j\in N} x_{ij}^k = 1 \qquad \forall i \in P \tag{3.2}$$

$$\sum_{j\in N} x_{ij}^k - \sum_{j\in N} x_{n+i,j}^k = 0 \qquad \forall i \in P, k \in K \tag{3.3}$$

$$\sum_{j\in N} x_{0j}^k = 1 \qquad \forall k \in K \tag{3.4}$$

$$\sum_{j\in N} x_{ji}^k - \sum_{j\in N} x_{ij}^k = 0 \qquad \forall i \in P \cup D, k \in K \tag{3.5}$$

$$\sum_{i\in N} x_{i,2n+1}^k = 1 \qquad \forall k \in K \tag{3.6}$$

$$B_j^k \geq (B_i^k + d_i + t_{ij}) x_{ij}^k \qquad \forall i \in N, j \in N, k \in K \tag{3.7}$$

$$Q_j^k \geq (Q_i^k + q_j) x_{ij}^k \qquad \forall i \in N, j \in N, k \in K \tag{3.8}$$

$$L_i^k = B_{n+i}^k - (B_i^k + d_i) \qquad \forall i \in P, k \in K \tag{3.9}$$

$$B_{2n+1}^k - B_0^k \leq T^k \qquad \forall k \in K \tag{3.10}$$

$$e_i \leq B_i^k \leq l_i \qquad \forall i \in N, k \in K \tag{3.11}$$

$$t_{i,n+i} \leq L_i^k \leq L \qquad \forall i \in P, k \in K \tag{3.12}$$

$$\max\{0, q_i\} \leq Q_i^k \leq \min\{Q_k, Q_k + q_i\} \qquad \forall i \in N, k \in K \tag{3.13}$$

$$x_{ij}^k \in \{0,1\} \qquad \forall i \in N, j \in N, k \in K \tag{3.14}$$

Objective is the minimization of travel cost (3.1). Constraints (3.2) ensure that every customer is visited exactly once and constraints (3.3) ensure that pick-up and drop-off location of a customer are visited by the same vehicle. Every route starts and ends at the depot (constraints (3.4) and (3.6), respectively). Route connectivity is addressed by constraints (3.5). Constraints (3.7) and (3.8) ensure consistency of time and load variables. Ride times of users are defined by constraints (3.9) and are bounded by (3.12). Latter ones also act as precedence constraints. Maximum route length is incorporated by (3.10), adherence to time windows and capacity restrictions is ensured by constraints (3.11) and (3.13), respectively.

The number of variables and constraints is reduced by replacing time and load variables with aggregate variables. Those aggregate variables can be applied at every node except the depots, since every node is visited exactly once. It has to be noted that load variables can only be aggregated if all vehicles are homogeneous ([2], p. 575).

$$B_j \geq (B_0^k + d_0 + t_{0j}) x_{ij}^k \qquad \forall\ j \in N, k \in K \tag{3.15}$$

$$B_j \geq (B_i + d_i + t_{ij}) \sum_{k \in K} x_{ij}^k \qquad \forall\ i \in N, j \in N \tag{3.16}$$

$$B_{2n+1}^k \geq (B_i + d_i + t_{i,2n+1}) x_{i,2n+1}^k \qquad \forall\ i \in N, k \in K \tag{3.17}$$

$$L_i = B_{n+i} - (B_i + d_i) \qquad \forall\ i \in P \tag{3.18}$$

$$Q_j \geq (Q_0^k + q_j) x_{0j}^k \qquad \forall\ j \in N, k \in K \tag{3.19}$$

$$Q_j \geq (Q_i + q_j) \sum_{k \in K} x_{ij}^k \qquad \forall\ i \in N, j \in N \tag{3.20}$$

$$Q_{2n+1}^k \geq (Q_i + q_{2n+1}) x_{i,2n+1}^k \qquad \forall\ i \in N, k \in K \tag{3.21}$$

Constraints (3.20) are further lifted. This is done with constraints

$$Q_j \geq Q_i + q_j - W_{ij}(1 - \sum_{k \in K} x_{ij}^k) + (W_{ij} - q_i - q_j) \sum_{k \in K} x_{ji}^k \forall\ i \in N, j \in N \tag{3.22}$$

taking into account that of an arc (i,j) and its reverse arc (j,i), only one will be traversed at most. W_{ij}^k is a constant introduced to linearize the former quadratic constraint. The

same method is applied to all other quadratic constraints:

$$B_j \geq (B_0^k + d_0 + t_{0j}) - M_{0j}^k(1 - x_{ij}^k) \qquad \forall\, j \in N, k \in K \qquad (3.23)$$

$$B_j \geq (B_i + d_i + t_{ij}) - M_{ij}(1 - \sum_{k \in K} x_{ij}^k) \qquad \forall\, i \in N, j \in N \qquad (3.24)$$

$$B_{2n+1}^k \geq (B_i + d_i + t_{i,2n+1}) - M_{i,2n+1}^k(1 - x_{i,2n+1}^k) \qquad \forall\, i \in N, k \in K \qquad (3.25)$$

$$Q_j \geq (Q_0^k + q_j) - W_{0j}^k(1 - x_{0j}^k) \qquad \forall\, j \in N, k \in K \qquad (3.26)$$

$$Q_{2n+1}^k \geq (Q_i + q_{2n+1}) - W_{i,2n+1}^k(1 - x_{i,2n+1}^k) \qquad \forall\, i \in N, k \in K \qquad (3.27)$$

where $M_{ij}^k \geq \max\{0, l_i + d_i + t_{ij} - e_i\}$ and $W_{ij}^k \geq \min\{Q_k, Q_k + q_i\}$. The calculation of M_{ij}^k and W_{ij}^k is shown in listing 3.1.

Listing 3.1: Calculation of M_{ij}^k and W_{ij}^k

```
//Value for M
for(var i in nodes){
 for(var j in nodes){
  for(var k in vehicles){
   M[i][j][k] = Math.max(0, (latestStartingTime[i]+serviceDuration[i]+
    distance[i][j]- earliestStartingTime[j]));
}}}

//Value for W
for(i in nodes){
 for(j in nodes){
  for(k in vehicles){
   W[i][j][k] = Math.min(maxcapacity,(maxcapacity+load[i]));
}}}
```

The final model then consists of constraints (3.1) - (3.6), (3.10) - (3.14), (3.22) and (3.23) - (3.27), as shown in Appendix A. Implementation of the model in CPLEX is shown in Code sample 3.2 below.

Listing 3.2: Model

```
//objective function
minimize
 sum(k in vehicles, i in nodes, j in nodes)
```

```
  distance[i,j]*travel[i,j,k];

subject to {
//every customer is visited exactly once
forall(i in pickUp)
 ct02:
 sum(k in vehicles, j in 1..2*nbcustomers+1) travel[i,j,k]==1;

//pick-up and drop-off nodes of customer i are visited by the same vehicle
forall(i in pickUp, k in vehicles)
 ct03:
 sum(j in 1..2*nbcustomers+1) travel[i,j,k] -
  sum(j in 1..2*nbcustomers+1) travel[nbcustomers+i,j,k]==0;

//every vehicle leaves the depot once
forall(k in vehicles)
 ct04:
 sum(j in nodes) travel[0,j,k] == 1;

//every node is visited and left by the same vehicle (Connectivity)
forall(i in pickUpDropOff, k in vehicles)
 ct05:
 sum(j in 0..2*nbcustomers) travel[j,i,k] -
  sum(j in 1..2*nbcustomers+1) travel[i,j,k] == 0;

//every route ends at depot
forall(k in vehicles)
 ct06:
 sum(i in nodes) travel[i,2*nbcustomers+1,k] == 1;

//consistency of time and load variables
//constraints (3.7) replaced by (3.23) - (3.25)
forall(i in pickUpDropOff, j in pickUpDropOff, k in vehicles)
 ct17:
 startService[j] >= B_0[k] + serviceDuration[0] + distance[0,j] -
  M[i,j,k]*(1 - travel[0,j,k]);

forall(i in pickUpDropOff, j in pickUpDropOff)
 ct18:
 startService[j] >= (startService[i] + serviceDuration[i] +
  distance[i,j]) -
```

```
  M[i,j,1]*(1 - sum(k in vehicles) travel[i,j,k]);

forall(i in pickUpDropOff, k in vehicles)
 ct19:
 B_2n1[k] >= startService[i] + serviceDuration[i] +
  distance[i,2*nbcustomers+1] -
  M[i,2*nbcustomers+1,k]*(1 - travel[i,2*nbcustomers+1,k]);

//constraints (3.8) replaced by (3.22), (3.26), (3.27)
forall(i in pickUpDropOff,j in pickUpDropOff, k in vehicles)
 ct21:
 aggLoad[j] >= (Q_0[k] + load[j]) - W[i,j,k]*(1 - travel[0,j,k]);

forall(i in pickUpDropOff, j in pickUpDropOff)
 ct24:
 aggLoad[j] >= aggLoad[i] + load[j] -
  W[i,j,1]*(1 - sum(k in vehicles) travel[i,j,k]) +
  (W[i,j,1] - load[i] - load[j])*sum(k in vehicles) travel[j,i,k];

forall(i in pickUpDropOff, j in pickUpDropOff, k in vehicles)
 ct23:
 Q_2n1[k] >= (aggLoad[i] + load[2*nbcustomers+1]) -
  W[i,j,k]*1- travel[i,2*nbcustomers+1,k]);

//constraints (3.9) replaced by (3.18)
forall(i in pickUp)
 ct20:
 rideTime[i] == (startService[nbcustomers+i] - (startService[i] +
  serviceDuration[i]));

//upper bound for route length
forall(k in vehicles)
 ct10:
 B_2n1[k] - B_0[k] <= maxroutelength;

//time windows
forall(i in pickUpDropOff, k in vehicles)
 ct11a:
 earliestStartingTime[i] <= startService[i];
forall(i in pickUpDropOff, k in vehicles)
 ct11b:
```

```
 startService[i] <= latestStartingTime[i];
forall(k in vehicles)
 ct11c:
 earliestStartingTime[2*nbcustomers+1] <= B_2n1[k];
forall(k in vehicles)
 ct11d:
 B_2n1[k] <= latestStartingTime[2*nbcustomers+1];

//bounds for customer ride times
forall(i in pickUp, k in vehicles)
 ct12a:
 distance[i,nbcustomers+i] <= rideTime[i];
forall(i in pickUp, k in vehicles)
 ct12b:
 rideTime[i] <= maxridetime;

//capacity bounds
forall(i in pickUpDropOff)
 ct13a:
 maxl(0,load[i]) <= aggLoad[i];
forall(i in pickUpDropOff)
 ct13b:
 aggLoad[i] <= minl(maxcapacity, (maxcapacity+load[i]));
```

3.2. Preprocessing

Before the model is solved, a number of preprocessing steps are performed to eliminate infeasible arcs and help therefore to reduce the problem size. The steps proposed by Cordeau ([2] Section 5.1, p.580) are presented in the following, as well as the implementation of these steps in CPLEX. As values of arcs are changed during this process, the distance matrix was copied to a temporary matrix where new values are stored, and after the preprocessing the distance matrix is overwritten by values of the temporary one. This is necessary to avoid that already changed values falsify calculations.

3.2.1. Time-window tightening

Time-windows are tightened to reduce the time frame where beginning of service is possible, and therefore some infeasible solutions can be eliminated. For outbound users, that are picked up from home and are carried to a destination, the time window $[e_i, l_i]$ at the origin node has the size $[0, T]$, where T is the maximum route length in minutes. This time frame can be tightened by setting $e_i = max\{0, e_{n+i} - L - d_i\}$ and $l_i = min\{l_{n+i} - t_{i,n+i} - d_i, T\}$. In the case of an inbound user, who is picked up at a speciefied location and is brought back home, the time window at the destination is set to $e_{n+i} = max\{0, e_i + d_i + t_{i,n+i}\}$ and $l_i = min\{l_i + d_i + L, T\}$. Construction of time windows for drop-off nodes of outbound requests and pick-up nodes of inbound requests are explained on page 36.
Time-windows for depots are tightened by setting $e_0 = e_{2n+1} = min_{i \in P \cup D}\{e_i - t_{0i}\}$ and $l_0 = l_{2n+1} = max_{i \in P \cup D}\{l_i + d_i + t_{i,2n+1}\}$.

