TEL AVIV UNIVERSITY

THE IBY AND ALADAR FLISCHMAN FACULTY OF ENGINEERING

School of Electrical Engineering

The Dial-A-Ride Problem with Transfers and Walking

Thesis submitted toward the degree of

Master of Science in Electrical and Electronic Engineering

in Tel-Aviv University

By

Idan Meshulami

March 2026

TEL AVIV UNIVERSITY

THE IBY AND ALADAR FLISCHMAN FACULTY OF ENGINEERING

School of Electrical Engineering

The Dial-A-Ride Problem with Transfers and Walking

Thesis submitted toward the degree of
Master of Science in Electrical and Electronic Engineering
in Tel-Aviv University

By

Idan Meshulami

This research work was carried out at Tel-Aviv University
in the School of Electrical Engineering,
Faculty of Engineering
under the supervision of Dr. Mor Kaspi

March 2026

Abstract

This paper introduces the Dial-A-Ride Problem with Transfers and Walking (DARPTW), a novel generalization of the classical DARP that permits passenger itineraries to include multiple vehicle transfers and short walking segments at any stage of their journey. Urban mobility challenges, such as long travel times and low service frequencies, necessitate efficient demand-responsive services that can balance operational costs with user service quality. We formulate the DARPTW as a four-index Mixed Integer Linear Problem (MILP) aimed at minimizing a multi-criteria objective function that combines total vehicle travel time and total user journey time.

Incorporating walking and transfers provides two primary saving opportunities: walking assists in reducing unnecessary vehicle detours to remote regions and facilitates shortcuts through the road network, while transfers allow the system to balance vehicle loads and reduce the service area covered by individual vehicles. While these features provide significant operational flexibility, they also increase problem complexity beyond the standard NP-hard DARP.

To address this challenge, we propose a modular solution approach. We first model the scheduling subproblem as a Simple Temporal Problem (STP), utilizing the Bellman-Ford algorithm to provide a computationally efficient feasibility check. This is followed by a four-step heuristic scheduler designed to construct near-optimal schedules in polynomial time. These components are integrated into an Adaptive Large Neighborhood Search (ALNS) framework, which features a novel similarity metric for user requests and specialized insertion operators for transfers and walking segments.

Computational experiments based on real-world data from Bubble Dan in Israel demonstrate the effectiveness of our approach. The Bellman-Ford feasibility check operates up to 200 times faster than exact solvers on small instances, while the heuristic scheduler achieves a mean optimality gap of just 1.3% for larger instances. At the routing level, the ALNS consistently identifies feasible solutions for larger problem sizes where exact methods become intractable. Finally, a focused analysis reveals that transfers serve as a critical structural mechanism, most beneficial under tight constraints such as restricted walking flexibility or limited vehicle capacity.

Table of Contents

List of Symbols

R	The set of requests
M	The set of vehicles
Φ	The set of physical transfers
P	The set of pick-up nodes
D	The set of drop-off nodes
T	The set of transfer nodes
N	The set of all possible nodes for passengers;
	The maximum number of ALNS iterations
O	The set of origin depot nodes
F	The set of destination depot nodes
V	The set of all possible nodes for vehicles
n	The number of requests
m	The number of vehicles
τ	The number of physical transfers
γ	The maximum times a vehicle can visit a physical transfer
e_r	The earliest time window of request $r \in R$
l_r	The latest time window of request $r \in R$
q_r	The size of request $r \in R$
J_r	The maximum journey time of request $r \in R$
Θ_r	The maximum transfers for request $r \in R$
W_r	The maximum total walking time of request $r \in R$
U_r	The maximum walking time per section of request $r \in R$
Q_k	The maximum capacity of vehicle $k \in M$
d_i	The duration of boarding time at node $i \in V$
α_{ij}	The walking time between node $i \in N$ and $j \in N$
β_{ij}	The riding time from node $i \in V$ to node $j \in V$
PH	The planning horizon
S_i	The schedule time when service begins at node $i \in V$
L_r	The journey time of user $r \in R$
η	The weighting factor in the objective function
n_d	The destruction rate in the destroy phase
λ	The decay parameter for weight updates
ψ	The scores update reward
T_0	The initial temperature used in the Simulated Annealing
K	The number of potential insertion options in the repair phase

List of Figures

List of Tables

1. Introduction

The ongoing urbanization process and its attendant centralization create many challenges and opportunities for the development of urban mobility. While concentrated populations make public transit a more viable alternative due to the high potential for demand pooling, many citizens may still find this alternative not sufficiently appealing. Prominent reasons include: long travel times, low service frequencies and long access distances that are not reasonable for walking. To overcome this, many cities have introduced demand responsive services with the aim of providing high quality, taxi-like, services at costs comparable to public transit.

Dial-A-Ride (DAR) services are a class of demand-responsive transportation originally developed to provide transit solutions for mobility-impaired passengers (Wilson et al., 1971; Cordeau and Laporte, 2003). In these systems, a dedicated fleet provides on-demand, point-to-point trips. Historically, users were required to reserve rides one or two days in advance via telephone, whereas modern implementations, often referred to as ride-pooling (Hansen and Sener, 2023), operate through smartphone applications and support both advance booking and near real-time requests.

The underlying operational optimization problem is known as the Dial-A-Ride Problem (DARP), in which a fleet of vehicles serves a set of passenger transportation requests, typically made in advance. Each request includes a pick-up location, a drop-off location, and service constraints such as time windows and maximum ride duration. Vehicles are subject to capacity limits and depot constraints, including predefined start locations and mandatory return to an end location. The objective is to determine vehicle routes and schedules that optimize a function typically combining operating costs with quality-of-service components. The DARP is a generalization of the Pickup and Delivery Problem (PDP) and is therefore NP-hard (Parragh et al., 2008). Unlike the PDP, which concerns the transport of goods, the DARP involves passenger transport and therefore requires explicit consideration of service quality aspects (Jørgensen et al., 2007; Lehuédé et al., 2014). Comprehensive surveys of the DARP literature are provided by Cordeau and Laporte (2007), Molenbruch et al. (2017), and Ho et al. (2018), which classify major variants of the problem and review a wide range of exact and heuristic solution methods.

Numerous variants of the DARP have been proposed to improve user service while reducing operational costs. Hall et al. (2008) introduced the Integrated Dial-a-Ride Problem (IDARP), in which portions of passenger journeys may be served by fixed-route transit. Posada

et al. (2016) extended this framework by incorporating fixed-route timetables, which require careful synchronization between vehicle arrivals and transfer opportunities.

Allowing transfers is another way to increase system flexibility while accepting a modest reduction in passenger convenience, an idea first introduced by Stein et al. (1978). Cortes et al. (2010) studied the Pickup and Delivery Problem with transfers and proposed a branch-and-cut formulation, and Masson et al. (2014) defined the Dial-A-Ride Problem with Transfers (DARPT), in which a passenger itinerary may be split at a designated transfer node into two segments connected by a single transfer, each served by a different vehicle. Although transfers create opportunities for improved system efficiency, they also substantially increase the complexity of the resulting optimization problem.

Incorporating short walking segments into passenger itineraries is an additional mechanism to consolidate requests and reduce operating costs. Zheng et al. (2019) introduced the checkpoint DARP, in which passengers may be required to walk to predefined meeting points in order to receive service, demonstrating that such concessions can significantly increase request acceptance rates. Fielbaum et al. (2021) studied on-demand ridesharing with optimized pick-up and drop-off walking locations and proposing a system in which users may be instructed in real time to walk to or from nearby pick-up or drop-off locations when this improves overall efficiency. Melis et al. (2021) formulated an optimization model to support the planning and routing of on-demand urban buses, showing that total user ride times can be significantly reduced in flexible public transport systems. Sarma et al. (2023) proposed a decomposition framework for dynamic matching and virtual stop selection in on-demand ride-pooling, reporting considerable savings in both operator and user costs even under conservative scenarios favoring private car use. Collectively, these studies highlight the substantial efficiency gains achievable by incorporating passenger walking, however, they restrict walking to access and egress segments near the trip origin and destination.

In a broader context, related ideas appear in workforce transportation problems. Fikar and Hirsch (2015) and Coindreau et al. (2019) study the routing of nurses or service employees between patient visits or service tasks while allowing walking segments between consecutive locations. This setting differs from the classical DARP, as each request is defined by a sequence of required visits with associated time windows rather than a single pick-up and drop-off pair. Both studies report substantial performance improvements from integrating walking, particularly in dense urban environments.

Although the DARP literature has extensively examined transfers and passenger walking as mechanisms for improving operational efficiency, these features have been studied in

isolation. Transfer-based formulations typically limit passengers to a single transfer and assume fully vehicle-based movement, while walking-based approaches restrict walking to access and egress segments near trip endpoints. Consequently, the interaction between transfers and walking, and their combined impact on system design, remains largely unexplored.

In this work we introduce the Dial-A-Ride Problem with Transfers and Walking (DARPTW), a new DARP variant that integrates both multiple transfers and walking segments within a unified optimization framework. Passengers may transfer several times and may walk at different stages of their itinerary, including between transfer stations. Multiple transfers enable finer decomposition of trips, improving load balancing and reducing the service area assigned to each vehicle. Walking, in turns, creates opportunities to avoid inefficient detours and to exploit network shortcuts that are unavailable to vehicles because of directed or constrained road networks.

The contribution of this paper is fourfold. First, we introduce the Dial-A-Ride Problem with Transfers and Walking (DARPTW) and formulate it as a four-index mixed-integer linear programming model. Second, we develop a four-step heuristic capable of constructing feasible schedules for routes that include multiple transfers and intertwined passenger-vehicle connections arising from vehicle exchanges. Third, we extend an Adaptive Large Neighborhood Search (ALNS) framework by proposing a new similarity metric between user requests and adapting the insertion heuristic to explicitly account for both transfers and walking segments. Finally, we provide a new set of benchmark instances derived from real-world operational data from Bubble Dan, which are used to quantify the operational and service improvements enabled by integrating transfers and walking.

The remainder of the paper is organized as follows. In Section 2, we define the proposed variant of the DARP and formulate it as a mixed-integer linear programming model. Section 3 is devoted to the description and solution of the scheduling subproblem. Building on this, Section 4 introduces an Adaptive Large Neighborhood Search (ALNS) framework and complementary algorithms for solving the DARPTW. Computational experiments based on real-world data are presented in Section 5, and Section 6 concludes the paper with a discussion of results and directions for future research.

2. Problem Statement and Model Formulation

Consider a complete directed graph $G = (V, A)$, where V denotes the set of vertices and A the set of edges. Let $M = \{1, \dots, m\}$ be the set of vehicles, each stationed at an origin depot $O = \{o_1, \dots, o_m\}$ and required to return to a dedicated final destination depot $F = \{f_1, \dots, f_m\}$. Vehicles are heterogeneous, with capacities Q_k for each $k \in M$. These vehicles serve n users, each specifying a pick-up location $P = \{1, \dots, n\}$ and a drop-off location $D = \{n+1, \dots, 2n\}$. Each user request $r \in R = \{1, \dots, n\}$ has a size q_r, a time window $[e_r, l_r]$ representing the earliest and latest allowable start times, and a maximum journey time J_r to reach the destination.

We assume that walking and riding times, α_{ij} and β_{ij}, are known for every pair of nodes. In addition, each node $i \in V$ has a service time d_i, representing the time required to board users, this can be constant for uncrowded stations.

There are τ transfer stations where users may change vehicles, and each vehicle is allowed to visit a given station at most γ times within the planning horizon (PH). To model this, we create $2m\gamma\tau$ replica nodes, with two nodes per visit for each vehicle. The factor of two distinguishes between arrival and departure transfer nodes, allowing for optional waiting times. Consequently, each transfer station has $2m\gamma$ replica nodes, resulting in a total of $2m\gamma\tau$ nodes across all stations. Specifically, the set of transfer nodes associated with physical transfer ϕ is defined as $\mathcal{T}_\phi = \{2n + 2m\gamma(\phi-1) + 1, \dots, 2n + 2m\gamma\phi\}$, and the subset of nodes corresponding to the visits of vehicle k at transfer station ϕ is given by $\mathcal{T}'_{\phi k} = \{2n + 2m\gamma(\phi-1) + 2(k-1)\gamma + 2, \dots, 2n + 2m\gamma(\phi-1) + 2k\gamma\}$.

