Research School for Operations Management and Logistics

An Adaptive Large Neighborhood Search Heuristic for the Share-a-Ride Problem

Baoxiang Li, Dmitry Krushinsky, Tom Van Woensel, Hajo A. Reijers

Beta Working Paper series 475

BETA publicatie	WP 475 (working paper)
ISBN	
ISSN	
NUR	804
Eindhoven	June 2015

An Adaptive Large Neighborhood Search Heuristic for the Share-a-Ride Problem

Baoxiang Li[a,*], Dmitry Krushinsky[a], Tom Van Woensel[a], Hajo A. Reijers[b,c]

[a] *Department of Industrial Engineering and Innovation Sciences, Eindhoven University of Technology, Eindhoven, The Netherlands*
[b] *Department of Computer Science, VU University Amsterdam, Amsterdam, The Netherlands*
[c] *Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands*

Abstract

The Share-a-Ride Problem (SARP) aims at maximizing the profit of serving a set of passengers and parcels using a set of homogeneous vehicles. We propose an adaptive large neighborhood search (ALNS) heuristic to address the SARP. Furthermore, we study the problem of determining the time slack in a SARP schedule. Our proposed solution approach is tested on three sets of realistic instances. The performance of our heuristic is benchmarked against a mixed integer programming (MIP) solver and the Dial-a-Ride Problem (DARP) test instances. Compared to the MIP solver, our heuristic is superior in both the solution times and the quality of the obtained solutions if the CPU time is limited. We also report new best results for two out of twenty benchmark DARP instances.

Keywords: Transportation, The Share-a-Ride Problem (SARP), Adaptive Large Neighborhood Search, Slack Time

1. Introduction

In most real-life situations, especially in urban areas, people and freight transportation operations are managed separately. New city logistics ap-

*Corresponding author. Email: B.li@tue.nl, Tel:+31 (0)402472693, Postal address: Paviljoen E18, Den Dolech 2, Eindhoven University of Technology, 5612 AZ Eindhoven, The Netherlands

proaches are needed to ensure an efficient urban mobility for both people and goods.

For a real-world application, online shopping becomes increasingly popular, and the delivery time is the key to success of online shopping business. To attract customers, many companies offer same-day delivery service (for instance, Amazon, 360Buy). Currently, the parcels are mainly delivered by vans. However, road congestion becomes a serious obstacle for a timely delivery. The main reason for road congestion is the significantly large number of vehicles on the road. Basically, there are two kinds of vehicles on the road: vehicles for passengers (e.g. bus, taxi, private car) and vehicles for freight (e.g. truck, van). Ensuring an efficient urban mobility for both people and goods becomes more and more critical. Furthermore, the accessibility of some districts is limited for trucks (e.g. no trucks or limited hours for trucks to enter a city center), while taxis are allowed almost everywhere at anytime. We propose a potential collaboration between people transportation companies and online shopping companies in Li et al. [7]. For example, Connexxion (a transportation company in the Netherlands) can deliver both passengers and parcels, and the parcels can be provided by bol.com (an online shopping website in the Netherlands). The parcels can be put in the trunk or under the seat. Thus, the use of a people-and-parcel ridesharing system reduces costs, alleviates urban congestion, and reduces environmental pollution. People-and-parcels sharing can be modeled as the Share-a-Ride Problem (SARP, see Li et al. [7]). The authors give an introduction to the SARP, which is mainly used in mixed commodity services where people and parcels are simultaneously handled by the same transportation network. An application for this problem is the taxi sharing system.

The problem under consideration can be described as follows. A number of taxis drive in a city to serve transportation requests coming from people. At the same time, they deliver some parcels in case it does not affect their passengers significantly. In Figure 1, an example is given of a combined route of the taxi for parcels and people service.

This involves planning the taxi routes capable of accommodating people and freight as much as possible, under a given set of constraints (related to pickup and delivery times, capacity of a taxi, etc.). This application can be extended to other transportation modes, such as bus, train, or tram.

