Universiteit Antwerpen

DEPARTMENT OF ENGINEERING MANAGEMENT

The on-demand bus routing problem: A large neighborhood search heuristic for a dial-a-ride problem with bus station assignment

Lissa Melis & Kenneth Sörensen

UNIVERSITY OF ANTWERP
Faculty of Business and Economics
City Campus
Prinsstraat 13, B.226
B-2000 Antwerp
Tel. +32 (0)3 265 40 32
www.uantwerpen.be

AACSB ACCREDITED

FACULTY OF BUSINESS AND ECONOMICS

DEPARTMENT OF ENGINEERING MANAGEMENT

The on-demand bus routing problem: A large neighborhood search heuristic for a dial-a-ride problem with bus station assignment

Lissa Melis & Kenneth Sörensen

RESEARCH PAPER 2020-005
OCTOBER 2020

University of Antwerp, City Campus, Prinsstraat 13, B-2000 Antwerp, Belgium
Research Administration – room B.226
phone: (32) 3 265 40 32
e-mail: joeri.nys@uantwerpen.be

The research papers from the Faculty of Business and Economics are also available at www.repec.org (Research Papers in Economics - RePEc)

D/2020/1169/005

The on-demand bus routing problem: A large neighborhood search heuristic for a dial-a-ride problem with bus station assignment

Lissa Melis*, Kenneth Sörensen

October 28, 2020

Abstract

We introduce a novel optimization problem to support the planning and routing of on-demand buses in an urban context. We call this problem the on-demand bus routing problem. Given are a fleet of identical buses with fixed capacity, a set of bus stations and travel times between them, and a set of transportation requests. Each transportation request consists of a set of potential departure and a set of potential arrival bus stations, as well as a time window, i.e., an earliest departure time and a latest arrival time. The aim of the on-demand bus routing problem is to (1) assign each passenger to a departure and arrival bus station and (2) develop a set of bus routes to fulfill each request in time while minimizing the total travel time of all users.

We present the on-demand bus routing problem, as well as a straightforward large neighborhood search heuristic to solve it. The results found by the heuristic for the on-demand bus system are compared to those of a simulated traditional public bus system with fixed lines and timetables. A thorough analysis of the comparison demonstrates that total user ride times can be significantly lower in an on-demand public bus system and shows that an on-demand bus system works best with a large number of small buses.

Keywords— Transportation, Routing, Public transport, Metaheuristic

1 Introduction

Local public transport, defined as the collective scheduled transport of passengers, is one of the backbones of urban mobility within the European Union with 57.6 billion public transport journeys in 2014. 55.7% of these were road-based services (buses and trolleybuses) [International Association of Public Transport (UITP), 2018]. Figures like these, together with the growing need for sustainable mobility, underline the need for a strong commitment to provide citizens with a high-quality public transportation system.

A traditional bus transportation system is composed of a set of lines that follow predefined routes and schedules. Passengers traveling from one station to another on such a public bus system are required to plan their journey taking into account the fixed network and timetable. By their very nature, however, fixed bus routes are inefficient: even after meticulous analysis and planning, such routes can never be more than an extremely crude approximation of the "optimal" routes, i.e., the routes that transport passengers as quickly as possible from where they are to where they want to be.

The inflexible nature of the traditional bus system is not a good match for the poorly predictable travel patterns of a modern urban population [Nelson et al., 2010]. Traditional public transport systems

*Corresponding author, email: lissa.melis@uantwerpen.be

were mainly designed for commuters going back and forth between home and work. Nowadays, however, both professional and social activities are dispersed in space and time. In addition, cities are active 24/7 [Finn, 2012] and transportation outside of normal working hours is common. In many cities, the public transport system is not able to accommodate the needs of many of its residents (parents with children, e.g.), forcing them to use their private cars. Essentially, there is a mismatch between the flexibility of transport demand and the inflexibility of transport supply. Both during peak times and during off-peak times, the capacity offered by a traditional bus transport system does not match the ideal capacity. In 2013 the average occupancy rate of public transport (trams and buses) in Flanders (Belgium) was 24% [Vlaams Parlement, 2014]. Such numbers, however, belie the fact that buses are filled to the brim during peak hours and run almost empty during the rest of the day. Some lines work at full capacity, while others are barely used. These inefficiencies cause extra costs and the loss of potential customers.

It is therefore not surprising that the importance of flexible transport services has steadily increased over the last years. Taxi services, including Uber [Uber, 2019] provide a maximum of flexibility and speed, and allow the user to choose the exact time and location for departure and arrival. However, compared to collective public transport, it is expensive and inefficient in terms of resource usage, and therefore places a relatively large burden on the environment.

In general, the popularity of demand responsive transport (DRT) — which includes taxi services, but also buses and other vehicles that alter their route based on customer demand (such as the Turkish "dolmuş") — has increased considerably over the last decades. Originally a service for niche markets, like transportation of disabled and elderly persons, and regions with low demand like rural areas, such systems have grown to be a feeder system for passengers to reach the massive public transport network [Archetti et al., 2018, Alonso-González et al., 2018], motivating the introduction of the term *flexible transport services* (FTS) [Mulley and Nelson, 2009, Nelson et al., 2010]. Over the last few years several alternatives for collective public transport came into existence, mostly initiated by the private sector. Examples are car sharing, employee commuter programs, taxi sharing, and on-demand carpooling. These alternatives, together with DRT-systems, fit under the umbrella term *flexible urban transport* (FUT) [Finn, 2012]. Research has found that FUT systems perform better when offered on a large scale [Archetti et al., 2018, Jokinen et al., 2011].

Notwithstanding the increase in flexible urban transport options, most public bus services in cities remain inflexible and based on fixed routes and time tables. This can be largely attributed to the fact that, until recently, it was impractical or even impossible for a transportation supplier to obtain the necessary information on its customers' origin, destination, and preferred departure or arrival time. For the first time in human history, however, mobile devices that can fulfil this function have become ubiquitous and automated vehicle location (AVL) systems, which provide real-time information on the location of vehicles, are well established [Nelson et al., 2010, Mulley and Nelson, 2009, Mageean and Nelson, 2003]. This recent development allows, at least in theory, a large-scale shift towards *on-demand public transport*, i.e., a move from a system in which buses drive along fixed routes, to one in which buses drive along routes completely determined by passenger demand for transportation.

In a fully on-demand public bus transportation system, a user would send a *transportation request* by indicating his departure location (usually the current location of the user) and his arrival location as well as his preferred arrival time (possibly "as early as possible") using a mobile application. The system then responds by sending a proposal to the user telling him which bus station[1] he has to walk to and which bus will pick him up, as well as a estimated time of arrival. Upon confirmation, the booking in the system will be executed, delivering the passenger to his destination as efficiently as possible (e.g., with as little possible waiting time). Fig. 1 shows a mock-up of a mobile interface in such a (hypothetical) fully on-demand public transport system. Because of the varying demand for transportation over time, bus routes will be different every day and change throughout the day. A fully on-demand system will avoid unnecessary empty kilometres and improve the quality of service for users.

In the scientific literature, on-demand public transportation systems have typically been modelled as a so-called *dial-a-ride problem* (DARP). An important underlying assumption of those models, however, is that passengers can be picked up and dropped off at their preferred locations. In practice, and especially

[1]To avoid ambiguity we will use the term *bus station* to refer to the physical location where a passenger can get off or on a bus, and reserve the term *bus stop* for the action of stopping the bus to pick up or drop off passengers.

Figure 1: Mock-up of a mobile interface to a fully on-demand public bus system

in an urban context, however, this is not the case. For safety reasons, buses should only stop at clearly signposted bus stations that have been determined suitable.

An important decision in the design of such a system is whether or not to allow the users to choose their departure and arrival bus stations, or whether to let the planning system handle this decision. We argue that a considerable amount of efficiency can be gained by letting the planning system *assign* bus stations to the departure and arrival of the users, of course within a reasonable walking distance of the passengers' actual departure and arrival locations. Even though the assigned stations might not be the ones closest to the (departure or arrival) location of the user, this decision allows the system to pool users in space and time, avoiding time-consuming unnecessary stops of the bus. Moreover, detours can be avoided if passengers are assigned in an intelligent way to stops along the route of a bus (e.g., just asking a passenger to cross a road and wait at a bus station in the driving direction of the bus can make a large difference). By assigning bus stations to passengers, the system can more efficiently plan all of the transport requests, to the benefit of all users.

To conclude, we argue in this paper that a fully on-demand public bus system (in which bus routes are completely based on passenger requests for transportation) can have considerable benefits in an urban context *if* it is designed well. This means that users should not be allowed to choose their departure and arrival bus station, but that these stations should be *assigned* by the planning system. This additional decision requires an extension to the DARP models proposed in the literature, as the planning component in such a system needs to make two decisions simultaneously: (1) the assignment of arrival and departure bus stations, and (2) the routing of the buses. We have called the resulting optimization problem that combines both decisions the *on-demand bus routing problem* or ODBRP. To the best of our knowledge, this problem has not been described before in the literature.

In reality, the on-demand bus routing problem will be highly dynamic, with requests being introduced in the system on a near-continuous basis. As is common, however, we first study the static variant of the problem. In the static ODBRP all requests for transportation are known in advance and routes are determined before the buses start their tours. Nevertheless, we think the static problem is not without its use. First, solutions of the static ODBRP can be used in a later stage to benchmark the performance of algorithms for the dynamic ODBRP. Secondly, the static ODBRP might be used, e.g., by companies or groups of companies wishing to organize collective transport for their employees. In such a situation, it is reasonable to expect that users register their request for transportation some time in advance.

The remainder of this paper is arranged as follows. A literature review can be found in Section 2. In Section 3 we describe the ODBRP in detail. In Section 4 a straightforward heuristic is presented to solve the ODBRP. Afterwards, in Section 5 we compare the results of an on-demand system based on the ODBRP

to those of a traditional public transport network with fixed lines using the developed heuristic. Finally, in Section 6, we conclude and discuss some opportunities for future research.

2 Literature review

Public bus planning typically occurs in several stages, which are represented in Fig. 2. The planning process usually is sequential, with the output of a higher stage serving as the input for a lower one.

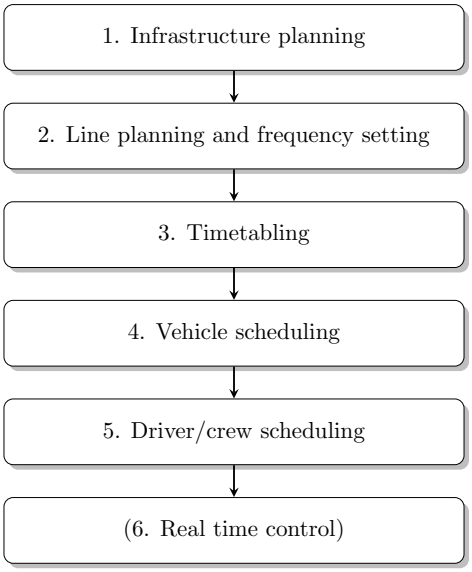


Figure 2: Classical order of public bus transport planning [Ceder and Wilson, 1986, Ibarra-Rojas et al., 2015]

The first decision is a long-term, strategical decision and covers the *infrastructure* (bus stations, number and capacity of the buses, ...). Decisions 2 and 3 are on a tactical level and include *line planning*, *frequency setting*, and *timetabling*. Line planning defines the paths (i.e., the sequence of stops) traveled by the buses, whereas frequency setting determines how many buses per hour will drive on each of the lines. For a review on line planning in public transportation we refer to Schöbel [2012]. Timetabling involves determining the exact times at which the buses will arrive and depart at the various stops in the line. A literature review on timetabling can be found in Ibarra-Rojas et al. [2015]. Decisions 4 and 5 assign the main resources of a public bus system, and are called *vehicle scheduling* and *driver/crew scheduling*. These decisions take place on an operational level. The first covers the assignment of vehicles to fulfill all trips planned and the second appoints drivers to vehicles. For a review on vehicle scheduling we refer to Bunte and Kliewer [2009] and Ibarra-Rojas et al. [2015]. For an overview on driver/crew scheduling we refer to Wren and Rousseau [1995] and Ibarra-Rojas et al. [2015]. Real time control (decision 6) takes into account unpredictability, e.g., traffic jams, and adjust the plan accordingly.