Additional constraints are included to enforce service quality requirements for each user. Each request $r \in R$is associated with a maximum number of transfers Θ_r, a maximum cumulative walking time W_r, and a maximum continuous walking time U_r within a single trip segment.

The model uses four binary decision variables to define routes. Specifically, x_{ij}^r indicates whether request r walks from node $i \in N$ to node $j \in N$; y_{ij}^k indicates whether vehicle $k \in M$ travels from node $i \in V$ to node $j \in V$; z_{ij}^{rk} indicates whether request r is transported by vehicle k between nodes i and j; and ε^{rk} equals 1 if request r uses vehicle k, representing the connection between users and vehicles. Finally, S_i denotes the scheduled start time of service at node $i \in V$. A summary of the DARPTW problem sets, parameters, and decision variables is provided in Table 1 and Table 2.

Table 1: Problem sets and parameters

Sets	
$\mathcal{R} = \{1, \dots, n\}$	set of requests
$\mathcal{M} = \{1, \dots, m\}$	set of vehicles
$\Phi = \{1, \dots, \tau\}$	set of physical transfers
$\mathcal{P} = \{1, \dots, n\}$	set of pick-up nodes
$\mathcal{D} = \{n+1, \dots, 2n\}$	set of drop-off nodes
$\mathcal{T} = \{2n+1, \dots, 2n+2m\gamma\tau\}$	set of transfer nodes
$\mathcal{N} = \mathcal{P} \cup \mathcal{D} \cup \mathcal{T}$	set of all possible nodes for passengers $\lvert\mathcal{N}\rvert = 2n + 2m\gamma\tau$
$\mathcal{O} = \{\lvert\mathcal{N}\rvert, \dots, \lvert\mathcal{N}\rvert + m\}$	set of origin depot nodes
$\mathcal{F} = \{\lvert\mathcal{N}\rvert + m + 1, \dots, \lvert\mathcal{N}\rvert + 2m\}$	set of destination depot nodes
$\mathcal{V} = \mathcal{N} \cup \mathcal{O} \cup \mathcal{F}$	set of all possible nodes for vehicles $\lvert\mathcal{V}\rvert = \lvert\mathcal{N}\rvert + 2m$
$\mathcal{T}_\phi$	set of transfer nodes associated with physical transfer ϕ
$\mathcal{T}'_{\phi k}$	set of transfer nodes corresponding to the visits of vehicle k at transfer station ϕ
Parameters	
n	number of requests
m	number of vehicles
τ	number of physical transfers
γ	maximum times a vehicle can visit a physical transfer
e_r	earliest time window of request $r \in R$
l_r	latest time window of request $r \in R$
q_r	size of request $r \in R$
J_r	maximum journey time of request $r \in R$
Θ_r	maximum transfers for request $r \in R$
W_r	maximum total walking time of request $r \in R$
U_r	maximum walking time per section of request $r \in R$
Q_k	maximum capacity of vehicle $k \in M$
d_i	duration of boarding time at node $i \in V$
α_{ij}	walking time between node $i \in N$ and $j \in N$
β_{ij}	riding time from node $i \in V$ to node $j \in V$
PH	the planning horizon

Table 2: Problem Decision Variables

Decision Variables	
x_{ij}^r	1 if request $r \in R$ walked from node $i \in N$ to node $j \in N$, 0 otherwise.
y_{ij}^k	1 if vehicle $k \in M$ traveled from node $i \in V$ to node $j \in V$, 0 otherwise.
z_{ij}^{rk}	1 if request $r \in R$ used vehicle $k \in M$ for travel from node $i \in N$ to node $j \in N$, 0 otherwise.
ε^{rk}	1 if request $r \in R$ used vehicle $k \in M$, 0 otherwise.
S_i	the schedule time when service begins at node $i \in V$
L_r	the journey time of user $r \in R$

A 4-index formulation for the DARPTW, is defined as follows:

$$Minimize \quad \sum_{k\in M}\sum_{j\in V}\sum_{i\in V}\beta_{ij}\cdot y_{ij}^{k} \;+\; \eta\sum_{r\in R}L_r \tag{1}$$

Subject to

$$\sum_{i\in N}x_{p_r i}^{r}+\sum_{k\in M}\sum_{i\in N}z_{p_r i}^{rk}=1 \qquad \forall r\in R \tag{2}$$

$$\sum_{i\in N}x_{i,d_r}^{r}+\sum_{k\in M}\sum_{i\in N}z_{i,d_r}^{rk}=1 \qquad \forall r\in R \tag{3}$$

$$\sum_{i\in N}x_{ij}^{r}+\sum_{k\in M}\sum_{i\in N}z_{ij}^{rk}\le 1 \qquad \forall r\in R,\forall\, j\in N \tag{4}$$

$$\sum_{j\in N\setminus\mathcal{T}_\phi}\sum_{i\in\mathcal{T}_\phi}x_{ji}^{r}+\sum_{k\in M}\sum_{j\in N\setminus\mathcal{T}_\phi}\sum_{i\in\mathcal{T}_\phi}z_{ji}^{rk}\le 1 \qquad \forall r\in R,\forall\phi\in\Phi \tag{5}$$

$$\sum_{j\in N}x_{ij}^{r}+\sum_{k\in M}\sum_{j\in N}z_{ij}^{rk}-\sum_{j\in N}x_{ji}^{r}-\sum_{k\in M}\sum_{j\in N}z_{ji}^{rk}=0 \qquad \forall r\in R,\forall\, i\in N\setminus\{r,r+n\} \tag{6}$$

$$\sum_{j\in N\cup\{f_k\}}y_{o_k,j}^{k}=1 \qquad \forall k\in M \tag{7}$$

$$\sum_{i\in N\cup\{o_k\}}y_{i,f_k}^{k}=1 \qquad \forall k\in M \tag{8}$$

$$\sum_{j\in V}y_{ij}^{k}\le 1 \qquad \forall k\in M,\ \forall\, i\in N \tag{9}$$

$$\sum_{j\in V}y_{ij}^{k}-\sum_{j\in V}y_{ji}^{k}=0 \qquad \forall k\in M,\ \forall\, i\in N \tag{10}$$

$$\sum_{j\in N}y_{ji}^{k}\le\sum_{j\in N}y_{j,i-1}^{k} \qquad \forall k\in M,\ \forall\phi\in\Phi,\forall\, i\in\mathcal{T}_{\phi k} \tag{11}$$

$$z_{ij}^{rk}\le y_{ij}^{k} \qquad \forall r\in R,\forall k\in M,\forall i\in N,\forall j\in N \tag{12}$$

$$e_r\le\ S_{p_r}\le l_r \qquad \forall r\in R \tag{13}$$

$$S_i+\alpha_{ij}-PH\cdot\left(1-x_{ij}^{r}\right)\le S_j-d_j \qquad \forall r\in R,\forall i\in N\setminus\{r+n\},\ \forall j\in N\setminus\{r\} \tag{14}$$

$$S_i+\beta_{ij}-PH\cdot\left(1-y_{ij}^{k}\right)\le S_j-d_j \qquad \forall k\in M,\forall i\in N\cup\{o_k\},\ \forall j\in N\cup\{f_k\} \tag{15}$$

$$S_{i-1}-PH\cdot\left(1-\sum_{j\in V}y_{ji}^{k}\right)\le S_i-d_i \qquad \forall k\in M,\ \forall\phi\in\Phi,\forall\, i\in\mathcal{T}_{\phi k} \tag{16}$$

$$S_i-PH\cdot\left(\sum_{k\in M}\sum_{j\in V}y_{ij}^{k}+\sum_{r\in R}\sum_{j\in N}x_{ij}^{r}\right)\le 0 \qquad \forall\, i\in\mathcal{T} \tag{17}$$

$$\sum_{\forall i\in N}\sum_{j\in N}x_{ij}^{r}\cdot\alpha_{ij}\le W_r \qquad \forall r\in R \tag{18}$$

$$\sum_{j\in N}x_{ij}^{r}\cdot\alpha_{ij}\le U_r \qquad \forall r\in R,\ \forall i\in N \tag{19}$$

$$L_r = \left(S_{d_r} - d_{d_r}\right) - \left(S_{p_r} - d_{p_r}\left(1 - \sum_{j \in N} x^r_{p_r j}\right)\right) \qquad \forall r \in R \qquad (20)$$

$$L_r \leq J_r \qquad \forall r \in R \qquad (21)$$

$$\sum_{k \in M} \varepsilon^{rk} \leq \Theta_r + 1 \qquad \forall r \in R \qquad (22)$$

$$z^{rk}_{ij} \leq \varepsilon^{rk} \qquad \forall i \in N, \forall j \in N, \forall k \in M, \forall r \in R \qquad (23)$$

$$\sum_{r \in R} z^{rk}_{ij} \cdot q_r \leq Q_k \qquad \forall k \in M,\ \forall i \in N,\ \forall j \in N \qquad (24)$$

$$0 \leq S_i \leq PH \qquad \forall i \in V \qquad (25)$$

The objective function (1) is defined as a weighted sum of the total travel time of all vehicles and the total journey time of all users. The first component captures the operational costs incurred by the fleet, whereas the second reflects the users' quality of service. Our aim is to examine the impact of transfers and walking on the overall objective. On the one hand, allowing transfers and walking may reduce perceived service quality from the users' perspective. On the other hand, these features can shorten total user travel times and lower vehicle operating costs, thereby potentially improving the overall system performance as measured by the objective value.

Constraints (2)–(6) ensure the feasibility of all user routes. Constraints (2)–(3) require each user to depart from its origin and arrive at its destination exactly once, either by vehicle or on foot. Flow conservation is enforced by constraints (4)–(6): constraint (4) limits each user to visiting any node at most once, constraint (5) restricts each user to visiting a physical transfer node at most once, and constraint (6) guarantees that, for every intermediate node, if a user arrives at a node it must also depart from it.

Constraints (7)–(12) establish the feasibility of the vehicle routes. Constraint (7) ensures that each vehicle departs from its origin depot exactly once, while constraint (8) requires each vehicle to reach its designated destination depot exactly once. Vehicle flow conservation is enforced by constraints (9)–(10): constraint (9) limits each vehicle to visiting any node at most once, and constraint (10) guarantees that a vehicle must depart from every intermediate node it enters. Because of the transfer-node replication used in the graph construction, vehicle visits to transfer nodes must be ordered. Without loss of generality, we impose that a vehicle may visit a given replica only if it has visited all preceding replicas of the same transfer node, this ordering is enforced by constraint (11). Constraint (12) links user and vehicle movements by

ensuring that a user can traverse an arc with a specific vehicle only if that vehicle itself uses the same arc.

Timing constraints, given in inequalities (13)–(17), define service start times and validate walking and riding durations. Waiting times are not explicitly modeled by decision variables, however, waiting may occur at any node and can be recovered in post-processing provided all timing constraints are satisfied. Constraint (13) requires each user to arrive at its origin within the time window $[e_r, l_r]$. Constraint (14) ensures that if a user walks between two nodes, the difference between the corresponding arrival times is at least the walking time. Similarly, constraint (15) ensures that if a vehicle travels between two nodes, the difference between the corresponding departure times is at least the sum of the travel time and the service time. In addition to constraint (11), which determines which replicas of a physical transfer node are activated, constraint (16) enforces that these replicas are visited in an ascending order of time.