From the modeling perspective, the SARP can be considered as an extension of the Dial-a-Ride Problem (DARP), which is known to be NP-hard [14]. What makes the SARP even more difficult is that it adds passenger

priority constraints to the classic DARP. The differences between the SARP and the DARP can be summarized as follows:

(i) The SARP ensures that any passenger request must be processed within a given time period, and parcels have no such constraints.

(ii) Two passengers cannot be in the same taxi at same time but two parcels can, and a passenger service has a higher priority: we can insert at most η requests between the pickup and the drop off point of a passenger ($\eta \in \{0, 1, 2, ...\}$).

(iii) In the DARP and the SARP, the time window and travel time constraints lead to time slacks. Most DARP models in the literature do not consider the time slacks. Even if some DARP model has the constraints related to the time slacks, the time slacks can always be put forward to the previous requests or postponed to the following requests. But for the SARP, the discount for passengers in the objective function is related to extra ride time as compared to the direct trip. Thus, when the taxi is serving a passenger, it is better not to assign any time slacks within the passenger trips.

Furthermore, the SARP is a generalization of the Pickup and Delivery problem (PDP). The main difference between the SARP and the PDP is that the SARP considers transportation of both passengers and parcels, it includes extra constraints for passengers, and constraints that describe the relationship between passengers and parcels.

The straightforward exact approach can only solve relatively small problems [7]. An efficient tool, however, is critical for the practical application of the SARP, which motivates us to develop a metaheuristic algorithm to solve it.

In this paper, we describe our heuristic that is based on an adaptive large neighborhood search (ALNS). Three main contributions of this paper are as follows:

- We propose a time slack strategy for the route scheduling.
- We describe an entropy-based diversity measurement. The measurement method can be used: (i) to monitor the performance of subroutines. (ii) adjust the heuristic during its execution.
- We show that medium-sized real life instances can be solved within a relatively short CPU time.

The remainder of this paper is organized as follows. The literature is reviewed in Section 2. Section 3 briefly introduces the problem and model

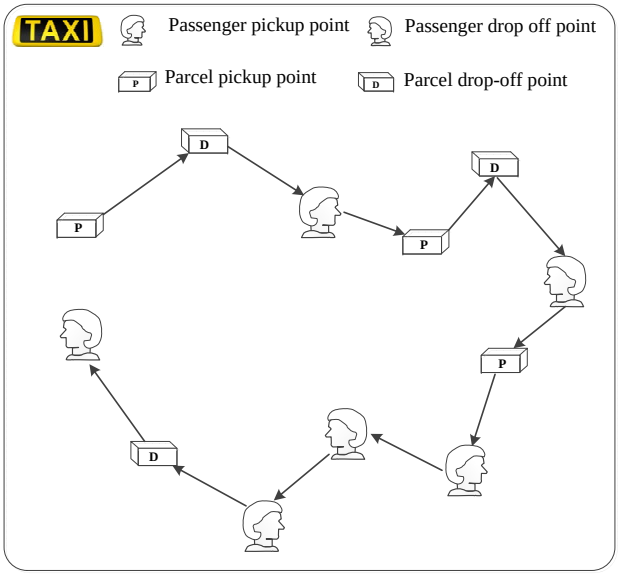


Figure 1: The SARP – taxis serve both passengers and parcels

formulations used in this paper. In Section 4, we describe our ALNS. The evaluation of the ALNS and computational results are presented in Section 5. Finally, Section 6 concludes the paper.

2. Literature Review

The description of the SARP was proposed recently. Li et al. [7] explained the conceptual and mathematical models in which people and parcels are handled in an integrated way by the same taxi network. To our knowledge, there is no specific solution algorithm available for the SARP. However, the SARP can be considered as an extension to the DARP, and most of the heuristics used for the DARP can be adjusted for solving the SARP. Several versions of the DARP were studied over the past four decades. We refer to three surveys on the DARP by Cordeau and Laporte [4, 5] [2]. In this section, we mainly review the class that was solved by tabu search, insertion-based, and cluster-based heuristics. The reason is that these approaches can easily accommodate a large variety of constraints, and allow solving instances with hundreds of requests, according to [5].