In the recent literature, a focus is on integrating and jointly optimizing several stages in order to obtain better solutions. Carosi et al. [2019] integrate stages 3 and 4 (timetabling and vehicle scheduling) into a single model solved by a matheuristic, while Michaelis and Schöbel [2009], Schöbel [2017], and Lübbecke et al. [2019] integrate stages 2 (line planning and frequency setting) to 4 or even 5 (driver scheduling). The ODBRP proposed in this paper integrates stages 2 to 5 into a single decision problem. As a consequence the number of stages is reduced to three: infrastructure, planning, and possibly control. The latter will be especially important when tackling the dynamic ODBRP, but this is, as mentioned, beyond the scope of this

paper.

Westerlund et al. [2000] and Mageean and Nelson [2003] make a distinction between different DRT systems based on the flexibility of the routes and the level of (un)certainty a station would be visited. Traditional public bus systems are situated on the end of the continuum where intermediate stops and terminals are fixed and fixed schedules and routes are used. On the other end of the continuum we have the classical DARP, with door-to-door transportation, and therefore non-predefined stops. The ODBRP is situated in between these two extremes: bus stations are predefined, but bus routes are not.

We expect that the ODBRP will be especially useful in the context of autonomous vehicles. Winter et al. [2018] demonstrate that a complete automation of the bus fleet would allow for substantial cost savings as driver costs vanish and the constraints of driver/crew scheduling (the fifth stage in Fig. 2) disappear completely. As DRT systems commonly fail because of excessive costs, autonomous vehicles might tip the balance in favor of an on-demand system [Enoch et al., 2006].

In terms of problem structure, the ODBRP is a new optimization problem which combines elements of two existing problems: the *dial-a-ride problem* (DARP) and the *school bus routing problem* (SBRP). In the DARP vehicle routes need to be defined in order to pick up and drop off passengers with the aim to minimize the total cost of transportation. It is implicitly assumed that pickup and drop-off can take place at the location of the passenger, i.e., the DARP essentially models a door-to-door service. Often, each event (pickup or drop-off) has a time window, and each user has a maximum user ride time. A final constraint often found in the DARP is that the capacity of the vehicle cannot be exceeded. For a recent overview and more detailed definition of the DARP we refer to Cordeau and Laporte [2007] and Molenbruch et al. [2017].

While the DARP is useful in planning an on-demand bus service, it lacks a feature necessary to render it useful in an urban context: bus station (or stop) assignment. In the realm of vehicle routing problems, this feature was introduced in the school bus routing problem (SBRP), originally formulated by Schittekat et al. [2013] and motivated by a real problem faced by the Flemish transportation company De Lijn. In the SBRP, a set of students is given, together with a list of stations each student can walk to. The objective is to assign students to bus stations and to find bus routes to visit these bus stations to pick up all students and bring them to school, while minimizing the total distance traveled and respecting the capacity of the buses. No individual time windows are defined in the original SBRP, but the solution should respect the starting time of the school. For a recent review on the SBRP we refer to Ellegood et al. [2019].

In Table 1 the most important characteristics of both the DARP and the SBRP are displayed side by side. Cells with grey background correspond to the features of the ODBRP, with a small remark being that the DARP usually has a different time window for the pickup and the drop-off event, while the ODBRP has a single time window within which the passenger needs to be both picked up and dropped off.

Table 1: Features of the DARP and the SBRP, grey cells contain features these problems share with the ODBRP

DARP	**SBRP**
Time windows	No time windows
Capacity constraints	Capacity constraints
Multiple stations for departure and arrival	Multiple stations for departure, only 1 arrival station
Pickup/drop-off anywhere	Pickup at bus station
No bus station assignment	Bus station assignment

3 Problem description and formulation

In Section 3.1 a detailed description of the ODBRP is given using an example as well as a description of an instance and a solution. In Section 3.2, the constraints of the problem are formally presented. The complete

mathematical model can be found in Appendix A.

3.1 Problem instance and solution

In the ODBRP, a set of bus stations is given, together with a set of buses with identical capacity. A travel matrix, expressed in units of time, between each pair of bus stations is also given, as is a set of *transportation requests*. A transportation request is issued for a single passenger and consists of a set of origin/departure bus stations and a set of destination/arrival bus stations, along with a time window within which a passenger needs to be transported[2]. A passenger can only be picked up at one of his origin bus stations after his earliest departure time and has to arrive at one of his possible destination stations before his latest arrival time.

Depending on the specific situation, three types of objective functions are possible for the ODBRP. The first type of objective function is profit-related. When determining the routes of the buses we could minimize the travel costs (or total distance traveled), maximize the profit of the system (based on the revenues of requests), minimize the fleet size, minimize total driver wages, etc. A second type of objective function is completion-related. Assuming it is mostly impossible to fulfill all the requests, the number of passengers served can be maximized. The last objective function type is quality-related. In this category are objective functions that minimize total delay, total waiting time, total or maximum user ride time, the number of passengers with delay, passenger travel time (including walking to a stop), ... Delay is defined as the amount of time a passenger arrives at the destination station after his latest arrival time. Waiting time represents the time between the earliest departure time and the pickup time at the origin station. User ride time is the time between pickup and drop-off, when a passenger is actually on the bus, while travel time is the user ride time plus the waiting time and the walking time to/from stations.

The choice of objective function will most likely hinge on the specific situation in which the ODBRP is solved, and may even change during the day. During off-peak hours, e.g., when demand for transportation is lower, the fleet of buses is presumably large enough to serve all passengers. In this case it might be more interesting to consider profit- or quality-related objective functions. During rush hours, when demand is high, it is perhaps impossible to fulfill all requests for transportation and a completion-related objective function makes more sense. In the remainder of this paper, we will use a quality-related objective function, namely *the minimization of total user ride time (URT)*. This implies that all requests for transportation should be planned for a solution to be feasible.

Note that walking time from the actual origin location to an origin station or from a destination station to the actual arrival location is not optimized and therefore not included in the time window (i.e., it is assumed that all departure bus stations are within a maximum walking distance of the origin location of the passenger, and all destination bus stations are within a maximum walking distance of the actual arrival location of the passenger, and that the earliest departure/latest arrival time is the same at all departure/arrival stations). It would not be difficult to model the walking time of the passenger in the model and include it in the time window. Even though this would entail different earliest departure and latest arrival times at the various possible bus stations that can be assigned to a given passenger, the impact on the model and the algorithm would be minimal. In this work the total URT, which is the time passengers spend on the bus, and not the total travel time, is minimized because this is the most logical option from the bus company's point of view. Walking speeds may be different for each passenger and preferences with respect to walking can differ. Of course the maximum walking time forbids the walking time to become too long. The same maximum walking time is currently used for every request, although this can be personalized according to the passenger in the future. In conclusion, the decision to not include the walking time in the objective function of the model is a design decision that we deem sensible, but that can easily be altered should this be required by the specific context.

Figure 3 shows a visual representation of a small instance of the ODBRP with one bus, eight bus stations, and two users (A and B). Passenger A can walk to three bus stations for pickup and can be dropped off at two different bus stations. Also, each passenger has a time window. In this case passenger A can only be picked up after 10:10 and needs to arrive at a destination station before 10:30. Passenger B is able to walk to two bus stations for his pick up, which has to be after 10:00. For the arrival of passenger B, two stations

[2]We will use the terms origin and departure, as well as the terms destination and arrival interchangeably.

are possible and the arrival has to happen before 10:40. A solution consists of two simultaneous decisions: (1) *bus station assignment/selection*: assigning a departure and arrival bus station within walking distance to each passenger, and (2) *routing*: defining the sequence of bus stations visited by each bus.

In Fig. 4 a feasible solution is represented for the instance in Fig. 3. In this case passenger A walks to the bus station in the northeast, takes the grey bus line and gets off the bus at the last stop. The time aspect is indicated in the figure in the boxes. Passenger A is picked up at his earliest departure time and is dropped off at 10:20 which is before his latest arrival time. Bus station assignment results in more flexibility for the routing as passenger pickup and drop-off can be grouped at bus stations to avoid extra stops during the route. In comparison, Fig. 5 visualizes a solution without bus station assignment. In this case passengers can choose their own bus stations for departure and arrival, e.g., the stations closest to their actual departure and arrival locations. This results in less possibilities for efficient routing, and therefore longer bus routes.

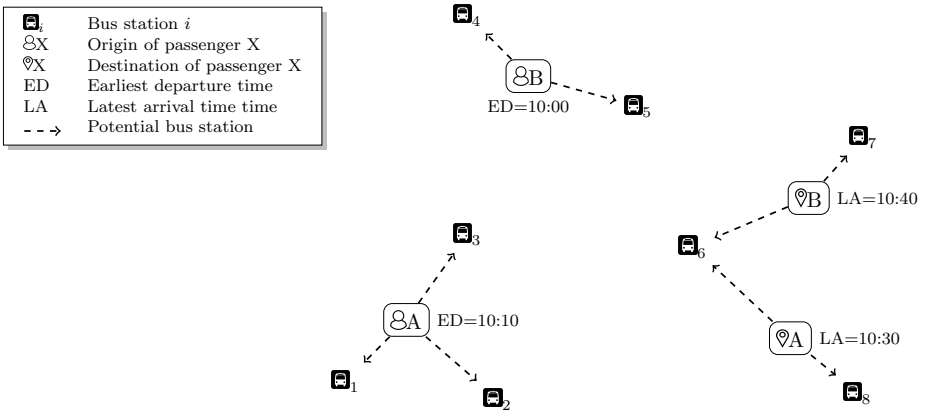


Figure 3: Example instance with eight bus stations, one bus, and two passengers

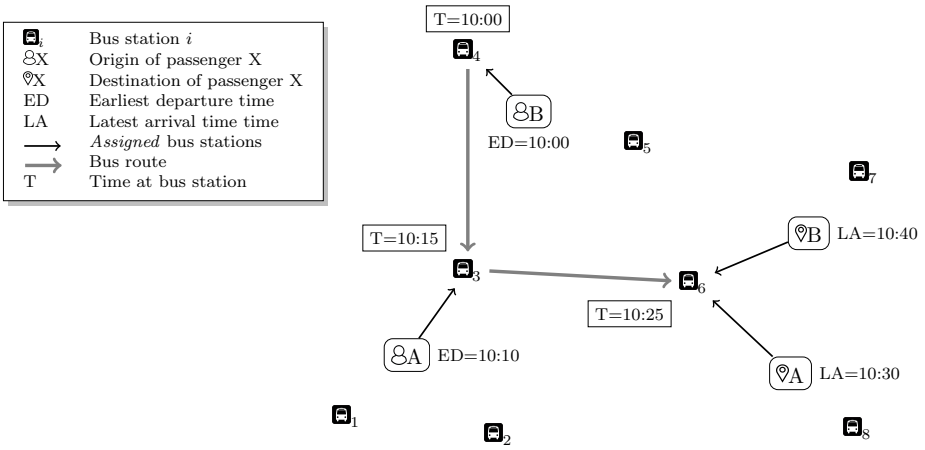


Figure 4: Example solution *with* bus station assignment

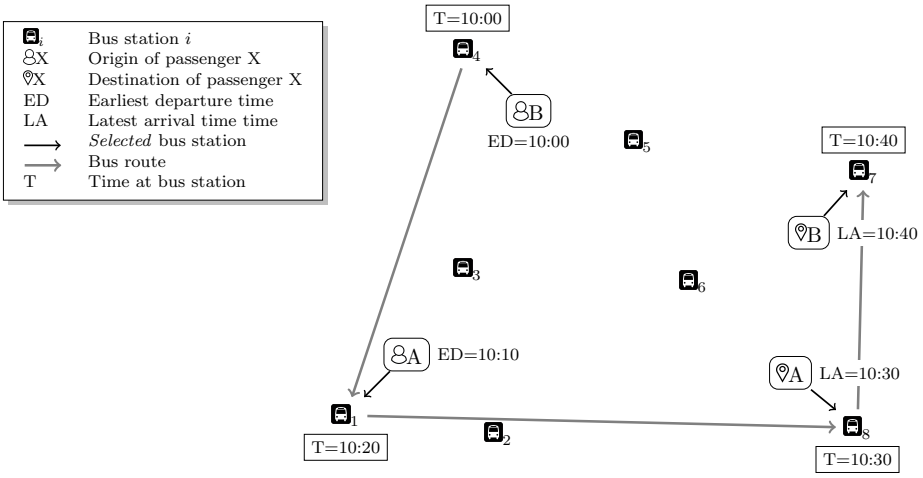


Figure 5: Example solution *without* bus station assignment

3.2 Formal problem formulation and constraints

In this section, we present a formal description for the ODBRP with the minimization of the total user ride time as the objective function. The mathematical model can be found in Appendix A. Table 2 contains the symbols used in the formal description and mathematical model. Using these symbols, the ODBRP can be formulated as follows:

In the ODBRP a fleet B of identical buses is used to fulfill a set R of requests for transportation by visiting bus stations from the set of stations S. The objective is to minimize the total user ride time (URT). Figure 6 shows an example of a bus route. Every bus $b \in B$ has a capacity C and the bus route of bus b consists of a number of filled positions $\{n_1, n_2, \ldots, n_{|N|}\} \in N$. If position n_i is filled with a station $s \in S$, it is called a bus stop. This makes a bus route a consecutive sequence of bus stops of which the first stop is located at the first position. Each position in the solution can only be served once, which means that a bus can only stop at one station at the same time, but one bus station can be served multiple times (even by the same bus). A solution is then a set of $|B|$ different bus routes. In addition, at least one event (a pickup or a drop-off) is linked to each bus stop: a passenger getting on and/or off the bus. A bus never stops without reason.

