

# Solving the On-Demand Bus Routing Problem: A Bus Stop Assignment and Transportation Problem

Jorge Mortes<sup>a,b</sup>, Martin Cousineau<sup>b</sup>, Fabien Lehuédé<sup>a,\*</sup>, Jorge E. Mendoza<sup>b</sup>, María I. Restrepo<sup>a</sup>

<sup>a</sup>IMT Atlantique, LS2N, UMR CNRS 6004, F-44307, Nantes, France

<sup>b</sup>HEC Montréal, 3000 chemin de la Côte-Sainte-Catherine, Montréal, H3T 2A7, Canada

---

## Abstract

This article investigates a static on-demand transportation problem in which users are picked up and dropped off at existing bus stops. Specifically, the selection of bus stops for each request, along with the bus routes, is simultaneously determined by the booking system using an optimization algorithm. We focus on service quality by lexicographically minimizing passenger travel time (including both walking time and time spent on the bus) and total route length. We introduce a new matheuristic algorithm based on small and large neighborhood search, incorporating state-of-the-art operators and a set covering component. This algorithm outperforms previous approaches by reducing the user ride time by over 24% and shows good performance on related problems benchmarks. The algorithm is tested on a new set of instances, based on real data from New York City, which we propose as a future benchmark. Additionally, we discuss implementation details for decision-makers and practitioners.

**Keywords:** on-demand transportation problem, public bus system, stop assignment, small and large neighborhood search, set covering

---






## 1. Introduction

Fixed-line buses are known for their efficiency in densely populated areas, where buses are often full during peak hours and operational costs can be offset by passenger fares. However, they become underutilized when demand decreases (Estrada et al., 2021). As a result, operators may reduce bus frequency, discontinue certain lines, or increase fares, ultimately leading to a decline in service quality (Brooks, 2023). In urban areas, the lack of nighttime service often forces people to walk long distances to bus stops and wait for buses, which may be unsafe (Plyushteva & Boussauw, 2020). In rural areas, passengers may spend significant time on nearly empty buses that make multiple stops without picking up or dropping off anyone. As stated by Estrada et al. (2021), these

---

\*Corresponding author  
E-mail address: fabien.lehuede@imt-atlantique.fr

challenges can be addressed through on-demand transportation systems, typically with door-to-door services, which allow bus routes to be adjusted daily based on demand. This approach enables the design of routes that avoid empty stops, resulting in lower operational costs, reduced pollution, and improved service quality. Two examples of existing on-demand systems are Via Transportation (2024) and Padam Mobility (2024).

Operating door-to-door on-demand transportation systems brings up several challenges. First, the presence of many one-way streets in cities can result in significant detours for picking up and dropping off passengers, leading to increased operating costs and pollution (Stiglic et al., 2018). Additionally, buses cannot stop anywhere on the road, as doing so could block traffic. These issues can be addressed by assigning each request an existing pick-up and drop-off point, such as a bus stop. To the best of our knowledge, Melis & Sørensen (2022) were the first to investigate the problem of simultaneously determining vehicle routes and selecting pick-up and drop-off locations per user from a limited but convenient set of bus stops, to the benefit of both the operator and the users. The authors referred to this problem as the on-demand bus routing problem (ODBRP). In their study, they suggest utilizing stops already existing in the city's transit network, allowing buses to stop in designated areas, and avoiding the need to invest in new infrastructure. While this approach results in many advantages, it also has some inconveniences. For instance, it requires passengers to walk to the stops and adds complexity to the planning process, as it involves not only creating routes but also assigning passengers to appropriate bus stops. Figure 1 shows a simplified example of the ODBRP. Figure 1a shows an instance with two customers, each with three potential pick-up and drop-off locations. The icons  and  represent the origin and destination for User 1, while the icons  and  represent the origin and destination for User 2. The icon  denotes the depot. Figure 1b shows a solution to this instance, where specific pick-up and drop-off locations have been selected for each request, and a route has been created that starts and ends at the depot.

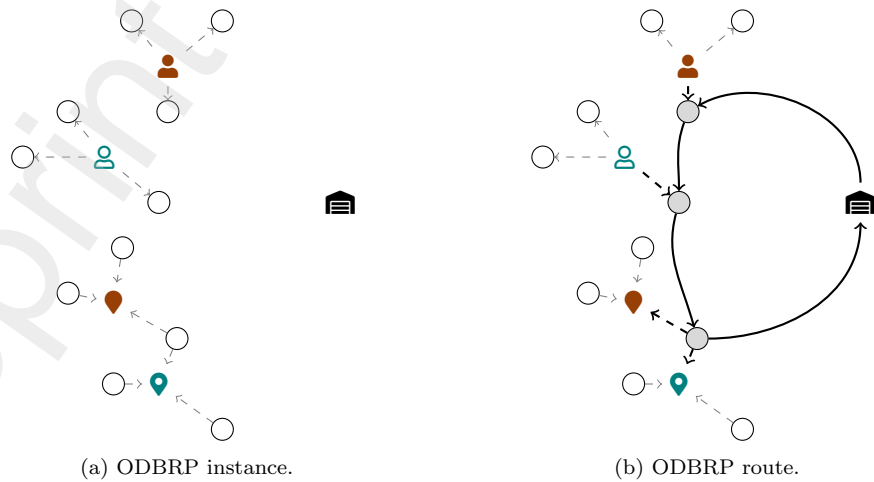


Figure 1: Example of an ODBRP route with two users and 3 potential bus stops per origin and destination.

In this paper, we revisit the ODBRP and present a new algorithm to solve it. Our contributions are fivefold. First, we introduce a new set of realistic instances for benchmarking algorithms developed for the ODBRP, as no publicly available instances for this problem existed previously. These instances are based on actual taxi trips from New York City, real bus stop locations, and realistic distance and travel time matrices. Second, we introduce a new matheuristic for solving the ODBRP, which utilizes small and large neighborhood search (SLNS) combined with a set covering (SCP) component. The SLNS component of the algorithm is based on an adaptation of state-of-the-art destroy operators combined with efficient repair operators, enabling faster solution improvements compared to conventional LNS algorithms. Third, we examine the conditions for accepting a large neighborhood iteration in SLNS—an aspect not previously studied—highlighting its impact on solution quality in highly constrained problems. Fourth, we show that existing stop assignment operators in the ODBRP literature exhibit limited performance, and that our method improves up to 24% the users’ time spent inside the bus upon previous results. Fifth, we provide managerial insights that are valuable to decision-makers addressing the ODBRP: We compare different objective functions and demonstrate that minimizing passenger travel time (i.e., a user-focused objective) leads to different solutions than minimizing total route length (i.e., an operator-focused objective); we evaluate the ODBRP in comparison to a door-to-door policy, showing significant reductions in fleet size and total route length; and we assess the impact of adding more potential pick-up and drop-off bus stops, finding no significant difference when more than five stops are included.

The remainder of the manuscript is organized as follows: Section 2 provides a literature review on the ODBRP and related problems; Section 3 formally defines the ODBRP and presents a mixed-integer linear formulation; Section 4 details the proposed matheuristic; Section 5 introduces the new instance sets; and Section 6 discusses the computational experiments and managerial insights. Finally, Section 7 concludes the manuscript with a summary and some important remarks.

## 2. Literature review

According to the survey by Vansteenwegen et al. (2022), on-demand or demand-responsive bus systems have been extensively studied over the years and can be classified based on their level of responsiveness and flexibility. The least responsive on-demand systems are those that utilize *static* bus routes, which complete their route planning before the service begins and are therefore unable to make adjustments afterward. An example of a static bus system is presented by Stiglic et al. (2018), where the authors utilize an on-demand system to connect passengers to public transit options, such as train stations in suburban areas. More recently, Jodeau et al. (2024) presented a static on-demand transportation system for sparsely populated areas, capable of operating across both road

and rail networks. A higher level of responsiveness is achieved with *dynamic offline* bus routes, where routes can be adjusted dynamically during operation but only for trips that have not yet started. In Li & Quadrioglio (2010), the authors develop an on-demand bus service that connects residential areas with a public transportation network, comparing its performance to that of a traditional fixed-route bus system. Similarly, Huang et al. (2020) propose a system in which customers can dynamically book trips up to one hour in advance. During this booking phase, the operator optimizes routes to determine fare offers for customers. In the final hour before operation, the operator optimizes the routes based on accepted requests. At the highest level of responsiveness, *dynamic online*, on-demand bus system can modify its routes at any point during the service period. An example of this type of system is provided by Archetti et al. (2018), where the authors use a simulator to compare an on-demand door-to-door bus service with more traditional options like fixed bus lines or walking. Users choose the option with the shortest travel time, and if none of the systems meet the user's requirements, they opt to use a private car. Additionally, Horn (2002) introduces software designed to simulate an on-demand bus service (among other transportation modes) using the LITRES-2 modeling system and a local search heuristic. The examples mentioned above are also classified as *fully flexible* on-demand systems, where the routes are determined solely by the demand. However, the literature has also explored *semi-flexible* systems, in which the routes must visit some mandatory locations, and optional locations can be added to the route based on the demand. An example of a semi-flexible system is provided by Pei et al. (2019), where the authors consider a set of mandatory fixed locations and another set of locations that are only visited if sufficient demand exists. Similarly, Crainic et al. (2012) introduce a system in which optional bus stops may be added between required stops if a request arises at an optional stop and can be accommodated. For a more comprehensive overview of on-demand bus systems, we refer the readers to the survey conducted by Vansteenwegen et al. (2022).

Many static fully flexible on-demand bus routing systems, such as the ODBRP, have been modeled as pickup and delivery problem with time windows (PDPTW) (Dumas et al., 1991) or as the dial-a-ride problem (DARP) (Cordeau & Laporte, 2003), as the three problems involve precedence (i.e., all requests must be picked up before being dropped off), coupling (i.e., a request should be picked up and dropped off by the same bus), and time window constraints (i.e., pick-ups and drop-offs must be done within the specified time windows). However, in the PDPTW and DARP, customers typically provide separate time windows for pickup and delivery, while in the ODBRP, a single time window is specified for the entire trip. The PDPTW has been successfully solved for instances up to 100 requests using exact methods, with the branch-and-price-and-cut approaches from Ropke & Cordeau (2009) and Baldacci et al. (2011) being particularly effective. For larger instances, researchers have developed several metaheuristic algorithms, with the LNS algorithms of

Curtois et al. (2018), Sartori & Buriol (2020), and Christiaens & Vanden Berghe (2020) considered state-of-the-art. Those three works apply their algorithms to the Li & Lim (2001) benchmark with up to 500 requests, letting the algorithm run for up to 60 minutes in the case of Curtois et al. (2018), up to 226 minutes in the case of Christiaens & Vanden Berghe (2020). Sartori & Buriol (2020) aimed to match the previous two works' computing times. Additionally, Sartori & Buriol (2020) introduced a new benchmark with up to 2,500 requests, allowing the algorithm to run on these instances for up to 60 minutes. In addition to the mentioned constraints, the DARP also has constraints on maximum route duration and maximum trip time per user. In terms of exact solution methods, the most widely used algorithms for the DARP are based on the branch-and-bound framework (Cordeau, 2006; Braekers et al., 2014). The branch-and-cut-and-price method developed by Gschwind & Irnich (2015) is considered state-of-the-art, having solved instances with up to 96 customers to optimality in under 15 minutes. For larger instances, numerous metaheuristics have been proposed, with the most popular being variable neighborhood search (Parragh et al., 2010) and hybrid methods (Gschwind & Drexl, 2019). The latter is considered the most efficient algorithm for solving the DARP due to its constant feasibility checks and the hybridization of LNS and dynamic programming.

The bus stop assignment in the ODBRP relates to the generalized vehicle routing problem (GVRP) (Ghiani & Improta, 2000). In this problem, each customer is associated with a cluster of nodes, but only one node within each cluster needs to be visited to service the customer. Similarly, in the ODBRP, the potential pick-up and drop-off bus stops can be viewed as clusters, with each request containing two sets of clusters (one for pick-up and another for drop-off), and only one node from each cluster needs to be visited. Numerous exact algorithms have been developed to solve the GVRP, with branch-and-cut, branch-and-price, and branch-and-cut-and-price being the most commonly used approaches (Bektaş et al., 2011; Bulhões et al., 2018; Reihaneh & Ghoniem, 2018). Among metaheuristic algorithms for the GVRP, ant colony optimization, genetic algorithms, and variable neighborhood search have been particularly popular (Bautista et al., 2008; Bulhões et al., 2018; Sadati et al., 2022). For a more comprehensive overview of the GVRP, we direct the readers to the survey by Jolfaei et al. (2023).

As mentioned above, the ODBRP presented in Melis & Sörensen (2022) falls into the categories of static and fully flexible on-demand bus systems. However, unlike previous studies that typically employ door-to-door transportation or assign requests to the nearest bus stops, the ODBRP offers multiple bus stops for each customer's pick-up and drop-off. This results in more flexibility to optimize routes at the cost of an increase in the problem's complexity. The authors solve a problem where all requests must be served, and the objective function is to minimize the total user ride time (i.e., the sum of user ride times), or URT. They use a large neighborhood search (LNS) algorithm paired with

a local search operator. Their LNS incorporates two destroy-repair operators: one aimed at minimizing URT and the other at restoring feasibility. The authors created artificial grid-based instances to benchmark their approach against a conventional fixed-line bus system and to assess their LNS performance against LocalSolver (Hexaly, 2024).

