The Dynamic Dial-a-Ride Problem with Time Windows in a Competitive Multi-Company Environment

Master's Thesis

Ferdi Grootenboers

The Dynamic Dial-a-Ride Problem with Time Windows in a Competitive Multi-Company Environment

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Ferdi Grootenboers
born in Ridderkerk, the Netherlands

Algorithmics Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
`www.ewi.tudelft.nl`

The Dynamic Dial-a-Ride Problem with Time Windows in a Competitive Multi-Company Environment

Author:	Ferdi Grootenboers
Student id:	1149911
Email:	`fgrootenboers@gmail.com`

Abstract

Door-to-door transportation for elderly and disabled people is for many governments an important instrument to increase the mobility of this group of people. Many issues arise in the implementation of such a system, which is often modeled as the Dynamic Dial-a-Ride System with Time Windows (DDARPTW). One of those issues is that taxi companies try to maximize their profit by combining as many rides as possible. This leads to longer travel times, a measure that is expressed in the service quality of a ride. Our main contribution is a system in which multiple companies compete on service quality to increase the average service quality of the rides. We use an auction mechanism to assign rides to companies and an on-line optimization technique to insert assigned rides into current schedules. To determine an offer for announced requests, we allow companies to use knowledge about the distribution of future requests by the use of a Monte Carlo simulation.

Preface

The completion of this thesis would not have been possible without the assistance of my supervisors Mathijs de Weerdt and Mahdi Zargayouna. I would like to thank you for the continuous support you provided and to motivate me to perform the research that I have done. The meetings I have had with both of you have been inspiring and very helpful.

Thank you Mathijs for introducing me to the field of mechanism design and auctions in the seminar Cooperative Agent-Based Systems. You motivated me to perform further research in this field and pointed me to the subject of door-to-door transportation for elderly and disabled people.

Although you left us in August, I would like to thank you too, Mahdi, for the time you have spent to help me with certain issues and to inspire me to keep on going. The fact that you even gave me feedback on certain chapters while you were back in Paris shows your great interaction with the project.

I would like to thank Cees Witteveen and Koen Hindriks for taking the time to read this thesis and taking place in my committee.

I would like to thank my fellow students with which I have worked together during my study. Thanks also go to my friends and my brother, which have always been there for me to relax, to play cards, to play darts, to dance, or to run. I cannot forget the boys of my soccer team, who gave me some great Saturdays when we won another game.

Ruby, sorry for sitting all that days and evenings behind my desk instead of spending more time with you. Thanks for keep supporting me through the last stages of my study.

Last, but not least, I thank my parents for giving me the opportunity to do this study, and for supporting me all that time.

Ferdi Grootenboers
Delft, the Netherlands
December 17, 2009

Contents

List of Figures

Chapter 1

Introduction

The service quality in transportation systems for elderly and disabled people is often more or less ignored. In this thesis we provide a mechanism for the allocation of transportation requests and the construction of a schedule for the serving of these requests that is designed from a customer's point of view, instead of from a company's point of view.

1.1 Problem statement

In the Netherlands, elderly and disabled people rely on so-called *door-to-door transportation* to travel. Often, they cannot use public transport services due to inconvenient vehicles, or just because the nearest bus stop or train station is too far away. Door-to-door transportation is used for various kinds of trips, from a trip to the zoo to a, often more necessary, trip to the hospital. Customers can request a ride by calling a taxi agency which has the exclusive rights to serve requests in that certain area. This agency then tells them what time they will be picked up.

Before a taxi agency can announce a pickup time for the request, it has to update its schedule by inserting the new request. Different vehicles might be available to serve the request, and the agency must search for the best way to insert this request. This problem is known as the *Dynamic Dial-a-Ride Problem with Time Windows* (DDARPTW). The problem is *dynamic*, because new requests are become known during the planning period, as opposed to a *static* setting, in which all requests are known beforehand. The definition of "the best way to insert a new request" can differ, but commonly used measures are total route costs, total route duration, and the total number of vehicles used. Intuitively, a company tries to maximize its profit (i.e. maximize its income and minimize its costs).

The biggest problem with the current approach used in the Netherlands, is the unsatisfactory service quality. There are many complains of customers about drivers that take big detours, drivers that behave inappropriately, and late pickup times. High costs are not directly a problem for the customers, because these are often covered by insurance companies or the government to increase the mobility of the elderly and disabled people.

We think that one of the reasons for this unsatisfactory service quality is the fact that there is only one company that has the rights to serve requests. There is no direct incentive to serve requests with a high quality, because profit will not increase at all. It is, of course, possible to give a company punishments for not serving requests with high quality, but this has some important disadvantages. When punishments are too low, companies can take these penalties for granted if they can still make profit, and when punishments are too high, companies have less money to spent on increasing the quality. It is also possible to dissolve the contract with the transportation company if this company does not meet specified service quality, but the disadvantage of this solution is that a new procurement auction has to be started, which will take even more time and money.

Our idea is to use multiple companies that serve requests in the same area, to add an element of competition between companies. Competition may lead to incentives to increase the service quality of requests. The question that we try to answer in this thesis is the following:

"Can the service quality of Dynamic Dial-a-Ride systems for the transportation of elderly and disabled people be improved using a competitive multi-company setting?"

In the next section we describe how we want to move the DDARPTW to a multi-company environment and what consequences this has for solving the DDARPTW.

1.2 Moving to a multi-company environment

In a single-company setting there is only one company that serves all requests in a specific area. All incoming requests are assigned to that company and the company itself must solve the problem of assigning the requests to specific vehicles, in order to serve them. Although local authorities can give penalties to a company for not meeting a certain quality level, the company does not have to worry too much about optimizing service quality, because it is sure that it will get assigned future requests. It just wants to optimize its total profit, so if a penalty is worth it, the company will take it.

By moving to a multi-company setting we allow multiple companies to serve requests in a specific area. Incoming requests must first be assigned to the company that can serve the request the best, and this company then has to assign the request to one of the vehicles it owns. The problem of deciding which company can serve the request the best, is solved by the customer (or its representing government agency). It can choose for example the cheapest company, the company that provides the highest quality, or the company with the highest reputation.

In this multi-company setting, companies indirectly work together to generate a solution for the assignment of requests to vehicles. We say that they are playing a game, with the assignment of all requests as outcome. The field of *game theory* studies the outcomes of games that are the result of the (self-interested) behavior of the players of the game. Game theory is closely related to the field of *mechanism design*, in which the rules of the game are designed to obtain preferable outcomes. What we want are outcomes with a high average service quality and low average costs.

In game theory, the players of the game are often called agents. In our setting, both customers and companies are agents. All of our players are self-interested, meaning that they try to maximize their own utility. Customer agents try to maximize the service quality of their request, and company agents try to maximize their profit. Intuitively, these two measures are opposites of each other. With higher costs (i.e. lower profit) the quality can be increased. In a single-company setting, the single company does not have the incentive to make higher costs for better quality. By using a multi-company setting, we try to design the rules of the game such that this incentive does exist.

In the next section we state our contributions in the context of prior work.

1.3 Prior work and our contributions

The basic problem in our dynamic setting is to assign incoming requests to vehicles and to develop routes for these vehicles to serve the assigned requests as efficient as possible. As mentioned earlier, this problem is known as the Dynamic Dial-a-Ride Problem with Time Windows (DDARPTW).

Requests dynamically enter the system (e.g. customers call a taxi agency) and these requests have to be assigned to vehicles with capacity to transport the passengers. Routes for the vehicles are implicitly defined by the start and end points of the requests a vehicle has been assigned to. A request also defines certain time windows for the pickup and delivery of the passengers, which decrease the flexibility to schedule these requests.

We notice that in almost all work done involving the DDARPTW or similar problems, the authors assume a cooperative environment, and that objective functions mainly includes the minimization of costs and route duration. In this thesis we assume a competitive environment, in which multiple companies compete for the rights to serve incoming requests. Our main contribution is a mechanism to assign vehicles of different companies to incoming requests, based on both service quality and costs. The proposed mechanism consists of two phases; an assignment phase, and a scheduling phase. In the assignment phase, companies make an offer which they announce to the customer. The customer chooses the best offer and the associated company has to serve the request according to the conditions defined in the second-best offer. In the scheduling phase, the winning company actually inserts the new request into its current schedule, conditioned on the constraints in the second-best offer.

The DDARPTW is NP-hard, even without the time windows, dynamic requests, and multiple vehicles. Without these extra constraints, the problem would be equal to the Traveling Salesman Problem (TSP), which is known to be NP-hard [21]. There exist some mathematical models for the DDARPTW that are defined as Mixed Integer Programs, such that they can be used to solve the problem exactly [5, 36]. Because the DDARPTW is NP-hard, using these algorithms in a dynamic context, would require too much computation time; the algorithm needs to be executed every time a new request comes in.

In order to rapidly derive a near-optimal solution to the DDARPTW, heuristics are designed which have a good empirical performance. One of the first heuristics for the static variant of the DDARPTW is proposed by Jaw et al. [17]. Strongly based on the work of Jaw et al. but in a dynamic setting is an approach by Madsen et al. [25], who propose an insertion heuristic called REBUS. They have solved a real-life problem in the city of Copenhagen involving transportation for elderly and disabled people. New requests are inserted taking into account their difficulty of insertion into an existing route. This *difficulty* is a combination value of different measures.

The disadvantage of using a heuristic is that it provides good solutions in most cases, but that it can provide very worse solutions in other cases. By slightly modifying the mathematical model in order to reduce the computation time, we make it possible to use an exact algorithm in a dynamic context (called *on-line optimization*). This approach is used to schedule assigned requests in the scheduling phase of the algorithm. However, we do use a simple insertion heuristic for situations that require less accurate solutions. We extend an approach by Solomon [40] to be able to check service quality constraints and to check feasibility of both pickup and delivery nodes, instead of only delivery nodes.

Instead of using centralized approaches, where a single authority is used to assign requests to vehicles, involved agents can negotiate among themselves about the assignment of requests. It is shown by Mes et al. that a properly designed multi-agent approach performs as good as or even better than traditional methods [28]. The basic concept of such a system is that every vehicle calculates the costs needed to serve an incoming request and proposes an offer to the customer, in an auction.

Most research in the transportation field deals with problems in a cooperative setting, where companies or vehicles are working together to obtain a common goal (e.g. the minimization of the total costs). In that case, the vehicle agents do not care whether they get assigned a request or not, if the assignment helps to optimize the common goal. This situation changes when the vehicle agents are competitive and self-interested (i.e. each has their own goal). There is uncertainty about future requests and vehicles have to deal with this when they want to maximize their profit. Mes et al. [27] and Figliozzi et al. [11] introduced one-step look-ahead capability for these vehicles, with which they incorporate the possible profit of the next incoming request into the price of the current request that is auctioned. These extra costs can be seen as opportunity costs that have to be made for the requests possibly forgone.

We propose a way of even further look-ahead capability by incorporating knowledge about the future request distribution into the calculation of current offers. This is where we use the modified insertion heuristic to perform a Monte Carlo simulation to estimate the expected profit of a request.

Opposed to approaches in prior work, the offers that are announced in our auction do not contain money elements, but contain a proposed service quality for the request. A money element (expected profit) is taken into account in the calculation of the service quality offer, but that is not announced to the auctioneer. Although there is no negotiation about the price, but about the service quality, costs for the serving of the requests need to be determined. A lower and an upper bound are presented for the price per kilometer a customer has to pay for its ride.

Summarizing, our contributions described in this thesis are:

- a mechanism to solve the Dynamic Dial-a-Ride Problem with Time Windows in which companies compete on service quality for the assignment of transportation requests,
- an extension to a mathematical model for the DARPTW by adding a constraint to preserve service quality,
- adding constraints to a Mixed Integer Program based on this mathematical model to compute exact solutions in an on-line optimization approach,
- an extension to a basic insertion heuristic to be able to use it in a dial-a-ride context and preserve service quality,
- define upper and lower bounds for the price per kilometer to be used in a setting with fixed customer costs, and
- an approach to use Monte Carlo simulations to calculate expected profit for future incoming requests to be used to incorporate knowledge about the future in a current auction.

1.4 Outline

This thesis is structured as follows. We start by presenting a mathematical model of the DARPTW in Chapter 2 and present both centralized and decentralized solution concepts for it. In Chapter 3 some concepts of mechanism design are discussed, and sequential auctions are introduced. Our main contributions are presented in Chapter 4 and Chapter 5. Chapter 4 is devoted to moving the DDARPTW to a competitive multi-company environment, and in Chapter 5 we present the results of some experiments we did. Finally, our conclusions and pointers to future work are discussed in Chapter 6.

Chapter 2

The Dial-a-Ride Problem with Time Windows

In this chapter the Dial-a-Ride Problem with Time Windows (DARPTW) is introduced, which is often used to model door-to-door transportation services for elderly and disabled people. The problem to solve is how to allocate incoming transportation requests from customers to vehicles that can serve these requests, and how to design a route for these vehicles to serve multiple requests. Different approaches to solve this problem exists, which can be categorized into centralized and decentralized methods.

Before we discuss multiple solution concepts, we introduce some basic elements and assumptions of the DARPTW in Section 2.1, together with a mathematical model that we use later on in this thesis. In Section 2.2 some approaches to solve the problem centrally, with a single authority, are treated. Opposed to centralized approaches, in decentralized approaches there is no single authority that has full information knowledge and full control of the solution process. Some important concepts of decentralized approaches are dealt with in Section 2.3.

2.1 The problem

The DARPTW is a generalization of the well-known Traveling Salesman Problem (TSP), in which an optimal route has to be found between different cities for minimum costs. The DARPTW adds multiple constraints to this problem like time windows, capacity constraints, and maximum travel time. In Section 2.1.1 these constraints and some other assumptions are discussed. All the constraints come together in the integer programming model that is treated in Section 2.1.2. This model is used in an optimization procedure in the mechanism we propose in Chapter 4.

2.1.1 Elements of the DARPTW

The DARPTW is the problem of how to design routes between multiple locations at which passengers need to be served (i.e. get in or get out of the vehicle). The possible routes are conditioned on several constraints that are either be stated by the transportation request (e.g. time windows and pickup locations) or need to be satisfied in general (e.g. route duration, number of vehicles). We follow an overview paper by Cordeau and Laporte [6] to state some specific features of the problem.

The most important difference between the DARPTW and similar routing problems, like the Pickup and Delivery Vehicle Routing Problem (PDVRP) and the Vehicle Routing Problem with Time Windows (VRPTW), is the human perspective. In the DARPTW a balance has to be made between user convenience and, for example, minimizing routing costs. The user convenience is often measured by the average waiting time or by the actual traveling time relative to a certain base time.

The transportation requests are made by a number of customers that wish to travel between a pickup and a delivery location. It is assumed that customers can impose a time window which includes the earliest possible time and the latest possible time they can be either picked up or delivered at a specific location. Most customers request both an *outbound* ride, which is a ride from home to a remote location, and an *inbound* ride, which is a ride from a remote location back home. In most models it is assumed that customers specify a time window for the arrival time of their outbound ride, and a time window for the departure time of their inbound ride.

Another difference with similar problems is that the capacity of a vehicle is often constraining, meaning that only a few passengers can be served at a time by a single vehicle. For example, the PDVRP is often used to model courier systems for small packages, in which dozens of packages can be transported at the same time by a single vehicle. This is opposed to DARPTW, in which normal taxis can only take three or four passengers at a time.

We assume that there exist multiple vehicles, but that they do not need to have the same capacity. It is allowed to serve passengers of different requests the same time (i.e. to combine requests), but a request must be served by a single vehicle. All vehicles start and end their route at a specific vehicle depot location.

The DARPTW can now be stated as to design efficient routes and schedules for the multiple vehicles to serve all known requests with respect to the constraints described above. The efficiency of the solution to the problem can be measured in different terms like the routing costs needed to serve the requests, the duration of the routes, or the actual time needed to serve individual requests relative to the time needed if these requests were served directly (i.e. no combinations of requests).

In the next subsection a mathematical model for the DARPTW is presented, which allows us to study the features of this problem in a formal way.

2.1.2 Model

The model that is presented here, contains all the features that are discussed above, and can be used to either study the problem in a formal way or to use as an input for a Mixed Integer Program (MIP) solver to solve the problem. Because we want to incorporate all the features described above, we use the approach of Cordeau [5].

The DARPTW can be represented as a directed graph of locations and rides between these locations, $G = (L, R)$. The set of locations L contains two vehicle depots that serve as start and end vertex of all vehicles (denoted by l_0 and l_{2n+1}), n pickup locations $P = \{l_1, \dots, l_n\}$, and n delivery locations $D = \{l_{n+1}, \dots, l_{2n}\}$. This implies that L is partitioned as $L = \{\{l_0, l_{2n+1}\}, P, D\}$, with $|L| = 2n + 2$.

A request is a combination of a pickup location $l_i \in P$ and a delivery location $l_{n+i} \in D$ and can be seen as an announcement from a customer that it wants to travel from location l_i to location l_{n+i}.

Each location l_i has an associated load q_i, which denotes the number of passengers that is pickup up or delivered at that location. We define $q_0 = q_{2n+1} = 0$, $q \geq 0$ for $i = 1, \dots, n$, and $q_i = -q_{i-n}$ for $i = n+1, \dots, 2n$. To account for service time at locations (i.e. time to get in and out the vehicle), we associate with every location l_i a service duration $d_i \geq 0$ and $d_0 = d_{2n+1} = 0$.

The set of rides is defined as $R = \{(i, j) : i = 0 \text{ and } j \in P, \text{ or } i, j \in P \cup D \text{ and } i \neq j \text{ and } i \neq n + j, \text{ or } i \in D \text{ and } j = 2n + 1\}$, so this set contains all possible connections between any two locations that a vehicle can actually drive. This means that a ride is of one of the following six types:

- start at the start depot and end at a pickup location,
- start at a pickup location and end at a another pickup location,
- start at a pickup location and end at a another delivery location,
- start at a delivery location and end at another delivery location,
- start at a delivery location and end at another pickup location, or
- start at a delivery location and end at the end depot.

Note that we make a distinction between requests and rides here. A request can be served directly by a single ride from the pickup location of the request to the delivery location, or it can be served by a sequence of rides that eventually ends at the delivery location of the request. So a request is a combination of a pickup location and a delivery location defined by the customer, and a ride could be any combination of two locations conditioned on the properties above.

The set of vehicles is denoted by K, and with each vehicle $k \in K$ a maximal capacity Q_k and a maximal route duration T_k are associated. The cost of a ride from location l_i to l_j with vehicle k is denoted by c_{ij}^k, and the travel time of a ride between l_i and l_j is denoted by t_{ij}.

To incorporate time elements into the model, we define a planning horizon H, which is the time period for which the routes are planned (e.g. an hour, a day, or a week). For example, if the time period for which we schedule requests is a whole day, we start at time 0 and end at time 1440. Note that time variables are measured in minutes. Time windows (inbound as well as outbound) are associated with a location l_i as $[s_i, e_i]$, with s_i the earliest possible time the request can be served at that location (either pickup up or delivery), and e_i the latest possible time the request can be served.

Further, let u_i^k be the time at which vehicle k starts servicing at a location l_i, w_i^k the load of vehicle k upon leaving location l_i, and r_i^k the ride time of a customer that places the request to travel from l_i to l_{n+i}. In the model that follows, x_{ij}^k is equal to 1 if and only if a ride from location l_i to l_j is allocated to vehicle k.

In Figure 2.1 the entire model is given, in which the objective function is to minimize total routing costs (see Equation 2.1). It should be obvious that many other objective functions can be stated.

We explain all the constraints denoted in the model above. Constraint 2.2 ensures that all requests are served only once and Constraint 2.3 ensures that every vehicle will once drive to the start and end depot. Together with 2.4 and 2.5 this guarantees that every request is served once by the same vehicle and that each vehicle starts and ends its route at a depot.

Constraint 2.6 says that the arrival time at a location must be higher or equal to the start time of servicing at the starting location, plus the service duration at that location, plus the time of the ride from start to end location. It is also obvious that the load of a vehicle at the end location of a ride is higher than or equals the load of that vehicle at the start location (Constraint 2.7).

The travel time of a user is higher than or equals the time the vehicle is at his delivery location minus the time he picked up the user and minus the duration of servicing at the pickup location; this is ensured by Constraint 2.8. Constraint 2.9 says that the duration of a vehicle to drive from the start depot to the end depot must be less than or equal to the total route duration specified for that vehicle, while Constraint 2.10 ensures that every location is visited within the specified time horizons. The ride time of a passenger is specified to be as least as great as the time of a direct ride between its pickup and delivery location and as most as great as the planning horizon (Constraint 2.11).

The load of a vehicle can never be higher than the highest possible load specified. Note that the load of a vehicle increases at a pickup location and decreases at a delivery location, so we can state Constraint 2.12. The last constraint ensures that we deal with binary variables, so that a variable de-

Minimize

$$\sum_{k\in K}\sum_{i\in L}\sum_{j\in L} c_{ij}^k x_{ij}^k \tag{2.1}$$

subject to

$$\begin{aligned}
&\sum_{k\in K}\sum_{j\in L} x_{ij}^k = 1 && (i\in P), && (2.2)\\
&\sum_{i\in L} x_{0i}^k = \sum_{i\in L} x_{i,2n+1}^k = 1 && (k\in K), && (2.3)\\
&\sum_{j\in L} x_{ij}^k - \sum_{j\in L} x_{n+i,j}^k = 0 && (i\in P,\ k\in K), && (2.4)\\
&\sum_{j\in L} x_{ji}^k - \sum_{j\in L} x_{ij}^k = 0 && (i\in P\cup D,\ k\in K), && (2.5)\\
&u_j^k \geq (u_i^k + d_i + t_{ij})x_{ij}^k && (i,\ j\in L,\ k\in K), && (2.6)\\
&w_j^k \geq (w_i^k + q_j)x_{ij}^k && (i,\ j\in L,\ k\in K), && (2.7)\\
&r_i^k \geq u_{n+i}^k - (u_i^k + d_i) && (i\in P,\ k\in K), && (2.8)\\
&u_{2n+1}^k - u_0^k \leq T_k && (k\in K), && (2.9)\\
&s_i \leq u_i^k \leq e_i && (i\in L,\ k\in K), && (2.10)\\
&t_{i,n+i} \leq r_i^k \leq H && (i\in P,\ k\in K), && (2.11)\\
&\max\{0,q_i\} \leq w_i^k \leq \min\{Q_k, Q_k+q_i\} && (i\in L,\ k\in K), && (2.12)\\
&x_{ij}^k = 0 \text{ or } 1 && (i,\ j\in L,\ k\in K). && (2.13)
\end{aligned}$$

Figure 2.1: Mixed Integer Program formulation to allocate requests to vehicles within a company. The objective function minimizes costs.

notes whether a ride is allocated to a vehicle or that it is not, and we cannot specify that half of that ride is allocated to another vehicle.

2.1.3 Static versus dynamic

A DARPTW can be either *static* or *dynamic*. In the static variant, all requests are known beforehand and routes are designed before the first request is served. In the dynamic variant, which is often denoted by the Dynamic Dial-a-Ride Problem with Time Windows (DDARPTW), requests become known during execution of the mechanism and these requests are dispatched to vehicles in an on-going fashion.

A problem is often not purely dynamic, because a part of the requests is known in advance and the other part is gradually revealed. In a study by

Lund et al. [24] the notion of the *degree of dynamism* (*dod*) for a problem is introduced. This most basic measure of dynamism is defined as the number of dynamic requests in proportion to the total number of requests. Let the number of requests that is revealed dynamically be denoted by n_{dyn}, and the number of requests that is known in advance by n_{adv}. The total number of requests, n_{tot}, is therefore $n_{dyn} + n_{adv}$, and the degree of dynamism is defined as:

$$dod = \frac{n_{dyn}}{n_{tot}}. \tag{2.14}$$

A purely static system has a *dod* of 0 and a purely dynamic system has a *dod* of 1.

Systems can be categorized by their degree of dynamism into one of three groups. Systems with a *dod* below $0.2 - 0.3$ are said to be *weakly dynamic* and systems with a *dod* above $0.8 - 0.9$ are said to be *strongly dynamic*. The other systems are said to be *moderately dynamic*.

