universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

A Large Neighbourhood Search for the Dynamic Pickup and Delivery Problem with Time Windows and Heterogeneous Fleet

verfasst von / submitted by

Marina Stiliyanova Ivanova, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2018 / Vienna 2018

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 915
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Betriebswirtschaft
Betreut von / Supervisor:	o. Univ.-Prof. Dipl.-Ing. Dr. Richard Hartl

Acknowledgements

I would like to thank my supervisor Prof. Richard Hartl for giving me the opportunity to work on this exiting topic and to write this thesis. I am very grateful for his ongoing encouragement and constructive feedback.

Furthermore, I also want to express my sincere gratitude to Mag. Belma Turan, PhD for allowing me to work with her, for her dedicated mentoring, continuing support, creative suggestions, infinite patience, and above all for always believing in me.

I want to thank the rest of my colleagues at the Chair of Production and Operations Management as well as to Prof. Karl Dörner and my other colleagues at the Chair of International Production and Operations Management. I am happy I got to work with all of these wonderful people. The years I have spent there had a huge impact on both my personal and professional development.

Thanks to my close friends Julia, Denny, and Helene. I would not have made it without their cheerfulness, support, and love. I would like to say *thank you* also to all the other good friends I have made in Vienna during my studies for the great student life we have had together.

Last but not least, special thanks to my best friend, Aneta, for always being by my side despite fights, distance, and time.

I want to dedicate this thesis to my dear parents who have always supported and loved me unconditionally.

List of Figures

List of Tables

Contents

1 Introduction

Pickup and delivery problems (PDPs) are a class of vehicle routing problems (VRPs) that have been widely studied over the past few decades. They arise in various real-life situations, requiring the transportation of goods or people between an origin and a destination. There are two main subclasses of the PDP. The first one comprises problems, dealing with the transportation of a common good between customers. As such good is interchangeable, it can be used to serve the demand of all pickup and all delivery vertices. This is known as an unpaired PDP. The other subclass is the paired PDP, where a non-interchangeable good has to be transported between a given origin and a given destination. Courier services are an example for such problems in real-life context. If people rather than goods are to be transported, this is then the so called dial-a-ride problem (DARP). The most notable example for this class of problems is the transportation of disabled or elderly people. When it comes to paired PDPs, especially in the context of courier operations, requests are often received during working hours and have to be fulfilled on the same day. The fact that not all input is available at the beginning of the planning horizon makes the problem dynamic. In the static case, all information is known beforehand. If only partial information is available, for example probability distribution of new requests over time, the problem is stochastic rather than deterministic.

The motivation for this work is precisely a problem faced by a courier company in an urban area, where a large proportion of the requests are received on the same day as the desired transportation. Such framework makes planning difficult, as routing must be changed continuously throughout the day. Especially in the presence of additional constraints such as time windows, these adjustments need to be done very quickly in order to minimise delays. Further aspects to be considered in this particular extension are vehicle capacities, drivers' working hours as well as different

time windows for different request types.

The rest of this work is structured as follows. Section 2 gives an overview of related research on dynamic PDPs, including solution methods used for tackling them. Section 3 describes in detail the problem studied in this work. The mathematical model for the static version of the problem is given in Section 4. Section 5 presents the solution method used for solving the problem. The computational results are summarized and analysed in section 6. This section also explains the structure of the test instances. Finally, section 7 summarizes the content of this work and at the same time provides thoughts on possible future research.

2 Literature Review

There is extensive literature on the topic of PDPs. Berbeglia et al. (2010) note that most works conducted on the topic of PDPs study the static problem, whilst research on the dynamic version remains rather scarce. Thanks to the advance of technology and increase in computational power, interest for problems of a dynamic nature has risen in general. Psaraftis et al. (2015) observe a "boom" in related literature on the DVRP since 2000.

A first comprehensive survey providing a overview of the various cases of PDPs was conducted by Parragh et al. (2008). It considers however only static PDPs. The survey clusters them in two main subclasses, paired and unpaired PDPs. It then distinguishes further between two subtypes of paired problems - the classical PDP and the DARP. (As Section 1 describes the difference between those three classes, we will refrain from repeating it here.) Parragh et al. (2008) present mathematical formulations and reports on exact and heuristic solution methods for all three variations, in both single-vehicle and multi-vehicle frameworks.

A recent survey by Psaraftis et al. (2015) on dynamic vehicle routing problems

(DVRPs) looks at some 40 papers on dynamic PDPs, with 37 works being on the paired version of the problem. The other three study the unpaired dynamic PDP. This literature review concentrates on existing research on dynamic and deterministic PDPs, in which dynamism stems only the appearance of new requests.

The latest published survey solely on the dynamic PDP is by Berbeglia et al. (2010). It focuses on the paired one-to-one subclass providing an overview of the existing literature on the dynamic PDP, related solution strategies and evaluation of the algorithmic performance. Berbeglia et al. (2010) also present some considerations for a general framework for dynamic PDPs. The describe the potential options that decision makers have and criteria for evaluating solution strategies. Examples for such criteria stated by them are total distance travelled, number of accepted requests, etc.

Psaraftis et al. (2015) develop a comprehensive taxonomy of DVRPs, which is depicted in Figure 1 on the following page. In their survey, the authors take into account 117 works on DVRPs to classify them. Psaraftis et al. (2015) use 11 different categories in their taxonomy, whereby the last criterion refers only to classification of DVRP papers and not of the problems themselves. The criteria are as follows: type of problem, logistical context, transportation mode, objective function, fleet size, time constraints, vehicle capacity constraints, ability to reject customers, nature of the dynamic element, nature of the stochasticity, and solution method. We will use this criteria in the remaining part of this work as well.

Savelsbergh and Sol (1998) propose a branch-and-price based algorithm for solving a real-world dynamic PDP with various problem-tailored constraints, such as time windows, heterogeneous fleet and maximal vehicle working hours. The solution concept is developed to serve as a tool for a decision support system within a big courier services company in the Benelux. The primary goal is to minimize costs for drivers' pay, which are a combination of lease costs, total distance travelled and

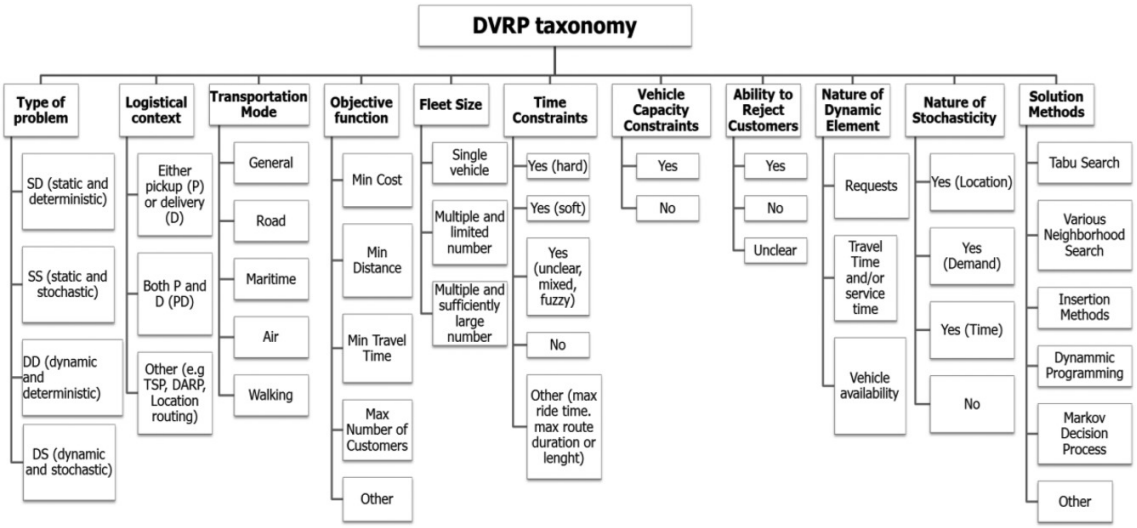

Figure 1: DVRP taxonomy, Psaraftis et al. (2015)

number of nights the driver spends away from home. The branch-and-bound core algorithm is combined with a column generation scheme and different heuristics. Using real-world generated instances, the algorithm is tested in a simulated dynamic environment. This is achieved by a rolling-horizon approach, for which the problem is broken into a few smaller static problems with a given number of known requests. This part is executed and, after some time passes and new requests become available, the optimisation takes place again. For large instance sets, the algorithm leads to better results than the solution of the dispatchers. It should be noted that the specific problem constraints presented in this work of Savelsbergh and Sol (1998) are similar to the ones characterising the problem proposed in this thesis, e.g. different vehicle capacity and pre-defined working hours.