To complement the work in Melis & Sörensen (2022), we propose a set of instances based on New York City taxi trip data, incorporating map-based distances and travel times. These instances serve as a benchmark for evaluating the performance of our algorithm. Furthermore, our SLNS incorporates state-of-the-art operators, combining small and large destruction operators that increase the number of iterations per run, along with an SCP component that aids the algorithm when it stalls. While Melis & Sörensen (2022) focused on minimizing URT to enhance service quality, we opted for a lexicographic objective function that prioritizes minimizing passenger travel time (PTT) as the primary objective and total route length as the secondary objective. PTT, as defined in Melis & Sörensen (2022), includes both URT and users' walking time. We selected this objective function because walking time is naturally linked to service quality. Additionally, we found that many equivalent solutions in terms of PTT or URT exist. Therefore, selecting the solution with the shortest route length among them is logical, as it helps reduce costs.

### 3. Problem description

We consider a known set of transportation requests, denoted as  $R$ . Each request  $r \in R$  is defined by an origin, a destination, a time window  $[e_r, l_r]$ , a number of passengers, a set of potential pick-up bus stops  $P_r$ , and a set of potential drop-off bus stops  $D_r$ . We define  $P = \bigcup_{r \in R} P_r$  and  $D = \bigcup_{r \in R} D_r$  as the sets of all potential pick-up and drop-off stops, respectively. Note that if a bus stop can be used by multiple requests (also referred to as a bus stop covering several requests), it is replicated in sets  $P$  or  $D$  for each request listing it as a potential pick-up or drop-off stop. Consequently,  $\bigcap_{r \in R} P_r = \bigcap_{r \in R} D_r = \emptyset$ , and only one bus stop from  $P_r$  and  $D_r$  must be visited for each  $r \in R$ . Note also that sets  $P_r$  and  $D_r$  can be constructed based on various criteria, such as proximity within walking distance or user preferences. Lastly, the time window  $[e_r, l_r]$  represents the earliest time the passengers are willing to be picked up and the latest time they are willing to be dropped off.

The ODBRP is formulated on a complete directed graph  $G = (N, A)$ , where  $N$  represents the set of nodes and  $A = \{(i, j) : i, j \in N\}$  represents the set of directed arcs. Let  $0^+$  and  $0^-$  denote two copies of the depot, defining the start and end of the bus routes. The set  $N$  includes all bus stops, along with nodes  $0^+$  and  $0^-$  (i.e.,  $N = P \cup D \cup \{0^+, 0^-\}$ ). Let  $K$  represent a set of homogeneous vehicles having a passenger capacity of  $Q$ . Each node  $i \in N$  is associated with a passenger load  $q_i$ , where  $q_{0^-} = q_{0^+} = 0$  and  $q_i = -q_{i'}$ ,  $\forall i \in P_r, i' \in D_r, r \in R$ . Additionally, each node is associated with a time window  $[a_i, b_i]$  and a walking time  $w_i$ , where  $w_i$  represents the walking time either

from the origin of the request to a pick-up stop (if  $i \in P$ ) or from a drop-off stop to the destination (if  $i \in D$ ). The time windows on the nodes  $[a_i, b_i]$  are calculated using the requests' time windows  $[e_r, l_r]$  and the walking times  $w_i$ . We refer the reader to Section 5 for more details on the nodes' time window computation and tightening. Let  $d_{ij}$  and  $t_{ij}$  be the length of the arc  $(i, j) \in A$  and the time needed to traverse it. Note that  $d_{ij}$  and  $t_{ij}$  are 0 if nodes  $i$  and  $j$  represent the same physical bus stop. Lastly, let  $\sigma_r$  be the service time spent picking up or dropping off request  $r \in R$ . These service times,  $\sigma_r$ , are incorporated into the travel times. Given these definitions, the ODBRP involves constructing a set of bus routes that lexicographically minimizes PTT and total route length, accommodates all requests, assigns a single pick-up and drop-off stop for each request within the same vehicle, respects vehicle capacity limits, serves requests within the time windows, and does not exceed the fleet size. For clarity, we have included a table in Appendix A that contains all the notation used throughout the document.

Although our solution methodology is heuristic, we provide a mathematical formulation of the ODBRP to formally define the problem under consideration. Let  $x_{ij}^k$  be a binary variable that equals 1 if and only if vehicle  $k \in K$  travels directly from node  $i$  to node  $j$ . For a given request  $r$ , the variables  $x_{ij}^k$  are responsible for assigning the pick-up and drop-off bus stops from the sets  $P_r$  and  $D_r$  and incorporating them into the route of vehicle  $k$ . Let  $A_i^k$  and  $L_i^k \geq 0$  represent the arrival time and departure load at node  $i \in N$ , respectively, for vehicle  $k \in K$ . Finally, let  $T_r \geq 0$  denote the travel time for request  $r \in R$ . The following is the three-index formulation for the ODBRP:

$$\text{lexmin} \quad \sum_{r \in R} T_r, \quad \sum_{k \in K} \sum_{i \in N} \sum_{j \in N} d_{ij} x_{ij}^k \quad (1)$$

$$\text{s.t.} \quad \sum_{k \in K} \sum_{i \in P_r} \sum_{j \in N} x_{ij}^k = 1 \quad \forall r \in R \quad (2)$$

$$\sum_{i \in P_r} \sum_{j \in N} x_{ij}^k - \sum_{i \in D_r} \sum_{j \in N} x_{ij}^k = 0 \quad \forall r \in R, k \in K \quad (3)$$

$$\sum_{j \in N} x_{0^+j}^k = 1 \quad \forall k \in K \quad (4)$$

$$\sum_{i \in N} x_{i,0^-}^k = 1 \quad \forall k \in K \quad (5)$$

$$\sum_{j \in N} x_{ji}^k - \sum_{j \in N} x_{ij}^k = 0 \quad \forall i \in P \cup D, k \in K \quad (6)$$

$$A_j^k \geq A_i^k + t_{ij} - M_{ij}^k (1 - x_{ij}^k) \quad \forall i \in N, j \in N, k \in K \quad (7)$$

$$L_j^k \geq L_i^k + q_j - W_{ij}^k (1 - x_{ij}^k) \quad \forall i \in N, j \in N, k \in K \quad (8)$$

$$a_i \leq A_i^k \leq b_i \quad \forall i \in N, k \in K \quad (9)$$

$$L_i^k \leq Q \quad \forall i \in P, k \in K \quad (10)$$

$$A_{i'}^k \geq A_i^k + t_{ii'} \quad \forall r \in R, i \in P_r, i' \in D_r, k \in K \quad (11)$$

$$T_r \geq A_{i'}^k - A_i^k + w_{i'} + w_i - \left[ 1 - \sum_{j \in N} x_{ji}^k \right] H_{ii'} \quad \forall r \in R, i \in P_r, i' \in D_r, k \in K \quad (12)$$

$$x_{ij}^k \in \{0, 1\} \quad \forall i \in N, j \in N, k \in K \quad (13)$$

$$A_i^k, L_i^k \geq 0 \quad \forall i \in N, k \in K \quad (14)$$

$$T_r \geq 0 \quad \forall r \in R \quad (15)$$

Equation (1) lexicographically minimizes the PTT and total route length. Constraints (2) and (3) make sure that all the requests are picked up and dropped off once at one of their potential bus stops, and that each request is picked up and dropped off by the same bus. Constraints (4)-(5) ensure that every route starts and finishes at the depot. Constraints (6) ensure flow conservation at each node. Constraints (7), (8) together with (9), (10) impose time windows and capacity constraints, where  $M_{ij}^k = \max\{0, b_i + t_{ij} - a_j\}$  and  $W_{ij}^k = \min\{Q, Q + q_i\}$ . Note that constraints (10) apply only to pick-up nodes, since any capacity violation at a drop-off node necessarily implies a capacity violation at a pick-up node. Constraints (11) ensure that each pick-up is done before the corresponding drop-off. Constraints (12) define the requests' travel time, where  $H_{ii'} = b_{i'} - a_i + w_{i'} + w_i$ . Lastly, constraints (13)-(15) define the variables' domain. The model presented above could potentially be improved by employing alternative modeling techniques, such as tracking passengers using flow variables on the arcs rather than on the nodes. However, as previously stated, the primary objective of this formulation is to provide a clear and readable formal definition of the problem.

#### 4. Solution method

Since the ODBRP is a generalization of the PDPTW, which is known to be  $\mathcal{NP}$ -hard, it is unlikely that an exact algorithm will scale well for real-world ODBRP instances. Furthermore, we implemented Model (1)-(15) in a commercial solver, and after 3 hours of computation time, the optimality gap was still larger than 50% for instances with 50 customers, even when using the best solution found by the matheuristic, which will be explained below, as a warm-up solution. Consequently, to solve the ODBRP, we propose a matheuristic that combines small and large neighborhood search (SLNS) with a SCP component.

SLNS is a recent adaptation of the well-known LNS meta-heuristic. As initially proposed by Shaw (1998), LNS is a meta-heuristic algorithm where a solution is partially destroyed and recreated over many iterations to improve its quality. Several years later, Ropke & Pisinger (2006a) popularized LNS by introducing the use of multiple heuristics to destroy and repair solutions (referred to as operators), along with an adaptive layer to guide the selection of these operators across different iterations. However, the adaptive layer was found to have limited impact in several studies (Turkeš et al., 2021). More recently, Christiaens & Vanden Berghe (2020) introduced a variation of LNS that employs small removals and greedy insertions (list heuristics). This approach allows for a larger



number of iterations compared to earlier LNS metaheuristics and achieves high-quality results across many vehicle routing applications. Following this idea, Le Colleter et al. (2023) introduced SLNS. This method differs from traditional LNS in its use of destroy and repair operators: it uses small, fast operators to intensify search, and large destroy and repair operators to diversify it. For further reading on LNS, we recommend the surveys by Pisinger & Ropke (2019) and Mara et al. (2022).

The combination of heuristic methods with exact optimization components is well-established in the vehicle routing field. An early work by Foster & Ryan (1976) explores this approach for the vehicle routing problem, utilizing a constructive heuristic to generate routes alongside a set partitioning problem (SPP) to select the best routes created by the heuristic. More specifically, the integration of LNS and SPP has been successfully implemented to solve complex vehicle routing problems, such as the vehicle routing problem with cross-docking (Grangier et al., 2017) or the fleet size and mix dial-a-ride problem with reconfigurable vehicle capacity (Tellez et al., 2018). These algorithms fall under the category of what Archetti & Speranza (2014) denote restricted master heuristics, where heuristics function as a column generation method (i.e., creating routes), and an SPP or SCP is employed to select the optimal columns. Within this category, we observe two types of algorithms: those that generate routes for a period and solve an SPP or SCP at the end to improve the best solution found by the LNS, as in Gschwind & Drexel (2019), and those that solve the SPP or SCP after a certain number of LNS iterations to better guide the LNS, as in Dumez et al. (2021a). For a more detailed discussion on restricted master heuristics and other matheuristics applied to vehicle routing problems, readers are encouraged to consult the survey by Archetti & Speranza (2014).

The remainder of this section is organized as follows: Section 4.1 presents the algorithm's framework. Sections 4.2 and 4.3 provide specific implementation details of the SLNS and SCP for the ODBRP. Finally, Section 4.4 introduces two bus stop selection policies.

#### 4.1. Matheuristic framework

Before detailing the main pipeline, we introduce the following definitions. A feasible solution  $Sol$  for the ODBRP is a set of routes that satisfies constraints (2)-(15). As is common in LNS algorithms for routing problems, the algorithm works with a specific class of infeasible solutions known as *partial solutions*. A partial solution does not include all requests but still respects the other constraints. We define the request bank  $B(Sol)$  of a solution  $Sol$  as the set of unserved requests of a solution  $Sol$ . Consequently,  $B(Sol) = \emptyset$  if  $Sol$  represents a feasible solution. To penalize partial solutions, we precompute the back-and-forth route with the longest possible PTT for a single request (i.e., a bus traveling directly from the depot to pick up the request, delivering the passenger to their bus stop,

and then returning to the depot). Then, for each request in  $B(Sol)$ , we add this precomputed value to the partial solution's PTT and apply a multiplication factor. Additionally, given two feasible routes  $\rho^1$  and  $\rho^2$ , we say that  $\rho^1$  dominates  $\rho^2$  if both routes serve the same customers (though not necessarily using the same bus stops) and  $\rho^1$  has a lower PTT than  $\rho^2$ .

As in Le Colleter et al. (2023), we define a set of large destroy operators  $\Sigma_{large}^-$ , a set of small destroy operators  $\Sigma_{small}^-$ , and a set of repair operators  $\Sigma^+$  (details about the operators in each set are provided in Section 4.2). To construct the initial solution, we select a repair operator  $o_{init}^+ \in \Sigma^+$ . Each set of destroy operators has predefined bounds on the number of bus stops to be removed:  $[\delta_{small}^{min}, \delta_{small}^{max}]$  for the small destroy operators and  $[\delta_{large}^{min}, \delta_{large}^{max}]$  for the large destroy operators. Additionally, we define  $\omega_{small}$  as the maximum number of iterations allowed between small and large destructions without improvement, and  $\omega_{worse}$  as the minimum number of iterations without improvement before triggering the SCP.