Listing 3.3: Time window tightening

```
//Time-Window tightening
for(var i in pickUp){
 if((latestStartingTime[i]-earliestStartingTime[i]) >
 (latestStartingTime[nbcustomers+i] - earliestStartingTime[nbcustomers+i])){
  earliestStartingTime[i]=Math.max(0,(earliestStartingTime[nbcustomers+i]-
   maxridetime-serviceDuration[i]));
  latestStartingTime[i]=Math.min((latestStartingTime[nbcustomers+i]-
   distance[i][nbcustomers+i]-serviceDuration[i]), maxroutelength);
 }
 else{
  earliestStartingTime[nbcustomers+i]=Math.max(0, earliestStartingTime[i]+
   serviceDuration[i]+distance[i][nbcustomers+i]);
  latestStartingTime[nbcustomers+i]=Math.min(latestStartingTime[i]+
   serviceDuration[i]+maxridetime, maxroutelength);
 }
}
//Time-Window tightening for depots
smallest=1000;
largest=0;
for(i in pickUpDropOff){
 if(earliestStartingTime[i] - distance[0][i] < smallest)
  smallest = earliestStartingTime[i] - distance[0][i];
}
earliestStartingTime[0]=smallest;
```

```
earliestStartingTime[2*nbcustomers+1]=smallest;

for(i in pickUpDropOff){
 if(latestStartingTime[i]+serviceDuration[i]+
  distance[i][2*nbcustomers+1] > largest)
  largest=latestStartingTime[i]+serviceDuration[i]+
   distance[i][2*nbcustomers+1];
}
latestStartingTime[0]=largest;
latestStartingTime[2*nbcustomers+1]=largest;
```

3.2.2. Arc elimination

Due to ride-time, precedence, and time-window constraints, it is possible to remove several arcs from the complete graph G since they cannot belong to a feasible solution. These are

- arcs from the start-depot to drop-off nodes,
- arcs from pick-up nodes to the final depot,
- arcs from the drop-off location of a customer to the corresponding pick-up node, as a customer always has to be picked up *before* he can be dropped off;

Further,

- arc (i,j) with $i, j \in N$ is infeasible if $e_i + d_i + t_{ij} > l_j$
- arcs (i,j) and (j, n+i) with $i \in P$ and $j \in N$ are both removed if $t_{ij} + d_j + t_{j,n+i} > L$

3.2.3. Infeasible path heuristic

Even more arcs can be removed by checking paths with respect to time-windows and pairing constraints. A heuristic is set up to check the feasibility of paths between two customers. This heuristic is called for every user pair (i, j). If the heuristic is not able to find a feasible path for these users, corresponding arcs are removed from the graph. Heuristic, paths and implementation in CPLEX are described in the following [2, 6].

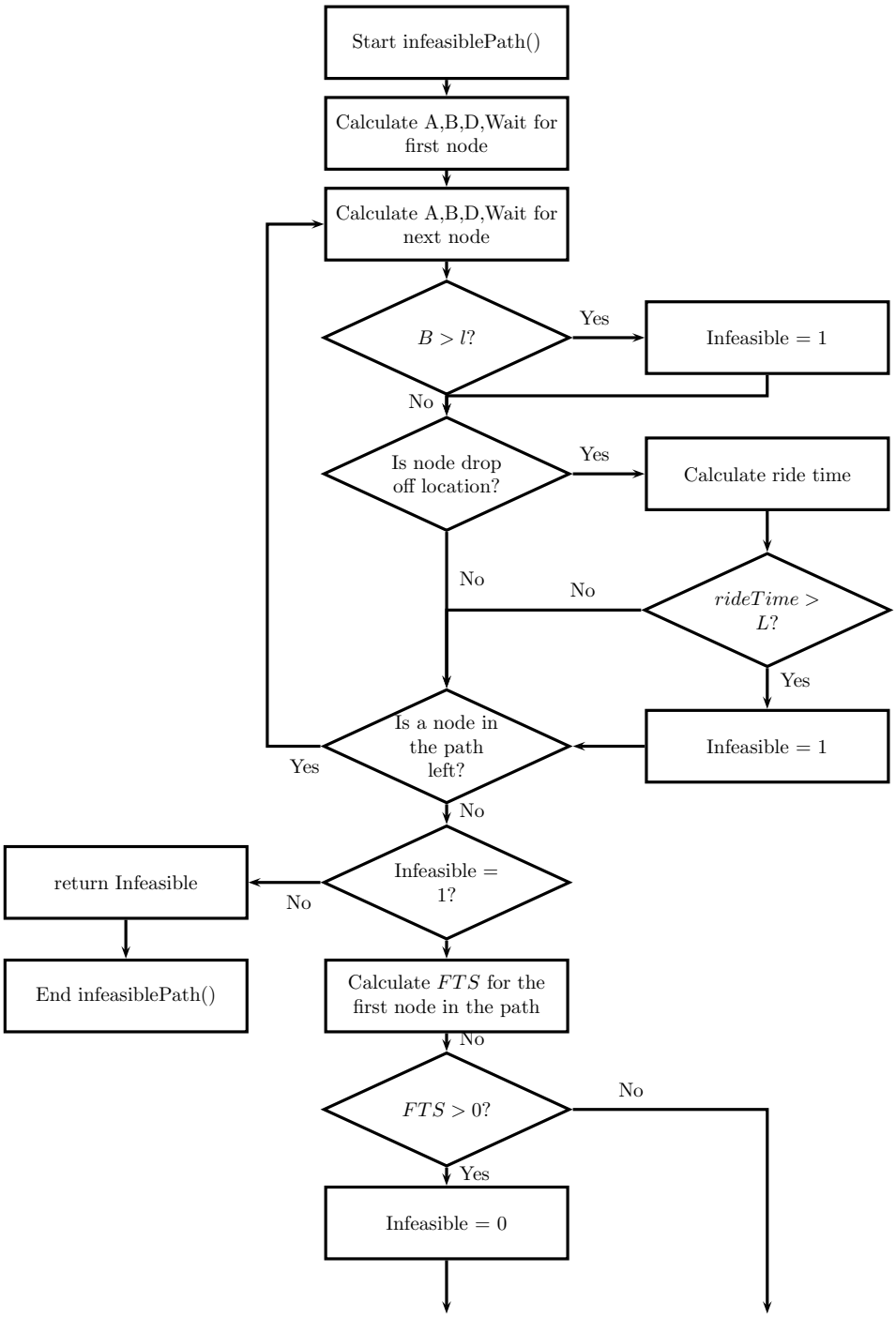

Figure 3.1.: Function infeasiblePath() - Part 1

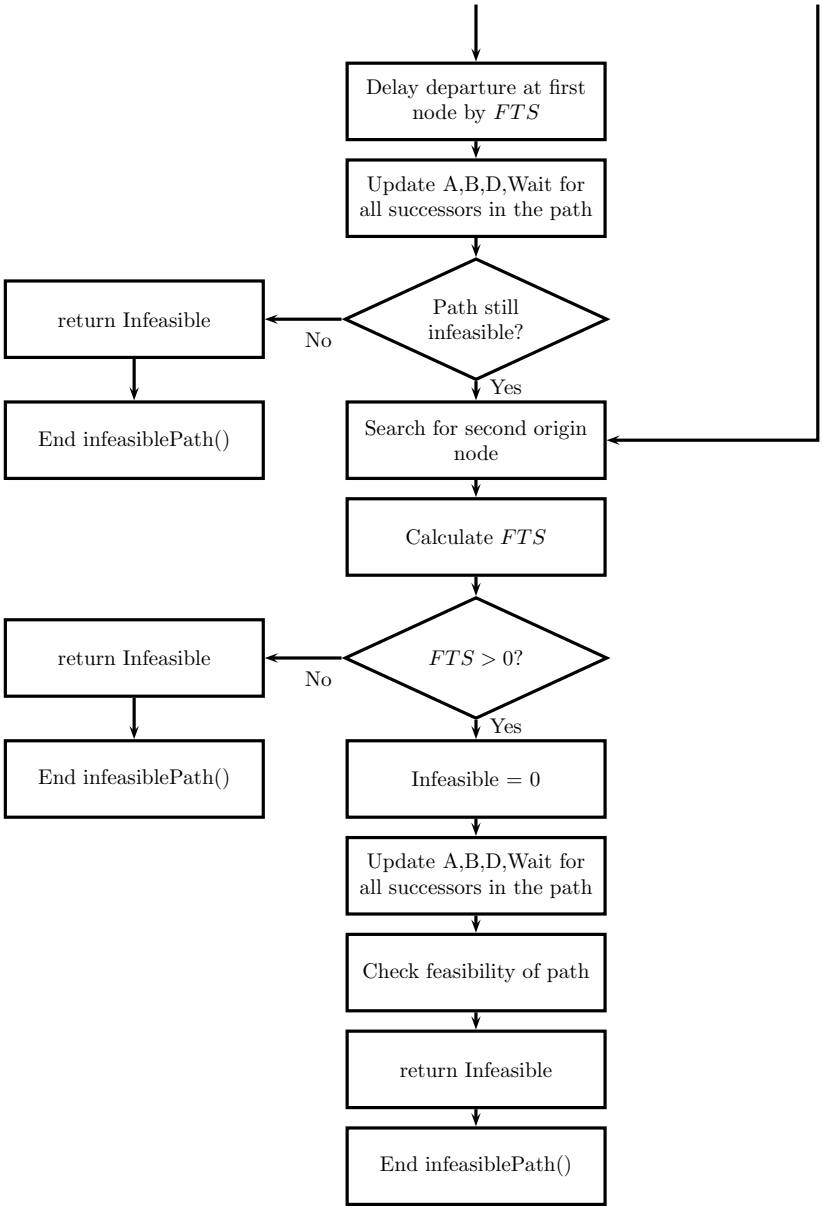

Figure 3.2.: Function infeasiblePath() - Part 2

checkPath calls for every user pair (r, s) the function *infeasiblePath()* (see Fig. 3.1 and 3.2 and Code samples 3.4 - 3.7). First, the path that has to be examined is stored in array *index*. This path consists of pick-up and drop-off nodes of users r and s, respectively. Function *infeasiblePath()* verifies feasibility of the path, and returns the value of variable *infeasible*, where *infeasible*= 0 if the path is feasible, and *infeasible*= 1, otherwise. The return value is stored in array *Path*, and corresponding arcs are removed from the graph if the path was infeasible. Arcs are removed by adding the constant *bigNumber* with value 999999 to the value of the arc.
In total, for every user pair (r, s), the function *infeasiblePath()* is called six times. After the last call, users r and s might be identified as incompatible user pair. This is the case if all six paths being examined are infeasible, and all eight arcs between $\{r, n + r\}$ and $\{s, n + s\}$ can be removed ([2] Section 5.1.2, p. 580). The call of the function *infeasiblePath()*, the generation of the paths and the processing of return values is shown in Code sample 3.4

Listing 3.4: checkPath - call of function *infeasiblePath()*

```
for(var r in pickUp){
 for(var s in pickUp){
  if(s!=r){
   for(var z in checkRange)
    Path[z]=0;
   //check Path1 s,r,n+s,n+r
   index[0]=s; index[1]=r; index[2]=nbcustomers+s; index[3]=nbcustomers+r;
   result=infeasiblePath();
   distanceTemp[r][nbcustomers+s]+=bigNumber*result;
   Path[1]=result;
   //check Path2 r,n+r,s,n+s
   index[0]=r; index[1]=nbcustomers+r; index[2]=s; index[3]=nbcustomers+s;
   result=infeasiblePath();
   distanceTemp[nbcustomers+r][s]+=bigNumber*result;
   Path[2]=result;
   //check Path3 r,s,n+r,n+s
   index[0]=r; index[1]=s; index[2]=nbcustomers+r; index[3]=nbcustomers+s;
   Path[3]=infeasiblePath();
   //check Path4 r,s,n+r,n+s,n+r
   index[0]=r; index[1]=s; index[2]=nbcustomers+s; index[3]=nbcustomers+r;
   Path[4]=infeasiblePath();
```

```
    //arc[r][s] is infeasible if path3 AND path4 both are infeasible
    if(Path[3]+Path[4]==2)
    distanceTemp[r][s]=bigNumber;
    //check Path5
    index[0]=s; index[1]=r; index[2]=nbcustomers+r; index[3]=nbcustomers+s;
    result=infeasiblePath();
    Path[5]=result;
    //arc[n+r][n+s] is infeasible if Path3 and Path5 both are infeasible
    if(Path[3]+Path[5]==2)
    distanceTemp[nbcustomers+r][nbcustomers+s]=bigNumber;
    //check Path6
    index[0]=s; index[1]=nbcustomers+s; index[2]=r; index[3]=nbcustomers+r;
    Path[6]=infeasiblePath();
    //incompatible users?
    summe=0;
    for(z in checkRange){
     summe+=Path[z]
     if( summe == 6){
      distanceTemp[r][s]=bigNumber;
      distanceTemp[s][r]=bigNumber;
      distanceTemp[r][nbcustomers+s]=bigNumber;
      distanceTemp[s][nbcustomers+r]=bigNumber;
      distanceTemp[nbcustomers+r][s]=bigNumber;
      distanceTemp[nbcustomers+s][r]=bigNumber;
      distanceTemp[nbcustomers+r][nbcustomers+s]=bigNumber;
      distanceTemp[nbcustomers+s][nbcustomers+r]=bigNumber;
    }
}}}
```

Function *infeasiblePath()* starts with the calculation of the values A(= arrival time), B(= beginning of service), D(= departure) and $Wait$(= waiting time) for the first node. The arrival time at the first node r in the path is set to the beginning of the time window of node r, e_r. The calculation of the four values for the remaining three nodes in the path is based on this time, since the arrival at the second node is equal to departure time of the predecessor plus the travel time between these two nodes. If this is earlier than the earliest starting time of the node, the vehicle has to remain idle until the beginning of the time window.