Tabu search is a classical method to solve the DARP. Cordeau and Laporte [12] described a tabu search heuristic for the DARP that has time

windows. The results presented in that paper were used as a benchmark in subsequent papers that attempted to solve the DARP by heuristics. The instances generated in that paper together with instances provided by Li and Lim [3] are mainly used in the literature related to the pickup and delivery problem with time windows (PDPTW) and the DARP problems.

Since large neighborhood search heuristics have shown excellent results in solving transportation and scheduling problems in recent years, researchers applied them to the Pickup and Delivery problem. Ropke and Pisinger [8] proposed an adaptive large neighborhood search heuristic for the PDP with time windows. The heuristic was tested on more than 350 instances. The instances were modifications of those first proposed by Li and Lim [3]. The regret insertion heuristic they used in this paper performs well: it decreases at least 10% of the gap from the best known solution as compared to the basic greedy operator (based on 16 tested problems).

Parragh et al. [16] proposed a variable neighborhood search-based heuristic, using three classes of neighborhoods: (1) the first class uses simple swap operations; (2) the second class is based on the so-called ejection chain idea (moving sequences of requests); (3) the third neighborhood class exploits the existence of trips where the vehicle load is zero. Regarding the test instances proposed by Cordeau and Laporte [12], they reported 16 new best solutions.

Good insertion heuristics are critically important to the performance of a neighborhood search. Lu and Dessouky [9] presented an insertion-based construction heuristic to solve the multi-vehicle pickup and delivery problem with time windows, which considered both incremental distance measures and the cost of reducing the time window slack due to the insertion. The proposed heuristic was tested on the instances provided by Li and Lim [3]. Diana and Dessouky [10] presented a new regret insertion-based construction heuristic to solve the large-scale DARP with time windows. In their paper, time window control was used to quickly check the feasibility of an insertion. The algorithm was tested on data sets of 100 and 500 requests generated from a para-transit service data. Häme [11] addressed an adaptive insertion algorithm for the single-vehicle DARP. The performance of the heuristic with different objective functions (related to route duration and time slack) was evaluated.

Other heuristics, such as 2-opt or 3-opt route construction methods and multi-phase construction heuristics, are described in the literature. Savelsbergh [13] investigated the implementation of edge-exchange improvement methods for the vehicle routing problem. According to this paper, the com-

puting time for 2-exchange and OR-exchange can be linear in the number of requests. Hernández-Pérez and Salazar-González [15] proposed one heuristic, which involves merging feasible paths, 2-opt and 3-opt methods. Schilde et al. [6] presented four different modifications of metaheuristic solution approaches for the dynamic stochastic DARP with expected return transports: dynamic versions of variable neighborhood search (VNS), stochastic VNS (S-VNS), modified versions of the multiple plan approaches (MPA), and the multiple scenario approach (MSA). For most of the tested cases, S-VNS outperforms other methods.

3. Mathematical model formulation

Following the notation of Li et al. [7], let σ denote the number of requests to be served, which includes m parcels and n passengers. The SARP is defined on a complete undirected graph $\mathcal{G} = (V, E)$ where $V = V^p \cup V^f \cup \{0, 2\sigma + 1\}$. Subsets V^p and V^f correspond to passenger and parcel stops, respectively, while stops 0 and $2\sigma + 1$ represent the origin and destination depots of the taxis. Moreover, $V^{p,o}$ and $V^{f,o}$ represent the set of passenger origins and parcel origins, respectively. For easy referencing, all stops in V are arranged in such a way that all origins precede all destinations, origins of passengers precede origins of parcels, and destinations of passengers precede those of parcels. Thus, the destination of each request can be obtained as its origin offset by a fixed constant σ. With each edge $(i, j) \in E$ are associated a distance d_{ij} and a travel time t_{ij}. Let K be the set of vehicles, for each arc $(i, j) \in A$ and each vehicle $k \in K$, let $X_{ij}^k = 1$, if vehicle k travels from node i directly to node j. For each passenger i ($i \in V^{p,o}$), let r_i^k be his/her ride time on vehicle k.