Barceló et al. (2007) propose also a modelled framework to be embedded in a decision support system. They develop an approach for designing and evaluating complex city logistics applications, whilst integrating all related components and underlying systems, such as database management systems, a GIS based Graphic User Interface, real-time vehicle tracking system and others. The implemented optimisation is based

on tabu search and simulated annealing.

A paper by Fiegl and Pontow (2009) studies an application of a dynamic PDP in the context of pickup and delivery jobs in hospitals. This is one of the few works on dynamic PDPs, where the transportation mode is not road. Here the employees, completing tasks such as transporting blood samples, are walking around the hospital, which means very small travelling distances. This implies also the need for a fast re-optimisation of incoming requests. The authors propose a solution method based on scheduling theory in combination with graph theory. As in most dynamic PDPs, the only unknown data input here are new tasks.

Another study that considers a transportation mode different from road is by Psaraftis et al. (1985). In this work, the authors develop a rolling-horizon heuristic for solving a sealift routing and scheduling problem that can be subsequently integrated within a scheduling subsystem of the maritime. The problem consists of assigning cargoes to ships in emergency situations so that they are reach given destination within a pre-defined time frame. As the cargoes contain identical goods, this is an unpaired dynamic PDP.

A large number of studies is dedicated to develop waiting and/or buffering strategies for dynamic PDPs.Mitrović-Minić and Laporte (2004) propose a two-stage heuristic approach for solving an uncapacitated dynamic PDP with time windows in a courier service setting. First, a simple cheapest insertion heuristic is applied, followed then by a tabu search. They are implemented for the static problems, i.e. in a rolling-horizon framework. The objective in this problem is minimising total travel distance. All requests must be served. For the scheduling, four different waiting strategies are introduced. The first two are simple drive-first and wait-first strategies, whilst the other ones are the more complex dynamic waiting and advanced dynamic waiting strategies. The results show that a sophisticated waiting strategy outperforms a simple waiting concept. This study is followed up by another one that

further develops their first algorithm. In Mitrović-Minić et al. (2004), they observe the fact that the travel distance is not suitable as a long-term objective, because routes change constantly. Therefore, the authors present a double-horizon based heuristic with a short-term and a long-term objective. The short-term one remains minimisation of the route length, whilst the long-term is to maximise the slack for inserting incoming requests. Tested on real-world inspired data from a hospital in Germany, this algorithm outperforms the first one.

Pureza and Laporte (2008) consider buffering next to waiting for developing adequate scheduling strategies in their study of a dynamic PDP with time windows and random travel times, whereby the aim is to minimise the number of vehicles as well as the proportion of lost requests. Minimizing total travel distance is considered in the objective, too. The authors study the impacts of a waiting concept that postpones the assignment of a request to a vehicle, whilst waiting at a given destination that is close to a potential next one. The buffering strategy consists of delaying the routing of non-urgent requests for as long as possible. The constructive-destructive heuristic approach guiding the waiting and buffering strategies is tested on a randomly generated data sets, containing up to 100 requests. The instances contain requests, which have different degrees of dynamism as well as both static and dynamic travel times.

The main difference between the classical PDP and the DARP is that people transportation is subject to user ride time constraints and very tight time windows. The primary goal in this kind of problems is proving good service quality and minimising user inconvenience. Studies on the dynamic and deterministic DARP include but are not limited to those by Attanasio et al. (2004), Beaudry et al. (2010), Berbeglia et al. (2011) and Berbeglia et al. (2012).

Attanasio et al. (2004) apply several parallel tabu search procedures for solving a formulation of the dynamic DARP. Initial solutions for the static case are created by

using a tabu search, which is then followed by a feasibility check upon receiving new requests. If the incoming requests can be accepted, a post-optimisation takes places. The algorithm aims to maximise the number of served customers. The authors are able to show a good performance of their solution method tested on the 26 instance sets of Cordeau and Laporte (2003).

Another work on the dynamic DARP is described in Beaudry et al. (2010) that consider an application in the context of patient transportation in hospitals. The problem is tailored to reflect various features and constraints arising in a hospital setting. The objective is based on minimizing both indirect and direct costs for the hospitals as well as the user inconvenience. The authors solve the problem using a two-phase heuristic algorithm, which consists of a simple insertion procedure in the first step, and of an improvement procedure guided by a tabu search heuristic in the second part. Tested on real-life data gathers from several German hospitals having 1000 beds or more, the algorithm shows a very good performance, delivering solutions that show a clear reduction in used vehicles and lower waiting times for users.

Berbeglia et al. (2011) propose an exact algorithm based on constraint programming to quickly determine feasible solutions for a DARP. The authors aim to provide a procedure that can be used to reject or accept new requests in a timely and efficient manner, depending on whether the users can be serviced without violating feasibility constraints. Their computational experiments are conducted on the two DARP instance sets by Ropke et al. (2007), with some modifications made to suit the specific problem. Results show that the algorithm finds in general a feasible solution within a few seconds. It performs better for more constrained instances.

Another article with participation of one of the authors of the above work presents a hybrid procedure for solving a dynamic DARP. This hybrid solution method introduced in Berbeglia et al. (2012) is made up of a sequential tabu search

heuristic and an exact constraint programming algorithm. The objective is to minimise route cost and to maximise the proportion of serviced customers. The authors modify the static instances introduced in Cordeau and Laporte (2003) and Ropke et al. (2007), converting them to suit a dynamic setting. Results show that the hybrid algorithm outperforms the other two when implemented alone.

Attanasio et al. (2007) describe their implementation of a framework developed for a real-time fleet management system of a big courier services provider. Their solution method combines a parallel tabu search as optimisation procedure with simpler insertion heuristics. The authors' general goals from a business perspective are to reduce needed human supervision for fleet management, to offer better service to customers, and to increase operational efficiency.

Gendreau et al. (2006) introduce a neighbourhood search heurstics for a dynamic PDP, which is also motivated by courier services in local area. The objective formulated in this paper is cost minimisation, whereby the cost is the weighted sum of total travel time, sum of overall tardiness, and sum of overall overtime that drivers had to make to complete their routes. The proposed algorithm is a sophisticated neighbourhood search with ejection chains. It is tested on instances that are generated using a self-developed simulator to recreate a real-life dynamic setting. The solution method outperforms simple heuristics even for the static case, given enough computational power. In general, results indicate good performance in complex dynamic scenarios.

A metaheuristic procedure based on neighbourhood search can be found also in Goel and Gruhn (2008), who propose an algorithm for solving a generalisation of the VRP and the PDP for real-life applications. This problem is known as the General Vehicle Routing Problem (GVRP) and represents a combined load acceptance and routing problem. The authors consider a high number of specific constraints that are typical for real-life settings, such as heterogeneous fleet with different load capacity, speed and cost, route restrictions, different start and end for vehicles, requests with

multiple pickup and delivery locations, etc. They aim at finding feasible profit-maximising routes, whereby the profit is difference between the revenue from all serviced requests and the operation costs. The authors develop and implement two neighbourhood search procedures, the Reduced Variable Neighbourhood Search (RVNS) and the Large Neighbourhood Search (LNS). Computational experiments performed on self-generated instances show encouraging results for problems with a high number of vehicles and requests, with average response time being mostly under a second, thus suggesting it could be used in a real-life planning system.