It is obvious that a dispatcher would prefer requests coming in early in the planning horizon to requests coming in late, because late requests need to be planned much faster (i.e. they have a shorter reaction time). To make a better distinction between problems, the *effective degree of dynamism* (*edod*) is introduced [23], which represents the average of how late the dynamic requests are received compared to the latest possible time these requests could be received. Consider a planning horizon that starts at time 0 and ends at time T, and denote by t_i the time at which dynamic request i is received ($0 < t_i \leq T$). Note that requests that are known in advance have $t = 0$. The effective degree of dynamism can now be defined as:

$$edod = \frac{\sum_{i=1}^{n_{dyn}} \left(\frac{t_i}{T}\right)}{n_{tot}}. \tag{2.15}$$

Again, a purely static system has an *edod* of 0 and a purely dynamic system has an *edod* of 1. Further note that the *edod* approaches 1, the later requests enter the system:

$$\lim_{t_i \to T \forall i} edod = 1. \tag{2.16}$$

A request often defines a time window for a pickup or delivery location i, $[s_i, e_i]$, with s_i the earliest time the request can be served, and e_i the latest time the request can be served. The *reaction time* is defined as the period between the arrival of the request t_i and the latest possible service time e_i, $e_i - t_i$. Now, the measure of effective degree of dynamism accounting for reaction time can be defined as [23]:

$$edod_{tw} = \frac{1}{n_{tot}} \sum_{i=1}^{n_{tot}} \left(\frac{T - (e_i - t_i)}{T}\right) \tag{2.17}$$

The degree of dynamism is important in the process of choosing an algorithm to solve the problem. Experimental results show that there is a significant difference in performance of algorithms applied to problems with a different degree of dynamism [23]. In the next section we take a look a centralized solution concepts that can be used to solve either static or dynamic variants of the problem.

2.2 Centralized solution concepts

In this section we discuss solution concepts in which a central authority has full information knowledge and full control of the solution process. Known algorithms can be roughly categorized into three groups: *simple policies*, *heuristics* and *exact algorithms*. Simple policies can serve as a base for developing more complex heuristics. Exact algorithms often need too much computation to use them in a dynamic environment, but they can be used as a benchmark for simple policies and heuristics. The DARPTW is often weakly to moderately dynamic and heuristics seem to perform better than simple policies in weakly dynamic systems [23].

Simple policies are rules used to dispatch individual requests to vehicles, such as First Come First Served (FCFS) and Nearest Neighbor (NN). We do not discuss these policies in this thesis, because they are not used as stand alone algorithms, but are often part of a preparation step for an exact algorithm or a heuristic.

Exact algorithms are treated first, because these algorithms give us the best answers. The problem with these algorithms is their time complexity, which is often exponential in the number of transportation requests. Heuristics have a good empirical performance and can be used to rapidly derive a near-optimal solution. Finally, we describe a concept called on-line optimization, in which exact algorithms are used in a dynamic environment.

2.2.1 Exact algorithms

The model that we presented in Section 2.1.2 allows us to solve the problem using exact algorithms to obtain an optimal solution. Due to its nature, such an algorithm needs computation time that is exponential in the number of transportation requests that has to be served. It can, however, be used as a benchmark to which simple policies and heuristics can be tested and can give insight into various operational trade-offs in dynamic problems. This can form the basis of optimization-based approaches [45].

Integer Linear Programming

The objective function can be optimized using Integer Linear Programming (ILP) packages like CPLEX[1], SCIP[2] and GLPK[3]. Areas of refinement for this kind of methods include the way cost is measured in assigning requests to vehicles and in the computational procedures to perform the algorithm. The first is exploited by comparing three different criteria to assign requests to vehicles, namely to minimize the total costs, the average costs, or the marginal costs of a vehicle [45]. The latter is exploited by adding additional constraints to the program [5, 36] and we briefly discuss this method, known as a branch-and-cut algorithm.

A branch-and-cut algorithm first solves the ILP without the integer constraints. This can result in integer variables being assigned fractional values. An algorithm is used to find additional linear constraints that are satisfied by all feasible integer points, but violated by the current fractional solution, i.e. the search space is reduced around the integer solution. The idea behind this method is that by repeatedly running the algorithm with additional constraints, the (fractional) solution is closer to the optimal integer solution than in the previous run.

When no more additional constraints can be added to the program, a branch-and-bound algorithm is used, which separates the program into two parts. One with a constraint in which the fractional variable is less than or equal to the previous integer value, and the other with a constraint in which the fractional variable is greater than or equal to the next integer value. This process is repeated until a solution is found that satisfies all integer constraints.

For the DARPTW, several additional constraints (called *valid inequalities*) can be derived during pre-processing [5, 36]. Time windows can be tightened, infeasible arcs can be eliminated and variables can be fixed. The separation of the problem during executing the branch-and-bound algorithm can also be led by certain heuristics which can identify invalid inequalities. Results are known for instances with 16 to 36 [5] and 16 to 96 [36] requests.

Other exact algorithms

There exist some other exact algorithms that are not based on Integer Linear Programming. We briefly mention a few interesting ones here.

An interesting approach is to assign one of three statuses (*request_waiting*, *request_being_served*, and *request_completed*) to each known request and then derive a graph containing all possible ways status changes can occur [9]. This graph contains status vectors as nodes and status transitions as arcs. The graph is reduced by eliminating duplicate status vectors that also appeared at

[1]`http://www.ilog.com/products/cplex/`

[2]`http://scip.zib.de/`

[3]`http://www.gnu.org/software/glpk/`

the same location. Once the graph cannot be further reduced, a shortest path algorithm is applied to derive routes trough the pickup and delivery location of the requests.

Several exact algorithms for vehicle routing problems are based on one of the following four formulations:

- arc formulation,
- arc-node formulation,
- spanning tree formulation,
- path formulation.

Because these algorithms are not especially developed for the DARPTW, we refer the interested reader to a survey by Kallehauge [20].

2.2.2 Insertion heuristics

In order to rapidly derive a near-optimal solution to the DARPTW, heuristics can be used which have a good empirical performance. A difference can be made between *insertion heuristics* and *metaheuristics*, but in this thesis we only consider insertion heuristics. The reason for this is that metaheuristics are not especially focused on the specific features of the kind of problem, and that these heuristics are hardly ever used for dynamic problems, because they tend to take more computation time.

Insertion heuristics insert new requests into the current routes at the best possible position known. The challenge is not only to insert a request at a position that is best for this request, but also leaves enough slack time for possible future requests. These future requests can be either known beforehand (i.e. in a static problem), which implies that there is already some information about these requests, or these requests can still be unknown (i.e. in a dynamic problem), which implies that there is no information yet. Because we want to focus on dynamic problems in this thesis, we mainly discuss heuristics for dynamic problems here.

Basic insertion heuristic

Most of the insertion heuristics for dynamic problems are based on heuristics for static problems, of which one of the first algorithms was proposed by Jaw et al. [17]. In this algorithm for the DARPTW a non-linear objective function combining several types of disutility is used. The heuristic selects requests in order of earliest feasible pickup time and gradually inserts them into vehicle routes so as to yield the least possible increase of the objective function. Note that for dynamic problems, this pre-processing of creating an insertion order is not possible, because the order is determined by the arrival of the requests.

Algorithm 2.1: Basic insertion algorithm

```
input  : The set of current vehicle schedules $S$, the set of requests that
         have to be inserted $R$
output: Multiple schedules in which all unassigned requests $r \in R$ are
         inserted
forall $r \in R$ do
    $c* \leftarrow -\infty$
    forall $s \in S$ do
        forall $i \in s$ do
            forall $j \in s$ where $j \geq i$ do
                if Feasible($r$, $i$, $j$) and Cost($r$, $i$, $j$) $< c*$ then
                    $s* \leftarrow s$
                    $i* \leftarrow i$
                    $j* \leftarrow j$
                    $c* \leftarrow$ Cost($r$, $i$, $j$)
                end
            end
        end
    end
    Insert($r$, $s*$, $i*$, $j*$)
end
```

Once a request is selected to be inserted into one of the vehicles routes two steps are executed: a *feasibility* step and an *optimization* step. During the feasibility step, all feasible ways to insert the pickup location and the delivery location of the request into the routes of one of the vehicles are investigated. An insertion is feasible if none of the constraints for the current request and for the requests already inserted, are violated. In the algorithm proposed by Jaw et al. [17] these constraints consist of time window constraints, capacity constraints, service time constraints, and service quality constraints. These service quality constraints guarantee that the ride times of customers will not exceed a pre-specified maximum and that the time of pickup or delivery will not deviate from the desired pickup or delivery time more than a pre-specified maximum.

In the feasibility step, an additional cost variable is maintained for each feasible insertion, to support the optimization step. When all vehicles are examined for feasible insertions, the request is inserted into the schedule with minimal additional costs. Notice that this cost variable can denote various measures or combinations of measures. For example, it can denote monetary costs needed to insert the request, it can denote a measure of additional route duration, or it can denote some other measure of disutility.

In Algorithm 2.1 a basic insertion algorithm [2] is shown, in which both the pickup and the delivery location are inserted into one of the vehicle routes (denoted by s). Such a route consists of a sequence of locations with corresponding departure times. The function *Feasible(r, i, j)* checks whether the pickup location of request r can be inserted after location i and whether the delivery location can be inserted after location j. The function *Cost(r, i, j)* calculates the additional costs for this insertion. The variables marked with a star ($*$) denote the best values find for these variables. For example, $c*$ denotes minimum additional costs, $i*$ denotes the best position after which the pickup node can be inserted, etcetera.

Time windows

The feasibility step in the insertion heuristic described above contains a check whether time window constraints of both the current request and requests that have already been inserted are not violated. Solomon [40] describes an efficient way to check these constraints.

We assume that there exists a current route with multiple locations. Every location i has an associated time window with an earliest possible time s_i and a latest possible time e_i at which the vehicle can depart from that location. The current departure time from a location i is denoted by u_i, and the departure time after inserting a new location is denoted by u_i^{new}. The waiting time at a location w_i is calculated as $\max\{e_i, u_{i-1} + d_{i-1} + t_{i-1,i}\}$, with d_i denoting the service time duration at a location and $t_{i-1,i}$ the direct travel time between location $i-1$ and location i. Assume further that we want to check upon time feasibility for the insertion of a location l before location j.

When location l will be inserted before location j, the departure time u_j will be *pushed forward* if there is not enough slack time. The insertion is only feasible if, with the push forward in mind, the vehicle still departs from all subsequent locations within the associated time windows. The push forward for location j can be defined as

$$PF_j = u_j^{new} - u_j \geq 0, \tag{2.18}$$

and for subsequent locations as

$$PF_{i+1} = \max\{0, PF_i - w_{i+1}\}, i \geq j. \tag{2.19}$$

When $PF_j > 0$, then some time window constraints can become infeasible, so we have to check these time window constraints for all locations that are scheduled after the location to insert. We can stop checking locations $i > r$ if $PF_r = 0$ (enough slack time to catch push forward), or if $u_r + PF_r > s_r$ (time window constraints become infeasible), or when we reach the end of the route (all time window constraints are feasible). The result of the first and third stop condition is that the new location can be inserted at the specified condition. The result of the second stop condition is that the new location cannot be inserted, because time window constraints are violated.

A similar approach can be used for checking the feasibility of capacity constraints. Notice that these constraints only involve the locations between the pickup and the delivery location that are inserted.

Implementations for dynamic problems

Implementations of the basic insertion heuristic described above mainly differ in the way that additional costs are calculated and the type of constraints tested in the feasibility phase. Remember that for dynamic problems, the order in which requests are inserted is determined by the order in which they arrive. In addition, the following approaches have in common that they have constraints that deal with travel time.

Madsen et al. [25] solve a real-life DDARPTW with an insertion heuristic called REBUS. This heuristic inserts new requests into existing vehicle routes taking into account the difficulty of insertion. The algorithm is tested on a 300-customer, 24-vehicle instance, and good quality solutions, derived within very short computing times, are reported. Constraints involving maximal route duration and maximal deviation between actual and shortest possible ride time are added.

A fuzzy logic approach is developed by Teodorovic and Radivojevic [42] and works with fuzzy functions to calculate a certain preference for a request to be assigned to a vehicle. Upon receiving a new request, this request is inserted into one of the vehicle routes according to one of nine rules which depend on whether the insertion has a ”small”, ”medium” or ”big” increase of traveled distance, vehicle waiting time, and passenger ride time. Depending on the outcome, the algorithm assigns a preference to each vehicle. This system was tested on and solved 900-requests instances within reasonably short computation time.

Finally, Coslovich et al. [7] propose a two-phase method, which first creates a feasible neighborhood of the current route in an off-line phase. An on-line phase is then used to insert the new request with the objective of minimizing dissatisfaction. Other constraints used in this method are the deviation from the desired service time and an upper bound on the excess ride time. Instances containing 25 to 50 requests are solved with this approach.

For more implementations of insertion heuristics, we refer the interested reader to an overview paper by Cordeau and Laporte [6].

2.2.3 On-line optimization

The disadvantage of using exact algorithms in a dynamic environment is that these algorithms take too much computation time. The disadvantage of using heuristics is that in most cases only near-optimal solutions are produced, and that in some cases the solutions are even significantly far from the optimal solution. In a technique called *on-line optimization* the optimal solution is searched for with exact algorithms, but by only taking into account that part of the problem that we are interested in. For example, when searching for the best departure times of the locations of a request to insert into a current schedule, only that part of the current schedule that can be influenced by inserting the new request needs to be considered in the solution process. This results in smaller problems as input for exact algorithms, which implies less computation time.

In an approach by Colorni and Righini [3] only the most urgent requests and the delivery locations of those requests currently served are taken into account, to reduce the computation time. They also recognize that the less requests are taken into account, the faster computation becomes, but that this results in less accurate solution (i.e. the further from the optimal solution).

More recently Mahr et al. [26] uses an on-line optimization technique that only takes into account the current state of the world and makes definite decisions just before a request is served. This technique is used to compare an agent-based assignment approach with a centralized optimization approach. In the following section we introduce such an agent-based approach and the elements that are contained in decentralized solution concepts.

2.3 Decentralized solution concepts

Instead of using centralized approaches, where a single authority is used to assign requests to vehicles, these vehicles can negotiate among themselves about the assignment of requests. It is shown by Mes et al. that a properly designed multi-agent approach performs as good as or even better than traditional methods [28]. In another article by Mahr et al. the authors conclude that for problem instances where less than 50% of the requests are known in advance, an agent-based approach is competitive to an optimization approach [26]. The basic idea of such a system is that every vehicle calculates the costs needed to serve an incoming request and proposes an offer to a special entity that subsequently assign the request to a specific vehicle.

2.3.1 Decentralized versus centralized approaches

In this section we compare decentralized scheduling approaches with centralized ones. We also mention the key issues to be handled in a distributed setting.

Approaches are decentralized if there is no single, central authority that is in control of solving the problem. The control is distributed over different agents, that act together to solve the problem. The way these agents act together can be different; when they cooperate then they have the same common goal to solve the problem, but if they are competitive then they are self-interested and the solution to the problem is a result of this competition.

Because different agents are involved, such a system is often called a *multi-agent system* (MAS). The basic idea in multi-agent based scheduling is that intelligent agents, which in our case are the vehicles, schedule their own routes. This is the opposite of a centralized scheduling approach in which the dispatcher assigns requests to vehicles and constructs the routes for these vehicles.

We can mention four reasons why a decentralized method can be more suitable for planning and control of dynamic transportation networks and that motivate the use of a multi-agent based system [44, 28]. The first reason is that the domain of transportation is naturally decentralized. Vehicles are autonomous entities that can make their own decisions. Rather than letting a central authority decide what they have to do, they can decide for themselves. The second reason is that global optimization algorithms can be sensitive to information updates, which can result in major modifications of the plan due to minor changes in information. Because in multi-agent systems decisions are made locally, problems can be solved and modifications can be made locally. Thirdly, timely response to unexpected events may not be possible in centralized solutions, because the algorithm may take too much time to complete. The central authority becomes a bottleneck in the computations, while in multi-agent systems, all agents have computational power and can aid in solving the problem. This can make the system more scalable. Finally, individual agents (e.g. vehicles or customers) can be autonomous, self-interested and not cooperative, and therefore are not always willing to share all their information. Centralized algorithms are based on the fact that the dispatcher has complete information about the system.

In contrast to centralized methods, there are two main issues that have to be dealt with. All agents have to process the information they are responsible for, and the agents have to communicate with each other.

A multi-agent system consists of several intelligent autonomous agents pursuing their own goals. When agents are self-interested these goals does not have to be equal (e.g. maximization of own utility), but when they are cooperative these goals are equal for each agent (e.g. maximization of total utility). As in centralized solutions, the decision to assign a specific request to a vehicle mainly depends on how well these two are suited for each other. In a centralized approach, the central authority has this information (or can calculate this) for all vehicles, but in a decentralized approach all vehicles have to determine and announce this themselves. We call this part of information of a vehicle a bid, and the announcement and assignment process can be seen as an auction.[4] In Section 2.3.3 we further discuss bid determination in cooperative and competitive settings.

As opposed to centralized methods, in which all information is known by

[4]In our definition of an auction, agents can behave strategically (competitive) or not strategically (cooperative).

the central authority, agents in a decentralized approach need a clear communication protocol to transfer information. Agents need to communicate their bids, they have to be noticed about the final assignments, and new requests have to be announced. Further details of how agents can communicate are described in Section 2.3.2.

2.3.2 Negotiation

Vehicles and requests for transportation have to find each other at a virtual market. In order to find each other they have to communicate and negotiate about prices and contracts. The Contract Net Protocol is one of the earliest approaches that precisely defines such a communication protocol. In this section we briefly discuss this protocol and variations of it.

The Contract Net Protocol (CNP) has been originally developed by Smith [39] to specify problem solving communication and control for nodes in a distributed solver. It facilitates distributed control of cooperative task execution with efficient high level internode communication. The key problem to be solved is how to distribute tasks among nodes that can process them. Nodes with tasks to be executed must find in one way or another a node that can execute these tasks. The basis for the CNP is the process of two nodes, the manager and the contractor, negotiating for a contract that says that a node has to execute the other node's task. First, available contractors evaluate task announcements from several managers and submit bids on the tasks for which they are suited. Then, the managers evaluate the received bids and determine which contractor is most appropriate to execute the task. The process can be repeated by further composing the task in smaller parts and letting the contractor act as a manager in the next round.

The protocol precisely defines the messages that can be send between the nodes and the processing of the messages after receiving a message. A distinction is made between task announcements, task announcement processing, bidding, bid processing, contract processing, reporting results and termination. In all these stages of the protocol different messages can be send and information can be exchanged. The message types have been designed to capture the types of interactions that arise in a task-sharing approach to distributed problem solving. Message contents have been selected to capture the types of information that must be passed between nodes to make these interactions effective.

The CNP is based on the fact that the agents (i.e. managers and contractors) are cooperative to solve a common problem. In a paper by Vokřínek et al. [43] the CNP is brought to competitive environments, a protocol they call the Competitive Contract Net Protocol (CCNP). This protocol does not only cover the phase of contracting the commitments, but does also allow for negotiation about decommitment and termination. It consists of three phases:

- a contracting phase, where conditions of agreement are concluded,

- an optional decommitment phase, where a contract can be broken, and
- a contract termination phase, where the compliance with the concluded contract conditions is evaluated.

To suite the CNP for transportation scheduling different modifications can be found in the literature. Fisher et al. [12] developed the Extended Contract Net Protocol (ECNP) which include better tools for task decomposition and task allocation suited for the transportation domain. This ECNP is used to further decompose requests and allocate requests to vehicles within a company.

Three shortcomings of the ECNP are the lack of backtracking in case of infeasible solutions, the impossibility for contractors to bid the same time on different requests, and the bid comparison, which is only based on price. Perugini et al. [35] developed the Provisional Agreement Protocol (PAP) which is based on the ECNP, but overcomes all the above shortcomings to suit their domain of global transportation scheduling.

An approach especially developed for the on-demand planning of passenger transportation requests is developed by Cubillos and Demartini [8]. Instead of direct negotiation between managers and contractors, a mediator is introduced through which all communication is passed. The task announcement and reception of bids is carried out by the mediator, who then forwards only a sub-group of the received bids to the manager, filtering out the other ones. This allows solution concepts that lie somewhere in-between a completely centralized and decentralized approach. The mediator has a more global vision of the underlying assignment problem and by filtering it can eliminate solutions that are too bad from a global perspective. This results in less communication and better solutions.

Miyamoto et al. [30] have developed a route planning method for the dial-a-ride problem. The method is very general and distinguishes the assignment problem and the routing problem. The assignment problem is solved using an implementation of the CNP and the routing problem is solved using a heuristic.

2.3.3 Value determination

Decentralized approaches use some kind of auction to assign transportation requests to vehicles. This is also the case in centralized approaches, but in those methods all information is known by the central authority. In this section we describe the way agents determine their bids. The determination of the value of a bid for a request is important for the vehicle (assignment of requests), the customer (price to be paid) and the system as a whole (efficiency).

The auction mechanism used by Mes et al. [28, 27] is the so called *second-price sealed-bid auction*, which the authors choose because of its simplicity. In this kind of auction, all vehicles announce their bids to the customer, without knowing about other agents' bids. The request is assigned to the vehicle with the lowest bid and the price to be paid by the customer is the value of the second-lowest bid. They also notice that the optimal bid for a single auction is the net cost price of the bidder. We notice later that such a bid is not always optimal in competitive environments. For more details about auctions we refer to Chapter 3.

Every time a new request becomes known, an auction is started and vehicles can bid to get the rights to serve the request. Note that we deal with a reverse auction here, because we have multiple sellers (the vehicles) and a single buyer (the customer that places a request). This means that the bidder with the lowest bid wins the auction and must serve the request.

A result of using a second-price sealed-bid auction is that (in an individual auction) the optimal bid of a vehicle for a new request is equal to the minimal additional costs from serving its current requests plus the new request. These additional costs are composed of additional waiting time, serving time, and possible penalty costs when some requests cannot be served on time anymore. All these components depend on the internal scheduling of the vehicle agent, so the bid price is dependent on how a vehicle inserts new requests into its current schedule. For this internal scheduling the heuristics and exact algorithms described in Section 2.2 can be used.

Often only direct costs of adding a new request to a current schedule are considered. We can argue that this is usually not the optimal bid in a competitive environment, because of the uncertainty about future requests assigned to a vehicle. Every request assigned to a vehicle influences the bids and assignments of this vehicle for future requests. For example, a vehicle can act by placing a very high bid for a request because he thinks in the near future a request with more potential profit will be announced. The set of auctions can be seen as a sequential auction instead of multiple single auctions. We further discuss this phenomenon in Chapter 3 (Section 3.3).

To incorporate some look-ahead capability, one can determine bids based on full opportunity costs instead of direct costs. If a new request is not the last request that will be auctioned, the total profit that a vehicle can obtain in successive auctions highly depends on the outcome of the current auction. This means that the vehicle agent must incorporate in the bid, the potential loss of profit in successive auctions by winning the current auction. Such a method is proposed by Figliozzi et al. who developed a one-step-look-ahead algorithm, which evaluates future profits of one step or period into the future [11]. Experiments show that this approach outperforms the naive method (using only direct costs) in loaded travel distance and that it results in higher profits.

Another aspect to take into account in the determination of a bid is the chance of new requests in specific regions of the serving area. If a new request has its delivery location in a popular area (i.e. where many requests have their pickup location), this can increase the bid for this request [41].

In the chapter that follows we take a deeper look into the details of auctions, and especially how to deal with uncertainty of future requests.

Chapter 3

Auctions

An auction can be seen as a mechanism to solve the problem of allocating a single item or multiple items among multiple self-interested entities. As described in the previous chapter, this is exactly what has to be done in the DDARPTW; allocating requests to vehicles. In this chapter we provide a short introduction into auction theory, which is used to develop an auction mechanism to allocate requests to multiple companies.

Auctions are actually applications of mechanism design, so before we take a look at auctions in Section 3.2, we start by introducing mechanisms in Section 3.1. One of the most powerful and most often used auction types is the second-price sealed-bid auction. We also use this auction in the mechanism we propose for the DDARPTW, so some theory about this type of auction is given in Section 3.2.3. The properties of such a second-price sealed-bid auction do not always hold when putting multiple auctions in a sequence. Such a sequence of auctions is called a sequential auction, and different techniques to avoid undesirable properties are discussed in Section 3.3.

For more details about the subjects discussed in this chapter we refer the interested reader to books by Krishna [22] and Shoham et al. [38].

3.1 Mechanisms

An auction is actually an application of mechanism design, just like for example voting mechanisms, and we therefore start to introduce mechanisms. First, we define the context in which mechanisms appear, known as a *Bayesian game*. We call this setting a *game* because we investigate strategic behavior between different players, and we call it *Bayesian* because mechanism design is most often studied in settings where agents' preferences are unknown to each other and so we deal with a setting with incomplete information. The latter statement implies that each player has a belief of the type of every other player, which this player can use to play the game. This belief can change over time, which adds a probabilistic element to the game.

