A large neighborhood search algorithm to optimize a demand-responsive feeder service

Bryan David Galarza Montenegro[a], Kenneth Sörensen[a], Pieter Vansteenwegen[b]

[a]*Department of Engineering Management (ENM) University of Antwerp, Prinsstraat 13, Antwerp, 2000, Antwerp, Belgium*
[b]*Mobility Research Centre - CIB KULeuven, Celestijnenlaan 300, Leuven, 3001, Vlaams-Brabant, Belgium*

Abstract

Feeder services are public transit services that transport people from a low demand, typically suburban, area to a high demand area, such as a transportation hub or a city. Here, passengers continue their journey using traditional forms of public transport. On the one hand, on-demand feeder services have been a topic of discussion in a number of recent studies, since these services can serve the demand efficiently. On the other hand, traditional feeder services with predetermined routes and timetables provide predictability and easier cost control. In this paper, a demand-responsive feeder service is considered, which combines positive characteristics of both traditional services as well as on-demand-only services. This feeder service has mandatory bus stops which are always serviced, as well as optional bus stops which are only serviced when there is demand for transportation nearby. To optimize the performance of this feeder service, a large neighborhood search heuristic is developed. Experimental results on 14 benchmark instances illustrate that the LNS algorithm obtains solutions with an average gap of 1% or less compared to the optimal solution, within 1s of runtime. Larger instances can also be solved, typically in less than 60s. The results also show that the demand-responsive feeder service generally outperforms a traditional service in terms of service quality, often by more than 60%.

Keywords: meta-heuristics, large neighborhood search, public bus transport, feeder service, demand-responsive transportation

1. Introduction

Public transport is an essential part of urban mobility in most European cities, more than 55 billion journeys on public transport were recorded in 2014 [1]. As such, public transport enables accessibility to social activities, goods and services and this can reduce social exclusion and poverty [2]. For secluded areas in particular, like residential areas and suburbs, a feeder service enables connectivity to major transit networks. In a feeder service, a fleet of buses transports passengers from typically sparsely populated areas to areas with a high demand for transportation or to transportation hubs, where the passengers can continue their journey. These services can be an answer to the problem of overfull parking lots at transportation hubs and congestion in the surrounding area. In order to increase the appeal of public transport, however, the feeder system needs to offer a convenient and reliable service that matches the demand considering a limited operational cost. In a smart city scenario in the near future, service providers will be able to communicate with their passengers and drivers and create a two-way communication. This will allow to further improve service

quality and to operate a high-quality demand-responsive feeder service while controlling the operational costs.

Currently, two types of transit services can typically be distinguished: traditional transport services (TTS) and on-demand transportation services (ODTS). TTS are composed of a set of bus lines that follow predefined routes and schedules. A crucial advantage of TTS is their low operational costs, which arises from their ability to transport large groups of passengers collectively. The predictable nature of TTS, due to fixed schedules and routes, makes them highly accessible to most commuters. At the same time, however, the inflexibility of TTS is a limitation, since it renders TTS inadequate in settings where demand for transportation is sparse and constantly changing. To deal with these inefficiencies, ODTS emerged. ODTS are better able to cope with an ever-changing demand for transportation; these services operate only when there is demand for transportation and meet the passenger's expectations accordingly [3]. However, ODTS do not offer a solution to deal with the "passengers without reservation", e.g., passengers that do not request a ride, because they are unfamiliar with the service, but could still benefit from such a ride. As such, this research will focus on a transportation service situated between TTS and ODTS, integrating the positive characteristics from both types of transit service.

The feeder service considered in this paper is a demand-responsive feeder service (DRFS). In a feeder service, all passengers have the same destination but different origins. The bus line in the DRFS considered in this paper serves two sets of bus stops: mandatory stops, and clustered optional stops. The mandatory stops are visited by each bus. The optional stops are only visited by a bus when a client nearby makes a request for transportation. Potential passengers make a request for transportation to the transportation hub, by stating their current location and the latest time they want to arrive at the hub. This means that the routes and the timetables of the buses are not fixed but change according to the demand. The mandatory stops provide some predictability and serve as a safety net for the "passengers without reservation" because these passengers can still board a bus at the mandatory stops.

In order to optimize the performance of the service, a meta-heuristic, namely a large neighborhood search (LNS) [4], is developed. In the LNS framework, solutions are iteratively destroyed and rebuilt. In the rebuilding part of the algorithm, a weighted random based method is used.

The main contribution of this paper is the design of an efficient LNS algorithm that aims to optimize the performance of this demand-responsive feeder service under different circumstances. The results obtained by this LNS algorithm prove that the use of this meta-heuristic framework can result in better quality solutions without increasing the runtime significantly. A randomized search combined with a large neighborhood search can explore more areas of the solution and improve the solution quality without an increased runtime. Furthermore, a comparison with a traditional feeder service and an on-demand only feeder system shows that the service described in this paper can significantly increase the service quality for the passengers, while using the same fleet of vehicles.

In the next section, a literature review on public transport feeder services is presented. In Section 3, we present a model to optimize the service provided by the DRFS. Section

4 presents the heuristic that is developed to solve the optimization problem. Section 5 analyses the performance of the heuristic in order to fine-tune its parameters. In Section 6, the results for several instances, obtained by solving the problem using the heuristic, are presented and discussed. Section 7 compares the performance of the demand-responsive feeder system with a traditional feeder system and a on-demand only feeder system. In the last section, conclusions are drawn and plans for future research are discussed.

2. Literature Review

We introduced the DRFS in a previous paper [5], together with a mathematical model and optimal solutions for a set of smaller benchmark instances. To the best of our knowledge, the DRFS has not been discussed before by others. Still, closely related types of services have been considered in the literature and will be discussed in this section. This will allow to situate the DRFS with respect to the state of the art.

Traditional feeder services have predetermined stops, routes, and timetables. Traditional feeder services result in feeder route structures that perform well in terms of satisfaction of demand, and waiting time. In these traditional feeder services, the demand is considered to be known and is often derived from historical data. In such services, the Feeder Bus Network Design Problem (FBNDP) is often tackled. The FBNDP determines the design of a set of feeder bus routes, as well as the service frequency of each route for the period of time. The inputs for the FBNDP are typically the location of bus stops, the demand of passengers at each stop during a given period of time, the distance between each pair of stops, and the capacities of the fleet of buses [6]. Ciaffi et al. [7] solve the FBNDP using a heuristic algorithm to generate routes, together with a genetic algorithm to find the optimal network of routes and their frequencies. This solution approach was implemented for two real-size networks as well, namely in Winnipeg and Rome. Lin et al. [8] present a multi-objective programming approach to solve the FBNDP, where route length and the maximum route travel time are minimized, while the service coverage is maximized simultaneously. This model was used for a case study of a metro station in Taichung City, Taiwan. Mohaymany et al. [9] consider a FBNDP with multiple transportation modes, each mode with different capacities and performances, and used ant colony optimization to solve it. Zheng et al. [10] introduced a "demand coefficient" to quantify the feeder demand and used a tabu search algorithm to optimize the design of the feeder network. The optimization approach was implemented in a downtown area of Suzhou, China. Often, the main objective while optimizing these services is to minimize travel times of passengers and transfer times to the main transit. Shrivastava et al. [11] use a genetic algorithm combined with a specialized heuristic to optimize the routes and the timetables of a traditional feeder service. In a different study, Shrivastava et al. [12] use a genetic algorithm to optimize both the routes and the timetable simultaneously.

It is clear that traditional feeder services have their limitations, such as lack of accessibility and flexibility. In traditional feeder services, it is difficult to accommodate the different desired arrival times of the passengers. Furthermore, it is inconvenient for some passengers, such as children, disabled, or senior passengers to reach one of the limited number of bus stops served by traditional feeder lines [13]. These drawbacks have led to the rise of ODTS to improve the accessibility of transit services by either offering a door-to-door service, such

as dial-a-ride (DAR) services [14] or ride-sharing services [15]. Other services use predefined bus stops and passenger-stop assignment to reduce walking times as well as travel times of passengers, such as the school bus routing problem [16]. Sun et al. [17] present a mixed-integer linear programming model for a demand-responsive feeder service similar to a DAR problem [18, 19]. The model is solved using a heuristic algorithm and is used in a case study in Nanjing City, China [17]. However, these services typically have a high operational cost and a high degree of deadheading, i.e., vehicles often operate without carrying or accepting passengers. This limits the application of services like DAR and ride-sharing in practice.

A service that is closely related to the DRFS that we consider here and that integrates characteristics from TTS and ODTS is the Mobility Allowance Shuttle Transit (MAST) service. In this service, vehicles have a fixed set of bus stops they always need to visit, e.g. a fixed path, and these stops also have fixed timetables. However, the vehicles may deviate from the fixed path. The customers that are served outside of the fixed path are served at their desired location and need to be within a certain radius from the fixed path in a so-called "zone". This service combines the high flexibility of door-to-door services with a fixed main route [20]. This concept has been applied to feeder services as well. This type of feeder service allows the bus to deviate from the predetermined route to serve the requested nearby passengers and return to its original route before arriving to the next stop while respecting the fixed timetable. Lu et al. [21] developed a three-stage heuristic algorithm to optimize such a problem, together with a bus assignment sub-problem. Furthermore, Qiu et al. [22] analyzed a feeder service similar to a MAST service, which has been implemented in Salt Lake city in the USA. The authors concluded that the service has an advantage with respect to a fixed service under certain environmental circumstances. Another type of feeder service with flexible characteristics is the so-called demand responsive connector (DRC). In the DRC, no mandatory stops are considered and buses transport passengers from their origin location to transfer hubs within a predefined service area [23, 24]. In both feeder services, passengers who need a ride are required to make a request for transportation one to two hours in advance.

Services with flexible characteristics, like MAST and DRC, have more success in low demand areas with a sparse population, while TTS services thrive in high-demand and densely populated areas. When and where to use which feeder service is further discussed by Li [25]. This study shows that flexible feeder services perform better with lower demand rates and become progressively more preferred when more importance is given to the walking time of the passengers.

The DRFS, which is presented in this research, resembles the feeder service variant of the MAST service the most. The DRFS has a fixed route where it can deviate from, just as in MAST services. The main difference is that MAST services provides a door-to-door service to some customers within a certain radius, while the DRFS groups passengers at a number of bus stops. This increases the efficiency of the bus assignment and the routing. The timetable for the fixed route is also predefined in MAST services, limiting the time they can devote to deviating from the main route to provide the door-to-door service. This is not the case for the DRFS. Furthermore, in contrast to the DRFS, most MAST optimization models do not consider the capacity of the buses.

3. Problem Description

This paper deals with a demand-responsive feeder service (DRFS). In the DRFS, passengers are typically picked up from low-demand areas and transported to a transportation hub or a city. All passengers thus have the same destination, but different origins. Our DRFS contains one bus line with two sets of bus-stops: mandatory bus stops, and clustered optional bus stops. The mandatory stops need to be visited by each bus. The optional stops will only be visited by a bus when a passenger, within walking distance d_w, needs to be picked up. The buses can have different routes and timetables according to the demand. However, it is assumed that each bus starts operating at the first mandatory stop and stops operating at the last mandatory stop. The mandatory stops provide a factor of predictability, as well as a safety net for potential clients that may benefit from this service but have not made a formal request for it. Passengers in need for transportation without any reservation can go to a mandatory bus stop where they can board a bus. Figure 1 shows a theoretical example of the DRFS, with four clusters, six optional stops in each cluster, and six mandatory stops. The mandatory stops are represented by large purple dots and are placed along the main route, e.g., along a highway. The optional stops are shown as black dots. Each cluster of optional stops is scattered across a small town, village or neighborhood that is close to the main route, i.e., a cluster corresponds to a small town, village or neighborhood with several optional stops. The passengers are denoted as green triangles. The route of the bus line is the solid red line. In this route, the bus starts in the first mandatory bus stop m_0, and transports the passengers to the destination, i.e., the last mandatory stop m_5. The bus picks up a passenger at the third mandatory stop and deviates from the main route to pick up one passenger in cluster c_1 and three passengers in cluster c_2. In cluster c_2, two passengers are grouped at a single bus stop in order to reduce the travel times of the buses.