3 Problem Description

The classical PDP is illustrated via a dummy example in Figure 2 on the next page. Two routes with a total of four requests can be seen. The dummy problem assumes only two available vehicles to service four requests, with one vehicle having a capacity of 7 units and the other of 12 units. $Q_{P(i)}$ denotes the positive demand of a pickup location $P(i)$, whilst $Q_{D(i)}$ stands for the negative demand of a delivery location $D(i)$. The demand of a pickup location is positive, because it adds to the load of the vehicle, whereas demand location are associated with a negative demand because the vehicle leaves with a reduced load after servicing them. In the example shown, the first vehicle with a capacity of 7 units starts servicing the requests on its route by first visiting the pickup location $P(1)$ that has a positive demand $Q_{P(1)}$=2. The corresponding delivery location could be serviced only after visiting the pickup vertex. The other possibility for routing of the requests is to service further pickup locations of other requests as long as it still has enough capacity left. The decision what location will be visited next depends on multiple factors, e.g. which locations are ready to be serviced, also known as time windows. The routing is also based on the goal of the decision-maker, e.g., cost minimisation in terms of distance travelled or

minimising delays. In this example, we define the objective as minimising the total distance travelled. This is why the first vehicle with a residual capacity of 5 units proceeds next to service the pickup location $P(2)$. Following that, it has a residual capacity of 1 unit, which is not sufficient to serve any other of the requests, even if they would be closer than the delivery locations of the first two requests. Therefore, the vehicle visits as next the delivery vertices of requests 1 and 2, after which it has enough to capacity to travel to the pickup location of request 3. However, in order to show the importance of time windows in PDPs, here we assume that time windows violations are not allowed and the first vehicle cannot reach the pickup or delivery location of request 3 before the corresponding time windows have closed. As for request 4, it can be serviced only by the second vehicle as its demand of 8 units exceed the maximal capacity of the first vehicle.

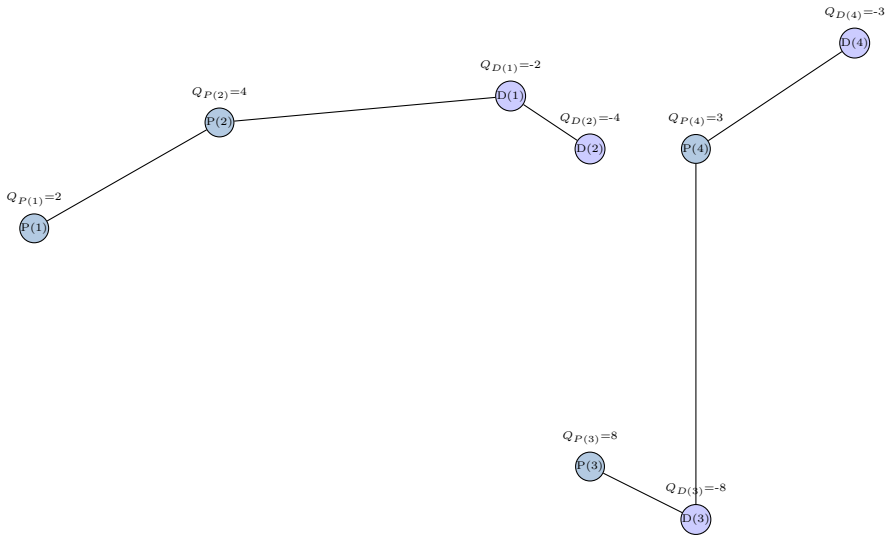

Figure 2: Graphic representation of a PDP

This thesis studies a variation of the classical PDP, which is motivated by a real-world application of intra-city courier services. It considers several characteristics encountered in a courier-operations setting, such as limited fleet size, different vehicle

capacity and speed, different time windows of requests, and drivers' working time.

The following assumptions were made in order to formulate the problem and develop an adequate solution method:

- A request is the pickup and delivery of a single order/package. If there is more than one order with the same pickup and/or delivery location, this is always regarded as a separate request.
- Time windows for servicing pickup and delivery locations are considered.
- There are different types of vehicles, i.e. the fleet is heterogeneous. Each vehicle type has a given maximum capacity.
- Depots are the starting and ending points of the vehicles (e.g., home locations of the drivers). The distance to the first served location and from the last served location is neglected.
- Once a vehicle starts driving to a given customer, it cannot be diverted. Cancellations of requests are not possible, too.
- As information about incoming requests is being revealed in real time during a given planning horizon, the problem is dynamic.
- Requests that are entered in the system after the end of the planning horizon for a given day are executed on the next day. For this subsequent planning horizon, such requests are part of the initial static solution as they are known at the start of planning.
- The only unknown or partially unknown input is the occurrence of incoming requests. Congestion, vehicle breakdowns, request cancellations, communication problems between drivers and planners, etc., are not considered in this formulation.
- The vehicle shipping cost is assumed to be equivalent to the travel distance.
- This is a less-than-full truck load problem as transportation does not have to occur directly from the pickup location to the delivery location.

- There are different types of requests, classified according to their time windows. Delays are allowed, but are penalised in the objective function.

Following the proposed taxonomy of DVRPs by Psaraftis et al. (2015), depicted in Figure 1 on page 4, the problem presented in this work is *dynamic and deterministic*, *both pickup and delivery* with a *multiple and limited fleet*. Transportation mode is *road*. There are *soft time constraints*, for the courier has only a certain number of vehicle at its disposal. Soft windows mean that delays are penalised in the objective function. Other time constraints arise from considering the working hours of the vehicles. Although various studies, e.g. Berbeglia et al. (2010), point out that capacity is rarely considered in the context of courier services, in this variant vehicle capacity is not unlimited. The reason for that is the heterogeneous fleet, which contains also vehicles that deliver parcels big enough to impose a logical restriction on maximum vehicle load. A number of the papers on dynamic PDP in the context of transportation services mentioned in Section 2, such as Savelsbergh and Sol (1998), Cheung et al. (2008), Barceló et al. (2007) and others, put a restriction on vehicle capacity. With regards to courier services, Attanasio et al. (2007) propose a model with a vehicle capacity, too. As for the *nature of dynamic element*, the dynamism in the problem is related only to the incoming unknown requests. *Stochasticity* is not considered here.

4 Mathematical Model

The following section contains the mathematical formulation of the static version of the problem in order to give a better understanding of the problem's complexity. It then introduces a general framework for dynamic aspects in PDPs as well as two different measures of dynamism used in the literature. A complete dynamic formulation is not included, as the problem is solved using not an exact but a

metaheuristic framework.

4.1 Notation

The notation used in the mathematical model is briefly summarised in Table 1 on the next page. The set of all pickup locations is denoted with P, whilst D specifies the set of all delivery locations. V describes the set of all locations, which comprises all pickup locations, delivery locations as well as the home locations (depots) of the vehicles. K stands for the set of all vehicles. Each pickup or delivery location i has a specific demand Q_i, associated with a positive value for pickup locations and with a negative value for delivery locations. The time windows are specified by E_i, the earliest time to begin service at a location i, and L_i, the latest time to begin service at a location i. A delay at location i is penalized with a penalty α_i. ST_i describes the service time at a location i. The travel time from location i to location j with vehicle k is denoted by T_{ij}^k. Each vehicle k has a defined start and end time, denoted by BT^k and ET^k, respectively. M_{ij}^k and N_{ij}^k are sufficiently large positive constants used for linearisation.

4.2 Decision Variables

The binary variable x_{ij}^k indicates if an arc $(i, j) \in A$ is traversed by vehicle k. In such case, the variable equals 1. The other type of binary variable, y_i^k, takes the value 1 if location i is served by vehicle k. It should be noted that it is sufficient to define y_i^k only for the set of pickup and delivery locations $(P \cup D)$. The load of vehicle k when leaving location i is represented by q_i^k. b_i^k is needed to identify the beginning of service of vehicle k at location i.