3.1.1 Context

Let us assume an environment of n agents with certain preferences that are private knowledge and that are modeled by a single type, denoted by $\theta \in \Theta$. In each stage of the game, agents are allowed to perform an action (e.g. place a bid, throw a dice) $a \in A$. The action they choose to perform depends on their strategy $s \in \Sigma$, which is a function that maps types to actions.

A Bayesian game is composed of a *Bayesian game setting* and a *mechanism.*

Definition 1 (**Bayesian game setting**). *A Bayesian game setting is a tuple* $(N, \Omega, \Theta, p, u)$*, where*

- N *is a finite set of* n *agents,*
- O *is a set of outcomes of the game,*
- $\Theta = \Theta_1 \times \cdots \times \Theta_n$ *is a set of possible joint type vectors,*
- p *is a probability distribution on* Θ*, and*
- $u = (u_1, \ldots, u_n)$*, where* $u_i : O \times \Theta \mapsto \mathbb{R}$ *is the utility function for each agent* i*.*

Note that we have not mentioned the actions that the players can take in the definition of a Bayesian game setting. We have only defined the utility function of the players, which is a function that denotes the value that a player has for a certain outcome of the game. A mechanism for the Bayesian game setting is defined as follows.

Definition 2 (**Mechanism**). *A mechanism in a Bayesian game setting is a pair* (A, M)*, where*

- $A = A_1 \times \cdots \times A_n$*, with* A_i *the set of possible actions for agent* $i \in N$*, and*
- $M : A \mapsto \Pi(O)$ *maps each action profile to a distribution over outcomes.*

The purpose of mechanism design is to design a mechanism for a given game setting such that the outcome of the game has certain desired properties. So roughly spoken, we try to design rules for the game, such that playing the game results in a desired outcome. Such a mechanism has to deal with the fact that the agents are self-interested and might lie about their true preferences to become better off. This strategic behavior of agents influences the properties of the outcome of the game. We therefore have to search for mechanisms that together with the game setting gives us, in equilibrium, a desired outcome, no matter what the actual preferences of the agents are. With *equilibrium* we refer to a state of the system in which competing influences are balanced and where, without distortion, no changes take place.

3.1.2 Equilibria

Different equilibria can be distinguished, but in this text we limit the discussion of equilibria to the *dominant strategy equilibrium* and the *Bayes-Nash equilibrium* because these two are the most commonly used in strategic environments with agents having incomplete knowledge.

We call the mapping from a player's utility function to the set of actions it performs a player's *strategy* and denote this by s_i for a player i. Denote the vector of other players' strategies by s_{-i}. Players try to maximize their own utility by taking actions that increases the probability of a high-valued outcome. When a player plays a strategy to maximize its utility independent of the other players' strategies, it plays a *dominant strategy.*

Definition 3 (**Dominant strategy**). *A strategy s_i of a player is called dominant if the following condition holds: $u_i(s_i, s'_{-i}) \geq u_i(s'_i, s'_{-i})$. This condition states that the utility gained by the agent playing strategy s_i is greater than or equal to the utility when it plays another strategy s'_i, independent of the other players' strategies s'_{-i}.*

When all players play their dominant strategy, the game is in equilibrium. We call this a *dominant strategy equilibrium* and the mechanism is said to be *strategyproof.*

Definition 4 (**Dominant strategy equilibrium**). *A Bayesian game is in a dominant strategy equilibrium if all strategies played by the players are dominant strategies.*

It does not always have to be the case that every player in a game has a dominant strategy. A player can also maximize its utility by anticipating upon the strategies the other players play. In this case a Bayes-Nash equilibrium can exist, when, taken into account the beliefs about the strategies the other players play, a player cannot gain by playing another strategy than the equilibrium strategy.

Definition 5 (**Bayes-Nash equilibrium**). *A Bayesian game is in a Bayes-Nash equilibrium if every player i plays a strategy s_i which is a best response to the expectation over the strategies played by the other players s_{-i}. Formally, this can be stated as follows: $E[u_i(s_i, s_{-i})] \geq E[u_i(s'_i, s_{-i})]$, where $E[]$ denotes the expectation over the other players' strategies according to some known distribution.*

Note that when a game is in a dominant strategy equilibrium, it is also in a Bayes-Nash equilibrium, but not the other way around.

3.1.3 Implementation

One of the most desired properties for the outcome of a game is the consistence of this outcome with a given social choice function $C : u \mapsto \Omega$. This function maps the utility functions of all players to a "best" outcome, that would have been chosen by a single authority (e.g. the mechanism designer) that has the power to select an outcome. The goal is to design a mechanism that implements this social choice function.

We consider two implementation concepts that differ in the way that other players' actions influences the actions of a single player. The first implementation concept is an *implementation in dominant strategies* and the second solution concept is an *implementation in Bayes-Nash equilibrium.*

Definition 6 (Implementation in dominant strategies). *Given a Bayesian game setting (N, O, Θ, p, u), a mechanism (M, A) is an implementation in dominant strategies of a social choice function C if for any vector of utility functions u, the game has an equilibrium in dominant strategies, and in any such equilibrium a^* we have $M(a^*) = C(u)$.*

Definition 7 (Implementation in Bayes-Nash equilibrium). *Given a Bayesian game setting $N, O, \Theta, p, u)$, a mechanism (M, A) is an implementation in Bayes-Nash equilibrium of a social choice function C if there exists a Bayes-Nash equilibrium of the game such that for every $\theta \in \Theta$ and every action profile $a \in A$ that can arise given type profile θ in this equilibrium, we have that $M(a) = C(u(\cdot, \theta))$.*

Besides the desire of the outcome to be consistent with a specific social choice function, there are some other desired properties. The most important of those properties is *truthfulness*, which means that agents act truthfully in the sense of disclosing their preferences.

3.1.4 The revelation principle

In the search for truthful mechanisms we can limit ourselves to a small subset of all possible mechanisms. Let us define a *direct mechanism* to be a mechanism in which the only action available to the players is to announce their type. Then it is possible to define the set of actions available to a player, A_i, by Θ_i, the set of possible types. A player can lie by announcing a type $\hat{\theta}_i$, that is different from its real type θ_i. A direct mechanism is truthful or *incentive compatible* if it is a dominant strategy for each player to announce its real type. So every direct mechanism that implements a social choice function in dominant strategies, is truthful. In order to search for truthful mechanisms that are not direct, we can use the following principle.

Theorem 1 (Revelation principle). *If there exists any mechanism that implements a social choice function in dominant strategies, than there is a direct mechanism that implements this social choice function in dominant strategies and is truthful.*

Proof. Assume an arbitrary mechanism that implements a social choice function in dominant strategies. Now we construct a new, direct mechanism, in which the players disclose their types to the mechanism. The mechanism determines the dominant strategies of the players and chooses the outcome, that would have been chosen by the original mechanism.

If a player can lie about its type in the new mechanism, it implies that it can be better off by following another strategy. This contradicts with the fact that the chosen strategy is a dominant strategy. This implies that the new mechanism is dominant-strategy truthful. □

With use of the revelation principle, we can limit the search for truthful dominant strategy implementations of social choice functions to direct mechanisms. The same reasoning holds for truthful Bayes-Nash equilibrium implementations.

3.1.5 Quasi-linear utility functions

It turns out that not all social choice functions can be implemented by a truthful mechanism. One of the reasons for this is that there are no restrictions for the preferences of the players. That way it is possible for a single agent to always determine the social choice by its preferences. We do not discuss this problem in further detail, because that is out of the scope of this text. We rather focus on the solution, to be able to design truthful mechanisms for social choice functions.

We limit the set of preferences that a player can have by introducing a money element and defining *quasi-linear utility functions.* The outcomes of the game are split up in a non-monetary part (e.g. the allocation of an object to an agent) and a monetary part (e.g. a payment to the auctioneer) and these parts are incorporated in the utility functions of the agents as follows.

Definition 8 (**Quasi-linear utility function**)**.** *Agents have quasi-linear utility functions in an n-player Bayesian game when the set of outcomes is $O = X \times \mathbb{R}^n$ for a finite set X, and the utility of an agent i given joint type θ is given by $u_i(o, \theta) = v_i(x, \theta) - f_i(p_i)$. Here, $o = (x, p)$ is an element of O, $v_i : X \times \Theta \mapsto \mathbb{R}$ is an arbitrary function to calculate the value for an outcome from an agent's type vector, and $f_i : \mathbb{R} \mapsto \mathbb{R}$ is a strictly monotonically increasing function that calculates the (possibly negative) payment an agents has to make.*

By using quasi-linear utility functions we are sure that the preference of an agent for a selection of any choice $x \in X$ is independent from its preference for having to pay a certain amount $p_i \in \mathbb{R}$ to the mechanism. A second implication is that agents only care about their own payments to the mechanism and not about the payments other agents have to make.

As can be derived from the function f_i, there is some flexibility in calculating and evaluating payments. Agents can have different *risk attitudes* in the way they value a unit amount of payment. This risk attitudes can be incorporated in the function f_i. Unless stated otherwise we assume that agents are risk neutral, such that f_i can be ignored.

We are now ready to define a general mechanism for the setting in which agents have quasi-linear utility functions. Actually, in such a mechanism the function M described above is split up into two functions; a function $\varkappa$ for the mapping of actions to choices (e.g. allocations) and a function φ that maps actions to payments.

Definition 9 (**Quasi-linear mechanism**). *A mechanism for a setting in which agents have quasi-linear utility functions (for a Bayesian game setting $(N, O = X \times \mathbb{R}^n, \Theta, p, u)$) is a triple $(A, \varkappa, \varphi)$, where*

- *$A = A_1 \times \cdots \times A_n$, in which A_i is the set of actions available to agent $i \in N$,*
- *$\varkappa : A \mapsto \Pi(X)$ is a function that maps each action profile to a distribution over choices, and*
- *$\varphi : A \mapsto \mathbb{R}^n$ is a function that maps each action profile to a payment for each agent.*

The definition of truthfulness can now be stated according to quasi-linear mechanisms.

Definition 10 (**Truthfulness**). *A quasi-linear mechanism is truthful if it is direct (i.e. $A_i = \Theta_i$) and agent's i equilibrium strategy is to adopt the strategy $\hat{v}_i = v_i$ (i.e. announcing its real valuation for the outcomes), for all i and v_i.*

With this brief introduction into mechanisms we are ready to focus on auctions, which are an important application of mechanisms.

3.2 Auctions as mechanisms

We noted earlier that auctions can be seen as mechanisms to solve problems related to allocating goods among different self-interested agents. In this section we define auctions as mechanisms, show how auctions can be used to solve the DDARPTW, and describe one of the most often used types of auctions.

3.2.1 Definition

Agents that compete to obtain the goods are called *bidders* or buyers, and agents that allocate the goods are called *sellers* or auctioneers. We assume that agents have quasi-linear utility functions, so the quasi-linear mechanism setting can be used.[1]

The following elements of a mechanism with respect to an auction have to be identified:

- a set of agents N,
- a set of outcomes $O = X \times \mathbb{R}^n$, where n denotes $|N|$,
- a set of actions A_i available to each agent $i \in N$,
- a choice function $\varkappa$ that selects one of the outcomes given the agents' actions, and
- a payment function $\wp$ that determines what each agent must pay given the agents' actions.

Obviously, the set of agents N contains all the agents that are willing to pay for the good, but in some settings the seller itself can also act as an agent (e.g. in cases where the objective of the auction is to maximize revenue), and then not all agents are bidders. Therefore, we denote by $B \subseteq N$ the set of bidders. The set of outcomes, O, consists of all possible ways to allocate the good and all possible ways of charging the agents.

The set of possible actions, A_i, possible for an agent i can vary for different auctions according to the defined rules. One can think of an auction in which an agent can make a single bid and one can think of a different auction in which multiple sequential bids can be released by an agent. In this text we only consider settings in which the only possible action for an agent is to place a single bid. In such a setting the strategy for bidder $i \in B$ is a function that maps its valuation for a certain outcome to a bid, $\beta_i : [0, \omega] \to \mathbb{R}^+$, where ω is the highest possible valuation.

The choice and payment functions depend on the objective of the auction. Some example objectives are to allocate the goods as efficient as possible, to maximize revenue for the seller, or to minimize costs for the buyers.

The agents are self-interested in the way that they try to maximize their utility. This utility is a quasi-linear function, described above, based on an agent's valuation for the auctioned good, the outcome of the auction and the payment it has to make.

To be able to decide what action an agent will take and to calculate its expected utility, an agent's valuation must be modeled. This can be done in different ways depending on the amount of information an agent has about the other agents. The purpose of the different settings is to simplify the many different valuations agents can have in order to better analyze specific problems.

[1]From now on we leave out the term "quasi-linear" in front of "mechanism" and assume that the mechanism used is always quasi-linear.

The most commonly used setting is the *Independent Private Value* (IPV) setting in which all agents' valuations are drawn independently from the same distribution. This distribution is commonly known by all agents and the type of an agent depends only on its own information. This setting is appropriate in auctions with buyers that have valuations based on personal tastes.

On the opposite of the IPV setting we find the *Common Value* (CV) setting. In this setting all agents have the same value for the good but that value is not known. Each agent has a private signal about this value, which makes it possible to determine its belief about the distribution of this value. An example can be found in the search for oil. Oil companies all have the same value for a piece of land; namely the value of the amount of oil it contains. However, it is uncertain how much oil the land exactly contains and therefore the beliefs about the value distribution depends on the information retrieval techniques of a company.

When we combine the IPV and CV settings we arrive at a setting called *Affiliated Values* (AV) in which the bidders have both a private-value component and a common-value component. This implies that a high valuation of one particular agent increases the probability that other agents have high valuations as well.

For the rest of this text, unless stated otherwise, we assume the IPV setting. This means that a valuation v_i from a bidder i is drawn, just before the auction starts, from a commonly known distribution function F.

3.2.2 Example of an auction mechanism for the DDARPTW

To give the reader an understanding of how an auction mechanism can be used in the DDARPTW, we identify the elements of a Bayesian game setting and an auction mechanism for the DDARPTW in a multiple-company environment.

The set of agents N that appears in our setting is the set of the n companies. The customer that initiates the auction is not included in this set. An outcome of the game played by the companies is an allocation of a request to one of these companies, together with a (possibly negative) payment for each of this companies. This means that the set of outcomes O consists of all possible combinations of allocations and payments. A type vector of an agent in our setting consists of values that companies can announce to the auctioneer, such that the auctioneer can determine an outcome. In our setting the type of a company is its current schedule of requests. The set Θ consists therefore of all the possible combinations of current schedules that companies can have.

It is hard to define a probability distribution p on the set of possible joint type vectors, because in our setting, the type of a company is (based on) its internal schedule. The utility function vector can be identified for the players in our setting. For each company the utility function is a function of the set of outcomes and the set of possible type vectors. The utility of a company is calculated by subtracting its costs from its income, where the costs arise from serving requests and the income arises from negative payments made by the company to the mechanism.

If we want to define an auction mechanism for this setting, the following additional elements need to be identified:

- a set of actions A_i available to each agent $i \in N$,
- a choice function $\varkappa$ that selects one of the outcomes given the agents' actions, and
- a payment function φ that determines what each agent must pay given the agents' actions.

The set of actions that is available to each company defines the way in which this company communicates with the customer that announces the request. The choice function $\varkappa$ is a function of the auctioneer that selects the company that has to serve the request, and the payment function φ is a function that determines the (negative) payment that each company must pay to the auctioneer. One possible way these functions can be defined is to follow a second-price sealed-bid auction. This often used type of auction is discussed next.

3.2.3 Sealed-bid second-price auctions

One of the most often used auction types is the so called *sealed-bid second-price auction.* In such an auction the bidders announce their bid in a "sealed envelop" to the auctioneer, so that these bidders cannot view each others' bid. The highest bidder gets the object and has to pay the amount of the second-highest bid.

If we let v_i be the true valuation of bidder i for the good, let b_i be the bid that i places and b_j denotes the bids of the other buyers. The utility for agent i is:

$$u_i = \begin{cases} v_i - \max_{j \neq i} b_j & \text{if } b_i > \max_{j \neq i} b_j \\ 0 & \text{if } b_i \leq \max_{j \neq i} b_j \end{cases} \tag{3.1}$$

So, if the bid of i is the highest bid, it has to pay the second-highest bid and its utility is the difference between its valuation and its payment, and if i does not place the highest bid, its utility is zero. Two or more bidders can also tie, which happens when the bids are equal. The winner is determined by flipping a coin and the utility in this case is also zero, because the price paid (i.e. the bid of the loser of the coin flipping) is equal to the bid of the winner.

In the following theorem it is showed that bidding behavior in a second-price sealed bid auction is straightforward and that the best bid an agent can place is equal to its valuation, $b_i = v_i$.

Theorem 2. *In a second-price sealed bid auction, it is a weakly dominant strategy for a bidder to bid its true valuation: $\beta(v) = v$.*

Proof. We start by describing why a bidder cannot gain from bidding less than its true valuation v_i and then describe why it cannot gain from bidding more than this value.

Suppose bidder i bids $\hat{v}_i < v_i$ and let $r_i = \max_{j \neq i} b_j$ denote the highest of the other agents' bids. The bidder still wins the auction if $v_i > \hat{v}_i > r_i$ and its profit is still $v_i - r_i$. If $r_i > v_i > \hat{v}_i$, bidder i still loses with zero profit. However, it is worse off if $v_i > r_i > \hat{v}_i$, because now it loses the auction, where it would have won the auction if it had bid v_i. So, when the bidder bids less than its true value, it will never be better off.

Now, suppose that bidder i bids more than its true value, $\hat{v}_i > v_i$. If $\hat{v}_i > v_i > r_i$, it still wins the auction for the same price with profit $v_i - r_i$. If the bidder did not won before, and does still not win, $r_i > \hat{v}_i > v_i$, then it makes the same zero profit. In the case of $\hat{v}_i > r_i > v_i$ the bidder makes a loss, because then it wins the auction, but the price it has to pay is more than its true valuation.

This leads to the conclusion that a bidder can never be better off by bidding more or less than its true valuation, it can only be worse off. It implies that truthful bidding is a weakly dominant strategy for a bidder. □

It is also possible to write down the expected payment of a bidder i with valuation v_i. Clearly, this expected payment can be written as:

$$\varphi(v_i) = \text{Probability of winning the auction} \times \text{Expected second-highest bid.}$$

What is the probability that v_i is higher than the maximum of all other bids? Let the random variable $Y \equiv Y^{(n-1)}$ denote the highest value among the $n-1$ bidders, other than i.[2] Also, let G denote the distribution function of Y. G can be derived from the distribution function of the bidders' valuations F: for all y, $G(y) = F(y)^{(n-1)}$. So, the probability that the value of a bidder v_i is higher than the maximum of the other bidders' values is $G(v_i)$.

We now have to find the expected second-highest bid, given that v_i is the highest value. Using the definition of Y, we can easily say that the expected second-highest value is Y.

To conclude, the expected payment of bidder i in a second-price auction can be given by:

$$\varphi(v_i) = G(v_i) \times E[Y \mid Y < v_i] \tag{3.2}$$

We derived the dominant strategy for a bidder and its ex ante expected payment in a sealed-bid second-price auction. We now turn our attention to a sequence of this kind of auctions and show some problems that arise.

[2]This variable is also called the highest order statistic.

3.3 Sequential auctions

A sequential auction is actually a sequence of single auctions of whatever which type and is often used in environments where the objects for sale become available at different points in time. The main difference with single auctions is that the bidding behavior of bidders in a sequential setting strongly depends on future auctions. In this section we describe a generalized model to study this kind of auctions and we show new equilibria for specific settings. We also discuss a solution to the problem that sequential auctions do not have a dominant strategy although their single auction elements do.

3.3.1 The model

The model we describe to study sequential auctions is a combination of the generalized model used by Juda and Parkes [18] and the model to incorporate different levels of information uncertainty discussed by Fatima et al. [10]. The model allows bidders to desire multiple non-identical objects and it allows bidders to arrive and depart dynamically.

We denote by B the set of bidders, which is a subset of the set of agents N.[3] The number of bidders is denoted by n and is equal to $|B|$. The different types of goods are denoted by $G = [1 \dots K]$ for a total number of K different goods. Because we also allow bidders to desire multiple goods, we need to specify valuations for bundles of goods instead of valuations for single goods. Therefore, we denote by L a bundle of goods, where $L \subseteq G$. We assume that a bundle contains at most one good per type.

In this online setting, the type of a bidder does not only include the valuation for a certain bundle of goods, but also the arrival and departure time of the bidder. Formally, the type of bidder $i \in B$ is denoted by the tuple (a_i, d_i, v_i), with arrival time $a_i \in \{0, 1, \dots\}$, departure time $d_i \in \{0, 1, \dots\}$ and private valuation $v_i(L) \geq 0$ for each bundle of goods L that can be received by the bidder between its arrival time a_i and its departure time d_i. For all other bundles a bidder has zero value. Again, bidders have quasi-linear utilities, so the utility of bidder i receiving bundle L is $u_i(L, p) = v_i(L) - \varphi$ with a payment φ.

To deal with different information uncertainties, we further denote by m_k the number of single auctions where a good of type k will be awarded and let $P_{\text{last}}(j_k)$ be the probability that auction j_k (for $1 \leq j_k \leq m_k$) is the last auction held for a good of type k before the bidder will depart. At last, let $P_{\text{bidders}}(j_k, r)$ denote the probability that auction j_k for a good of type k has r bidders.

[3]Note that the seller is also an agent, but that it is not included in B.

3.3.2 The sequential auction problem

Because bids of a bidder do not only depend on the current auction, but also on future auctions, it is not obvious that a dominant bidding strategy exists, even if each individual auction in the sequence is strategy-proof (i.e. the sequential auction is locally strategy-proof). When this is the case, one talks about the sequential auction problem, which is stated in Definition 11.

Definition 11 (**The sequential auction problem**). *The phenomenon that, given a sequence of auctions and despite each single auction being locally strategy-proof, it does not follow that a bidder has a dominant bidding strategy in the sequential auction.*

There are a few causes for this problem to arise. The first cause is related to the bids of the competitors of a bidder. These competitors' bids are conditioned on the bids of the bidder in previous auctions, because there is uncertainty about the number of bidders in the next auction (did the bidder won the previous auction?) and the number of auctions still to come. Therefore, a bidder does not know how to optimally influence the bids of competitors.

The second cause is called the *multiple copies problem*. The multiple copies problem occurs when a buyer can act in multiple auctions for the same good. It often happens that the winning bid is not the same in all these auctions and therefore, the final price of the good for the buyer depends on the auction it acts in. At the end, the buyer could end up winning the auction with a higher winning bid than the winner in another auction, even if the latter agent's real value is higher.

The existence of bundles that are substitutes is also a reason for the problem to appear. Two bundles L_1 and L_2, such that $L_1 \cap L_2 = \emptyset$, are substitutes if $v(L_1 \cup L_2) < v(L_1) + v(L_2)$. A bidder does not know which bundle among the substitutes to pursue and therefore does not have a dominant strategy.

The fourth cause is the existence of items with uncertain marginal value[4], which happens if the marginal value of an item depends on the other goods held by the bidder. The problem is that the bidder does not no what goods it retrieves until its departure time. This is called the *exposure problem*. Informally, the exposure problem occurs whenever an agent has a valuation for a bundle of goods which is greater than the sum of the valuations for the single goods. In other words, there exists complement bundles. Two bundles L_1 and L_2, such that $L_1 \cap L_2 = \emptyset$, are complements if $v(L_1 \cup L_2) > v(L_1) + v(L_2)$. This agent can buy a single good for a price higher than its valuation, in the hope that it will win the rest of the bundle of goods in succeeding auctions at a profitable price. If the agent fails to obtain the rest of the bundle, it will end up with a loss, due to its overbid for the single good in the earlier auction.

[4]The marginal value of a good is the extra value this good adds to the total value of a bundle.

As by Juda and Parkes [18], the causes of the sequential auction problem can be summarized as follows:

Theorem 3. *Given locally strategy-proof single-item auctions, the sequential auction problem exists for a bidder if and only if (1) its competitors' bids are conditioned on its bids in previous auctions, (2) the multiple copies problem exists, (3) there exist bundles that are substitutes, or (4) the exposure problem exists.*

Proof. (Sketch) ($\Rightarrow$) It can be seen from the discussion of the sequential auction problem above, that if one of these causes exists, there can be no dominant strategy. ($\Leftarrow$) If none of the causes exists, then in all single auctions it is a dominant strategy to bid the marginal value of the auctioned good. $\square$

Next, we search for solutions to the sequential auction problem, which can be divided into options-based solutions, and techniques to find other than dominant strategy equilibria.