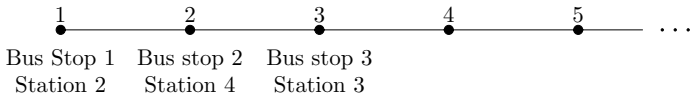


Figure 6: A bus route in the ODBRP

A request $r_p \in R$ from passenger p consists of a time window (an earliest departure time e_p^u and a latest arrival time l_p^o at an origin and destination station respectively), a list of stations $S_p^u \in S$ within walking distance of the origin location and a set of stations $S_p^o \in S$ within walking distance of the arrival location of passenger p.

A solution of the ODBRP consists of a set of assignments of passengers to departure and arrival stations, and a set of routes visiting those stations. Bus stops have a time window linked by logical relations. If s is the n-th stop of the bus and s' is the $n+1$-th stop of the bus, then the time of arrival at s' should be greater than the departure time at s plus the travel distance between stops s and s'. Also, the departure time (arrival time) of the bus at the n-th stop is not smaller than the earliest departure time (latest arrival time) of a passenger assigned to that stop. Figure 7 shows an example of the time aspect at a bus stop.

Table 2: Variables and parameters of the ODBRP

	Discrete decision variables
x_{snb}	1 if the n-th stop of bus b is bus station s and 0 otherwise
y^u_{pnb}	1 if passenger p is picked up at the n-th stop of bus b and 0 otherwise
y^o_{pnb}	1 if passenger p is dropped off at the n-th stop of bus b and 0 otherwise
q_{nb}	net number of passengers picked up (or dropped off) at the n-th stop of bus b
	Continuous decision variables
t^a_{nb}	arrival time of bus b at its n-th stop
t^d_{nb}	departure time of bus b at its n-th stop
T_p	user ride time of passenger p
	Parameters
B	the fleet of buses
R	the set of transportation requests, $\|R\|$ denotes the number of requests
S	the set of bus stations
C	capacity of a bus
a^u_{ps}	1 if passenger p can be assigned to stop s for pick-up
a^o_{ps}	1 if passenger p can be assigned to stop s for drop-off
e^u_p	earliest pick-up time for passenger p
l^o_p	latest drop-off time for passenger p
$\tau_{ss'}$	travel time between stop s and stop s'
N	positions in the representation of the bus tours, $\|N\|$ denotes the number of positions available

Three passengers are picked up and two passengers are dropped off. The maximal latest arrival time of the passengers getting off the bus, in this case the latest arrival time of passenger 2, needs to be earlier than or equal to the arrival time at this bus stop. Similarly, the maximal earliest departure time of the passengers getting on the bus, needs to be later than or equal to the departure time of the bus at this stop. This allows a bus to wait at a bus station for extra passengers to get on the bus before leaving for the next stop.

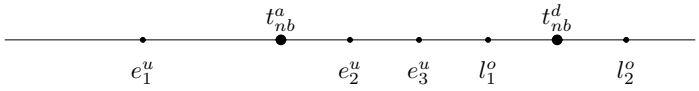


Figure 7: Time aspect at bus stop

A request p is served when a station from the set S^u_p is scheduled at an earlier position than a station from S^o_p in the same bus route, the event of this passenger getting on the bus at the stop from S^u_p happens after e^u_p and the event of the passenger getting off the bus at the stop from S^o_p happens before l^o_p. Of course a passengers is picked up before he can be dropped off, and both events need to be executed by the same bus.

Every bus has a capacity C. At no point can there be more than C passengers on the bus, while every request needs to be served to have a feasible solution. If a passenger is picked up at the n-th position of bus b and dropped off at the n'-th position of this bus, the URT of this passenger is defined by the arrival time of bus b at the n'-th position minus the departure time of this bus at the n-th stop. The sum of all the URT's of all passengers is minimized.

4 A heuristic for the ODBRP

The ODBRP is a generalization of both the DARP and the SBRP, both of which are NP-hard. As a result, it is highly unlikely that an exact method will be able to solve instances of realistic size. A naive implementation of the mathematical model (found in Appendix A) in a commercial solver did not yield any useful results but for the smallest of instances. Solving realistic instances of the ODBRP will definitely require a heuristic. In recent literature large neighborhood search heuristics are successfully used for the DARP, e.g., Gschwind and Drexl [2019], Tellez et al. [2018] Masson et al. [2014], and Parragh and Schmid [2013], as well as for other ridesharing problems, e.g., by Adawiyah et al. [2019]. With these examples in mind we propose a straightforward large neighborhood search heuristic. The aim of this heuristic is not to achieve maximum performance, something which is left for future research, but rather to present an intuitive and easy-to-understand solution method which may be used in our experiments (Section 5) to quantify the benefits of an on-demand public bus system.

Our method uses a greedy constructive heuristic both to generate an initial solution and to repair a partially destroyed solution. The constructive heuristic adds customers, one by one, in an intelligent and greedy manner. The large neighborhood search (LNS) proper iteratively destroys and repairs a solution until a termination criterion (either a time limit or a limit on the number of iterations) is met. Each iteration, after partly destroying and repairing the solution and if the solution is still feasible, some local search is performed. We first explain the constructive heuristic and then discuss how it is embedded in the large neighborhood search framework.

4.1 Constructive heuristic

To build the initial solution, first all passenger requests are sorted on a *priority list* in ascending order of their latest arrival times. As we are not minimizing the number of buses used, all buses can operate. Starting from the top of the priority list, each request is allocated to one available bus, until all buses have one allocated request. Then, requests are added one by one in a greedy manner.

Both for the origin and the destination of a request, every possible insertion position is checked in every bus route. Algorithm 1 shows in pseudo-code how a request is added to the solution. To illustrate the algorithm a toy example is used consisting of one available bus with capacity 8 and 2 requests. The priority list is shown in Table 3. For each request the following numbers are indicated: the passenger number, the earliest departure time, the latest arrival time, the possible stations for departure, and the possible stations for arrival. The distance matrix between the stations is given in Table 4.

As mentioned above the constructive heuristic first schedules one initial request in every available bus. Because the toy example has a fleet size of one bus, the first request is already planned in this bus. The outcome is visible in Fig. 8. Bus stations 3 and 6 are chosen because they result in the smallest user ride time. The bus can only depart at time 40, because this is the earliest departure time of the first passenger. The distance between the two stops is equal to 10.

Table 3: Toy example - Priority list

p	e_p^u	l_p^o	Departure stations	Arrival stations
1	40	100	1, 2, or 3	5 or 6
2	30	130	7, 8, or 9	10 or 11

When adding the request of passenger 2 to the solution, we start by finding the first possible insertion position in the current solution to add the origin of passenger 2. In the example there are three possible origin insertion positions: before the first stop, in between stop 1 and 2, or after the last stop. We start by looking at the first: before the first stop. Because of bus station assignment, the best origin stop is chosen from the list of possible origin stations of passenger 2, to enter in this insertion position. Because we insert before the existing route starts, there is only one existing neighboring station: station 3. The station

Table 4: Toy example - Distance matrix

	1	2	3	4	5	6	7	8	9	10	11
1	0	5	5	20	15	13	20	25	20	35	35
2		0	5	20	15	14	15	15	20	35	35
3			0	20	15	10	12	14	14	25	30
4				0	30	35	40	40	40	60	60
5					0	5	30	28	32	40	41
6						0	30	28	32	40	38
7							0	5	5	40	45
8								0	5	45	50
9									0	45	50
10										0	5
11											0

Bus stop	1	2
Bus station	3	6
Arrival time	40	50
Departure time	40	50
P on	1	
P off		1

Figure 8: Toy example - Initial bus route

causing the smallest increase in route duration is chosen. In the best case station 3 would be present in the set of potential departure stations of passenger 2. This way, there would be no increase in route duration. However this is not the case in this example. In this example, bus station number 7 is closest to 3 and therefore chosen as best origin station for this insertion. Figure 9 shows this possible origin insertion with an asterisk.

Bus stop	*	1	2
Bus station	7	3	6
Arrival time	30	42	52
Departure time	30	42	52
P on	2	1	
P off			1

Figure 9: Toy example - Possible origin insertion

Every time a potential insertion location for the origin of a request is examined, the feasibility of the origin insertion is checked. The rationale for this is that, when this departure insertion is infeasible, it is pointless to look for an accompanying arrival insertion position. The feasibility check examines both the capacity and the time window violations of the possible insertion. The capacity check is trivial, the current capacity at every scheduled stop of every bus is being tracked. When an origin insertion is being checked, only the capacity at the departure stop is checked. If this stop is already at full capacity, continuing with

Algorithm 1: Constructive heuristic: Add one passenger request

```
for Every bus b do
    for Every position n in the bus do
        Find best origin station ;
        Check feasibility (time windows and capacity violations) ;
        if Feasible origin insertion then
            for Every position ≥ n in bus b do
                Find best arrival station ;
                Check feasibility (time window and capacity violations) ;
                Calculate insertion criterion;
                if Feasible and insertion criterion is smaller then
                    Save this insertion ;
Perform best insertion ;
```

this origin insertion is not possible. However the full capacity check can only be done when trying to insert the arrival of a request. Only then it is known between what range the capacity needs to be checked. The range goes from the departure insertion position up to the arrival insertion position. Checking the time window constraints requires some more calculations. When inserting a stop before, between or after existing bus stops, a detour occurs, which is calculated in minutes using the travel matrix. Once the detour has been calculated the algorithm examines whether it would cause any existing passengers in the route to arrive at their destination after their latest arrival time. If this is the case, the insertion is infeasible.

In the example we calculate a detour of length 2, i.e., the bus needs to drive 2 additional minutes when a new stop is inserted like in Fig. 9. Because the latest arrival time of passenger 1 is 100, the insertion would be feasible. This means we can continue by looking for an arrival insertion position of which the position has to be larger than the one of the origin insertion position, as in the bus route drop-off has to occur after the pickup of a passenger. There are three possible insertion positions for the arrival of this passenger: immediately following the departure position, between stop 1 and 2, or after stop 2.

For each possible insertion position, the algorithm looks for the best possible arrival stop for passenger 2. Figure 10 illustrates the first possible insertion of the arrival. The best arrival station for this insertion is station 10, because it extends the route the least. The rationale behind this is when choosing the best station for a certain insertion, minimizing the route extension automatically increases the user ride times of already existing passengers in the route the least. Again, we have to do a feasibility check first. The capacity check is positive, but the time window check is not. This insertion would cause passenger 1 to arrive at time 105, after her latest arrival time, and is therefore infeasible. The constructive heuristic will always try to find a feasible insertion. If none is found, the passenger is added to a list of unserved requests. In the LNS-part of the algorithm, we try to add the passengers present in this list into the solution once again. Figure 11 is an example of a feasible arrival insertion.

Bus stop	*	*	1	2
Bus station	7	10	3	6
Arrival time	30	70	95	105
Departure time	30	70	95	105
P on	2		1	
P off		2		1

Figure 10: Toy example - Infeasible arrival insertion

Bus stop	*	1	2	*
Bus station	7	3	6	11
Arrival time	30	42	52	90
Departure time	30	42	52	90
P on	2	1		
P off			1	2

Figure 11: Toy example - Feasible arrival insertion

The previous paragraphs have described how the algorithm checks the feasibility of possible insertions of arrival and destination stops of a customer in a route. To decide which of the possible feasible insertions will be performed, an *insertion criterion* is used consisting of three parts. As inserting a new request might cause a detour or longer URT for the requests already planned in a route, the first part of the insertion criterion is the increase in total URT caused by this possible insertion. The second part is the URT of the passenger of the request that is going to be inserted. Thirdly, a penalty factor ensures that not all requests are planned consecutively. Figure 12 illustrates consecutive placing of requests. Passenger 1 is picked up and dropped off, then passenger 2 is picked up and dropped off. User ride times are minimal, as the URT of each passenger is equal to the direct distance from a passengers' origin station to an arrival station in minutes. However, this would not be the most efficient when a large number of requests need to be inserted, possibly causing problems to plan every request, knowing that there is a limited bus fleet.