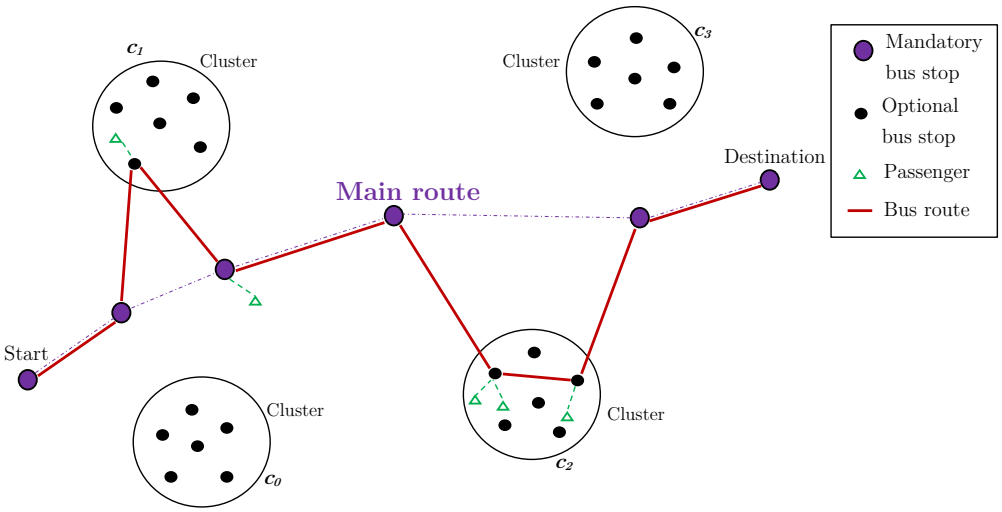


Figure 1: Theoretical example for the DRFS

Potential passengers make a request for transportation to the transportation hub, by stating their current location and their latest desired arrival time. After all requests are processed, the passengers are notified about the bus stop that is assigned to them with the departure time of their bus at that stop. Furthermore, the DRFS may offer an information service that notifies potential passengers about both the scheduled departure times and the optional bus stops of each bus for the current planning. This allows passengers that use this

application but did not make a reservation to go to one of the visited optional stops to catch a bus. Passengers that need transportation and are not aware of this information can only catch a bus at the mandatory bus stops. The requests from the passengers are assumed to be known at the moment the planning is made. This assumption can be viewed as a shortcoming of the service since it limits flexibility. Nevertheless, this assumption is not far-fetched since services have been implemented under these conditions before. An example of such a service is the "Belbus" from DeLijn in Belgium [26], a DAR service that allows passengers to make a request for transportation until one day before operation. Another example is the flexible feeder service in Salt lake city in the USA [22] that requires a request for transportation before the service starts. The ideal situation, however, is to offer a service that accepts requests in real-time, which is the future step in this research line.

The service works with a fixed number of buses. In the case that all passengers have been assigned to a bus and there are still buses available, the buses will still operate and only visit the mandatory stops in order to serve the potential passengers that have not made a reservation. The travel times of these buses are considered in the objective function as well. The timetables of these buses are assumed to be determined by the service provider.

A mathematical model that allows to optimize the performance of a DRFS on smaller instances is presented in [5]. Here, we limit ourselves to presenting the notations of the different parameters, sets and variables in Table 1, a textual explanation of the problem and a discussion on the objective function.

Different inter-depending decisions need to be made in this optimization problem. All passengers need to be assigned to a departure bus stop, taking into account the walking time. Consequently, a passenger is also assigned to a bus that will bring him or her to the destination on time. The routing of each bus needs to be determined based on the optional bus stops that are assigned to passengers and that are selected for each bus. Furthermore, the departure time of each bus at the first mandatory bus stop needs to be determined. This departure time will then determine the departure time at each bus stop and the arrival time at the destination. All these decisions are intertwined and affect one another. This problem can be viewed as an integration of a routing problem, an assignment problem, and a timetabling problem.

These decisions are also subject to a number of restrictions. First, the buses can have different routes according to the demand for transportation but all mandatory stops need to be visited by each bus. Second, the capacity of the buses is limited; no more passengers can be accepted in a bus if this capacity is reached. Third, the passengers are not allowed to walk longer than a certain amount of time from their origin location to their assigned departure bus stop. Fourth, given that the requests of the passengers are known before the optimization process starts, it is assumed that all requests need to be served. As a last restriction, all passengers need to arrive within their time window and, preferably, as close as possible to their desired arrival time. To model this, both a soft and a hard arrival time window is introduced for each passenger. The amount of time any passenger is allowed to arrive earlier or later than their desired arrival time is denoted by d_{early} and d_{late} respectively, i.e., all passengers $p \in P$ need to arrive within time window $[dat_p - d_{early}, dat_p + d_{late}]$. Furthermore, arriving earlier or later than the desired arrival time is penalized in the objective function. The arrival times of the passengers are thus part of both the constraints and the

Sets	
B	Set of buses
S	Set of all bus stops
O	Set of optional bus stops
F	Set of mandatory bus stops
P	Set of passengers using the service during the optimization horizon
Parameters	
K_k	Number of optional bus stops in cluster k
M	Number of clusters
tt_{ij}	Travel time from bus stop $i \in S$ to bus stop $j \in S$
τ	Dwell time per passenger boarding
wt_{pi}	Walking time of passenger $p \in P$ to departure bus stop $i \in S$
dat_p	Desired arrival time of passenger $p \in P$ at the destination bus stop $m_{\|F\|-1}$
δ	Average acceleration and deceleration time of a bus
d_w	Maximum value for walking time of any passenger
d_{late}	Maximum value for arriving later than the desired arrival time of any passenger
d_{early}	Maximum value for arriving earlier than the desired arrival time of any passenger
C	Capacity of the buses
w_1	Relative weight given to the travel time of the buses
w_2	Relative weight given to the walking time of the passengers
w_3	Relative weight given to the absolute difference in desired and actual arrival time of the passengers
Decision Variables	
x_{bij}	0-1 variables determining if bus $b \in B$ visits bus stop $j \in S$ after visiting bus stop $i \in S$
y_{pbi}	0-1 assignment variables which assume value 1 if passenger $p \in P$ is assigned to bus $b \in B$, with departure bus stop $i \in S$
a_p	Arrival time of passenger $p \in P$ in destination bus stop $m_{\|F\|-1}$
D_b	Departure time of bus $b \in B$ at the first mandatory bus stop m_0

Table 1: Notation for the parameters, sets and variables of the optimization problem

objective function.

While optimizing this demand-responsive feeder system the objective is the service quality, which is modeled as a weighted sum of three factors. First, the travel time for all buses is minimized (1). Shorter travel times for the buses will reduce the operating costs and imply shorter ride times for the passengers and thus improve the service quality. The acceleration and deceleration time δ and, the dwell time τ of a bus are included as well, in order to model the travel time in a more realistic way. Second, the total walking time from the origin location of each passenger to the departure bus stop needs to be as low as possible (2). Third, the absolute time difference between the desired arrival time and the actual arrival time at the destination, of each passenger, needs to be as close to zero as possible (3). The weights w_1, w_2 and w_3 of this sum can be determined depending on the situation and on the preferences of the service provider.

$$\text{Minimize: } z = w_1 \left[\sum_{b \in B} \sum_{i \in S} \left(\sum_{j \in S} (tt_{ij} + \delta) x_{bij} + \tau \sum_{p \in P} y_{pbi} \right) \right] \tag{1}$$

$$+ w_2 \left[\sum_{b \in B} \sum_{i \in S} \sum_{p \in P} wt_{pi} y_{pbi} \right] \tag{2}$$

$$+ w_3 \left[\sum_{p \in P} |dat_p - a_p| \right] \tag{3}$$

It needs to be noted that the time that the passengers wait for a bus to show up is considered to be zero. In this model, it is assumed that there are no significant delays and that the passengers will make a request beforehand. This means passengers can plan to arrive at the bus stop on time without incurring longer waiting times. Furthermore, we choose to minimize the travel time of each bus rather than the travel time of each passenger. This is done in order to optimize the routes of the buses rather than the onboard time of each passenger individually. Optimal bus routes, however, also imply shorter travel times for passengers onboard. The onboard times are typically less important to customers compared to the reliability of the service and arrival time at their destination [27, 28]. Service reliability is inherently part of this service as it deals with passenger requests and a timely arrival at the destination is covered by the third component of the objective function.

4. Solution approach

The DRFS that is considered in this paper was previously modeled mathematically and solved with exact solution techniques [5]. However, the required runtimes are too large for instances of realistic size and do not allow an in-depth analysis of the circumstances appropriate for a DRFS or a comparison with a traditional system or an on-demand system. Heuristic algorithms can overcome the limitations of exact approaches, and are generally much faster than exact methods. The main drawback is that optimal solutions are not guaranteed. Nevertheless, well-developed heuristics can deliver near-optimal results in a very short time span. There are several definitions and classifications of heuristics in the state-of-the-art, however, we opted to use the terminology presented by Sörensen and Glover

(2013) [29]. This terminology has been used in several other works, such as Kayvanfar et al. [30], Asta et al. [31], and Sun et al. [32]. According to Sörensen and Glover [29], there is an important distinction between a heuristic, which is simply an optimization algorithm that does not guarantee optimality, and a meta-heuristic, which is defined by the authors as: "a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms". Sörensen and Glover [29] further subdivide meta-heuristics into three main categories: local search algorithms, constructive algorithms, and evolutionary algorithms. The heuristic that is developed in this study is a constructive heuristic, where solution elements of an existing, incomplete solution are added in each iteration, and mainly follows the framework of LNS [4]. In the LNS framework, an existing solution is partially destroyed and rebuilt in each iteration. A weighted random based method is present in our heuristic as well, namely the randomized construction of solutions. Aspects of the Pilot method [33] were considered too: looking ahead during the construction of the solution. However, it became evident that the random based method made our algorithm perform better in this problem setting.

In order to develop an appropriate algorithm for optimizing the performance of a DRFS, we start from an in-depth understanding of the problem. In this problem, four main decisions need to be made. First, the assignment of passengers to buses needs to be determined. Second, the decision on the assignment of passengers to departure bus stops, which is dependent on the walking distance, needs to be made. Third, the routing of buses needs to be determined based on the location of the stops and according to which stops are used. Lastly, a decision is made on the timetabling of the buses, which depends on the requests of the passengers. The assignments of passengers to buses and passengers to bus stops have implications on all other decisions. After the passengers are assigned to a bus and a bus stop, the problem becomes a routing problem. When the passengers are assigned to a bus and its route is determined, the timetabling is limited to determining the optimal departure time. Furthermore, the assignment of passengers to buses must consider all buses, while the routing and timetabling problems can be solved for each bus independently.