3.3.3 Options-based solution

A decentralized solution to the earlier noticed sequential auction problem is to introduce *options*. An option can be seen as a right to buy the specific object for an agreed price, called the *exercise price*. Instead of acquiring the good directly, the buyers first bid to acquire an option for that good. At the end of the sequence of auctions, a buyer can decide whether or not it will exercise this option and pay the pre-agreed exercise price to retrieve the good. The result is that the risk of not winning subsequent auctions (in which complementary objects are auctioned), is (partly) transfered to the seller of the item. This results in a solution to the exposure problem. A distinction can be made between zero-priced options [18], which transfer all the risk to the seller, and priced options [31], which transfer only a part of the risk to the seller.

Zero-priced options

We first take a look at a market design with zero-priced options and it turns out that in this setting truth-telling is a dominant strategy. It must be noted that in the original paper by Juda and Parkes [18], the buyers do not bid directly to the seller, but they do this via a so called proxy-agent. We first describe the design by assuming the buyers are intelligent agents and that they communicate directly to the market mechanism. However, this is not a direct revelation mechanism and it can be shown that this auction is not incentive compatible. At the end of the discussion, we conclude that making an explicit distinction between buyers and proxy-agents solves this problem.

In each single auction, an option to obtain a good k is auctioned. Every buyer i determines the value of its bid by calculating its maximum marginal value for the good:

$$v_i(k) = \max_L [v_i(L \cup \{k\}) - v_i(L)] \tag{3.3}$$

The reason for a buyer to bid only its maximum marginal value for the good is that, in this way, it will win any auction that could possibly benefit him and it will only lose those auctions that could never be of benefit. Suppose that it places a bid that is higher than its maximum marginal value. If it wins that auction, it takes the risk of paying an exercise price higher than the maximum benefit of obtaining the good.

The bidder with the highest bid receives the option and the exercise price of this option is set to the second-highest bid. Let the set of all acquired options be denoted by Φ, the bundle of goods corresponding to the set of options ϕ by $L(\phi)$ and denote by $\pi(L(\phi))$ the sum of exercise prices of the bundle of goods corresponding to the set of options ϕ. At its departure time, a buyer determines which options to exercise by maximizing its utility:

$$\phi = \arg\max_{\phi \subseteq \Phi} (v_i(L(\Phi)) - \pi(L(\phi))). \tag{3.4}$$

If there is no combination of options with a positive utility, then all options are returned.

To overcome the *multiple copies problem*, sellers allow buyers to match exercise prices when, in a succeeding auction for the same good, the option for that good has a lower exercise price. It would be obvious to just match the lowest winning price they observe during their lifetime, but it turns out that this scheme will not lead to a truthful mechanism. Assume a bidder who reports a higher value than its real value and another bidder whose real value is higher than the former bidder's value, but less than the former bidder's reported value. Obviously, the first bidder wins the option with the exercise price of the second bidder. Suppose that the same good will be auctioned the next day, but that the losing bidder of the first day is departed. A third bidder, whose real value is less than the others', wins this second auction for an exercise price of zero (because there are no other bidders). If the exercise prices are matched, the exercise price of the winning bidder of the first auction matches zero and it benefits from misreporting its value, because it wins the second auction instead of the first auction. In Table 3.1, setting b, an example is given.

For the reason described above, the following price matching scheme is used. First, when a buyer wins an auction, it memorizes the buyer with the second-highest bid. Second, although a buyer can only acquire one option for a specific type of good, it keeps track of future auctions for this good by checking the exercise price for the auctioned option. If this exercise price is lower than the exercise price of the option it acquired and the winning proxy is not in its local memory (i.e. this is the bidder to which it would have

Setting	Bidder	Type	Day1	Day2
a. Truth	Alice	{Day1,Day1,\$10}	8_{Bob}	-
	Bob	{Day1,Day2,\$8}	-	6_{Claire}
	Claire	{Day1,Day2,\$6}	-	-
b. Misreport (Low price)	Alice	{Day1,Day1,\$10}	-	-
	Bob	{Day1,Day2,$\$\hat{12}$}	10	$10 \rightarrow 0$
	Claire	{Day1,Day2,\$6}	-	0
c. Misreport (Scheme)	Alice	{Day1,Day1,\$10}	-	-
	Bob	{Day1,Day2,$\$\hat{12}$}	10_{Alice}	$8_{\text{Alice}} \rightarrow 6$
	Claire	{Day1,Day2,\$6}	-	0

Table 3.1: Examples of price matching. (a) All bidders report their true valuation. (b) One bidder misreports its valuation and exercise prices are match to the lowest price seen. (c) One bidder misreports its valuation and exercise prices are match using bookkeeping.

competed if it delayed bidding to this later auction), it will match its own exercise price down to the new exercise price. When the highest bidder in the second auction is not the same as the one already in its local memory, it will replace this bidder with the second-highest bidder in the latter auction. If it is the same bidder as in its local memory, it will clear its memory. This way, a winning bidder always keeps track of the bidder who would have won the option, if it itself was not present in that auction. In Table 3.1 three examples are given to show (a) what happens when all bidders tell the truth, (b) what happens when one is misreporting its value and prices are matched with the lowest price, and (c) what happens when one bidder is misreporting its value and prices are matched using the aforementioned price matching scheme.

The algorithm above is not truthful. The reason for this is that the options are priceless and a buyer can therefore obtain as many options it wants by representing itself as a bidder with high values. Suppose a bidder acts as if it values all goods it wants higher than its true valuations. This way it can collect many options and can decide at the end, what options to exercise. This makes the auction very inefficient.

To overcome this problem, so called proxy-agents are introduced. Buyers tell this proxy-agents their valuations for different combinations of bundles of goods, their arrival time, and their departure time. Proxy-agents transform the market into a direct revelation mechanism. In a study by Juda et al. [18] it is proved that truthful revelation of valuation, arrival and departure to the proxies is a dominant strategy for a buyer in this options-based market. It is also showed that the mechanism is individually-rational for both buyers and sellers, which means that taking part of the auction can never give them less utility than not taking part. The utility for buyers is never negative, because the proxy-agent chooses a set of options that maximizes a buyer's utility or it chooses no set at all. For a seller, the exercise price of each option is at least zero, so the utility (i.e. revenue) for a seller is also never negative.

A problem that arises when proxy-agents are introduced is how to elicit the preferences (i.e. valuations) for certain bundles of goods from a buyer. In other words, how can a buyer tell its proxy-agent what bundles of goods it like, or does not like? Some techniques for preference elicitation are developed by Conen and Sandholm [4] and by Parkes [34].

Priced options

An advantage of using priced options over free options is that the options can be auctioned instead of the goods, and that the exercise prices can be fixed by the seller. This can attract more buyers, resulting in a higher revenue for the auctioneer. A disadvantage of a method with priced options is that bidders must also take into account the expected value of an option to justify the costs, which makes it harder to support a simple, truthful dominant bidding strategy [18]. Because we prefer simple, dominant strategies over higher revenue, methods with priced options are only briefly discussed.

Mous et al. [31] propose an approach of using priced options to solve the exposure problem in sequential auctions. In this approach, not the goods, but the options are sold using a first-price sealed bid auction. The exercise price is determined by the seller and it can fix this price in such a way to influence the market entry effect (i.e. to encourage buyers to stay in the market).

It is showed analytically, that using priced options can increase the expected profit for both the buyer and the seller, compared to auctioning the goods without options. The buyer can increase expected profit, because by not winning a next auction, it limit its loss to the option price. The seller can increase expected profit, because a buyer has more money to spend to place a higher option bid, to increase its chances of winning a single auction.

A method to solve the multiple copies problem with priced options is discussed by Gopal et al. [15]. A comparison is made with options in financial markets and it is demonstrated that these options cannot be employed in auction markets, because the fundamentals of pricing the options are different. Experiments discussed in their paper suggest that options can provide significant benefits for the seller as well as the buyer.

3.3.4 Finding other equilibria

The problem described in the previous subsection can be solved in different ways. One way is to find other equilibrium strategies instead of the dominant strategy. We show that a Bayes-Nash equilibrium can be designed for a setting in which the number of bidders and the number of auctions is known, and for a setting in which these numbers are not known.

Known number of bidders and number of auctions

In this first setting, in which we assume identical objects ($K = 1$), the valuations for the m objects to be auctioned are independently and identically distributed across the bidders. We denote by $V_j : \mathbb{R}^+ \to [0, 1] (1 \leq j \leq m)$ the probability distribution function for the valuation for object j. We also assume that each bidder needs only a single object, which implies that there are $n - j + 1$ bidders for auction j, and that these objects are auctioned using sealed bid second-price rules.

In a sequential auction, a bidder's bid does not only depend on the utility it can derive in the current auction, but also on the expected utility of future auctions. When a bidder expects to get a higher utility in a future auction, it will not bid (or will place a zero bid) in the current auction. It turns out that the expected utility of a bidder in a specific auction depends on the number of total bidders in that auction. To calculate this expected utility for a specific auction y we need the ex ante probability[5] that a bidder wins this yth auction in a series from the current to the last auction. Let $\psi(y, j, m, n)$ denote this probability, with $j \leq y \leq m$. Obviously, before the first auction, all bidders have equal probability of winning this auction, so $\psi(1, 1, m, n) = 1/n$. In the second auction, the winner of the first auction does not place a bid anymore, so the ex ante probability of just winning the second auction is $1/(n - 1)$. Then the ex ante probability before the first auction of winning the second auction becomes $(1 - 1/n)(1/(n - 1)) = 1/n$, because the probability of losing the first auction is $(1 - 1/n)$. Following the same reasoning, the ex ante probability before the first auction of winning the yth auction is also $1/n$. We can generalize this to the ex ante probability before the jth auction of winning the yth auction to $\psi(y, j, m, n) = \frac{1}{n-j+1}$. The expected utility for the $(y - 1)$th auction depends on this probability. In the following formula $u_1(j, m, n)$ is the ex ante expected utility from winning any auction in the series of auctions from the jth to the last one m, and $u_y(m, n)$ is the expected utility for the yth auction in the series of m auctions with n bidders for the first auction:

$$u_1(j, m, n) = \sum_{y=j}^{m} \psi(y, j, m, n) u_y(m, n) = \frac{1}{n - j + 1} \sum_{y=j}^{m} u_y(m, n). \tag{3.5}$$

The value of $u_y(m, n)$ can be calculated considering every possible outcome for a bidder in that specific auction.[6] This brings us to the following theorem:

[5]This is the probability calculated before the object values are drawn.

[6]For the last auction, u_y is equal to the expected profit for a single sealed bid second-price auction, but for the other auctions this is more complicated. In our opinion this does not contribute to a better understanding of the theorem, and therefore we refer to existing literature [10, 22].

Theorem 4 (Equilibrium in a sequential second-price auction). *If each auction in a series is conducted using the second-price rules, then the equilibrium for auction j $(1 \leq j \leq m)$ is:*

$$\beta_j(v_j) = \max\{0, v_j - u_1(j+1, m, n)\} \tag{3.6}$$

Proof. (Sketch) Intuitively a bidder has to make a bid of zero if it expect to make more profit in any of the upcoming auctions (when $v_j - u_1(j+1, m, n)$ is negative). When the difference between its valuation and the expected profit in upcoming auctions is positive, the bidder has to spread its chances of winning an object, starting with the current auction. □

For the full proof of Theorem 4 and the calculation of $u_y(m, n)$ we refer interested readers to an article by Fatima et al. [10] or a book by Krishna [22].

Uncertainty about number of bidders and number of auctions

We now point our attention to finding an equilibrium in a setting with an uncertain number of bidders and an uncertain number of auctions [10]. This means that new bidders can enter the series of auctions before every new single auction starts and that the bidders do not know how many objects will be auctioned. Let us denote by $P_{\text{bidders}}(j, r)$ the probability that auction j has r bidders and by $P_{\text{last}}(j)$ the probability that auction j is the last auction. We assume that all bidders know that there are no more than n bidders and no more than m auctions. Also, let R be a vector of size m with element R_j denoting the number of bidders in auction j. First, we find an equilibrium for an uncertain number of bidders, which is then extended to an equilibrium for both uncertainties. The ex ante probability of winning auction y in the series of auctions from j to m, given the number of bidders per auction in R, is:

$$\psi(y, j, m, R) = \frac{1}{R_y} \times \Big(\prod_{k=j}^{y-1}(1 - \frac{1}{R_k})\Big), \tag{3.7}$$

which is just the probability of winning auction y, times the probability of losing all previous auctions.

The ex-ante expected utility of winning any auction y from the jth to the mth then becomes, like the first setting (3.5):

$$u_2(j, m, N) = \sum_{y=j}^{m} \psi(y, j, m, N) u_y(m, N). \tag{3.8}$$

In general, we can say that the ex ante expected utility of winning any auction in the series from the jth to the mth one is:

$$u_2(j, m) = \sum_{R_j=1}^{n} \cdots \sum_{R_m=1}^{n} \Big((\prod_{i=j}^{m} P_{\text{bidders}}(i, R_i)) \times \psi(y, j, m, R) \times u_y(m, R) \Big). \tag{3.9}$$

The equilibrium bids are the same as for the first setting, although the ex ante expected profits $u(j,m)$ are different:

$$\beta_j(v_j) = \max\{0, v_j - u_2(j+1,m)\}. \tag{3.10}$$

This equilibrium can easily be extended to an equilibrium for the setting with an unknown number of objects. We have to insert the probability of y being the last auction into the ex-ante probability of winning this auction in the series from the jth to the mth auction:

$$\psi(y,j,m,R) = \Big(\prod_{k=j}^{y-1}(1 - P_{\text{last}}(k))\Big) \times \psi(y,j,m,R) \tag{3.11}$$

Substituting this in the previous ex ante expected profit, gives:

$$u_2(j,m) = \sum_{R_j=1}^{n} \cdots \sum_{R_m=1}^{n} \Big((\prod_{i=j}^{m} P_{\text{bidders}}(i,R_i)) \times \psi(y,j,m,R) \times u_y(m,R)\Big). \tag{3.12}$$

which leads to an equilibrium bid of:

$$\beta_j(v_j) = v_j - u(j+1,m,n). \tag{3.13}$$

The complexity of the computation of the Bayesian-Nash equilibria depends on the calculation of the functions u_1 and u_2. These functions only depend on information known by all bidders before the start of the first auction. Therefore, these functions can be calculated once and computing the equilibrium bids in an auction takes constant time, $O(1)$. The time to compute the function u_1 is $O(m)$ and u_2 can be pre-computed in time $O(mn^m)$.

Notice that in both settings, the equilibrium bid is build up from the true valuation for the good, minus the expected utility for future auctions. The assumptions that are made (i.e. that we are dealing with identical objects and that each bidder only needs a single object) do not hold for our transportation setting, because each transportation request is different and taxi companies need multiple requests to create a route. Therefore the bid determination method described above cannot be used in the mechanism we describe in the next chapter, but we do use a similar approach in places where we have to deal with uncertainty about future transportation requests. Because we are dealing with two different measures for the bid value and utility, we have to try to express one in the other. The bid is determined by subtracting that part of the valuation that would lead to a significant decrease in probability to win future auctions (i.e. when future requests result in negative profit).

Chapter 4

Moving to a multi-company environment

Service quality is low in the current situation, where only one company has the rights to serve requests made by customers. In this chapter a multi-company environment is proposed, in which multiple companies compete with each other on the assignment of requests. We try to create an incentive for these companies to serve requests with a higher service quality. The general idea is that multiple companies announce an offer to the customer, and that subsequently the customer chooses the company that must serve its request. Conditioned on some negotiated constraints, the winning company can insert the request into its schedule. This way, companies and customers together influence the final assignment of requests to vehicles.

The multi-company setting can be seen as a Bayesian game setting, a concept that is already discussed in Chapter 3. Before we discuss the elements that are contained in a such a setting (e.g. allocation function or payment function), we provide in Section 4.1 an overview of the mechanism we propose. In subsequent sections we discuss the elements of a Bayesian game setting and the corresponding auction in greater detail. Section 4.2 is devoted to describing the type of bid that is used. Opposed to earlier work, bids do not contain monetary values, but do contain a measure of service quality.

In Section 4.3 we describe the type of auction that is incorporated into the mechanism. Payments in this auction are not determined within the auction, but before the auction starts. We show that if payments are too low or too high, undesirable situations arise, and therefore we provide a lower and an upper bound on this payments.

The outcome of an auction is an allocation of the auctioned request and a payment for the winning company. After a company is allocated a request, it must insert this request into its current schedule. By adding a constraint for service quality to an existing Mixed Integer Program (MIP), we ensure that the request is served according to the negotiated conditions. To speed up the solving of the MIP, extra constraints are added for locations that cannot be changed anymore. The added constraints and the insertion process are discussed in Section 4.4.

In Section 4.5 we show that calculating bid values in our setting is not straightforward. Therefore, companies are allowed to incorporate some knowledge about future requests in their calculation, to be able to announce better bids. To simulate the insertion of future requests a basic insertion heuristic is used, to which we add a service quality component. By the use of a Monte Carlo simulation, the best bid value is calculated. In Section 4.5 we show that calculating bid values in our setting is not straightforward. Therefore, companies are allowed to incorporate some knowledge about future requests in their calculation, to be able to announce better bids. To simulate the insertion of future requests the MIP that was discussed earlier could be used, but that would take too much computation time. Therefore, we use a basic insertion heuristic, to which we add a service quality component. This heuristic is repeatedly used in a Monte Carlo simulation to calculate the best bid value for a request.

As mentioned before, we start by giving an overview of the proposed mechanism.

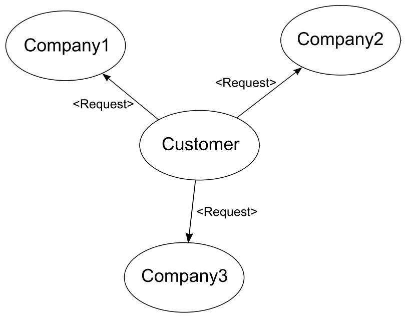
4.1 Mechanism overview

In this section an overview is given of the mechanism we propose to allocate incoming requests to multiple companies. From our knowledge of auctions, described in Chapter 3, we know that these kind of mechanisms are an efficient approach to allocate items to resources. The question is how to fill in the elements of an auction such as the bid type and the bid calculation method. For the sake of completeness of the dial-a-ride system, we also want to know what happens after a request is assigned to a company, i.e. how the request is inserted into a company's schedule. In Figure 4.1 an overview is given of the whole process from announcing a request by a customer to inserting that request into a company's schedule.

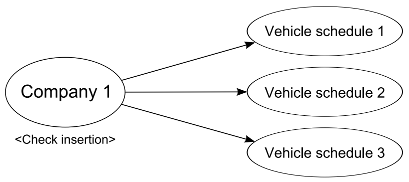
When a customer decides that it wants to travel between two locations, it announces a request to all known companies, shown in Figure 4.1(a).[1] Each company has a number of vehicles that can be used to transport customers, and we assume that all vehicles have equal capabilities (e.g. equal capacity, same speed).

Once a company receives a new request, it checks whether it is possible to insert this request into one of its vehicle schedules in any possible way (see Figure 4.1(b)). This is done by trying to insert both the pickup node and

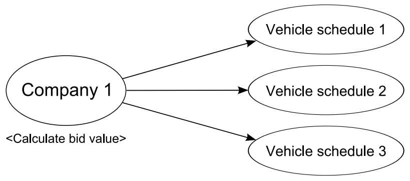
[1]In a real-life setting, a customer does not directly announce a request to all companies, but it announces this request to a call center. This call center can check whether this request is valid, and can then forward the request to all known companies. With the assumptions we make in this thesis, there is no difference in the outcome when using a call center instead of using a direct negotiation between the customer and the companies.



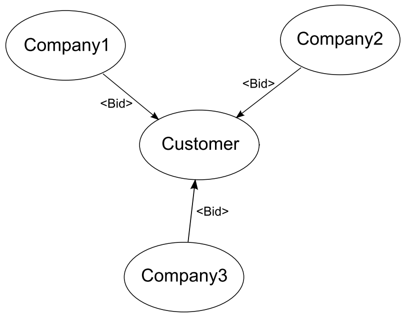
(a) Customer announces request to all known companies.



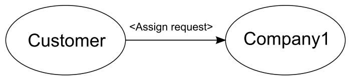
(b) Each company checks whether the incoming request can be inserted into one of its current vehicle schedules and possible future requests.



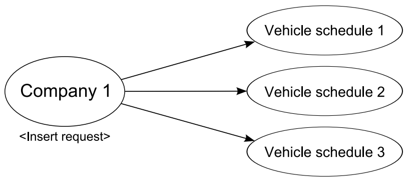
(c) Each company calculates the bid value that it will announce, taking into account its current vehicle schedules.



(d) Each company announces its bid to the customer.



(e) Customer assigns the request to the company that announces the best bid.



(f) The company to which the customer has assigned the request, inserts the request into one of its vehicle schedules.

Figure 4.1: The whole process of the assignment of a request to a company.

the delivery node into the current schedule using a heuristic that is described in Section 4.5.1. If it is not possible to insert the request into one of the company's vehicle schedules, the company cannot serve the request at all, and it will not place a bid in the current auction.[2] If it is possible to insert the incoming request into one the company's vehicle schedules, a bid value has to be calculated for this request (see Figure 4.1(c)). This is done in the way that

[2]In our implementation a company announces a bid with a negative bid value in order to let the customer know that the company cannot serve the request.

is discussed in Section 4.5.

When a customer has received all bids (Figure 4.1(d)), it can determine the best bid (e.g. highest service quality, lowest costs) and the conditions that have to be met by the winning company in serving the request. A message is sent to the winning company, that it must serve the request with these determined conditions, and all other companies are sent a message that they have not win the auction.

The company to which the request is assigned has to insert the request into one of its vehicle schedules (Figure 4.1(f)). This is done by an on-line optimization technique which is further handled in Section 4.4.

To be clear about the properties of the environment in which the algorithm operates, we make the following assumptions:

- Travel times between two locations are constant. This implies that a ride between these locations always takes the same amount of time, whatever how busy it is.
- Vehicles are always driving according to their schedule; they cannot be late and they always show up on a location.
- Vehicles are always available; there are no breakdowns.
- One request is auctioned at a time, so a company cannot delay proposing an offer to wait for future requests.
- All requests made by customers are valid in the sense that if there are no other requests in the system, this request can be served according to the constraints of the model.

In the next sections, all the elements of the process described above are discussed in more detail, starting with the type of the bids that are announced by the companies.

4.2 Bidding service quality

Different types of bids can be used, from which the customer that initiated the request has to decide upon the winning company. For example, a bid value can denote the price that has to be paid for serving a request, or it can denote the profit a company can make by serving the request. Actually, what is done in similar work on the assignment of transportation requests is to bid the additional costs needed to serve the request [11, 27]. To minimize overall costs, the request is assigned to the vehicle that has announced the bid with the lowest additional costs. The settings in which this approach is used are single-company settings or settings in which companies cooperate, with the objective to minimize costs. This is opposed to our competitive, multi-company setting, in which we want to maximize service quality.

Because we strive for high average service quality, intuitively, we want to let the companies compete on the service quality for an incoming request. Therefore, the bid value in our setting contains the service quality that a company promises to provide. The customer can then determine the company that can promise the highest service quality, and so companies need to bid a service quality as high as possible to win the auction.

Different definitions of service quality exist, which can be categorized into two types: technical quality and functional quality [16]. Technical quality is defined as what a customer receives as a result of its interaction with the company, and functional quality is defined as how the outcome a customer receives is obtained, i.e. the process itself. In the transportation domain technical quality often contains measures like the difference between actual and desired delivery time, waiting time during the ride, maximum ride time, and the ratio of actual ride time on direct ride time [33]. Functional quality elements contain measures like the type and amount of information given to a customer, the comfort of the service, and the way in which reservations can be made.

For our research we define a technical service quality, because we are mainly interested in the final outcome of the mechanism, and less in how this outcome is produced. A measure that is understandable by both customers and companies is the ratio of the actual ride time on the direct ride time. For example, when the time to travel from A to B directly (i.e. no detours) is 5 minutes, and the vehicle drives from A to B via C, in 7 minutes, then the service quality is 5/7. One can see that the service quality converges to 1, the less detours are taken. This means that the higher the ratio, the better the service quality. For customers, this measure emphasizes one of their biggest complaints, namely large detours. For companies, this ratio is a measure of how efficiently different rides are combined. Another advantage of defining service quality as this ratio is that it is directly influenced by the scheduling process, allowing us to use scheduling techniques to increase the ratio.