Bus stop	1	2	3	4
Bus station	3	6	8	10
Arrival time	40	50	78	123
Departure time	40	50	78	123
P on	1		2	
P off		1		2

Figure 12: Toy example - Consecutive placing of requests

The penalty factor mentioned above contains only delays caused by inserting the origin bus stop of the passenger (e.g. the 2 minute delay shown in Fig. 9) or increases of the route length caused by the insertion of either the origin or the arrival bus stop of the passenger (an extension of the route of 38 minutes by inserting station 11, as illustrated in Fig. 11) to avoid consecutive placing of stops. For the first block of requests, the penalty factor is multiplied by M. The magnitude of this block of requests is chosen to be five times the fleet size. M is a random number between 0 and 1 to make sure the weight of the penalty factor varies and is not too large. This is important as the penalty factor is not actually optimizing the objective function, as it does not contribute to minimizing the total URT. After the first block of requests is inserted, M remains zero, i.e., no more penalty factor is used. In the example shown in Fig. 11 the insertion criterion would be $0 + 60 + (2 + 38)M$. The first part is zero, because the insertion does not cause the existing total URT in the route to rise. The second part is the penalty factor for making the route longer and the last part is the URT of the request that is to be added. Depending on M, the value of this criterion is between 60 and 100. In the example where requests are consecutively placed (see Fig. 12), the insertion criterion equals $0 + 45 + (28 + 45)M$. The first part is zero again, with the same reason as in the previous example. The second part is the penalty factor for extending the route and the last part is the user ride time of the newly added requests, namely the request of passenger 2. Depending on M the insertion criterion is between 45 and 118. When one has to choose between the insertion implemented in Fig. 11 or the one in Fig. 12, it would depend on M. If M would be 1, we would choose the first insertion. However when this would be an insertion after the first block of requests, and M would be zero, we would choose consecutive placing of the

requests.

When inserting stops either as the first or the last stop of a route under construction, the route length increases and some idle time might occur. Idle time occurs when the bus needs to wait for passengers to get on the bus, i.e., when the arrival time of the bus at a certain stop occurs before the earliest departure time of a passenger who is assigned to get on the bus at this stop. For example, if the earliest departure time of passenger 2 would have been 20 and assuming all other circumstances are the same, a waiting time of 8 would occur, as is illustrated in Fig. 13. In the figure we can see that this insertion would not cause the route to be delayed, thus idle time is not added in the penalty factor, it is incorporated in the URT of the passenger that is added to the route.

After checking all combinations with the first possible origin insertion, all other possible origin insertions (and accompanying possible arrival insertions) in this and potentially other bus routes are also checked. The insertion with the smallest insertion criterion is performed. This is repeated for all requests.

Bus stop	*	1	2
Bus station	7	3	6
Arrival time	20	32	50
Departure time	20	40	50
P on	2	1	
P off			1

Figure 13: Toy example - Example origin insertion with waiting time

4.2 Large neighborhood search (LNS)

To further improve the solution, the constructive heuristic (Section 4.1) is embedded in a large neighborhood search (LNS) framework. The main idea underlying this framework (also called *destroy-and-repair* or *ruin-and-recreate*) is to iteratively destroy (part of) the solution and rebuild it using a destructive/constructive operator combo. LNS is also a frequently used method for the DARP.

For the ODBRP, the destructive operator clears one or more bus routes by removing all stops from it. Most of the time, the algorithm focuses on minimizing the total user ride time. However, in some situations it is beneficial to temporarily focus on trying to schedule as many requests as possible. Therefore, we use two different destructive and two different constructive operators. One focuses primarily on minimizing the total URT (further called the default or minURT set of operators) and another focuses on serving as much passengers as possible (the maxPassengers operators). We choose to work with the default set of operators until a solution is reached in which not all requests can be served. Then, we switch operators and attempt to serve as much requests as possible. When all requests can be served, the default set of operators is used again. Thus, for smaller instances or for instances with a relatively large fleet size, only the default set of operators is used. For larger instances or instances with a small fleet size compared to the number of requests, the results improve significantly by changing operators when necessary.

The bus routes to clear for the default destructive operator are selected according to a criterion, randomly chosen among one of the following three criteria: largest total user ride time, smallest number of passengers, and largest URT to distance ratio. The second covers the bus route with the smallest number of passengers served. The third criterion is determined by calculating the ratio of user ride time and direct distance in minutes from departure to arrival station for each passenger and summing the results per route. The number of bus routes to clear is decided as a uniformly distributed random number between 1% and 5% of the total fleet size. Because of the computation time needed to rebuild bus routes and the fact that we want to keep the most efficient bus routes in the solution intact, the number of bus routes to clear is deliberately kept low for the default operators.

When a criterion is chosen more than once, the routes that are next on the list according to this criterion are also cleared. For example, if the largest total URT criterion is chosen twice, then the bus route with the

largest and the second largest total URT will be selected. The requests that were scheduled in the cleared bus routes are added to the list of unserved passenger requests.

In the next step, first the emptied bus routes are filled with the origin and destination stations of the first requests of the list of unserved requests. Finally we try to add the other unserved requests to the current solution with the constructive heuristic explained in Section 4.1.

As was mentioned before, the change of operators happens when the algorithm has trouble serving every request, thus when the algorithm is working with an unfeasible solution. We conduct the change immediately if the initial solution does not serve every request or after 5 LNS-iterations of not serving everyone. In this case, it is desirable to have a feasible solution as quickly as possible. Therefore the number of bus routes to clear is set to a higher number determined as a uniformly distributed random number between 1% and 30% of the total fleet size. The bus routes to clear are again chosen according to a criterion, randomly chosen among three criteria: smallest number of passengers, random, maximum amount of waiting time. The constructive heuristic accompanying these destructive operators is very similar to the one explained in Section 4.1, it only uses a different insertion criterion.

A second toy example is shown in Fig. 14. The distance matrix shown in Table 4 still stands. Three requests are already planned in the bus route. At the first stop passenger 3 and 4 are getting on the bus, at the second stop passenger 5. Passenger 3 gets off the bus at the second stop and the others get off at the third stop. Also the arrival and departure time of the bus at the stops are indicated. To not complicate the example stop time is not included. At this point passenger 6 needs to be inserted in the schedule. In Fig. 15 the best insertion when minimizing the total URT is demonstrated. The origin and destination stop of passenger 6 are added to the back of the route. This way, the URT of the already existing passengers in the route is not changed. The total URT only increases with the URT of passenger 6, which amounts 15. The insertion criterion for minimizing the total URT thus equals 15. When we would be maximizing the amount of passengers served, the insertion criterion is equal to the absolute value of the route length increase. This criterion is also minimized, because the smaller the route length increase, the more time is left to serve other passengers. In this case the extension aggregates to 43, 22 for adding the origin stop and 15 for adding the arrival stop. In Fig. 16 a different insertion is demonstrated. In this case the insertion criterion for maximizing the number of passengers served is 7 for the origin stop (the arrival time of stop 2 now minus the arrival time of stop 2 in the initial solution) and 9 (same for stop 3) for adding the arrival stop of passenger 6, together this equals 16. As 16 is less than 43, this insertion is better when maximizing the number of passengers served compared to the one in Fig. 15. When minimizing the total URT, the insertion criterion for the origin would become 7 times the number of passengers affected, thus 7 times 2, because both passenger 3 and 4 will spend more time on the bus as a result of this insertion. For the arrival it would be 9 times 2, because both passengers 4 and 5 are affected now. Also, the URT of passenger 6 has to be added to the sum. In this case his URT would be 19 ($= 34 - 15$). The insertion criterion sums up to 51 ($= 7 \times 2 + 9 \times 2 + 19$). This is larger than the 15 from Fig. 15. The toy example demonstrates the difference in final insertion choices between the to operators.

Bus stop	1	2	3
Bus station	7	3	6
Arrival time	10	22	32
Departure time	10	22	32
P on	3,4	5	
P off		3	4,5

Figure 14: Toy example - Initial solution

Bus stop	1	2	3	*	*
Bus station	7	3	6	8	2
Arrival time	10	22	32	60	75
Departure time	10	22	32	60	75
P on	3,4	5		6	
P off		3	4,5		6

Figure 15: Toy example - Insertion under minimizing total URT

Bus stop	1	*	2	*	3
Bus station	7	8	3	2	6
Arrival time	10	15	29	34	48
Departure time	10	15	29	34	48
P on	3,4	6	5		
P off			3	6	4,5

Figure 16: Toy example - Insertion under maximizing number of passengers served

4.2.1 Local Search (LS)

Additionally to the LNS framework, Local Search (LS) is included in every iteration. An overview of the local search part of the algorithm can be found in Algorithm 2. LS makes small changes to the solution to improve the solution quality. In this case, we use a remove-insert operator on the entire neighborhood of the solution. The LS is only performed if all requests are served in the iteration. Every request is considered in turn. If the request is not scheduled alone in the route, it is removed. When a request is the only one scheduled in a bus, his URT is minimal as the trip is actually a direct taxi ride. In this case, a better insertion will not be found and the request is kept in place. If a request is removed out of bus route $b1$, the difference in URT before and after removal is calculated. Afterwards, the algorithm searches for a better insertion position for request i in terms of minimizing total URT in the same or a different bus route and inserts the request in the bus route found to be the best ($b2$). Then, the difference in URT is calculated before and after adding the request to $b2$. If the URT decreases by the removal and reinsertion, the new insertion is kept. Otherwise, the algorithm goes back to the saved original solution.

We choose to remove-insert *every* request in every iteration, because the aim of our heuristic is not to perform in time-efficiency, but rather to present a straightforward, valuable heuristic to use in our experiments to compare the user ride times in an on-demand system with those of a traditional public transport network. It is highly likely that more efficient strategies exist, but these are left for future research. Also, as the static ODBRP is investigated, computation time is not the main issue. Contrary to the dynamic ODBRP, no dynamic requests have to be inserted last minute and no real-time optimization is required.

Nevertheless, this LS component has a major influence on the solution quality, and is especially important when the algorithm has difficulties scheduling every request. This is further discussed in Section 4.4.2.

4.2.2 Overview Algorithm

To conclude this section Algorithm 3 represents an overview of the entire algorithm. First an initial solution is built with the constructive heuristic explained in Algorithm 1. If the number of passengers served (P) is not equal to the number of requests ($|R|$), the LNS first focuses on maximizing the number of passengers served until P equals $|R|$ or until a termination criterion is met (we use 1000 iterations). If, after building the initial

Algorithm 2: Local Search

```
for every request i scheduled in the solution do
    Save original solution ;
    b1 = the bus in which i is currently scheduled ;
    if size b > 2 then
        Remove i ;
        Delta_1 = URT_b1 with i − URT_b1 without i ;
        Re-add i with constructive heuristic (Section 4.1, Algorithm 1) ;
        b2 = the bus in which i is currently scheduled ;
        Delta_2 = URT_b2 without i − URT_b2 with i ;
        Final delta = Delta_1 − Delta_2 ;
        if Final delta < 0 then
            Go back to original solution ;
```

Algorithm 3: LNS for ODBRP

```
Function MaxP():
    while P < |R| and iterations < 1000 do
        Destroy and repair maxP;
        iterations++;
        if P = |R| then
            Local search 20 iterations;
            iterations += 20;

Function Main():
    Build initial solution with constructive heuristic (Section 4.1, Algorithm 1);
    iterations = 0;
    if P ≠ |R| then
        maxP();
    while iterations < 1000 do
        Destroy and repair minURT;
        iterations++;
        if P = |R| then
            Local search;
        else
            if count = 5 then
                count = 0;
                maxP();
            else
                count++;
```

solution, all requests are served, the actual algorithm starts with a while-loop until the termination criterion is met. First, bus routes are destroyed and repaired with the default operators explained in Section 4.2. If all passengers are served, the LS explained in Section 4.2.1 is executed. The focus shifts to maximizing the number of passengers served if not everyone was served during 5 iterations. This is done until all requests are again scheduled in the solution. As is shown in the function "MaxP", the destroy and repair operators maximizing the number of passengers served are called without using LS. However every destroy and repair combination also counts as an iteration, because LS is deliberately not included. The focus is really on getting P to equal $|R|$. LS is done when P equals $|R|$ and this immediately for 20 iterations. This is to quickly counter the rise in total URT caused by using the destroy and repair operators to maximize the number of passengers served. By doing only LS it is certain the number of passengers served stays the same. When a passenger is removed, it is certain he can be inserted again, if necessary in the same insertion position as before. After these 20 iterations the algorithm jumps back to line 15, focusing on minimizing the total URT.