Since all decisions are intertwined and affect each other, we start the optimization by making one decision random. The heuristic is based on a randomized search procedure for the optimal passenger-bus assignment. Afterwards, the other subproblems are each solved independently for each bus. We propose a large neighborhood search (LNS) heuristic to tackle this problem. The LNS framework is often used for the optimization of timetables and routes [34, 35, 36], which makes it a suitable meta-heuristic to use for the optimization of the DRFS. The main steps for this framework are presented in Algorithm 1. The LNS algorithm requires an initial feasible solution as an input. After this, the algorithm enters a loop. In first step of the loop, the solution is partly destroyed by a "destroy operator" $d(.)$. The destroyed solution is then rebuilt using a "repair operator" $r(.)$ in the second step. Lastly, in the third step, the new solution is compared to the previous solution and it is then determined if this new solution will replace the current one. This is done based on a given acceptance criterion. For example, by comparing the objective function value of both solutions. The process of destroying and rebuilding continues until a stopping criterion is met, for example, after a number of destroy-repair cycles or after a time limit [4].

Algorithm 1: Large Neighborhood Search algorithm

```
Input: initial solution s_0
while stopping criterion is not met do
    Destroy s_0 by using d(s_0).
    Rebuild the destroyed solution. A new solution s = r(d(s_0)) is generated.
    if acceptance criterion is met then
        s is accepted.
        s_0 = s.
    end
end
```

4.1. Initial solution generation

To start the LNS algorithm, an initial solution is needed. The algorithm for the construction of the initial solution is given in Algorithm 2. The initial solution is constructed by iteratively constructing a solution for each bus b separately. All passengers are put in a list L_p and are sorted according to their desired arrival time dat_p. The passengers are then added to each bus $b \in B$ in this order. Passengers that are assigned to bus b are added to list L_b and removed from list L_p. The algorithm will stop adding passengers to a bus b either when the bus reaches its capacity or when $\max_{p \in L_b} dat_p - \min_{p \in L_b} dat_p < d_{late} + d_{early}$. The last expression ensures that all passengers reach the destination within their time window. In each bus b, all passengers are assigned to the closest bus stop to their location, as their departure bus stop. Afterwards, the route of bus b is determined. To build the bus route, all the mandatory stops are added to the route first. Next, all the optional stops that are assigned to passengers assigned to bus b are inserted in the existing route, between the two closest mandatory stops but otherwise in an arbitrary manner. It needs to be noted that this is a simple algorithm, designed to quickly generate feasible routes for the initial solution. Later, in the repair operator, a better and more complex algorithm will be used. In the last step of the construction of the initial solution of bus b, the departure time of each bus is determined. Since the route of the bus is already determined, its departure time at each stop and the arrival time at the destination follow directly from the departure time from the first mandatory bus stop. The bus needs to arrive at the destination within the given time window $\left[\max_{p \in L_b} dat_p - d_{early}, \min_{p \in L_b} dat_p + d_{late}\right] = [LB_b, UB_b]$. Here, UB_b and LB_b denote the upper and lower bounds of the interval, for bus b. The arrival time of the buses is determined in such a way that the arrival time of the passengers is optimized. This is expressed in the objective function as $\sum_{p \in L_b} |dat_p - a_p|$, $\forall b \in B$. This is a sum of absolute deviations, and as shown by Dodge [37], the median, in this case the median of the desired arrival times of the passengers onboard the bus, minimizes this sum. However, an additional constraint is that the solution must be within the interval $[LB_b, UB_b]$. Whenever the median is larger than UB_b the arrival time will be set to UB_b. When the median is smaller than LB_b, the arrival time will be set to LB_b. The correction of the arrival time still gives the best possible solution since $\sum_p |dat_p - a_p|$ is a convex function of a_p. For the next bus, the remaining passengers in the list are considered and the construction of the solution for the next bus starts. This construction process is repeated until there are no passengers left in the list or all available buses have been used.

Algorithm 2: Construction of initial solution

Add all passengers $p \in P$ to list L_p
Order passengers in L_p according to desired arrival time.
while L_p *is not empty* **do**
    Choose a bus b with empty list L_b
    while $\sum_{i \in S} \sum_{p \in P} y_{pbi} < C$ **do**
        Select the next passenger in L_p as passenger p
        if $\max_{p \in L_b} dat_p - \min_{p \in L_b} dat_p < d_{late} + d_{early}$ **then**
            break
        else
            Assign passenger p to bus b
            Determine and assign the closest bus stop to passenger p
            Add passenger p to L_b
            Remove passenger p from L_p
        end
    end
    Add a route visiting all mandatory bus stops $m \in F$.
    for $\forall o \in O$ **do**
        if *o is assigned to a passenger onboard bus b* **then**
            Add o to the route
        end
    end
    Calculate the best arrival time for bus b: $a_b = \underset{p \in L_b}{\text{median}}\{dat_p\}$
    if $a_b > UB_b$ **then**
        $a_b = UB_b$
    else if $a_b < LB_b$ **then**
        $a_b = LB_b$
    end
    Calculate total travel time TT_b. Set departure time: $D_b = a_b - TT_b$
end

During the construction of the initial solution, an infeasible instance is detected. If the passenger cannot arrive within their time window, the initial solution will not have assigned all passengers to a bus, which makes the instance infeasible. If the distance to the closest bus stop to any passenger is larger than d_w, then the instance is infeasible too. If the capacity of the buses times the number of available buses is smaller than the number of passengers, the capacity constraint cannot be respected and the instance is infeasible as well.

4.2. Destroy operator

In order to create a new and possibly better solution, the current solution needs to be partially destroyed by a destroy operator. The destroy operator randomly selects a predetermined number of passengers σ and removes them from the solution. This is done by going through each bus in the fleet and removing the randomly selected passengers from their bus. When passengers are removed from a bus, these passengers are not assigned to a departure bus stop or to a bus anymore. This means that this bus has less optional stops it needs to visit. The optional stops that each bus b needs to visit are kept in list S_b. The timetables of the buses where passengers were removed from, are removed from the solution. The routes, i.e., the sequence of the stops that are visited, are removed from the solution too. The algorithm for the destroy operator is given in Algorithm 3.

Algorithm 3: Destroy operator

```
1  Randomly choose σ passengers and place them in a list R
2  for each bus b do
3      Add all optional stops that are vsited by b to list S_b
4      for ∀p ∈ R ∩ L_b do
5          if there exists a new feasible assignment then
6              Undo passenger-bus and passenger-stop assignments for passenger p
7              Remove p from L_b
8              if p was assigned to an optional stop and no other passenger in L_b is
                assigned to the same stop then
9                  Remove the optional stop from S_b
10             end
11         end
12     end
13     if passengers were removed from bus b then
14         Destroy route and timetable of bus b
15     end
16 end
```

It needs to be noted that the destroy operator determines whether a new feasible bus assignment exists, i.e., the new assignment does not violate the capacity constraints or the time window constraints. If no new bus assignment is feasible, the passenger is not removed. This allows the repair operator to always find a feasible new assignment.

4.3. Repair operator

The repair operator is given in Algorithm 4 and works similarly to the initial solution algorithm. The first step is to assign the passengers that were removed by the destroy operator to a bus. For each passenger, this new bus assignment needs to be feasible and should be different from his previous bus due to the feasibility checks in the destroy operator. The reassignment is done by taking into account the desired arrival times dat_p of the passengers onboard the buses in the following way. First, the average arrival time $\hat{dat}_b$ of the passengers that are still onboard bus b is calculated, for each bus $b \in B$. Second, the deviation $\left|\hat{dat}_b - dat_{p'}\right|$ is determined for each bus b and each passenger p' that was removed from the solution by the destroy operator. Lastly, the buses are ranked for each removed passenger p', according to the deviation $\left|\hat{dat}_b - dat_{p'}\right|$. The rank of a bus b determines the probability $P_{bp'}$ of being selected as the new bus for passenger p'.

The probability $P_{bp'}$ of a bus being selected is determined as follows. First, $|B|$ different ranges are determined and assigned to each bus. This is done by calculating the cumulative sum of buses: $N_b = \sum_{b=1}^{|B|} b$. The lowest ranked bus, with rank 0, is assigned to range $[N_b - |B|, N_b[= [r_1, r_0[$, the b^{th} ranked bus is assigned to $[r_{b-1} - |B| + b, r_{b-1}[= [r_b, r_{b-1}[$ and so on. The range becomes smaller as the rank of the bus increases. Finally, a random number r between 0 and N_b is drawn and a bus b is selected if r is within the range corresponding with bus b. This means lower ranked buses are chosen with a higher probability because their range is proportional to their rank.

The next step is to assign new departure bus stops to newly assigned passengers in each bus $b \in B$. First, it is determined if the closest stop is already a part of the solution of bus b, i.e., if the bus stop is either a mandatory stop or an optional stop in L_b. If that is the case, obviously, this stop is assigned to the passenger. Otherwise, the second and third closest stops for this passenger are chosen and added to L_b with a probability of $\pi_2\%$ and $\pi_3\%$ respectively. These probabilities are relatively low, so the closest bus stop is still chosen most often. The second and third closest bus stops are chosen at times to ensure more possible solutions are explored. If the closest stop is not in list L_b already, there is a possibility to reduce the objective function value by trading longer walking times for shorter travel times. Another strategy that we considered, and that perhaps seems to make more sense, is to proactively predict the best option between choosing the closest stop and choosing a stop that reduces the travel time of the bus, in a manner similar to the Pilot heuristic framework [33]. This was done by estimating the extra walking and travel time that would be added to the objective if a certain bus stop within walking distance was chosen. The pilot approach, however, often yields slightly higher (worse) objective function values than the weighted random based approach and has on average 28% larger runtimes when it is run on the instances described in Section 5. The results for the LNS algorithm that uses the pilot method for the bus stop assignment are presented in Table A.6 in Appendix A.

The algorithm to rebuild the routes starts at the first mandatory stop m_0 with no stops added to the route yet. The algorithm stops when the last mandatory stop $m_{|F|-1}$ is reached. A route is built by always choosing the closest eligible bus stop to the current stop as a successor. A stop is eligible for a bus if it is a mandatory stop or if it is an optional

Algorithm 4: Repair operator

for *all removed passengers* **do**
  Reassigned removed passenger p' with probability $P_{bp'}$ to a different bus b
  if *constraints are violated* **then**
    go back to 2
  end
  Determine and assign the closest bus stop s_i to p'
  if s_i *is an optional stop* **and** $s_i \notin L_b$ **then**
    Reassign and add the second closest stop s_j to L_b with $\pi_2\%$ probability
    if s_j *is not reassigned* **then**
      Reassign and add the third closest s_k to L_b with $\pi_3\%$ probability
    end
  end
end
for *each bus* $b \in B$ **do**
  Rebuild the route: $s_i = m_0$
  while $s_i \neq m_{|F|-1}$ **do**
    Determine which **optional** stops are *eligible*
    Determine the closest eligible stop s_k
    $s_i = s_k$
    if *no optional stop is eligible* **then**
      Determine the next **mandatory** stop s_k
      $s_i = s_k$
    end
  end
  Calculate the best arrival time for bus b: $a_b = \underset{p \in L_b}{\text{median}}\{dat_p\}$
  if $a_b > UB_b$ **then**
    $a_b = UB_b$
  else if $a_b < LB_b$ **then**
    $a_b = LB_b$
  end
  The best arrival time for each passenger p onboard is then: $a_p = \sum_{i \in S} y_{pbi} a_b$
  Calculate total travel time TT_b on bus b
  The best departure time for bus b is then: $D_b = a_b - TT_b$
end

stop assigned to a passenger that is also assigned to this bus. It needs to be noted that in this search, the optional stops are considered before the mandatory stops in order to always go to all the stops within a cluster before returning to a mandatory stop, in the main road. The remaining variables are determined in the same manner as in Algorithm 2.