Symbol	Description
i, j	locations, $(i, j) \in A$
k	vehicle
K	set of all vehicles, $K = \{1, \ldots, r\}$
V	set of all locations, $V = \{1, \ldots, 2n + r\}$
A	set of all arcs
P	set of pickup locations, $P = \{1, \ldots, n\}$
D	set of delivery locations, $D = \{n + 1, \ldots, 2n\}$
α_i	penalty for a delay at location i
Q_i	demand at location i, associated with a positive value for pickup locations and with a negative value for delivery locations
E_i	earliest time to begin service at a location i
L_i	latest time to begin service a location i
ST_i	service time at a location i
T_{ij}^k	travel time from location i to location j with vehicle k
BT^k	start time of vehicle k
ET^k	end time of vehicle k
C^k	capacity of vehicle k
M_{ij}^k	a sufficiently large positive constant used for linearisation
N_{ij}^k	a sufficiently large positive constant used for linearisation

Table 1: Notation of the mathematical model

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

$$y_i^k = \begin{cases} 1, & \text{if location } i \text{ is served by vehicle } k \\ 0, & \text{else} \end{cases}$$

q_i^k load of vehicle k when leaving location i

b_i^k beginning of service of vehicle k at location i

4.3 Objective Function

The objective (1) is to minimise the overall cost for delays. Here a delay is defined as the positive difference between the earliest and latest time to start service at location i, whereby $i \in P \cup D$. This delay is weighted with a penalty factor, α_i. The sum over all delays multiplied with their respective penalty factor builds the overall cost, which then has to be minimised.

$$min \sum_{i \in P \cup D} \alpha_i (b_i - L_i) \tag{1}$$

4.4 Constraints

Constraint (2) guarantees that all requests are fulfilled. The flow conservation is given by (3). Coupled with (4), it ensures that a request is always served by the same vehicle, i.e. if a pickup location i is visited by a vehicle k, then this vehicle must also serve the corresponding delivery location $i + n$. Constraint (5) states that the service at a given location i cannot begin before the time window for that location

has opened. Here it should be noted that there is no restriction regarding the latest possible start of service for delays are not explicitly forbidden but rather heavily penalised in the objective function. The next two constraints address the issue of precedence. As stated by (6), the beginning of service at a delivery location cannot occur before the start of service at the respective pickup location. Constraint (7) defines that service at a given location can start only after completion of service at its predecessor plus the respective travel time between the two. Load consistency is ensured in (8). Constraint (9) guarantees that capacity restrictions are not breached. The left-hand side of constraint (9) imposes that the load q of a vehicle k when leaving a location i is always non-negative, while the right-hands bounds a vehicle load to never exceed the maximum capacity C^k. Constraints (10) and (11) specify the ranges for the two constants used for linearisation as described by Cordeau et al. (2001). The working times of the vehicles are considered in constraints (12) and (13). Constraint (12) defines that the completion of service at any location is before the end working time of a vehicle. In line with this, a vehicle cannot start service at any location before the start time of its working hours, as stated by (13). Non-negativity is given in (14), whilst the last two constraints (15) and (16) guarantee binarity of the respective decision variables.

$$\sum_{k \in K} y_i^k = 1 \qquad \forall i \in P \cup D, \tag{2}$$

$$\sum_{j \in V} x_{ij}^k = \sum_{j \in V} x_{ji}^k = y_i^k \qquad \forall i \in V, k \in K, \tag{3}$$

$$y_i^k - y_{i+n}^k = 0 \qquad \forall i \in P \cup D, k \in K, \tag{4}$$

$$b_i^k \geq E_i \qquad \forall i \in P \cup D, k \in K, \tag{5}$$

$$b_{i+n}^k \geq b_i^k \qquad \forall i \in P \cup D, k \in K, \tag{6}$$

$$b_j^k \geq b_i^k + ST_i + T_{ij}^k - M_{ij}^k(1 - x_{ij}^k) \qquad \forall i, j \in V, k \in K, \tag{7}$$

$$q_j^k \geq q_i^k + Q_j - N_{ij}^k(1 - x_{ij}^k) \qquad \forall i, j \in V, k \in K, \tag{8}$$

$$max\left\{0, Q_i\right\} \leq q_i^k \leq min\left\{C^k, C^k + Q_i\right\} \qquad \forall i \in V, k \in K, \tag{9}$$

$$M_{ij}^k \geq max\left\{0, L_i + D_i + T_{ij}^k - E_j\right\} \qquad \forall i, j \in V, k \in K, \tag{10}$$

$$N_{ij}^k \geq min\left\{C_k, C_k + q_i\right\} \qquad \forall i, j \in V, k \in K, \tag{11}$$

$$x_{ij}^k = 1 \Rightarrow b_i^k + ST_i \leq ET^k \qquad \forall i, j \in V, k \in K, \tag{12}$$

$$x_{ij}^k = 1 \Rightarrow b_i^k \geq BT^k \qquad \forall i, j \in V, k \in K, \tag{13}$$

$$q_i^k, b_i^k \geq 0 \qquad \forall i \in V, k \in K, \tag{14}$$

$$x_{ij}^k \in \{0, 1\} \qquad \forall i, j \in V, k \in K, \tag{15}$$

$$y_i^k \in \{0, 1\} \qquad \forall i \in P \cup D, k \in K. \tag{16}$$

4.5 Dynamism

The are numerous studies dedicated to defining dynamism formally. The first known formulation is by Lund et al. (1996). They define dynamism as ratio of dynamic requests over the total number of requests in a given problem instance, and denote it as *degree of dynamism* (*dod*), formally:

$$dod := \frac{n_{imm}}{n_{tot}},$$

Hereby, n_{imm} is the number of dynamic or immediate requests and n_{tot} is the number of all requests. This definition of dynamism is very simplistic, because it takes into account only one aspect, precisely whether a request is known in advance or not. As van Lon et al. (2016) notes, in a purely dynamic scenario, the *dod* would be always 1. In such cases, the *dod* could not be used to make any conclusions about dynamism. Hence, van Lon et al. (2016) proposes to call it "proportion of dynamic requests". This first definition is however a valuable milestone for subsequent studies on the topic of dynamism that endeavours to develop a more precise formulation, e.g. Larsen (2000), Larsen et al. (2007), Berbeglia et al. (2010), van Lon et al. (2016),

etc. Larsen (2000) remarks that the definition of Lund et al. (1996) has limitation as it does not consider the disclosure time, which is the point when requests become known to the planner. He proposes therefore another measure called the *effective degree of dynamism* (*edod*), and defines it formally as

$$edod := \frac{\sum_{i=1}^{n_{imm}} (\frac{t_i}{T})}{n_{tot}},$$

where T is the length of the planning horizon and t_i is the disclosure time of a request i, with $0 < t_i < T$. The dynamic or immediate requests are denoted by n_{imm} and n_{tot} is the number of all requests. This measure shows how late the requests are received on average compared to the last possible point in the planning horizon new requests can be accepted. The value of the *edod* is between 0 and 1. $edod = 0$ means a purely static problem, $edod = 1$ indicates a purely dynamic scenario, where no requests are known in advance. Larsen (2000) also proposes an extension of this definition that takes into account the impact of time windows. Depending on when a request is received (disclosure time) and when service can start at the latest (latest due date), a request has a specific reaction or response time. The *effective degree of dynamism with time windows* ($edod_{tw}$) is formally defined as

$$edod_{tw} := \frac{1}{n_{tot}} \sum_{i=1}^{n_{tot}} (1 - \frac{r_i}{n_{tot}}),$$

where r_i is the reaction time of a request i and $r_i = L_i - t_i$. Larsen (2000) makes the observation that the planner would obviously prefer longer reaction times for dynamic requests. A later study by Larsen et al. (2007) tries to create a taxonomy of dynamic vehicle routing problems, using the $edod_{tw}$ as a basis. According to this classification, weakly dynamic problems have a $edod_{tw}$ lower than 0.3, whilst strongly dynamic problems are characterized by a $edod_{tw}$ higher than 0.8. An $edod_{tw}$ between 0.3 and 0.8 describes a moderately dynamic problem.

5 Solution Approach

As mentioned in Berbeglia et al. (2010), it is a commonly used strategy to adapt a method used to solve the static version of the problem in order to tackle the dynamic one. The PDP is a generalisation of the NP-hard Travelling Salesmen Problem (TSP), which was first introduced in Garey and Johnson (1979). Therefore, it is also NP-hard and has a high computational complexity. More sophisticated heuristic approaches, e.g. metaheuristics, are known to produce a very good quality solution in an acceptable run time. As the presented variation belongs to the dynamic class of pickup and delivery problems, it requires a very quick optimisation and re-optimisation. Otherwise, a real-world application would not be possible under the so defined framework.