Every time the SLNS generates a new solution, we add the non-dominated routes to the so-called pool of routes  $\Omega$ , checking dominance with the routes already in the set. We define  $\beta$  as the minimum number of routes in  $\Omega$  to launch the SCP. To prevent excessive time consumption by the SCP, we define  $\omega_{time}$  as the time limit for each SCP run, meaning that the SCP may not be solved to optimality at every launch. Lastly, as explained in Dumez et al. (2021b), we dynamically adjust  $\beta$  in the following way: If the SCP has been solved to optimality twice in a row, we increase  $\beta$ . Otherwise, if the SCP cannot prove optimality twice in a row, we decrease  $\beta$ . We define  $\tau \in [0, 1]$  as the ratio that increases or decreases  $\beta$ . This dynamic adjustment of  $\beta$  helps prevent initiating the SCP with an excessive number of routes, which could result in wasted time in solving a complex SCP, or with too few routes, which could result in not improving the best solution.

Algorithm 1 summarizes the main pipeline of the matheuristic. In this pseudo-code,  $Sol^{best}$  is the best solution found by the algorithm,  $Sol^{curr}$  is the current solution, and  $Sol^{new}$  is a copy of the current solution that is destroyed and repaired. The variables  $it_{worse}$  and  $it_{small}$  count the number of iterations without solution improvement and using small destructions, respectively. Additionally,  $opt$  and  $opt_{prev}$  keep track of when the SCP has been solved to optimality. After the algorithm's initialization in Line 1, Line 2 builds a first current solution  $Sol^{curr}$  using  $o_{init}^+$ . Line 3 saves the first solution as the best-found solution and adds its routes to  $\Omega$ . The main loop of the matheuristic is found in lines 4 to 39, which finishes once the termination criterion is met (i.e., a time limit is reached). Line 5 verifies whether the current iteration should be large or small. Based on this, a destroy operator and a corresponding destroy size are randomly selected (lines 6 to 10). Line 11 randomly selects a repair operator. Line 12 applies the destroy and repair operators to  $Sol^{curr}$  to create  $Sol^{new}$ . Line 13 updates  $\Omega$  with the non-dominated routes

---

**Algorithm 1** SLNS with SCP

---

**Input:** parameters:  $[\delta_{small}^{min}, \delta_{small}^{max}]$ ,  $[\delta_{large}^{min}, \delta_{large}^{max}]$ ,  $\omega_{small}$ ,  $\omega_{worse}$ ,  $\omega_{time}$ ,  $\beta$ ,  $\tau$ ; operators:  $\Sigma_{small}^{-}$ ,  $\Sigma_{large}^{-}$ ,  $\Sigma^{+}$ ,  $o_{init}^{+}$

**Output:**  $Sol^{best}$

```
1:  $\Omega \leftarrow \emptyset$ ,  $it_{worse} \leftarrow 0$ ,  $it_{small} \leftarrow 0$ ,  $opt \leftarrow 0$ ,  $opt_{prev} \leftarrow 0$ 
2: Build an initial solution  $Sol^{curr}$  with the operator  $o_{init}^{+}$ 
3: Add routes of  $Sol^{curr}$  to  $\Omega$ ,  $Sol^{best} \leftarrow Sol^{curr}$ 
4: while time limit not reached do
5:   if  $it_{small} < \omega_{small}$  then
6:     Select a destroy operator  $o^{-} \in \Sigma_{small}^{-}$  and a destroy size  $\delta \in [\delta_{small}^{min}, \delta_{small}^{max}]$ 
7:      $it_{small} \leftarrow it_{small} + 1$ 
8:   else
9:     Select a destroy operator  $o^{-} \in \Sigma_{large}^{-}$  and a destroy size  $\delta \in [\delta_{large}^{min}, \delta_{large}^{max}]$ 
10:  end if
11:  Select a repair operator  $o^{+} \in \Sigma^{+}$ 
12:   $Sol^{new} \leftarrow o^{+}(o^{-}(Sol^{curr}, \delta))$ 
13:  Update  $\Omega$  with the non-dominated routes of  $Sol^{new}$ 
14:  if  $Sol^{new}$  is better than  $Sol^{best}$  then
15:     $Sol^{best} \leftarrow Sol^{new}$ ,  $\Omega \leftarrow \emptyset$ ,  $it_{worse} \leftarrow 0$ ,  $it_{small} \leftarrow 0$ 
16:  else
17:     $it_{worse} \leftarrow it_{worse} + 1$ 
18:  end if
19:  if  $Sol^{new}$  meets the record-to-record or ( $it_{small} = \omega_{small}$  and  $Sol^{new}$  feasible) then
20:     $Sol^{curr} \leftarrow Sol^{new}$ 
21:    if  $it_{small} = \omega_{small}$  then
22:       $it_{small} \leftarrow 0$ 
23:    end if
24:  end if
25:  if  $it_{worse} \geq \omega_{worse}$  and  $|\Omega| \geq \beta$  then
26:    Improve  $Sol^{best}$  with the SCP for  $\omega_{time}$  seconds
27:    if SCP solution is optimal then
28:       $opt \leftarrow 1$ 
29:    end if
30:     $Sol^{curr} \leftarrow Sol^{best}$ ,  $\Omega \leftarrow \emptyset$ ,  $it_{worse} \leftarrow 0$ 
31:    if  $opt = 1$  and  $opt_{prev} = 1$  then
32:       $\beta \leftarrow (1 + \tau)\beta$ 
33:    end if
34:    if  $opt = 0$  and  $opt_{prev} = 0$  then
35:       $\beta \leftarrow (1 - \tau)\beta$ 
36:    end if
37:     $opt_{prev} \leftarrow opt$ ,  $opt \leftarrow 0$ 
38:  end if
39: end while
```

---

of  $Sol^{new}$ , and replaces any dominated routes in  $\Omega$  with the new ones. Line 14 checks if  $Sol^{new}$  should become the best-found solution  $Sol^{best}$  by checking the following criteria: 1)  $Sol^{new}$  has fewer requests in the request bank than  $Sol^{best}$ ; 2)  $Sol^{new}$  has the same number

of requests in the request bank but has a lower PTT; or 3)  $Sol^{new}$  has the same number of requests in the request bank, same PTT, and  $Sol^{new}$  has a shorter total route length. If so,  $\Omega$  is emptied and the iteration counters  $it_{small}$  and  $it_{worse}$  are reset. Otherwise, we increment  $it_{worse}$ . Line 19 checks if  $Sol^{new}$  should become the current solution  $Sol^{curr}$ . This happens if  $Sol^{new}$  meets the record-to-record criterion (i.e., if  $Sol^{new}$ 's PTT is within  $\alpha\%$  of  $Sol^{best}$ 's PTT (Dueck, 1993)) or  $Sol^{new}$  is a feasible solution with a large destroy operator in this iteration. In the second case, the value of  $it_{small}$  is reset. We note that, unlike Le Colleter et al. (2023), if a large destroy operator does not find a feasible solution, the algorithm keeps doing large destruction iterations until a feasible solution is found (see Section 6.1.1 for more details). Line 25 checks if the SCP should be launched (i.e.,  $Sol^{best}$  has not changed for at least  $\omega_{worse}$  iterations and the pool has at least  $\beta$  routes). If so, the SCP tries to improve  $Sol^{best}$ ,  $\Omega$  is emptied, and  $it_{worse}$  is reset in Lines 26 to 30. Lastly, lines 31 to 37 are devoted to the dynamic adaptation of  $\beta$ .

From a PTT perspective, an efficient route would involve serving requests consecutively, ensuring that the bus serves only one request at any given time. However, this approach often results in long, but compact routes, where adding additional customers would risk violating the time windows of those already on the route. As a result, constructing an initial feasible solution can be challenging, particularly when the bus fleet is limited. To address this issue and ensure that the PTT minimization begins with a feasible solution, we allocate 10% of the algorithm's time budget to minimizing total route length. As discussed in Section 6.3.1, while minimizing total route length leads to routes with significantly higher PTT, it facilitates the generation of feasible solutions. We refer to this as a *warm-up phase*.

#### 4.2. SLNS

In this section, we describe the operators implemented in the SLNS to solve the ODBRP and give some implementation hints. We first describe the destroy operators:

- **Random destroy:** Introduced in Ropke & Pisinger (2006a), this operator chooses  $\delta$  requests at random and removes them from the existing solution.
- **Historical node-pair destroy:** Introduced in Ropke & Pisinger (2006b) as neighbor graph removal and renamed in Pisinger & Ropke (2007), this operator removes  $\delta$  requests from the current solution that were "better placed" in earlier solutions. To do so, the operator keeps track of a score for each arc  $(i, j) \in A$ . The operator initializes these scores to infinity and updates the score of an arc  $(i, j) \in A$  whenever a solution using this arc has a lower PTT than the previous score, replacing it with the new solution's PTT. Afterward, the operator determines each request's cost by summing the arcs' scores connected to its pickup and delivery nodes. Lastly,

$\delta$  requests are selected using a sampling method biased towards requests with the highest costs.

- **String removal and split string removal:** Introduced in Christiaens & Vanden Berghe (2020), the string removal operator randomly selects a seed node  $v^*$  and removes several sequences of nodes close to  $v^*$  from different routes. In the case of split string removal, the sequence of nodes to remove can be split in two sub-strings, leaving nodes in between the sub-strings. In the ODBRP case, when a pick-up (resp. drop-off) node is selected to be removed, the related drop-off (resp. pick-up) node is also deleted.

For the random and historical node-pair operators, when a request is selected to be removed, the pick-up and drop-off nodes related to the request are removed from the route and the request is placed in the request bank. As the destruction size of the String Removal and Split String Removal operators is intentionally kept small in Christiaens & Vanden Berghe (2020), the set  $\Sigma_{small}^-$  includes all three operators, while the set  $\Sigma_{large}^-$  comprises only the Random and Historical Node-Pair Destroy operators. However, during small destroy iterations, the destruction size for the Random and Historical Node-Pair Destroy operators is reduced. Lastly, since  $\delta_{small}^{max}$  and  $\delta_{large}^{max}$  depend on the number of requests, we define  $\Delta_{small}$  and  $\Delta_{large}$  as upper bounds on the number of removed requests for small and large destroy operators, respectively. This prevents excessively large destruction sizes when  $R$  is large, which would otherwise lead to increased diversification and slow down the algorithm. All destroy operators have a uniform probability of being selected.

All the implemented repair operators follow the idea of fast insertion heuristics, also called list heuristics (Christiaens & Vanden Berghe, 2020). In this type of repair operator, a list with all the non-inserted requests is made and, following a specific criterion, the list is sorted. Lastly, a greedy insertion algorithm is used to insert all the requests following the order in the list. We use the following criteria to sort the lists:

- **Time window width:** As commonly used in VRP problems with time windows, inserting first the requests with the tightest time windows increases the chances of finding feasible solutions. We use the request's time window  $[e_r, l_r]$  width.
- **Pick-up and drop-off times:** We use two operators based on the pick-up and drop-off times: The first one sorts the customers by pick-up time (increasing) and the second one sorts the customers by drop-off time (decreasing).
- **Passengers per request:** In the same line as the time window width, inserting first requests with a larger number of customers increases the chances of finding feasible solutions.

- **Distance from the depot:** We use two operators using the request's distance from the origin location to the depot: The first operator sorts the requests in ascending order, while the second sorts them in descending order.
- **Random:** Sort the customers at random.

As this algorithm spends a lot of time checking insertion feasibility (i.e., all the combinations of pick-ups and drop-offs), we adapted the forward time slacks proposed in Savelsbergh (1992) to make them work with pick-up and drop-off nodes. We have also implemented blinks at insertions, proposed by Christiaens & Vanden Berghe (2020). Blinks at insertions mean that every time a combination of pick-up and drop-off is checked for insertion, it can be skipped with a certain probability  $\Pi$ . This adds a probability of deviating from the greedy insertion. All repair operators have a uniform probability of being selected and we choose the increasing distance from the depot list heuristic as the initial repair operator.

#### 4.3. Set covering

As mentioned above, using a SCP formulation on top of a metaheuristic can be seen as a heuristic column generation where the metaheuristic generates columns and the SCP chooses the best column combination. In the specific case of the ODBRP, the SLNS creates routes (i.e., columns) at each iteration and, whenever the algorithm gets stuck, it solves the SCP formulation to assemble the generated routes and improve the current solution. Additionally, the solutions found by the SCP lead to diversification as the SCP selects unexplored combinations of routes found by the SLNS. Since all routes considered in the SCP are generated by the SLNS, the assignment of bus stops is inherently incorporated when selecting the best set of routes from  $\Omega$ .

Let  $T_\rho$  be the PTT of route  $\rho \in \Omega$ . Let  $a_{r\rho} \in \{0, 1\}$  be a constant indicating if the request  $r \in R$  is served by route  $\rho \in \Omega$ . Let  $\lambda_\rho$  be a binary variable that takes the value 1 if and only if route  $\rho \in \Omega$  is used in the solution. We state the following SCP formulation for the ODBRP:

$$\text{minimize} \quad \sum_{\rho \in \Omega} T_\rho \lambda_\rho \quad (16)$$

$$\text{s.t.} \quad \sum_{\rho \in \Omega} a_{r\rho} \lambda_\rho \geq 1 \quad \forall r \in R \quad (17)$$

$$\sum_{\rho \in \Omega} \lambda_\rho \leq |K| \quad (18)$$

$$\lambda_\rho \in \{0, 1\} \quad \forall \rho \in \Omega \quad (19)$$

Equation (16) minimizes the PTT of the routes. Constraints (17) ensure that every request is covered at least once, and constraint (18) bounds the maximum number of routes by the fleet's size. Lastly, constraints (19) define the variables' domain.