4.4. Acceptance criterion

In each iteration of the LNS heuristic, the current solution is transformed into a new solution by the destroy and repair operators, and the objective function value of the new solution is compared to the objective function value of the original solution. The objective function value is calculated as described in Section 3. If the objective improves, then the current solution is replaced by the new solution in the next iteration, i.e., in the next destroy-repair cycle of the LNS algorithm.

4.5. Stopping criterion

After the initial solution is computed, the loop of the LNS algorithm starts. To ensure the algorithm does not stop prematurely, i.e., before the last significant improvement is made, the algorithm will stop if there was no improvement in the last 5000 iterations. In each iteration, the LNS algorithm uses a destroy and a repair operator.

4.6. Post-processing

When the LNS algorithm finishes its destroy-repair cycles, the heuristic stops. After the LNS algorithm finishes, the final solution can still have some inefficiencies in the routing of the buses. These inefficiencies can be corrected with a 2-opt algorithm. This is necessary because the routing part of the repair operator is a fast and greedy algorithm that is designed to obtain the best assignment of passengers to buses. The 2-opt algorithm implemented here is a first-improvement algorithm, and selects two edges of the existing route and swaps them if and only if the objective function value is lowered by this swap. The algorithm randomly chooses the first edge in the existing route, the second edge is chosen from either clusters nearby or edges connecting mandatory stops nearby. This is done because most or all inefficiencies are localized in the routes of the buses. In this paper, this operation is performed 200 times.

5. Experimental set-up

To test the LNS algorithm, different instances are used. Instances I_1 to I_{14} are benchmark instances that were previously solved in [5]. The remainder of the instances are randomly generated for this paper. All instances are listed in Table 2. The instances have different number of buses $|B|$, requests $|P|$, bus capacity C and bus stops $|S|$. Furthermore, there is always one cluster in-between two mandatory stops and each cluster has the same number of bus stops K. This means there are $M = |F|-1$ clusters in each instance, with $|F|$ the number of mandatory stops. The last column shows the number of variables V of the mathematical model built based on the instance, this is an indication of the size of each instance. The list of instances can be divided into different subsets as follows. First, the optimal objective function values are known for instances I_1 to I_{14}. Second, instances I_{15} to I_{18} are larger instances and have the same instance parameters except for the number of

optional stops per cluster, which increases gradually. Third, instances I_{19} to I_{24} have the same instance parameters except for the number of passenger requests. Fourth, in instances I_{25} to I_{30}, only the number of buses changes. Finally, in instances I_{31} to I_{36}, only the bus capacity changes.

For the remainder of Section 5, a representative subset TS_a of these instances, namely I_3, I_7, I_{10}, I_{14}, I_{15}, I_{18}, I_{20}, I_{23}, I_{26}, I_{28}, I_{32} and I_{36}, is selected. These instances are indicated in bold in Table 2. This subset is a representative sample of the entire set of instances since it contains instances from each subset that is described above. In this section, different parameters of the LNS algorithm are evaluated using statistical tests, with the purpose of fine-tuning the LNS algorithm and obtaining better results. The LNS algorithm is run on a computer with a Windows 10 Enterprise operating system, an Intel Core$^{\text{TM}}$ i7-8850H, 2.60Ghz CPU and 16 GB of RAM. The details of the parameters of the instances, as well as the solutions discussed in this paper are available in detail online: `https://www.mech.kuleuven.be/en/cib/drbp#section-4`. The objective weights w_1, w_2 and w_3 are set to 0.25, 0.35 and 0.40 respectively. The values are chosen with the goal of giving each component of the objective function approximately equal importance. The objective weight values are not exactly equal in order to compensate for the fact that the travel times are typically larger than the walking times, which in turn are typically larger than the difference in arrival times. A detailed study of the effect of the weight values on the solutions is beyond the scope of this paper.

Section 5.1 describes the methodology that is used to analyze the performance of the LNS algorithm with different parameter settings. In Section 5.2, the results of the analysis of the destruction factor σ are discussed and the best destruction factor is determined, and in Section 5.3, the best parameters π_2 and π_3 are determined.

5.1. Methodology

The analysis is performed by running the LNS algorithm on the training set TS_a. For each of these instances, the LNS algorithm is run with different values of the σ, π_2 and π_3 parameters. Only one parameter is changed at a time, in order to isolate their effect on the results. Different values may result from the LNS algorithm, even with the same set of parameter values, due the randomness of the algorithm. To obtain an adequate estimate of the objective function value, the LNS algorithm was run 50 times for each set of parameter values for each instance.

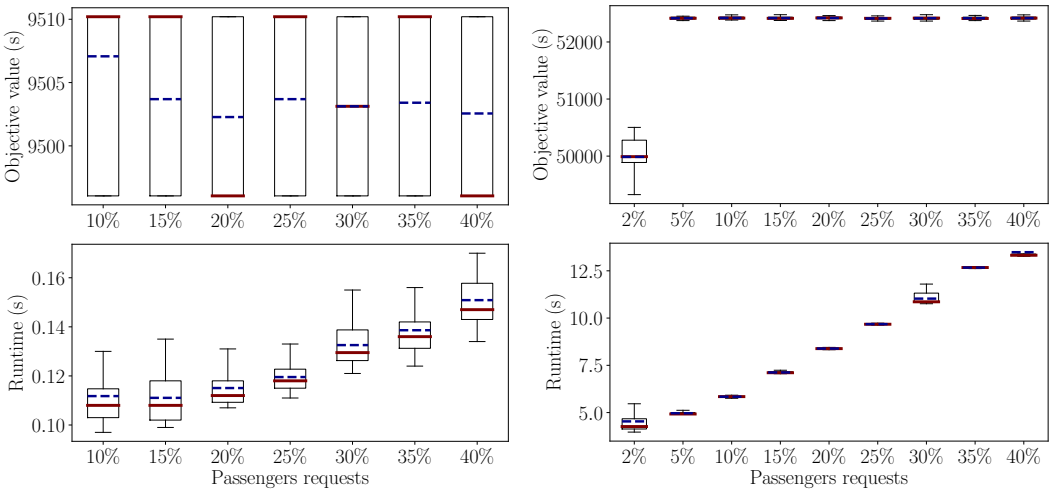
The LNS parameter values that optimize the performance of the LNS algorithm, for each instance, are determined as follows. For each instance, the mean μ_{v_1}(of the 50 runs) of the objective function values that are computed by the LNS algorithm, using a certain set of LNS parameters v_1, is compared to another mean value μ_{v_2} that resulted from using a different set of LNS parameters v_2 in the LNS algorithm. For example, let v_1 correspond with the parameter values $[\sigma = 20, \pi_2 = 30, \pi_3 = 5]$ and v_2 with $[\sigma = 15, \pi_2 = 30, \pi_3 = 5]$. These mean values are compared using paired, one-sided t-tests [38]. The null hypothesis is H_0: $\hat{\mu_{v_2}} - \hat{\mu_{v_1}} = 0$ and the alternative hypothesis is H_A: $\hat{\mu_{v_2}} - \hat{\mu_{v_1}} > 0$. The level of significance to reject the null hypothesis is $\alpha = 0.05$.

Inst.	|B|	|F|	K	|S|	|P|	C	V
I_1	2	3	3	9	12	15	1226
I_2	3	3	3	9	12	15	1335
I_3	**2**	**4**	**3**	**13**	**16**	**15**	**3170**
I_4	3	4	3	13	16	15	3379
I_5	2	5	3	17	20	15	6522
I_6	3	5	3	17	20	15	6863
I_7	**4**	**5**	**3**	**17**	**20**	**15**	**7204**
I_8	3	6	3	21	25	15	12678
I_9	4	6	3	21	25	15	13204
I_{10}	**3**	**6**	**3**	**21**	**30**	**15**	**15213**
I_{11}	4	6	3	21	30	15	15844
I_{12}	4	7	3	25	30	15	21844
I_{13}	5	7	3	25	30	15	22595
I_{14}	**5**	**10**	**3**	**37**	**40**	**15**	**62285**
I_{15}	**7**	**12**	**2**	**34**	**50**	**15**	**69857**
I_{16}	7	12	3	45	50	15	117157
I_{17}	7	12	4	56	50	15	176557
I_{18}	**7**	**12**	**5**	**67**	**50**	**15**	**248057**
I_{19}	24	12	5	67	45	30	274524
I_{20}	**24**	**12**	**5**	**67**	**90**	**30**	**549024**
I_{21}	24	12	5	67	158	30	963824
I_{22}	24	12	5	67	225	30	1372524
I_{23}	**24**	**12**	**5**	**67**	**395**	**30**	**2409524**
I_{24}	24	12	5	67	510	30	3111024
I_{25}	7	12	5	67	158	30	783845
I_{26}	**8**	**12**	**5**	**67**	**158**	**30**	**794432**
I_{27}	12	12	5	67	158	30	836780
I_{28}	**16**	**12**	**5**	**67**	**158**	**30**	**879128**
I_{29}	20	12	5	67	158	30	921476
I_{30}	26	12	5	67	158	30	984998
I_{31}	24	12	5	67	158	10	963824
I_{32}	**24**	**12**	**5**	**67**	**158**	**15**	**963824**
I_{33}	24	12	5	67	158	20	963824
I_{34}	24	12	5	67	158	25	963824
I_{35}	24	12	5	67	158	40	963824
I_{36}	**24**	**12**	**5**	**67**	**158**	**50**	**963824**

Table 2: List of test instances

5.2. Destruction factor

The destruction factor σ is the percentage of passenger requests that are removed from the solution by the destroy operator of the LNS algorithm. As stated in [4], if this factor is too low the solution space cannot be explored sufficiently. On the other hand, if the factor is too large, the search algorithm takes on a random character, yielding worse results and increasing the runtime. For these reasons, the largest values of σ will be limited to 40% of the number of passenger requests, which means that $\sigma \in [1\%, 40\%]$. The lower bound is not 0% because the LNS algorithm needs to remove at least one passenger from the solution in order to obtain a different solution in each iteration of the destroy-repair cycles.



(a) Instance I_{10}

(b) Instance I_{26}

Figure 2: Boxplots for destruction factor σ

Statistical tests reveal that for most instances there is no simple optimal value for σ. Figure 2 shows the box plots for the objective function values that are obtained after running the LNS algorithm with different values of σ, for two different instances. The mean values are denoted by a striped blue line. The box plots of the runtime of the LNS algorithm are given as well. It can be noted that, generally, a higher value of σ corresponds to a larger runtime. However, when σ is too low, the runtimes fluctuate significantly and the algorithm may take longer to stop. In some instances, there are several values which are not statistically different from the lowest mean value that was observed, i.e., for which the null hypothesis H_0 could not be rejected. Instance I_{10}, for example, has no clear optimal σ while instance I_{26} does have an optimal σ, namely the lowest percentage of passenger requests that was tested for I_{26}: 2%. This parameter value yields lower objective function values and the lowest mean value for I_{26}. It also results in one of the lowest mean values for the runtime of I_{26} as well.