Hence, a heuristic method was used for solving the problem, namely a Large Neighbourhood Search metaheuristic (LNS), embedded in a dynamic framework. LNS was first proposed by Shaw (1998) and has been successfully applied to numerous variants of PDPs. The LNS implemented in this work is based on the papers by Ropke and Pisinger (2006) and Pisinger and Ropke (2010), whereby slight modifications were necessary with regard to the specific problem formulation.

5.1 General Dynamic Framework

The general solution framework is as follows. We create an initial solution with all requests known at the start of the planning horizon. Approximately 35% of the requests are known beforehand, and the rest arrives during the planing horizon. The initial solution is generated by using a greedy insertion operator and is improved with LNS. Incoming requests are also inserted with a greedy operator. The dynamic optimisation takes place every couple of minutes. All requests, for which

$$t_i <= t_{current} + 5 \tag{17}$$

is fulfilled, are identified as *pending requests*. Here, $t_{current}$ is the current time and t_i is the disclosure time of a request i. The destroy and repair operators are performed however only on those pending requests that are also *non-fixed*. Fixed requests are all completed requests as well as already known requests, for which

$$b_i^k + T_{ij}^k <= t_{current} \tag{18}$$

This states that for such requests the driver should be already on his way from his previous position j to arrive to location i at time b_i^k so that the feasibility of service start is not violated. As a delivery location must be serviced by the same vehicle that already visited the corresponding pickup vertex, the route for these delivery locations is then fixed as well and cannot be changed anymore.
The algorithm incorporates also a waiting strategy to improve request scheduling. We use a modified wait-first waiting strategy i.e. the waiting time at a location i before service start is 0. The driver waits at the predecessor location after finishing service. If the driver has to wait, he could be assigned additional requests before the next location. No detour is allowed when the driver is already on his way to the next location.

5.2 Large Neighbourhood Search

The LNS applied here is based on the one developed by Pisinger and Ropke (2010) and first proposed in Shaw (1998). Each neighbourhood examined is implicitly defined by a destroy and a repair heuristic. Part of the current request-to-driver assignments are deleted using a destroy method. The solution is then repaired with a respective repair operator. This destroy and repair scheme yields a new solution. During the LNS various destroy and repair heuristics are applied to make sure that, in every iteration, different parts of the solutions are destructed and are then rebuilt in a way that leads to a different new current solution. For this purpose, operators

not only use different criteria for deciding what part of the solution to destroy and how to consequently rebuilt it, but also include a certain randomness degree.

5.2.1 Algorithm

This section introduces the LNS algorithm as presented in Pisinger and Ropke (2010), and explains the individual steps in detail. The pseudo code of the LNS algorithm is shown in Algorithm 1 on the following page.

For the LNS, it is first necessary to build a feasible initial solution. The starting solution here is generated with a greedy insertion heuristic that is also used later as a repair operator. A detailed description of the greedy insertion's implementation can be found in Section 5.4.1. The initial solution is denoted as x. The best solution found during search is denoted as x^b. In line 4, part of the current solution x is destroyed ($d(x)$) and subsequently repaired ($r(d(x))$). In other words, some requests are first removed from the current solution and then inserted again. The resulting solution x' is the incumbent solution that can be accepted as a new current solution if the acceptance criteria are met. If that is the case, the current solution is updated. Otherwise, x' is discarded and no update takes place. Lines 8 and 9 determine whether the best solution is updated or not. The algorithm is repeated until meeting a given stop criterion. At the end, the algorithm returns x^b, which is the best solution found during the entire LNS.

Algorithm 1 Large Neighbourhood Search, Pisinger and Ropke [2010], p. 407)

1: input: a feasible solution x
2: $x^b = x$
3: **repeat**
4: $x' = r(d(x))$;
5: **if** accept (x',x) **then**
6: $x = x'$
7: **end if**
8: **if** $c(x') < c(x^b)$ **then**
9: $x^b = x'$
10: **end if**
11: **until** stop criterion is met
12: **return** x^b

5.2.2 Operator Selection

The selection of destroy and repair operators for the LNS is done on a roulette wheel principle. Following Ropke and Pisinger (2006), each heuristic i is assigned a weight w_i. This means that a given operator j is chosen with a probability of

$$\frac{w_i}{\sum_{i=1}^{k} w_i},$$

where k is the number of the insertion and destroy heuristics, respectively. It is important to note that the selection of insertion and destroy operators occurs independently from one another. As opposed to Ropke and Pisinger (2006), the weights remain unchanged during the entire algorithm, which means that each insertion heuristic has always an equal chance to be selected. The same applies to the destroy operators.

5.2.3 Acceptance Scheme

There are various possibilities to guide the LNS, i.e. how to decide which new solutions are to be accepted. Pisinger and Ropke (2010) notes that for the original LNS in Shaw (1998) only better solutions are accepted as new incumbent solutions. Rather than this simple descent approach, Ropke and Pisinger (2006) use another acceptance criteria that is based on simulated annealing, a probabilistic local search method first proposed in Kirkpatrick et al. (1983) and Černý (1985). Ropke and Pisinger (2006) argue that a pure descent approach that accepts only better solutions as new current solutions can easy lead to a local minimum that cannot be escaped from, hence the idea to allow also worse solutions sometimes.

A simulated annealing framework was used here for guiding the LNS, too. The objective function is minimising total costs of delays. Its value is denoted by $c(x)$. Better temporary solutions, i.e. $c(x') < c(x)$, are always accepted. A solution x' is accepted as new incumbent with certain probability, or more precisely if the probability $e^{(c(x_b)-c(x'))/T}$ is higher than a random number from the interval [0,1]. Hereby, T is the temperature which is decreased in each iteration by a cooling rate c, so that $T = cT$. The cooling rate is a number from the interval [0,T]. In this algorithm, T was set to 0.9.

5.2.4 Stopping Criteria

The algorithm stops after 25000 iterations or after 250 iterations without an improvement. A third stopping criterion is an objective function of zero.

5.3 Destroy Operators

Four different destroy heuristics are applied within the LNS framework in a similar fashion as the implementation of Ropke and Pisinger (2006).

5.3.1 Random Removal

All requests in the solution, i.e. all scheduled requests, are randomly shuffled, and then a random number q of them are deleted. Here q is set to be a random integer number between 20% and 30% of all requests that have already been inserted into the solution at the current time, and are marked as *non-fixed*. Non-fixed requests are explained at the beginning of Section 5.1 on page 19. The algorithm makes sure that at least one request is removed. Ropke and Pisinger (2006) also use a random removal heuristic, remarking that it can be regarded as a special case of the related removal operator proposed by Shaw (1998).

5.3.2 Worst Removal

The worst removal is applied as described in Ropke and Pisinger (2006). The intuition behind this operator is that poorly scheduled requests lead to a high objective function value and hence to a bad quality solutions. In order to identify "costly" requests, the hypothetical cost savings that would result from ejecting a request have to be computed. After calculating overall cost savings for all requests that are both already routed and marked as non-fixed, these requests are then sorted by descending cost savings. In the next step, a random integer number q of requests to be removed is drawn. Just as in the random removal operator, the algorithm deletes at least one request from the solution. q is defined as random number between 20% and 30% of all routed requests.

5.3.3 Critical Removal

The critical removal is a variation of the worst removal heuristic. It also aims to find the poorly scheduled requests, but instead of focusing only on "costly" requests, it also examines the other requests on the route. The idea is that requests with large waiting times are (more) likely to cause delays later on the route, and therefore the

predecessors of the "costly" requests could be scheduled better.

The algorithm first identifies delayed requests. These are marked as "critical" requests. For every critical request, its predecessors on the route are detected and the respective waiting times at each one of them are calculated. In the next step, one of the requests up to and including the critical requests is then deleted, whereby a larger waiting time increases the probability for removal. Deletion is allowed only for requests that are marked as non-fixed at the current time.

5.3.4 Related Removal

The related removal heuristic is applied after worst or critical removal. It was first introduced by Shaw (1998). As remarked by Ropke and Pisinger (2006), the intuition behind this neighbourhood is that interchanging variables that are somehow related to each other should be easy, and could lead to better solutions. The authors argue that removing requests that are very different and then reinserting them is very likely to yield the same solution as before or even to a worse one. Different measures of relatedness can be defined and used. For example, Ropke and Pisinger (2006) apply a relatedness measure where requests' similarity is defined as a combination of distance, time, capacity, and possibility to be served by the same vehicle. In Pisinger and Ropke (2010), customer demand is mentioned as another potential measure.