We considered working with a SPP formulation; however, as it has been mentioned in past studies (Costa et al., 2019), if the triangular inequality holds for travel times or distance matrices, using a SCP formulation might lead to a faster solution convergence, as the dual variables associated with the set covering constraints are positive instead of free. However, the model might find solutions that serve customers in two or more routes. In that case, any request served in two or more different routes can be removed from the extra routes following a simple heuristic rule and this further decreases the objective function. In preliminary experiments, we observed that using a SCP formulation helps at the beginning of the algorithm to find solutions with a better value for the objective function than the solutions found with the SPP formulation.

#### 4.4. Bus stop selection policy

Several methods for selecting appropriate bus stops can be considered, with the simplest being the selection of the bus stop nearest to the requests' origins and destinations. In this work we propose two different bus selection policies, namely *sampled* policy and *cheapest insertion* (CI) policy.

The sampled policy divides a single ODBRP instance into multiple sub-instances, or samples. To generate a sample from an ODBRP instance, we randomly select one pick-up and one drop-off bus stop per request, disregarding all other bus stops in the instance. Figure 2 illustrates an example of an ODBRP instance with two users and three potential bus stops per origin and destination, along with a sample derived from the instance.

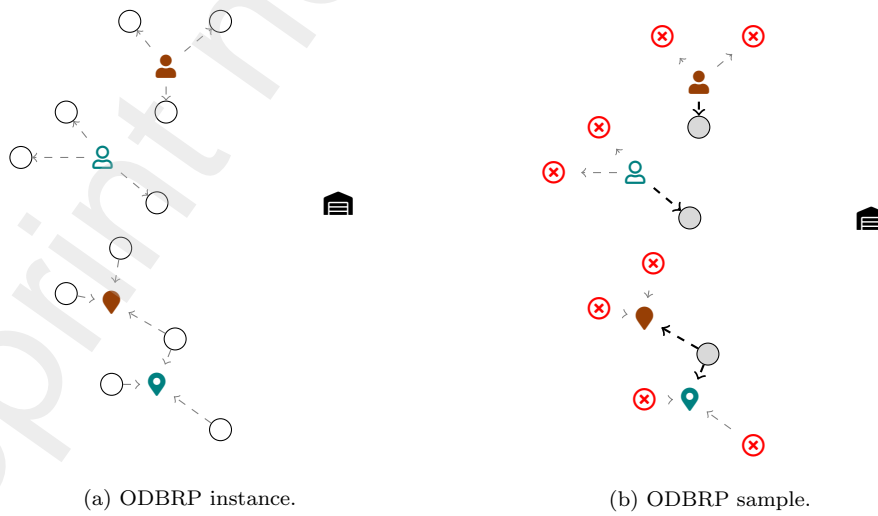


Figure 2: Example of an ODBRP instance and a sample obtained from the instance.

Solving a sample is simpler than the original instance, as the problem is reduced to the

routing component (i.e., a PDPTW instance). Therefore, several samples of one instance may be solved in the same amount of time as the original instance. Consequently, the sampled bus stops selection policy consists of solving several samples of the same instance and, then, selecting the best solution from all the samples. This implies that a PDPTW algorithm with a sampled policy could be used to solve the ODBRP. To further increase the number of solved samples, we modified Algorithm 1 to solve them in parallel. The parallelization of this algorithm is rather simple: as the samples of one instance can be considered independent from each other, we solve the different samples in parallel at the same time while collecting in a route pool the routes belonging to the best 100 solutions of each sample; once all samples have been solved, we solve the SCP (16)-(19) using the route pool and letting the algorithm assemble routes from different samples. To complement this section, an experiment demonstrating that our algorithm can efficiently solve PDPTW instances is provided in Appendix B.

Although the sampling strategy reduces complexity by converting ODBRP instances into PDPTW instances—allowing a PDPTW solver to generate ODBRP solutions—a sufficiently large number of samples must be considered to ensure a thorough exploration of the solution space, particularly as the instance size increases. Therefore, we propose the CI policy, a more computationally intensive bus stop assignment policy for the ODBRP. The main idea is rather simple: For each non-inserted request  $r$ , the insertion heuristic tries all possible combinations of stops in  $P_r$  and  $D_r$  in all possible pick-up and drop-off positions of the different routes in a cheapest insertion fashion. However, since the walking speed is lower than the driving speed, we prioritize exploring the bus stops in  $P_r$  and  $D_r$  in order of proximity, starting with those closest to the origin or destination and progressing to those further away.

## 5. Instances description

To generate the instances used in this study, we utilize two publicly available datasets from the city of New York. The first is the New York City Taxi Trip dataset (New York City Taxi and Limousine Commission, 2024), which provides detailed information about taxi trips recorded over previous years. The second is the New York City Bus Stops dataset (GIS Lab, 2019), containing the bus stops locations for MTA NYC Transit bus routes, as compiled by the GIS Lab at Newman Library, Baruch CUNY, in December 2019. We restrict both datasets to the Manhattan area. Figure 3 depicts the taxi zones in Manhattan, with all MTA NYC bus stops indicated by blue dots.

Before describing the instance generation process, we define the instance parameters. Each request is limited to a maximum of 10 bus stops, with up to 5 stops for pick-up and 5 for drop-off. Additionally, the maximum walk time per request is set at 10 minutes: 5 minutes walking to the pick-up stop and 5 minutes from the drop-off stop. Therefore,



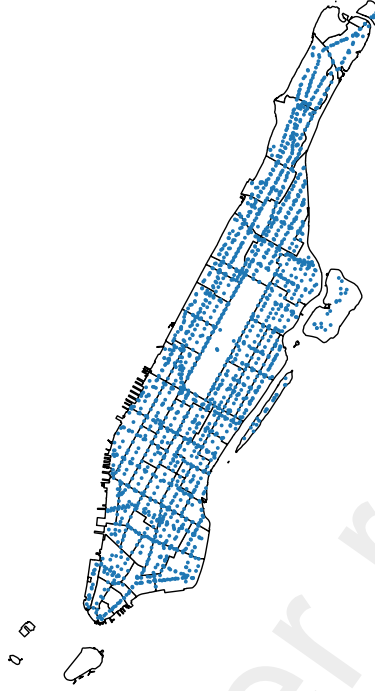


Figure 3: Manhattan map divided into taxi zones and with MTA NYC bus stops.



for each request, we select the 5 closest bus stops within a 5-minute walking radius of the origin and the destination. To simulate the low-density conditions that are favorable for on-demand shared transportation, we sample taxi trips occurring between midnight and 6 AM on weekdays. The time horizon for the buses is established between 11:30 PM and 7 AM, allowing sufficient time for buses to reach early requests and return to the depot. The depot corresponds to an MTA NYC bus depot<sup>1</sup>. We use the original taxi trip pick-up and drop-off times, subtracting 20 minutes from the pick-up time and adding 20 minutes to the drop-off time to simulate when each request is willing to be picked up and dropped off  $[e_r, l_r]$ . The number of passengers per request is randomly drawn from a discrete distribution with support  $[1, 2, 3, 4, 5]$  and probabilities  $[0.7, 0.13, 0.07, 0.06, 0.04]$ . Each request is assumed to have a service time  $\sigma_r = 1$  minute and a maximum bus capacity of nine passengers, as smaller vehicles are preferred for on-demand transportation in low-demand areas (Estrada et al., 2021).

Due to data protection regulations, the taxi trip dataset does not provide the exact origin and destination coordinates for each request; instead, it provides the corresponding origin and destination taxi zones. To address this, we randomly sample coordinates within the specified taxi zones to determine precise origin and destination locations. Any sampled request where the origin and destination are less than a 10-minute walk apart

<sup>1</sup>Located at coordinates (40.788202965040774, -73.94955970193467).

(i.e., the maximum allowed walking time) is discarded, as we assume these users could walk directly to their destination. For each remaining request, we identify the five nearest bus stops within a 5-minute walking radius of both the origin and destination. A request is also discarded if it has no bus stops within this 5-minute radius at either the origin or destination (e.g., a request located within the Central Park taxi zone).

Once all requests have been sampled and the corresponding bus stops identified, we calculate realistic distances and travel times using the openrouteservice backend (openrouteservice, 2024). Finally, we round down the distance and travel time matrices and enforce the triangular inequality.

As the final step, we calculate the time windows for each request at the bus stops  $[a_i, b_i], \forall i \in P \cup D$  as follows: First, we account for the walking time to determine the earliest possible arrival time at the pick-up bus stop and the latest possible arrival time at the drop-off bus stop. Specifically, the earliest time a bus can pick up request  $r$  at node  $i$  is calculated as  $a_i = e_r + w_i$ , and the latest time it can drop off request  $r$  at node  $i$  is  $b_i = l_r - w_i$ . Then, using these calculations, along with the distances between pick-up and drop-off bus stops, we determine for each request the latest possible departure time from each pick-up stop that allows the bus to reach all of its drop-off stops on time, as well as the earliest possible arrival time at each drop-off stop from all of its pick-up stops. Specifically, the latest time a bus can pick up request  $r$  at node  $i$  is calculated as  $b_i = \max_{j \in D_r} \{b_j - d_{ij}\}$ , and the earliest time it can drop off request  $r$  at node  $i$  is  $a_i = \min_{j \in P_r} \{a_j + d_{ij}\}$ . Figure 4 illustrates an example of this calculation. In the example, the icons  and  indicate the request's origin and destination, respectively. Dashed and solid arrows represent walking and driving paths, with the time required to traverse each arc noted on the arcs.

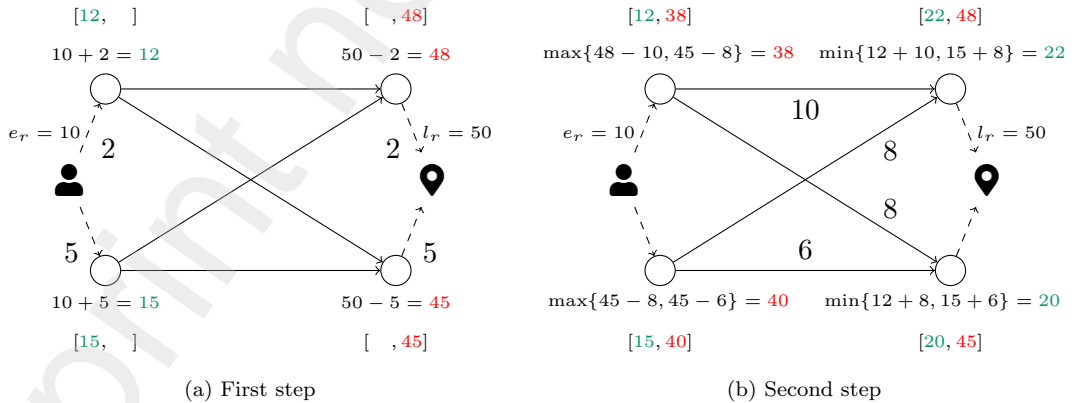


Figure 4: Bus stop time windows calculation example.

Following this procedure, we create 40 instances split into four 10-instance groups containing 50, 100, 250, and 500 requests (up to 500, 1,000, 2,500, and 5,000 bus stops respectively) and we name them M50, M100, M250, and M500. Figure 5 presents an instance from the M100 set, where the origins of requests and pick-up bus stops are

marked respectively with green crosses and gray dots in Figure 5a, and destinations and drop-off bus stops with red x's and gray dots in Figure 5b.

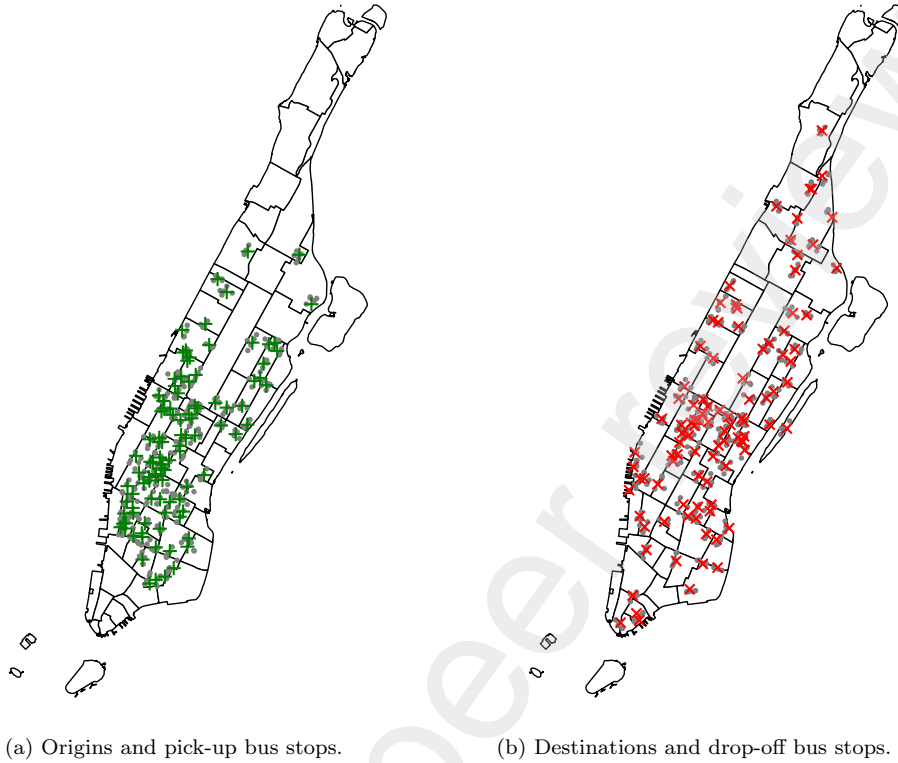


Figure 5: Instance example on the M100 set.