Service and capacity constraints further restrict the solution space to respect operational and user-specific limitations. Constraints (18)–(19) impose upper bounds on the total and single walking times for each user, while constraints (20)–(21) define the total journey times of the users and restrict them to the predefined maximal journey times. Constraints (22) restricts the number of transfers per user by ensuring that the number of vehicles used does not exceed the allowed maximum number of transfers plus one. Constraints (23) links assignment and routing decisions by enforcing that a user can ride with a vehicle between two nodes only if that vehicle traverses the corresponding arc in the solution. Capacity feasibility is guaranteed by constraint (24), which ensures that the number of passengers carried by a vehicle on any arc never exceeds its capacity. Finally, constraint (25) bounds all schedule times, including user pick-ups and drop-offs and vehicle departures and arrivals, within the planning horizon.

The formulation highlights the inherent trade-off between operator efficiency and user experience. From the operator's perspective, the objective is to minimize total vehicle operating time and, consequently, overall system cost while serving all users. From the users' perspective, the goal is to reduce total journey time. Allowing transfers and walking introduces additional flexibility that may shorten travel times, even if such options are less convenient than direct rides, provided that individual service limits are respected. As the DARPTW generalizes the DARP, it also belongs to the class of NP-Hard problems.

3. The Scheduling Subproblem

The introduction of transfers and walking substantially enlarges the solution space, rendering even small instances difficult to solve to optimality using exact methods. We therefore develop a heuristic framework to address the problem. Our approach decomposes the decision process into two interrelated components: scheduling decisions and routing decisions. In this section, we focus on the scheduling subproblem and present a dedicated solution method, which is later embedded as a subroutine within the ALNS algorithm developed for the DARPTW, as described in Section 4.

In the scheduling subproblem, the routing plan is assumed to be fixed, that is, the sequence of locations visited by each vehicle and the passengers' itineraries are given. The objective is to construct a feasible schedule that minimizes total passenger travel time, thereby improving the service-quality component of the objective function. Because this subproblem must be solved repeatedly within the overall heuristic, we first test whether a feasible schedule exists and only then attempt to optimize it.

Scheduling feasibility has been widely studied in the DARP literature, and several solution approaches have been proposed (e.g., Hunsaker and Savelsbergh, 2002; Parragh et al., 2010). However, in the DARPT setting, transfers substantially complicate feasibility verification (Masson et al., 2014). Transfers introduce strong interdependencies between vehicle routes, hence a modification in one route may propagate to others through shared transfer nodes. To address this challenge, we adopt a feasibility-checking framework similar to those proposed in prior transfer-enabled DARP studies, ensuring an exact feasibility assessment, while employing a dedicated algorithm of our own design to construct high-quality schedules.

The scheduling subproblem can be formulated as a Simple Temporal Problem (STP), a special case of the Temporal Constraint Satisfaction Problem introduced by Dechter et al. (1991). An STP consists of a set of time-point variables T, representing events, each defined over a continuous domain. The model includes a set of temporal difference constraints, where each constraint specifies a relationship between two time points. A constraint takes the form $T_i - T_j \in [a_n, b_n]$, where $[a_n, b_n]$ denotes the interval associated with the n-th constraint.

From these constraints, a distance graph $G' = (V', A')$ can be constructed, in which each time point corresponds to a vertex and each constraint is represented by directed arcs encoding upper bounds on time differences. It is standard to introduce a dummy vertex with variable $X_0 = 0$. The STP can then be written in canonical form as $T_i - T_j \leq d_{ij} \forall (i,j) \in A', T_i \geq 0 \forall i \in V'$, where d_{ij} denotes the upper bound induced by the constraint set.

An STP is consistent if and only if the associated distance graph contains no negative cycles. Consistency guarantees the existence of at least one feasible assignment of time values, i.e., a valid schedule. Detecting negative cycles can be performed using the Richard Bellman–Lester Ford algorithm, which runs in $O(|V'||A'|)$ time. Consequently, the feasibility of the scheduling subproblem can be determined in polynomial time with complexity $O(|V'||A'|)$.

In our setting, the scheduling subproblem can be viewed as a restricted version of the DARPTW formulation presented above. Once the routing plan is fixed, the binary decision variables x_{ij}^r, y_{ij}^k, z_{ij}^{rk}, and ε^{rk} are known parameters rather than variables. The model therefore reduces to the following scheduling formulation, in which only the temporal variables remain to be determined.

$$Minimize \quad \sum_{r \in R} L_r \tag{26}$$

Subject to

$$S_i + \alpha_{ij} \leq S_j - d_j \qquad \forall i,j \in \mathcal{N}\ 1 \tag{27}$$

$$S_i + \beta_{ij} \leq S_j - d_j \qquad \forall i,j \in \mathcal{V} \tag{28}$$

$$e_r \leq S_{p_r} \leq l_r \qquad \forall r \in R \tag{29}$$

$$S_{d_r} - S_{p_r} \leq J_r \qquad \forall r \in R \tag{30}$$

After rewriting inequalities (27)–(30) in the canonical STP form described above, we obtain constraints (31)–(37), which define the arcs of the corresponding distance graph. Feasibility of the scheduling problem is then verified by applying the Bellman–Ford algorithm to detect negative cycles. The absence of such cycles guarantees the existence of a feasible schedule.

$$S_i - S_j \leq -\alpha_{ij} - d_j \qquad \forall i,j \in \mathcal{N} \tag{31}$$

$$S_i - S_j \leq -\beta_{ij} - d_j \qquad \forall i,j \in \mathcal{V} \tag{32}$$

$$S_0 - S_{p_r} \leq -e_r \qquad \forall r \in \mathcal{R} \tag{33}$$

$$S_{p_r} - S_0 \leq l_r \qquad \forall r \in \mathcal{R} \tag{34}$$

$$S_{d_r} - S_{p_r} \leq J_r \qquad \forall r \in \mathcal{R} \tag{35}$$

$$S_i \geq 0 \qquad \forall i \in \mathcal{V} \tag{36}$$

$$S_0 = 0 \tag{37}$$

Finding an optimal schedule for all users and vehicles remains computationally challenging even when feasibility is guaranteed. Although an exact optimization model could be used to compute an optimal schedule, repeatedly solving such models would be prohibitively expensive within the overall heuristic. While the feasibility checking step already eliminates many infeasible routing solutions, additional computational savings in the scheduling phase translate directly into substantial runtime improvements for the full algorithm. We therefore propose a four-stage polynomial-time heuristic designed to produce high-quality schedules efficiently.

Stage 1 – Forward update. The first stage constructs a greedy schedule using a forward propagation rule similar to the procedure introduced by Savelsbergh (1992). Routes are processed from start to end, and each vehicle or user departs to its next node at the earliest feasible time. No deliberate waiting is introduced at this stage. For example, if a vehicle serves multiple users at a transfer station, it departs as soon as the last relevant user arrives. The result is a feasible earliest-start schedule whenever one exists under purely forward propagation.

Stage 2 – Backward update. The second stage performs a backward pass to reduce unnecessary early departures. Routes are processed from end to start, and departure times are shifted to the latest feasible values that preserve downstream feasibility. This postponement reduces users' total journey times and may repair schedules that were infeasible after Stage 1 by redistributing slack along the routes.

Stage 3 – Relaxed finish-time backward update. Fixing vehicle completion times to their Stage 1 values may prevent certain users from reaching their destinations within their time windows. To address this, Stage 3 repeats the backward update while allowing vehicle finish times to move to their latest feasible values. This controlled relaxation can create feasible schedules by reallocating slack to late users or to users whose trips depend on shared route segments, thereby improving both feasibility and service quality.

Stage 4 – Slack-adjusted backward update. The relaxation introduced in Stage 3 may create excessive waiting that is not operationally necessary. In the final stage, we compute, for each vehicle route, the minimal excess waiting (route slack) and perform another backward update in which vehicle finish times are reduced by the corresponding slack amount. The slack is determined by propagating excess times from the end of the route to the beginning, where the excess at each node equals the minimum of (i) the time difference between the Stage 1 and Stage 3 schedules at that node and (ii) the excess propagated from its successor nodes. This stage removes superfluous waiting while preserving feasibility.

The final schedule is selected as the best feasible outcome among the four stages. Some intermediate schedules may be infeasible, and only valid schedules are considered in the final comparison. If none of the stages produces a feasible result, the algorithm reports that no schedule could be constructed for the given routing plan. Because time is modeled as continuous, multiple schedules may achieve identical objective values while being uniformly shifted in time, such equivalent schedules are treated as indistinguishable by the heuristic. An overview of the algorithm is presented in Table 3.

Table 3: Scheduling algorithm - four-stage heuristic for constructing a feasible schedule from a given route

Algorithm 1 - *find_schedule*

function $find\_schedule\ (G)$:
  S1 ← $forward\_update(G)$
  S2 ← $backward\_update(\mathrm{G}, \text{vehicles finish times}(\mathrm{S1}))$
  S3 ← $backward\_update(\mathrm{G}, \mathrm{PH})$
  S4 ← $backward\_update(\mathrm{G}, \mathrm{PH} - slack\_times(\mathrm{S1}, \mathrm{S3}))$
  if $all(\mathrm{S1}, \mathrm{S2}, \mathrm{S3}, \mathrm{S4})$ not valid:
    return False
  return $best\_of\,(S1, S2, S3, S4)$

function $forward\_update(\mathrm{G})$:
  Times[vehicles origin depots(and walking users pickups)] ← 0
  while not all visited:
    ind ← earliest unvisited node which all former nodes have arrived and mark as visited
    Times[ind] ←
earliest time for departure that everyone had arrived from their former nodes
  update Times[pickups] to the latest possbile according to the TW
  return Times

function $backward\_update(\mathrm{G}, \text{times})$:
  Times[vehicles destination depots(and walking users dropoffs)] ← times
  while not all visited:
    ind ← latest unvisited node and mark as visited
    Times[ind] ←
latest time for departure that everyone would arrive to their subsequent nodes
  update Times[dropoffs] to the earliest possbile according to the TW and Journey time
  return Times

function $slack\_times(\mathrm{S1}, \mathrm{S3})$:
  for each node from latest to ealiest:
    Excess[node] ← minimun (S3(node) − S1(node),
minimum(all excess times of all connected nodes))
  ST[vehicle] ← Excess[first dropoff vehicle has visited]
  return ST

4. An ALNS Algorithm

We develop an Adaptive Large Neighborhood Search (ALNS) algorithm to generate high-quality routing solutions for the DARPTW. ALNS is a metaheuristic that explores large neighborhoods by iteratively destroying and repairing portions of a solution. The concept of Large Neighborhood Search (LNS) was first introduced by Shaw (1998), demonstrating that selectively removing and reinserting parts of a solution allows the algorithm to escape local optima and explore a vast solution space. Building on this framework, Ropke & Pisinger (2006) introduced the adaptive variant ALNS, which is designed to learn over time which destroy and repair operators are most effective.

The algorithm begins with an initial solution, often empty with no users served, which is also considered the current best solution. Weights for the destroy and repair operators are initialized uniformly and guide operator selection via a roulette-wheel mechanism. In each iteration, a destroy operator partially dismantles the current solution, and a repair operator reconstructs a feasible solution, generating a new candidate. The new solution is evaluated and, if it satisfies predefined acceptance criteria, it replaces the current solution. Simulated Annealing controls acceptance by gradually reducing a "temperature" parameter, allowing occasional acceptance of worse solutions to escape local optima. If the new solution improves upon the best found so far, the best solution is updated. Operator weights are adjusted based on performance using the formula $w \leftarrow \lambda \cdot w + (1 - \lambda) \cdot \psi$, where λ is a decay parameter controlling sensitivity, and ψ is the score of the new solution. The process repeats until a stopping criterion is reached, in this case a maximum number of iterations (N), after which the best solution is returned.

Figure 1 schematically illustrates the overall ALNS procedure and its integration with the scheduling heuristic, while Table 4 provides an overview of the ALNS algorithm. For the DARPTW, each candidate route generated by the destroy-repair process is first checked for scheduling feasibility using a distance graph derived from the STP formulation and the Bellman-Ford algorithm. Only if a feasible schedule exists the four-stage heuristic is applied to construct a schedule, and the solution is evaluated.