This idea is implemented also in the algorithm described here, although in a slightly different manner. Two relatedness measures are used. The first one is the distance between the pickup locations, and the second one is the beginning of the time window for pickup.

In the first step, the algorithm identifies requests that have been already removed, and saves them all in a random order. Following that, between one and five of these removed requests are chosen. According to the relatedness measures described above, five requests related by distance as well as five requests related by ready time are

erased from the solution. It must be noted that the requests that are considered for removal have to be already routed and marked as non-fixed at the current time.

5.4 Repair Operators

Following the framework of Ropke and Pisinger (2006), two different types of repair heuristics are used - greedy insertion and regret insertion.

5.4.1 Greedy Insertion

The greedy insertion is used both as a repair operator as well as for building the initial solution. Requests are sorted by their ready time. In the next step, the potential insertion in a route of every request, starting from the one with the earliest ready time, is checked for feasibility in terms of vehicle capacity and working hours of the driver. If the request is feasible and identified as a pending, non-fixed request at the current time, it is then marked as *available for insertion*. The algorithm proceeds to calculate its insertion cost for the given route, returning the position yielding the lowest insertion cost for the route. This calculation is done for all routes where feasible insertion can be conducted, and the request is inserted at the position and in the route, leading to the smallest increase in the objective function. Then this steps are repeated for the second request in the sorted request list and so on until all requests are routed.

5.4.2 Regret Insertion

In contrast to the greedy insertion, the regret insertion heuristic introduced in Ropke and Pisinger (2006) does not only consider what will happen in the next step or iteration. This operator is a so called far-sighted heuristic where the decision-making process is influenced by the future steps and iterations, respectively. In order to find out which insertion will be regretted the most if it is not done now, the regret

value must be calculated. This is the cost difference between inserting a given still unplanned request in the cheapest route and the second cheapest route. Following this pattern, one can calculate *3*-regret, *4*-regret or even k-regret values, whereby k is the number of routes. In this case of a k-regret insertion heuristic, all insertion possibilities are considered, and the difference in cost between the cheapest route and every other route is calculated and summed. The heuristic then inserts the request for which this sum of regret values is the highest.

The algorithm here differs somewhat from the application in Ropke and Pisinger (2006). In order to produce more diversified solutions, randomness was included. Three different variants of a regret operator were implemented. The first one incorporates a randomness measure for deciding what kind of a regret heuristic to use. The second one chooses for insertion not the request with the largest regret value, but a random request bias on regret values. The third regret operator is a *2*-regret insertion heuristic that does not use any randomness.

5.5 Random Restart

In order to additionally diversify and randomize the search, a random restart was implemented in the algorithm. All best solutions found during the LNS are saved in a pool. If no improvement is found in 250 iterations, i.e. the best solution is not updated for this number of iterations, the search is restarted using a randomly chosen solution from this pool with overruled best solutions. The random restart takes place only once during the LNS for a given current time. At the end of the LNS algorithm for each current time, the solution pool is emptied.

6 Results

6.1 Test Instances

20 test instances were generated from real-world data. One instance contains information for all requests on a given day. Due to a non-disclosure agreement with the courier company, we cannot provide detailed information about the requests. Each instance consists of a different number of requests and available vehicles. Table 2 on the facing page shows an overview with the number of requests and vehicles. It should be noted that the size of the fleet is fixed, i.e. its minimisation is not considered in this problem.

For each request, there are various known data parameters. The *input time* is the point in time when a request is entered into the system or in other words, when a request becomes known. Each incoming request has a specific ready and due date as well as a given weight and volume. In the instances, there are also different *pickup type* and *delivery types* that refer to the type of time windows for the pickup and delivery location, respectively. There are three different types of time windows in the instances. Table 3 on page 30 shows an analysis of the time windows of all locations over all isntance sets, broken down in intervals of 15 minutes.

As for the fleet, each vehicle has a specific *start time* and *end time*. Furthermore, as this problem deals with a heterogeneous fleet, vehicles also have a *type*, which refers to the different weight and volume capacity of a vehicle as well to its speed.

6.2 Computational Results

All experiments were conducted on a Lenovo Y50-70 with a processor core i7-4720HQ, processor speed 2.60 - 3.60GHz, and capacity RAM 16GB. This section gives detailed information about the computational results. It reports on the evaluation of the algorithm's performance and discloses various figures with regards to the objective

instance	vehicles	requests
1dyn	26	275
2dyn	29	258
3dyn	38	368
4dyn	41	469
5dyn	16	61
6dyn	41	497
7dyn	41	504
8dyn	35	456
9dyn	35	498
10dyn	33	449
11dyn	14	71
12dyn	36	500
13dyn	43	501
14dyn	46	438
15dyn	46	441
16dyn	44	415
17dyn	37	519
18dyn	31	470
19dyn	39	471
20dyn	36	500

Table 2: Number of vehicles and requests per instance

locations	% of total locations	time windows interval
666	4%	(0;15]
2770	17%	(15;30]
3780	23%	(30;45]
984	6%	(45;60]
8122	50%	(60;∞)

Table 3: Time windows

function. The section contains also information about the dynamism of the presented problem.

6.2.1 Measures of Dynamism

As presented in Section Table 4.5 on page 17, the dynamism of PDPs can be evaluated using different measures. Table 4 on page 32 shows the *dod* and $edod_{tw}$ for all instances. It is reasonable to use *dod* for these test instances as all of them contain some advance requests. If we take the proposal of van Lon et al. (2016) to name it *proportion of dynamic requests*, we see only two out of 20 instances have less than 50% dynamic requests. For the rest, the *dod* varies mostly between 55% and 75%, which indicates fairly dynamic instance sets.

The next column shows the respective $edod_{tw}$ values. The planning horizon used for calculating them is $T = 1200$. The results clearly show that the consideration of additional aspects translates to a significantly higher level of dynamism. Following the classification introduced by Larsen et al. (2007), there are no weakly dynamic instances. Nine out of 20 instances have a $edod_{tw}$ between 0.3 and 0.8, and are thus moderately dynamic. The other 11 instance sets are strongly dynamic with a $edod_{tw}$ value above 0.8. van Lon et al. (2016) point out that the usage of $edod_{tw}$ for

measuring dynamism is limited for scenarios with different lengths of the planning horizon. As we are defining it always as $\theta = 1200$, it is reasonable to argue it is a plausible measure of dynamism for the presented problem.

As noted by Pillac et al. (2013), both the *edod* and $edod_{tw}$ capture only dynamism related to time. Following the taxonomy in Psaraftis et al. (2015), the source of this type of dynamism is the appearance of new requests.

6.2.2 Additional Experiments

As described in Section 5.5, a random restart was implemented additionally to assess a potential further improvement of results despite the already very good performance of the algorithm that leads to optimal results for 6 out of the 20 instance sets. Table 5 on page 33 provides a comparison of the objective values for all instances, yielded with and without the random restart approach. The column *no rr* contains the results for the algorithm without random restart, whereas the column *rr* discloses the objective value for the implementation with random restart. For four of the instances, the LNS enhanced by the random restart approach manages to outperform the respective implementation without it. The improvement for two of the sets is as high as 57%. Therefore, the final results presented in the rest of the section are those of the algorithm enhanced by the random restart approach.