The objective function includes four parts: (i) the profit obtained from passengers; (ii) the profit obtained from parcels; (iii) the cost related to the distance traveled; (iv) the discount related to extra ride time of passengers as compared to the direct trip. Furthermore, the initial profits obtained from a passenger and a parcel are represented by α and β, respectively; while the average profits per unit distance are denoted by γ_1 and γ_2, and the cost per unit distance is γ_3. The discount factor for exceeding the direct delivery time of passengers is represented by γ_4.

Thus, we aim at optimizing the objective:

$$\max \Bigg[\sum_{i \in V^{p,o}} \sum_{j \in V} \sum_{k \in K} (\alpha + \gamma_1 d_{i,i+\sigma}) X_{ij}^k + \sum_{i \in V^{f,o}} \sum_{j \in V} \sum_{k \in K} (\beta + \gamma_2 d_{i,i+\sigma}) X_{ij}^k - \gamma_3 \sum_{i \in V} \sum_{j \in V} \sum_{k \in K} d_{ij} X_{ij}^k - \gamma_4 \sum_{i \in V^{p,o}} \sum_{k \in K} (r_i^k / t_{i,i+\sigma} - 1) \Bigg], \quad (1)$$

while satisfying the following constraints:

- capacity constraints of taxis;
- time window constraints for passengers and parcels;
- working hours limitation constraints for taxi drivers;
- every passenger or parcel can be served at most once by one taxi;
- any passenger request must be processed within a given time period;
- at most η requests can be inserted between the pickup and drop-off point of a passenger.

For the mathematical formulation of constraints, we refer to Li et al. [7].

4. An Adaptive Large Neighborhood Heuristic for the SARP

In this section, we describe our ALNS heuristic. An initial solution is constructed by using a basic greedy insertion heuristic to insert the randomly selected requests to the vehicles. Next, the ALNS heuristic is used to improve the original solution. All parameters mentioned in this section are summarized in Table 1.

4.1. The ALNS framework

The heuristic used is based on the ALNS described by Ropke and Pisinger [8]. The ALNS algorithm with simulated annealing as a local search framework is presented in Algorithm 1. In the algorithm, each iteration includes two subroutines: request selection and perturbation. Let s be the current solution, s' be the new solution, and $f(s)$, $f(s')$ – the corresponding objective values. If $f(s')$ is worse than $f(s)$, we accept the solution s with probability $p(s', s)$:

Table 1: Notations for the ALNS and solution evaluation

A_i	Arrival time at stop i
$[e_i, l_i]$	Time window for stop i
W_i	Waiting time at stop i $W_i = \max(0, e_i - A_i)$
F_i	Time slack of request i
B_i	The time when service at i starts, $B_i = A_i + W_i + F_i$
s_i	Service time of request i
D_i	Departure time from stop i, $D_i = B_i + W_i + F_i + s_i$
L_i	Ride time of request i, $L_i = D_{i+\sigma} - D_i$
T	Maximum route duration
ϖ	Last request on the route
R_i	Ride time of the request i
I_i	Number of insertions between the pickup point and drop off point of passenger i, for parcels, $I_i = 0$
M	Big enough number (we used 10^5)

$$p(s', s) = \min\{1, e^{(f(s')-f(s))/\overline{T}}\}, \tag{2}$$

where $\overline{T} \geq 0$ is the "temperature" that starts at $\overline{T_0}$ and decreases every iteration using the expression $\overline{T} := 0.9999 \cdot \overline{T}$, $\overline{T_0}$ is defined in such a way that the objective value of the first iteration is accepted with a probability 0.5. The simulated annealing structure is the same as in [8].

Two solution evaluation approaches are used for the ALNS:

(1) $ALNS_F$: only feasible solutions are allowed during the search;

(2) $ALNS_I$: infeasible solutions are considered and a penalty of the violated constraints is added to the objective.

Considering infeasible solutions provides more flexibility in averting local optima. Nevertheless, the run time increases as all the constraints must be checked, while the $ALNS_F$ stops checking when a violated constraint is found.