As was mentioned before, the MaxP-function of Algorithm 3 (line 1-7) is not used if the fleet size is relatively large compared to the number of requests, as this causes no problems to serve everyone. Notwithstanding, this part is extremely useful otherwise.

4.3 Computational speedup

When naively implemented the feasibility checks in Algorithm 1 have a computational complexity of $O(n^2)$. In the worst case, when we would have one available bus and we would have planned all the requests in this bus except for one, we have $2n-2$ insertion positions in a route, for each of the $n-1$ requests a different origin and arrival stop. When we would want to insert an origin stop for the last request at the beginning of the route, we need to check $2n-3$ positions further in the route for lateness caused by the detour. To reduce this complexity, a data structure is introduced in which the slack times of all drop-offs are pre-processed. This was initially introduced as the forward time slack by Savelsbergh [1992] for the Vehicle Routing Problem (VRP) with Time Windows, but an adapted version was also used by Cordeau and Laporte [2003] in the DARP. Because the ODBRP has different characteristics than the DARP, it is not possible to simply copy the same formula. One of the differences with Cordeau and Laporte [2003] is the use of 3 time stamps per vertex (or stop). They use the arrival time of a vehicle at a stop, the beginning of the service at the stop and the departure time of the vehicle at the stop. In the ODBRP only 2 time stamps are used as is explained in Fig. 7: the arrival time of the vehicle at the stop and the departure time of the vehicle at the stop. Although Fig. 7 is right in theory, the algorithm will slightly optimize this time frame automatically. This is shown in Fig. 17. The algorithm will make sure that the bus departs as early as possible to the next stop, this means that t^d_{nb} moves more to the front of the time line compared to the time line of Fig. 7, to the moment when the last scheduled passenger (passenger 2 in the figure) can get on the bus. All passengers that need to get off the bus, get off at time t^a_{nb} and all passengers that have to get on the bus, are scheduled at time t^d_{nb}. As service time is negligible, this is currently not included in the algorithm. Therefore, if e^u_2 and e^u_3 would lie before t^a_{nb} in the time line, no waiting time would occur and t^a_{nb} would be equal to t^d_{nb}. This is shown in Fig. 18. Here the pick up and drop off events happen at the same time.

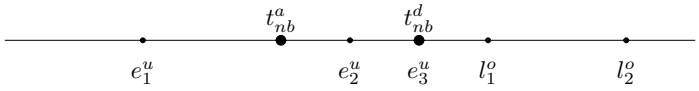


Figure 17: Time aspect at bus stop with waiting in the algorithm

Another difference with the DARP is the use of the maximum ride time, this is not necessary in the ODBRP because maximum ride times are incorporated in the time window of each passenger.

The slack time of a certain passenger is calculated by subtracting his latest arrival time by the actual arrival time of the bus at her destination bus stop. When constructing a solution, every available bus has a slack vector of the same size as its route composed until now. This means that the slack vector has the

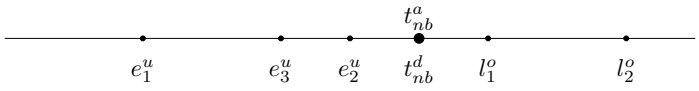


Figure 18: Time aspect at bus stop without waiting in the algorithm

same number of filled positions or stops as the bus route constructed. In every position of the vector the minimum slack time for this position and every position further in the route is represented. The minimum slack time is given because this is the most restrictive when wanting to insert other requests in the route. Assuming that the route is feasible, i.e., all passengers arrive at their destination on time, this slack time is always positive or equal to zero. Because the slack vector indicates the minimum slack time for the current position and every position further in the route, the slack vector can only become larger or stay equal when iterating over the positions from left to right. For example, at the last position, only the passengers getting off at this stop have to be taken into account, while at the first position, the minimum slack time of all the passengers scheduled in the route is shown. Every time a passenger request is inserted in the solution, the slack vector is updated. More formally the forward slack time for the ODBRP at a certain position i of bus b is presented in equation 1.

$$F_{ib} = min_{i \leq j \leq n}(\sum_{p}(l^o_p - t^a_{jb})y^o_{pjb}) \qquad b \in B, \{i,j,n\} \in N, p \in R \tag{1}$$

In the route that we obtained in the toy example in Fig. 11 the slack time of passenger 1 is $100 - 52 = 48$ and the slack time of passenger 2 is $130 - 90 = 40$. As a consequence the slack vector would have four positions (one for every bus stop) and would have a value of 40 in every position as the second passenger gets off the bus last and has the smallest slack time. A follow up of the toy example from Fig. 11 with slack vector is illustrated in Fig. 19. A new request needs to be inserted and we want to check if it would be feasible to insert station 4 after the second stop of the bus route. The delay caused by this insertion amounts $-10 + 20 + 35 = 45$ (-10 to delete the connection between station 3 and 6, 20 for the distance between station 3 and 4, and 35 for the distance between station 4 and 6). Using the slack vector, it can be immediately discovered that this insertion would be unfeasible, without checking the latest arrival times of every passenger drop-off later on in the bus route.

Bus stop	1	2	*	3	4
Bus station	7	3	4	6	11
Arrival time	30	42		52	90
Departure time	30	42		52	90
P on	2	1			
P off				1	2
Slack vector	40	40		40	40

Figure 19: Toy example - Slack vector

Without the slack vector, every time a feasibility check is performed, the algorithm loops over all the positions further in the route and over all the passengers getting off the bus at these positions. For every passenger it is checked if they would be too late because of the detour, by comparing their latest arrival time to the actual arrival time of the bus plus the detour. If the latest arrival time is earlier than the possible new arrival time of the bus, the passenger would be too late and the insertion would be infeasible. Since the feasibility check function is called many times during the heuristic and since the number of requests is relatively high in the more realistic instances, it is important that the feasibility of an insertion can be checked efficiently.

4.4 Algorithm analysis

In this section instances are used with 121 equally distributed stations in a 100 × 100 Euclidean plane. The stations form a grid on the plane (the same grid of stations is used in Section 5 in Fig. 27). If driving along an edge of the graph takes 100 minutes, the driving distance between each station and the next is 10 minutes. A maximum of 2000 requests are randomly generated, of which the earliest departure times are all in the first hour of the time horizon.

4.4.1 Algorithm quality assessment with LocalSolver

To asses the quality of our algorithm, we compare our results with the results of the heuristic software LocalSolver [LocalSolver, 2020a]. LocalSolver is a black-box, global optimization solver that uses both exact and heuristic methods. Since pure exact methods are incapable of solving instances of realistic scale, this is a more meaningful alternative. Our implementation is based on the pick-up and delivery problem with time windows, with the necessary adaptations to include bus station assignment and was checked and approved by the developers at LocalSolver [LocalSolver, 2020b] [N. Stott, priv. comm.]. Figure 20 shows the comparison in computation time and solution quality for an instance with 500 requests. If every request would be served by a different bus, thus if every passenger would have a direct taxi ride from origin to destination, the total URT of the instance is 22802. This small total URT can also be reached with a fleet size of 182 buses with a capacity of 8 people. With this fleet size, LocalSolver finds a first feasible solution after 77 seconds with a total URT of 33442. The solution quality keeps improving until a total URT of 22802 is found after 9977 seconds. This is the optimal solution, because obviously, the total user ride time cannot be smaller than in the situation where all passengers are transported without delay from their origin to their destination. Our algorithm on the other hand finds an initial solution with a total URT of 30827 in 0.10 seconds, which quickly improves to 23119 within one second. After 46 seconds our algorithm also finds the optimal solution.

To further investigate the algorithm's quality, we solved several larger instances in LocalSolver with a tighter fleet size. An instance of 1000 randomly distributed requests was solved with a fleet size of 221 buses with capacity 8. The time limit for LocalSolver was set at 24 hours. The best URT found, was 48757. Our heuristic algorithm solved the instance three separate times with a limit of 1000 iterations. On average, after 386 iterations the last improvement in URT was seen, requiring a computation time of approximately 120 seconds. The gap in solution quality was 0.3% on average in favor of LocalSolver. The improvement in total URT by our algorithm is presented in Fig. 21. Within the first second the solution improves by 19%, after five seconds the improvement is 22% and after 2 minutes 22.9%. This indicates that once again our algorithm finds a qualitative solution in a reasonable amount of time. The same experiment was performed for an instance with 2000 requests and a fleet size of 259 buses. In this case LocalSolver did not succeed in finding a feasible solution after a week.

4.4.2 Impact of the LNS operators change and LS

In this section we first discuss the benefit of the change in LNS-operators. Then, we shortly examine the influence of the penalty factor with an example. Lastly we show the effect of adding the local search components to the algorithm.

As was explained before, this LNS-operator change is located from line 1 to 7 in Algorithm 3 and is carried out when the algorithm has trouble finding a solution that serves every request. As an example an instance with 2000 requests and a fleet size of 400 buses with capacity 8 is used. The results are shown in Fig. 22. The dotted black line represents the improvement of the solution with our heuristic. If the LNS-operator change is not included, the improvement stops after 28 seconds. The heuristic reaches a point were 1999 of the 2000 passengers are served and the algorithm is not able to find a feasible solution that includes this one passenger. By adding the LNS-operator change a feasible solution is found after only 1 iteration and after a few seconds the algorithm finds feasible solutions with a better objective function value, as is shown in the graph. In this case, the difference in total URT is only 1%. However if the fleet size is smaller this difference grows, when the same instance is solved using a fleet size of 250 buses. The initial solution does not serve all requests (it serves only 1784 out of the 2000 requests) and a LNS-operator change

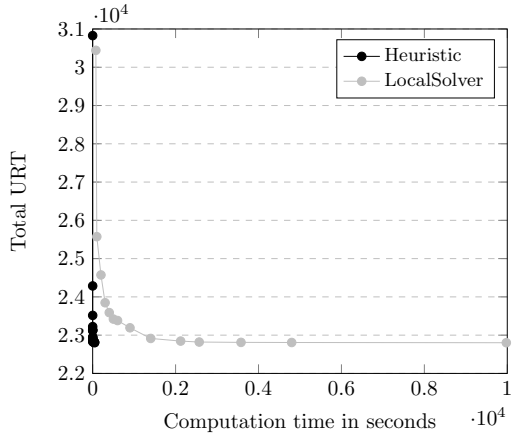


Figure 20: Comparison LocalSolver (121 stations, 500 requests, 182 buses with capacity 8)

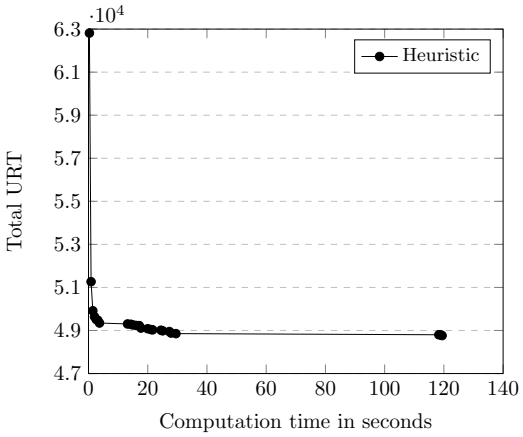


Figure 21: Improvement in total URT compared to initial solution (121 stations, 1000 requests, 221 buses with capacity 8)

is necessary to obtain a feasible solution. Figure 23 plots the number of passengers served as a function of time with and without using the operator change. Without it, no feasible solution is found, while with the change a first feasible solution is found after 1.6 seconds. The solution becomes in-feasible twice during a computation time of two minutes, but is quickly restored each time. After these restorations the total URT is further improved.

Figure 24 shows the contribution of the penalty factor explained in Section 4.1. By using the penalty factor the algorithm finds a first feasible solution in 1.6 seconds, without it, this time almost doubles to 3 seconds.