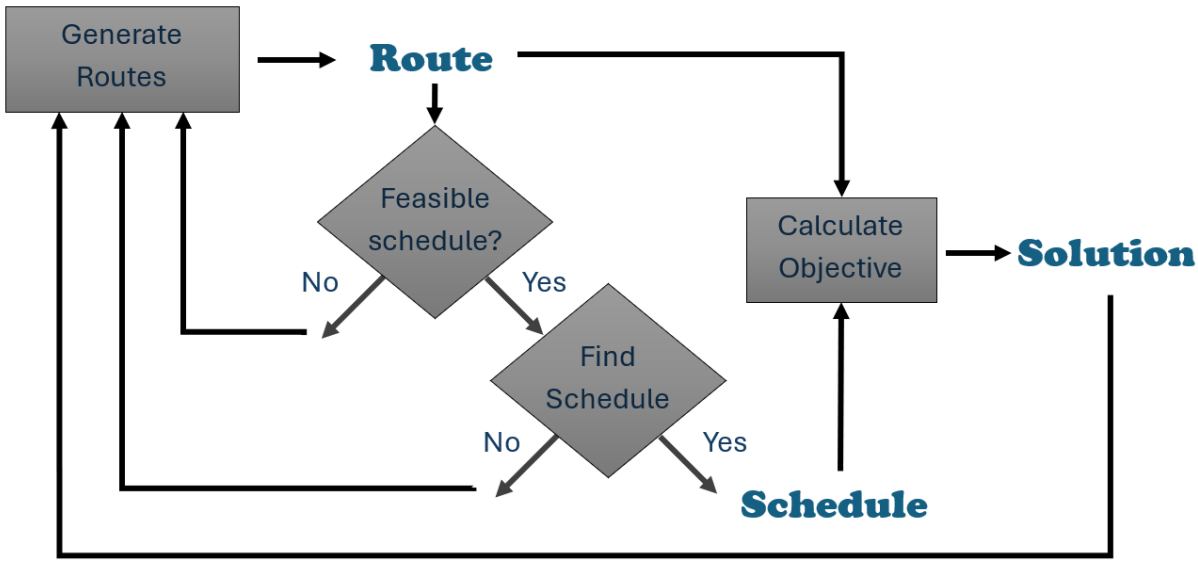

Figure 1: Illustration of the ALNS-based solution approach, including route generation, schedule feasibility check, and schedule construction

Table 4: The ALNS Algorithm

Algorithm 2 - *ALNS*
function *ALNS* (s):
$s^{best} \leftarrow s$
initiate destroy weights and repair weights
while not *stopping criteria* is met:
$destroy \leftarrow$ *select operator*(destroy weights)
$repair \leftarrow$ *select operator*(repair weights)
$s^{new} \leftarrow repair(destroy(s))$
if *acceptance criteria* (s^{new}, s) is met:
$s \leftarrow s^{new}$
if s^{new} is better than s^{best}:
$s^{best} \leftarrow s^{new}$
update destroy weights and repair weights
return s^{best}

For each destroy operator, we first determine the number of users to remove from the current solution. This quantity, named as the destruction rate, is drawn from a uniform distribution, $n_d \sim \text{Uniform}(\lfloor a \cdot n \rfloor, \lceil b \cdot n \rceil)$, where $0 < a < b < 1$. Once the removal size is fixed, we select the specific users to remove using one of the following operators. **Random removal** selects users sequentially with uniform probability until $n_d$users have been removed. **Transfer-based removal** repeatedly selects a transfer node at random and removes all users associated with that node. **Vehicle-based removal** repeatedly selects a vehicle at random and removes all users assigned to that vehicle.

In addition, we propose a new removal operator, termed **Similarity Removal**, which targets groups of highly related users. To quantify similarity, we define two complementary measures: temporal similarity and spatial similarity. Temporal similarity is computed by first defining each user's potential service interval as the span from the beginning to the end of their requested time window, extended by their maximum allowable journey time. The temporal similarity between two users is then defined as the ratio between the intersection and the union of their respective potential intervals, analogous to the Intersection-over-Union (IoU) metric. Spatial similarity captures the degree of trip overlap. For each pair of users, we compute the average direct trip length (i.e., the distance from pickup to drop-off for each user) and compare it to the shortest combined route that serves both users. In the case of two users with distinct pickup and drop-off locations, six possible visit sequences exist, the shortest of these defines the combined trip length. Spatial similarity is defined as the ratio between the average direct trip length and the shortest combined trip length.

The overall similarity score is calculated as the product of temporal and spatial similarities, preserving the range $[0, 1]$, where a value of 1 indicates maximal similarity in both dimensions. Figure 2 illustrates these concepts: the upper panel shows temporal similarity via the intersection and union of potential time windows (highlighted in blue), while the lower panel shows spatial similarity by comparing individual direct trips with the shortest combined route. The corresponding mathematical expressions appear in the bottom-right corner of the figure.

The destruction phase proceeds as follows: a user is selected at random, and the remaining $n_d - 1$ users with the highest similarity to this reference user are removed together with it. This operator encourages the joint removal of structurally related requests, enabling more effective restructuring of correlated portions of the solution.

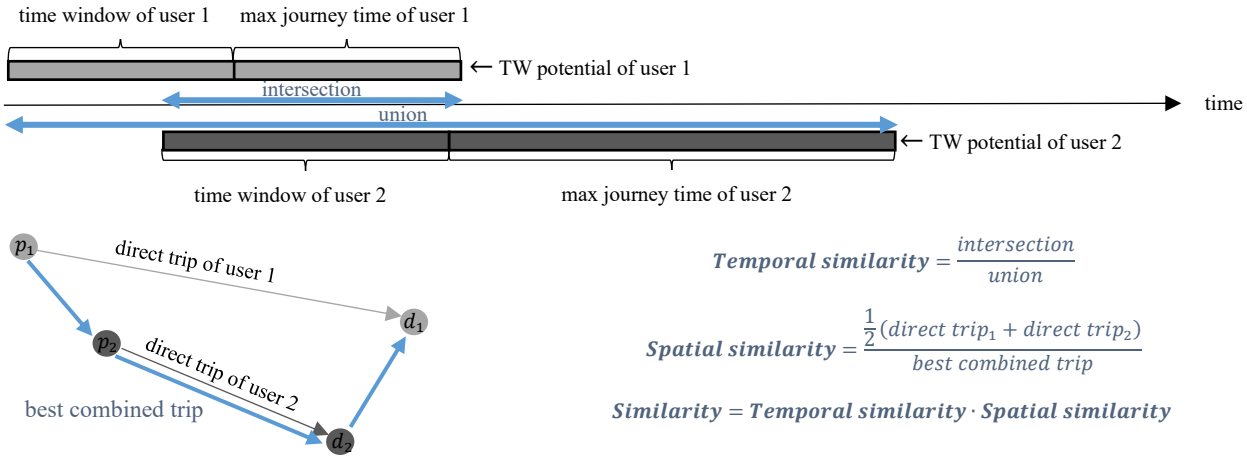

Figure 2: Similarity as a function of Temporal similarity (top) and Spatial Similarity (bottom left)

We employ several repair operators, each defining a different ordering mechanism according to which removed users are reinserted into the partial solution. Specifically, we consider the following strategies: **random ordering**, in which users are selected uniformly at random; **journey-time ordering**, in which users are inserted in ascending order of their direct journey time; **time-window ordering**, in which users are inserted in ascending order of time-window width; **earliest ordering**, in which users are inserted in ascending order of their time-window start; and **similarity ordering**, in which a reference user is selected at random and the remaining users are inserted in descending order of their similarity to that user.

Once the insertion order is fixed, the central challenge remains how to place each user back into the evolving solution. Exhaustively searching for the best insertion location is computationally expensive, whereas a purely greedy reconstruction is fast but may yield poor-quality solutions. We therefore adopt a balanced approach and develop a dedicated heuristic insertion procedure that trades off solution quality and runtime efficiency. For brevity and flow, we provide an overview of the heuristic here and defer the full details to Appendix A.

The insert_user function, presented in Table 5, reintegrates a user into a previously destroyed solution. It receives as input a user request and the current partial solution. First, the request is decomposed into journey segments using guided sampling over a set of recommended transfer nodes. Each segment corresponds to a portion of the trip, such as pickup-to-transfer, transfer-to-transfer, or transfer-to-dropoff. For each segment, the algorithm heuristically evaluates insertion weights that quantify the cost or benefit of inserting the corresponding node (pickup, transfer, or drop-off) between any pair of consecutive nodes in the current routes. Based on these weights, the procedure generates up to K candidate insertion patterns and filters out infeasible ones using the scheduling feasibility check. Finally, the best remaining insertion is selected, possibly including transfer and walking operations, yielding an updated repaired solution.

Table 5: Insertion Heuristic

Algorithm 3 - *insert_user*

```
function insert_user(user, destroyed_solution):
    user_segments ← choose beneficial transfers to split the user's route into segments
    for each segment in user_segments:
        riding_weights ← estimate the cost of inserting the segment into the destroyed_solution
        walking_weights ← estimate how much walking is recommended for the segment
    potential_insertions ← generate K potential insertions based on the weights
    best_insertion ← select the best from potential_insertions to insert into destroyed_solution
    new_solution ← add all user nodes into the destroyed_solution according to best_insertion
    return new_solution
```

5. Computational Experiments

In this section, we present a series of numerical experiments designed to evaluate the performance of the proposed methods. The experiments are conducted on a set of problem instances, whose construction is described in Section 5.1. The results are reported and analyzed in Section 5.2, where we examine the performance of the scheduling algorithm, the overall ALNS framework, and the impact of incorporating transfers and walking into the decision-making process.

5.1.Problem Instances

The problem instances were derived from real-world data provided by Bubble-Dan, a demand-responsive transit operator operating in Tel Aviv, Israel. The master dataset consists of 390 user requests with specified pickup and drop-off locations, 50 designated transfer stations, and 10 depots serving as vehicle origins and destinations. To approximate realistic urban travel conditions, all driving and walking times were computed using OpenRouteService based on OpenStreetMap data. To reflect heavy traffic congestion in the city center, a polygon was defined around central Tel Aviv, and any driving segment intersecting this area was penalized by multiplying its travel time by a factor of two.

Problem instances were generated by randomly sampling requests from the master dataset under a three-hour planning horizon (PH). Each sampled request was assigned a randomized time window (TW) with a starting time within the first hour, a duration between 30 and 60 minutes, and a latest arrival capped at 1.5 hours from the beginning of the horizon. Service quality parameters included a maximum journey time J of 60–90 minutes, a maximum number of transfers Θ between 1 and 2, and walking constraints of $W = 60$ minutes (total walking allowance) and $U = 20$ minutes (per-segment walking limit). A service time of $d = 30$ seconds was assumed at each boarding node, and the replica limit $\gamma$was set to 1 for simplicity. Vehicle capacities Qranged from 2 to 4 passengers, each user request had demand $q = 1$, and all vehicles were available throughout the entire planning horizon.

5.2.Results and Analysis

All computational experiments were conducted on an Intel® Xeon® Gold 6230 CPU @ 2.10GHz. The algorithms were implemented in Python, and CPLEX 12.10 was employed as the exact solver for benchmarking and validation. The ALNS metaheuristic incorporated a Simulated Annealing acceptance criterion with an initial temperature of $T_0 = 1000$ and a

geometric cooling rate of 0.99. For the adaptive operator-weight adjustment mechanism, we used a decay parameter $\lambda = 0.95$ and score updates $\psi = [5,2,1,0.95]$, corresponding to new global best, improved, accepted, and rejected solutions, respectively. To discourage request rejection, a penalty cost of 100 was added to the objective function for each unserved user.

The first experiment aimed to evaluate the performance of the scheduling sub-problem. It was conducted within the ALNS metaheuristic framework on instances with a small number of users ($n = 5$) and a larger number of users ($n = 50$). In each ALNS iteration, a complete route was generated, recorded, and subsequently processed in parallel by two approaches: an exact solution using CPLEX and our proposed two-stage scheduling algorithm. The two-stage method first applies a feasibility check using the Bellman–Ford algorithm on the corresponding scheduling graph and, if feasibility is confirmed, constructs a schedule using the *find_schedule* heuristic. For both methods, we recorded CPU time (in milliseconds) and scheduling quality metrics across all generated routes.