To determine a reasonable fleet size for the instances, we solved each instance using the best version of our algorithms (as described in Section 6.1), setting the total route length as the objective function and imposing the time limit specified in Section 6. Assuming that bus operators cannot hire and fire drivers based on daily demand fluctuations and would prefer to maintain a consistent number of drivers, we selected the largest fleet size obtained across all instances of each group. Accordingly, the chosen fleet sizes are 5, 8, 15, and 26 buses for the M50, M100, M250, and M500 sets, respectively.

Finally, an analysis of the instances revealed that request arrivals exhibit a peak of over 45% of request arrivals within the first hour, followed by a second peak of almost 20% in the last hour, with the remaining 35% spread over the four hours between these peaks. Serving all customers in these two peak hours, and particularly between midnight and 1 a.m., makes the instances challenging to solve due to the limited availability of vehicles.

All instances, along with the best solutions and the lower bounds reported in Section 6, are available on our GitHub repository<sup>2</sup>.

<sup>2</sup>Note that the repository is currently private. It will be made public as the manuscript progresses through the review process. [https://github.com/Jormoral/ODBRP\\_instances](https://github.com/Jormoral/ODBRP_instances)

## 6. Computational experiments

The algorithms presented in this work are implemented in C++ and executed on an Intel E5-2683 v4 Broadwell CPU @ 2.1GHz. The exact components are solved using the commercial solver CPLEX 22.1.1.0, operating on a single thread. The parameters for the matheuristic have been tuned using the IRACE package (López-Ibáñez et al., 2016), with the exception of those related to the string and split string removal destroy operators (Christiaens & Vanden Berghe, 2020), for which we used the recommended parameters for the PDPTW. Table 1 lists the algorithm’s parameters and their corresponding values. Each instance is solved 10 times, with runtime stop criteria of 15, 45, 90, and 180 minutes for instance sizes of 50, 100, 250, and 500, respectively. In the sampled policy, we created 25 samples for each run, and solved them in parallel over 5 threads, allocating one-fifth of the total time specified above to each sample.

Table 1: Algorithm’s parameters and their values.

Parameter	Value
$\Pi$	1%
$\delta_{small}^{min}$	$\max \{ \lfloor 0.01 \times  R  \rfloor, 1 \}$ requests
$\delta_{small}^{max}$	$\min \{ \lfloor 0.1 \times  R  \rfloor, \Delta_{small} \}$ requests
$\Delta_{small}$	5 requests
$\delta_{large}^{min}$	$\max \{ \lfloor 0.15 \times  R  \rfloor, 1 \}$ requests
$\delta_{large}^{max}$	$\min \{ \lfloor 0.2 \times  R  \rfloor, \Delta_{large} \}$ requests
$\Delta_{large}$	50 requests
$\omega_{small}$	6,000 iterations
$\omega_{worse}$	10,000 iterations
$\alpha$	1%
$\omega_{time}$	15 seconds
$\tau$	60%
$\beta$	2,000 routes

For each instance, we compare our algorithm to a lower bound, as optimal solutions could not be obtained. We calculate this lower bound on the assumption that there are enough vehicles to serve all requests without overlapping requests in the same bus (e.g., one vehicle per request in the worst-case scenario). For each request, we determine a lower bound by selecting the bus stops that minimizes PTT for a direct trip. The sum of these individual lower bounds constitutes the total lower bound. Figure 6a illustrates an example of this calculation. In the example, the same notations as in Figure 4 is used. By choosing the two top bus stops, a minimum PTT of 14 minutes is achieved.

Additionally, we compute a URT lower bound for some experiments where we minimize URT. We do this by applying the same method explained above for calculating the PTT lower bound, but focusing solely on the request’s time spent inside the bus. Figure 6b continues from the earlier example, showing that selecting the two bottom bus stops results

in a minimum URT of 6 minutes.

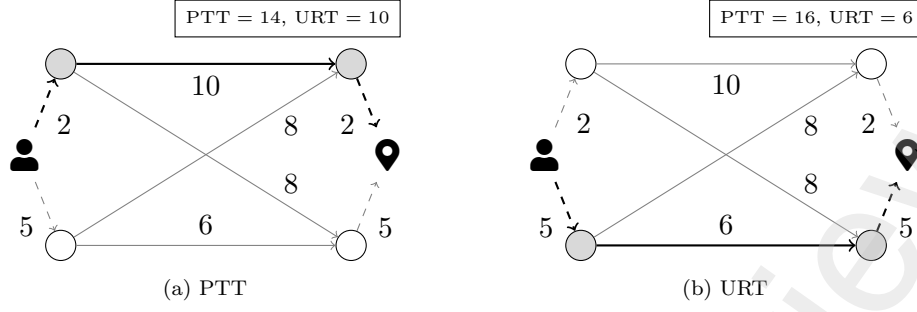


Figure 6: Lower bounds calculation example.

These two lower bounds correspond to the optimal solution value when enough vehicles are available. However, as the fleet size is limited in our instances, the lower bound may not be reachable and, therefore, this may not be the optimal solution value anymore.

The remainder of this section is organized as follows: Section 6.1 compares different versions of our algorithm. Section 6.2 assesses the algorithm’s performance with previous literature. Lastly, Section 6.3 introduces managerial insights.

#### 6.1. Large iteration acceptance, algorithm components, and bus stop selection policies

We devote this section to establishing what is the best version of our algorithm. This means choosing when to accept large destroy iterations to escape local optima in the SLNS algorithms, checking whether SLNS outperforms LNS both with and without the SCP, and finding the best policy to select bus stops.

##### 6.1.1. Large iteration acceptance

In the work of Le Colleter et al. (2023), the proposed SLNS algorithm accepts every solution generated from a large destroy iteration as the new current solution. The rationale behind this approach is to escape local optima by significantly altering the current solution. Following this strategy, a solution generated from a large destroy iteration is accepted even if it has a poor objective function value. However, we encountered challenges in restoring feasibility after an infeasible large destroy iteration when dealing with highly constrained problems, such as minimizing PTT with a limited fleet and time windows in the ODBRP. To address this issue, we adopted the strategy proposed by Soleilhac (2022), where their SLNS algorithm does not accept infeasible solutions after a large destroy iteration. Instead, large destroy iterations are repeated until a feasible solution is obtained. This approach allows for escaping local optima without sacrificing feasibility. We refer to this strategy as the *rolling* large iteration acceptance. Note that in the LNS versions of our algorithms, keeping feasibility is warranted by the acceptance criterion described in Section 4.

Figure 7 illustrates the search profile of SLNS and SLNS+SCP with (and without) the rolling strategy applied across all M500 instances and runs. The figure shows that

SLNS (represented by a solid green line) tended to become stuck after early large destroy iterations, unable to regain feasibility. This results in approximately a 5% performance loss compared to the same algorithm with the rolling strategy (represented by a solid blue line). The case of SLNS+SCP is somewhat different, as the algorithm can recover feasibility through the SCP. However, there is still a notable performance improvement when the rolling strategy is applied to SLNS+SCP (depicted by a dashed orange line) compared to the version without the rolling strategy (depicted by a dashed red line).

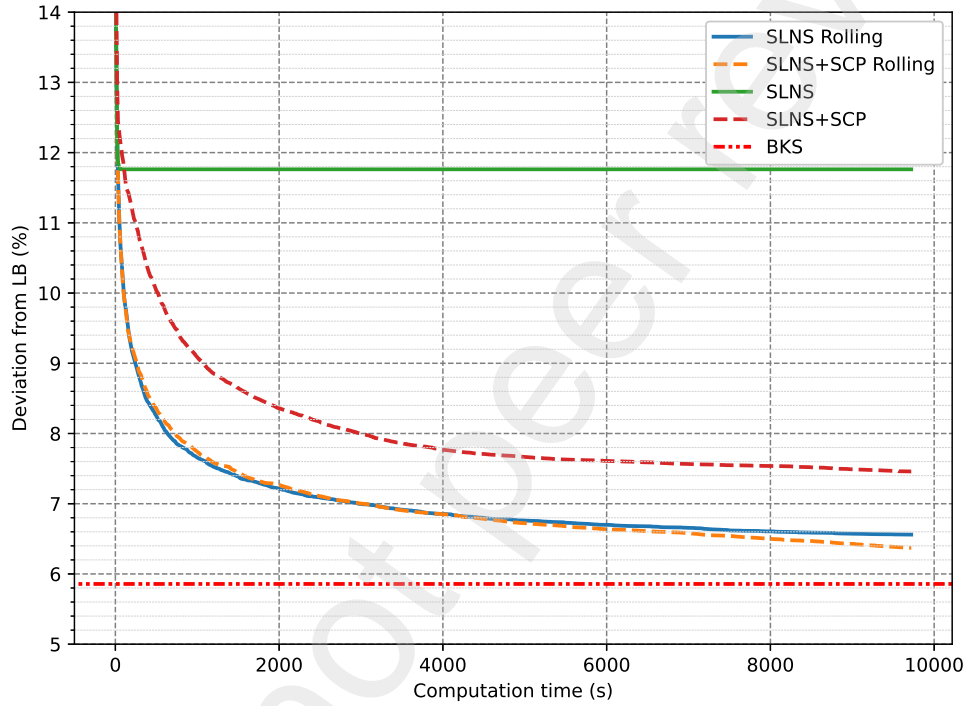


Figure 7: Algorithm search profile with and without the rolling large destroy acceptance over the M500 instances.

Given the demonstrated superiority of SLNS algorithms with the rolling iteration acceptance criterion, all SLNS algorithms in the subsequent sections adopt this approach. However, for the sake of simplicity, we omit the term “Rolling” from the algorithm’s name.

#### 6.1.2. Algorithm components

Tables 2 and 3 present the experimental results comparing SLNS and LNS, both with and without the SCP (i.e., LNS, SLNS, LNS+SCP, and SLNS+SCP). We evaluate the algorithms based on the following indicators: *PTT* (averaged per request in minutes), *length* of the routes (in kilometers), and  $\Delta LB$  (percentage deviation from the lower bound of the PTT). We average the values for these indicators across all instances and algorithm runs for each instance set. All algorithms have identical runtimes, as described in Section

6. The tables show that all four algorithms exhibit similar performance on the two smallest instance sets, with SLNS being slightly ahead. However, when examining the two bottom rows (representing the largest instance sets), we observe that both SLNS and SLNS+SCP achieve solutions with lower PTT (closer to the lower bound) and shorter routes than their LNS counterparts. Furthermore, the algorithms incorporating the SCP (i.e., LNS+SCP and SLNS+SCP) show superior performance compared to their versions without the SCP component. Consequently, it can be concluded that our SLNS implementation outperforms our LNS implementation for the ODBRP and that the addition of the SCP component contributes to finding better solutions.

Set	LB	LNS			SLNS		
		PTT	Length	$\Delta$ LB	PTT	Length	$\Delta$ LB
M50	11.615	11.631	339.738	0.131%	<b>11.630</b>	<b>337.490</b>	<b>0.128%</b>
M100	11.663	11.696	664.314	0.276%	<b>11.693</b>	655.815	<b>0.250%</b>
M250	11.596	12.105	1,442.760	4.376%	12.000	1,418.641	3.478%
M500	11.595	12.523	2,667.895	8.001%	12.355	2,621.367	6.550%

Table 2: Algorithm components without SCP comparison summarized by instance set.

Set	LB	LNS+SCP			SLNS+SCP		
		PTT	Length	$\Delta$ LB	PTT	Length	$\Delta$ LB
M50	11.615	11.631	339.026	0.131%	11.633	338.480	0.150%
M100	11.663	11.694	658.133	0.265%	11.697	<b>652.934</b>	0.289%
M250	11.596	12.053	1,443.645	3.932%	<b>11.999</b>	<b>1,407.924</b>	<b>3.462%</b>
M500	11.595	12.456	2,675.313	7.422%	<b>12.334</b>	<b>2,611.683</b>	<b>6.366%</b>

Table 3: Algorithm components with SCP comparison summarized by instance set.

To further analyze the differences between the algorithms, Figure 8 illustrates the search profile of the four algorithms over all M500 instances and runs, depicting how the algorithms approach the lower bound (y-axis) over time (x-axis). This figure reveals that the SLNS variants not only perform better after 3 hours, but also find better solutions more quickly. We can also see that the SCP enhances solution quality when the algorithms begin to get stuck at the cost of reduced performance at the beginning of the experiment. The horizontal red dash-dotted line (BKS) depicts the averaged best solution found on each instance (always found by the SLNS+SCP). We can see that the SLNS+SCP algorithm is consistent in each instance run as its search profile line ends less than 0.6% away from the BKS. In summary, tables 2 and 3, along with Figure 8, demonstrate that SLNS+SCP outperforms the other three algorithms. Note that the SLNS+SCP algorithm does not appear to have fully converged in Figure 8. We conducted an additional experiment in which the algorithm was run for an additional hour. In this extended run, the algorithm maintained a similar decreasing trend and still showed no signs of convergence. However, after the extra hour, the algorithm only reduced the gap to the lower bound by 0.15% (i.e.,

still 0.45% away from the BKS). Therefore, we conclude that the additional computational effort is not justified by the marginal improvement in solution quality. Finally, for the sake of brevity, the results presented in this section utilize the best bus selection policy, which is chosen in the subsequent section.