In our approach the service times at both the pickup and the delivery location are included in the travel time. We assume that a plan is a sequence of locations with associated departure times. If the time at which service starts at location i is denoted by u_i, service duration at location i is denoted by d_i, and the time needed to drive from location i to location j is denoted by t_{ij}, then the service quality for a request with pickup location i and delivery location j can be calculated as follows:

$$SQ_{ij} = \frac{t_{ij} + d_i + d_j}{(u_j + d_j) - (u_i)} \tag{4.1}$$

Many complaints of customers in this domain are about the large detours (i.e. low ratio) that vehicles make, which implies longer travel times. By letting companies compete on this service quality measure, we hope to decrease the average travel time of customers. How this competition can be added to the system is discussed in the next section.

4.3 Auction type

In this section we discuss the type of auction that is used in our request assignment mechanism. This auction type determines the rules to be followed by the companies, the payments to be made, and the allocation function.

4.3.1 Rules and allocation

The mechanism that we propose is based on a *reversed sealed-bid second-price auction*. In such an auction, all bids are private to the company that announces it, and the winner of the auction has to pay the second-highest bid value that is announced.[3]

We call the auction *reversed*, because there are multiple sellers (the companies) and a single buyer (the customer). This single buyer announces to all the sellers what exactly it wants to buy by communicating the characteristics of its request. Once the companies are informed about this request they can announce their type by determining a bid value. As is described in the previous section, service quality is used as a bid value.

The winning company is the company that announces the highest service quality, and if multiple companies announce the same highest bid, one of these companies is arbitrarily selected as winner. The request that has been auctioned is allocated to the winning company, which now has to serve the request.

4.3.2 Payments

The payments that are used in our auction are a little bit different from the usual setting. Before our auction starts, a payment that the customer must made to the winning company is defined. This payment is always equal to the price per kilometer that is determined multiplied by the direct distance between the pickup and the delivery location of the customer's request. The company that wins the auction, has to do something back to the customer; serving the request. This can be seen as a "payment" of the company to the customer.

We first discuss customer payments, and then briefly describe the payment of the company.

[3]We speak of "paying" here, but actually, in our context, this is not the correct term. Rather than paying something to the customer, the winning company has to serve a request with at least the second-highest quality. The term "paying" is used, because we want to use auction terminology.

Customer payments

The price per kilometer is fixed for a certain type of request and is determined before the auction takes place, resulting in the fact that the total payment made by a customer does not depend on the company that actually serves the request. The price paid by the customer to the winning company can be seen as a negative payment made by the winning company. Note also, that this payment is independent from a company's own bid, since it is determined before the auction start and because it is fixed.

Companies try to maximize their utility, which in this setting is their *profit*. Profit is defined as the total *income* a company receives from serving requests minus the total *costs* needed to serve these requests. The income is dependent on the price that customers have to pay to travel from their pickup location to their delivery location. In this thesis we assume that the price a customer has to pay for its ride, is linear to the distance between its pickup location and its delivery location. The reason for this is that it is always clear for a customer what the total price of its ride will be.[4] It is also possible to work with dynamic prices (e.g. dependent on the difficulty of the ride) but this would create a very disordered situation from a customer's point of view.

To be able to calculate the income of a served request (i.e. the price a customer has to pay), the price per kilometer, C_{km}, has to be defined. We make two intuitive observations:

1. When C_{km} is too low, a company serving the request can never make profit, because it cannot combine any request in such a way that costs are lower than income.

2. When C_{km} is too high, companies will bid 1.0 for every request, because they can make profit for this request, even without combining it, and by bidding 1.0 they have the highest chance to win the auction. The problem is that in this case, all requests have to be served with service quality 1.0, and that this results in the fact that less requests can be served.

The question arises, when is C_{km} too low and when is C_{km} too high? It seems to be the case that these bounds can be defined.

The lower bound is the lowest price per kilometer that can be paid by the customers, such that the companies make a zero profit. The lower bound can be calculated by dividing the minimal total costs needed to serve all requests by the total direct distance traveled by all customers. This means that for this price, a company is sure that he will make zero profit. For lower prices per distance, the company can never make profit, and it is not worth it to serve requests. So, if the minimal total costs are denoted by $TC_{\min}$, and the

[4]In the current situation in the Netherlands, prices are also fixed.

distance between the pickup location and the delivery location of customer i is denoted by D_i, with N customers, then:

$$C_{km} \leq \frac{TC_{\min}}{\sum_{i=1}^{N} D_i}.$$

The higher bound, C_{km} can be computed in a similar way as the lower bound. Again we search for the point where the company makes zero profit, but now assuming that all requests are served with service quality 1.0. Every vehicle is driving from the depot to the pickup location, to the delivery location, and back to the depot before serving the next vehicle (i.e. each vehicle schedule contains only one request). This way, the maximal total costs $TC_{\max}$ can be determined. Similar to computing the lower bound, these maximal total costs are divided by the total direct distance traveled by all customers, to obtain the upper bound:

$$C_{km} \geq \frac{TC_{\max}}{\sum_{i=1}^{N} D_i}.$$

A higher price per kilometer will not result in higher service quality, but will only result in a higher profit for the company.

Now that the upper and lower bounds of the price per kilometer can be determined, a price that lies between these bounds can be chosen and the profit of a served request can be calculated.

Company payments

The winning company must ensure that the request is served with at least the service quality that is announced by the company with the second-highest bid. This can be seen as a positive payment made by the company, because it loses some of its flexibility to insert and combine rides, resulting in possible higher costs. This positive payment is also independent from a company's own bid. It only depends on the bids of the other companies.

4.4 Insertion into schedule

After a request is assigned to one of the companies, the winning company has to actually insert the request into one of its current vehicle schedules. We also use the insertion of requests for the calculation of bid values, but this is done in another way (see Section 4.5.2). For the purpose of inserting requests after these are assigned to a company, we use the mathematical model of Cordeau and Laporte [6], which is already discussed in Chapter 2. This model is formulated as a Mixed Integer Program (MIP) with an optimization function that minimizes costs. The basic MIP is shown in Figure 2.1 and we add some additional constraints to allow for promising service quality and to speed up the solution process.

To ensure that requests are served with a service quality higher or equal than the promised service quality, we add an extra constraint to the MIP. The promised service quality between locations i and j is denoted by SQ_{ij}. We can add a constraint for a request from i to j to the MIP formulation as is shown in Equation 4.2. This equation is made linear as shown in Equation 4.3.

$$\frac{t_{ij} + d_i + d_j}{(u_j^k + d_j) - (u_i^k)} \geq \mathrm{SQ}_{ij} \qquad (i \in P, j \in D, k \in K) \tag{4.2}$$

$$(u_j^k + d_j) - (u_i^k) \leq \frac{1}{\mathrm{SQ}_{ij}}(t_{ij} + d_i + d_j) \quad (i \in P, j \in D, k \in K) \tag{4.3}$$

It is already known that the request can be inserted (i.e. the MIP is feasible), and the company tries to find a schedule for all assigned requests that have not been served yet, by solving the MIP. It is not needed to incorporate requests that have already been served, because the departure times at the corresponding locations cannot be changed anymore. It is sufficient to incorporate for every vehicle the delivery location of the last request that has been completely served by that vehicle, together with all the locations of requests that have not been served completely yet.

Departure times of locations that have already been visited, and departure times of delivery locations of which the corresponding pickup location have already been visited, cannot be changed anymore. It is clear that the departure times of locations that have been visited cannot be changed anymore, but the second statement is not so straightforward. Departure times of delivery locations of which the corresponding pickup location have already been visited cannot be changed anymore to be able to give the customer information about the time it will arrive at its destination. The same concept is used for the vehicle that is assigned to a location. This way we avoid modifying the schedule a lot once a customer is pickup up. It is also possible to make locations definite earlier than the time the pickup location is visited. This makes it possible to give the customer information about its pickup earlier, but inserting requests and minimizing costs becomes less flexible.

The following constraints can be added to the MIP to handle definite departure times and definite vehicle assignments. The constraint for definite departure times is given in Equation 4.4 and in Equations 4.5 and 4.6 the constraints for the definite vehicle are given. In the constraints, L_{def} denotes the set of definite nodes, k_i denotes the vehicle to which location i is already assigned, and u_i denotes the departure time that is definite for location i:

$$u_i^k = u_i \qquad (i \in L_{\mathrm{def}}, k \in K) \tag{4.4}$$

$$\sum_{j \in L} x_{ij}^{k_i} = 1 \ (i \in L_{\mathrm{def}}) \tag{4.5}$$

$$\sum_{j \in L} x_{ji}^{k_i} = 1 \ (i \in L_{\mathrm{def}}) \tag{4.6}$$

By optimizing a company's vehicle schedules every time a request is assigned to the company, an on-line optimization technique is used. Because for every optimization not all the locations are taken into account, performance and solution quality are increased. It is also possible to further decrease the computation time, by collecting requests to insert, and generate solutions every once in a while. Because we want to be as up-to-date as possible, we choose to generate a solution every time a new request comes in.

4.5 Bid calculation

Now that the type of the bids that are announced is determined, the auction type is known, and it is known how an assigned requests is inserted in the final schedule, a method to actually calculate the bids is needed. Before a company starts to calculate a proper bid value, it is wise to first check whether the announced request can be inserted into one of its schedules. For this check a heuristic is used, which is described in Section 4.5.1. If this is not possible, the company does not have to bid, or it can bid a negative value, indicating that it cannot serve the request. If it is possible to insert the request, the company can start to calculate a proper bid value. In Section 4.5.2, we explain why bidding is not straightforward in our setting and we propose a way to include some knowledge about future requests into the current bid.

4.5.1 Insertion check

Before a bid value is calculated, it is wise to check whether the incoming request can be inserted at all. This can save expensive computation time. The heuristic that is used for this check is based on insertion heuristics developed by Jaw et al. [17] and Solomon [40], and is shown in Algorithm 4.1. Rather than searching for a best place to insert the request, our algorithm stops when an insertion of the request is possible.

The function IsFeasible determines whether or not a node can be inserted into a schedule at a specified location. This function is further discussed in Section 4.5.3, in which it is used in an approach to estimate expected profit. For now, it is only important to note that this function has time complexity $O(N)$, with N the number of locations that already exist in the schedules. In the worst case, the locations can be inserted at the end of the last schedule. That means that the function IsFeasible is called $(N+1)$ times for the pickup location. The delivery location can only be inserted after the pickup location, so the further we proceed in a schedule, the less checks have to made for the delivery location per check of the pickup location. Actually, the total number of checks for the delivery location is $\frac{(N+1)(N+2)}{2}$, so in total the function IsFeasible is called $(N+1)+\frac{(N+1)(N+2)}{2}$ times. Because each check has time complexity $O(N)$, the result is that the whole algorithm has a time complexity of $O(\frac{1}{2}N^3 + 2\frac{1}{2}N^2 + 2N)$.

Algorithm 4.1: Check whether a request can be inserted into one of the vehicle schedules.

```
input  : The request to insert $r$, and the set of current vehicle
         schedules $S$
output: Whether $r$ can be inserted into one of the vehicle schedules

forall $s \in S$ do
    forall $i \in s$ do
        $p_r \leftarrow$ pickup node of r
        $d_r \leftarrow$ delivery node of r
        if IsFeasible($p_r$, $i$) then
            forall $j \in s$ where $j \geq i$ do
                if IsFeasible($d_r$, $j$) then
                    return True
                end
            end
        end
    end
end
return False
```

4.5.2 Future requests

Remember from the previous chapter, that bidding in a sequential auction is not as straightforward as in a single independent auction. The outcome of the current auction can influence the outcome of auctions of future requests and can therefore influence the total profit of a company after a whole day.

Let us assume a setting with two myopic companies. These companies both calculate and announce the best possible service quality they can provide for an incoming request. The value of the winning bid of the very first request will be 1.0, and because to none of the companies requests have been assigned, the second-highest bid will also be 1.0. This means that this request can never be combined with future requests, other than inserting future pickup nodes before the pickup node of the current request and inserting future delivery nodes after the current delivery node. The losing company of the first auction can again bid a service quality of 1.0 for the second request to be auctioned, but for the winning company there are three possibilities:

- bid 1.0 if it is possible to insert the second request as a direct ride,

- bid a service quality < 1.0 if it is possible to insert the request by inserting the pickup node before the pickup node of the first request and inserting the delivery node after the delivery node of the first request,
- or bid nothing if none of the above insertions is possible.

If the company bids 1.0, then the same as with the first request happens; the request is inserted as a direct ride by either the first or the second company. If the company bids a service quality < 1.0, then the losing company of the first request can insert the second request with a service quality < 1.0, meaning that for upcoming requests, it has more possibilities to insert these (e.g. nodes can now also be inserted between the pickup and delivery nodes of earlier requests). The same happens when the winning company of the first request does not bid at all, because then the other company can insert the request with service quality 0.

The above sequence is repeated for every next request that is auctioned, resulting in the observation that once a company has to insert a request with service quality 1.0, it has less chance to win successive auctions of request that could have been combined with the current request.

The previous example shows that it is important for a company to incorporate knowledge about future requests into the calculation of the bid value for the current request. A company wants to maximize its profit (minimize costs) and there are different costs associated with the different service quality values it can bid. The bid a company announces does not have to be its best bid in terms of service quality. It can bid a low service quality for low internal costs, or a high service quality for higher internal costs. Since the payments of the customers are fixed (linear to direct distance), the company must decide what bid value to announce.

The problem for the company now becomes one of announcing the best bid. A company wants to optimize its profit, which is defined as income minus costs. It can gain income by serving requests (winning auctions) and it can decrease costs by combining requests. This combining of requests often leads to a lower service quality (because there are less direct rides), which can lead to winning less auctions. Promising a higher service quality results in a higher probability to win the auction, but decreases the flexibility to insert future requests.

For this reason (i.e. to incorporate the expected profit of future requests), the companies must have some knowledge about the distribution of the time and space components of future requests. From this distribution they can calculate the expected profit for an incoming request, based on future requests that can give the companies possibilities to combine rides and lower costs.

The risk of future requests not coming in according to the expectation can be fully covered by the companies, or can be distributed among the companies and the customers (or call center) by using options [15, 19]. When all the risk lies with the companies, the quality of the expectation is very important.

If they expect requests that can be well combined with currently inserted requests in the future, but these do not come, companies can make a loss. When using options, the risk is shared by the companies and the customers. Options are rights to serve a requests, but no obligations. So, a company receives an option when winning the auction, and it can exercise this option if the expected future requests really come in. If they do not come in, the company can discard the option, and the request has to be re-auctioned. A disadvantage of using options is that problems can arise when a company decides to not exercise an option just before the corresponding request has to be served. There must be enough time to re-auction the request. In this thesis, we assume that all the risk of future requests lies with the companies and therefore do not use options.

Another way to incorporate the expectation of future requests is to use combinatorial auctions [32, 37]. With these type of auctions, the assignment of incoming requests is postponed until the last possible moment, such that future requests up until that moment can be taken into account when calculating the costs for that single request. The problem is that this leads to an exponential amount of combinations to calculate and therefore this can take an exponential amount of time. Another problem is that agents must be able to bid on combinations of requests (or at least must be able to value combinations of requests). These problems make us to decide that we do not use this technique, but a slightly different one.

Rather than the techniques mentioned above, a simple and fast approach to estimate the best bid is proposed. A so called Monte Carlo [29] simulation is used in combination with an insertion heuristic. This approach is discussed next.

4.5.3 Estimating expected profit

The idea of this approach is to estimate the *expected profit* that a company makes in the future assuming that the current request is inserted into the schedule according to some conditions. The conditions that have to be met are a specified level of service quality and all the conditions of the original problem (e.g. time window constraints, capacity constraints). To calculate the expected profit, a distribution has to be known on the arrival time of future requests, the space components of these requests, and the time components of these requests. With the help of these distributions, a set of possible future requests can be generated and inserted into the schedule. Once this is done, the total costs needed to insert these requests, and the total income gained by inserting these requests, can be calculated. This results in an expected profit gained by inserting the future requests, assumed that the stated conditions of the current request hold.

Algorithm 4.2: Calculate the expected profit of inserting a request

```
input  : The current request $r_{\text{current}}$, the set of current vehicle
         schedules $S$, and the service quality value that must hold for
         the current request $q_{\text{current}}$
output: The expected profit gained from serving all inserted requests
R ← set of generated, unassigned requests; current request included;
sorted on ascending arrival time
$q_{r_{\text{current}}} \leftarrow q_{\text{current}}$
R ← R $\cup r_{\text{current}}$
$u \leftarrow 0$
forall $r \in R$ do
    $c* \leftarrow \infty$
    forall $s \in S$ do
        forall $i \in s$ do
            $p_r \leftarrow$ pickup node of r
            $d_r \leftarrow$ delivery node of r
            if IsFeasible($p_r$, $i$) then
                forall $j \in s$ where $j \geq i$ do
                    if IsFeasible($d_r$, $j$) and ServiceQuality($r$) $\geq q_r$
                    then
                        if Costs($p_r$, $i$, $d_r$, $j$) $< c*$ then
                            $c* \leftarrow$ Costs($p_r$, $i$, $d_r$, $j$)
                            $i* \leftarrow i$
                            $j* \leftarrow j$
                        end
                    end
                end
            end
        end
    end
    Insert($p_r$, $i*$, $d_r$, $j*$)
    $u \leftarrow u +$ (Income($p_r$, $d_r$) - $c*$)
end
return $u$
```

Insertion heuristic

In Algorithm 4.2 it is shown how the expected profit is calculated by using an insertion heuristic based on those developed by Jaw et al. [17] and Solomon [40]. The original algorithm is modified such that is capable of inserting both a pickup and delivery node, and that it accounts for service quality. Although we can use the MIP that was defined before for the insertion of requests, we choose for a faster algorithm. The disadvantage of an insertion heuristic (no optimal results) does not matter here so much, because the results are averaged in the Monte Carlo approach.

The input of our algorithm consists of the current request, r_{current}, for which a bid value has to be determined, the set of current vehicle schedules S, and the service quality q_{current} that must hold for r_{current}. A vehicle schedule $s \in S$ is a list of nodes that the vehicle will visit, sorted on node departure time. Multiple future requests are generated that are inserted after r_{current}. These future requests are inserted conditioned on the service quality q_{current} that must hold for r_{current}.

First, it is checked whether or not it is possible to insert the pickup node of a request before a node i that is already in the schedule. If that is possible, it is checked whether or not the delivery node can be inserted before a node j that is already in the schedule. The locations at which the pickup and delivery nodes can be inserted for the lowest costs are saved, and after all locations are checked, these nodes are inserted into the schedule at these locations. The procedures IsFeasible and Insert are discussed next.

Feasibility check

The feasibility checks that appear in the heuristic are shown in Algorithm 4.3. For a node i we denote by e_i the earliest time to visit this node, by l_i the latest time to visit this node and by q_i the service quality that has to be met for the request this node belongs to. Further, the time that a vehicle depart from a node i is denoted by d_i, and the time that a vehicle can arrive at node i is denoted by a_i. The procedure TravelTime(i, j) calculates the time needed to travel between node i and j, and the procedure ServiceQuality$(i, \text{pushForward})$ calculates the service quality for the request node i belongs to, after the departure time at node i is pushed forward. Both procedures have time complexity $O(1)$. Calculating the amount of time succeeding nodes needs to be pushed forward can also be done in constant time, and this has to be done for at most $N + 1$ times, with N the number of nodes in the schedule. This results in a time complexity of $O(N + 1)$ for Algorithm 4.3.

Algorithm 4.3: Check whether or not a node can be inserted

```
input : The node to insert $n$, the node before which $n$ has to be
        inserted, $i$, and the schedule in which the new node has to be
        tested $s$
output: True if and only if it is possible to insert the new node in the
        schedule at the specified position
1  $d_n \leftarrow \max(\texttt{TravelTime}(n, (i-1)), e_n)$
2  if $d_n > l_n$ then
3      return False
4  end
5  forall $j \in s$ where $j \geq i$ do
6      if $j == i$ then
7          pushForward $\leftarrow \max(d_n + \texttt{TravelTime}(n, j), e_j) - d_j$
8      else
9          pushForward $\leftarrow \max(0, \text{pushForward} - (d_j - a_j))$
10     end
11     if pushForward $== 0$ then
12         return True
13     else if $d_j + \text{pushForward} > l_j$ or
       $\texttt{ServiceQuality}(j, \text{pushForward}) < q_j$ then
14         return False
15     end
16 end
17 return True
```

Insertion

During the insertion process, the departure times of the pickup and delivery nodes are chosen in a way to maximize the service quality for that request. This is done by departing as late as possible from the pickup node, and as early as possible from the delivery node (i.e. decreasing the travel time for this request). The insertion process is shown in Algorithm 4.4. Again, the procedure TravelTime(i, j) calculates the travel time between node i and node j in constant time. If there is a negative push forward for the first node after the pickup node p, this means that there is some slack time available. This slack time is used to decrease the time between the pickup and delivery departure time of the request that is currently inserted.

The time of complexity of the insertion of a pickup node and a delivery node is $O(2N)$, because the whole schedule of N nodes has to be traversed at most two times.

Algorithm 4.4: Insert pickup and delivery node in schedule

input : The pickup node p and delivery d node to insert, the node i before which p has to be inserted, the node j before which d has to be inserted, and the schedule s

output: A schedule s in which the pickup and delivery nodes are inserted

```
1  $d_p \leftarrow \max(\texttt{TravelTime}(p, (i-1)), e_p)$
2  pushForward $\leftarrow \max(d_p + \texttt{TravelTime}(p, i), e_i) - d_i$
3  if pushForward $< 0$ then
4      $d_p \leftarrow \min(d_p + 0 - \text{pushForward}, l_p)$
5  else
6      forall $k \in s$ where $k \geq i$ do
7          $d_k \leftarrow d_k + \text{pushForward}$
8          pushForward $\leftarrow \max(0, \text{pushForward} - (d_k - a_k))$
9      end
10 end
11 $s \leftarrow s \cup p$
12 $d_d \leftarrow \max(\texttt{TravelTime}(d, (j-1)), e_d)$
13 pushForward $\leftarrow \max(d_d + \texttt{TravelTime}(d, j), e_j) - d_j$
14 forall $l \in s$ where $l \geq j$ do
15     $d_l \leftarrow d_l + \text{pushForward}$
16     pushForward $\leftarrow \max(0, \text{pushForward} - (d_l - a_l))$
17 end
18 $s \leftarrow s \cup d$
```

Monte Carlo simulations

The specific set of generated future requests can have a big influence on the calculated expected profit. Therefore, a Monte Carlo simulation [29] is performed. The above algorithm is repeated a significant number of times, and the final expected profit is taken as the average expected profit of these repetitions.

To obtain the highest service quality level for the current request, taking into account future requests, Monte Carlo simulations are performed for different levels of service quality. The level for which the expected profit is closest to zero is taken as the bid value. This procedure is shown in Algorithm 4.5.

4.6 Summary

In this chapter a mechanism is proposed in which multiple companies compete on service quality for the assignment of transportation requests. This requests are announced by customers that want to travel between two locations, and

Algorithm 4.5: Calculate the service quality for which the company makes a zero profit.

```
input  : The current request $r_{\text{current}}$, the set of current vehicle
         schedules $S$
output: The service quality level for which the expected profit is
         closest to zero

closestEP ← 0
foreach quality level to test do
    totalEP ← 0
    repeat
        totalEP ← totalEP + CalculateExpectedProfit()
    until all simulations done
    averageEP ← totalEP/ number of simulations
    if Absolute(0− averageEP) ≤ closestEP then
        closestEP ← Absolute(0− averageEP)
        bestSQ ← current quality level
    end
end
return bestSQ
```

an auction is used to allocate these requests. In order to not only make profit in the short term, but also in the longer term, possible future requests are taken into account in the calculation of a bid value.

The company that bids the highest service quality is chosen by the customer to serve the request, and gets paid a before-calculated amount. The assigned request must be served with a minimal service quality that is defined by the second-highest bid value. This is ensured by adding an additional constraint to the insertion process of an assigned request into a company's schedule.

In the next chapter experiments are discussed that show the performance of our mechanism in terms of computation time, average service quality, and costs.

Chapter 5

Experiments

In this chapter we provide different experimental results of the use of our mechanism, to be able to get information about the influence of specific elements of the mechanism on measures like service quality and total company costs. A brief overview of our implementation is given in Section 5.1. For a more extensive model of our implementation we refer to Appendix B.

Before the experiments can be performed a set of problem instances is needed that can either be generated or can be obtained from real data. We choose to use a generated set, because obtaining real data is a difficult task. The set that is used is described in Section 5.2.

The experiments themselves are discussed by first stating the hypotheses we want to test, describing the experiment set-up, presenting the results, and finally coming back upon the hypotheses in a conclusion. We start in Section 5.3 by comparing average service quality and total costs for two different bid types, namely a service quality bid and a bid that contains an additional costs value. It is shown that there exists a significant difference between the two settings.