The fact that there is often no clear optimal σ value means that there are several values of σ which can be considered to be optimal with certain probability. These σ values will be called ranges of optimality and are displayed in Figure 3, for all instances. The lowest observed means are displayed as well. If a σ value is not statistically different from the lowest mean, i.e., if the null hypothesis could not be rejected, this σ value is added to the range of optimality. These ranges of optimality are based on the objective function values

and not on the runtimes. This is done because, generally, the LNS parameters that yield better solutions either have the lowest runtimes as well, or the increase of the runtime is not significant in comparison to other solutions with higher objective function values. From the results, it is clear that very low percentages of destruction are preferred for larger instances with more passenger requests. If a relatively larger percentage of passengers is removed from these larger instances, the LNS algorithm probably becomes too random. For this reason, Figure 3 shows the number of passenger requests that are destroyed, rather than a percentage of the total number of passenger requests.

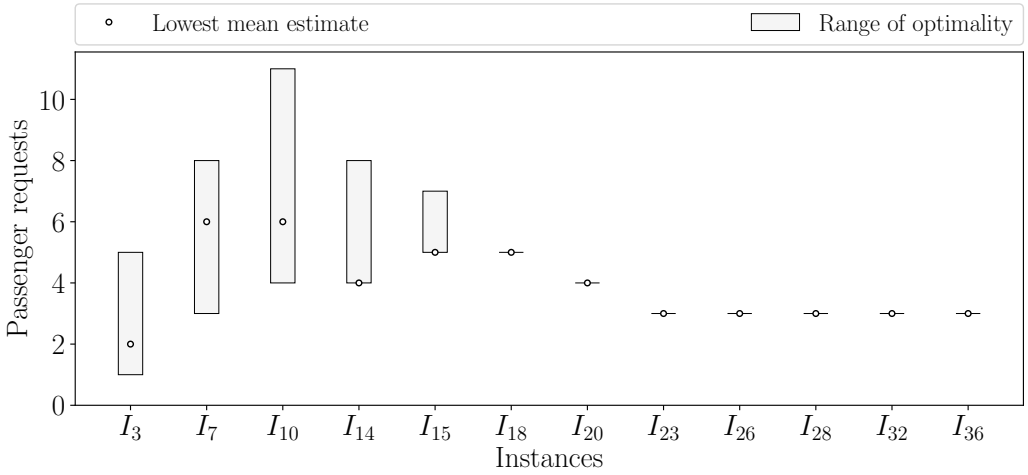
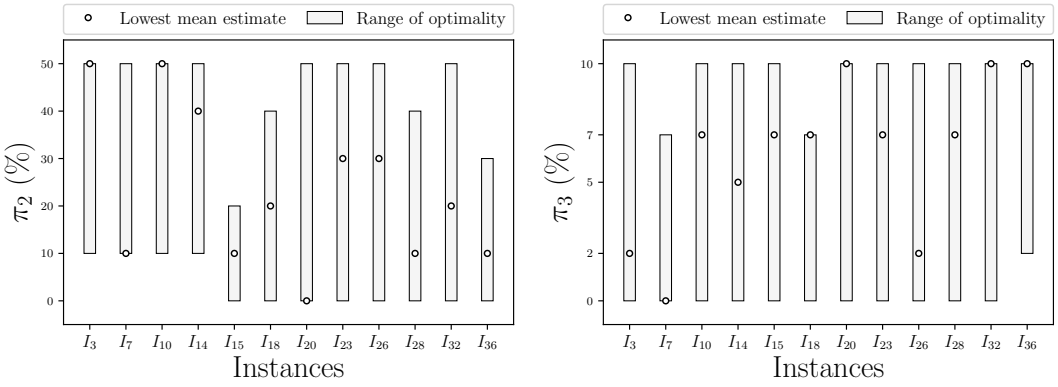


Figure 3: Ranges of optimality for destruction factor σ

It can be seen that instances I_{18} to I_{36} have simple optimal values for σ. The other instances have ranges of optimality. Furthermore, there is no common value for σ which optimizes the LNS algorithm for all instances. The best σ value depends on the instance. A good compromise is to make σ stochastic, randomly choosing the value of $\sigma \in [3, 5]$ in each repair-destroy cycle of the LNS algorithm. This strategy will increase the randomness of the algorithm slightly, but it might yield better results in general, for different kinds of instances. With this strategy, the LNS algorithm assumes the best σ value in some of its destroy-repair cycles when it is run on any of the test instances in the representative subset TS_a.

5.3. Bus stop parameters

In these experiments, the influence of the π_2 and π_3 parameters on the performance of the LNS algorithm is analyzed. Parameters π_2 and π_3 are the probabilities that the second closest and third closest bus stop respectively is assigned to a passenger. The parameters assume the following values: $\pi_2 \in [0\%, 50\%]$ and $\pi_3 \in [0\%, 10\%]$. Both parameters have ranges of optimality. The range of optimality and the lowest observed mean values for different π_2 and π_3 values are given in Figure 4. However, π_2 has less influence on the objective function value compared to σ, and π_3 in turn has even less influence on the results. The value of these parameters does not seem to have a significant influence on the runtime of the LNS algorithm. This being said, however, Figure 4 shows that the inclusion of these mechanisms in the LNS algorithm improves the objective function values in most instances.



(a) Second closest stop factor π_2

(b) Third closest stop factor π_3

Figure 4: Ranges of optimality for bus stop parameters π_2 and π_3

It is apparent that values of $\pi_2 = 25\%$ and $\pi_3 = 5\%$ are a good option. The ranges of optimality are quite large, when the significance level α is set to 0.05. This means that the choice of these parameters is less important to the performance of the LNS algorithm. However, more often than not π_2 and π_3 have non-zero optimal values, which justifies their inclusion.

6. Performance of the LNS algorithm

In this section, the results for all the instances that are discussed in Section 5 are presented. In the previous section, the parameters of the LNS algorithm were determined on a training set of instances. It is decided that parameter σ will be randomly chosen between 3 and 5 passengers in each destroy-repair cycle, parameter π_2 will be set to 25% and parameter π_3 to 5%. First, the performance of the LNS algorithm on a single instance, namely instance I_{14}, is discussed. This instance is run several times in order to have insight into the behavior of the LNS algorithm. Afterwards, all instances are evaluated. To test the LNS algorithm, each instance was run 10 times. The best solution, the mean value of these runs and the standard deviation are reported.

6.1. Behavior of the LNS algorithm

Figure 5 shows the progress of improvement of an individual run for instance I_{14}. In this figure, each black circle represents a lower feasible objective function value that was found by the LNS algorithm during the destroy-repair cycles. The red triangle represents the objective function value that is obtained after the 2-opt algorithm. The blue lines correspond with the runtime until the last improvement is found. The red line corresponds with runtime after the last improvement and includes the 2-opt algorithm.

As expected, the LNS algorithm makes most of its improvements at the start. Most of the runtime is devoted to reaching the stopping criterion of the LNS algorithm: 5000 destroy-repair cycles without an improvement. This means that the algorithm can sometimes yield good results, even when the number of cycles without improvement, in the stopping criterion, is lowered.

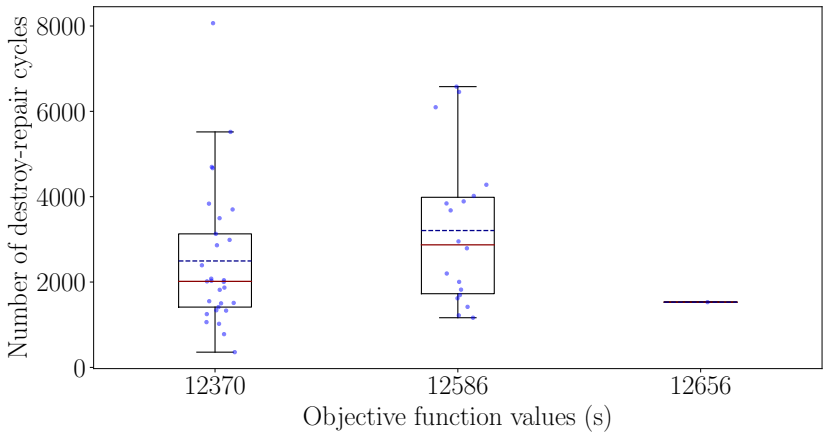


Figure 6: Required number of destroy rebuild cycles for the observed objective function values of I_{14}

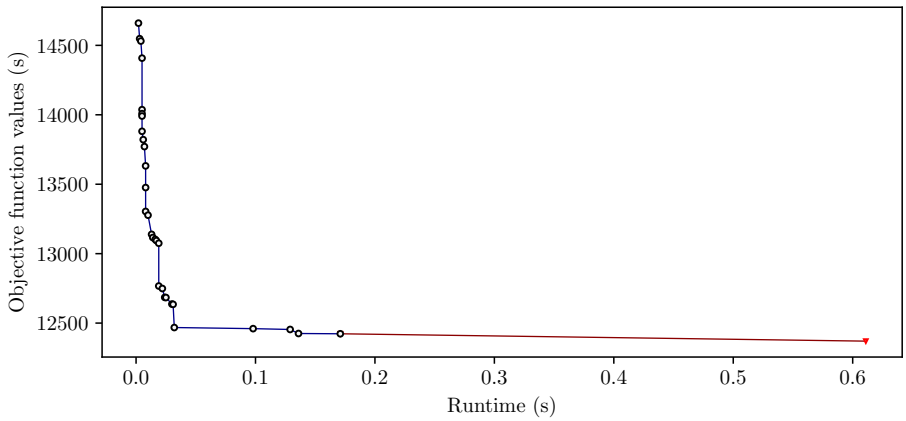


Figure 5: Progress of improvement, instance I_{14}

Instance I_{14} is run 50 times to observe the different objective function values that the LNS algorithm produces when different streams of random numbers are utilized. During these 50 runs, the algorithm produces three different objective function values. In figure 6, the number of required destroy-repair cycles, i.e., the last cycle that found an improvement, is given for each objective function value. It can be seen that each objective function value has a wide range of required number of cycles. Furthermore, the best objective function value does not always require a larger number of cycles. This means that better quality solutions, for an LNS algorithm with identical LNS parameters, are obtained when the stream of random numbers is favorable and not necessarily when more destroy-repair cycles are performed.

It can be concluded that, in practice, it is more advisable to limit the runtime and increase the number of runs of the LNS algorithm, rather than increasing the runtime and limiting the number of runs, to obtain better solutions. Furthermore, the runs are independent from each-other and are thus easily implemented using parallel computing, which can reduce the overall computing time.

6.2. Experimental results

The LNS algorithm is used to solved all of the instances that were described in Section 5, the results are summarized in Table 3. Here, the best (lowest) objective function value and runtime for a single run are given. The mean and the standard deviation of the objective function values and the mean and the relative standard deviation of the runtimes are presented as well.

For the first 14 instances, the optimal values are known [5]. This allows us to determine the optimality gap, between the optimal and the best observed solution of the LNS algorithm. The gap makes it possible to assess the performance of the LNS algorithm. The minimum and mean gaps from the 10 runs are presented in Table 3 as well. Column "Exact", in Table 3, shows the runtimes to solve the instances with the fastest mathematical model presented in [5]. The mathematical model is solved with the help of CPLEX, and both the LNS algorithm and the mathematical model are run on the same computer. It needs to be noted, however, that CPLEX uses 12 threads to solve the mathematical model, while the LNS algorithm only uses one. If we assume that the LNS algorithm can easily be run in parallel and that the 12 threads used by the CPLEX solver are equivalent to 10 parallel runs of the LNS algorithm, then the minimal gaps can be used for a fair comparison.