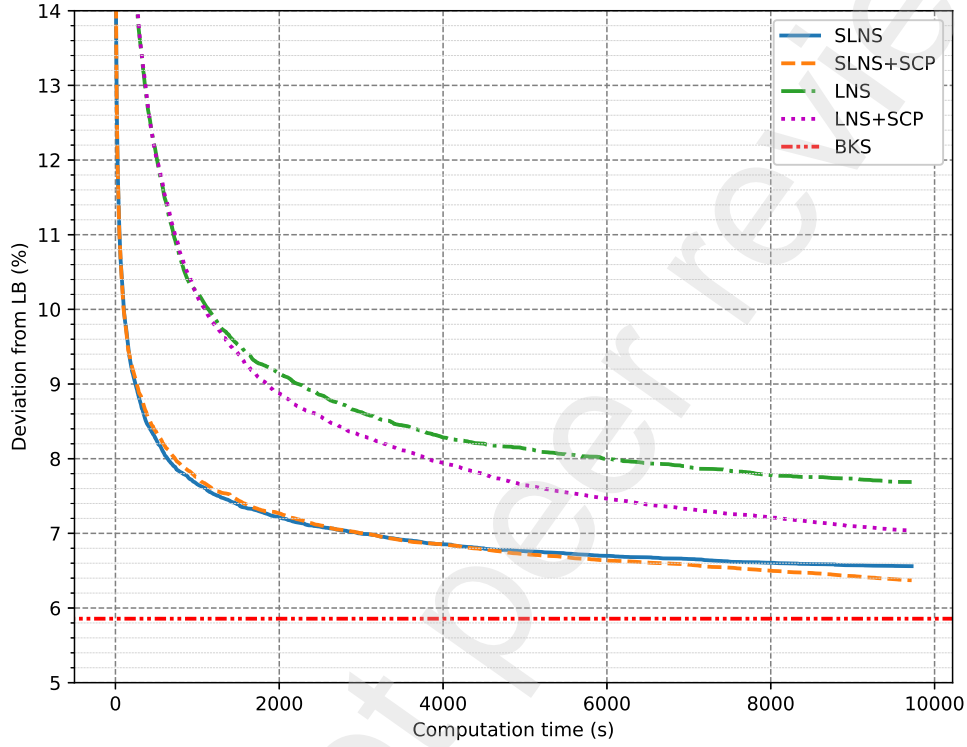


Figure 8: Algorithm components search profile over the M500 instances.

### 6.1.3. Bus stop selection policy

Table 4 compares the algorithm SLNS+SCP with three different bus stop assignment policies: *Nearest*, which assigns the closest bus stop to both the origin and destination; *Sampled*, which evaluates multiple samples; and *CI*, which allows the algorithm to select the most suitable bus stops. Note that in this experiment, one of the samples considered in the Sampled Policy selects the nearest bus stops, as it provides routes where walking time is minimized. However, since the runtimes per sample in the Sampled policy are five times shorter than the runtimes per instance in the Nearest policy, it is unlikely that the Sampled policy will find the same solutions as the Nearest policy. We average the values of the previously introduced indicators across all instances and algorithm runs for each instance set. The algorithm has identical runtimes for the three policies, as described in Section 6. The table clearly shows that allowing the algorithm to select the most suitable



bus stops is the most effective policy across all instance sets, as it reduces both PTT and route length.

Set	LB	Nearest			Sampled			CI		
		PTT	Length	$\Delta LB$	PTT	Length	$\Delta LB$	PTT	Length	$\Delta LB$
M50	11.62	11.86	345.42	2.09%	11.86	347.91	2.14%	<b>11.63</b>	<b>338.48</b>	<b>0.15%</b>
M100	11.66	11.94	667.98	2.38%	11.99	676.75	2.79%	<b>11.70</b>	<b>652.93</b>	<b>0.29%</b>
M250	11.60	12.49	1,416.46	7.64%	13.08	1,433.93	12.74%	<b>12.00</b>	<b>1,407.92</b>	<b>3.46%</b>
M500	11.60	13.05	2,634.27	12.54%	13.56	2,666.30	16.93%	<b>12.33</b>	<b>2,611.68</b>	<b>6.37%</b>

Table 4: Bus stop assignment methods comparison summarized by instance set.

For the rest of the paper, we focus on the best version of the algorithm: SLNS+SCP with the CI bus selection policy and rolling large destroy acceptance. From here on, we refer to this version simply as SLNS+SCP.

### 6.2. Algorithm's performance

We devote this section to checking the performance of our algorithm against known ODBRP algorithms from the literature. As discussed in Section 2, Melis & Sörensen (2022) introduced the ODBRP and proposed an LNS algorithm to solve it. Since the instances used in Melis & Sörensen (2022) were not publicly available, we implemented their LNS algorithm based on the description provided in the original article and tested it on our instances. We refer to our implementation of the algorithm proposed in Melis & Sörensen (2022) as *M-LNS*. For detailed information on the LNS algorithm, we refer the reader to the original article, and for the specific implementation choices we made, please refer to Appendix C. Additionally, the authors designed their algorithm with URT as the objective function and with a number of vehicles as an input parameter. Consequently, we have adapted our SLNS+SCP algorithm to minimize URT instead of PTT.

Table 5 compares the M-LNS to our SLNS+SCP. In this experiment, we compare the indicator *URT* (averaged per request in minutes), and, for simplicity, the columns  $\Delta URT$  and  $\Delta Length$  show our algorithm's deviation from theirs. Note that the  $\Delta LB$  column in Table 5 refers to the URT lower bound explained in Section 6. We average the values for these indicators across all instances and algorithm runs for each instance set. The two algorithms have identical runtimes, as described in Section 6. These results show that our algorithm outperforms the M-LNS with a URT value of up to 24% lower on average in the largest set of instances and with a shorter route length. In the  $\Delta URT$  column, it is clear that our algorithm increasingly outperforms the M-LNS as the instance size grows. However, the opposite trend is observed in the  $\Delta Length$  column. As will be discussed in Section 6.3.1, minimizing URT tends to result in longer routes. Thus, SLNS+SCP sacrifices some route length advantages in larger instances to achieve better URT outcomes. Nevertheless, due to our lexicographic objective function, we still obtain shorter routes than those produced by the M-LNS.

Set	LB	M-LNS			SLNS+SCP				
		URT	Length	$\Delta$ LB	URT	$\Delta$ URT	Length	$\Delta$ Length	$\Delta$ LB
M50	7.91	8.11	383.65	2.47%	<b>7.92</b>	<b>-2.35%</b>	<b>323.83</b>	<b>-15.59%</b>	<b>0.09%</b>
M100	7.96	8.24	721.12	3.49%	<b>7.98</b>	<b>-3.17%</b>	<b>619.69</b>	<b>-14.07%</b>	<b>0.21%</b>
M250	7.90	8.97	1,533.25	13.48%	<b>8.15</b>	<b>-9.15%</b>	<b>1,359.17</b>	<b>-11.35%</b>	<b>3.21%</b>
M500	7.91	11.21	2,611.83	41.73%	<b>8.48</b>	<b>-24.38%</b>	<b>2,523.71</b>	<b>-3.37%</b>	<b>7.17%</b>

Table 5: Comparison of SLNS+SCP with a previous ODBRP algorithm.

To conclude this experiment, we note that when the maximum fleet size constraint is removed, SLNS+SCP consistently reaches the lower bound. In contrast, the M-LNS algorithm, provided with the same number of vehicles used by SLNS+SCP, is unable to achieve the lower bound.

### 6.3. Managerial insights

In this section, we highlight key aspects of our research that may be of interest to managers and decision-makers when addressing the ODBRP.

#### 6.3.1. Impact of optimizing different objective functions

This study addresses the ODBRP from a public service perspective, assuming the operator is a public entity aiming to maximize service quality. Accordingly, we focus on a lexicographic minimization of the PTT and the total route length. However, alternative objectives may also be considered: Previous studies have focused on minimizing URT, and a common objective for private operators is minimizing the total cost, which is directly related to the distance traveled by vehicles (i.e., total route length). In this section, we will compare all three objectives.

Figure 9 presents a comparison of the best solutions found by SLNS+SCP, averaged across all instance sets and runs. The algorithm has identical runtimes for the three objectives, as described in Section 6. This chart displays the average PTT per request when minimizing PTT, URT, and total route length. The PTT (shown with green labels) is divided into URT (represented by orange bars) and walk time (represented by blue bars). The results indicate that when minimizing total route length, the URT nearly doubles compared to minimizing PTT and URT, with passengers spending over 17 minutes on the bus, compared to around 8 to 9 minutes when PTT or URT is minimized. This outcome is likely due to routes involving multiple customers being on the bus simultaneously when total route length is minimized. While this reduces route length and improves vehicle capacity efficiency, it negatively impacts URT, as having more requests onboard increases ride times. When comparing the columns for minimizing PTT and URT, as expected, URT is lower when it is the objective function, and similarly, PTT is lower when it is the objective function. However, while the URT per request is relatively similar when minimizing URT and PTT, there is a more significant difference in walk time. Because

walking speed is slower than bus speed, bus stops closer to origins and destinations are chosen when minimizing PTT. In this work, we opted for minimizing PTT over URT, as we assume that users would prefer getting to destination as soon as possible at nighttime. Lastly, we notice that PTT, URT, and walk time values are consistent across all sets of instances.

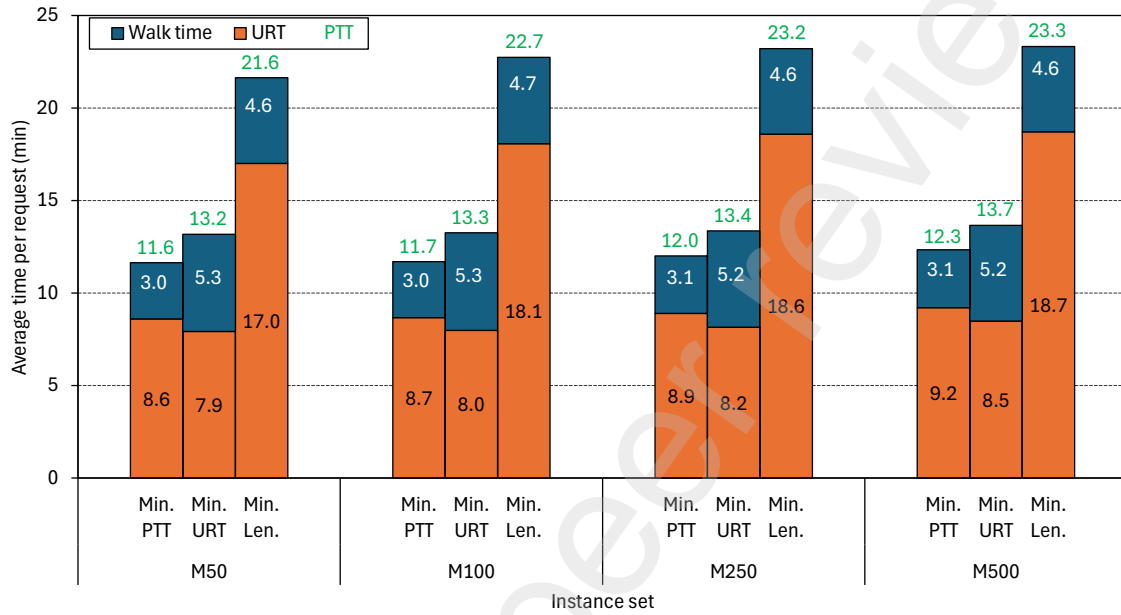


Figure 9: URT and walk time comparison when minimizing URT, PTT, and total route length.

Similarly to Figure 9, Figure 10 presents a chart comparing the minimization of PTT, URT, and total route length, this time focusing on the total route length measure. In the same line as the previous figure, the total route length is significantly shorter when the distance is minimized. This difference in total route length increases with instance size, reducing route lengths by half compared to when PTT or URT is minimized. This is because solutions that minimize URT and PTT tend to serve requests consecutively, minimizing overlap but resulting in longer routes. The slight difference between PTT and URT is directly related to the URT variations observed in the previous figure. Additionally, it is worth noting that when minimizing URT and PTT, multiple solutions may exist with varying distances for the same URT or PTT value. Therefore, we chose to minimize distance as a secondary objective. Without this secondary objective, the disparity in route length between PTT/URT minimization and distance minimization could be even larger.

From a practical perspective, the choice of objective function will depend on the operator's priorities. However, it is important to recognize that minimizing PTT or URT is inefficient for reducing total route length, and vice versa. Therefore, aiming for a balance

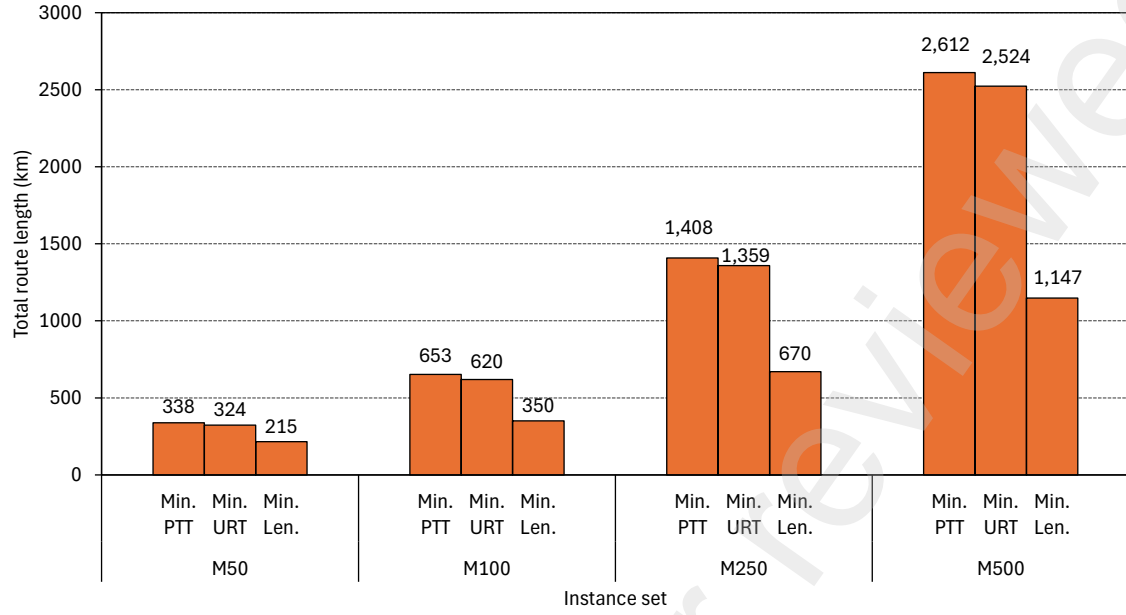


Figure 10: Total route length comparison when minimizing URT, PTT, and total route length.

between these objectives could be a valuable consideration for future optimization efforts.