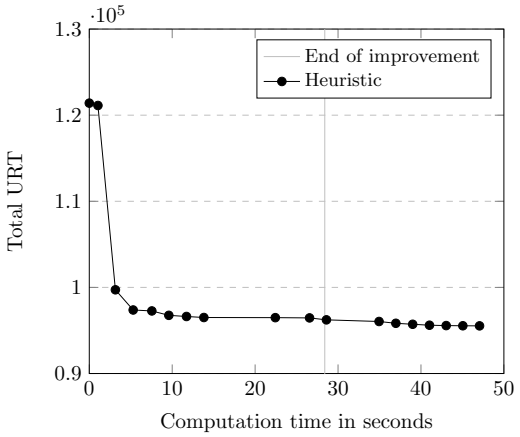


Figure 22: Impact of the change in LNS operators (121 stations, 2000 requests, 400 buses, capacity 8)

Also the influence of the local search component is investigated by running the algorithm with and without the LS-component (i.e. lines 6 and 18 in Algorithm 3). Of course the LS is very time-consuming because

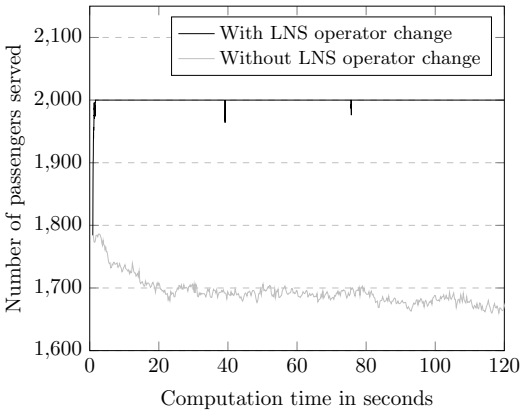


Figure 23: Impact of the change in LNS operators (121 stations, 2000 requests, 250 buses with capacity 8)

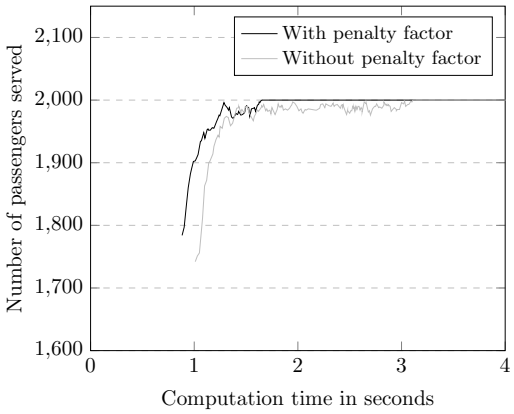


Figure 24: Impact of the penalty factor (121 stations, 2000 requests, 250 buses with capacity 8)

every request is removed en reinserted into the solution. To make the comparison fair, the algorithm is run for instances with a different number of requests with LS for 1000 iterations and the average computation time needed for this configuration is used as a termination criterion to run the algorithm without LS. The fleet size is once again set to 250 and the capacity to 8. The differences in URT when using LS or not are relatively small for instances with respectively 500 and 1000 requests, around 8%. This is shown in Fig. 25. However when the number of requests increases compared to the fleet size, a larger difference can be seen, around 20% difference in total URT for instances with 1500 requests. Furthermore it should be noted that the last improvement in URT is found a lot faster when using LS in the framework. This indicates that adding LS results in significantly lower total URTs and is especially important when the fleet size is small compared to the number of requests.

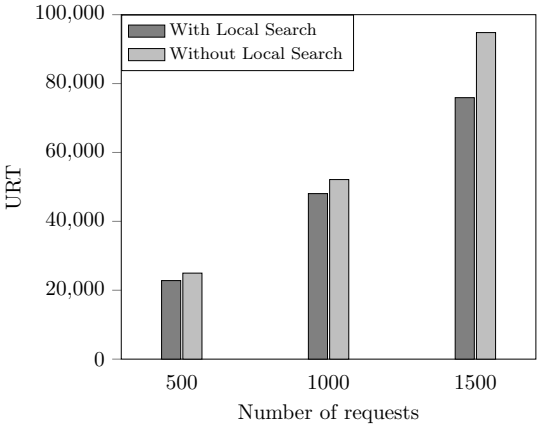


Figure 25: Effect of Local Search (121 stations, 250 buses with capacity 8)

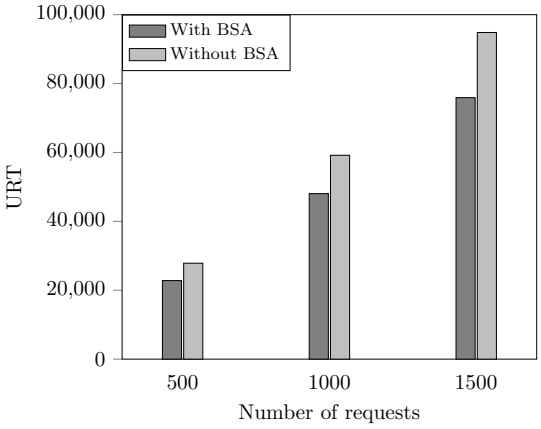


Figure 26: Effect of BSA (121 stations, 250 buses with capacity 8)

4.4.3 Impact of bus station assignment

Lastly we briefly compare the effect of bus station assignment (BSA) on the total URT. When BSA is excluded all passengers are allocated to their closest stop both for arrival and departure. Hence passengers no longer have a set of potential departure and arrival stations, but the algorithm has to work with one station for the arrival and one for the departure. In an experiment, our algorithm was run both with and without BSA for 2 minutes. The results of these experiments are shown in Fig. 26. It is clear that the lack of flexibility significantly increases the objective function value of the solutions produced by the algorithm. The larger the number of requests, the larger the difference in URT between the solution with and without BSA. For instances with 500 requests the difference is around 18% on average, while for instances with 1500 requests, it increases to 24%. Instances with more than 1500 requests are not incorporated in the graph because it became impossible to schedule all the requests when BSA was excluded.

5 Comparison to traditional public bus transport (TPBT)

In this section we compare the on-demand bus system (as modeled by the ODBRP) with a traditional bus system with fixed lines and fixed timetables. To this end we model the traditional urban public bus transport network as a grid with eleven bus lines going north to south, and eleven bus lines going west to east, both back and forth (see Fig. 27). A bus station is located at every intersection of two lines, which results in a total of 121 bus stations. The standard frequency is set at one bus every 30 minutes and two adjacent stops are located at a distance of 10 minutes from each other. The capacity of the buses is set to 8 in order to make a fair comparison with the on-demand buses used in the ODBRP explained in Section 3.

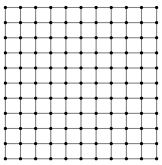


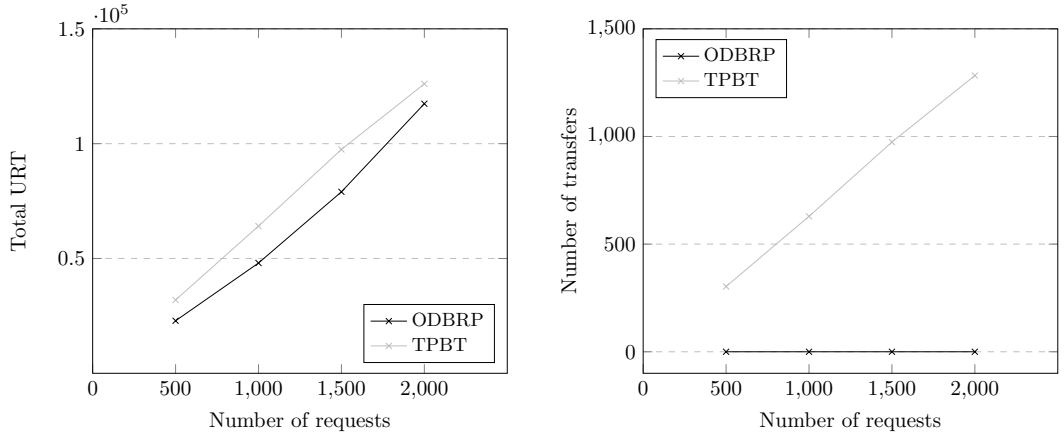
Figure 27: Traditional public transport grid

To compute the total URT in the traditional public bus transport (TPBT) network we use the same instances as for the ODBRP. Every request consists of a passenger, with a number of possible departure and arrival stations and a time window. The maximum walking distance between a passenger's origin (arrival) location and origin (arrival) station is set to 10. In this case the maximum number of possible departure or arrival stations is set to 4: the corner points of the square in the grid in which the passenger's arrival or destination are generated. Because buses run either horizontally or vertically, passengers whose departure and arrival cell are not in the same or adjacent rows or columns, will need to transfer. To simplify the solution of the traditional bus system, passengers are only allowed to make one transfer. If a passenger, for example, wishes to travel from the upper right corner to the lower left corner of the grid, there are two travel options: first north to south, then east to west, or the other way around. Given the fact that every passenger can choose utmost 4 stops, both for departure and arrival, this means that every passenger using the traditional network can have a maximum of 32 ($4 \times 4 \times 2$) ways to travel from his origin to his destination. For every request, the travel option with the shortest total URT is chosen. However, capacity constraints are taken into account. In this experiment it is assumed that passengers know whether or not the bus they want to take is full, meaning that their routes are partly optimized. If a passenger would like to take a certain bus, but this bus has already reached full capacity, he can wait at home until the next bus is coming over this bus line or he can choose a different travel option. Waiting time at home is not included in the total URT, as it is also not included in the URT calculated in the ODBRP. However, waiting time at a transfer station is included in the URT. Both in the ODBRP and in the TPBT, passengers need to be transported within their given time window. Partly optimizing the routes of the TPBT and not including

the waiting time at the first stop is not entirely realistic. However we want to give some advantage to the TPBT to be sure that the ODBRP actually performs better and that the improvement is not caused by the way of comparing the two systems.

After calculating the best total URT of the instance with the traditional public bus transport network, we know which lines are used, and the number of buses needed to serve those lines is calculated. Note that only the buses needed to serve the lines that were actually used are taken into account. For lines that are not used, no buses are allocated. We assume equal costs for both systems. Especially with the rise of autonomous electrical vehicles as was mentioned in Section 2, the cost per kilometer declines as the driver's cost vanishes, making this a reasonable assumption. Subsequently, the ODBRP-heuristic explained in Section 4 is used to approach the total URT in the on-demand system. The same requests and bus stations are used, and the fleet size is set to the number of buses actually used in the traditional system.

The LNS-heuristic is run for 1000 iterations for every instance. The instances differ in number of requests (500 to 2000) and are randomly generated. The earliest departure times of the requests are all in the first hour of the time horizon. The latest arrival times are set accordingly to the distance that needs to be traveled. In this case we have set the latest arrival time equal to the earliest departure time plus two times the (Euclidean) distance that needs to be traveled (in minutes) plus twenty minutes. The fixed term of twenty minutes is chosen to make sure that requests with short distances still have a reasonable time window.



(a) The total URT

(b) Number of transfers

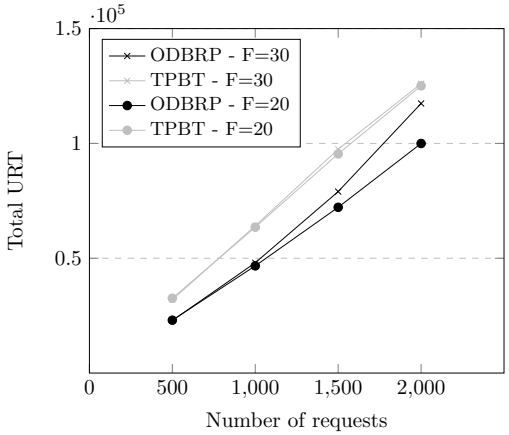
Figure 28: Comparison ODBRP and TPBT for different numbers of requests

Figure 28a shows the difference in total URT between the TPBT and the ODBRP. For every instance size (expressed in number of requests) 15 separate instances are solved and the average URT is plotted. For instances with 500 requests, the on-demand system performs 28.4% better than the TPBT, while for 2000 requests this percentage declines to 6.8%. The latter can be explained by the increase in URT for the ODBRP when the number of requests is relatively high compared to the fleet size. Logically, an on-demand system will never replace e.g., a high-capacity metro-line.