6.2.3 Algorithm Performance

As the test instances were generated based on real-world data, and tailored to the specific problem formulated in this work, no direct comparison with other solution approaches or problems is possible. Therefore, the algorithm performance and quality were evaluated by comparing the results of the dynamic problem and the static one. This measure is known as *value of of information* and was first introduced in Mitrović-Minić et al. (2004). Pillac et al. (2013) point out that it assesses the

inst.	**dynamic req.**	**total req.**	*dod*	$edod_{tw}$	**gap**
1dyn	174	275	0.6327	0.7789	0.1462
2dyn	164	258	0.6357	0.7845	0.1488
3dyn	249	368	0.6766	0.7984	0.1218
4dyn	258	469	0.5501	0.6865	0.1364
5dyn	9	61	0.1475	0.4489	0.3014
6dyn	359	497	0.7223	0.8265	0.1042
7dyn	404	554	0.7292	0.8416	0.1123
8dyn	308	456	0.6754	0.8091	0.1337
9dyn	357	498	0.7169	0.8350	0.1181
10dyn	251	449	0.5590	0.7232	0.1642
11dyn	9	71	0.1268	0.4471	0.3203
12dyn	321	500	0.6420	0.7825	0.1405
13dyn	339	501	0.6766	0.8072	0.1306
14dyn	316	438	0.7215	0.8311	0.1096
15dyn	311	441	0.7052	0.8179	0.1127
16dyn	265	415	0.6386	0.7851	0.1466
17dyn	356	519	0.6859	0.8113	0.1254
18dyn	351	470	0.7468	0.8523	0.1055
19dyn	350	471	0.7431	0.8501	0.1070
20dyn	350	500	0.7000	0.8168	0.1168

Table 4: Measures of dynamism

inst.	**no rr**	**rr**	**gap**	**% gap**
1dyn	0	0	0	0%
2dyn	0	0	0	0%
3dyn	0	0	0	0%
4dyn	9.56612	9.56612	0	0%
5dyn	0	0	0	0%
6dyn	0	0	0	0%
7dyn	14.2196	14.2196	0	0%
8dyn	2.788	1.20614	1.58186	57%
9dyn	71.4577	55.9259	15.5318	22%
10dyn	3.75754	3.75754	0	0%
11dyn	0	0	0	0%
12dyn	2.08673	2.08673	0	0%
13dyn	10.0551	10.0551	0	0%
14dyn	1.3915	1.3915	0	0%
15dyn	219.398	219.398	0	0%
16dyn	1.28297	1.28297	0	0%
17dyn	2.60092	2.60092	0	0%
18dyn	4.19394	1.7881	2.40584	57%
19dyn	53.1097	53.1097	0	0%
20dyn	4.26406	3.95738	0.30668	7%

Table 5: Objective value without and with random restart

influence of dynamism on the solution quality, without requiring an optimal solution for the static problem as opposed to other measures, such as the the *competitive analysis* proposed by Sleator and Tarjan (1985).

Table 6 on the next page and Table 7 on page 36 provide a comparison between the static and dynamic problem with regards to run time and objective value. The results used for the dynamic case are the ones for the algorithm containing the random restart. As shown in Table 6 on the next page, the algorithm finds an optimal solution for all instances but one in the static case. This is also the instance with the worst objective value in the dynamic case. It results from a delayed request with a time window from the most restrictive type. More details on that are given below in Section 6.2.4. As for the dynamic problem, the algorithm finds an optimal solution for 6 instances out of 20. The *value of information* is between 0% and 1.5%. Following the definition of this measure, we can conclude that the algorithm performance is not greatly affected by the dynamism of the instances.

6.2.4 Delays and penalty factor

As described in Section 4, the objective value is the sum over all delays multiplied with their respective penalty factor, α. The penalty cost is calculated based on how big the delay is, using the intervals defined in Table 8 on page 37.

Detailed results with regards to the objective value, minimising delays, are shown in Table 9 on page 38. Average, maximum, and total delay (in minutes) for each instance are reported. The last two columns contain the number of delayed requests and the number of delayed locations.

For six of the instances, including one of the large instances with 994 locations, the algorithm finds an optimal solution with regards to the objective value, i.e. none requests are delayed.

The average delay is calculated by adding up all delays in a instance and dividing

inst.	loc.	req.	veh.	OV dyn.	OV stat.	gap OV
1	550	275	26	0	0	0
2	516	258	29	0	0	0
3	736	368	38	0	0	0
4	938	469	41	9.56612	0	9.56612
5	122	61	16	0	0	0
6	994	497	41	0	0	0
7	1008	504	41	14.2196	0	14.2196
8	912	456	35	1.20614	0	1.20614
9	996	498	35	55.9259	0	55.9259
10	898	449	33	3.75754	0	3.75754
11	142	71	14	0	0	0
12	1000	500	36	2.08673	0	2.08673
13	1002	501	43	10.0551	0	10.0551
14	876	438	46	1.3915	0	1.3915
15	882	441	46	219.398	88.9113	130.4867
16	830	415	44	1.28297	0	1.28297
17	1038	519	37	2.60092	0	2.60092
18	940	470	31	1.7881	0	1.7881
19	942	471	39	53.1097	0	53.1097
20	1000	500	36	3.95738	0	3.95738

Table 6: Dynamic versus static problem - objective value

inst.	loc.	req.	veh.	CPU time dyn.	CPU time stat.	CPU time gap
1	550	275	26	1.608	1.241	0.367
2	516	258	29	1.313	1.134	0.179
3	736	368	38	2.966	2.966	0
4	938	469	41	156.367	93.397	62.97
5	122	61	16	0.161	0.160	0.001
6	994	497	41	6.588	6.588	0
7	1008	504	41	124.494	114.88	9.614
8	912	456	35	54.481	8.148	46.333
9	996	498	35	120.721	15.326	105.395
10	898	449	33	186.172	5.802	180.37
11	142	71	14	0.296	0.138	0.158
12	1000	500	36	193.208	6.306	186.902
13	1002	501	43	85.027	37.917	47.11
14	876	438	46	24.328	5.685	18.643
15	882	441	46	59.882	16.159	43.723
16	830	415	44	12.492	3.867	8.625
17	1038	519	37	124.896	18.795	106.101
18	940	470	31	77.626	62.15	15.476
19	942	471	39	79.337	7.874	71.463
20	1000	500	36	110.88	37.199	73.681

Table 7: Dynamic versus static problem - run time

factor	delay interval
1	(0;5]
2	(5;15]
3	(15;30]
4	(30;60]
5	(60;∞)

Table 8: Penalty cost

this sum by the number of delayed locations. In the cases, where there is only one delayed location, the average, maximum, and total delay are the same. The biggest delay is found for the instance $21dyn$, with only one location being delayed. In this case, not only the time windows are from the most restrictive type, but also the pickup and delivery location are geographically very far from each other. This is the only time to apply a penalty factor of 4.

The results show that for 10 out of 14 instances with delays, these occur exactly for requests with very restrictive time windows.

inst.	OV	loc.	veh.	req.	∅ delay [min]	delayed req.	delayed loc.	max. delay [min]	total delay [min]
1dyn	0	550	26	275	0	0	0	0	0
2dyn	0	516	29	258	0	0	0	0	0
3dyn	0	736	38	368	0	0	0	0	0
4dyn	9.56612	938	41	469	4.78306	1	2	4.85239	9.56612
5dyn	0	122	16	61	0	0	0	0	0
6dyn	0	994	41	497	0	0	0	0	0
7dyn	14.2196	1008	41	504	4.238065	2	2	5.74352	8.47613
8dyn	1.20614	912	35	456	1.20614	1	1	1.20614	1.20614
9dyn	55.9259	996	35	498	9.32099	2	3	10.0895	27.96297
10dyn	3.75754	898	33	449	3.75754	1	1	3.75754	315.208
11dyn	0	142	14	71	0	0	0	0	0
12dyn	2.08673	1000	36	500	2.08673	1	1	2.08673	2.08673
13dyn	10.0551	1002	43	501	5.02755	1	1	5.02755	5.02755
14dyn	1.3915	876	46	438	1.3915	1	1	1.3915	1.3915
15dyn	219.398	882	46	441	54.8495	1	1	54.8495	54.8495
16dyn	1.28297	830	44	415	1.28297	1	1	1.28297	1.28297
17dyn	2.60092	1038	37	519	2.60092	1	1	2.60092	2.60092
18dyn	1.7881	940	31	470	1.7881	1	1	1.7881	1.7881
19dyn	53.1097	942	39	471	17.7032	1	1	17.7032	17.7032
20dyn	3.95738	1000	36	500	1.97869	2	2	2.70015	3.95738

Table 9: Number of delayed requests; average, maximum delay, and total delay

7 Conclusion

In this work we study an extension of the classical PDP, which is inspired by a complex real-world application in courier operations. The problem formulated and tackled here is a dynamic PDP that has specific for courier services characteristics, such as heterogeneous and limited in size vehicle fleet, maximal working hours of drivers, and different time windows of requests. We present a mathematical model for the static version as well as relevant dynamic aspects.