### 6.3.2. Comparison to a door-to-door policy

It is reasonable to compare the ODBRP with a more traditional door-to-door policy. As the name suggests, in a door-to-door approach, requests are picked up directly from their origins and dropped off at their destinations, traveling from door to door. These policies are often designed for people with reduced mobility. As expected, while door-to-door policies offer a lower PTT (i.e., users do not need to walk), they tend to be less efficient from a cost and operational standpoint.

Table 6 compares our algorithm's performance on the ODBRP with its application to the door-to-door bus routing problem (denoted D2DBRP for this comparison), where pickups occur directly at the origins and drop-offs at the destinations. In this experiment, we compare the maximum number of vehicles used  $|K|$ , and, for the ODBRP we added the walking time averaged per request in minutes  $Walk$ . For simplicity, the columns  $\Delta PTT$  and  $\Delta Length$  show our algorithm's deviation from the D2DBRP policy. We average the values for these indicators across all instances and algorithm runs for each instance set. The two algorithms have identical runtimes, as described in Section 6. Note that the  $Walk$  and  $URT$  columns are omitted for the D2DBRP, as users do not walk, making the PTT identical to the URT. The first notable observation is that to obtain feasible solutions for the D2DBRP, we had to increase the fleet size by one vehicle for the M250 instance set and by two vehicles for the M500 instance set. This is a direct consequence of the longer

origin-to-destination routes for each request. Similarly, while the ODBRP was able to find solutions using four vehicles in the M50 instance set, this was not the case for the D2DBRP. As a result, the time spent inside the bus per passenger (shown as PTT in the door-to-door policy, and as URT in the ODBRP) is larger when using a door-to-door policy. However, this increase in URT is mitigated when adding the walk time, having up to 23% more PTT per request. We also observe a clear decreasing trend in  $\Delta$ PTT as the instance size grows. Similarly, the total route length is shorter in the ODBRP, with  $\Delta$ Length increasing as the instance size increases.

Set	D2DBRP			ODBRP						
	$ K $	PTT	Length	$ K $	PTT	$\Delta$ PTT	Walk	URT	Length	$\Delta$ Length
M50	5	9.42	363.78	4.99	11.63	23.51%	3.03	8.60	338.48	-6.96%
M100	8	9.57	701.28	8	11.70	22.25%	3.03	8.67	652.93	-6.89%
M250	16	9.98	1,516.87	15	12.00	20.17%	3.10	8.90	1,407.92	-7.18%
M500	28	10.65	2,835.20	26	12.33	15.85%	3.13	9.20	2,611.68	-7.88%

Table 6: Comparison of the ODBRP with a door-to-door bus routing service.

In conclusion, from a service quality perspective, although users spend more time inside the bus in a door-to-door service, the PTT is lower since users do not need to walk to bus stops, thereby enhancing convenience. From a cost perspective and when minimizing PTT, it is more efficient for the operator to slightly reduce service quality (i.e., increase users' PTT) while reducing the fleet size and cutting the total route length by approximately 8%. Moreover, this advantage over the D2DBRP is likely to continue to increase as the number of requests further increases. From an operational standpoint, it is more practical to have passengers walk to existing bus stops, rather than stopping the bus at arbitrary points along the streets, which could temporarily obstruct traffic.

### 6.3.3. Impact of the bus stop options number

From a managerial point of view it is also worth considering whether increasing or decreasing the number of potential bus stops for each request would lead to an improvement in solution quality. To investigate this, we generated a set of instances in which up to the 10 nearest bus stops within a 5-minute walking distance were considered. We then solved these instances using the SLNS+SCP algorithm and compared the results with instances that limited the number of bus stops to 5 and 1 (i.e., the same instances as in the nearest bus stop selection policy). Lastly, all instances have the same runtime limits defined at the beginning of Section 6.

Figure 11 presents a comparison of the best solutions found by SLNS+SCP when minimizing PTT, averaged across all instance sets and runs. The chart displays the average PTT per request with 1, 5, and 10 bus stops. The PTT (shown with green labels) is divided into URT (represented by orange bars) and walk time (represented by blue bars).

The results reveal very similar solutions when using either 5 or 10 bus stops, with only a slight difference in the largest instance set, which may be attributable to the increased computational effort. However, compared to using the nearest bus stop, while a shorter walking time is achieved, the increase in URT results in a worse PTT, as shown in Section 6.1.3.

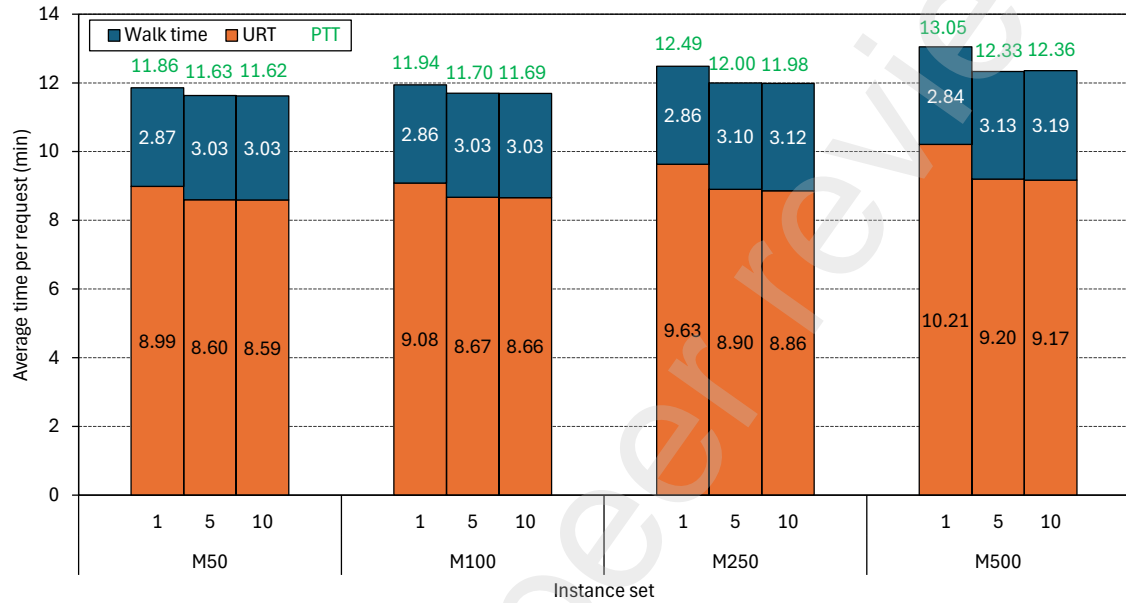


Figure 11: URT and walk time comparison with 1, 5, and 10 bus stops per pick-up and drop-off.

## 7. Conclusions

We presented a novel matheuristic algorithm, called SLNS+SCP, for solving the ODBRP. The SLNS improves over the LNS at the beginning of the runs by focusing on doing small iterations. Additionally, the SCP component aids in improving performance when the SLNS stalls. An implementation analysis of SLNS's large iterations revealed that SLNS struggles to restore feasibility in highly constrained problems, but accepting only feasible solutions after large destruction iterations helps to address this issue.

The SLNS+SCP was able to outperform our implementation of a previous algorithm from the literature, not only in terms of service quality (URT) but also by generating shorter (and therefore more cost-effective) routes. This success is due to two aspects. First, to our chosen lexicographic objective function, which prioritizes finding the solution with the shortest route length among those with equivalent URT or PTT values, since many equivalent solutions may exist. Second, through a more efficient combination of destroy and repair operators, supported by an SCP component capable of assembling

routes from different solutions.

A new set of realistic instances, based on data from New York City, was developed and utilized for the computational experiments. This dataset incorporates historical taxi trips, actual bus stop locations, and realistic time and distance matrices.

Our analyses revealed that minimizing URT or PTT results in solutions with entirely different structures when compared to the ones resulting from minimizing the total route length. Specifically, minimizing service quality (i.e., URT or PTT) tends to produce more costly solutions, characterized by longer routes, and vice versa. Additionally, a comparison with a door-to-door policy showed that using bus stops results in lower service quality but requires a smaller fleet and shorter route lengths, leading to a more cost-effective bus system. Furthermore, no significant difference was observed in PTT minimization when additional bus stops were included.

We propose two future work streams: first, the development of an exact algorithm that would provide a more accurate benchmark for evaluating the algorithm performance, tightening the proposed lower bound; and second, the integration of this algorithm into a simulator for handling real-time requests.

## References

- Archetti, C., & Speranza, M. G. (2014). A survey on matheuristics for routing problems. *EURO Journal on Computational Optimization*, 2, 223–246.
- Archetti, C., Speranza, M. G., & Weyland, D. (2018). A simulation study of an on-demand transportation system. *International Transactions in Operational Research*, 25, 1137–1161.
- Baldacci, R., Bartolini, E., & Mingozzi, A. (2011). An exact algorithm for the pickup and delivery problem with time windows. *Operations Research*, 59, 414–426.
- Bautista, J., Fernández, E., & Pereira, J. (2008). Solving an urban waste collection problem using ants heuristics. *Computers & Operations Research*, 35, 3020–3033.
- Bektaş, T., Erdoğan, G., & Røpke, S. (2011). Formulations and branch-and-cut algorithms for the generalized vehicle routing problem. *Transportation Science*, 45, 299–316.
- Braekers, K., Caris, A., & Janssens, G. K. (2014). Exact and meta-heuristic approach for a general heterogeneous dial-a-ride problem with multiple depots. *Transportation Research Part B: Methodological*, 67, 166–186.
- Brooks, L. (2023). Decision to scrap Glasgow night bus service prompts outcry. *The Guardian*, . URL: <https://www.theguardian.com/uk-news/2023/jul/>

11/decision-to-scrap-glasgow-night-bus-service-prompts-outcry. [Online; accessed 29-August-2024].

- Bulhões, T., Ha, M. H., Martinelli, R., & Vidal, T. (2018). The vehicle routing problem with service level constraints. *European Journal of Operational Research*, 265, 544–558.
- Christiaens, J., & Vanden Berghe, G. (2020). Slack induction by string removals for vehicle routing problems. *Transportation Science*, 54, 417–433.
- Cordeau, J.-F. (2006). A branch-and-cut algorithm for the dial-a-ride problem. *Operations research*, 54, 573–586.
- Cordeau, J.-F., & Laporte, G. (2003). A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, 37, 579–594.
- Costa, L., Contardo, C., & Desaulniers, G. (2019). Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Science*, 53, 946–985.
- Crainic, T. G., Errico, F., Malucelli, F., & Nonato, M. (2012). Designing the master schedule for demand-adaptive transit systems. *Annals of Operations Research*, 194, 151–166.
- Curtois, T., Landa-Silva, D., Qu, Y., & Laesanklang, W. (2018). Large neighbourhood search with adaptive guided ejection search for the pickup and delivery problem with time windows. *EURO Journal on Transportation and Logistics*, 7, 151–192.
- Dueck, G. (1993). New optimization heuristics: The great deluge algorithm and the record-to-record travel. *Journal of Computational Physics*, 104, 86–92.
- Dumas, Y., Desrosiers, J., & Soumis, F. (1991). The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54, 7–22.
- Duméz, D., Lehuédé, F., & Péton, O. (2021a). A large neighborhood search approach to the vehicle routing problem with delivery options. *Transportation Research Part B: Methodological*, 144, 103–132.
- Duméz, D., Tilk, C., Irnich, S., Lehuédé, F., & Péton, O. (2021b). Hybridizing large neighborhood search and exact methods for generalized vehicle routing problems with time windows. *EURO Journal on Transportation and Logistics*, 10, 100040.
- Estrada, M., Salanova, J. M., Medina-Tapia, M., & Robusté, F. (2021). Operational cost and user performance analysis of on-demand bus and taxi systems. *Transportation Letters*, 13, 229–242.