Whenever B is calculated for a node r, feasibility is checked, what means that B must not be later than the latest starting time l_r. If this is the case, variable $infeasible$ is set

equal to 1. In addition, if the node is a destination node, customer ride time is computed. When computation of values A, B, D, and $Wait$ for all nodes is finished, and the path is feasible, function $infeasiblePath()$ is ended and the value of $infeasible$ is returned. The described procedure is shown in Code sample 3.5.

Listing 3.5: infeasiblePath() - calculation of times and first feasibility check

```
function infeasiblePath(){
 infeasible=0;               //reset infeasible & forward time slacks
 for(var p=0; p<=3; p++){
  FTS[p]=0;
 }
 //calculate times for first Element
 var h=0;
 A[h]=earliestStartingTime[index[h]];       //A=arrival time
 B[h]=A[h];                                 //B=beginning of service
 D[h]=B[h]+serviceDuration[index[h]];       //D=departure time
 Wait[h]=B[h] - A[h];                       //Wait=waiting time
 //claculate times for Elements 2-4
 for(h=1; h<=3; h++){
  A[h]=D[h-1]+distance[index[h-1]][index[h]];
  B[h]=Math.max(A[h], earliestStartingTime[index[h]]);
  if( B[h] > latestStartingTime[index[h]])
   infeasible=1;
  D[h]=B[h]+serviceDuration[index[h]];
  Wait[h]=B[h] - A[h];
  if(index[h]>=nbcustomers+1){
  //if node is destination -> calculate ride time of user
   for(var v=0; v<h; v++){
    if(index[v]==index[h]-nbcustomers){
     RT[v]=B[h]-D[v];
     if(RT[v] >= maxridetime)          //is ride time feasible?
      infeasible=1;
 }}}}
```

If $infeasible = 1$ at this point, the path might be infeasible. Therefore, the forward time slack as described on page 8 is calculated. If the slack is positiv, beginning of service at the first node can be delayed and the path might become feasible. (See Listing 3.6.) Since a path consists of the pick-up and drop-off nodes of two requests, respectively, and a user has to be picked up in any case before drop-off, the first node in a path is always an origin node.

Listing 3.6: infeasiblePath() - calculation of first forward time slack

```
if(infeasible==1){
 h=0;
 mini=9999;
 for(var j=0; j<=3; j++){
  summe=0;
  for(var s=1; s<=j; s++) summe=summe+Wait[s]
  if( summe+(Math.max(latestStartingTime[index[j]]-B[j],0)) < mini){
   mini= summe + Math.max(latestStartingTime[index[j]] - B[j],0);
  }
 }
 if(mini<9999) {FTS[h]=mini;}
 else {FTS[h]=0;}
 if(FTS[h] > 0){        //Forward time slack is positive
  infeasible=0;
  B[h]=B[h]+FTS[h];
  D[h]=B[h]+serviceDuration[index[h]];
  Wait[h]=B[h]-A[h];
  //update times with new departure and check feasibility
  for(h=1; h<=3; h++){
   A[h]=D[h-1]+distance[index[h-1]][index[h]];
   B[h]=Math.max(A[h], earliestStartingTime[index[h]]);
   if( B[h] > latestStartingTime[index[h]])
    infeasible=1;
   D[h]=B[h]+serviceDuration[index[h]];
   Wait[h]=B[h] - A[h];
   if(index[h]>nbcustomers){     //if index[h] is destination node => check ride time
    for(v=0; v<h; v++){
     if(index[v]==index[h]-nbcustomers){
      RT[h]=B[h]-D[v];
      if(RT[h] >= maxridetime)
       infeasible=1;
}}}}}}
```

If $infeasible$ is still equal to 1 after the calculation of FTS for for the first origin, the function calculates the forward time slack for the second origin node in the path and checks again if the path is now feasible. Either way, the value of variable $infeasible$ is returned at the end of this code block as the function is now finished. (See Code sample 3.7.)

Listing 3.7: infeasiblePath() - calculation of second forward time slack

```
if(infeasible==1){
 for(var i=1; i<=3; i++){
  if(index[i] <= nbcustomers){     //2nd origin node in path
   mini=9999;
   for(j=i; j<=3; j++){
    if(index[j]>=nbcustomers+1){   //calculate P if node is destination
     for(var c=0; c<j; c++){       //search for corresponding origin node
      if(index[c] == index[j]-nbcustomers)
       P[j]=B[j]-D[c];         //P=ride time of customer index[c]
    }}
    else P[j]= -9999;                          //P=-9999 if node is origin
    summe=0;
    for(s=i+1; s<=j; s++){ summe+=Wait[s] }
    if(summe+Math.min(Math.max(0,latestStartingTime[index[j]]-B[j]),
     Math.max(0,maxridetime-P[j])) < mini)
     mini= summe+Math.min(Math.max(0,latestStartingTime[index[j]]-B[j]),
      Math.max(0,maxridetime-P[j]));
    }
    FTS[i] = mini;
   }
   if(FTS[i]>0){        //if slack of 2nd origin > 0, update times
    infeasible=0;
    //calculate new A,B,D and check feasibility
    B[i]=B[i]+FTS[i];
    D[i]=B[i]+serviceDuration[index[i]];
    //update times with new departure and check feasibility
    for(h >i; h<=3; h++){
     A[h]=D[h-1]+distance[index[h-1]][index[h]];
     B[h]=Math.max(A[h], earliestStartingTime[index[h]]);
     if( B[h] > latestStartingTime[index[h]]) {
     infeasible=1;                                    //check feasibility of B
    }
    D[h]=B[h]+serviceDuration[index[h]];
    Wait[h]=B[h] - A[h];
    if(index[h]>nbcustomers){
     for(v=0; v<h; v++){
      if(index[v]==index[h]-nbcustomers){
       RT[v]=B[h]-D[v];
       if(RT[v] >= maxridetime){               //check feasibility of ride time
        infeasible=1;
```

```
    }}}}
   }
}}}
```

3.2.4. Initial pool of inequalities

Finally, a pool of inequalities is added to the model. These constraints are checked exhaustively at every node in the branch-and-bound tree. Although this procedure increases the time spent at every node, it helps to decrease overall solution time. In the following, inequalities proposed by Cordeau ([2] p. 581) are described and their implementation in CPLEX is presented.
The following constraints are true for every pair of users $i, j \in P$.

Subtour elimination constraints

- $S = \{i, j\} \rightarrow x_{ij} + x_{ji} + x_{n+i,j} + x_{n+j,i} \leq 1$
- $S = \{i, n+j\} \rightarrow x_{i,n+j} + x_{ji} + x_{n+j,i} + x_{n+i,n+j} \leq 1$
- $S = \{i, n+i, j\} \rightarrow x_{ij} + x_{ji} + x_{i,n+i} + x_{j,n+i} + x_{n+i,j} + x_{n+j,i} + x_{n+j,n+i} \leq 2$
- $S = \{n+i, n+j\} \rightarrow x_{n+i,n+j} + x_{n+j,n+i} + x_{n+i,j} + x_{n+j,i} \leq 1$
- $S = \{i, n+j\} \rightarrow x_{i,n+j} + x_{n+j,i} + x_{ij} \leq 1$
- $S = \{i, n+i, n+j\} \rightarrow x_{i,n+j} + x_{n+j,i} + x_{i,n+i} + x_{n+j,n+i} + x_{n+i,n+j} + x_{ij} + x_{n+i,j} \leq 2$
- $S = \{n+i, j, n+j\} \rightarrow x_{n+i,j} + 2x_{j,n+i} + x_{ji} + x_{i,n+i} + x_{n+j,n+i} \leq 2$
- $S = \{i, n+i, n+j\} \rightarrow x_{i,n+i} + x_{n+i,n+j} + x_{n+j,i} + 2x_{i,n+j} \leq 1$
- $S = \{i, j, n+i, n+j\} \rightarrow x_{ij} + x_{ji} + x_{i,n+i} + x_{i,n+j} + x_{j,n+j} + x_{j,n+i} + x_{n+j,n+i} + x_{n+i,n+j} \leq 3$

Listing 3.8: Subtour elimination constraints

```
//Subtour elimination constraints
forall(i in pickUp, j in pickUp, k in vehicles: j!=i)
 ctSubEl1:
  travel[i][j][k]+travel[j][i][k]+travel[nbcustomers+i][j][k]+
  travel[nbcustomers+j][i][k] <= 1;

forall(i in pickUp, j in pickUp, k in vehicles: j!=i)
 ctSubEl2:
  travel[i][nbcustomers+j][k] + travel[nbcustomers+j][i][k] +
  travel[nbcustomers+i][nbcustomers+j][k] <= 1;

forall(i in pickUp, j in pickUp, k in vehicles: j!=i)
 ctSubEl3:
  travel[i][j][k] + travel[j][i][k] + travel[i][nbcustomers+i][k] +
  travel[j][nbcustomers+i][k] + travel[nbcustomers+i][j][k]+
  travel[nbcustomers+j][i][k] + travel[nbcustomers+j][nbcustomers+i][k] <= 2;

forall(i in pickUp, j in pickUp, k in vehicles: j!=i)
 ctSubEl4:
  travel[nbcustomers+i][nbcustomers+j][k] +
  travel[nbcustomers+j][nbcustomers+i][k] +
  travel[nbcustomers+i][j][k] + travel[nbcustomers+j][i][k] <= 1;

forall(i in pickUp, j in pickUp, k in vehicles: j!=i)
 ctSubEl5:
  travel[i][nbcustomers+j][k] + travel[nbcustomers+j][i][k] +
  travel[i][j][k] <= 1;

forall(i in pickUp, j in pickUp, k in vehicles: j!=i)
 ctSubEl6:
  travel[i][nbcustomers+j][k] + travel[nbcustomers+j][i][k]+
  travel[i][nbcustomers+i][k] + travel[nbcustomers+j][nbcustomers+i][k] +
  travel[nbcustomers+i][nbcustomers+j][k] + travel[i][j][k] +
  travel[nbcustomers+i][j][k] <= 2;

forall(i in pickUp, j in pickUp, k in vehicles: j!=i)
 ctSubEl7:
  travel[nbcustomers+i][j][k] + 2*travel[j][nbcustomers+i][k] +
  travel[j][i][k] + travel[i][nbcustomers+i][k] +
  travel[nbcustomers+j][nbcustomers+i][k] <= 2;
```

```
forall(i in pickUp, j in pickUp, k in vehicles: j!=i)
 ctSubEl8:
  travel[i][nbcustomers+i][k] + travel[nbcustomers+i][nbcustomers+j][k] +
  travel[nbcustomers+j][i][k] + 2*travel[i][nbcustomers+j][k] +
  travel[i][j][k] <= 2;

forall(i in pickUp, j in pickUp, k in vehicles: j!=i)
 ctSubEl9:
  travel[i][j][k] + travel[j][i][k] + travel[i][nbcustomers+i][k] +
  travel[i][nbcustomers+j][k] + travel[j][nbcustomers+i][k] +
  travel[j][nbcustomers+j][k] +
  travel[nbcustomers+i][nbcustomers+j][k] +
  travel[nbcustomers+j][nbcustomers+i][k] <= 3;
forall(i in pickUp, j in pickUp, k in vehicles: j!=i)
   ctSubEl9:
      travel[i][j][k] + travel[j][i][k] + travel[i][nbcustomers+i][k] +
      travel[i][nbcustomers+j][k] + travel[j][nbcustomers+i][k] +
      travel[j][nbcustomers+j][k] + travel[nbcustomers+i][nbcustomers+j][k] +
      travel[nbcustomers+j][nbcustomers+i][k] <= 3;
```