Table 6 indicates that both methods achieved perfect accuracy in determining schedule feasibility. However, the Bellman–Ford feasibility check was substantially faster than CPLEX: approximately 200× faster on small instances (0.49 ms vs. 97.74 ms) and 28× faster on larger instances (7.18 ms vs. 201.01 ms). The performance gap is even more pronounced for feasible routes, about 224× faster for $n = 5$ (0.44 ms vs. 98.53 ms) and 82× faster for $n = 50$ (2.50 ms vs. 205.28 ms). This improvement is particularly important because feasible routes require an additional scheduling construction step. Overall, the lightweight Bellman–Ford feasibility checking proves highly suitable for real-time routing environments, where rapid feasibility assessments are required across instances of varying sizes.

Table 6: : Route feasibility performance comparing CPLEX and $feasibility\_check$ via Bellman-Ford.

Instances size	# Instances	Metric	CPLEX	$feasibility$
n=5	105,362	Average solution time [ms]	97.74	0.49
		Unfeasible time (21%) [ms]	94.66	0.67
		Feasible time (79%) [ms]	98.53	0.44
		Correctness	100%	100%
n=50	47,908	Average solution time [ms]	201.01	7.18
		Unfeasible time (67%) [ms]	198.93	9.46
		Feasible time (33%) [ms]	205.28	2.50
		Correctness	100%	100%

In Table 7, we focus the comparison on instances in which a feasible solution was found. As can be observed, the proposed heuristic constructs complete schedules rapidly (13.73 ms for $n = 5$ and 28.86 ms for $n = 50$), achieving approximately a 7× speedup compared to CPLEX for both instance sizes. On small instances, the heuristic attains the exact optimum in 94.27% of cases, with an average optimality gap of 8.12%, and a valid schedule is found in 99.78% of feasible routes. For larger instances, the exact-optimal rate decreases to 20.93%, however, the mean optimality gap narrows to 1.31%, and a valid schedule is found in 95.64% of feasible routes. Overall, the two-stage approach demonstrates a strong balance between solution quality and computational efficiency across scales. By integrating ALNS with a fast Bellman–Ford feasibility check and a near-optimal scheduling heuristic, the framework evaluates substantially more routes per unit time than exact optimization while maintaining practically near-optimal performance, particularly as instance size increases.

Table 7: Route scheduling performance comparing CPLEX and *find_schedule*, only when such schedule exists.

Instance size	Metric	CPLEX	*find_schedule*
n=5	Average solution time [ms]	98.53	13.73
	Average optimality GAP	---	8.12%
	Found optimal solution	100%	94.27%
	Found a feasible solution	100%	99.78%
n=50	Average solution time [ms]	205.28	28.86
	Average optimality GAP	---	1.31%
	Found optimal solution	100%	20.93%
	Found a feasible solution	100%	95.64%

The second experiment was designed to evaluate the performance of the proposed ALNS framework. Prior to the main experiments, we conducted a preliminary calibration study to determine suitable default values for the algorithm's hyperparameters. Based on this analysis, we set $K = 30$ and $N = 500$, adopted a destruction rate of 10–30% (reduced to 5–20% for larger instances), and employed an adaptive combination of repair orderings: journey, random, similarity, and time-window, allowing the ALNS to alternate dynamically among them during the insertion phase, while excluding the earliest ordering. This configuration achieves a balanced trade-off between diversification and intensification, maintains competitive runtimes, and demonstrated robust performance across the calibration set. Further details of the calibration procedure are provided in Appendix B.

We conducted a series of experiments on a diverse set of problem instances characterized by varying numbers of users (n), transfer stations (τ), and available vehicles (m). For each unique combination of (n, τ, m), 50 randomized instances were generated, each representing a distinct DARPTW scenario. The ALNS algorithm was executed 10 times per instance to reflect a realistic competitive setting and was benchmarked against the CPLEX solver. Both ALNS and CPLEX were evaluated under time constraints: CPLEX was allowed a maximum runtime of one hour per instance, whereas ALNS completed all runs in substantially less time, typically within a few minutes due to the predefined limit on the number of iterations.

In Table 8, we summarize the results of this experiment. The first three columns report the instance characteristics. Columns 4–6 indicate, out of 50 instances per configuration (n, τ, m), the number of cases in which CPLEX found a feasible solution, the number in which it proved optimality, and the number in which ALNS found a feasible solution. Columns 7 and 8 report the ALNS performance relative to the instances with a proven optimal solution, namely the number of times ALNS identified the optimal solution and the average optimality gap for the remaining instances. The following column presents the number of cases, out of 50, in which ALNS obtained a better feasible solution than CPLEX, and the final two columns report the average runtimes in seconds for CPLEX and ALNS, respectively.

As can be observed, across all instance groups, ALNS consistently produced feasible solutions, achieving a 100% feasibility rate, whereas CPLEX struggled to identify feasible solutions for larger instances, particularly in groups with $n \geq 20$. This underscores the approach's reliability as problem size increases. For smaller instances, ALNS frequently matched the optimal solutions identified by CPLEX, exhibiting a high rate of exact agreement in objective value. As expected, this frequency declines as instance size grows. Nevertheless, ALNS remained highly competitive and often outperformed CPLEX in more challenging instances, especially when CPLEX failed to find any feasible solution. The average optimality gap of ALNS, computed only for instances in which CPLEX identified a solution, increased moderately with problem size. These findings indicate that ALNS not only closely approximates optimality in smaller cases but also maintains strong performance when CPLEX encounters computational or time limitations. Finally, the runtime analysis shows that ALNS required substantially less computational time across nearly all scenarios, except for very small instances in which CPLEX quickly identified optimal solutions.

Table 8: Numerical results for groups of 50 unique instances with numbers of users (n), number of transfer stations (τ), and number of vehicles (m).

n	τ	m	Cplex feasible	Cplex optimal	ALNS feasible	ALNS eq. optimal	Avg gap [%]	ALNS better than CPLEX	Avg ALNS time [s]	Avg Cplex time [s]
5	2	2	50	50	50	41	1.02	0	70.40	0.87
5	3	2	50	50	50	35	2.21	0	77.05	1.14
10	2	2	50	47	50	8	4.22	0	180.77	535.98
10	3	2	48	44	50	7	7.66	2	203.03	576.57
10	4	2	47	45	50	3	13.61	3	204.26	695.28
10	4	3	46	43	50	1	13.84	4	312.72	844.26
15	2	2	39	16	50	3	16.47	11	322.02	2792.36
15	2	3	34	6	50	0	22.41	16	420.29	3307.53
15	3	2	35	20	50	0	23.57	11	328.51	2337.98
15	3	3	38	9	50	0	26.73	12	233.89	3194.80
15	4	2	37	15	50	0	45.61	13	365.30	2668.75
15	4	3	34	10	50	0	37.88	16	270.21	3049.49
20	2	2	20	1	50	0	32.60	30	491.09	3563.08
20	2	3	6	0	50	0	27.78	44	397.71	3600.00
20	3	2	24	0	50	0	43.24	26	319.93	3600.00
20	3	3	15	2	50	0	34.27	35	422.46	3519.02
20	4	2	18	4	50	0	55.26	32	377.54	3435.18
20	4	3	4	2	50	0	54.33	46	494.10	3503.07
25	2	2	0	0	50	0	---	50	430.69	3600.00

Next, we conducted an experiment to assess the scalability of the proposed ALNS heuristic beyond the practical limits of exact methods. To this end, a set of large-scale DARPTW instances was generated with the number of users ranging from 30 to 80. For each selected problem size (n), a random combination of stations (τ) and vehicles (m) was assigned. The instances were constructed using realistic parameters, including time windows and service constraints, however, due to the inherent complexity of such large problems, feasibility could not be guaranteed for every generated instance. Each instance was solved using 10 independent ALNS runs, with success defined as the identification of a feasible solution within the prescribed time limit. Instances for which no feasible solution was found in any of the 10 runs

were discarded to avoid bias from potentially infeasible cases. For the remaining instances, the success rate out of 10 runs, standard deviation, and runtime statistics were recorded.

Figure 3 indicates a gradual decline in the success rate as n increases, while maintaining robust performance up to approximately $n = 50$, where feasible solutions were obtained in more than 8 out of 10 runs on average. For instances with $n \geq 60$, performance degradation becomes more pronounced, although a non-negligible share of instances remains solvable. The standard deviation increases moderately with instance size, reflecting greater variability in algorithmic performance for larger problems. As expected, runtime grows exponentially with problem size, due to the expanding solution space and the increased computational effort required. Despite this growth, ALNS was still able to produce feasible solutions within practical timeframes for instances with n = 70 and n = 80, although the results suggest that the algorithm is approaching its scalability limits as instance size continues to increase.

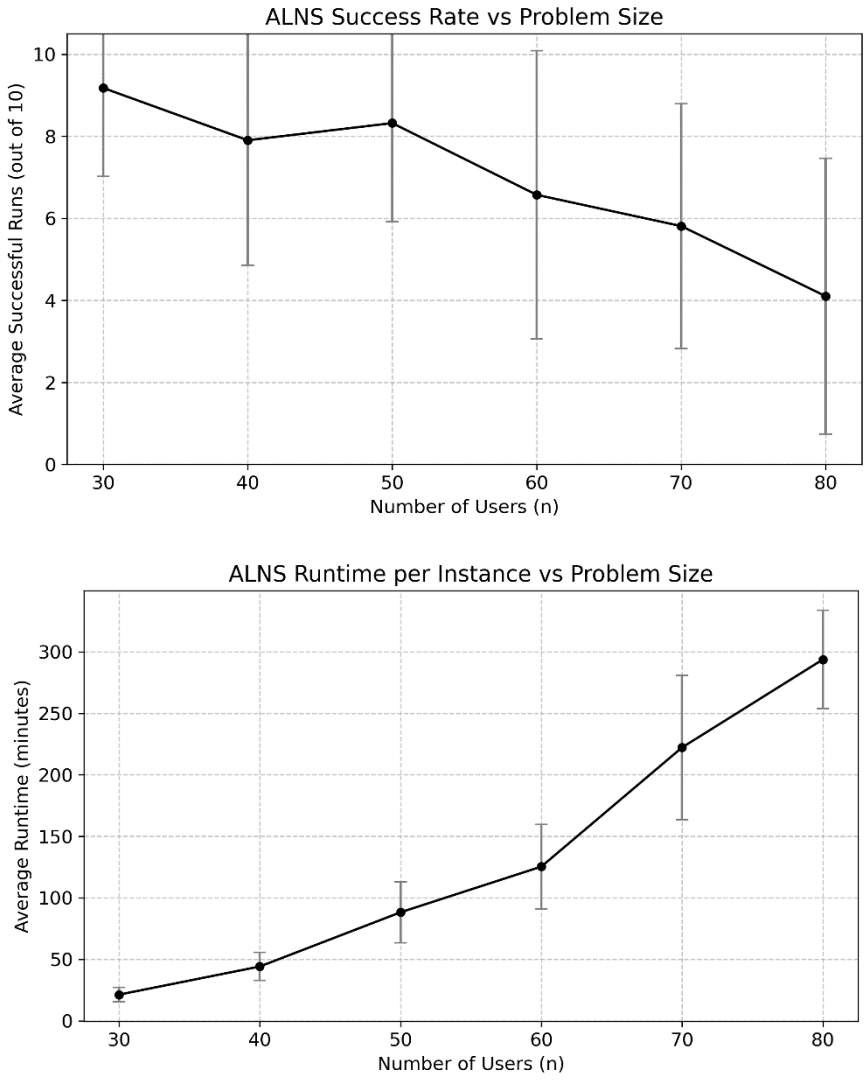

Figure 3: Average number of successful runs and average run times for varying number of users

The final experiment examined the impact of transfers on solution quality. We generated 100 instances with $n = 15$ users and, for each instance, evaluated multiple configurations defined by Θ, the maximum number of transfers, W, the total walking time, U, the maximum walking time per segment, and Q, the vehicle capacity. The objective was to assess whether allowing transfers with $\Theta \geq 1$ improves solution quality relative to the no-transfer case with $\Theta = 0$ under varying levels of walking flexibility and vehicle capacity. The baseline configuration used default parameters $W = 20$ minutes, $U = 5$ minutes, and $Q = 4$. In each comparison between allowing and prohibiting transfers, one parameter was modified at a time to isolate its effect. In Figure 4, we report for each configuration the number of valid comparisons, defined as cases in which at least one of the two runs produced a feasible solution, and the proportion of instances in which the transfer-enabled solution strictly improved the objective and included at least one actual transfer. In some cases, both runs achieved the same objective value without using transfers, and in others, the transfer-enabled configuration achieved a better objective even though the resulting solution did not include an actual transfer, since it generalizes the no-transfer setting.