The second experiment is described in Section 5.4 and is about the influence of having knowledge about the distribution of future requests. A comparison is made between a setting in which companies have future knowledge, and a setting in which they have not.

In this thesis, we try to move from a single-company setting to a multi-company setting. In an experiment discussed in Section 5.5 we compare average service quality and total costs for a multi-company setting and a single-company setting like it is used nowadays in the Netherlands. Because this experiment does not give us insight in the reasons behind the differences, another experiment that simulates a minimal service quality level in the single-company setting is described in Section 5.6.

Before we move on to the experiments we start by mentioning some implementation characteristics and describing the problem instances.

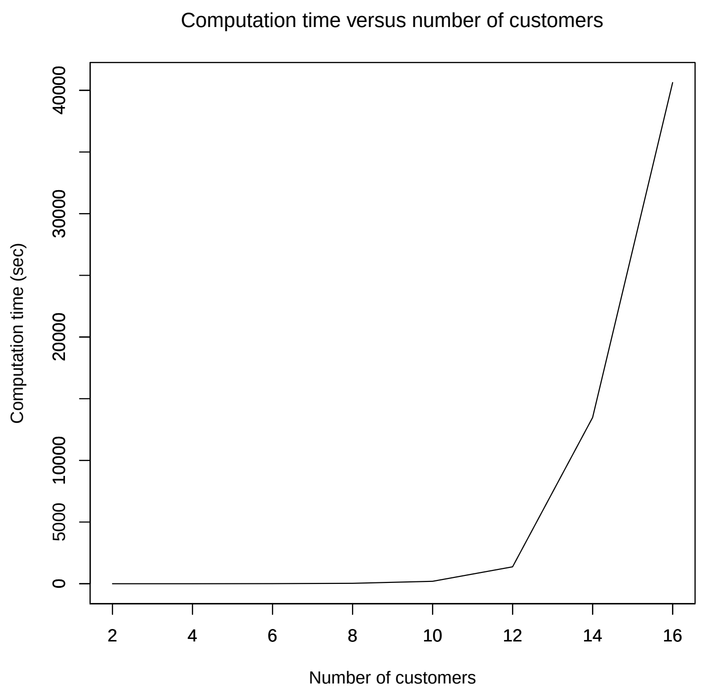


Figure 5.1: Computation time of solving problem instances *a priori* relative to the number of customers.

5.1 Implementation

Our approach is implemented in the Java programming language, and the agents that occur in our setting (i.e. companies and customers) are implemented in the Java Agent DEvelopment Framework (JADE) [1]. This framework simplifies the implementation of multi-agent systems through a middleware that complies with the FIPA specifications [13] for agent communication.

The MIP that must be solved to insert assigned requests into the companies' schedules is solved by the MIP-solver SCIP, which is one of the fastest non-commercial mixed integer programming solvers [14].

All of the experiments are performed on an Intel Xeon E5345 2.33GHz Windows XP 64-bits system with 16 Gb RAM.

5.2 Problem instances

As we mentioned before, we use a set of generated problem instances, instead of instances that are obtained from real data. The reason for this is that there is not much real data available, because either companies does not maintain history files of served requests, or they simply do not want make these files

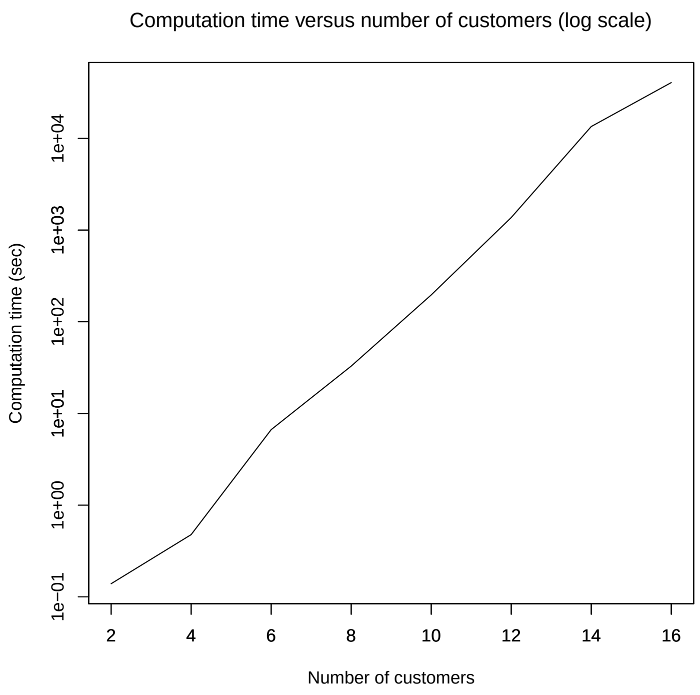


Figure 5.2: Computation time of solving problem instances *a priori* relative to the number of customers, with computation time axis in logarithmic scale.

publicly available. The advantages of generating instances are that a lot of different instances are available, and that we can decide upon the format ourselves. Real data must first be converted into a format that is convenient for our system.

Two sets of problem instances are used. These sets differ on the distribution of incoming requests during a planning period on a specific day. In the first set R_u, the pickup and departure times of the requests are distributed following a uniform distribution, and in the second set R_n these times are spread following a normal distribution. By generating the second set, we try to simulate a busier period (e.g. traffic hour). Test set R_n is used in the experiment to compare settings with and without future knowledge. For the other experiments only test set R_u is used, because there we do not expect different results for these sets.

Both sets of problem instances contain 100 instances of 16 customers. This limited number of customers is used because calculating the upper and lower bounds for the price per kilometer takes exponential time. The lower bound for a problem instance is calculated by solving an MIP as if all requests are known *a priori*. To give an estimate of the time needed to compute these bounds, computation times for instances with different numbers of customers

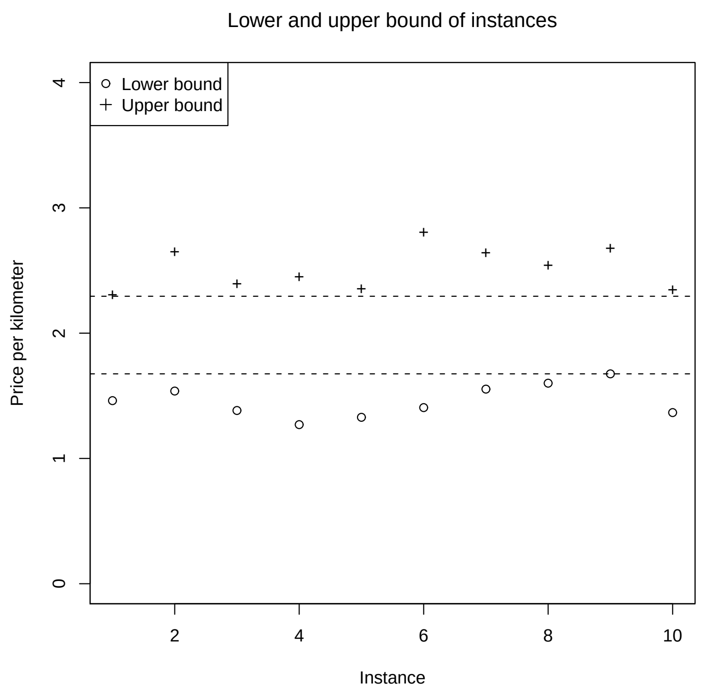


Figure 5.3: Lower and upper bounds of 10 instances, together with bounds that hold for every instance.

are given in Figure 5.1. In Figure 5.2 the axis for computation time is displayed in logarithmic scale to show a straight line that represents the exponential increase in computation time.

Calculating the upper bound is not as computationally difficult as calculating the lower bound, because this bound is calculated as if all requests are served with service quality 1.0 and with a different vehicle. In Figure 5.3 the lower and upper bounds for all instances are shown, together with the lower and upper bound over all instances. A price of 2 units per kilometer is used for the experiments, unless stated otherwise. We refer to Section 4.3 for more details about the price per kilometer.

The planning period that is considered for each instance starts at 10h00 and ends at 14h00. This implies that the earliest time a vehicle can leave its depot is 10h00 and the latest time a vehicle has to be back at its vehicle is 14h00. The reason why we choose to use such a relatively short period is that we want to increase the difficulty, because only a limited number of customers is used. The time elements of the requests become closer to each other in the time space, and more combinations are possible.

The coordinates of the pickup and delivery locations are randomly and independently chosen, according to a uniform distribution, in the $[-10, 10] \times$

$[-10, 10]$ square. The start and end depot, from which every vehicle must start and to which every vehicle must end its route, are located at the location $(0, 0)$. The routing costs and the travel time between two locations are equal to the Euclidean distance between the two nodes. We further assume that all time variables are measured in minutes and that every vehicle travels with a constant speed of 1 kilometer per minute.

In an instance with n customers, customers $1, \dots, n/2$ formulate outbound requests (i.e. specify a delivery time window), and users $n/2+1, \dots, n$ formulate inbound requests (i.e. specify a pickup time window). For an outbound customer, a time window for its delivery time is generated by randomly choosing a number in the interval $[660, 810]$ (i.e. between 11h00 and 13h15) as the earliest possible time, and setting the latest possible delivery time 15 minutes later (i.e. a time window of 15 minutes). The interval between 11h00 and 13h15 is used, because a vehicle driving from the depot at 10h00 via a pickup location to a delivery location takes at most 1 hour. Similarly, driving from the last delivery location back to the depot takes at most half an hour, so it must leave the last delivery location before 13h30 (latest possible time). For the generation of test set R_n the mean of the normal distribution is set to 12h00, with a standard deviation of 40 minutes.

For inbound customers, a time window for its pickup time is generated by choosing a number in the interval $[630, 780]$ (i.e. between 10h30 and 13h00) as the earliest possible pickup time. Again, the time window has a length of 15 minutes. The interval between 10h30 and 12h45 is used for a similar reason as with outbound requests. It takes at most half an hour to drive from the starting depot to the first pickup location, so the earliest possible pickup time is 10h30. It takes at most 1 hour to drive from the latest pickup location via the latest delivery location to the end depot, so the latest possible pickup time is 12h45.

To derive dynamic problem instances, we need to add times at which customers announce their request. This is done by randomly choosing a number in the interval $[e_i - 90, e_i - 60]$ (i.e. between 90 and 60 minutes before the earliest pickup or delivery time). We choose these values, because we want the instances as dynamic as possible. This means that customers announce their requests quite late, but such that vehicles are able to respond to schedule changes.

The maximum capacity for each vehicle is 3 passengers and the demand from each customer is 1. The service duration is equal to the demand and is set to 1 minute. The maximum route duration is 240 minutes, which means that each vehicle must be returned to its end depot before 14h00. The maximum ride time per customer is set to 30 minutes.

5.3 Comparing bidding service quality to bidding additional costs

We have developed an auction mechanism in which the bids that companies announce do not contain a money element, but do contain a measure of service quality. It is good to know what influence this bid type has on the final outcome of the assignment of requests. From the research question that is stated in Chapter 1, the following specific question is derived: *"What is the difference in total costs and in average service quality between an auction where a bid contains the price of a request, and an auction where a bid contains the promised service quality?"*.

The costs of serving multiple requests depend on the way how these requests are combined. By combining requests (i.e. not driving directly from pickup to delivery location, but first pickup or deliver some other passengers) a company can cut costs, but the service quality of individual requests will decrease because of detours. For that reason, if customers can choose for the lowest costs, the costs for companies will be low and service quality will be low. The other way around, if customers can choose for the highest service quality, costs for the companies will be higher, and service quality will also be higher. Based on this line of reasoning, we define the following hypotheses:

1. The average service quality is higher in an environment where requests are auctioned based on service quality than in environments where requests are auctioned based on costs.

2. The average total costs are higher in an environment where requests are auctioned based on service quality than in environments where requests are auctioned based on costs.

The set-up to test these hypotheses is discussed next.

5.3.1 Set-up

For this experiment we run our algorithm on both test sets with two companies, each having two vehicles with the capacity of transporting customers. The algorithm is run twice for each problem instance. The first time we let companies compete on service quality, and the second time we let them compete on costs. When companies compete on service quality, they calculate their bid values with respect to future requests as discussed in Chapter 4. When companies compete on costs, they announce a bid such that this bid represents the minimal additional costs needed to serve the request. This is a technique that we have seen in earlier work [11, 27].

The company that wins the auction is the company that bids the highest service quality, or the company that bids the lowest costs, respectively. In the first setting, the winning company must serve the request with at least

Instance	SQ_q	SQ_c	TC_q	TC_c	$\#R_q$	$\#R_c$
p_16_4_1	0.96	0.74	378.126	309.953	16	16
p_16_4_2	0.84	0.72	248.533	274.055	15	16
p_16_4_3	0.93	0.73	340.686	272.673	16	16
p_16_4_4	0.90	0.74	366.362	306.678	16	16
p_16_4_5	0.93	0.69	314.486	236.057	16	16
p_16_4_6	0.95	0.62	330.310	284.028	15	16
p_16_4_7	0.98	0.62	358.547	287.354	16	16
p_16_4_8	0.96	0.73	302.308	280.054	16	16
p_16_4_9	1.00	0.79	353.176	275.527	16	16
p_16_4_10	0.83	0.74	259.103	269.487	16	16

Table 5.1: Average service quality and total costs for instances solved by bidding service quality (SQ_q and TC_q) and bidding costs (SQ_c and TC_c). The last two columns contains the number of requests that have been served in both settings.

the service quality of the second-highest bid, and in the second setting, the winning company receives a positive amount equal to the second-lowest bid. In the second setting, there are no restrictions on the insertion of a request.

For all problem instances the average service quality SQ_q is calculated for companies bidding service quality, and the average service quality SQ_c is calculated for the companies bidding costs. Total costs TC_q are calculated for companies bidding service quality, and total costs TC_c are calculated when the companies bid costs. We perform a paired t-test on all SQ_q and SQ_c values to investigate the difference in means of the distributions of values of both settings. Similarly, we perform a paired t-test on all TC_q and TC_c values of both settings. A 95%-confidence interval is used for the mean difference between the two settings for both service quality and costs.

5.3.2 Results

In Table 5.1 we present the results of 10 instances of the experiments done. Observe that there are two instances for which not all of the requests can be served when competing on service quality. This can be explained by the fact that due to a service quality promise, it is not possible anymore to combine rides such that all requests can be served.

It is clear that for all instances the average service quality is higher in the setting where companies compete on service quality than in the setting where they compete on costs. It is also shown in Table 5.1 that for eight problem instances the total company costs are higher in the first setting than in the second setting.

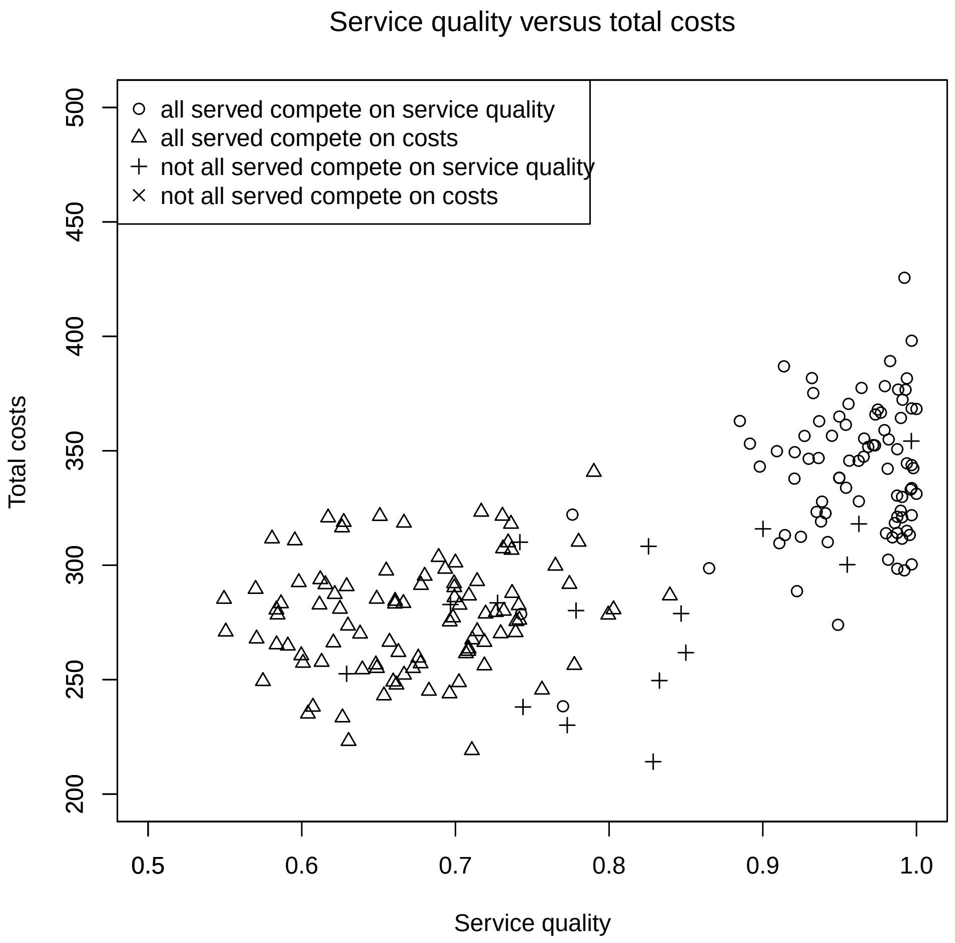


Figure 5.4: Service quality versus total costs for a setting in which companies compete on service quality and a setting in which companies compete on costs.

To give an overview of the results of all the instances, a scatter plot is shown in Figure 5.4, in which the service quality is pointed out against total costs. Two clouds of points can be distinguished; one cloud for each setting. It is also shown that the instances for which not all requests have been served, have lower total costs. This can be explained by the fact that the requests that have not been served, do not increase the total costs; the additional cost for this requests is zero.

For both service quality and total costs, we perform a paired t-test to test whether the means of the two distributions of values are indeed different (i.e. whether the service quality values of both settings are from different distributions, and whether the total costs values of both settings are from different distributions). The result of a paired t-test for service quality gives us a mean difference of 0.262 with a 95%-confidence interval of $[0.242, 0.282]$, between the average service quality in the first setting and the second setting. For the paired t-test of total costs, the result is a mean difference of 52.7 with a 95%-confidence interval $[45.0, 60.4]$.[1] The p-values of both tests are smaller than 2.20×10^{-16}.

5.3.3 Conclusion

Taking our hypotheses mentioned in the beginning of this section, and looking at the results in the previous subsection we have no reason to reject these hypotheses. The p-values of the paired t-tests are very low, implying that the probability to get the same results assuming that respectively the service quality values and the total costs values of both settings come from the same distribution, can be neglected. With this information we come to the conclusion that, on the average, both service quality and total costs are higher in an environment in which companies compete on service quality, than in an environment in which they compete on costs.

5.4 Incorporation of future requests into bid values

A company wants to optimize its profit, which is defined as income minus costs. It can gain income by serving requests (winning auctions) and it can decrease costs by combining requests. This combining of requests often leads to a lower service quality (because there are less direct rides), which can lead to winning less auctions. Promising a higher service quality results in a higher probability to win the auction, but decreases the flexibility to combine future requests with the current request.

To investigate the influence of information about future requests on the average service quality, the total company costs, and the number of requests that can be served, we compare a setting in which companies run Monte Carlo simulations to incorporate future requests in the current bid calculations, with a setting in which the bid is calculated only based on the current situation.

Based on the reasoning above and the research question about bid calculation stated in Chapter 1, the following hypotheses are defined:

1. The average service quality of served requests is lower in a setting where information about future requests is incorporated into the calculation of current bids than in a setting where this information is not incorporated.

[1]The values of instances for which not all requests have been served are also included in the paired t-test for completeness. As shown in the results, this has no influence about our final conclusions, because without these instances average total costs will be even higher in the setting where companies compete on service quality.

2. Average total costs are lower in a setting where information about future requests is incorporated into the calculation of current bids than in a setting where this information is not incorporated.

3. The number of served requests is higher in a setting where information about future requests is incorporated into the calculation of current bids than in a setting where this information is not incorporated.

The set-up for the experiments to test these hypotheses is discussed next.

5.4.1 Set-up

In this experiment we run our algorithm twice for each problem instance. Again, two companies are used, both with two vehicles. The first time, companies calculate their bid value based on their current schedule and the current request only. This means that they do not incorporate their knowledge about future requests, and that they will bid the best service quality they can provide for the request that is being auctioned.

The second time the algorithm is run on a problem instance, the companies have knowledge about the time and space distribution of future requests. To incorporate this knowledge they use the algorithm described in Section 4.5.2. The levels of service quality to test are chosen from $0, 0.1, \ldots, 0.9, 1$ and per level 200 simulations are performed.

For this experiment we use both the test set containing problem instances in which requests arrive following a uniform distribution (R_u), and the test set in which the requests arrive following a normal distribution (R_n).

Paired t-tests are performed for service quality, costs, and the number of served requests, similar to the tests performed in Section 5.3.

5.4.2 Results

In Figure 5.5 and Figure 5.6 a scatter plot is given with the service quality and total costs for all instances. We cannot observe two clouds as clearly as we could in the previous experiment, but we notice that the values for instances tested in the setting with future knowledge are more widely spread than the values for instances tested in the setting without future knowledge.

A noteworthy observation is that there are quite some instances tested in the setting with future knowledge, for which not all requests have been served. An explanation for this result could be the fact that with future knowledge, companies bid more conservative values, resulting in lower service quality promises and more possible future combinations. Because both companies can make more combinations, they both announce bids for future requests, and the service quality promise that is made after the requests has been auctioned is equal to the second-highest bid. When companies have no future knowledge, they bid the highest service quality that is possible for the current request,

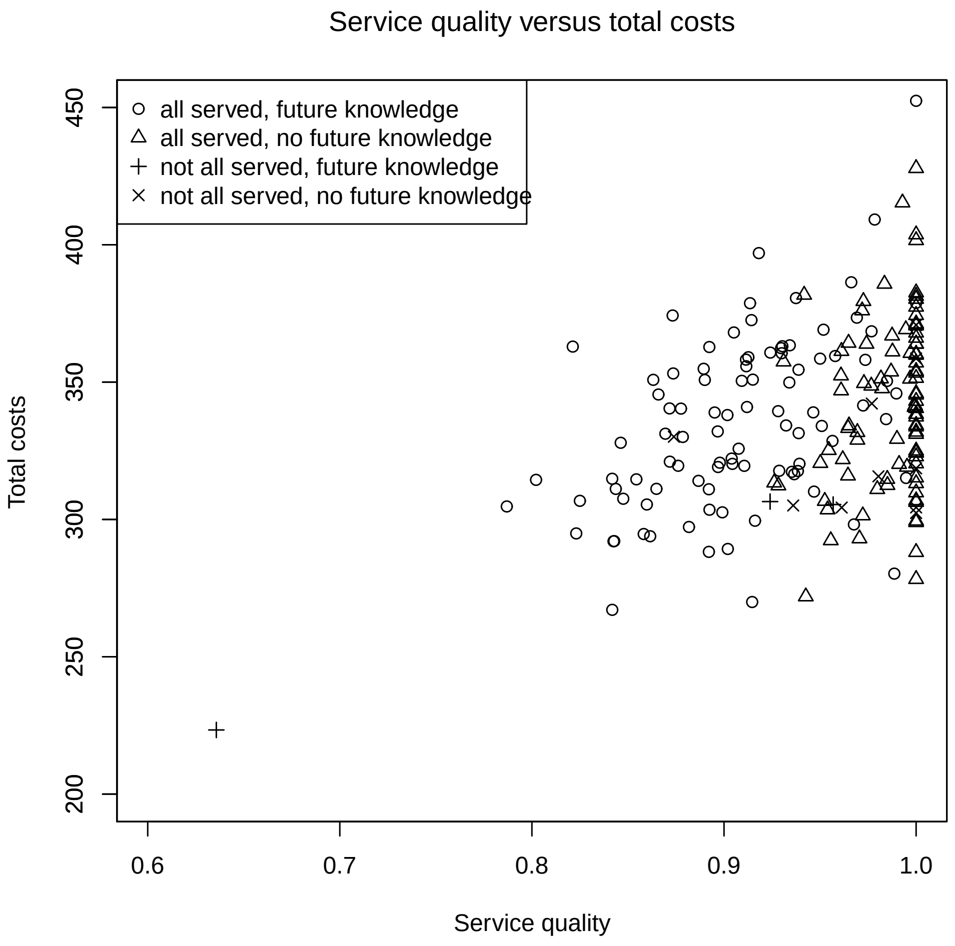


Figure 5.5: Service quality versus total costs for a setting in which companies have knowledge about future requests, and a setting in which they have not. The requests arrive following a uniform distribution.

which will lead to higher average service quality (as shown in Figure 5.5). This results in less possible combinations of future requests for the winning company, probably forcing this company to bid nothing for upcoming requests. When this company bids nothing and the other company does bid something, the other company does not have to make a service quality promise. This leads to more possibilities to insert future requests than in the setting in which companies does have future knowledge.