Precedence constraints

- $S = \{0, i, n+j\} \rightarrow x_{0i} + x_{i,n+j} + x_{n+j,i} \leq 1$
- $S = \{i, n+j, 2n+1\} \rightarrow x_{i,n+j} + x_{n+j,i} + x_{n+j,2n+1} \leq 1$
- $S = \{0, i, n+i, n+j\} \rightarrow x_{0i} + x_{i,n+i} + x_{i,n+j} + x_{n+j,i} + x_{n+i,n+j} + x_{n+j,n+i} \leq 2$
- $S = \{i, j, n+j, 2n+1\} \rightarrow x_{ij} + x_{ji} + x_{i,n+j} + x_{n+j,i} + x_{j,n+j} + x_{n+j,2n+1} \leq 2$

Listing 3.9: Precedence constraints

```
//Precedence constraints
forall(i in pickUp, j in pickUp, k in vehicles: j!=i)
 ctPrec1:
  travel[0][i][k] + travel[i][nbcustomers+j][k] +
   travel[nbcustomers+j][i][k] <= 1;

forall(i in pickUp, j in pickUp, k in vehicles: j!=i)
```

```
 ctPrec2:
  travel[i][nbcustomers+j][k] + travel[nbcustomers+j][i][k] +
   travel[nbcustomers+j][2*nbcustomers+1][k] <= 1;

forall(i in pickUp, j in pickUp, k in vehicles: j!=i)
 ctPrec3:
  travel[0][i][k] + travel[i][nbcustomers+i][k] +
   travel[i][nbcustomers+j][k] + travel[nbcustomers+j][i][k] +
   travel[nbcustomers+i][nbcustomers+j][k] +
   travel[nbcustomers+j][nbcustomers+i][k] <= 2;

forall(i in pickUp, j in pickUp, k in vehicles: j!=i)
 ctPrec4:
  travel[i][j][k] + travel[j][i][k] + travel[i][nbcustomers+j][k] +
   travel[nbcustomers+j][i][k] + travel[j][nbcustomers+j][k] +
   travel[nbcustomers+j][2*nbcustomers+1][k] <=2;
```

Generalized order constraints

- $x_{i,n+j} + x_{n+j,i} + x_{n+i,j} + x_{j,n+i} \leq 1$

Listing 3.10: Generalized order constraints

```
//Generalized order constraints
forall(i in pickUp, j in pickUp: j!=i)
 ctGenOrder:
  sum(k in vehicles) travel[i][nbcustomers+j][k] +
   sum(k in vehicles) travel[nbcustomers+j][i][k] +
   sum(k in vehicles) travel[nbcustomers+i][j][k] +
   sum(k in vehicles) travel[j][nbcustomers+i][k] <= 1;
```

Infeasible path constraints

Finally, for every pair of users $i, j \in P$ such that $t_{ij} + d_j + t_{j,n+j} + d_{n+j} + t_{n+j,n+i} > L$, inequality

- $x_{ij} + x_{j,n+j} + x_{n+j,n+i} \leq 1$

is generated.

Listing 3.11: Infeasible path constraints

```
//Infeasible path constraints
forall(i in pickUp, j in pickUp, k in vehicles: j!=i){
 if(distance[i][j] + serviceDuration[j] + distance[j][nbcustomers+j] +
  serviceDuration[nbcustomers+j] +
  distance[nbcustomers+j][nbcustomers+i] > maxridetime){
   travel[i][j][k] + travel[j][nbcustomers+j][k] +
   travel[nbcustomers+j][nbcustomers+i][k] <= 1;
}}
```

4. Parameters and test instances

Five parameters of CPLEX, that are evaluated in this work, are described in detail in Section 4.1. These parameters were chosen because they were expected to have the most potential to affect the solution process. An overview of available settings for these parameters is given in Table 4.1. Test instances are described in Section 4.2.

4.1. Parameters

Epsilon Gap (EpGap)
This parameter sets a relative tolerance on the gap between the best integer objective value and the objective value of the best node remaining. The gap is calculated by $|bestnode - bestinteger|/(1E{-}10 + |bestinteger|)$. If this value is smaller than the gap, the optimization process is stopped. The default value is 0.0001, therefore, CPLEX stops by default if the integer solution found is at most 0,01% away from the optimum.

Heuristic Frequency (HeurFreq)
The parameter *HeurFreq* determines how often a periodic heuristic is applied. The user can choose not to use the heuristic at all, or can set a node interval for heuristic-requests (e.g. setting the parameter to 10 causes CPLEX to call the heuristic at every 10^{th} node). It is also possible to let CPLEX choose by setting the parameter to 0.

MIP Emphasis (MIPEmph)
This parameter enables the user to choose between receiving a feasible solution early, or to wait longer for CPLEX to solve the problem, but to receive an optimal solution possibly faster, because CPLEX concentrates on finding the optimal solution and produces less feasible solutions. In total, 5 different settings are possible. The default setting balances feasibility and optimality, other settings are: emphasize feasibility over optimality, emphasize optimality over feasibility, emphasize moving best bound (placeing even more effort

on optimality), and emphasize finding hidden feasible solutions, a setting that should be used if the algorithm has difficulties to find feasible solutions of acceptable quality.

Mixed Integer Rounding Cuts (MIRCuts)
The mixed integer rounding cut indicator determines whether or not MIR cuts are generated for the problem. The user is free to choose an aggressive generation of cuts, a moderate generation, no generation at all, or to let CPLEX decide if and how many cuts are generated.

Variable Selection (VarSel)
The last parameter affects the branching strategy. CPLEX offers six different possibilities, on which variable to branch (see also Table 4.1 below). From a chosen node, the variable with minimum infeasibility is chosen for branching by setting $VarSel$ to -1, the most infeasible variable is chosen by setting the parameter equal to 1. Other possible strategies are branching based on pseudo cost, or on pseudo reduced cost, or to apply strong branching, where CPLEX solves a number of subproblems to see which variable is the most promising and is therefore chosen for branching. By default, CPLEX choses automatically on which variable to branch next.

4.2. Instances

Testing was performed on a 3.2 GHz Pentium 4 computer with 4 GB of memory. The model was implemented with the ILOG OPL Development Studio 5.0, using CPLEX 10.2.

In total, twelve instances were used. These were the same ones used by Cordeau [2], to maintain comparability of results. They are described in Table 4.2. Instances are generated with two, three or four vehicles, serving between 16 and 48 customers. Half the requests constitute inbound requests, the other half are outbound requests. Every vehicle has a maximum capacity $Q = 3$. A customer who is picked up has a load of $q_i = 1$, and a negative load $q_{n+i} = -1$ is associated with every drop-off node.

The coordinates for pick-up and drop-off nodes are chosen in the square $[-10, 10] \times [-10, 10]$, randomly and independently, and the depot is located in the center of the square. For

EpGap	1.00E-04	Values
	$0.0 \leq$ EpGap ≤ 1.0	any Value between 0 and 1
HeurFrequ	0	Frequency is chosen by CPLEX (Defaultvalue)
	-1	heuristic isn't applied at all
	any positive integer	e.g. 10: heuristic is applied at every 10^{th} node
MIPEmph	0	balance optimality and feasibility(Defaultvalue)
	1	emphasize feasibility over optimality
	2	emphasize optimality over feasibility
	3	emphasize moving best bound
	4	emphasize finding hidden feasible solutions
MIRCuts	0	let CPLEX choose (Defaultvalue)
	1	generate cuts moderately
	2	generate cuts aggressively
	-1	no generation of cuts
VarSel	0	automatic selection (Defaultvalue)
	-1	branch on variable with minimum infeasibility
	1	branch on variable with maximum infeasibility
	2	branching based on pseudo cost (shadow prices)
	3	strong branching (CPLEX solves subproblems to see which one is the most promising)
	4	branching based on pseudo reduced cost

Table 4.1.: Paramter values

Instance	Customers	Vehicles	T	Q	L	Upper bound
a2-16	16	2	480	3	30	294.26
a2-20	20	2	600	3	30	344.84
a2-24	24	2	720	3	30	431.13
a3-18	18	3	360	3	30	300.49
a3-24	24	3	480	3	30	344.84
a3-30	30	3	600	3	30	496.53
a3-36	36	3	720	3	30	600.76
a4-16	16	4	240	3	30	282.89
a4-24	24	4	360	3	30	375.08
a4-32	32	4	480	3	30	486.57
a4-40	40	4	600	3	30	571.08
a4-48	48	4	720	3	30	681.00

T... max. route duration, Q... max. capacity per vehicle,
L... max. ride time per user

Table 4.2.: Test instances by Cordeau [2], available on `http://www.hec.ca/chairedistributique/data/darp`

every node pair (i, j) the Euclidean distance is computed, which act concurrently as travel time t_{ij} and travel cost c_{ij} of arc $(i, j) \in A$. Calculation of the Euclidean distance for the distance matrix is shown in Code sample 4.1.
Further, a time window $[e_i, l_i]$ for every node $i \in V$ is generated. Time windows are only allowed on drop-off nodes for outbound requests, and on pick-up nodes for inbound requests, respectively. For an outbound user, a window is generated by choosing a number l_i in the interval $[60, T]$, where T is the length of the planning horizon, and the beginning of the time window is fixed by setting $e_i = l_i - 15$. For an inbound user, first e_i is set to a value in the interval $[0, T - 60]$ and then the end of the time window is defined by setting $l_i = e_i + 15$.
Finally, maximum ride time L is set to 30 minutes and is equal for all users, as well as service time d_i, which is three minutes for every user.

Listing 4.1: Calculation of Euclidean distances

```
//coordinates of vertices
tuple coordinates{
 float x;
 float y;
}
coordinates point[nodes]=...;

//Distancematrix
float distance[nodes][nodes];
float distanceTemp[nodes][nodes];

execute CalculateDistance{
//calculate euclidian distances between vertices
for(var v in nodes){
 for(var d in nodes)
  distance[v][d]=Math.sqrt(Math.pow((point[v].x - point[d].x),2) +
   Math.pow((point[v].y - point[d].y),2));
 }
//copy distance matrix into temp matrix
 for(v in nodes){
  for(d in nodes)
   distanceTemp[v][d]=distance[v][d];
}}
```

5. Results

In total, four different analyses were performed. First, a general analysis was performed where the five parameters described above were tested on twelve instances. In addidion, every instance was solved with all parameters left as default, to compare them with the results achieved when parameter values were changed. Testing was done by changing every parameter successively, leaving all other paramters as default, and running CPLEX with every possible value for this parameter. Results of the general analysis can be examined in Tables 5.1 - 5.5. In every table, the first column indicates the instance, and the first row indicates the value for the parameter tested. For every parameter setting, column "Obj. Val." shows the value of the best solution found. If this solution is optimal, the value is marked with an asterisk. In some cases the optimal solution value is indicated in a table but is not marked with an asterisk. This means that CPLEX was not able to prove optimality for this solution bevore the search was aborted due to time out. Column "CPU" shows the solution time in seconds. If neither a feasible nor an optimal solution was found before the process was stopped after two hours (7200 sec.), the solution field remains empty.
Secondly, two parameters, the ones that reached the best results in the first analysis, were combined. The third analysis is similar to the first one, with the difference that upper bounds are set. Finally, for the last analysis, test instances are solved with XPRESS.