It can be seen that the LNS algorithm performs quite well for the first 14 instances, yielding minimum optimality gaps of 0.2% or less, when the best run is considered. The objective function values of the other, larger instances are more difficult to evaluate because there is no known optimal value. However, the low gaps in the first 14 instances give an indication that the LNS algorithm performs well. Furthermore, it can be seen that the LNS algorithm consistently yields good solutions, since the relative standard deviation is often less than 2% and never larger than 3.16%. The LNS algorithm is also quite fast, with low runtimes that are typically less than a minute for very large instances and typically a few seconds or less in small and mid-size instances. These runtimes are considerably smaller than the runtimes obtained by the mathematical model. The difference in runtime becomes greater as the size of the instances increases, e.g., for instance I_1 the LNS algorithm is only 214 times faster than the exact method, while for I_{14} the LNS algorithm is almost 70000 times faster. In some instances, the runtime is less consistent and it has a relatively large standard deviation, typically this occurs when the LNS algorithm solves larger instances.

7. Comparison with other services

In this section, the DRFS is compared to other feeder services. More specifically, the best way to operate the DRFS, obtained by the LNS algorithm, will be compared to operating an on-demand only service (ODOS) that serves the same demand. The ODOS uses predefined stops and stop assignment to reduce walking times and travel times of the passengers in the same way as the DRFS. In the ODOS, however, there are no mandatory stops so all stops are optional. The only restriction on the routing is that each bus needs to start at the first mandatory bus stop and end at the last mandatory stop. The ODOS is a service that has maximum flexibility (i.e., no mandatory stops). Like the DRFS, this service can be classified as an on-demand transportation service (ODTS).

The DRFS is also compared to traditional feeder services (TFS). In the TFS, the buses have

	Objective function value (s)					Runtime (s)			
Inst.	**LNS mean**	**LNS min**	**Relative LNS Std**	**Mean gap**	**Min gap**	**LNS mean**	**LNS min**	**LNS Std**	**Exact**
I_1	3156,7	3149,6	0,22%	0,40%	0,20%	0,02	0,02	0,00	4,3
I_2	2947,0	2932,7	0,85%	0,30%	0,00%	0,03	0,03	0,00	4,1
I_3	4898,1	4889,7	0,14%	0,30%	0,10%	0,03	0,03	0,00	27,6
I_4	4460,8	4447,2	0,23%	0,20%	0,00%	0,05	0,04	0,01	19,8
I_5	7312,2	7300,5	0,13%	0,20%	0,10%	0,04	0,04	0,00	74,2
I_6	6131,2	6117,4	0,18%	0,10%	0,00%	0,07	0,06	0,00	81,0
I_7	5938,1	5908,9	0,41%	0,50%	0,10%	0,09	0,09	0,00	57,7
I_8	7851,2	7838,5	0,05%	0,30%	0,20%	0,11	0,11	0,01	487,2
I_9	7300,0	7300,0	0,00%	0,30%	0,20%	0,17	0,15	0,02	309,7
I_{10}	9503,1	9496,1	0,07%	0,20%	0,10%	0,12	0,11	0,00	1822,2
I_{11}	8883,2	8801,8	2,01%	0,20%	0,10%	0,17	0,15	0,03	1730,9
I_{12}	9044,0	9033,3	0,35%	0,40%	0,20%	0,22	0,19	0,04	2314,5
I_{13}	9008,6	8949,4	0,56%	0,80%	0,10%	0,28	0,23	0,05	1449,7
I_{14}	12470,5	12370,4	1,03%	1,10%	0,10%	0,60	0,49	0,11	41832,5
I_{15}	17668,9	17495,6	0,91%			1,27	0,91	0,26	
I_{16}	16912,7	16810,6	0,74%			1,50	1,18	0,30	
I_{17}	15939,0	15907,1	0,28%			2,32	1,53	0,76	
I_{18}	15579,0	15520,6	0,16%			3,00	2,26	0,36	
I_{19}	26910,1	26443,7	1,47%			7,47	5,76	1,10	
I_{20}	37979,0	37251,3	1,09%			15,95	10,11	4,71	
I_{21}	47899,7	46740,0	1,87%			28,22	17,30	6,56	
I_{22}	63466,8	62350,5	1,60%			40,57	18,32	17,40	
I_{23}	114148,3	111971,0	1,18%			78,22	42,00	24,75	
I_{24}	146553,6	141483,0	1,99%			119,50	47,80	59,08	
I_{25}	51326,7	51288,8	0,05%			3,40	3,23	0,14	
I_{26}	49828,6	49311,3	0,71%			4,42	4,07	0,34	
I_{27}	47128,1	44539,5	2,90%			9,79	6,38	2,76	
I_{28}	45560,0	43848,9	3,16%			16,06	10,11	4,00	
I_{29}	46364,3	45192,1	2,83%			24,97	13,06	12,72	
I_{30}	48567,5	47762,9	0,93%			31,88	15,39	11,00	
I_{31}	47179,4	46805,9	0,59%			23,78	13,79	7,21	
I_{32}	47333,4	46950,1	0,79%			24,07	14,15	10,93	
I_{33}	47410,9	47158,1	0,49%			24,11	13,24	8,19	
I_{34}	47788,3	46938,6	1,18%			30,01	19,93	7,24	
I_{35}	47899,7	46740,0	1,87%			27,13	16,60	6,28	
I_{36}	47899,7	46740,0	1,87%			27,10	16,80	6,20	

Table 3: Results of the LNS algorithm

Service	Predetermined bus stop locations	Bus stop assignment	Fixed route	Fixed timetable
TFS	Yes	No	Yes	Yes
DRFS	Yes	Yes	Partially	No
ODOS	Yes	Yes	No	No

Table 4: Overview of the characteristics of the different services

fixed routes and timetables. Two different TFS are considered: one that visits all the stops (TFS1) and one that visits all mandatory stops and one randomly chosen, predetermined bus stop per cluster (TFS2). In this service, a number of passengers can make requests for transportation, however, the service will only assign passengers to stops of the fixed routes of the buses and to a bus with a predetermined timetable. Both services are not flexible at all and can be classified as traditional transportation services (TTS). An overview of the different characteristics of each service is shown in table 4.

7.1. Optimizing the operation of ODOS and TFS

In order to have a fair comparison, the operation of the TFS and the ODOS are optimized as well. The LNS algorithm is modified to obtain the most efficient operation of the ODOS for a given demand and is used to solve all instances described in Section 5. For both the DRFS and the ODOS, 10 runs are performed and the best run is considered.

For the TFS, first, the routing of the buses is optimized using a greedy algorithm to construct an initial solution and a 2-opt algorithm is used to improve upon it. Second, each passenger will be assigned to its closest stop included in the fixed bus route. Third, the timetables of the buses are determined by evenly spreading the departure times of the buses from the first mandatory bus stop over a certain time period, in accordance to the number of available buses. Then, the departure time of the first bus is determined based on the best case scenario, i.e. the arrival of the different buses coincides the most with the desired arrival times of the passengers at the hub. Lastly, the assignment of passengers to buses is optimized using a Mixed Integer Linear Program (MILP). The MILP minimizes the (absolute value of the) difference between the actual arrival time and the desired arrival time of the passengers. Furthermore, the MILP is subject to the same capacity and arrival constraints as the LNS algorithm. The MILP formulation can be found in Appendix B.

7.2. General comparison

The objective function values of all instances for all services (DRFS, TFS1, TFS2, ODOS) are summarized in Table 5. This table also indicates the difference in objective function value of TFS1, TFS2 and ODOS compared to DRFS. A negative difference corresponds to a better (lower) objective function value. It should be noted that the passenger requests have time windows and that the TFS services could not respect these time windows for some instances. This results in infeasible solutions, colored red in Table 5. The value that is displayed is the objective function value that the service yields when these constraints are removed.

From the results it is clear that the DRFS preforms much better than both versions of the TFS. The decrease in the objective is often more than 60% and at times over 100%. When the instance size increases, the improvement of DRFS with respect to TFS increases too. This means that the DRFS improves the TFS due to the customized timetables and routes. TFS2 seems to perform slightly better than TFS1 in most instances. TFS1 only outperforms TFS2 in instances with a small number of buses.

As expected, the fully flexible ODOS performs better when it comes to optimizing the service quality of the known demand. These results could be considered as a lower bound, when the DRFS would ignore the mandatory stops. This also means that the ODOS cannot serve any passengers without reservation, i.e., passengers that need a ride for transportation but do not make a request for it. Nevertheless, the difference in objective function value between ODOS and DRFS remains limited, mostly below 7%. This means that when DRFS (or ODOS) would be implemented in practice, this 7% operational gain should be traded off with (not) serving passengers without reservation.

7.3. Effect of instance parameters

In what follows, the number of optional stops per cluster, the number of requests, the number of buses and the bus capacity are modified in different experiments. There are four sets of experiments and in each set one of the previously mentioned parameters varies while the other parameters remain constant. Instances I_{15} to I_{36} have been designed in four subsets specifically for these experiments (see Table 2). Instances I_{15} to I_{18} have the number of optional stops per cluster as a variable, while the other instance parameters remain constant. In instances I_{19} to I_{24}, the number of requests changes, in instances I_{25} to I_{30} the number of buses changes, and in instances I_{31} to I_{36} the capacity of the buses changes. This is done to isolate the effect of each variable on the performance of the DRFS, the ODOS and the TFS.

7.3.1. Number of optional stops per cluster

Figure 7 shows the objective function values of DRFS, ODOS, TFS1 and TFS2. The number of optional stops per cluster is the only variable in these experiments. Evidently, the ODOS and the DRFS optimize their service quality further when a larger number of optional stops is considered. Even though it is not evident from the figure below, we noticed that the advantage of the ODOS over the DRFS is more pronounced when there are more optional stops. This is to be expected since a greater number of optional stops gives the service more flexibility, which can be utilized more efficiently in the fully flexible ODOS. The traditional services, however, behave the opposite way; when the number of optional stops increases, the performance of DRFS further improves compared to both TFS. TFS1 benefits from a smaller number of optional stops; less optional stops means that the buses need to travel less. The objective function value of TFS2 is unaffected because only a single optional stop is visited in each cluster.