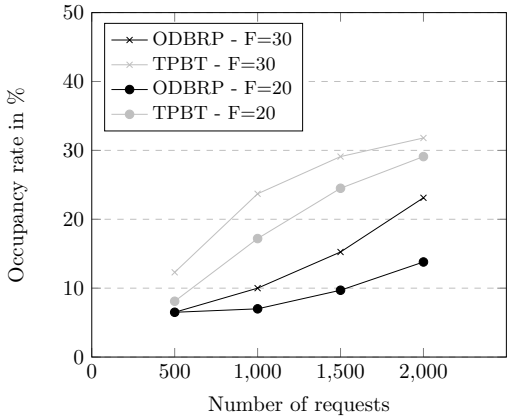
In Fig. 28b the number of transfers in the on-demand system and in the traditional system are shown. In the ODBRP, transfers are not allowed, every request is served by one bus, while in the TPBT the number of transfers is in our instances approximately 60% of the total number of requests. In the simulation we took into account a 5 minute buffer to be able to make the transfer. However we did not include a penalty cost for making a transfer. Frei et al. [2017] found in their work that the cost of a transfer amounts to 4.9 equivalent in vehicle minutes (EIVM), while Garcia-Martinez et al. [2018] found a number of 15.2 EIVM for making one transfer. These figures suggest that a transfer has a (perceived) cost in itself, regardless of the delay it causes, and indicates that passengers would prefer a longer direct bus trip above a shorter journey

with a transfer.

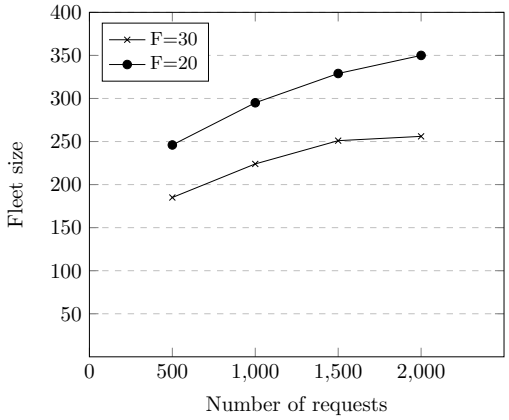
When the number of requests is 2000, in the TPBT, 30% of the passengers face the fact that the bus they initially want to take is full. To reduce the number of passengers that have to wait due to capacity constraints, either the frequency or the capacity of the buses needs to increase.



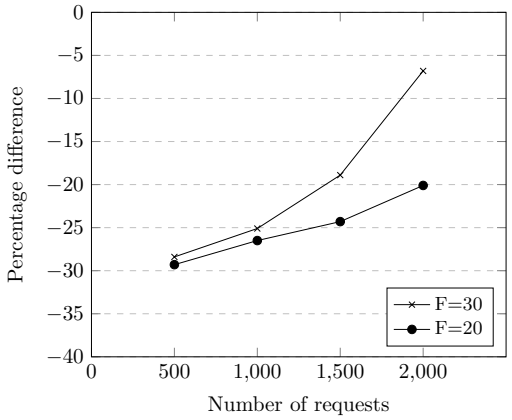
(a) Effect F on total URT



(b) Effect F on occupancy rate



(c) Effect F on fleet size



(d) Percentage difference between total URT obtained with TPBT and total URT obtained with ODBRP

Figure 29: The effect of a frequency (F) change

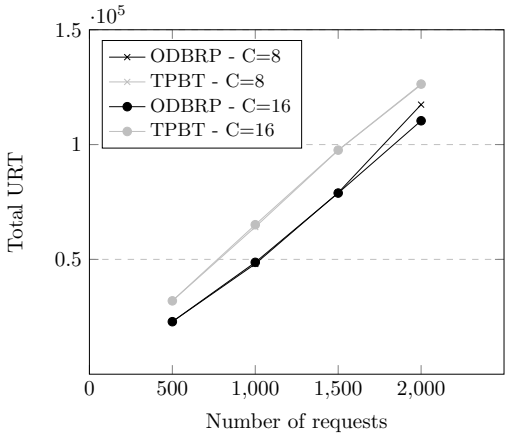
In Fig. 29a the effect of an increased bus *frequency* on the total URT is shown. The frequency in the TPBT is raised from one bus every 30 minutes to one bus every 20 minutes over all the lines. The first effect visible is the decreased total URT for the on-demand system, especially for the larger instances. This can be explained by the fact that the fleet size is larger when a frequency of 20 minutes needs to be maintained. For an honest comparison the fleet size used in the on-demand system is equal to the exact amount of buses used in the TPBT system. The larger the fleet size, the larger the flexibility for the routing of the on-demand buses, which is especially important when the number of requests is large.

The effect of the frequency on the number of buses used in the TPBT is shown in Fig. 29c. Because of

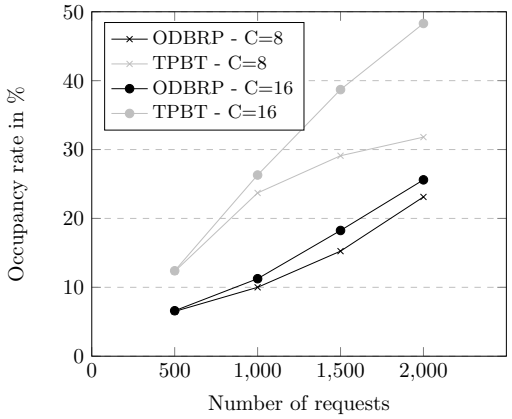
the increased fleet size we can see a lower occupancy rate per bus for both the on-demand as the traditional system (see Fig. 29b). The occupancy rate for the TPBT is generally larger because of two reasons. Firstly, passengers have to transfer, thus using multiple lines. And secondly, a single bus can serve multiple lines.

Figure 29d shows the percentage difference between the URT's of the TPBT and the on-demand system. For instances with 500 requests, the on-demand system performs 29.3% better than the traditional one, which is a slightly bigger difference than previously when the frequency was lower. In contrast, for instances with 2000 requests the difference decreases from -6.8% to -20.1%. The number of transfers stays nearly independent of the frequency of the buses.

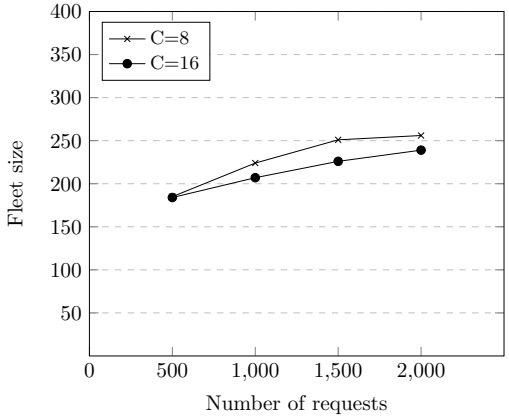
In Fig. 30a the influence of a larger *capacity* of the buses on the total URT is demonstrated in an experiment in which the bus capacity is doubled from 8 people per bus to 16. With these experimental parameters the difference in URT between the on-demand and the traditional system still stands. However, both systems barely encounter any changes in URT, only for instances with 2000 requests the URT of the ODBRP slightly decreases. Nevertheless, a paired t-test over the instances concludes that the URTs in an on-demand system with bus capacity 16 are significantly (albeit only a little bit) lower than the ones with capacity 8. There is also a small decrease in the fleet size, as can be seen in Fig. 30c. The difference between the total URTs in the on-demand and traditional system when using a capacity of 16 is demonstrated in Fig. 30d. There are almost no people that have to wait due to capacity constraints in the TPBT when the capacity is 16. In Fig. 30b it is no surprise that the occupancy rate lowers as the capacity was doubled.



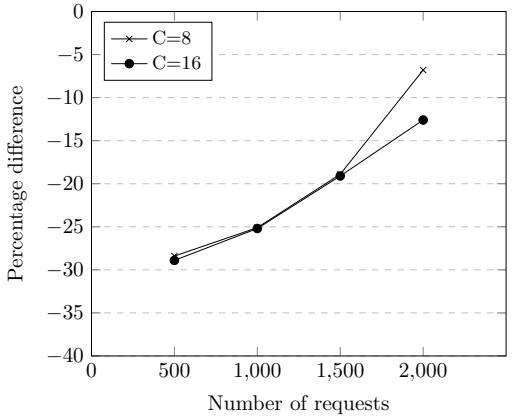
(a) Effect C on total URT



(b) Effect C on occupancy rate



(c) Effect C on fleet size



(d) Percentage difference between total URT obtained with TPBT and total URT obtained with ODBRP

Figure 30: The effect of a change in capacity (C) of the vehicles

6 Conclusions

This paper introduced the on-demand bus routing problem (ODBRP) to support the routing of on-demand buses in an urban environment. An on-demand bus system is a logical next step in the evolution of public transport, as flexible urban transport (FUT) becomes more popular and important. A straightforward LNS heuristic is proposed to solve the ODBRP, minimizing the total user ride time (URT). Bus station assignment increases the flexibility of the routing, adding a new layer of complexity to the optimization problem. The LNS heuristic is used to compare the ODBRP with a traditional public bus system, where lines and timetables are fixed. In general total URT is always lower when using on-demand buses instead of the traditional system and transfers are completely avoided. An on-demand bus system is therefore shown to yield shorter passenger trips than a traditional bus system. Additionally our experiments offer some conclusions that might prove useful in the design of a fully on-demand public bus system: our results indicate that the performance of such a system increases when the fleet size is large and the capacity of the buses is rather small.

Future research is needed to expand the static ODBRP to the dynamic version of the problem where only part of the requests are known beforehand. In this case new requests will pop up during the planning horizon. Bus routes will change while some passengers are already scheduled on them. In the LNS heuristic, when clearing some of the routes, a passenger who is already on the bus at a certain time, can not be cleared off the schedule, which will lower the solution possibilities. The algorithm will need some adaptations with the aim of lowering the computation time. Secondly the demand for transportation in the instances can be made more realistic, by including for example busy peak hours, a city center and suburbs, etc. Finally, this study has presented a rather straightforward heuristic for the ODBRP, which leaves ample opportunities for studies into more complicated (and therefore potentially more effective) ways to tackle this problem.

Acknowledgements

This research was supported by a grant of the Flemish Fund for Scientific Research (FWO Strategic Basic Research Fellowship grant).

References

R Adawiyah, Y Satria, and H Burhan. Application of large neighborhood search method in solving a dynamic dial-a-ride problem with money as an incentive. In *Journal of Physics: Conference Series*, volume 1218, page 012008. IOP Publishing, 2019.

M. Alonso-González, T. Liu, O. Cats, N. Van Oort, and S. Hoogendoorn. The potential of demand-responsive transport as a complement to public transport: An assessment framework and an empirical evaluation. *Transportation Research Record*, 2672(8):879–889, 2018.

C. Archetti, M. Speranza, and D. Weyland. On-demand public transportation. *Int. Trans. Oper. Res*, 2018.

S. Bunte and N. Kliewer. An overview on vehicle scheduling models. *Public Transport*, 1(4):299–317, 2009.

S. Carosi, A. Frangioni, L. Galli, L. Girardi, and G. Vallese. A matheuristic for integrated timetabling and vehicle scheduling. *Transportation Research Part B: Methodological*, 127:99–124, 2019.

A. Ceder and N. Wilson. Bus network design. *Transportation Research Part B: Methodological*, 20(4): 331–344, 1986.

J. Cordeau and G. Laporte. The dial-a-ride problem: models and algorithms. *Annals of operations research*, 153(1):29–46, 2007.

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

W. Ellegood, S. Solomon, J. North, and J. Campbell. School bus routing problem: Contemporary trends and research directions. *Omega*, 2019.

M. Enoch, S. Potter, G. Parkhurst, and M. Smith. Why do demand responsive transport systems fail? Working paper, 2006.

B. Finn. Towards large-scale flexible transport services: A practical perspective from the domain of para-transit. *Research in transportation business & management*, 3:39–49, 2012.

C. Frei, M. Hyland, and H. Mahmassani. Flexing service schedules: Assessing the potential for demand-adaptive hybrid transit via a stated preference approach. *Transportation Research Part C: Emerging Technologies*, 76:71–89, 2017.

A. Garcia-Martinez, R. Cascajo, S. Jara-Diaz, S. Chowdhury, and A. Monzon. Transfer penalties in multimodal public transport networks. *Transportation Research Part A: Policy and Practice*, 114:52–66, 2018.

Timo Gschwind and Michael Drexl. Adaptive large neighborhood search with a constant-time feasibility test for the dial-a-ride problem. *Transportation Science*, 53(2):480–491, 2019.