5.1. General analysis

Results of the general analysis
Results for parameter *Epsilon Gap* in Table 5.1 show that changing the value for this parameter does not influence the solution compared to default values. The parameter indicates the gap between upper and lower bound that is allowed to call a solution optimal. Average solution time and solutions found (optimal and feasible ones) were the same for all settings.

Varations of *heuristic frequency* led to different results. Whereas results were worse when the heuristic was not applied at all, the best results were reached by setting HeurFreq = 10. Applying the heuristic at every node ($HeurFreq = 1$) was slightly more time consuming. Results are shown in Table 5.3.

Table 5.4 shows results for parameter *MIP Emphasis*. By changing this parameter, more effort is applied on finding either optimal or feasible solutions, depending on setting. By default, optimality and feasibility are balanced. Finding many feasible solutions appeared to be time consuming, and did not result in more feasible solutions found for bigger instances. When the parameter was set to 2, emphasizing the search for the optimal solution, average solution time was considerably lower. Unfortunately, no more instances were solved to optimality than with other settings. Emphasizing the search for hidden feasible solutions did not lead to the generation of more feasible solutions, but those found were better (lower objective value) than the ones found with $MIPEmph = 1$ (emphasize feasibility over optimality). Setting $MIPEmph = 3$ (moving best bound) does not seem to be recommendable.

Results for parameter *MIRCuts* are presented in Table 5.2. By default, CPLEX generates cuts only if this seems to be helping. If no cuts were generated at all, as many optimal and feasible solutions were found as with default settings during testing, and average solution time was nearly the same. This is also true for paramter setting $MIRCuts = 2$. On the other hand, telling CPLEX to generate cuts moderatly led to more feasible solutions found, and mostly better objective values, at some sacrifice in average solution speed.

The last parameter, *Variable selection*, determines the branching strategy performed by CPLEX. Branching on the variable with minimum infeasibility did not proof to be efficient, as time spent per instance on average was low, but results computed are worse than with default settings. Branching based on pseudo reduced costs led to same results as default values. The three remaining settings led at all instances, except one, to better results for the objective value compared to results achieved with default values (see Table 5.5).

Value EpGap	1.00E-02		1.00E-06		Default settings	
Instance	Obj.Val.	CPU	Obj.Val.	CPU	Obj.Val.	CPU
a2-16	*294.248	3.58	*294.248	3.60	*294.248	3.61
a2-20	*344.834	19.86	*344.834	21.54	*344.834	21.36
a2-24	*431.120	85.59	*431.120	89.14	*431.120	89.45
a3-18	*300.483	258.15	*300.483	274.30	*300.483	274.54
a3-24	351.045	7229.71	351.045	7229.96	351.045	7230.20
a3-30	503.897	7230.93	503.897	7230.98	503.897	7230.53
a3-36	613.284	7248.93	613.284	7249.03	607.286	7249.09
a4-16	*282.677	3333.77	*282.677	3491.72	*282.677	3476.41
a4-24	375.020	7228.8	375.020	7229.06	375.020	7228.91
a4-32	584.408	7229.66	584.408	7229.47	584.408	7229.32
a4-40		7200.00		7200.00		7200.00
a4-48		7200.00		7200.00		7200.00

Obj.Val. . . . best solution found, marked with an asterisk * if optimality was proven
CPU. . . solution time in seconds

Table 5.1.: General analysis - Results for parameter EpGap and default settings

Value MIRCuts	-1		1		2	
Instance	Obj.Val.	CPU	Obj.Val.	CPU	Obj.Val.	CPU
a2-16	*294.248	3.60	*294.248	2.18	*294.248	3.58
a2-20	*344.834	21.33	*344.834	50.22	*344.834	21.39
a2-24	*431.120	87.89	*431.120	67.47	*431.120	90.51
a3-18	*300.483	275.26	*300.483	383.44	*300.483	276.04
a3-24	351.045	7229.93	345.232	7252.57	351.045	7230.09
a3-30	503.897	7230.84	504.817	7226.63	503.897	7230.76
a3-36	607.286	7249.21	594.444	7231.43	602.39	7248.7
a4-16	*282.677	3482.28	*282.677	6344.54	*282.677	3508.4
a4-24	375.020	7228.79	379.117	7226.53	375.02	7229
a4-32	591.142	7229.36	514.666	7223.6	584.408	7229.59
a4-40		7200.00	626.651	7226.07		7200.00
a4-48		7200.00		7200.00		7200.00

Obj.Val. . . . best solution found, marked with an asterisk * if optimality was proven
CPU. . . solution time in seconds

Table 5.2.: General analysis - Results for parameter MIR Cuts

ValueHeurFreq	-1		1		10		20	
Instance	Obj.Val.	CPU	Obj.Val.	CPU	Obj.Val.	CPU	Obj.Val.	CPU
a2-16	*294.248	3.18	*294.248	4.20	*294.248	4.17	*294.248	4.15
a2-20	*344.834	14.38	*344.834	21.38	*344.834	17.90	*344.834	16.44
a2-24	*431.120	44.40	*431.120	102.37	*431.120	68.84	*431.120	69.42
a3-18	*300.483	172.10	*300.483	337.50	*300.483	348.44	*300.483	221.22
a3-24	347.596	7228.71	344.834	7231.44	347.341	7230.31	350.512	7230.01
a3-30	526.483	7229.56	501.460	7231.20	501.220	7230.81	501.855	7230.02
a3-36	608.178	7245.13	588.511	7253.25	584.441	7251.95	587.727	7247.37
a4-16	*282.677	3435.69	*282.677	2138.41	*282.677	1549.20	*282.677	1775.9
a4-24	401.365	7229.13	375.316	7231.83	381.082	7230.71	385.103	7228.9
a4-32		7200.00	527.628	7228.79		7200.00	508.774	7228.76
a4-48		7200.00		7200.00		7200.00		7200.00
a4-48		7200.00		7200.00		7200.00		7200.00

Obj.Val. . . . best solution found, marked with an asterisk * if optimality was proven
CPU. . . solution time in seconds

Table 5.3.: General analysis - Results for parameter Heuristic Frequency

Value MIPEmph	1		2		3		4	
Instance	Obj.Val.	CPU	Obj.Val.	CPU	Obj.Val.	CPU	Obj.Val.	CPU
a2-16	*294.248	2.22	*294.248	3.92	*294.248	5.43	*294.248	3.58
a2-20	*344.834	49.56	*344.834	23.07	*344.834	63.89	*344.834	19.15
a2-24	*431.120	162.08	*431.120	73.49	*431.120	465.71	*431.120	75.24
a3-18	*300.483	71.13	*300.483	162.04	*300.483	493.44	*300.483	351.66
a3-24	344.834	7229.33	363.741	7229.18	353.749	7253.93	345.232	7230.29
a3-30	494.847	7229.83	502.65	7229.55	498.717	7254.83	495.555	7231.88
a3-36	583.187	7231.40	598.648	7239.22	587.755	7257.02	585.218	7247.44
a4-16	*282.677	815.51	*282.677	1863.27	282.677	7240.52	*282.677	3991.42
a4-24	376.212	7228.03	416.617	7229.24		7200.00	375.316	7230.49
a4-32		7200.00		7200.00		7200.00		7200.00
a4-40	648.213	7228.44		7200.00		7200.00		7200.00
a4-48		7200.00		7200.00		7200.00		7200.00

Obj.Val. . . . best solution found, marked with an asterisk * if optimality was proven
CPU. . . solution time in seconds

Table 5.4.: General analysis - Results for parameter MIP Emphasis

Value VarSel	-1		1		2		3		4	
Instance	Obj.Val.	CPU	Obj.Val.	CPU	Obj.Val.	CPU	Obj.Val.	CPU	Obj.Val.	CPU
a2-16	*294.248	2.46	*294.248	2.13	*294.248	3.52	*294.248	3.56	*294.248	2.3
a2-20	*344.834	445.02	*344.834	48.56	*344.834	20.72	*344.834	31.7	*344.834	60.16
a2-24	*431.120	1249.92	*431.120	65.02	*431.120	86.79	*431.120	131.95	*431.120	143.09
a3-18	*300.483	4684.57	*300.483	370.59	*300.483	266.52	*300.483	116.4	*300.483	89.65
a3-24	358.388	7236.84	345.232	7227.99	351.045	7229.61	345.232	7236.43	346.808	7227.26
a3-30		7200.00	502.592	7227.15	503.897	7230.27	496.518	7235.82	494.847	7227.82
a3-36		7200.00	593.572	7232.36	614.896	7248.87	588.401	7241.82	583.299	7228.75
a4-16	285.987	7241.06	*282.677	6088.00	*282.677	3368.04	*282.677	4945.82	*282.677	786.74
a4-24	381.018	7225.18	379.117	7226.6	375.02	7228.61	384.179	7232.7	379.117	7226.18
a4-32	536.857	7228.67	514.819	7224.51	591.142	7229.42	502.178	7228.49	517.428	7223.19
a4-40		7200.00	625.811	7226.75		7200.00		7200.00	647.427	7225.47
a4-48		7200.00		7200.00	723.120	7234.80		7200.00		7200.00

Obj.Val... best solution found, marked with an asterisk * if optimality was proven
CPU... solution time in seconds

Table 5.5.: General analysis - Results for parameter Variable Selection

5.2. Parameter combination

After examining results of the first analysis, two conclusions can be drawn: Firstly, changing one parameter alone does in general not have great impact on solutions, except parameters *MIRCuts* and *VarSel*. Therefore, in the second run the most promising parameter values are combined. It has to be noted that combining all parameters among one another would clearly go beyond the scope of this work.
Secondly, for bigger instances, CPLEX was not able to find a feasible solution at all (empty cells in tables 5.1 to 5.5). To tackle this conclusion, upper bounds, that were computed by Cordeau [2, 4] by running a tabu search heuristic for 1000 iterations, are added in the third run. Upper bound values are indicated in Table 4.2 on page 34.

Results of combined parameters

	VarSel=1,MIRCuts=1		VarSel=1,MIRCuts=3	
Instance	Obj.Val.	CPU	Obj.Val.	CPU
a2-16	*294.248	2.52	*294.248	3.45
a2-20	*344.834	58.28	*344.834	34.70
a2-24	*431.120	154.29	*431.120	78.71
a3-18	*300.483	294.69	*300.483	146.04
a3-24	346.808	7227.38	344.834	7236.56
a3-30	500.969	7226.92	503.168	7236.34
a3-36	583.187	7232.72	587.064	7241.93
a4-16	282.677	7234.24	*282.677	2357.67
a4-24	375.020	7227.22	389.828	7233.51
a4-32	509.69	7223.50	513.061	7228.74
a4-40	607.519	7227.07	599.143	7231.51
a4-48		7200.00		7200.00

Obj.Val... best solution found, marked with an asterisk * if optimality was proven, CPU... solution time in seconds

Table 5.6.: Results Variable Selection and MIP Emphasis

The combination of the two parameters was successive in finding at least feasible solutions for all instances except the biggest one (a4-48), as can be seen in table 5.6. Interestingly, instance a3-24 could still not be solved to optimality, as it was by Cordeau (compare Table 5.7 on page 44). This might be due to upper bounds provided, as done by Cordeau [2], therefore they are added in the next run, too.

5.3. Improved upper bounds

In the third analysis, upper bounds computed by Cordeau with a tabu search heuristic, were set. These bounds are set by adding 0.01 to the computed solution. This is done to avoid that too many branches are cut up due to rounding. The same twelve instances were solved again, with all parameter settings that were tested in the general analysis. Results are presented in Tables 5.7 - 5.12.
Graphs 5.1 - 5.4 in Section 5.4 show a summary of the first and third analysis, average solution times per instance, per parameter, and number of solutions found (optimal and feasible ones), for every parameter value.

Results of the analysis with upper bounds
The provision of upper bounds did not lead to much better results than without those values. Nearly the same instances were solved to optimality, and for even less instances CPLEX could find feasible solutions with default settings. In general, for all instances considerably less feasible solutions were generated than without upper bounds. (On average, 5 feasible solutions without, and only one feasible solution with upper bounds, respectively). This might be because, by providing upper bound values, branches with feasible solutions are pruned earlier, therefore less feasible solutions are found.