4.2. Time slack calculation strategy for the SARP

As seen in (1), the ride time of passengers is a part of the objective function. Thus, the distribution of the time slacks affects the objective value. Figure 2 gives an example: suppose $\alpha = \gamma_1 = \gamma_3 = \gamma_4 = 1$, there are three

Algorithm 1: Adaptive Large Neighborhood Search

```
Input: Initial solution s, solution s_best := s;
while stopping criteria not reached do
    s' := s
    Apply selection operator to select requests for removal
    Apply perturbation operator to remove selected requests from s'
    and reinsert as many unserved requests as we can into s'
    if f(s') > f(s_best) then
        s := s', s_best := s'
    else
        if f(s') > f(s) then
            s := s'
        else
            s := s' with probability p(s', s) defined in Equation (2)
    end while
Output: s_best;
```

solutions with different distributions and different cost for the given feasible route (P1 and P2 denote two passengers). No time slack exists in Figure 2a, and the ride times of passengers are 16 and 20 units, respectively. However, if we add 9 units of time slack at "P1 pickup" (Figure 2b), the ride time of P1 decreases to 12 and that of P2 changes to 15. Instead, if we add 10 and 3 units of time slack at "P1 pickup" and "P2 pickup" (Figure 2c), the ride times change to 15 and 12, respectively. The three time slack scheduling options indicate that different time slacks lead to different ride times, which affect the objective value (2.40, 3.30 and 3.30 in these three scheduling options).

Cordeau and Laporte [12] proposed a forward time slack strategy, where all the pickup points are checked for time slacks. Nevertheless, we can get a better objective value if we only add slacks on part of the requests. For example, if a passenger shares a ride with parcels, he/she can always get a better solution if the time slacks of parcels are zero. This is formalized in the following theorem (the proof can be found in Appendix A):

Theorem 4.1. *Consider two sub-routes* $(i_1, \ldots, i, \ldots, j_1)$ *and* $(i_2, \ldots, j_2)$ *(where* $j_1 = i_1 + \sigma$*,* $j_2 = i_2 + \sigma$*) of one route* $(0, \ldots, i_1, \ldots, j_1, \ldots, i_2, \ldots, j_2, \ldots, 2\sigma + 1)$*, suppose the ride time of* i_1 *and* i_2 *are* L_{i_1} *and* L_{i_2}*, respectively.*

(i) L_{i_2} *does not depend on the time slack at* i.

(ii) L_{i_1} *does not increase if time slacks at* i *and/or* i_2 *are increased.*

If it is allowed that passengers share a ride with other passengers, then optimizing the time slacks is complex and time consuming. Nevertheless, the objective function includes four parts, and the impact of the ride time part is limited. We simplified the time slack strategy as follows: 1) if the model forbids two passengers to share a ride, we only check the time slacks of pickup points of passengers and requests that are not served between passengers; 2) if passengers can share a ride with other passengers, all the pickup points and part of drop off stops (stops between passengers) will be checked.

4.3. Solution evaluation

Let $c(s)$ be the routing profit (value of the objective function (1)). The solution is evaluated by $c(s)$ plus the penalty of load violation $\bar{q}(s)$, duration violation $\bar{d}(s)$, time window violation $\bar{w}(s)$, and ride time violation $\bar{t}(s)$ as follows:

$$f(s) \quad = \quad c(s) + \alpha_q \bar{q}(s) + \alpha_d \bar{d}(s) + \alpha_w \bar{w}(s) + \alpha_t \bar{t}(s) \tag{3}$$

For the $ALNS_F$, $f(s) = c(s)$ holds, because all constraints must be satisfied and $\bar{q}(s)$, $\bar{d}(s)$, $\bar{w}(s)$, and $\bar{t}(s)$ are equal to zero.

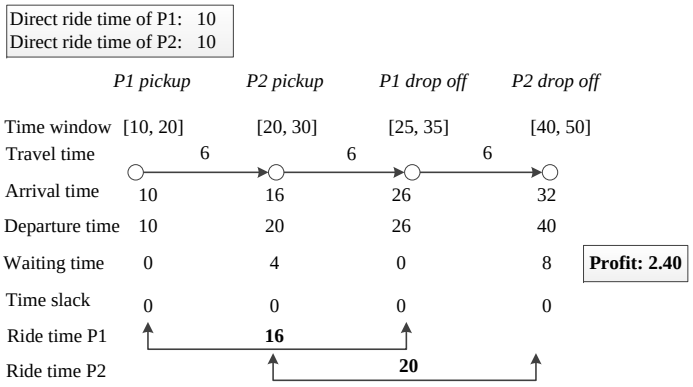
At the end of each iteration, the values of the parameters α_q, α_d, α_w and α_t are modified by a factor $1+\delta$, with $0 < \delta \leq 1$. If the current solution is feasible with respect to load constraints, the value of α_q is divided by $1+\delta$. Otherwise, it is multiplied by $1+\delta$. The same rule applies for α_d, α_w, and α_t.