O. Ibarra-Rojas, F. Delgado, R. Giesen, and J. Muñoz. Planning, operation, and control of bus transport systems: A literature review. *Transportation Research Part B: Methodological*, 77:38–75, 2015.

International Association of Public Transport (UITP). Statistics brief - Local public transport trends in the European Union. `http://www.uitp.org/sites/default/files/cck-focus-papers-files/UITP_Statistics_PT_in_EU_DEF_0.pdf`, 2018. Online; accessed 18 September 2018.

J. Jokinen, T. Sihvola, E. Hyytiä, and R. Sulonen. Why urban mass demand responsive transport? In *Integrated and Sustainable Transportation System (FISTS), 2011 IEEE Forum on*, pages 317–322. IEEE, 2011.

LocalSolver. Optimization Solver and Services. `https://www.localsolver.com/`, 2020a. Online; accessed 15 April 2020.

LocalSolver. Pick up and delivery with time windows. `https://www.localsolver.com/docs/last/exampletour/pdptw.html`, 2020b. Online; accessed 12 May 2020.

M. Lübbecke, C. Puchert, P. Schiewe, and A. Schöbel. Integrating line planning, timetabling, and vehicle scheduling: integer programming formulation and analysis. Working paper, 2019.

J. Mageean and J. Nelson. The evaluation of demand responsive transport services in europe. *Journal of Transport Geography*, 11(4):255–270, 2003.

Renaud Masson, Fabien Lehuédé, and Olivier Péton. The dial-a-ride problem with transfers. *Computers & Operations Research*, 41:12–23, 2014.

M. Michaelis and A. Schöbel. Integrating line planning, timetabling, and vehicle scheduling: a customer-oriented heuristic. *Public Transport*, 1(3):211, 2009.

Y. Molenbruch, K. Braekers, and A. Caris. Typology and literature review for dial-a-ride problems. *Annals of Operations Research*, 259(1-2):295–325, 2017.

C. Mulley and J. Nelson. Flexible transport services: A new market opportunity for public transport. *Research in Transportation Economics*, 25(1):39–45, 2009.

J. Nelson, S. Wright, B. Masson, G. Ambrosino, and A. Naniopoulos. Recent developments in flexible transport services. *Research in Transportation Economics*, 29(1):243–248, 2010.

Sophie N Parragh and Verena Schmid. Hybrid column generation and large neighborhood search for the dial-a-ride problem. *Computers & Operations Research*, 40(1):490–497, 2013.

M. W.P. Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA journal on computing*, 4(2):146–154, 1992.

P. Schittekat, J. Kinable, K. Sörensen, M. Sevaux, F. Spieksma, and J. Springael. A metaheuristic for the school bus routing problem with bus stop selection. *European Journal of Operational Research*, 229(2): 518–528, 2013.

A. Schöbel. Line planning in public transportation: models and methods. *OR spectrum*, 34(3):491–510, 2012.

A. Schöbel. An eigenmodel for iterative line planning, timetabling and vehicle scheduling in public transportation. *Transportation Research Part C: Emerging Technologies*, 74:348–365, 2017.

Oscar Tellez, Samuel Vercraene, Fabien Lehuédé, Olivier Péton, and Thibaud Monteiro. The fleet size and mix dial-a-ride problem with reconfigurable vehicle capacity. *Transportation Research Part C: Emerging Technologies*, 91:99–123, 2018.

Uber. How it works. `https://www.uber.com/be/nl/ride/how-it-works/`, 2019. Online; accessed 12 February 2019.

Vlaams Parlement. De Lijn: Actualisatie van de internationale benchmarkstudie openbaar vervoer. Eindrapport. `http://docs.vlaamsparlement.be/pfile?id=1074230`, 2014. Online; accessed 19 November 2018.

Y. Westerlund, A. Stahl, J. Nelson, and J. Mageean. Transport telematics for elderly users: Successful use of automated booking and call-back for demand responsive transport services in gothenburg. In *PROCEEDINGS OF THE 7TH WORLD CONGRESS ON INTELLIGENT SYSTEMS*, 2000.

K. Winter, O. Cats, G. Correia, and B. van Arem. Performance analysis and fleet requirements of automated demand-responsive transport systems as an urban public transport service. *International Journal of Transportation Science and Technology*, 7(2):151–167, 2018.

A. Wren and J.M. Rousseau. Bus driver scheduling—an overview. In *Computer-aided transit scheduling*, pages 173–187. Springer, 1995.

Appendix A: Mathematical model on-demand bus routing problem (ODBRP)

$$\min \text{URT} = \sum\nolimits_p T_p \tag{2}$$

s.t.

$$\sum\nolimits_s x_{snb} \leq 1 \qquad \forall n \in N, b \in B \tag{3}$$

$$\sum\nolimits_s x_{snb} = 0 \implies \sum\nolimits_s x_{s(n+1)b} = 0 \qquad \forall n \in N, b \in B \tag{4}$$

$$\sum\nolimits_s x_{snb} = 0 \implies \sum\nolimits_p y^u_{pnb} = 0 \qquad \forall n \in N, b \in B \tag{5}$$

$$\sum\nolimits_s x_{snb} = 0 \implies \sum\nolimits_p y^o_{pnb} = 0 \qquad \forall n \in N, b \in B \tag{6}$$

$$\sum\nolimits_s x_{snb} = 1 \implies \sum\nolimits_p (y^u_{pnb} + y^o_{pnb}) \geq 1 \qquad \forall n \in N, b \in B \tag{7}$$

$$x_{snb} = 1 \wedge y^u_{pnb} = 1 \implies a^u_{ps} = 1 \qquad \forall s \in S, n \in N, p \in R, b \in B \qquad (8)$$

$$x_{snb} = 1 \wedge y^o_{pnb} = 1 \implies a^o_{ps} = 1 \qquad \forall s \in S, n \in N, p \in R, b \in B \qquad (9)$$

$$x_{snb} = 1 \wedge x_{s'(n+1)b} = 1 \implies t^a_{(n+1)b} \geq t^d_{nb} + \tau_{ss'b} \qquad \forall s, s' \in S \mid s \neq s', n \in N, b \in B \qquad (10)$$

$$y^u_{pnb} = 1 \implies t^d_{nb} \geq e^u_p \qquad \forall p \in R, n \in N, b \in B \qquad (11)$$

$$y^o_{pnb} = 1 \implies l^o_p \geq t^a_{nb} \qquad \forall p \in R, n \in N, b \in B \qquad (12)$$

$$\sum\nolimits_n (n y^u_{pnb} - n y^o_{pnb}) \leq 0 \qquad \forall p \in R, b \in B \qquad (13)$$

$$\sum\nolimits_n (y^u_{pnb} - y^o_{pnb}) = 0 \qquad \forall p \in R, b \in B \qquad (14)$$

$$\sum\nolimits_b \sum\nolimits_n y^u_{pnb} \leq 1 \qquad \forall p \in R \qquad (15)$$

$$x_{snb} + x_{s(n+1)b} \leq 1 \qquad \forall s, n, b \qquad (16)$$

$$\sum\nolimits_p (y^u_{pnb} - y^o_{pnb}) - q_{nb} = 0 \qquad \forall n \in N, b \in B \qquad (17)$$

$$\sum\nolimits_{n' \leq n} q_{nb} \leq C \qquad \forall n, n' \in N \mid n \geq n', b \in B \qquad (18)$$

$$y^u_{pnb} = 1 \wedge y^o_{pn'b} = 1 \implies T_p = t^a_{n'b} - t^d_{nb} \qquad \forall n, n' \in N \mid n' > n, p \in R, b \in B \qquad (19)$$

$$\sum\nolimits_p \sum\nolimits_b \sum\nolimits_n y^u_{pnb} = |R| \qquad (20)$$

$$x_{snb} \in \{0, 1\} \qquad \forall s \in S, n \in N, b \in B \qquad (21)$$

$$y^u_{pnb} \in \{0, 1\} \qquad \forall p \in R, n \in N, b \in B \qquad (22)$$

$$y^o_{pnb} \in \{0, 1\} \qquad \forall p \in R, n \in N, b \in B \qquad (23)$$

$$q_{nb} \in \mathbb{Z} \qquad \forall n \in N, b \in B \qquad (24)$$

In this formulation some of the constraints use *if-then* ($\implies$) and *and* ($\wedge$) logical operators, as this is easier to read and understand. However they can be easily linearized and written as follows:

$$\sum\nolimits_s (x_{snb} - x_{s(n+1)b}) \geq 0 \qquad \forall n \in N, b \in B \qquad (3')$$

$$2C \sum\nolimits_s x_{snb} - \sum\nolimits_p (y^u_{pnb} + y^o_{pnb}) \geq 0 \qquad \forall n \in N, b \in B \qquad (4\text{-}5\text{-}6')$$

$$\sum\nolimits_s x_{snb} - \sum\nolimits_p (y^u_{pnb} + y^o_{pnb}) \leq 0 \qquad \forall n \in N, b \in B \qquad (4\text{-}5\text{-}6'')$$

$$x_{snb} + y^u_{pnb} - a^u_{ps} \leq 1 \qquad \forall s \in S, n \in N, p \in R, b \in B \qquad (7')$$

$$x_{snb} + y^o_{pnb} - a^o_{ps} \leq 1 \qquad \forall s \in S, n \in N, p \in R, b \in B \qquad (8')$$

$$t^a_{(n+1)b} - t^d_{nb} - \tau_{s's} + (x_{snb} + x_{s'(n+1)b} - 2)(-M) \geq 0 \qquad \forall s, s' \in S \mid s \neq s', n \in N, b \in B \qquad (9')$$

$$t^d_{nb} - e^u_p + (y^u_{pnb} - 1)(-M) \geq 0 \qquad \forall p \in R, n \in N, b \in B \qquad (10')$$

$$t^a_{nb} - l^o_p + (y^o_{pnb} - 1)M \leq 0 \qquad \forall p \in R, n \in N, b \in B \qquad (11')$$

$$T_p + (2 - y^o_{pn'b} - y^u_{pnb})M - t^a_{n'b} + t^d_{nb} \geq 0 \qquad \forall n, n' \in N \mid n' > n, p \in R, b \in B \qquad (18')$$

$$T_p + (2 - y^o_{pn'b} - y^u_{pnb})(-M) - t^a_{n'b} + t^d_{nb} \leq 0 \qquad \forall n, n' \in N \mid n' > n, p \in R, b \in B \qquad (18'')$$

The objective function (2) minimizes the total user ride time (URT).

Constraints (3) enforce the fact that a bus can only stop at one station at the same time.

Constraints (4) make sure stops that the positions used in the bus route are used consecutively and start at the first position. Constraints (3') are the linearized form of constraints (4).

Constraints (5) and (6) impose that no passengers can get on or off the bus at position n, when the bus does not make a stop at position n, while constraints (7) ensure that when a stop is made, at least one passenger gets either on or off the bus. The linearized way of writing these three groups of constraints can be found in constraints (4-5-6') and (4-5-6").

Bus station assignment is modeled in constraints (8) and (9) which enforce that if a bus stops at a certain bus station to pick up (drop off) a passenger, the station has to be in the set of allowed stops for the pick-up or drop-off of this passenger. The linearized form can be found in constraints (7') and (8') respectively.

The time aspect of the ODBRP is incorporated in constraints (10), (11), and (12), or in (9'), (10') and (11') in the linearized form. If s is the n-th stop of the bus and s' is the $n+1$-th stop of the bus, then the time of arrival at s' should be greater than the departure time at s plus the travel distance between stops s and s'. Also, the departure time (arrival time) of the bus at the n-th stop is not smaller than the earliest departure time (latest arrival time) of a passenger assigned to that stop.

Constraints (13), (14), and (15) enforce that a passenger is picked up before he is dropped off and that both pickup and arrival are carried out only once by the same bus. For easy reading purposes, constraints (16) forbid that the same bus station is visited twice in a row.

Constraints (17) and (18) enforce the capacity of the buses. The net number of passengers picked up at the n-th stop of the bus is equal to the number of passengers picked up minus the number of passengers dropped off at that stop. This variable is used in the last constraint ensuring that the capacity of the buses is not exceeded. Constraints (19) (or both (18') and (18") ln the linearized form) define the variable used in the objective function. Constraints (18") is non-essential when minimizing the total URT, however is included for completeness and necessary if a different objective function would be used. Constraint (20) makes sure every request is served. Lastly, constraints (21), (22), and (23) define the binary decision variables, and constraints (24) enforce the quantity decision variable to be integer.