Again, changing parameter *Epsilon Gap* (EpGap) did not influence solutions compared to default settings, as can be examined in Table 5.8.

Changing parameter *Heuristic Frequency* (HeurFreq) did not enhance solution finding, either. As shown in Table 5.10, for five instances optimal solutions were found, and at most two feasible ones. This was the case when the heuristic was applied at every node in the search tree. Results also approved the conclusion of the first run, that, if the heuristic was not applied at all ($HeurFreq = -1$), no improvement of solution times could be achieved. On the other hand, forbidding the heuristic led to worse results than if the heuristic is applied. The best results were achieved by applying the heuristic at every node. This might be time consuming for bigger problems, but results showed that the solution for the biggest instance solved to optimality, instance a4-16, was found much faster than if the heuristic was applied at every 20^{th} node (1914,39 and 2496,04 seconds, respectively).

Table 5.11 presents results for parameter *MIP Emphasis*. Here, the difference between emphasizing feasibility over optimality, or vice versa, can be observed very well, as three feasible solutions are found with setting $MIPEmph = 1$ (emphasizing feasibility), as opposed to all other parameter values, where only one feasible solution was found. Results also confirmed the author's impression that setting $MIPEmph = 3$ (emphasize moving best bound) is overall not very recommendable, as it did not provide better solutions and was slower than other settings.

Results for parameter *MIR Cuts* can be examined in Table 5.9. As in the first analysis, best results were achieved by telling CPLEX to moderately genererate MIR cuts. It should be noted that, during the whole testing phase, this was the only time where instance a3-24 could be solved to optimality. Interestingly, solution time was nearly the same for settings $MIRCuts$ =-1, no generation of cuts, and $MIRCuts = 2$, aggressive generation of cuts, though expectations were that the latter one would be more time consuming.

The last parameter, *Variable Selection* (VarSel), provided again good results for all settings except branching on variable with minimum infeasibility ($VarSel$ =-1). Best results in both, terms of solutions found and CPU-time were delivered by strong branching, as shown in Table 5.12.

Instance	Obj.Val.	CPU	Upper bound	Results Cordeau
a2-16	*294.248	3.58	294.26	*294.25
a2-20	*344.834	18.94	344.84	*344.83
a2-24	*431.120	74.6	431.13	*431.12
a3-18	*300.483	237.55	300.49	*300.48
a3-24		7200.00	344.84	*344.83
a3-30		7200.00	496.53	472.17
a3-36	583.299	7245.24	600.76	570.26
a4-16	*282.677	3808.45	282.69	*282.68
a4-24		7200.00	375.08	359.52
a4-32		7200.00	486.58	427.65
a4-40		7200.00	571.08	462.21
a4-48		7200.00	681.00	466.7

Obj.Val. . . . best solution found, marked with an asterisk * if optimality was proven, CPU. . . solution time in seconds,
Upper bound & Results Cordeau . . . compare [2], pp. 582 and 584

Table 5.7.: Improved upper bounds - Results for default setting

Value EpGap	0.01		0.000001	
Instance	Obj.Val.	CPU	Obj.Val.	CPU
a2-16	*294.248	3.62	*294.248	3.66
a2-20	*344.834	18.48	*344.834	19.39
a2-24	*431.120	71.61	*431.12	75.66
a3-18	*300.483	229.54	*300.483	242.43
a3-24		7200.00		7200.00
a3-30		7200.00		7200.00
a3-36	585.155	7245.57	585.155	7245.91
a4-16	*282.677	3747.97	*282.677	3878.64
a4-24		7200.00		7200.00
a4-32		77200.00		7200.00
a4-40		7200.00		7200.00
a4-48		7200.00		7200.00

Obj.Val. . . . best solution found, marked with an asterisk * if optimality was proven
CPU. . . solution time in seconds

Table 5.8.: Improved upper bounds - Results for parameter EpGap

Value MIRCuts	-1		1		2	
Instance	Obj.Val.	CPU	Obj.Val.	CPU	Obj.Val.	CPU
a2-16	*294.248	3.68	*294.248	3.5	*294.248	3.54
a2-20	*344.834	19.46	*344.834	17.54	*344.834	19.72
a2-24	*431.120	76.7	*431.120	69.73	*431.12	95.7
a3-18	*300.483	243.67	*300.483	225.35	*300.483	209.6
a3-24		7200.00	*344.834	5318.96	344.834	7241.97
a3-30		7200.00		7200.00		7200.00
a3-36	585.155	7246.46	585.074	7249.66		7200.00
a4-16	*282.677	3929.05	*282.677	2564.32	*282.677	3594.88
a4-24		7200.00		7200.00		7200.00
a4-32		7200.00		7200.00		7200.00
a4-40		7200.00		7200.00		7200.00
a4-48		7200.00		7200.00		7200.00

Obj.Val. . . . best solution found, marked with an asterisk * if optimality was proven
CPU. . . solution time in seconds

Table 5.9.: Improved upper bounds - Results for parameter MIRCuts

Value HeurFreq	-1		1		10		20	
Instance	Obj.Val.	CPU	Obj.Val.	CPU	Obj.Val.	CPU	Obj.Val.	CPU
a2-16	*294.248	3.07	*294.248	3.70	*294.248	3.67	*294.248	3.67
a2-20	*344.834	12.67	*344.834	34.52	*344.834	18.92	*344.834	20.78
a2-24	*431.120	88.42	*431.120	99.34	*431.120	59.51	*431.120	83.03
a3-18	*300.483	200.13	*300.483	339.84	*300.483	257.53	*300.483	256.33
a3-24		7200.00	344.834	7231.09		7200.00		7200.00
a3-30		7200.00		7200.00		7200.00		7200.00
a3-36		7200.00	583.299	7258.1	592.640	7249.17		7200.00
a4-16	*282.677	969.7	*282.677	1914.39	*282.677	2127.62	*282.677	2496.04
a4-24		7200.00		7200.00		7200.00		7200.00
a4-32		7200.00		7200.00		7200.00		7200.00
a4-40		7200.00		7200.00		7200.00		7200.00
a4-48		7200.00		7200.00		7200.00		7200.00

Obj.Val. . . . best solution found, marked with an asterisk * if optimality was proven
CPU. . . solution time in seconds

Table 5.10.: Improved upper bounds - Results for parameter Heuristic Frequency

Value MIPEmph	1		2		3		4	
Instance	Obj.Val.	CPU	Obj.Val.	CPU	Obj.Val.	CPU	Obj.Val.	CPU
a2-16	*294.248	2.02	*294.248	4.12	*294.248	6.18	*294.248	3.57
a2-20	*344.834	58.77	*344.834	18.37	*344.834	63.2	*344.834	20.43
a2-24	*431.120	338.5	*431.120	60.4	*431.120	289.16	*431.120	74.04
a3-18	*300.483	55.06	*300.483	139.51	*300.483	468.53	*300.483	240.09
a3-24	344.834	7228.77	344.834	7231	*344.834	3710.43		7200.00
a3-30	494.847	7229.42		7200.00		7200.00		7200.00
a3-36	583.187	7230.26		7200.00		7200.00	583.299	7244.79
a4-16	*282.677	653.83	*282.677	2035.81	*282.677	7241.72	*282.677	3873.57
a4-24		7200.00		7200.00		7200.00		7200.00
a4-32		7200.00		7200.00		7200.00		7200.00
a4-40		7200.00		7200.00		7200.00		7200.00
a4-48								

Obj.Val.... best solution found, marked with an asterisk * if optimality was proven
CPU... solution time in seconds

Table 5.11.: Improved upper bounds - Results for parameter MIP Emphasis

Value VarSel	-1		1		2		3		4	
Instance	Obj.Val.	CPU	Obj.Val.	CPU	Obj.Val.	CPU	Obj.Val.	CPU	Obj.Val.	CPU
a2-16	*294.248	2.50	*294.248	2.18	*294.248	3.69	*294.248	3.23	*294.248	2.14
a2-20	*344.834	291.43	*344.834	60.58	*344.834	19.45	*344.834	14.41	*344.834	56.02
a2-24	*431.120	3367.26	*431.120	270.19	*431.120	75.83	*431.120	82.93	*431.120	235.86
a3-18	*300.483	3920.95	*300.483	285.19	*300.483	243.39	*300.483	84.88	*300.483	78.34
a3-24		7200.00		7200.00		7200.00		7200.00		7200.00
a3-30		7200.00		7200.00		7200.00	495.555	7235.74		7200.00
a3-36		7200.00	591.399	7233.12	585.155	7246.11	590.316	7241.71	588.401	7227.41
a4-16		7200.00	*282.677	6000.98	*282.677	3908.32	*282.677	1442.62	*282.677	571.36
a4-24		7200.00		7200.00		7200.00		7200.00		7200.00
a4-32		7200.00		7200.00		7200.00		7200.00		7200.00
a4-40		7200.00		7200.00		7200.00		7200.00		7200.00
a4-48		7200.00		7200.00		7200.00		7200.00		7200.00

Obj.Val. . . . best solution found, marked with an asterisk * if optimality was proven
CPU. . . solution time in seconds

Table 5.12.: Improved upper bounds - Results for parameter Variable Selection

5.4. Comparing results of first and third analysis

Results of the first and third analysis are summed up in the following. Graphs 5.1 and 5.3 show for every parameter value the number of instances that were solved optimally or feasible, and how many instaces could not be solved before the process was aborted. As already mentioned above, feasible solutions were found for more instances during the general analysis.

Graphs 5.2 and 5.4 show average solution time per instance per parameter setting. Obviously, solution time was in both runs very high for $VarSel = -1$ (branching on variable with minimum infeasibility), whereas all other solution times are nearly the same.

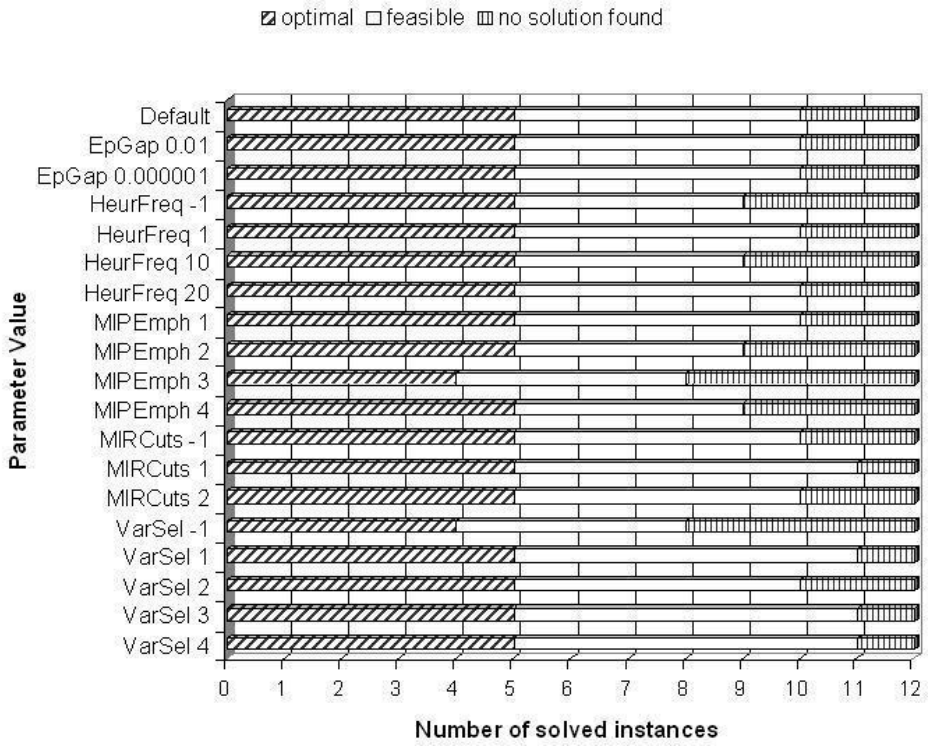

Figure 5.1.: General analysis - solved instances without initial solution.