The result of a paired t-test for the average service quality between the two settings for test set R_u gives us a mean difference of 0.072 between the setting with future knowledge and without future knowledge, where the av-

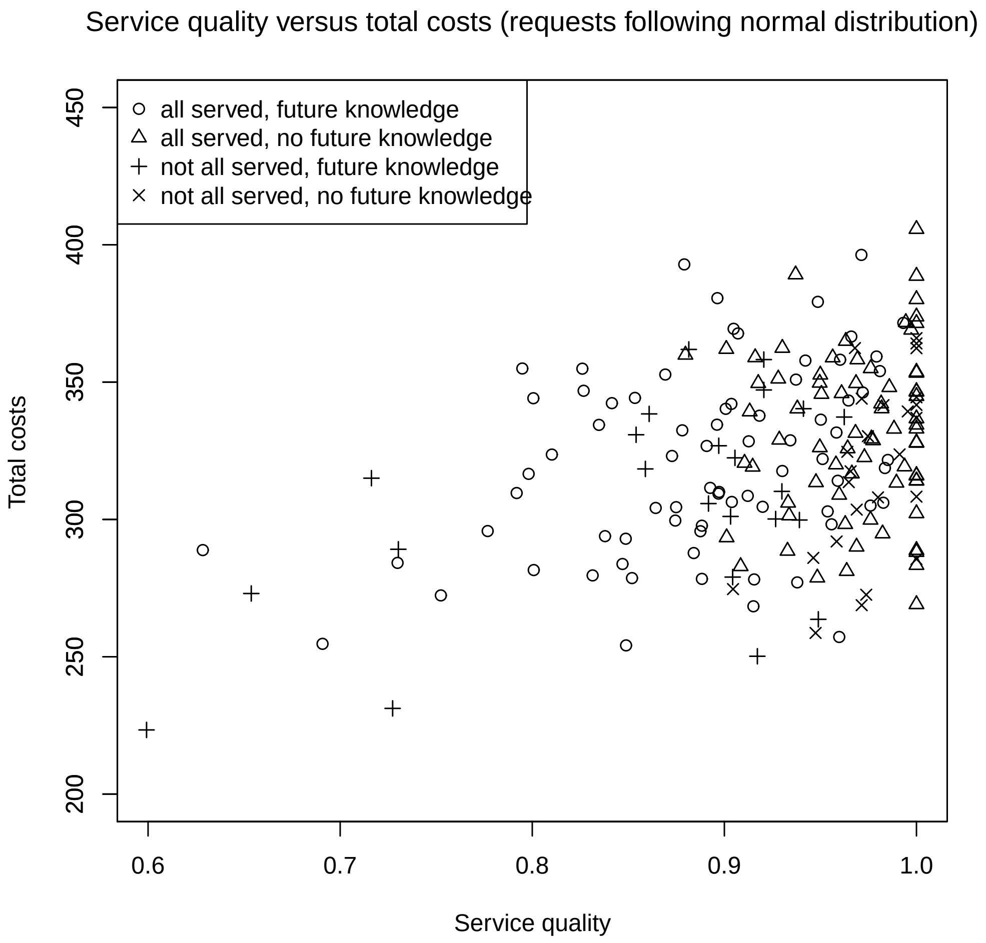


Figure 5.6: Service quality versus total costs for a setting in which companies have knowledge about future requests, and a setting in which they have not. The requests arrive following a normal distribution.

erage quality in the latter setting is higher. The 95%-confidence interval is $[0.057, 0.086]$, with a p-value of 2.27×10^{-16}. The difference in service quality for test set R_n is a bit larger; the mean difference for that test set is 0.091 in the confidence interval $[0.072, 0.110]$, with a p-value of 3.93×10^{-16}.

For the total costs, the paired t-test gives us a mean difference of 7.98, with lower costs in the setting with future knowledge. The p-value is still less than 0.05, namely 1.06×10^{-3}. This means that the probability of obtaining the same test results when we assume that total costs in both settings are equal, is 0.106%. The 95%-confidence interval lies between 3.29 and 12.7. For test set R_n, the difference in total costs is also larger than for the problem instances in R_u, namely 12.65 in the confidence interval $[7.13, 18.2]$ and a p-value of 1.55×10^{-5}. This implies that the total costs are on average 3.8% higher when companies do not take into account their future knowledge.

As noticed before, it seems to be the case that in the setting without future knowledge, more requests can be served, than in the setting with future knowledge. For test set R_n we observe less served requests in both settings, but this can be explained by the fact that during the peak (i.e. traffic hour) not all requests can be served at all with the vehicle capacity that is assumed.

To be able to judge about the hypothesis stated previously, we also perform a paired t-test for the number of served requests in both settings. This test results in the following values for test set R_u: a mean difference of 0.04 with more requests served in the setting without future knowledge, a 95%-confidence interval of $[-0.039, 0.11]$, and a p-value of 3.20×10^{-1}. When using a normal distribution of the arrival of requests, the difference is less significant. The mean difference is only 0.01 with a large p-value of 9.04×10^{-1}.

5.4.3 Conclusion

From the results stated above, we conclude that both service quality and total costs are lower in a setting where companies have some knowledge about the distribution of future requests than in a setting in which they have not. The p-values of both paired t-tests are lower than 5%, implying that the null hypotheses can be rejected.[2] Because the tests show that the means in both settings are different, and that the means of the setting without future knowledge are greater, we can accept both hypotheses.

For test set R_u, the p-value obtained from the paired t-test for the number of served requests is also smaller than 5%, so we can reject the null hypothesis that says that the average number of served requests is equal in both settings. So for R_u, we also conclude that more requests can be served in the setting without future knowledge. This cannot be concluded for R_n, because the p-value is way too high, which indicates that there is a probability of about 90% to obtain the same number of served requests assuming that there is no difference in the number of served requests between the two settings.

When we compare the results for the two test sets, we see that the differences in service quality and total costs are larger for R_n than for R_u. The conclusion is that the type of distribution for the arrival of requests influences the way companies can incorporate their knowledge about future requests. It needs further investigation to conclude about the exact influence.

Following from the results of the tests, we conclude that in a setting with future knowledge average service quality is approximately 8% lower and average total costs are approximately 3% lower than without future knowledge.

[2]Notice that the hypotheses we stated for this experiment are actually alternative hypotheses, and that the null hypotheses state that there is no difference in means. To reject a null hypothesis it is common that the p-value has to be smaller than 5%.

The fact that service quality decreases more than twice as much as the total costs surprises us, because we thought that the two measures would have decreased more equally. An explanation for this could be the fact that we use only two companies in our setting, and that if one company bids a low service quality, the promise that has to be made for the quality of serving the request is also low. The more companies, the probability increases that the second-highest service quality bid is higher than in the two-company setting, and thus the higher the average service quality.

5.5 Multi-company setting versus currently used setting

One of the main ideas in this project is to use multiple companies to serve requests instead of only one company. We want to know what the influence of moving to such a multi-company setting is on total costs and average service quality compared with these measures in the setting that is currently used in the Netherlands.

First of all, a company in a single-company setting can switch vehicles and combine requests until just before the moment that a request has to be served. Of course, after a request is assigned to a company, it cannot be switched to another company anymore, only to other vehicles within the same company. So, in a multi-company setting, a company is less flexible to combine rides and to decrease costs, which implies higher costs.

Because service quality cannot be less than the service quality promised upon assignment and because there is competition between companies, the average level should be higher in a multi-company setting than in a single-company setting. Notice that this reasoning is only valid if the single company announces bids that are calculated by minimizing costs. This is a realistic assumption, because a single company has no incentive to bid high service quality values, and only cares about minimizing its total costs.

We define the following hypotheses based on the reasoning above:

1. Average total costs are higher in a multi-company setting than in the currently used single-company setting.

2. Average service quality is higher in a multi-company setting than in the currently used single-company setting.

To make fair comparisons, the total number of vehicles in both settings is equal. Further experiment characteristics are discussed next.

5.5.1 Set-up

For each problem instance, the algorithm is run two times. The first setting consists of two companies, each having two vehicles and competing on service

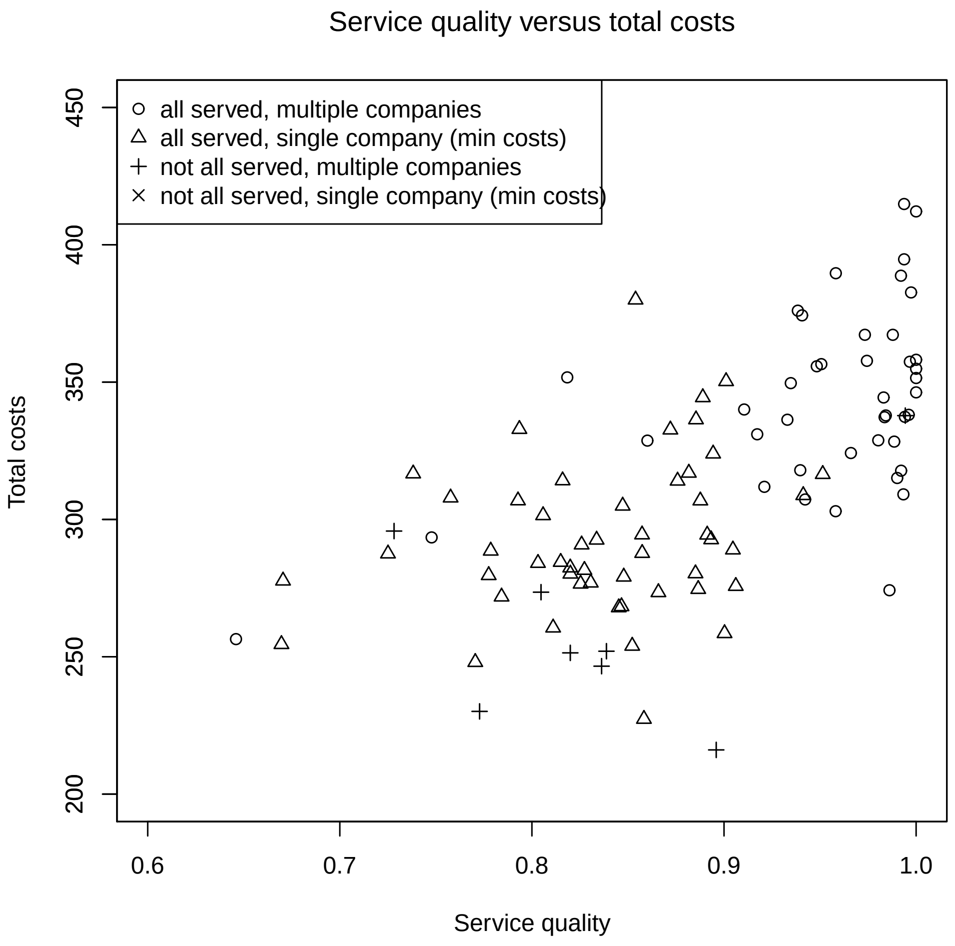


Figure 5.7: Service quality versus total costs for a multiple company setting and a single company setting in which the single company minimizes costs.

quality. They incorporate knowledge about future requests into the calculation of their current bid.

The second time the algorithm is run, there is only a single company, having four vehicles and minimizing costs. In a single-company setting, that company does not have any incentive to bid high service quality, because it knows already that it gets assigned the request.

Paired t-tests are performed once more between the first and the second setting to investigate the differences in values for total costs and service quality.

5.5.2 Results

In Figure 5.7 a plot is given for the comparison of service quality and total costs between the first and the second setting. Two clouds of points can be distinguished, similar to the clouds we found in Section 5.3. The cloud of points of instances with multiple companies is situated with relative high service quality and high total costs. The other cloud of points has less quality and less total costs and mainly contains instances with a single company.

A paired t-test is performed for both average service quality and total costs to test the hypotheses 1 and 2 discussed above. The mean difference for service quality is 0.097 with a higher quality in the multi-company setting. The confidence interval is $[0.071, 0.122]$ and the probability that this results are obtained assuming that there is no difference between the two settings is 5.98×10^{-10}.

For the mean of the total costs for both settings, the t-test gives us a mean difference of 37.5 higher costs for the multi-company setting, with a confidence interval of $[25.3, 49.8]$, and a p-value of 1.25×10^{-7}.

5.5.3 Conclusion

In the t-test on total costs, we discovered that total costs are higher in the multi-company setting than in the single-company setting, namely 13% higher. This is in line with the hypothesis we defined in the introduction of this experiment.

The fact that companies in a multi-company setting have the incentive to bid higher service quality instead of minimizing costs, does result in a higher average service quality. This follows from the results of the paired t-test performed on the values of setting 1 and setting 2, where the mean of setting 2 is 12% lower than that of setting 1, and a p-value such that the null hypothesis can be rejected.

5.6 Single-company setting with minimal service quality

In the previous experiment we compared our multi-company setting to the single-company setting that is currently used in the Netherlands. We assumed that there are no incentives for serving requests with high quality in this single-company setting. The results of that experiment show a significant difference in both service quality and total costs, but the cause of these differences is not very clear yet. The question remains whether these differences occur because of the difference in the number of companies directly, or whether these differences occur because of the incentives to promise higher service quality (which follow from the fact that there are multiple companies).

In this experiment we assume that there is a minimal service quality for which requests need to be served by a company in the single-company setting. This is not unrealistic, because in the real world these minimal service quality conditions are also used, together with punishments given if a company does not meet these conditions. By using such minimal service quality levels, we are able to investigate how the number of companies influences the total costs. By increasing the minimal service quality in the single-company setting, we can compare the costs in both settings with more or less equal service quality.

We define the following hypothesis for the average total costs:

1. The average total costs in a single-company setting in which the company minimizes costs increases when the minimal service quality this company has to provide increases.

2. The average total costs are lower in a single-company setting than in a multi-company setting for equal average service quality.

5.6.1 Set-up

The first hypothesis is tested by running our mechanism with a single company and setting the promised service quality for all requests to a specific minimal level. For this experiment minimal service quality values are used from $\{0.00, 0.05, \ldots, 1.00\}$. To be able to draw a conclusion about the first hypothesis, we set out the minimal service quality level against the average total costs in a plot and calculate the correlation coefficient between these two measures. This way we can conclude whether or not the values are increasing, the higher the minimal service quality.

To compare the single-company setting with the multi-company setting with equal average service quality, we force the company in the single-company setting to derive the same average service quality as it would have done in a multi-company setting. This means that we set a minimal service quality that is equal to the level for which a single company derives the same average service quality as in a multi-company setting. We can get this information from the experiment above, where we investigate the relation between minimal service quality levels and total costs and average service quality. A paired t-test is performed to show the difference in costs between the two settings.

5.6.2 Results

In Figure 5.8 it is shown that the average total costs are increasing as from a service quality level of 0.4, with a decrease at level 1.0. This decrease of average total costs can be declared by the fact that if the minimal service quality level is 1.0, less requests can be served, which leads to lower total costs. The non-increasing part of the figure (from level 0.0 to 0.4) can be declared by the fact that the levels do not influence the outcome, because

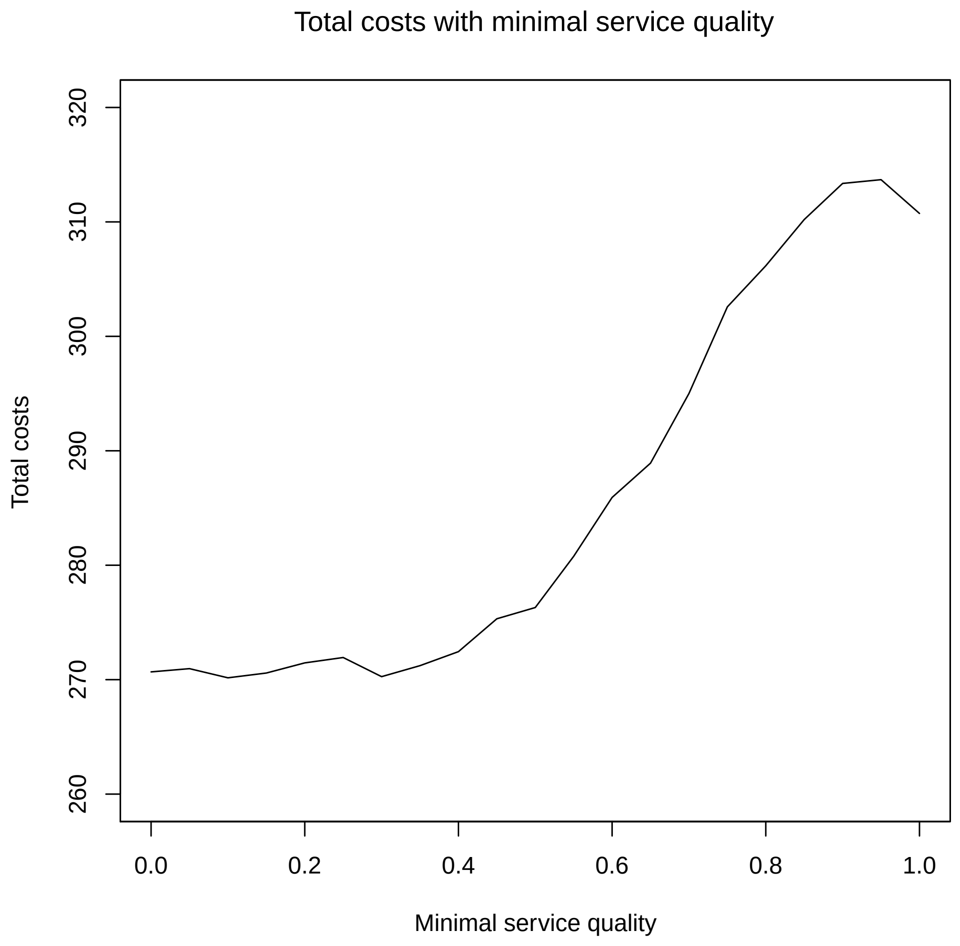


Figure 5.8: Average total costs for instances tested with a minimal service quality level.

even when a single company minimizes costs, it still serves requests with an average service quality of about 0.66. This is also shown in Figure 5.9, in which we see no increase in service quality at this interval.

To show the strong correlation between total costs and minimal service quality, we calculate the correlation coefficient between these two measures. Taken the whole interval of quality levels from 0.0 to 1.0, the correlation coefficient is equal to 0.93. When we take only the interval from 0.40 to 0.95 into account, the coefficient is even higher: 0.99.

The minimal service quality level that is needed in order to let the company in the single-company setting serve requests with an equal average service quality as in the multi-company setting is derived by searching in Figure 5.9

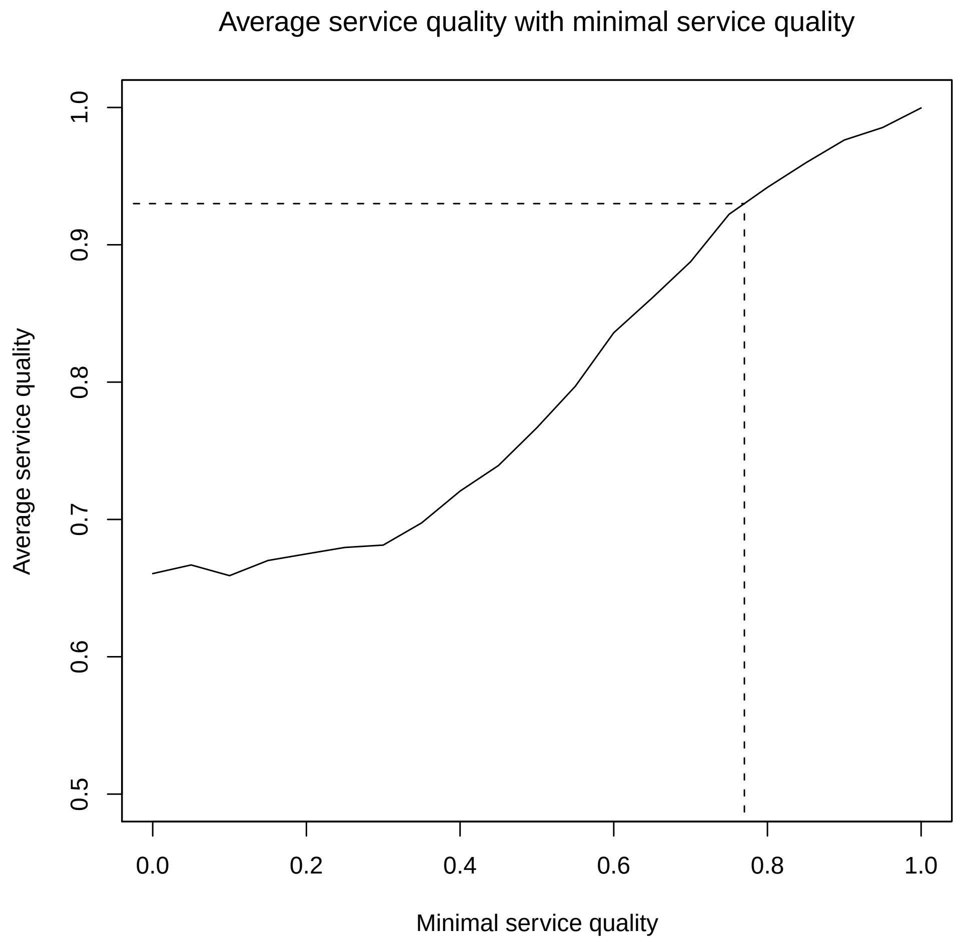


Figure 5.9: Average service quality for instances tested with a minimal service quality level.

for the corresponding level. The average service quality over all instances in the multi-company setting is 0.93 and when we search for the corresponding minimal service quality level we found a value of 0.77.

In Figure 5.10 a scatter plot is shown in which service quality is plotted against total costs, for the multi-company and single-company setting. The costs are somewhat higher in the multi-company setting, but the points in the plot are too close to each other to give a proper judgment about this measure. The paired t-test gives us a mean difference of 23.5, a 95%-confidence interval of $[11.9, 35.0]$, and a p-value of 1.71×10^{-4}, with higher total costs in the multi-company setting.

Another paired t-test is performed to judge about the difference in average

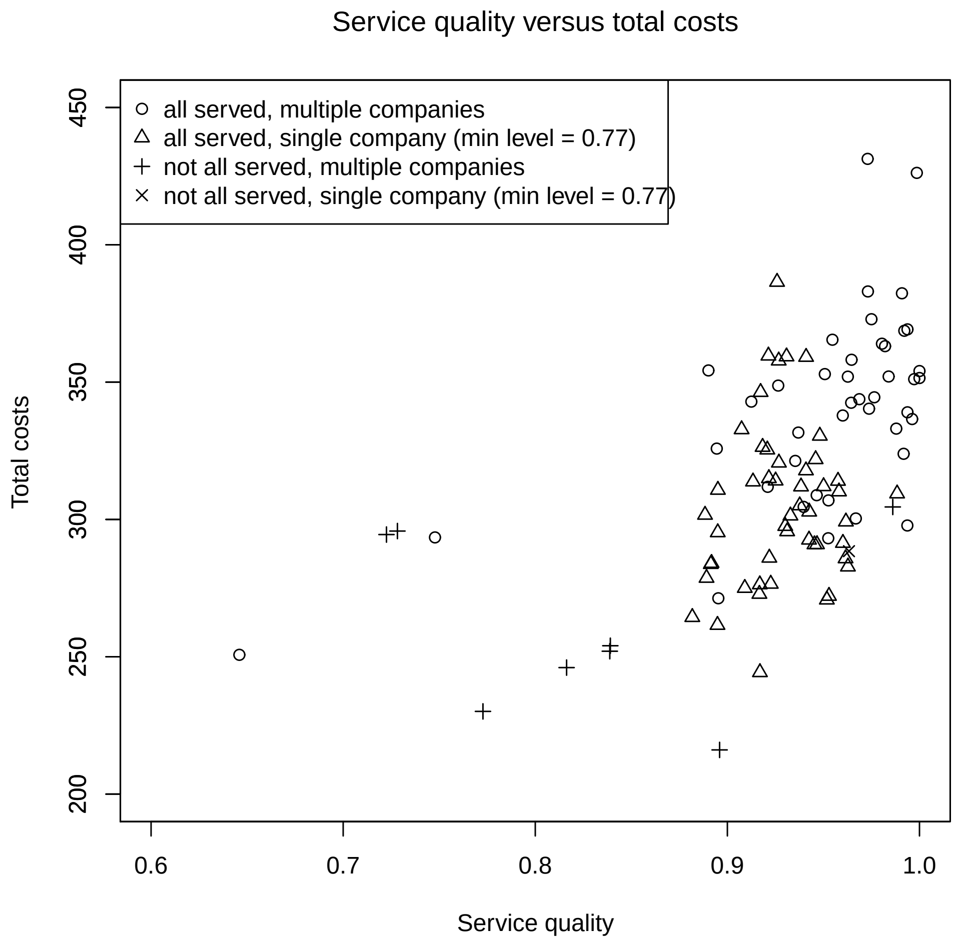


Figure 5.10: Service quality versus total costs for a multiple company setting and a single company setting in which the single company acts like a company in a multi-company setting.

service quality between the two settings. There should be almost no difference between the two settings, because of the way we determine the minimal service quality level for the single-company settings. The results of this test give us a mean difference of 0.0013 higher service quality in the multi-company setting, a confidence interval of $[-0.023, 0.026]$, and a p-value of 0.92.