7.3.2. Number of passenger requests

Figure 8 shows the impact of the number of passenger requests on the performance of the different services. It is easy to see that the objective function value of all services increases

Inst.	DRFS (s)	ODOS (s)	Diff	TFS1 (s)	Diff	TFS2 (s)	Diff
I_1	3149,6	3149,6	0,00%	3984,0	26,49%	4405,7	39,88%
I_2	2932,7	2932,7	0,00%	3860,4	31,63%	4186,9	42,77%
I_3	4889,7	4882,5	-0,15%	6160,2	25,98%	7091,8	45,04%
I_4	4447,2	4355,6	-2,06%	6046,6	35,96%	6839,8	53,80%
I_5	7300,5	7280,1	-0,28%	9510,6	30,27%	10312,8	41,26%
I_6	6117,4	6074,6	-0,70%	8023,2	31,15%	8622,3	40,94%
I_7	5908,9	5756,9	-2,57%	7830,7	32,52%	8226,7	39,23%
I_8	7838,5	7607,5	-2,95%	10130,1	29,24%	10971,1	39,96%
I_9	7300,0	6965,5	-4,58%	9981,4	36,73%	10592,6	45,10%
I_{10}	9496,1	9418,4	-0,82%	12709,2	33,84%	13661,5	43,87%
I_{11}	8801,8	8543,2	-2,94%	11562,1	31,36%	12281,6	39,54%
I_{12}	9033,3	8822,2	-2,34%	12270,6	35,84%	12281,6	35,96%
I_{13}	8949,4	8705,5	-2,73%	12841,6	43,49%	12825,3	43,31%
I_{14}	12370,4	11628,6	-6,00%	18244,5	47,49%	18737,6	51,47%
I_{15}	17495,6	16060,5	-8,20%	25076,4	43,33%	24483,6	39,94%
I_{16}	16810,6	15418,3	-8,28%	25296,9	50,48%	24483,6	45,64%
I_{17}	15907,1	14474,7	-9,00%	25254,8	58,76%	24483,6	53,92%
I_{18}	15520,1	14093,5	-9,19%	25726,5	65,76%	24483,6	57,75%
I_{19}	26443,7	22699,0	-14,16%	62031,5	134,58%	51531,9	94,87%
I_{20}	37251,3	33339,5	-10,50%	72250,9	93,96%	65404,8	75,58%
I_{21}	46740,0	43695,7	-6,51%	88124,5	88,54%	86474,2	85,01%
I_{22}	62350,5	59327,6	-4,85%	107896,0	73,05%	114345,0	83,39%
I_{23}	111971,0	107784,0	-3,74%	151458,0	35,27%	172296,0	53,88%
I_{24}	141483,0	141188,0	-0,21%	180732,0	27,74%	210854,0	49,03%
I_{25}	51288,8	50283,4	-1,96%	103441,0	101,68%	113079,0	120,48%
I_{26}	49311,3	48269,7	-2,11%	94311,4	91,26%	103285,0	109,46%
I_{27}	44539,5	42846,5	-3,80%	80078,8	79,79%	86396,7	93,98%
I_{28}	43848,9	41799,5	-4,67%	79963,5	82,36%	83625,3	90,71%
I_{29}	45192,1	42765,1	-5,37%	83485,1	84,73%	84490,8	86,96%
I_{30}	47762,9	44182,8	-7,50%	92417,2	93,49%	89438,8	87,26%
I_{31}	46805,9	43365,7	-7,35%	99469,8	112,52%	97819,5	108,99%
I_{32}	46950,1	43428,6	-7,50%	90280,8	92,29%	88630,5	88,78%
I_{33}	47158,1	43169,3	-8,46%	89814,1	90,45%	88163,8	86,95%
I_{34}	46938,6	43365,9	-7,61%	88124,5	87,74%	86474,2	84,23%
I_{35}	46740,0	43695,7	-6,51%	88124,5	88,54%	86474,2	85,01%
I_{36}	46740,0	43695,7	-6,51%	88124,5	88,54%	86474,2	85,01%

Table 5: Comparisons between DRFS, an on-demand only service and two traditional services

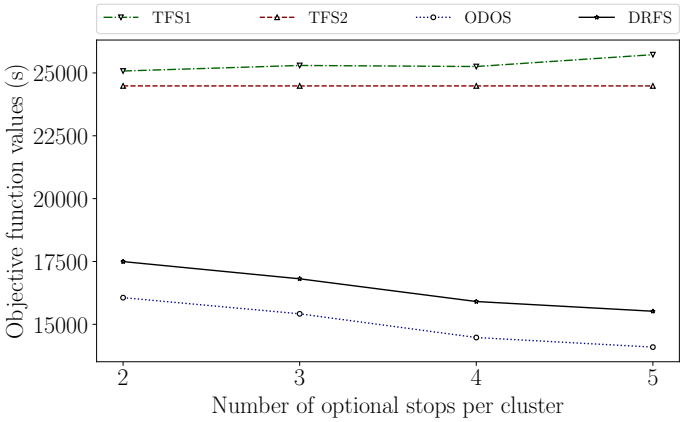


Figure 7: Effect of the number of optional stops per cluster

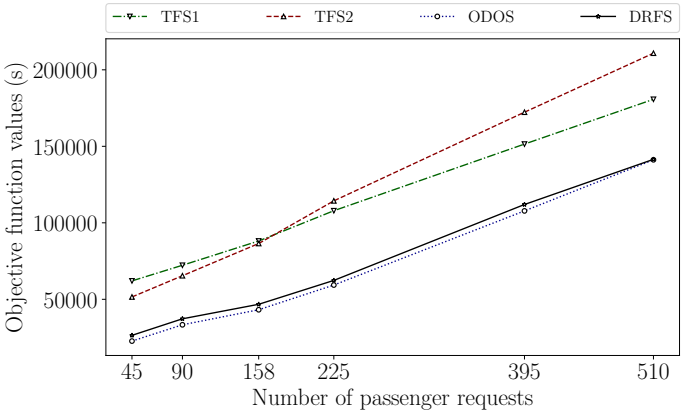


Figure 8: Effect of the number of passenger requests

with the number of passenger requests. TFS2 outperforms TFS1 when the number of passenger requests is low, while TFS1 performs better than TFS2 when the number of passenger requests is higher. This is likely due to the fact that the smaller travel times in TFS2 compensate for the larger walking times of the passengers. When more passengers request a ride, the walking times become more important and it is more beneficial to visit more stops so passengers walk less, as in TFS1, rather than visiting less optional stops, as in TFS2. The relative difference between the objective function values of DRFS and the objective function values of both the ODOS and the TFS become smaller when the number of requests is higher. The relative difference decreases from 9% to 0.2% for ODOS, from 135 % to 28% for TFS1 and from 95% to 49% for TFS2. This is expected; when there are more passenger request, the flexibility of the services has less added value because the demand is high enough to justify a rigid timetable (desired arrival times are more uniformly spread in time) and bus routing (visiting more stops makes more sense).

7.3.3. Number of buses

Figure 9 shows the results of the experiments where the number of buses is modified. Evidently, an increasing number of buses decreases the objective function values of the services, but too many buses make the objective function value increase again. Having more buses available means that the difference in desired and actual arrival time of the

passengers can be reduced. However, if there are too many buses, the added travel time of the buses becomes more important. The DRFS and the ODOS force every bus to operate in order to serve potential passengers that need transportation but have not made a request for it. This means that the travel time of all buses is added to the objective function, even if some buses have no passengers onboard. The ideal number of buses for TFS1 is lower than the ideal number buses for TFS2. This is due to the fact that, in TFS1, the travel-time of each bus is higher because more optional stops are visited. The objective function values of ODOS approach the values of DRFS when there are fewer buses; having less buses decreases the flexibility, which in turn decreases the advantage the ODOS has over the DRFS.

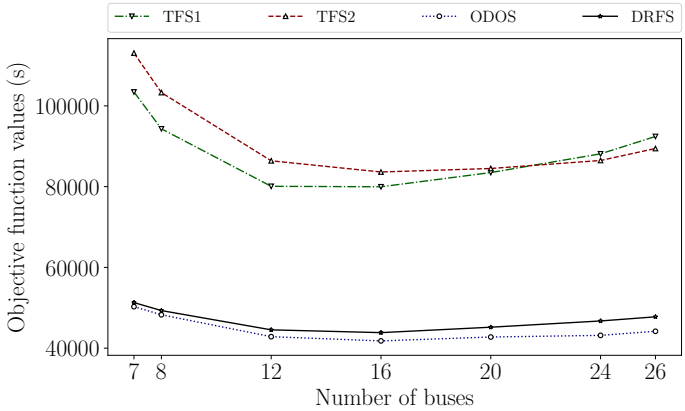


Figure 9: Effect of the number of buses

7.3.4. Bus capacity

Figure 10 shows the impact of the bus capacity. The capacity does not have a major effect on the ODOS and the DRFS. The behavior of the TFS shows that more capacity improves the TFS by a significant margin. More capacity means that the buses can transport more passengers at once, which means more passengers with similar desired arrival times are grouped, making the services more efficient. After a certain capacity, however, the TFS services stop improving because all possible efficient groupings of passengers have been realized already. ODOS and DRFS are essentially unaffected by the bus capacity because the difference in desired and actual arrival time is reduced by customizing the routes and the timetables and not necessarily by grouping a larger number of passengers.

8. Conclusion

In this research, a new type of feeder service, with both optional and mandatory bus stops, is considered: a Demand Responsive Feeder Service (DRFS). This feeder service incorporates positive characteristics of both traditional transport services (TTS) and on-demand transport services (ODTS). The DRFS has flexibility in selecting which of the clustered optional bus stops are visited, based on passenger requests. Furthermore, there is predictability in the mandatory bus stops, which need to be visited by each bus. Passengers can have a customized service by making requests online. However, if such requests are not made it is still possible to catch a bus in a mandatory bus stop. This will likely improve service quality.

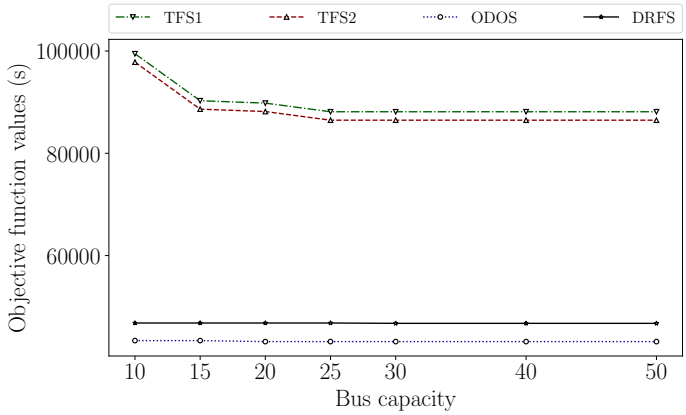


Figure 10: Effect of the number of bus capacity

In order to optimize the performance of the DRFS, a heuristic algorithm is developed. The algorithm follows a large neighborhood search (LNS) framework. In this framework, a part of the solution is destroyed and rebuilt several times. The destruction operator randomly destroys part of an existing solution. The reconstruction operator starts from the partially destroyed solution and rebuilds it. After the destroy-repair cycles, a 2-opt algorithm further improves the routing part of the solution. The parameters of the LNS algorithm are fine-tuned, using a statistical analysis, to obtain good results for any kind of instance. This approach yields high quality results in short runtimes for 36 different instances. Optimality gaps of typically less than 0.2% are found for 14 small and mid-size benchmark instances, for which optimal solutions are available, with very small runtimes of under a second. The 22 larger instances are typically solved within a minute.

This demand responsive feeder service (DRFS) is compared to a traditional feeder service (TFS), where routes and timetables are fixed, and to an on-demand only service (ODOS), where all bus stops are optional. The DRFS performs better than the traditional service in all cases. Often, there is more than 60% improvement in service quality. In some instances, the TFS is even unable to satisfy all the constraints of the service. The DRFS consequently needs less buses than the traditional services to satisfy the demand. The DRFS and the ODOS perform better in situations with relatively low demand for transportation, while the TFS performs relatively better when there is a high demand for transportation. Furthermore, the improvement in quality of both the DRFS and the ODOS, with respect to TFS, is greater when a smaller bus capacity and/or bus fleet is available. The ODOS improves the objective function value obtained by DRFS by 6 % on average. We consider this a small increase considering that this service does not serve any passengers without reservation, while such passengers can be served by the DRFS at the mandatory stops. We conclude that in many circumstances the DRFS could be an attractive, cost efficient and passenger friendly alternative.

Further research could focus on making the DRFS even more flexible, by optimizing the service in real-time. This will make the problem considerably more complex because it needs to modify routes and timetables to accommodate new requests, while still satisfying all constraints. The LNS algorithm presented in this work can be used as a starting point for the solution method of such a service. However, the LNS algorithm should be modified

significantly to be able to work with real-time requests. Quantifying how well the DRFS can serve the passengers that do not make a request for transportation is an interesting next step as well. The service could also be further improved towards passengers, by also considering earliest departure times of the passengers instead of only the latest arrival times. Another improvement is to replace the mandatory stops by stops with a guaranteed minimal frequency, allowing to still service the passengers without reservation, but in a more flexible way.