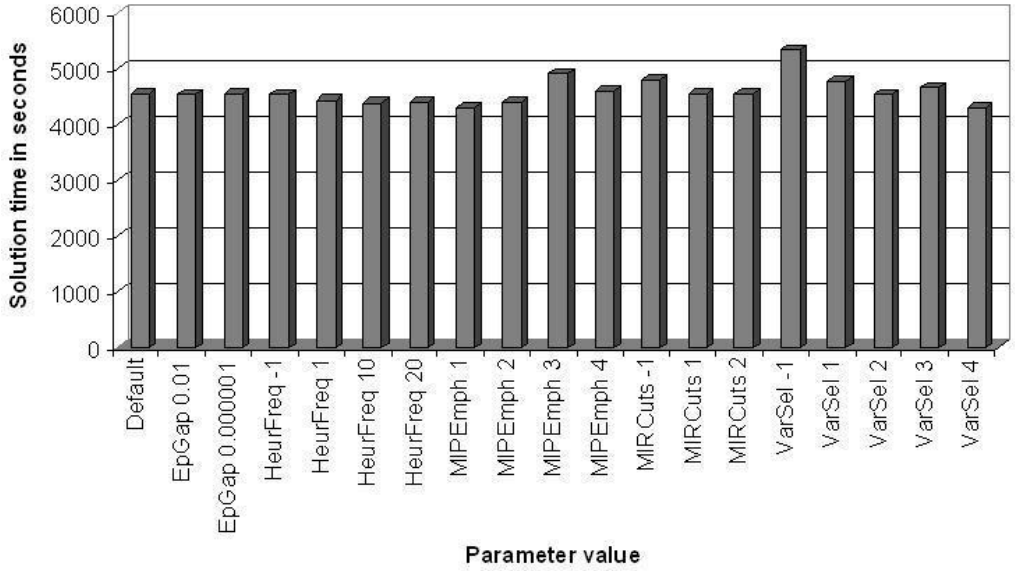

Figure 5.2.: General analysis - average solution time per instance, shown for every parameter setting.

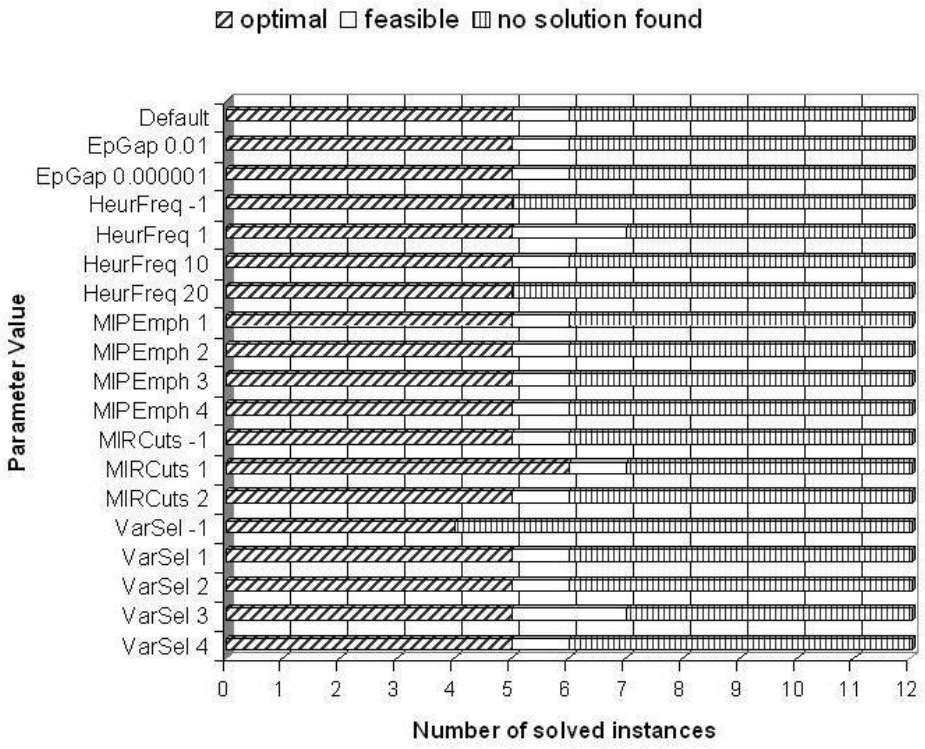

Figure 5.3.: Third analysis - solved instances with upper bounds provided.

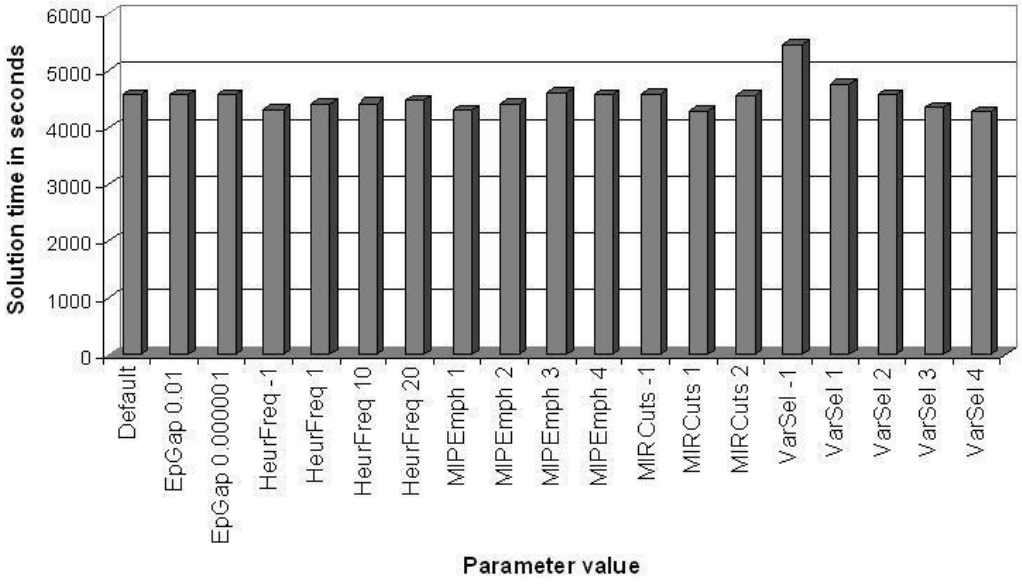

Figure 5.4.: Third analysis - average solution time per instance, shown for every parameter setting

5.5. Comparison to XPRESS

In the last run, a problem matrix for each of the twelve test instances was generated with CPLEX, and were then solved with XPRESS, where all parameters were set to default. These tests were run on an Intel(R) Pentium(R) D 3.20 GHz, with 3 GB RAM, using XPRESS 2007.

Results Results of the last analysis are shown in table 5.13. Six instances were solved to optimality, for four instances a feasible solution was found, and only one instance could not be solved by XPRESS. Solution times compared to CPLEX were a bit slower for smaller instances, but this might be due to the problem matrix that was passed to XPRESS, and the PC the problems were solved with, which is also a bit slower than the one where CPLEX is installed. On the other hand, average solution time per instances was 3967 seconds, as opposed to CPLEX, where (with default settings) average solution time was 4565 sec. per instance.

Instance	Obj.Val.	CPU
a2-16	*294.248	56
a2-20	*344.834	226
a2-24	*431.120	40
a3-18	*300.483	85
a3-24	*344.834	2561
a3-30	495.555	7252
a3-36	583.187	7260
a4-16	*282.677	1165
a4-24	377.553	7257
a4-32	504.323	7252
a4-40	604.194	7246
a4-48		7200

Obj.Val... best solution found, marked with an asterisk * if optimality was proven
CPU... solution time in seconds

Table 5.13.: Results achieved with XPRESS

6. Conclusion

The scope of this work was to examine parameter settings of CPLEX, to find improvements in parameter settings to speed up solution generation, or to improve solutions found, or both. For testing, a model presented by Cordeau [2] was implemented in CPLEX. To test settings, five parameters were chosen, and every value of a parameter was tested on twelve instances, respectively, where all other parameters remained as default.

In sum, default settings of CPLEX provided good results. Variations of parameter *Epsilon Gap* did not affect solutions, therefore this parameter can be neglected. Results for parameter *Heuristic Frequency* varied in the first and third test run, in general the application of the heuristic always led to better results than if it was not applied, therefore it is recommended to leave the parameter as default, or to set it to a value between 1 and 10. To set parameter *MIP Emphasis*, the user has to decide if more feasible solutions are desired, or if CPLEX should only search for the optimal one. For the generation of *MIR Cuts*, two settings can be recommended. One is the default value, the other one is to tell CPLEX to generate cuts moderately. For the last parameter tested, setting $VarSel = 3$ is recommended, as this led to best results generated, though all other settings (including default value), except branching on the variable with minimum infeasibility, led to good results.
The combination of parameters *MIR Cuts* and *Variable Selection* did not lead to much better solutions, therefore no appropriate conclusion can be drawn at this point.

In the last test run, results of CPLEX and XPRESS were compared, where XPRESS had slightly better solution values in terms of objective values and instances solved to optimality or feasibility. On the other hand, CPLEX succeeded in terms of solution time, but this might be due to the slower PC that was used for testing XPRESS. A closer examination of differences between CPLEX and XPRESS might be subject of future work.

A. Final Model

$$\min \sum_{k\in K}\sum_{i\in N}\sum_{j\in N} c_{ij}^k x_{ij}^k \tag{A.1}$$

subject to:

$$\sum_{k\in K}\sum_{j\in N} x_{ij}^k = 1 \qquad \forall i \in P \tag{A.2}$$

$$\sum_{j\in N} x_{ij}^k - \sum_{j\in N} x_{n+i,j}^k = 0 \qquad \forall i \in P, k \in K \tag{A.3}$$

$$\sum_{j\in N} x_{0j}^k = 1 \qquad \forall k \in K \tag{A.4}$$

$$\sum_{j\in N} x_{ji}^k - \sum_{j\in N} x_{ij}^k = 0 \qquad \forall i \in P \cup D, k \in K \tag{A.5}$$

$$\sum_{i\in N} x_{i,2n+1}^k = 1 \qquad \forall k \in K \tag{A.6}$$

$$B_{2n+1}^k - B_0^k \leq T \qquad \forall k \in K \tag{A.7}$$

$$e_i \leq B_i \leq l_i \qquad \forall i \in P \cup D \tag{A.8}$$

$$e_i \leq B_0^k \leq l_i \qquad \forall i \in N, k \in K \tag{A.9}$$

$$e_i \leq B_{2n+1}^k \leq l_i \qquad \forall i \in N, k \in K \tag{A.10}$$

$$t_{i,n+i} \leq L_i \leq L \qquad \forall i \in P \tag{A.11}$$

$$\max\{0, q_i\} \leq Q_i \leq \min\{Q, Q + q_i\} \qquad \forall i \in N, k \in K \tag{A.12}$$

(A.13)

$$B_j \geq (B_0^k + d_0 + t_{0j}) - M_{0j}^k(1 - x_{ij}^k) \qquad \forall j \in P \cup D, k \in K$$

(A.14)

$$B_j \geq (B_i + d_i + t_{ij}) - M_{ij}(1 - \sum_{k\in K} x_{ij}^k) \qquad \forall i, j \in P \cup D \tag{A.15}$$

$$B^k_{2n+1} \geq (B_i + d_i + t_{i,2n+1}) - M^k_{i,2n+1}(1 - x^k_{i,2n+1}) \qquad \forall i \in P \cup D, k \in K \tag{A.16}$$

$$L_i = B_{n+i} - (B_i + d_i) \qquad \forall\ i \in P \tag{A.17}$$

$$Q_j \geq (Q^k_0 + q_j) - W^k_{0j}(1 - x^k_{0j}) \qquad \forall\ j \in P \cup D, k \in K \tag{A.18}$$

$$Q_j \geq Q_i + q_j - W_{ij}(1 - \sum_{k \in K} x^k_{ij}) + (W_{ij} - q_i - q_j) \sum_{k \in K} x^k_{ji} \qquad \forall\ i, j \in P \cup D \tag{A.19}$$

$$Q^k_{2n+1} \geq (Q^k_i + q_{2n+1}) - W^k_{i,2n+1}(1 - x^k_{i,2n+1}) \qquad \forall\ i \in P \cup D \tag{A.20}$$

$$x^k_{ij} \in \{0, 1\} \qquad \forall i, j \in P \cup D, k \in K \tag{A.21}$$

B. Implementation

The following tables show a list of variables used in the model, equivalent names in CPLEX, and their meaning. The source code and one of the test instances are available on the CD enclosed.