Varying U highlights the strong benefit of transfers when walking flexibility is limited. $With\, U = 0$, transfers improved the objective in 21.8% of valid instances, demonstrating their value in resolving infeasibilities and enhancing routing efficiency. As U increases, the share of instances benefiting from transfers declines to 10.4% at $U = 5$ minutes and to zero at $U = 20$ minutes, since greater walking flexibility enables users to bridge spatial gaps without requiring vehicle detours. In contrast, varying W yields a relatively stable pattern, with improvements from transfers remaining around 12% across different values, while feasibility stays high. Once limited single-segment walking is allowed, increasing total walking time mainly enhances feasibility but does not substantially alter the structural trade-off. Varying Q reveals the clearest structural effect. When vehicle capacity is tight at $Q = 2$, transfers improve 21.6% of valid cases. This share decreases to 14.1% at $Q = 3$, 10.4% at $Q = 4$, and 6.1% at $Q = 5$. As capacity increases, the need for transfers diminishes because more requests can be accommodated within a single route. Feasibility also declines as capacity tightens, emphasizing that transfers are most valuable when resources are scarce and conflicts between requests are more frequent. Overall, the results indicate that transfers provide the greatest benefit under constrained conditions, particularly when walking flexibility is limited or vehicles operate near capacity. In such regimes, enabling transfers allows the formation of more cooperative and flexible route structures, improving both feasibility and objective value.

As walking flexibility or vehicle capacity increases, transfers remain feasible but become less critical, since comparable benefits can be achieved through simpler routing adjustments.

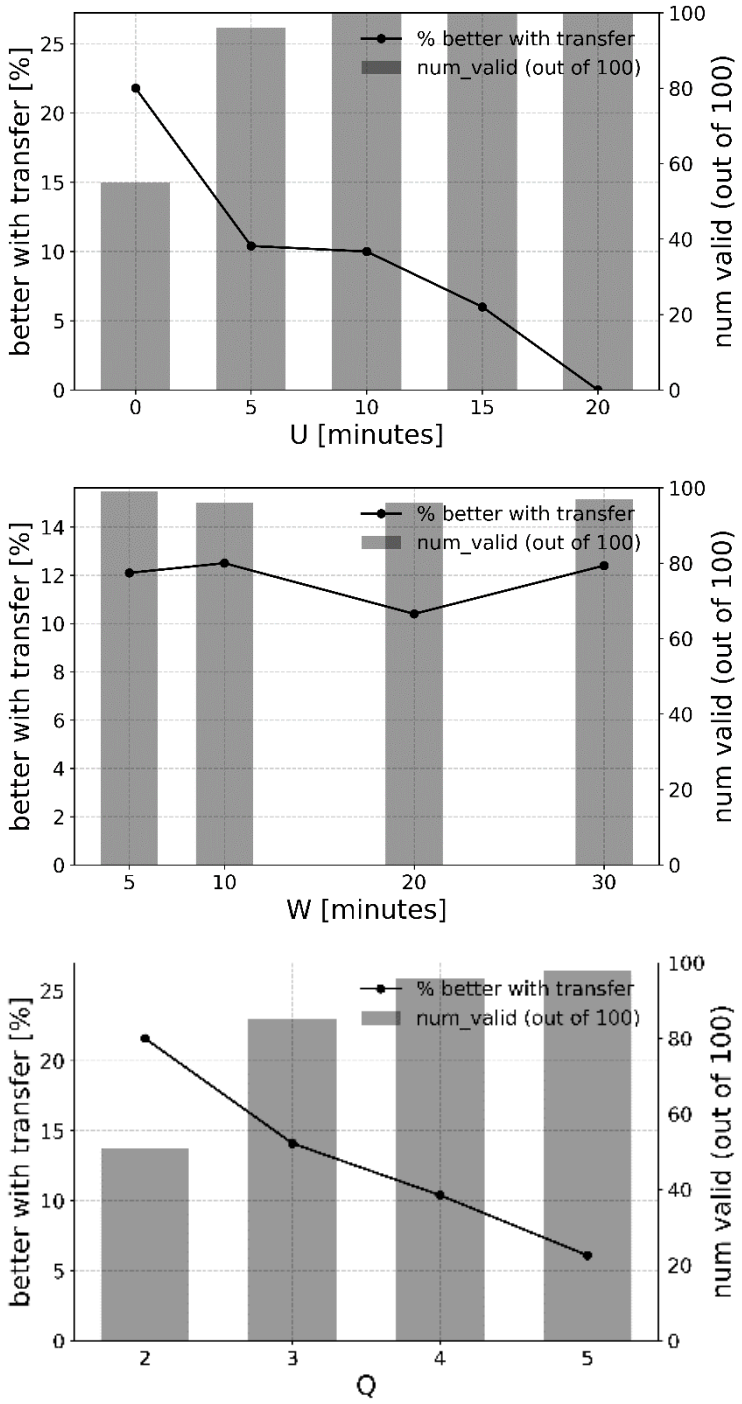

Figure 4: The impact of walk time constraints on each segment or the complete roune, and the vehicle capacity on the value of transfers

6. Conclusions

This paper introduced the Dial-A-Ride Problem with Transfers and Walking (DARPTW), extending the classical DARP by enabling coordinated vehicle transfers and bounded walking segments within passenger trips. We developed a four-index MILP formulation capturing time windows, capacity, transfer, and walking constraints, and proposed a two-stage scheduling mechanism combining a graph-based feasibility check with a constructive heuristic scheduler. These components were embedded within an Adaptive Large Neighborhood Search framework enhanced by transfer-aware operators and a dedicated request similarity measure. A realistic benchmark set was generated based on operational data from Bubble Dan.

The computational study demonstrates that the proposed feasibility check and scheduling modules are both accurate and computationally efficient, making them well suited for iterative or real-time routing contexts. At the full problem level, the ALNS approach consistently produced high-quality feasible solutions across diverse instance settings and remained robust as problem size increased, while exact methods rapidly became impractical. Scalability experiments indicate that the heuristic maintains strong performance well beyond the range tractable by exact optimization, although natural degradation appears as instance size grows due to the combinatorial expansion of the search space.

A dedicated analysis of transfer policies shows that transfers provide the greatest value under constrained operating conditions, particularly when walking flexibility or vehicle capacity is limited. In such settings, transfers act as a structural coordination mechanism that enhances feasibility and routing efficiency. As operational flexibility increases, the marginal benefit of transfers declines, suggesting that their primary contribution lies in mitigating tight resource interactions rather than universally improving all scenarios.

Overall, DARPTW offers a principled modeling framework and a practical algorithmic solution for jointly leveraging transfers and walking in demand-responsive systems. The results highlight the operational potential of controlled passenger flexibility to improve system performance while maintaining service guarantees, and open several avenues for future research, including dynamic extensions, infrastructure co-design, multi-objective considerations, and hybrid exact-heuristic scaling strategies.

References

1. Coindreau, M. A., Gallay, O., & Zufferey, N. (2019). Vehicle routing with transportable resources: Using carpooling and walking for on-site services. European Journal of Operational Research, 279(3), 996-1010.
2. Cordeau, J., & Laporte, G. (2003). The Dial-a-Ride Problem (DARP): Variants, modeling issues and algorithms. 4OR, 1(2).
3. Cordeau, J., & Laporte, G. (2007). The dial-a-ride problem: models and algorithms. Annals of Operation Research/Annals of Operations Research, 153(1), 29–46.
4. Cortés, C. E., Matamala, M., & Contardo, C. (2010). The pickup and delivery problem with transfers: Formulation and a branch-and-cut solution method. European Journal of Operational Research, 200(3), 711–724.
5. Dechter, R., Meiri, I., & Pearl, J. (1991). Temporal constraint networks. Artificial Intelligence, 49(1–3), 61–95.
6. Fielbaum, A., Bai, X., & Alonso–Mora, J. (2021). On-demand ridesharing with optimized pick-up and drop-off walking locations. Transportation Research. Part C, Emerging Technologies, 126, 103061.
7. Fikar, C., & Hirsch, P. (2015). A matheuristic for routing real-world home service transport systems facilitating walking. Journal of Cleaner Production, 105, 300-310.
8. Häll, C. H., Andersson, H., Lundgren, J. T., & Värbrand, P. (2008). The integrated Dial-a-Ride problem. Public Transport, 1(1), 39–54.
9. Hansen, T., & Sener, I. N. (2023). Strangers on this road we are on: a literature review of pooling in on-demand mobility services. *Transportation Research Record*, *2677*(3), 1368-1381.
10. Ho, S. C., Szeto, W. Y., Kuo, Y. H., Leung, J. M., Petering, M., & Tou, T. W. (2018). A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, *111*, 395-421.
11. Hunsaker, B., & Savelsbergh, M. (2002). Efficient feasibility testing for dial-a-ride problems. Operations Research Letters, 30(3), 169–173.
12. Jorgensen, R. M., Larsen, J., & Bergvinsdottir, K. B. (2007). Solving the dial-a-ride problem using genetic algorithms. Journal of the operational research society, 58(10), 1321-1331.

13. Lehuédé, F., Masson, R., Parragh, S. N., Péton, O., & Tricoire, F. (2014). A multi-criteria large neighbourhood search for the transportation of disabled people. Journal of the Operational Research Society, 65(7), 983-1000.
14. Masmoudi, M. A., Hosny, M., Demir, E., & Pesch, E. (2019). Hybrid adaptive large neighborhood search algorithm for the mixed fleet heterogeneous dial-a-ride problem. Journal of Heuristics, 26(1), 83–118.
15. Masson, R., Lehuédé, F., & Péton, O. (2014). The dial-a-ride problem with transfers. Computers & Operations Research, 41, 12-23.
16. Melis, L., & Sörensen, K. (2021). 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(3), 1417–1453.
17. Molenbruch, Y., Braekers, K., & Caris, A. (2017). Typology and literature review for dial-a-ride problems. Annals of Operations Research, 259(1-2), 295-325.
18. Parragh, S. N., Doerner, K. F., & Hartl, R. F. (2008). A survey on pickup and delivery problems. Journal für Betriebswirtschaft, 58(2), 81-117.
19. Parragh, S. N., Doerner, K. F., & Hartl, R. F. (2010b). Variable neighborhood search for the dial-a-ride problem. Computers & Operations Research, 37(6), 1129–1138.
20. Posada, M., Andersson, H., & Häll, C. H. (2016). The integrated dial-a-ride problem with timetabled fixed route service. Public Transport, 9(1–2), 217–241.
21. Røpke, S., & Pisinger, D. (2006). An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. Transportation Science, 40(4), 455–472.
22. Sarma, N. J., Gurumurthy, K. M., Hyland, M., Bahk, Y., De Souza, F., & Wang, Z. (2023, January 1). On-Demand Ride-Pooling with Walking Legs: Decomposition Approach for Dynamic Matching and Virtual Stops Selection. https://doi.org/10.2139/ssrn.4383040
23. Savelsbergh, M. W. P. (1992). The Vehicle Routing Problem with Time Windows: Minimizing Route Duration. ORSA Journal on Computing, 4(2), 146–154.
24. Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In Lecture notes in computer science (pp. 417–431).
25. Stein, D. M. (1978). Scheduling dial-a-ride transportation systems. Transportation Science, 12(3), 232-249.
26. Wilson, N. H., Sussman, J. M., Wong, H. K., & Higonnet, T. (1971). Scheduling algorithms for a dial-a-ride system. Massachusetts Institute of Technology. Urban Systems Laboratory.