5.6.3 Conclusion

We conclude that in a single-company setting in which the company must ensure a minimal service quality, the total costs increase the higher the minimal service quality level. This is concluded from the fact that the correlation coefficient between the two measures is quite large over the whole interval, and almost equal to 1 at some points.

Compared to the multi-company setting, the total costs in a single-company setting are less, even if this single company provides equal service quality. This means that the difference in total costs between a multi-company setting and the currently used setting discussed in Section 5.5 (i.e. single company minimizing costs) is mainly due to the higher average service quality and is not due to number of companies.

The main conclusion of this experiment is that although higher service quality in the door-to-door transportation can also be obtained by increasing the minimal service quality level for which the requests have to be served in a single-company setting, a multi-company setting introduces natural minimal service quality levels that result in the same average quality for a minimum of extra costs. Because increasing the minimal service quality levels is difficult, as we mentioned in the introduction chapter, it is surely worth it to pay attention to a multi-company solution.

Chapter 6

Summary and future work

In this concluding chapter we give a brief overview of this thesis and we present some pointers for future work.

6.1 Summary

In this thesis we focused on developing a mechanism for an environment in which multiple companies compete with each other on service quality to get assigned transportation requests. Increasing the average service quality in the door-to-door transportation for elderly and disabled people in the Netherlands has been the motivation for the development of such a mechanism.

In Chapter 2 we introduced the Dynamic Dial-a-Ride Problem with Time Windows (DDARPTW), which is often used to model door-to-door transportation. In this problem, routes have to be designed for vehicles that transport passengers from their pickup to their delivery location, where time windows decrease the flexibility to insert requests into the current schedule. We added a constraint for preserving service quality to the model by Cordeau and Laporte [6], such that this model could be used in our mechanism to guarantee promised service quality. We also added some constraints, such that the model could be used in an on-line optimization algorithm. A Mixed Integer Program based on the model is used to solve the problem exactly, by running a specialized solver.

The DDARPTW is NP-hard, so while solution concepts such as the one mentioned above give optimal solutions, the time to compute these solutions increases exponentially in the size of the problem input. We discussed a basic insertion heuristic that can be used to obtain near-optimal results in most cases and has good empirical performance. To this heuristic we added checks for the preserving of service quality, and we modified it such that it could be used in a dial-a-ride system.

In dynamic environments, information becomes known during the execution of the system. This implies that the solution needs to be updated frequently and that using exact algorithms takes too much time. We described a way in which these algorithms can be used in dynamic environments, called on-line optimization. By only taking into account that parts of the problem that can change or can influence the information needed at the current moment, problem sizes decrease.

Another way to deal better with dynamic settings is to use a decentralized multi-agent approach. According to Mahr et al. [26] an agent-based approach is competitive to an optimization approach for problem instances where less than 50% of the requests are known in advance, and Mes et al. [28] shows that a properly designed multi-agent approach performs as good as or even better than traditional methods. In such a decentralized approach all agents have to process the information they are responsible of, and the agents have to communicate with each other. We described the Contract Net Protocol (CNP) which contains rules for negotiation, contracting and decommitting.

A concept that is often used for assigning requests in a decentralized environment is an auction. In Chapter 2 we briefly mentioned how an auction can be used in a transportation context, but in Chapter 3 this concept is discussed in more detail. An auction is an application of mechanism design, which deals with the design of rules and strategies for games, such that the outcome of these games have some desirable properties.

In Chapter 4 we proposed an auction mechanism for the assignment of requests to multiple transportation companies. Different from other approaches is the bid type, which in our method is a service quality value. The company with the highest bid gets assigned the request and promises to serve this request with a minimal service quality. To ensure that quality, an additional constraint was added to a Mixed Integer Program, which we used in an on-line optimization approach to insert requests into vehicle schedules.

The prices that customers pay in the auction are not depended on the outcome of the auction. However, these prices do influence the bidding behavior of the companies. We showed an upper and lower bound for the price per kilometer, and discussed why the actual price must not exceed these bounds. A price that is too low will result in continuous losses for the companies, and a price that is too high will result in less requests that can be served.

A type of auction in which players have the incentive to truthfully announce their preferences is the widely used sealed-bid second-price auction, but problems with this truthfulness arises when multiple auctions are put in a sequence. Based on models by Juda and Parkes [18] and Fatima et al. [10], we provided a model in which sequential auctions can be studied. It seems that for different reasons it does not follow that a bidder has a dominant strategy in sequential auctions, while it has one in the single auctions.

There are ways to overcome the sequential auction problem; using options to allow decommitment, or finding other equilibrium strategies, such as a Bayes-Nash equilibrium. To deal with the sequential auction problem in our mechanism, we chose a simple method that is similar to the Bayes-Nash equilibrium approach discussed in Chapter 3. A Monte Carlo simulation is performed to determine the level of service quality for which the company makes about zero profit, taking into account future requests. This way, knowledge about the distribution of future requests is used to determine a bid for a current request. Because many insertions of requests need to be done in the Monte Carlo simulation, we chose to use an insertion heuristic, instead of exact solution concepts.

The results of the experiments that are discussed in Chapter 5 showed that the service quality is increased by 39% using our mechanism, instead of using an approach where companies compete on the price of a ride. We observed that total costs are also increasing when service quality is increasing. In this case the increase in total costs is 19%.

The incorporation of knowledge about future requests into the calculation of bids results in less optimistic bids, and therefore in less average service quality (−8%). We did not notice an increase in the number of served requests, but we think this is because the problem instances are too small to notice this difference.

It also follows from the results of the experiments that the same average service quality can be obtained in a single-company setting for less costs than in a multi-company setting. This is done by introducing a minimal service quality level in the single-company setting. The incentive to provide higher service quality must be explicitly added to the single-company setting, while in the multi-company setting the competition takes care of this incentive.

What we have reached in our work is providing a mechanism for door-to-door transportation in which companies compete on service quality for the assignment of customer requests, which leads to higher average service quality. There are some additional costs opposed to a single-company setting in which a minimal service quality level is determined, but due to problems that result from keeping to this minimal service quality, the additional costs for a multi-company setting fade away against the costs to solve the problems in a single-company setting.

6.2 Future work

To conclude this thesis, we state here some departure points for future work.

- An important assumption that we made in Section 4.1 is that vehicles are always driving exactly according to their schedule. It is obvious that in real-life this is not true, and because this can have a big influence on the scheduling and the final service quality, it is interested to investigate a setting where this assumption is not made.

- Another assumption that we made is that only one request is being auctioned at a time. It is also possible for a company to collect multiple request announcements and to calculate a combined bid value. Instead of a sequential auction, this would be a combinatorial auction, where players can bid on combinations of goods.

- In a real-life environment, all the valid requests made by customers must be served. This can be accomplished by hiring extra vehicles in the case that a request cannot be inserted in any of the vehicles' routes, but this results in additional costs. How to deal with rejected requests and what influence these extra costs have on the final solution needs to be further investigated.

- In Section 2.1.3 the degree of dynamism is defined, but for computational reasons we only considered problem instances in which all the requests are dynamic. The earlier a request arrives into the system, the longer it is present in the MIP formulation, so the more computation time is needed to insert subsequent requests. With the presence of static requests, companies are able to use knowledge about multiple requests in the calculation of a bid for a single request. This might lead to a different bidding behavior by the companies.

- Our definition of service quality as the ratio between direct and actual travel time is quite simple, but we think it was a good definition to start with. In the future, combined measures can be used, and even functional quality (instead of technical quality) measures can be used.

- The problem instances we used in our experiments are randomly generated and therefore it is hard to draw conclusions about our mechanism in real-life. By using real, historic data from taxi companies or insurance companies it would become possible to say something about the usefulness of the mechanism we proposed in real-life. Our problem instances are also quite small, containing only 16 requests. To present more accurate experimental results, it is important to investigate in future work how larger instances can be used.

- In Section 4.5.2 we provided a simple approach to incorporate future knowledge into the calculation of the current bid. This approach is similar to defining new equilibria for sequential auctions, which is described in Section 3.3.4, but we implemented it very simple. A pointer for future work is a theoretical analysis of defining equilibria in the sequential auction we used in our mechanism.

- The work in this thesis is quite practical and therefore we did not provide many theoretical analyses of our approach. Our idea was to present another way to assign requests to vehicles in a dial-a-ride system (i.e. by bidding service quality). In future work these analyses have to be provided and they may lead to new insights in the use of a mechanism like the one we provided.

Bibliography

[1] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE - a FIPA-compliant agent framework. In *Proceedings of the Practical Applications of Intelligent Agents*, 1999.

[2] Ann Melissa Campbell and Martin Savelsbergh. Efficient insertion heuristics for vehicle routing and scheduling problems. *Transportation Science*, 38(3):369–378, 2004.

[3] Alberto Colorni and Giovanni Righini. Modeling and optimizing dynamic dial-a-ride problems. *International Transactions in Operational Research*, 8(2):155–166, 2001.

[4] Wolfram Conen and Tuomas Sandholm. Preference elicitation in combinatorial auctions. In *EC '01: Proceedings of the 3rd ACM conference on Electronic Commerce*, pages 256–259, New York, NY, USA, 2001. ACM.

[5] Jean-Franois Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Oper. Res.*, 54(3):573–586, 2006.

[6] Jean-Franois Cordeau and Gilbert Laporte. The dial-a-ride problem: models and algorithms. *Annals of Operations Research*, 153(1):29–46, 2007.

[7] Luca Coslovich, Raffaele Pesenti, and Walter Ukovich. A two-phase insertion technique of unexpected customers for a dynamic dial-a-ride problem. *European Journal of Operational Research*, 175(3):1605–1615, December 2006.

[8] Claudio Cubillos and Claudio Demartini. An on-demand passenger transportation architecture based on a mediated contract-net. In *SAINT-W '05: Proceedings of the 2005 Symposium on Applications and the Internet Workshops*, pages 388–391, Washington, DC, USA, 2005. IEEE Computer Society.

[9] Anke Fabri and Peter Recht. Online dial-a-ride problem with time windows: An exact algorithm using status vectors. *Operations Research Proceedings 2006*, pages 445–450, 2006.

[10] Shaheen. S. Fatima, Michael Wooldridge, and Nicholas R. Jennings. Sequential auctions in uncertain information settings. In *Proc. 9th International Workshop on Agent-Mediated Electronic Commerce*, pages 15–28, 2007.

[11] Miguel Andres Figliozzi, Hani S. Mahmassani, and Patrick Jaillet. Pricing in dynamic vehicle routing problems. *Transportation Science*, 41(3):302–318, 2007.

[12] Klaus Fischer, Jorg P. Muller, and Markus Pischel. Cooperative transportation scheduling: an application domain for dai. *Journal of Applied Artificial Intelligence*, 10:1–33, 1996.

[13] Foundation for Intelligent Physical Agents. Specifications, 1997. Available from `http://www.fipa.org`.

[14] Konrad-Zuse-Zentrum für Informationstechnik Berlin. Scip. Available from `http://scip.zib.de/`.

[15] R. Gopal, S. Thompson, Y. A. Tung, and A. B. Whinston. Managing risks in multiple online auctions: An options approach. *Decision Sciences*, 36(3):397–425, 2005.

[16] Cristian Grönross. A service quality model and its marketing implications. *European Journal of Marketing*, 18(4):36–44, 1984.

[17] Jang-Jei Jaw, Amedeo R. Odoni, Harilaos N. Psaraftis, and Nigel H. M. Wilson. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B: Methodological*, 20(3):243–257, June 1986.

[18] Adam I. Juda and David C. Parkes. The sequential auction problem on ebay: an empirical analysis and a solution. In *EC '06: Proceedings of the 7th ACM conference on Electronic commerce*, pages 180–189, New York, NY, USA, 2006. ACM.

[19] Adam I. Juda and David C. Parkes. An options-based solution to the sequential auction problem. *Artif. Intell.*, 173(7-8):876–899, 2009.

[20] Brian Kallehauge. Formulations and exact algorithms for the vehicle routing problem with time windows. *Comput. Oper. Res.*, 35(7):2307–2330, 2008.

[21] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[22] Vijay Krishna. *Auction Theory.* Academic Press, 2002.

[23] A. Larsen, O. Madsen, and M. Solomon. Partially dynamic vehicle routing - models and algorithms. *Journal of the Operational Research Society*, 53:637–646, jun 2002.

[24] K. Lund, O.B.G. Madsen, and J.M. Rygaard. Vehicle routing problems with varying degrees of dynamism. Technical report, Institute of Mathematical Modelling, Technical University of Denmark, 1996.

[25] O.B.G. Madsen, H.F. Ravn, and J.M. Rygaard. A heuristic algorithm for the a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. *Annals of Operations Research*, 60:193–208, 1995.

[26] Tamas Mahr, Jordan Srour, Mathijs M. de Weerdt, and Rob Zuidwijk. Agent performance in vehicle routing when the only thing certain is uncertainty. In *Proceedings of the workshop on Agents in Traffic and Transportation (ATT)*, 2008.

[27] Martijn Mes. *Sequential Auctions for Full Truckload Allocation.* PhD thesis, Universiteit Twente, Enschede, The Netherlands, 2008.

[28] Martijn Mes, Matthieu van der Heijden, and Aart van Harten. Comparison of agent-based scheduling to look-ahead heuristics for real-time transportation problems. *European Journal of Operational Research*, 181(1):59–75, August 2007.

[29] Nicholas Metropolis and S. Ulam. The monte carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949.

[30] T. Miyamoto, K. Nakatyou, and S. Kumagai. Route planning method for a dial-a-ride problem. *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, 4:4002–4007 vol.4, Oct. 2003.

[31] Lonneke Mous, Valentin Robu, and Han La Poutré. Using priced options to solve the exposure problem in sequential auctions. In *Proc. of the 10th Int. Workshop on Agent Mediated Electronic Commerce (AMEC'08), Estoril, Portugal*, 2008. (Full paper, post-proceedings to appear in Springer LNAI).

[32] Noam Nisan. Bidding and allocation in combinatorial auctions. In *EC '00: Proceedings of the 2nd ACM conference on Electronic commerce*, pages 1–12, New York, NY, USA, 2000. ACM.

[33] Julie Paquette, Jean-François Cordeau, and Gilbert Laporte. Quality of service in dial-a-ride operations. *Computers & Industrial Engineering*, 56(4):1721–1734, May 2009.

[34] David C. Parkes. Auction design with costly preference elicitation. *Annals of Mathematics and Artificial Intelligence*, 44(3):269–302, 2005.

[35] D. Perugini, D. Lambert, L. Sterling, and A. Pearce. A distributed agent approach to global transportation scheduling. *Intelligent Agent Technology, 2003. IAT 2003. IEEE/WIC International Conference on*, pages 18–24, Oct. 2003.

[36] Stefan Ropke, Jean-François Cordeau, and Gilbert Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Netw.*, 49(4):258–272, 2007.

[37] Ilya Segal. The communication requirements of combinatorial allocation problems. In Y. Shoham P. Cramton and R. Steinberg, editors, *Combinatorial Auctions*, chapter 11, pages 265–295. Cambridge, Massachusetts: MIT Press, 2006.

[38] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations.* Cambridge University Press, 2008.

[39] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Trans. Comput.*, 29(12):1104–1113, 1980.

[40] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res.*, 35(2):254–265, 1987.

[41] P. J. 't Hoen, V. Robu, and J. A. La Poutre. *Decommitment in a Competitive Multi-Agent Transportation Setting.* Birkhäuser Basel, 2005.

[42] Dusan Teodorovic and Gordana Radivojevic. A fuzzy logic approach to dynamic dial-a-ride problem. *Fuzzy Sets Syst.*, 116(1):23–33, 2000.

[43] Jiří Vokřínek, Jiří Bíba, Jiří Hodík, Jaromír Vybíhal, and Michal Pěchouček. Competitive contract net protocol. In *SOFSEM '07: Proceedings of the 33rd conference on Current Trends in Theory and Practice of Computer Science*, pages 656–668, Berlin, Heidelberg, 2007. Springer-Verlag.

[44] Michael P. Wellman, William E. Walsh, Peter R. Wurman, and Jeffrey K. MacKie-Mason. Auction protocols for decentralized scheduling. *Games and Economic Behavior*, 35(1-2):271–303, April 2001.

[45] Jian Yang, Patrick Jaillet, and Hani S. Mahmassani. On-line algorithms for truck fleet assignment and scheduling under real-time information. *Transportation Research Record: Journal of the Transportation Research Board*, 1667(1):107–113, 1999.

Appendix A

Glossary

In this appendix we give an overview of frequently used terms and abbreviations.

a priori with the information known beforehand

AV Affiliated Value

ex ante in expectation from the beginning of execution

CCNP Competitive Contract Net Protocol

CNP Contract Net Protocol

CV Common Value

DARPTW The Dial-a-Ride Problem with Time Windows

DDARPTW The Dynamic Dial-a-Ride Problem with Time Windows

dod degree of dynamism

ECNP Extended Contract Net Protocol

edod effective degree of dynamism

ILP Integer Linear Program

IPV Independent Private Value

MIP Mixed Integer Program

PAP Provisional Agreement Protocol

PDVRP Pickup and Delivery Vehicle Routing Problem

SCIP A non-commercial mixed integer programming solver

TSP Traveling Salesman Problem

VRPTW Vehicle Routing Problem with Time Windows

Appendix B

Software model

In this appendix we present a class diagram that give some insight into our implementation of the mechanism proposed in this thesis. In Section B.2, some software libraries that we use in our implementation are discussed.

B.1 Class diagram

In Figure B.1 a class diagram is shown of our Java implementation of the mechanism proposed in this thesis.

B.2 Software parts used

In this section we discuss some software parts that we used in our implementation. A framework to develop agent-based systems, called JADE, is described in Section B.2.1 and a Mixed Integer Program solver, called SCIP, is treated in Section B.2.2.

B.2.1 JADE

Description

The Java Agent DEvelopment Framework (JADE) is a software framework fully implemented in the Java language. It simplifies the implementation of multi-agent systems through a middle-ware that complies with the FIPA specifications and through a set of graphical tools that supports the debugging and deployment phases. The communication between the agents is all taking care of and behaviors of agents can simply be added. Because it is fully implemented in Java, it can be easily extended.

The communication architecture offers flexible and efficient messaging, where JADE creates and manages a queue of incoming ACL messages, private to each agent; agents can access their queue via a combination of several modes: blocking, polling, timeout and pattern matching based.

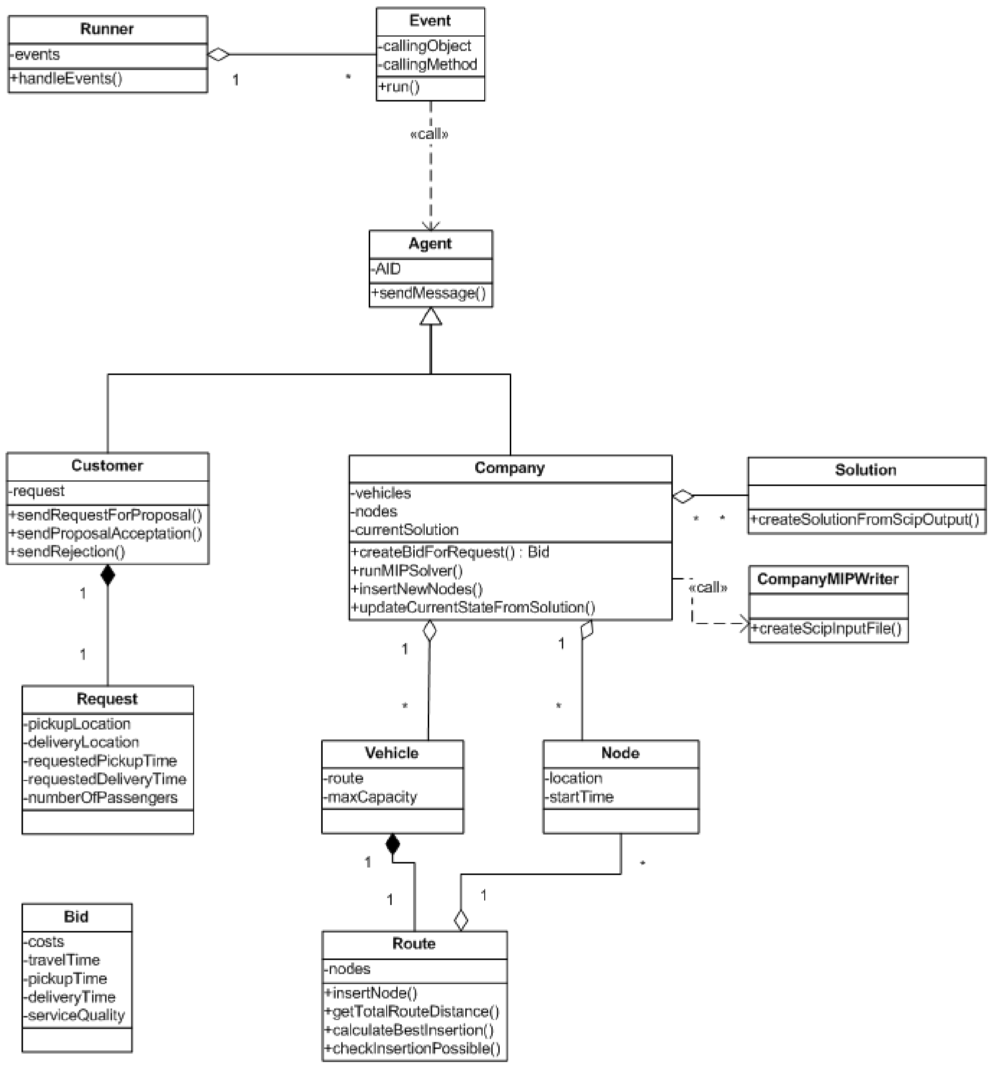


Figure B.1: A simple class diagram of our Java implementation of the mechanism proposed in this thesis.

The agent platform provides a Graphical User Interface (GUI) for the remote management, monitoring and controlling of the status of agents, allowing, for example, to stop and restart agents. The GUI allows also to create and start the execution of an agent on a remote host, provided that an agent container is already running.

Review

JADE can be used as the framework for our implementation. The behavior of the agents is strictly separated from the communication, so they can be developed independently from each other. It is very easy to extend agents and define behaviors for these agents. It is also very easy to send and receive messages. The underlying infrastructure is all taking care of by the framework.

An important benefit is that we can make the system act as a real multi-agent system, because all agents live in their own threads and can even exist on different computers (e.g. have their own cpu). Another benefit is the experience that already exists in working with Java.

A drawback could be that the framework is too extensive, in the sense that there are a lot of superfluous features that we won't use.

Source

Documentation, tutorials and source code of JADE can be found at `http://jade.tilab.com/`.

B.2.2 SCIP and ZIMPL

Description

SCIP is currently one of the fastest non-commercial mixed integer programming solver. It is also a framework for Constraint Integer Programming and branch-cut-and-price. It allows total control of the solution process and the access of detailed information down to the guts of the solver.

To simplify the development of problem instances, the language ZIMPL is developed. In this language, higher level data structures (e.g. sets) and operations over these data structures (e.g. summation) can be used. Input files written in ZIMPL can be fed directly to SCIP or they can be converted to LP files (default SCIP input).

Review

When the model is developed using mixed integer programming, SCIP can be used to solve this problem. To solve sub-problems (e.g. within a vehicle), we can define this problem in ZIMPL and give this to SCIP to solve the problem.

Source

SCIP is maintained by Konrad-Zuse-Zentrum für Informationstechnik Berlin, and can be downloaded from `http://scip.zib.de/`.