Name in model	Name in CPLEX	Meaning
variables		
n	int nbcustomers	number of customers
	int nbvehicles	number of vehicles available
T^k	float maxroutelength	max length of a route in minutes
Q	int maxcapacity	max number of customers in a vehicle
L	float maxridetime	max ride time of a customer between pick-up and drop-off
	float bigNumber=999999	Just a big number
ranges		
N	range nodes=0..2*nbcustomers+1	all vertices including pick-up, drop-off points and 2 depots
K	range vehicles=1..nbvehicles	set of vehicles
$P \cup D$	range pickUpDropOff=1..2*nbcustomers	Pick-up and drop-off nodes (without depots)
P	range pickUp=1..nbcustomers	Pick-up nodes
	range pathRange=0..3	Range for infeasible path array
	range checkRange=1..6	
more variables		
e_i	float earliestStartingTime[nodes]	Earliest time to start service at node i
l_i	float latestStartingTime[nodes]	Latest time to start service at node i
d_i	float serviceDuration[nodes]	Service duration at node i
q_i	int load[nodes]	Load at node i (1 for pickup, -1 for delivery)
Q_0	int Q_0[vehicles]	Load of vehicle k after leaving depot
Q_{2n+1}	int Q_2n1[vehicles]	Load of vehicle k at final depot
M_{ij}^k	float M[nodes][nodes][vehicles]	
W_{ij}^k	float W[nodes][nodes][vehicles]	

Table B.1.: List of variables - Part 1.

Name in model	Name in CPLEX	Meaning
decision variables		
x_{ij}^k	dvar boolean travel[nodes][nodes][vehicles]	1 if arc (i, j) is traversed by vehicle k, 0 otherwise
B_0^k	dvar float+ B_0[vehicles]	time vehicle k starts at depot 0
B_{2n+1}^k	dvar float+ B_2n1[vehicles]	time vehicle k arrives at depot 2n+1
L_i	dvar float+ rideTime[pickUp]	time customer i travels in vehicle k
B_i	dvar float+ startService[pickUpDropOff]	aggregate time variables
Q_i	dvar float+ aggLoad[pickUpDropOff]	aggregate load variables
Variables for infeasible path check		
	float smallest, float largest	auxiliary variable
	int index[pathRange]	array with order of nodes in path
	int infeasible	1 if path is infeasible, 0 otherwise
P_j	float P[pathRange]	Ride time of user j
	float result	result of heuristic infeasiblePath()
	float summe, float mini	auxiliary variable
	float A[pathRange]	Arrival time
B_j	float B[pathRange]	Beginning of service
	float D[pathRange]	Departure time
W_p	float Wait[pathRange]	Waiting time
	float RT[pathRange]	Customer Ride time
F_i	float FTS[pathRange]	Forward time slack at node i
	int Path[checkRange]	Path[i] is 1 if path is infeasible, 0 otherwise
Distancematrix and temporary matrix		
	float distance[nodes][nodes]	Distance matrix
	float distanceTemp[nodes][nodes]	Temporary distance matrix

Table B.2.: List of variables - Part 2.

C. Summary (in German)

Wir leben in einer Gesellschaft, deren Lebenserwartung sowohl bei Männern als auch bei Frauen stetig ansteigt, daher wird es zukünftig viele Menschen geben, die zwar mobil sein wollen oder müssen (Einkäufe, Arztbesuche, etc.), selbst aber nicht mehr in der Lage sind, ein Auto zu fahren. Auch öffentliche Verkehrsmittel bieten in einer solchen Situation oft wenig oder keine Alternative, da sie entweder nicht entsprechend ausgestattet sind, oder kein Personal zur Verfügung steht, das Menschen mit besonderen Bedürfnissen unterstützt. Aus diesem Grund entstanden bereits in den 1970er Jahren sogenannte "dial-a-ride"-Systeme, also Fahrtendienste, die Personen von einem Ort (zB zu Hause) abholen, und an einen gewünschten Ort bringen und später bei Bedarf auch wieder zurück.
Aus Sicht des Operations Management ist das dial-a-ride Problem (DARP) in die Gruppe der Tourenplanungsprobleme einzuordnen. Zur Durchführung dieser Arbeit wurde ein Modell von Cordeau [2] zur exakten Lösung von DARP verwendet. Daher werden im ersten Teil dieser Arbeit Tourenplanungsprobleme vorgestellt und speziell auf das DARP eingegangen, um den theoretischen Hintergrund zu erläutern. Das verwendete Modell sowie dessen implementierung werden detailiert beschrieben.

Inhalt dieser Arbeit ist eine Untersuchung der Parametereinstellung in CPLEX[1] mit dem Ziel, mögliche Verbesserungen aufzuzeigen um den Lösungsprozess zu beschleunigen, oder bessere Lösungen zu generieren, oder sogar beides. Dazu wurde das Modell von Cordeau [2] in CPLEX implementiert und fünf verschiedene Parameter wurden ausgewählt und einzeln getestet. Das DARP-Modell von Cordeau wurde aus zwei Gründen gewählt: Erstens, weil es zum Zeitpunkt der Entscheidung eines der neuesten Modelle war. Zweitens ist bei der Lösung keine Generierung von cuts erforderlich, daher ist dieses Modell besonders gut zur Parameterevaluierung geeignet. Für jeden Parameter gibt es neben der Standardeinstellung mehrere Wahlmöglichkeiten, die vom Benutzer bestimmt werden können. Jede dieser Möglichkeiten wurde einzeln an zwölf Instanzen getestet, während alle anderen

[1] CPLEX ist ein Optimierungstool der Firma ILOG.

Werte in Standardeinstellung belassen wurden. Die Ergebnisse der Testläufe werden hinsichtlich zweier Werte untersucht: Erstens, die Anzahl der gefundenen Lösungen, und Zweitens, wieviel Zeit für die Lösungsfindung benötigt wurde. Generierte Lösungen werden zusätzlich noch als "optimal" oder "zulässig" (feasible) unterschieden. Die maximale Rechenzeit pro Instanz wurde auf zwei Stunden festgelegt, daher ist es möglich, und oft der Fall, dass in dieser Zeit nur eine zulässige, aber nicht optimale, oder gar keine Lösung von CPLEX generiert werden konnte, bevor der Suchlauf abgebrochen wurde. Insgesamt wurden vier Analysen durchgeführt. Beim ersten und dritten Lauf wurden alle fünf verschiedenen Parameter in allen wählbaren Ausprägungen getestet, wobei für die dritte Analyse zusätzlich *upper bounds* gesetzt wurden. Im zweiten Testlauf wurden zwei Parameter, die in der allgemeinen Analyse besonders vielversprechend aussahen, miteinander kombiniert. Für eine vierte Analyse wurde schliesslich für jedes Problem eine Matrix mit CPLEX generiert, die dann mit dem Programm XPRESS[2] gelöst wurde, um einen ersten Vergleich zwischen beiden Produkten zu ziehen.

Zusammenfassend kann gesagt werden, dass die Standardeinstellung der Parameter in CPLEX gute Lösungen bringt. Variationen des Parameters *Epsilon Gap* hatten keine Auswirkung auf die Lösungsfindung, daher kann dieser Parameter vernachlässigt werden. Stärkere Auswirkungen hatte die Änderung des Parameters *Heuristic Frequency*, wobei die Ergebnisse der ersten und dritten Analyse schwankten. Generell kann jedoch gesagt werden, dass eine Anwendung der Heuristik jedenfalls von Vorteil ist. Bei Parameter *MIP Emphasis* muss der Benutzer unterscheiden, welches Ziel verfolgt wird. Es konnten sehr wohl Unterschiede zwischen der Verlagerung des Schwerpunktes der Suche auf optimale oder zulässige Lösungen beobachtet werden. Für Parameter *MIR Cuts* ist die Grundeinstellung gleichzeitig eine der beiden besten Einstellungen, die besten Ergebnisse wurden mit der Einstellung $MIRCuts = 1$ ("generate cuts moderatly") erzielt. Der letzte untersuchte Parameter war schliesslich auch der mit den grössten Unterschieden in den Ergebnissen. Insgesamt ist wohl die Einstellung $VarSel = 3$ diejenige, die zu den besten Ergebnissen führt.
Im Vergleich zwischen CPLEX und XPRESS konnte letzteres Tool etwas bessere Ergebnisse erzielen. Ein detailierter Vergleich der beiden Tools könnte der Inhalt einer weiteren Untersuchung sein.

[2]XPRESS ist ein weiteres Optimierungstool, entwickelt von der Firma Dash Optimization

D. Curriculum vitae of the author

Personal information

Surname, First name	Pfliegl, Elisabeth
E-Mail	e.pfliegl@gmx.at
Nationality	Austria
Date and place of birth	30.08.1982 in Vienna

Education

October 2001 onwards	University of Vienna Faculty of Business, Economics and Statistics 1210 Vienna, Brünnerstr. 72 Field of studies: International business administration Principal subjects: Productionmanagement and logistics, information management
09/2006 - 12/2006	Semester abroad in Paris ESCP-EAP Paris 79, Avenue de la Rpublique 75011 Paris, France
09/1996 - 06/2001	HTL Vienna 22 Department for data processing 1220 Vienna, Donaustadtstr. 45

Mother tongue	German
Other languages	English, French

Bibliography

[1] A.S. Alfa: "Scheduling of vehicles for transportation of elderly", *Transp. Planning and Techn.* Vol. 11, 1986, p. 203-212

[2] J.F. Cordeau: "A Branch-and-Cut Algorithm for the Dial-a-Ride Problem", *Operations Res.* Vol. 54, No. 3, May-June 2006, p. 573-586

[3] J.F. Cordeau, G. Laporte: "The dial-a-ride problem: Variants, modeling issues, and algorithms", *4OR* 1, 2003, p. 89-101

[4] J.F. Cordeau, G. Laporte: "A tabu search heuristic for the static multi-vehicle dial-a-ride problem", *Transportation Res.* Part B 37 2003, p. 579-594

[5] J.F. Cordeau, G. Laporte: "The dial-a-ride problem: models and algorithms", *Ann. Operations Res.* Vol. 153, 2007, p. 29-46

[6] Y. Dumas, J. Desrosiers, F. Soumis: "The pickup and delivery problem with time windows", *Europ. J. Operations Res.* 54, 1991, p. 7-22

[7] M. Gendreau, G. Laporte, F. Semet: "A dynamic model and parallel tabu search algorithm for real-time ambulance relocation", *Parallel Comp.* 27, 2001, p. 1641 - 1653

[8] S.C. Ho, D. Haugland: "Local search heuristics for the dial-a-ride problem", Nov. 2004 , Technical Report 286, 2004, University of Bergen

[9] B. Hunsaker, M.W.P. Savelsbergh: "Efficient feasibility testing for dial-a-ride problems", *Operations Res. Lett.* 30, 2002, p. 169-173

[10] I. Ioachim, J. Desrosiers, Y. Dumas, M.M. Solomon, D. Villeneuve: "A Request Clustering Algorithm for Door-to-Door Handicapped Transportation", *Transportation Sci.* Vol. 29, No. 1, 1995

[11] S. Parragh, K.F. Doerner, R.F. Hartl: "A survey on pickupp and delivery models, Part II", *Journal fuer Betriebswirtschaft*, to appear

[12] S. Ropke, J.-F. Cordeau, G. Laporte: "Models and branch-and-cut algorithms for pickup and delivery problems with time windows", *Networks* 49, 2007, p. 258-272

[13] M.W.P. Savelsbergh: "The vehicle routing problem with time windows: Minimizing route duration", *ORSA J. Comp.* 4, 1992, p. 285-305

[14] M.W.P. Savelsbergh, M. Sol: "The General Pickup and Delivery Problem", *Transportation Sci.* Vol. 29, No. 1, February 1995

[15] T. Sexton, L.D. Bodin: "Optimizing single vehicle many-to-many operations with desired delivery times: I. scheduling.", *Transportation Sci.* 19, 1985, p.378-410

[16] T. Sexton, L.D. Bodin: "Optimizing single vehicle many-to-many operations with desired delivery times: II. routing.", *Transportation Sci.* 19, 1985, p.378-410

[17] P. Toth, D. Vigo: "The Vehicle Routing Problem", siam, 2002