27. Zheng, Y., Li, W., Qiu, F., & Wei, H. (2019). The benefits of introducing meeting points into flex-route transit services. Transportation Research Part C: Emerging Technologies, 106, 98-112.

Appendix A: Insertion Heuristic

In Section 4 we describe our ALNS-based solution framework for the DARPTW, including the mechanism for reinserting a user request into a partially destroyed solution. The heuristic insertion algorithm (Table 5) schematically illustrates how transfers and walking segments are combined when constructing a repaired solution. For clarity of exposition, the main text omits several implementation details. This appendix provides additional explanation of the insertion logic, without entering low-level technicalities, so that the reader can better understand how transfers and walking options are integrated.

The first stage of the insertion algorithm decomposes the candidate user request into journey segments. The original pickup-to-dropoff trip may be optionally split by inserting transfer stations, after which each segment can either be served by a vehicle or completed by walking. Candidate transfer stations are selected from those lying inside an ellipse whose foci are the pickup and dropoff locations and whose major axis is determined by a predefined radius. From this set, up to the user's maximum allowable number of transfers is sampled. When multiple transfers are selected, they are ordered to minimize the total segment length, consistent with the shortest-combined-trip principle used in the spatial similarity metric. This design balances two goals: prioritizing geographically efficient transfers that are likely to preserve feasibility, while maintaining stochasticity in the insertion process to diversify the search.

Once the sequence of nodes to be visited is fixed, the algorithm determines how each node should be inserted into the existing routes. For every node, two modalities are considered: (i) vehicle service, in which a detour is inserted into a vehicle segment, or (ii) walking, in which the user independently moves to the next node. For vehicle insertions, a heuristic weight is computed for placing a candidate node $cand$between consecutive nodes $i$and j:

$$w_{i,j,cand} = \frac{\text{Slack\_time}(i,j) - \text{Travel\_time}([i,j])}{\text{Travel\_time}([i,cand,j]) - \text{Travel\_time}([i,j])}.$$

Here, $\text{Slack\_time}(i,j) = l_j - e_i$ denotes the allowable temporal slack between nodes based on their time windows, and $Travel\_time(\cdot)$ is the total driving time along a route segment. The numerator measures residual slack, while the denominator captures the incremental detour caused by the insertion. Positions with large slack and small detour receive higher weights. Insertions that exceed the available slack are deemed infeasible and discarded.

For walking insertions, the arc (i,j) in the user's personal route is scored by

$$w_{i,j}^{r} = \frac{\text{Walking\_segment\_limit}(r)}{\text{Walking\_time}(i,j)}.$$

If the walking time exceeds the user's per-segment walking limit, the option is infeasible, otherwise, shorter walks relative to the allowance receive higher scores.

Vehicle and walking weights share a common budget-to-cost structure, allowing them to be compared in a unified stochastic draw. Ratios greater than one indicate feasible insertions that preserve slack, while larger values correspond to increasingly conservative resource usage. To ensure numerical stability, small denominators are regularized by adding $\varepsilon > 0$and weights are capped at a large constant without altering their relative ordering. Infeasible options are excluded from sampling. This matched construction biases the repair toward low-impact insertions while retaining randomness to promote search diversification.

Using these weights, the algorithm constructs candidate repaired solutions by inserting the nodes sequentially through weighted random sampling. Each tentative solution undergoes global feasibility validation, including the scheduling check. The process is repeated K times to generate multiple candidate repairs, after which the solution with the best objective value is selected. This multi-sample strategy leverages randomness to explore the search space while still enforcing a quality-driven selection step. The resulting solution defines the final insertion of the user, including any selected transfers and walking segments.

Appendix B: Hyper Parameter Calibration

Prior to the main experimental analysis, a dedicated hyperparameter calibration was conducted to identify the most effective settings for the ALNS heuristic on the DARP instances considered. The tuning focused primarily on two critical parameters: K, which controls the behavior of the repair mechanism, and N, which defines the total number of iterations performed. In addition, complementary mechanisms governing the search dynamics, specifically the proportion of users removed in each destruction phase and the policy used to order users for reinsertion, were also evaluated.

Calibration experiments were conducted on 50 representative instances with $n = 15$ users, $\tau = 4$ stations, and $m = 3$ vehicles. Benchmark solutions from CPLEX, computed under a one-hour time limit, served as a reference. Within this limit, CPLEX found feasible solutions for only 34 of the 50 instances and optimal solutions for just 10, so the average gap to CPLEX was computed over instances where feasible solutions were available.

As can be observed in Figures B1-B2, increasing K consistently improved solution quality, but gains became marginal beyond K = 30, while runtime increased sharply. Consequently, K = 30 was selected as the preferred setting, providing a balance between solution quality and computational efficiency. Similarly, raising N improved quality, but iteration counts above 1000 produced limited additional benefit while substantially increasing runtime. A setting of N = 500 was therefore chosen to ensure sufficient search depth without excessive computational cost.

The analysis indicates that increasing K improves solution quality up to a point of diminishing returns, beyond which runtime grows without meaningful gains. A similar pattern was observed for N, where moderate iteration limits achieved most of the attainable improvement while avoiding excessive computational cost.

Regarding the destruction rate, moderate removal proportions consistently outperformed both very small and very large perturbations. This intermediate range provides an effective balance between diversification and intensification, introducing sufficient structural change to escape local minima without excessively disrupting promising solution components. For larger instances, slightly smaller removal ranges are preferable to preserve runtime efficiency while maintaining diversification in absolute terms.

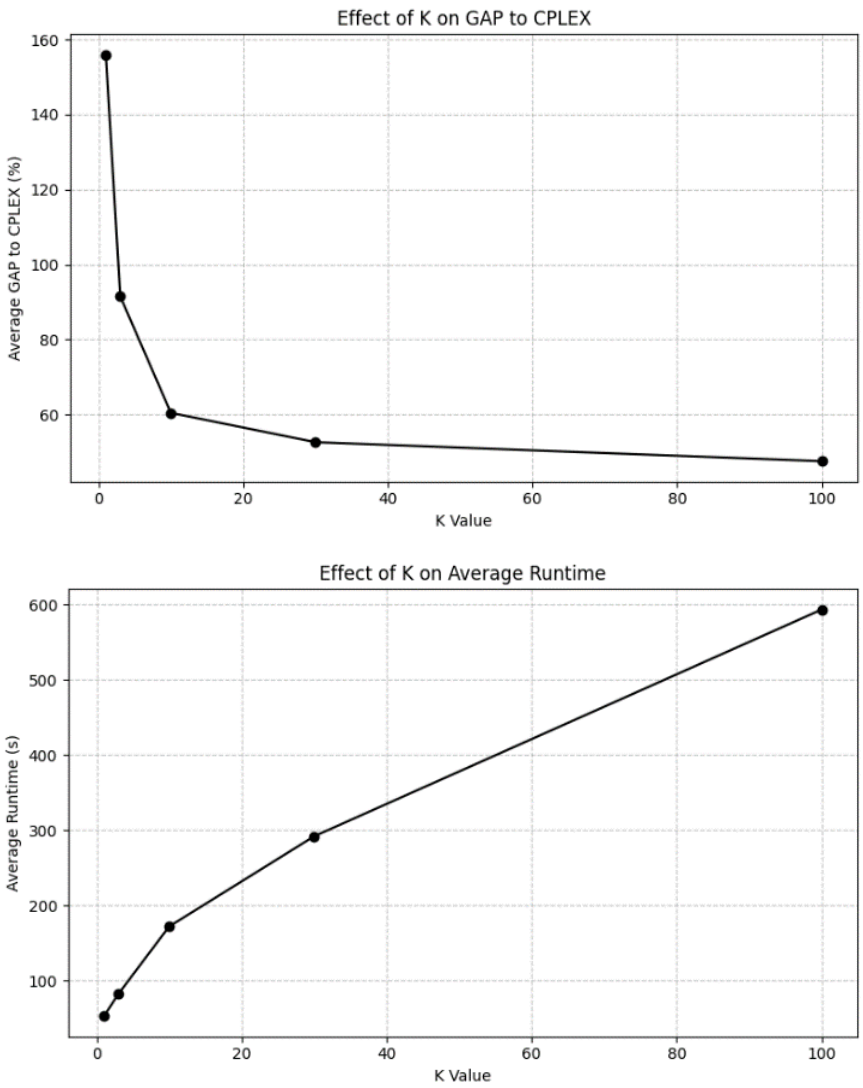

Figure B1: Average solution gap in percent from CPLEX and average runtimes as a function of the number of repair options (K)

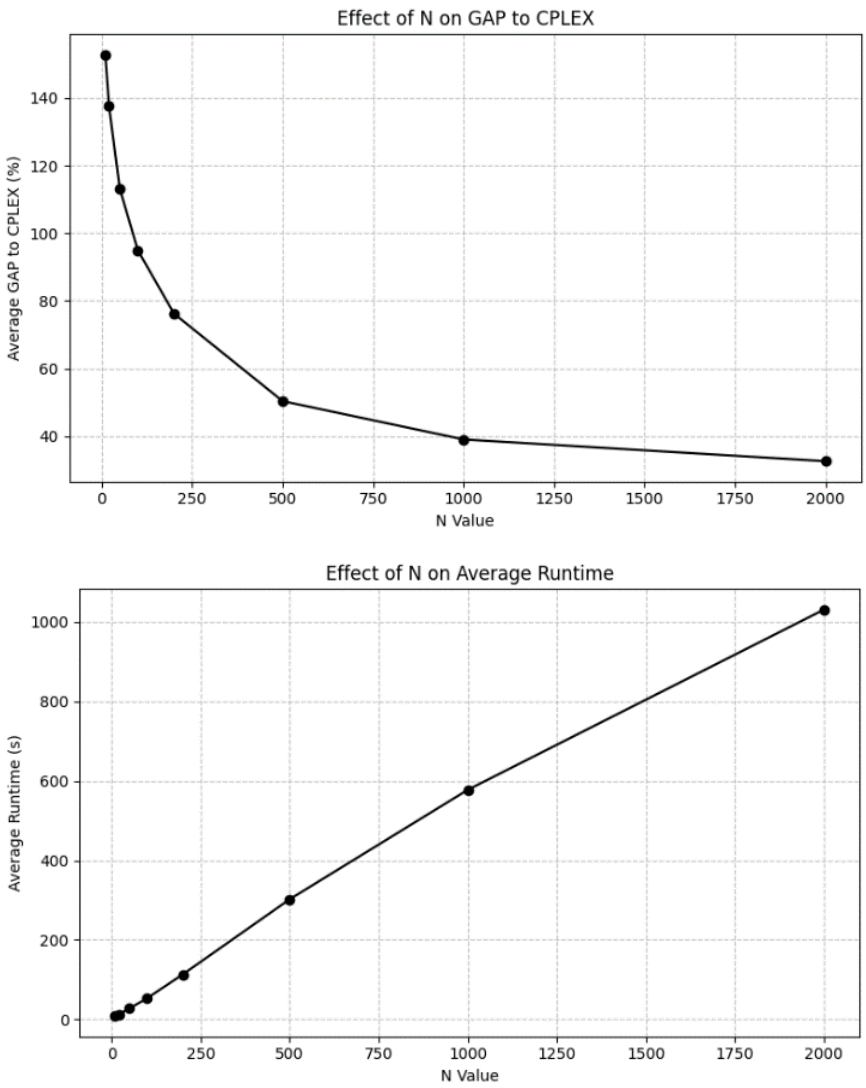

Figure B2: Average solution gap in percent from CPLEX and average runtimes as a function of the number of ALNS iterations (N)

In Figure B3 we present the results obtained while varying the proportion of users removed in each iteration. Moderate removal rates consistently produced the lowest average gap. Specifically, removing 10–30% of users per iteration achieved the best trade-off between diversification and intensification: small removals (5–10%) were shortsighted and large removals (20–50%) inflated runtime without improving quality. For larger instances, a slightly smaller range of 5–20% remains effective, as it still removes a substantial absolute number of users while reducing runtime.