- Foster, B. A., & Ryan, D. M. (1976). An integer programming approach to the vehicle scheduling problem. *Journal of the Operational Research Society*, 27, 367–384.
- Ghiani, G., & Improta, G. (2000). An efficient transformation of the generalized vehicle routing problem. *European Journal of Operational Research*, 122, 11–17.
- GIS Lab (2019). New York City Bus Stops. URL: <https://archive.nyu.edu/handle/2451/60059/> [Online; retrieved 25-September-2023].
- Grangier, P., Gendreau, M., Lehuédé, F., & Rousseau, L.-M. (2017). A matheuristic based on large neighborhood search for the vehicle routing problem with cross-docking. *Computers & Operations Research*, 84, 116–126.
- Gschwind, T., & Drexler, M. (2019). Adaptive large neighborhood search with a constant-time feasibility test for the dial-a-ride problem. *Transportation Science*, 53, 480–491.
- Gschwind, T., & Irnich, S. (2015). Effective handling of dynamic time windows and its application to solving the dial-a-ride problem. *Transportation Science*, 49, 335–354.
- Hexaly (2024). Optimization solver and services. URL: <https://www.hexaly.com/> [Online; accessed 29-August-2024].
- Horn, M. E. (2002). Fleet scheduling and dispatching for demand-responsive passenger services. *Transportation Research Part C: Emerging Technologies*, 10, 35–63.
- Huang, D., Gu, Y., Wang, S., Liu, Z., & Zhang, W. (2020). A two-phase optimization model for the demand-responsive customized bus network design. *Transportation Research Part C: Emerging Technologies*, 111, 1–21.
- Jodeau, J., Absi, N., Chevrier, R., & Feillet, D. (2024). The rail-road dial-a-ride problem. *European Journal of Operational Research*, 318, 486–499.
- Jolfaei, A. A., Alinaghian, M., Bahrami, R., & Tirkolaee, E. B. (2023). Generalized vehicle routing problem: Contemporary trends and research directions. *Heliyon*, 9, e22733.
- Le Colleter, T., Dumez, D., Lehuédé, F., & Péton, O. (2023). Small and large neighborhood search for the park-and-loop routing problem with parking selection. *European Journal of Operational Research*, 308, 1233–1248.
- Li, H., & Lim, A. (2001). A metaheuristic for the pickup and delivery problem with time windows. In *Proceedings 13th IEEE International Conference on Tools with Artificial Intelligence. ICTAI 2001* (pp. 160–167).
- Li, X., & Quadrioglio, L. (2010). Feeder transit services: Choosing between fixed and demand responsive policy. *Transportation Research Part C: Emerging Technologies*, 18, 770–780.

- López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3, 43–58.
- Mara, S. T. W., Norcahyo, R., Jodiawan, P., Lusiantoro, L., & Rifai, A. P. (2022). A survey of adaptive large neighborhood search algorithms and applications. *Computers & Operations Research*, 146, 105903.
- Melis, L., & Sörensen, K. (2022). The static on-demand bus routing problem: large neighborhood search for a dial-a-ride problem with bus station assignment. *International Transactions in Operational Research*, 29, 1417–1453.
- New York City Taxi and Limousine Commission (2024). TLC Trip Record Data. URL: <https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page> [Online; retrieved 25-September-2023].
- openrouteservice (2024). Smart Mobility made easy! URL: <https://openrouteservice.org/> [Online; accessed 29-August-2024].
- Padam Mobility (2024). Demand-responsive transport reinvented. URL: <https://www.padam-mobility.com/> [Online; accessed 29-August-2024].
- Parragh, S. N., Doerner, K. F., & Hartl, R. F. (2010). Variable neighborhood search for the dial-a-ride problem. *Computers & Operations Research*, 37, 1129–1138.
- Pei, M., Lin, P., Du, J., & Li, X. (2019). Operational design for a real-time flexible transit system considering passenger demand and willingness to pay. *IEEE Access*, 7, 180305–180315.
- Pisinger, D., & Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34, 2403–2435.
- Pisinger, D., & Ropke, S. (2019). Large neighborhood search. *Handbook of metaheuristics*, (pp. 99–127).
- Plyushteva, A., & Boussauw, K. (2020). Does night-time public transport contribute to inclusive night mobility? Exploring Sofia’s night bus network from a gender perspective. *Transport Policy*, 87, 41–50.
- Reihaneh, M., & Ghoniem, A. (2018). A branch-cut-and-price algorithm for the generalized vehicle routing problem. *Journal of the Operational Research Society*, 69, 307–318.
- Ropke, S., & Cordeau, J.-F. (2009). Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science*, 43, 267–286.

- Ropke, S., & Pisinger, D. (2006a). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40, 455–472.
- Ropke, S., & Pisinger, D. (2006b). A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research*, 171, 750–775.
- Sadati, M. E. H., Akbari, V., & Çatay, B. (2022). Electric vehicle routing problem with flexible deliveries. *International Journal of Production Research*, 60, 4268–4294.
- Sartori, C. S., & Buriol, L. S. (2020). A study on the pickup and delivery problem with time windows: Matheuristics and new instances. *Computers & Operations Research*, 124, 105065.
- Savelsbergh, M. W. (1992). The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing*, 4, 146–154.
- Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *International Conference on Principles and Practice of Constraint Programming* (pp. 417–431). Springer.
- Soleilhac, G. (2022). *Optimisation de la distribution de marchandises avec sous-traitance du transport : une problématique chargeur*. Ph.D. Thesis Ecole nationale supérieure Mines-Télécom Atlantique.
- Stiglic, M., Agatz, N., Savelsbergh, M., & Gradisar, M. (2018). Enhancing urban mobility: Integrating ride-sharing and public transit. *Computers & Operations Research*, 90, 12–21.
- Tellez, O., Vercraene, S., Lehuédé, F., Péton, O., & Monteiro, T. (2018). The fleet size and mix dial-a-ride problem with reconfigurable vehicle capacity. *Transportation Research Part C: Emerging Technologies*, 91, 99–123.
- Turkeš, R., Sörensen, K., & Hvattum, L. M. (2021). Meta-analysis of metaheuristics: Quantifying the effect of adaptiveness in adaptive large neighborhood search. *European Journal of Operational Research*, 292, 423–442.
- Vansteenwegen, P., Melis, L., Aktaş, D., Montenegro, B. D. G., Vieira, F. S., & Sörensen, K. (2022). A survey on demand-responsive public bus systems. *Transportation Research Part C: Emerging Technologies*, 137, 103573.
- Via Transportation (2024). Software and operations for flexible public transit. URL: <https://ridewithvia.com/> [Online; accessed 29-August-2024].

## Appendix A. Notation

Symbol	Meaning	Section
$R$	Transportation requests set	3
$r$	Transportation request in $R$	3
$P_r$	Set of potential pick-up bus stops of request $r$	3
$P_r$	Set of potential drop-off bus stops of request $r$	3
$e_r$	Earliest time request $r$ wants to be picked up	3
$l_r$	Latest time request $r$ wants to be dropped off	3
$P$	Set including all potential pick-up bus stops of all requests	3
$D$	Set including all potential drop-off bus stops of all requests	3
$N$	Set of nodes containing $P$ , $D$ , and the two copies of the depot	3
$A$	Set of arcs connecting all the nodes in $N$	3
$G$	Graph defined by the set of nodes $N$ and the set of arcs $A$	3
$K$	Fleet of vehicles	3
$Q$	Capacity of vehicle $k \in K$	3
$q_i$	Passenger load of visiting node $i$	3
$a_i$	Earliest time node $i$ can be visited	3
$b_i$	Latest time node $i$ can be visited	3
$w_i$	Walking time from the origin towards node $i$ or from node $i$ towards the destination	3
$d_{ij}$	Length of the arc $(i, j) \in A$	3
$r_{ij}$	Time needed to traverse arc $(i, j) \in A$	3
$\sigma_r$	Service duration of picking up or dropping off request $r$	3
$x_{ij}^k$	Binary variable that takes the value 1 if and only if the vehicle $k$ uses the arc $(i, j)$	3
$A_i^k$	Non-negative variable defining arrival time at node $i$ of vehicle $k$	3
$L_i^k$	Non-negative variable defining load at node $i$ of vehicle $k$	3
$T_r$	Non-negative variable defining the travel time of request $r$	3
$M_{ij}^k$	Big constant where $M_{ij}^k = \max \{0, b_i + t_{ij} - a_j\}$	3

**Table A.7 continued from previous page**

Symbol	Meaning	Section
$W_{ij}^k$	Big constant where $W_{ij}^k = \min \{Q, Q + q_i\}$	3
$H_{j'j}^k$	Big constant where $H_{j'j}^k = b_{j'} - a_j + w_j + w_{j'}$	3
$Sol$	Solution of the ODBRP	4.1
$B(Sol)$	Request bank (i.e., non-served requests) of $Sol$	4.1
$\rho$	Route serving some requests	4.1
$\Sigma_{large}^-$	Set of large destroy operators	4.1
$\Sigma_{small}^-$	Set of small destroy operators	4.1
$\Sigma^+$	Set of repair operators	4.1
$o_{init}^+$	Initial repair operator	4.1
$\delta_{small}^{min}$	Minimum number of customers to be removed by a small destroy operators	4.1
$\delta_{small}^{max}$	Maximum number of customers to be removed by a small destroy operators	4.1
$\delta_{large}^{min}$	Minimum number of customers to be removed by a large destroy operators	4.1
$\delta_{large}^{max}$	Maximum number of customers to be removed by a large destroy operators	4.1
$\omega_{small}$	Maximum number of iterations without improvement between small and large destroy operators	4.1
$\omega_{worse}$	Minimum number of iterations without improvement before launching the exact component	4.1
$\Omega$	Pool of routes	4.1
$\beta$	Minimum number of routes in $\Omega$ before launching the exact component	4.1
$\omega_{time}$	Maximum time we spend in the exact component every time it is called	4.1
$\tau$	Ratio to increase or decrease $\beta$ dynamically	4.1
$Sol^{best}$	Best solution found by the algorithm	4.1
$Sol^{curr}$	Current solution in the algorithm to modify	4.1
$Sol^{new}$	Solution obtained by modifying $S^{current}$	4.1
$it_{worse}$	Counter of iterations without improvement before launching the SCP	4.1
$it_{small}$	Counter of iterations without improvement between small and large destroy operators	4.1
$\alpha$	Record-to-record threshold for accepting a new solution as the current solution	4.1

**Table A.7 continued from previous page**

Symbol	Meaning	Section
$\Delta_{small}$	Maximum number of removed customers by a small destroy operator	4.2
$\Delta_{large}$	Maximum number of removed customers by a large destroy operator	4.2
$\Pi$	Probability of blinking in a certain insertion of a request $r$ in a route $\rho$	4.2
$T_\rho$	Total PTT of route $\rho$	4.3
$a_{r\rho}$	Constant indicating if the request $r$ is visited in route $\rho$	4.3
$\lambda_{r\rho}$	Binary variable indicating if route $\rho$ is used in the solution	4.3

## Appendix B. Comparison on PDPTW instances

Since the ODBRP is a generalization of the PDPTW, we tested our algorithm on the PDPTW instances proposed by Li & Lim (2001). As the PDPTW literature typically focuses on minimizing distance, we adapted our algorithm to this objective function and compared its performance to the known optimal solutions identified by Ropke & Cordeau (2009) and Baldacci et al. (2011).

Table B.8 presents the performance of SLNS+SCP on the Li & Lim (2001) benchmark. In this table, the column *Set* indicates the instance set, the column *#Instances* shows the number of instances in each set, the column *Known optimal* indicates the number of instances with a proven optimal solution, the column *Found optimal* shows how many of these optimal solutions the SLNS+SCP finds, and the column *Deviation* represents the average deviation from optimality for the instances where SLNS+SCP does not find the known optimal solution. The results show that SLNS+SCP finds 74 out of 75 known optimal solutions, and in the one instance where the algorithm does not find the optimal solution, the best solution identified by SLNS+SCP is only 0.14% above the optimal value.

Set	#Instances	Known optimal	Found Optimal	Deviation
50	56	39	39	—
100	60	23	22	0.14%
200	60	4	4	—
300	60	3	3	—
400	60	3	3	—
500	58	3	3	—
Total	354	75	74	

Table B.8: Comparison to the optimal solutions of Li &amp; Lim (2001) instances.

## Appendix C. Implementation choices on the LNS

Re-implementing algorithms from the literature can be challenging, as certain implementation details are often omitted for the sake of brevity. Consequently, we made some implementation choices that might differ slightly from those made by Melis & Sörensen (2022). For the sake of transparency, we report those choices here.

When determining which bus stop should be used for picking up or dropping off a request, our algorithm evaluates all potential bus stops, while their algorithm pre-selects the stop that “causes the smallest increase in route duration”. In their synthetic instances, trip length is linearly related to trip duration; however, this is not necessarily the case in our real-world instances. Therefore, to also minimize total route length, our implementation of their algorithm selects the bus stop that results in the smallest increase in route length. Additionally, their algorithm uses a penalty factor when constructing the initial routes in spite of finding feasible solutions. This penalty is also based on the increase in route duration when a request is inserted. For the same reasons as mentioned earlier, we have instead based the penalty on the increase in route length.

Their algorithm does not account for the presence of a depot for starting and finishing routes. Therefore, as there is a depot in our instances, our implementation of their algorithm enforces that all routes begin and end at the depot.

Since no acceptance criterion is specified for accepting new solutions as the current solution, we adopt the following approach: when minimizing URT (intensification), we accept any solution that improves the penalized objective function value. This means either the solution is feasible, or, if infeasible, a penalty is added to the objective function. However, when minimizing distance (aiming to restore feasibility), we only accept solutions that either serve more requests or, if the same number of requests is served, have a lower penalized objective function. This last decision was made as it helps to recover feasibility.

To enhance intensification, the authors incorporated a local search mechanism. While the number of local search iterations is specified after recovering feasibility, it is not detailed for the URT minimization phase. In this case, we have assumed that the local search is only called once.