Acknowledgments

This project was supported by the FWO (Research Foundation Flanders) project G.0759.19N.

References

[1] M. Steriu, Statistics brief - Local public transport trends in the European Union (2016). URL https://www.uitp.org/statistics-brief-public-transport-in-the-EU

[2] J. Hine, F. Mitchell, Better for everyone? Travel experiences and transport exclusion, Urban Studies 38 (2) (2001) 319–332.

[3] M. J. Alonso-González, T. Liu, O. Cats, N. Van Oort, 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) (2018) 879–889.

[4] D. Pisinger, S. Ropke, Large neighborhood search, International Series in Operations Research and Management Science 272 (2019) 99–127.

[5] B. D. Galarza Montenegro, K. Sörensen, P. Vansteenwegen, A demand-responsive feeder system with mandatory and optional, clustered bus-stops, under review, available as working paper (2020).
URL https://EconPapers.repec.org/RePEc:ant:wpaper:2020006

[6] C. L. Martins, M. V. Pato, Search strategies for the feeder bus network design problem, European Journal of Operational Research 106 (1998).

[7] F. Ciaffi, E. Cipriani, M. Petrelli, Feeder Bus Network Design Problem: a New Metaheuristic Procedure and Real Size Applications, Procedia - Social and Behavioral Sciences 54 (2012) 798–807.

[8] J. J. Lin, H. I. Wong, Optimization of a feeder-bus route design by using a multiobjective programming approach, Transportation Planning and Technology 37 (5) (2014) 430–449.

[9] A. S. Mohaymany, A. Gholami, Multimodal feeder network design problem: Ant colony optimization approach, Journal of Transportation Engineering 136 (4) (2010) 323–331.

[10] M. Zheng, R. Zhou, S. Liu, F. Liu, X. Guo, Route Design Model of Multiple Feeder Bus Service Based on Existing Bus Lines, Journal of Advanced Transportation 2020 (2020).

[11] P. Shrivastava, M. O'Mahony, Design of Feeder Route Network Using Combined Genetic Algorithm and Specialized Repair Heuristic, Journal of Public Transportation 10 (2) (2007) 109–133.

[12] P. Shrivastava, M. O'Mahony, A model for development of optimized feeder routes and coordinated schedules — A genetic algorithms approach, Transport Policy 13 (2006) 413–425.

[13] M. Mistretta, J. A. Goodwill, R. Gregg, C. DeAnnuntis, Best Practices in Transit Service Planning. Final Report No. BD549-38, Tech. rep., Center for Urban Transportation Research for the Florida Department of Transportation (2009).

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

[15] N. Agatz, A. Erera, M. Savelsbergh, X. Wang, Optimization for dynamic ride-sharing: A review, European Journal of Operational Research 223 (2) (2012) 295–303.

[16] W. A. Ellegood, S. Solomon, J. North, J. F. Campbell, School bus routing problem: Contemporary trends and research directions, Omega (United Kingdom) 95 (2020) 102056.

[17] B. Sun, M. Wei, S. Zhu, Optimal design of demand-responsive feeder transit services with passengers' multiple time windows and satisfaction, Future Internet 10 (3) (2018).

[18] M. W. P. Savelsbergh, M. Sol, The General Pickup and Delivery Problem, Transportation Science 29 (1) (1995) 17–29.

[19] J. F. Cordeau, G. Laporte, The Dial-a-Ride Problem (DARP): Variants, modeling issues and algorithms, 4or 1 (2) (2003) 89–101.

[20] L. Quadrifoglio, M. M. Dessouky, F. Ordóñez, Mobility allowance shuttle transit (MAST) services: MIP formulation and strengthening with logic constraints, European Journal of Operational Research 185 (2) (2008) 481–494.

[21] X. Lu, J. Yu, X. Yang, S. Pan, N. Zou, Flexible feeder transit route design to enhance service accessibility in urban area, Journal of Advanced Transportation 50 (4) (2015) 507–521.

[22] F. Qiu, W. Li, A. Haghani, An exploration of the demand limit for flex-route as feeder transit services: a case study in Salt Lake City, Public Transport 7 (2) (2015) 259–276.

[23] X. Li, Optimal design of demand-responsive feeder transit services, Ph.D. thesis, Texas A&M University (2009).

[24] A. Ceder, Integrated smart feeder/shuttle transit service: simulation of new routing strategies, Journal of Advanced Transportation 47 (June 2010) (2013) 595–618.

[25] X. Li, L. Quadrifoglio, Feeder transit services: Choosing between fixed and demand responsive policy, Transportation Research Part C: Emerging Technologies 18 (5) (2010) 770–780.

[26] DeLijn, DIAL-A-BUS A SOLUTION FOR YOUR JOURNEY? (2021).
URL `https://www.delijn.be/en/belbus/?vertaling=true`

[27] L. Dell'Olio, A. Ibeas, P. Cecin, The quality of service desired by public transport users, Transport Policy 18 (1) (2011) 217–227.

[28] D. A. Hensher, P. Stopher, P. Bullock, Service quality - developing a service quality index in the provision of commercial bus contracts, Transportation Research Part A: Policy and Practice 37 (6) (2003) 499–517.

[29] K. Sörensen, F. Glover, Metaheuristics (2013).

[30] V. Kayvanfar, M. Zandieh, E. Teymourian, An intelligent water drop algorithm to identical parallel machine scheduling with controllable processing times: a just-in-time approach, Computational and Applied Mathematics 36 (1) (2017) 159–184.

[31] S. Asta, E. Özcan, A. J. Parkes, CHAMP: Creating heuristics via many parameters for online bin packing, Expert Systems with Applications 63 (2016) 208–221.

[32] H. Sun, C. Y. Yang, C. W. Lin, J. S. Pan, V. Snasel, A. Abraham, Genetic and Evolutionary Computing: Proceeding of the Eighth International Conference on Genetic and Evolutionary Computing, October 18–20, 2014, Nanchang, China, Advances in Intelligent Systems and Computing 329 (August 2014) (2015).

[33] S. Voß, A. Fink, C. Duin, Looking ahead with the pilot method, Annals of Operations Research 136 (1) (2005) 285–302.

[34] A. Goel, Vehicle scheduling and routing with drivers' working hours, Transportation Science 43 (1) (2009) 17–26.

[35] D. Pisinger, S. Ropke, A general heuristic for vehicle routing problems, Computers and Operations Research 34 (8) (2007) 2403–2435.

[36] L. F. Muller, An Adaptive Large Neighborhood Search Algorithm for the Multi-mode Resource-Constrained Project Scheduling Problem, European J. of Industrial Engineering (EJIE) 11 (6) (2010) 1–10.

[37] Y. Dodge, An introduction to L1-norm based statistical data analysis, Computational Statistics and Data Analysis 5 (4) (1987) 239–253.

[38] F. Campelo, F. Takahashi, Sample size estimation for power and accuracy in the experimental comparison of algorithms, Journal of Heuristics 25 (2) (2018) 305–338.

Appendix A. Results for the LNS algorithm with pilot method

	Objective function value (s)			Runtime (s)		
Inst.	**Mean**	**Min**	**Relative std**	**Mean**	**Min**	**Std**
I_1	3167.3	3158.8	0.22%	0.024	0.023	0.002
I_2	2957.2	2932.4	1.11%	0.040	0.035	0.005
I_3	4904.5	4898.8	0.14%	0.038	0.037	0.002
I_4	4462.1	4446.9	0.16%	0.063	0.057	0.013
I_5	7309.3	7300.5	0.12%	0.056	0.055	0.002
I_6	6127.3	6117.0	0.16%	0.102	0.085	0.025
I_7	5934.3	5918.1	0.28%	0.119	0.114	0.007
I_8	7850.5	7842.0	0.09%	0.141	0.127	0.015
I_9	7303.5	7303.5	0.00%	0.210	0.173	0.049
I_{10}	9505.2	9499.6	0.07%	0.153	0.143	0.008
I_{11}	8867.7	8805.3	1.78%	0.211	0.188	0.046
I_{12}	9089.5	9036.8	1.74%	0.274	0.227	0.053
I_{13}	9041.3	8952.9	0.57%	0.323	0.288	0.027
I_{14}	12542.5	12402.7	1.18%	0.616	0.576	0.028
I_{15}	17663.0	17534.8	0.43%	1.460	1.101	0.234
I_{16}	16974.0	16886.0	0.36%	1.940	1.221	0.692
I_{17}	16009.9	15956.4	0.58%	2.449	1.688	0.548
I_{18}	15616.0	15551.0	0.34%	3.116	2.457	0.470
I_{19}	26894.3	26474.9	1.21%	10.173	7.067	2.764
I_{20}	37773.6	37105.7	1.27%	20.552	15.247	4.036
I_{21}	47632.8	46734.6	1.57%	31.258	16.237	7.054
I_{22}	62954.4	62252.5	1.24%	56.205	25.938	16.831
I_{23}	114987.3	111862.0	1.45%	109.787	60.824	35.973
I_{24}	147335.6	142274.0	2.11%	121.540	83.253	33.906
I_{25}	51325.2	51304.5	0.03%	4.463	4.409	0.030
I_{26}	49907.6	49341.1	0.79%	5.481	5.142	0.567
I_{27}	47002.9	44494.2	2.70%	14.006	10.052	2.783
I_{28}	45681.6	44044.6	2.80%	17.046	11.705	3.822
I_{29}	46714.8	45009.8	2.12%	26.173	16.702	8.393
I_{30}	48558.4	47863.3	1.16%	43.483	18.308	18.302
I_{31}	47387.8	46918.4	1.24%	37.392	17.996	16.887
I_{32}	47237.6	46794.4	0.56%	37.516	18.219	14.239
I_{33}	47629.8	47004.8	1.26%	39.489	22.280	15.150
I_{34}	47563.8	46877.3	0.99%	43.009	21.378	16.358
I_{35}	47632.8	46734.6	1.57%	32.721	17.590	7.368
I_{36}	47632.8	46734.6	1.57%	32.678	17.033	7.396

Table A.6: Results of the LNS algorithm when a pilot method is used for bus stop assignment

Appendix B. MILP for the traditional system

Decision Variables	
y_{pb}	0-1 assignment variables which assume the value of 1 if passenger $p \in P$ is assigned to bus $b \in B$
la_p	The amount of time passenger $p \in P$ is late
ea_p	The amount of time passenger $p \in P$ is early
Parameters	
a_b	Arrival time of bus $b \in B$ at the destination
d_{late}	Maximum value for arriving later than the desired arrival time of any passenger
d_{early}	Maximum value for arriving earlier than the desired arrival time of any passenger
C	Capacity of the buses

Table B.7: Variables and parameters of the MILP

Minimize:

$$z = \sum_{p \in P} (l_p + ea_p)$$

S.t.

$$\sum_{b \in B} y_{pb} = 1 \qquad \forall\ p \in P \tag{B.1}$$

$$\sum_{p \in P} y_{pb} \leq C \qquad \forall\ b \in B \tag{B.2}$$

$$la_p \leq d_{late} \qquad \forall\ p \in P \tag{B.3}$$

$$ea_p \leq d_{early} \qquad \forall\ p \in P \tag{B.4}$$

$$dat_p - \sum_{b \in B} y_{pb} a_b + la_p - ea_p = 0 \qquad \forall\ p \in P \tag{B.5}$$