The solution method proposed for solving the problem is a LNS, embedded in a dynamic framework and enhanced by an additional experiment denoted as random restart approach. The algorithm is tested on 20 instance sets, generated from real-life data of a big courier company. Each instance includes data on various parameters such as input, ready and due time of the requests, weight and volume of the requests as well as their different time windows. Information about the working hours of the drivers and the capacities of the different vehicle types is also available.

Results show a near-optimal performance of the solution method for all dynamic instance sets with regards to the specified objective to minimise tardiness, whereby the implementation containing the random restart approach improves those even further. Analysis of the dynamism of the test instances via two different measures shows that these are moderatly to highly dynamic. Nevertheless, it can be observed that the algorithm results for the dynamic setting are close to those for the corresponding static case, which indicates a very good performance.

A potential next optimisation step would be considering other goals, e.g., minimising the size of the fleet, which is currently declared as undesirable by the courier company because of long-standing cooperation with the drivers and the already established trust. The courier provider argued that the lost of the established trust and relationships would incur irreparable damage, which would make it impossible to work again with the drivers. At the same time, inefficiencies might be regarded

as a hurdle to the current intention of the courier company to expand its scope of services.

Abstract

The thesis is motivated by a problem arising in real-world same day courier services in urban areas. We propose and solve an extension of the dynamic pickup and delivery problem, tailored to the the specific problem faced by courier companies. The formulation introduced in this work considers additional components such as different time windows, maximal vehicle capacities and heterogeneous fleet. The objective of the proposed mathematical model is minimising delays. A metaheuristic based on a large neighbourhood search and embedded in a dynamic framework is developed and implemented for solving the problem. Dynamism of the test instances is analysed via two different measures. Algorithm performance is evaluated using real-world inspired test instances. Computational experiments show that the proposed solution method yields optimal or near-optimal results for all instance sets.

Zusammenfassung

Die Motivation zu dieser Masterarbeit bereitet das beim Kurierdienst auftretende Problem für Lieferungen im Stadtgebiet, die am selben Tag ausgeführt werden müssen. Hier wird eine Erweiterung des klassischen Pickup and Delivery Problem vorgeschlagen und gelöst, die auf das spezifische Problem zugeschnitten ist, vor dem Kurierunternehmen stehen. Die in dieser Arbeit vorgestellte Formulierung berücksichtigt zusätzliche Komponenten wie unterschiedliche Zeitfenster, maximale Fahrzeugkapazitäten und das Vorliegen einer heterogenen Flotte. Als Zielsetzung des vorgeschlagenen mathematischen Modells ist die Minimierung von Verzögerungen festgelegt. Eine Metaheuristik, die auf Large Neighbourhood Search basiert und in ein dynamisches Framework eingebettet ist, ist entwickelt und implementiert, um das Problem zu lösen. Für die Analyse der Dynamik der Testinstanzen werden zwei verschiedene Messwerte herangezogen. Die Algorithmusleistung wird auf mehreren von der realem

Welt inspirierten Testinstanzen ausgewertet. Die rechnerischen Untersuchungen zeigen optimale oder fast optimale Ergebnisse für alle Instanzen.

References

A. Attanasio, J.-F. Cordeau, G. Ghiani, and G. Laporte. Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing*, 30(3):377 – 387, 2004.

A. Attanasio, J. Bregman, G. Ghiani, and E. Manni. *Real-Time Fleet Management At Ecourier Ltd*, pages 219–238. Springer US, Boston, MA, 2007.

J. Barceló, H. Grzybowska, and S. Pardo. *Vehicle Routing And Scheduling Models, Simulation And City Logistics*, pages 163–195. Springer US, Boston, MA, 2007.

A. Beaudry, G. Laporte, T. Melo, and S. Nickel. Dynamic transportation of patients in hospitals. *OR Spectrum*, 32(1):77–107, 2010.

G. Berbeglia, J.-F. Cordeau, and G. Laporte. Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1):8–15, 2010.

G. Berbeglia, G. Pesant, and L.-M. Rousseau. Checking the feasibility of dial-a-ride instances using constraint programming. *Transportation Science*, 45(3):399–412, 2011.

G. Berbeglia, J.-F. Cordeau, and G. Laporte. A hybrid tabu search and constraint programming algorithm for the dynamic dial-a-ride problem. *INFORMS Journal on Computing*, 24(3):343–355, 2012.

V. Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985.

B. K.-S. Cheung, K. Choy, C.-L. Li, W. Shi, and J. Tang. Dynamic routing model and solution methods for fleet management with mobile technologies. *International*

Journal of Production Economics, 113(2):694 – 705, 2008. Special Section on Advanced Modeling and Innovative Design of Supply Chain.

J.-F. Cordeau and G. Laporte. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, 37(6):579 – 594, 2003.

J.-F. Cordeau, G. Desaulniers, J. Desrosiers, M. M. Solomon, and F. Soumis. Vrp with time windows. In P. Toth and D. Vigo, editors, *The vehicle routing problem*, pages 157–193. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

C. Fiegl and C. Pontow. Online scheduling of pick-up and delivery tasks in hospitals. *Journal of Biomedical Informatics*, 42(4):624 – 632, 2009.

M. R. Garey and D. S. Johnson. Computers and intractability: A guide to the theory of npcompleteness (series of books in the mathematical sciences), ed. *Computers and Intractability*, 340, 1979.

M. Gendreau, F. Guertin, J.-Y. Potvin, and R. Séguin. Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. *Transportation Research Part C: Emerging Technologies*, 14(3):157 – 174, 2006.

A. Goel and V. Gruhn. A general vehicle routing problem. *European Journal of Operational Research*, 191(3):650–660, 2008.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

A. Larsen. *The Dynamic Vehicle Routing Problem*. PhD thesis, Department of Mathematical Modelling, The Technical University of Denmark, 2000.

A. Larsen, O. B. Madsen, and M. M. Solomon. Classification of dynamic vehicle routing systems. In V. Zeimpekis, C. D. Tarantilis, G. M. Giaglis, and I. Minis, editors, *Dynamic Fleet Management: Concepts, Systems, Algorithms & Case Studies*, pages 19–40. Springer US, Boston, MA, 2007.

K. Lund, O. Madsen, and J. Rygaard. Vehicle routing problems with varying degrees of dynamism. Technical report, IMM Institute of Mathematical Modelling, Technical University of Denmark, 1996.

S. Mitrović-Minić and G. Laporte. Waiting strategies for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 38(7):635–655, 2004.

S. Mitrović-Minić, R. Krishnamurti, and G. Laporte. Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 38(8):669 – 685, 2004.

S. Parragh, K. Doerner, and R. Hartl. Part ii: Transportation between pickup and delivery locations. 58:81–117, 01 2008.

V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1 – 11, 2013.

D. Pisinger and S. Ropke. Large neighborhood search. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 399–419. Springer US, 2010.

H. N. Psaraftis, J. B. Orlin, D. Bienstock, and P. M. Thompson. Analysis and solution algorithms of sealift routing and scheduling problems. 1985.

H. N. Psaraftis, M. Wen, and C. Kontovas. Dynamic vehicle routing problems: Three decades and counting. 67, 08 2015.

V. Pureza and G. Laporte. Waiting and buffering strategies for the dynamic pickup and delivery problem with time windows. *INFOR: Information Systems and Operational Research*, 46(3):165–175, 2008.

S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4): 455–472, 2006.

S. Ropke, J.-F. Cordeau, and G. Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49(4):258–272, 2007.

M. W. P. Savelsbergh and M. Sol. Drive: Dynamic routing of independent vehicles. *Operations Research*, 46:474–490, 1998.

P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming*, CP '98, pages 417–431. Springer-Verlag, 1998.

D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

R. R. van Lon, E. Ferrante, A. E. Turgut, T. Wenseleers, G. Vanden Berghe, and T. Holvoet. Measures of dynamism and urgency in logistics. *European Journal of Operational Research*, 253(3):614–624, 2016.