In Figure B4, we compare different reinsertion orderings: earliest, journey, random, similarity, and time-window. Runtime differences were negligible, allowing quality to dominate the comparison. All strategies except earliest performed comparably, supporting the use of an adaptive mix of journey, random, similarity, and time-window orderings, which

enables the algorithm to exploit different insertion biases during the search. The earliest policy was excluded due to consistently weaker performance.

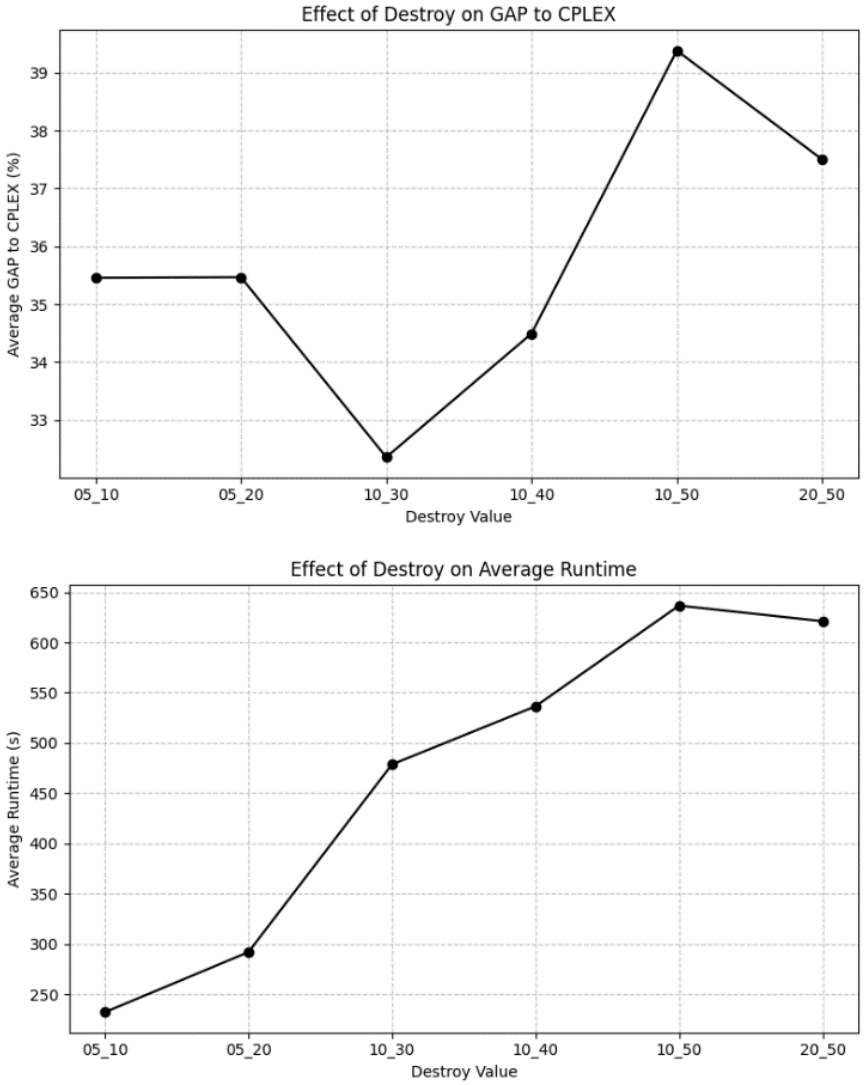

Figure B3: Average solution gap in percent from CPLEX and average runtimes as a function of removal percentage (n_d)

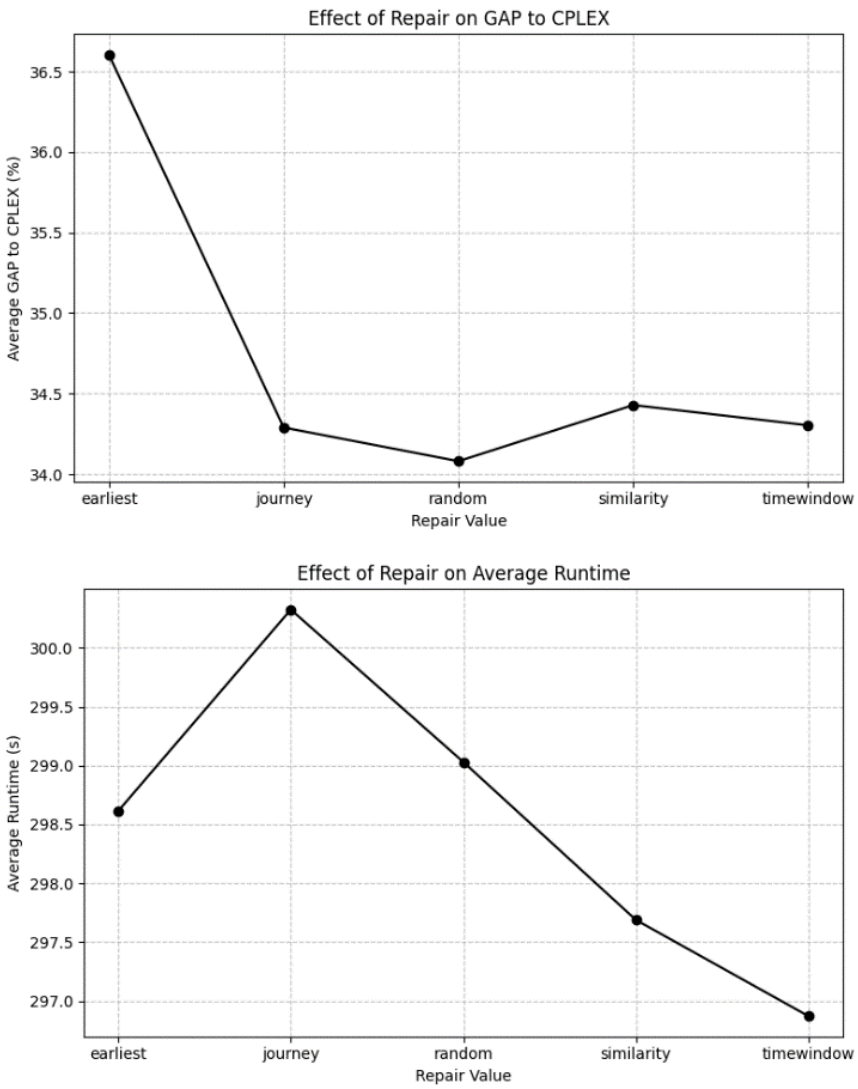

Figure B4: Average solution gap in percent from CPLEX and average runtimes for the different repair ordering rules

In summary, the calibration results identify K = 30, N = 500, a destruction interval of 10–30% (5–20% for larger instances), and an adaptive mix of journey, random, similarity, and time-window reinsertion policies as default settings for the ALNS heuristic. This configuration balances solution quality and runtime effectively and proved robust across the calibration suite, forming a solid foundation for all subsequent experiments.

תקציר

מאמר זה מציג את בעיית "הזמן וסע" (Dial-A-Ride) עם החלפות והליכה (בקיצור DARPTW), הכללה חדשנית של בעיית "הזמן וסע" הקלאסית המאפשרת למסלולי הנוסעים לכלול החלפות מרובות בין כלי רכב ומקטעי הליכה קצרים בכל שלב של נסיעתם. אתגרי הניידות העירונית, כגון זמני נסיעה ארוכים ותדירות שירות נמוכה, מחייבים שירותים יעילים מבוססי ביקוש שיכולים לאזן בין עלויות תפעוליות לבין איכות השירות למשתמש. אנו מנסחים את "הזמן וסע עם החלפות והליכה" כבעיית תכנון ליניארי בשלמים מעורבים (MILP) בעלת ארבעה אינדקסים, שמטרתה למזער פונקציית מטרה מרובת משתנים המשלבת את זמן הנסיעה הכולל של הרכבים ואת זמן הנסיעה הכולל של המשתמשים.

שילוב הליכה וההחלפות מייצר שתי הזדמנויות חיסכון מרכזיות: הליכה מסייעת בצמצום מעקפים מיותרים של הרכבים לאזורים מרוחקים ומאפשרת קיצורי דרך ברשת הכבישים, בעוד שהחלפות מאפשרות למערכת לאזן עומסים על הרכבים ולצמצם את אזורי השירות המכוסים על ידי כל רכב. למרות שמאפיינים אלו מספקים גמישות תפעולית משמעותית, הם גם מגדילים את מורכבות הבעיה מעבר לבעיית "הזמן וסע" הקלאסית המוכרת גם כבעיה NP-קשה.

כדי להתמודד עם אתגר זה, אנו מציעים גישת פתרון מודולרית. ראשית, אנו ממדלים את תת-בעיית התזמון כבעיית זמן פשוטה (STP) תוך שימוש באלגוריתם בלמן-פורד (Bellman-Ford) כדי לספק בדיקת ישימות באופן יעיל חישובית. לאחר מכן, מיושמת היוריסטיקת תזמון בת ארבעה שלבים שנועדה לבנות לוחות זמנים לכלי הרכב והשמתשמים בצורה קרובה לאופטימלית אך בזמן פולינומי. רכיבים אלו משולבים במסגרת של חיפוש בסביבה גדולה אדפטיבית (ALNS) הכוללת מטריקת דמיון חדשנית לבקשות משתמשים ואופרטורים ייעודיים להכנסת החלפות ומקטעי הליכה.

ניסויים חישוביים המבוססים על נתונים מהעולם האמיתי משירות "באבל דן" (Bubble Dan) בישראל מדגימים את יעילות הגישה שלנו. בדיקת הישימות באמצעות בלמן-פורד פועלת במהירות הגבוהה פי 200 מאשר שימוש ב-CPLEX למציאת פתרונות מדויקים, בדוגמאות קטנות, בעוד שהיוריסטיקת התזמון מגיעה לפער אופטימליות ממוצע של כ-1.3% בלבד עבור דוגמאות גדולות. עבור הניתוב, ה-ALNS מזהה באופן עקבי פתרונות ישימים עבור בעיות בגדלים שבהם שיטות מדויקות הופכות לבלתי ישימות חישובית. לבסוף, ניתוח ממוקד מגלה כי המעברים משמשים כמנגנון מבני קריטי, המועיל ביותר תחת אילוצים הדוקים כגון גמישות הליכה מוגבלת או קיבולת רכב מוגבלת.

אוניברסיטת תל – אביב

הפקולטה להנדסה ע״ש איבי ואלדר פליישמן

בית הספר להנדסת חשמל

בעיית ״הזמן וסע״
עם החלפות והליכה

חיבור זה הוגש כעבודת גמר לקראת התואר ״מוסמך אוניברסיטה״

בהנדסת חשמל ואלקטרוניקה באוניברסיטת תל- אביב

על - ידי

עידן משולמי

העבודה נעשתה בבית הספר להנדסת חשמל

בהדרכת ד״ר מור כספי

אדר תשפ״ו

אוניברסיטת תל – אביב

הפקולטה להנדסה ע״ש איבי ואלדר פליישמן

בית הספר להנדסת חשמל

בעיית ״הזמן וסע״
עם החלפות והליכה

חיבור זה הוגש כעבודת גמר לקראת התואר ״מוסמך אוניברסיטה״

בהנדסת חשמל ואלקטרוניקה באוניברסיטת תל- אביב

על - ידי

עידן משולמי

אדר תשפ״ו