In Algorithm 2, the solution evaluation scheme is given. Table 1 lists all the relevant parameters used in Algorithm 2. Compared to [12], three adjustments are made:

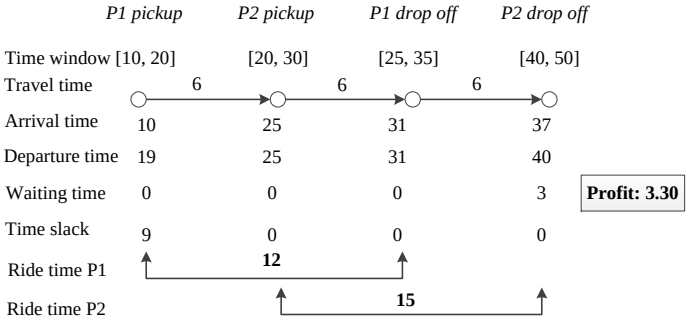
(1) If the constraints prohibiting insertion of more than η parcels during one passenger service are violated, $f(s)$ is set to M;

(2) In the SARP, parcels have no ride time constraints and the related time slacks are computed as:

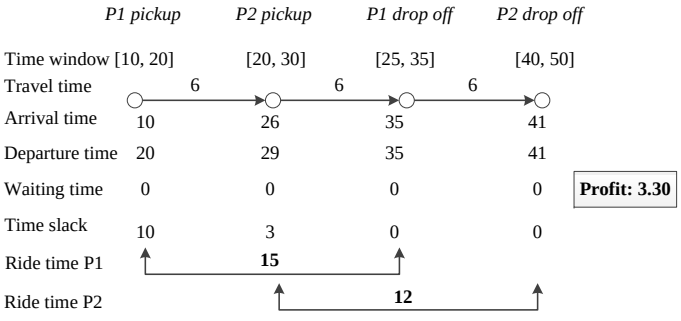
$$F_j := \min_{j \leq k \leq \varpi} \Big\{ \sum_{j < p \leq k} W_p + (l_k - B_k)^+ \Big\} \tag{4}$$



(a) No time slack



(b) The first option for adding time slacks



(c) The second option for adding time slacks

Figure 2: The effect of time slacks

For passengers:

$$F_j := \min_{j \le k \le \varpi} \Big\{ \sum_{j < p \le k} W_p + (\min\{l_k - B_k, L_j - R_j\})^+ \Big\} \tag{5}$$

where $L_j - R_j = 2t_{j,j-\sigma} - R_j$ for drop off points, otherwise $L_j - R_j = 2t_{j,j-\sigma}$. A loop is needed to calculate F_j in Equation (4) and (5), if there exist k that makes $\sum_{j<p\le k} W_p \ge F_j$, stop the loop and output F_j.

(3) A maximum total time slack TWT is defined; at first it equals the total waiting time ($\sum_{0<i<\varpi} W_i$), then it decreases by the time slacks of the checked node.

4.4. Adaptive weight adjustment procedure

The choice of the selection and perturbation heuristics is governed by a roulette wheel mechanism. We have ten selection operators and seven perturbation operators. On the one hand, we diversify the search by combining different operators. On the other hand, a good balance between the quality of the solution and the running time can be reached by choosing a suitable operator at every iteration.

We defined P_d^t as the probabilities of choosing operator d at iteration t. Starting from a predefined value, they are updated as $P_d^{t+1} := P_d^t(1-\rho) + \rho\chi_i/\zeta_i$, where ρ is the roulette wheel parameter, χ_i is the score of operator i, and ζ_i is the number of times it was used during the last 100 iterations. The score of an operator is updated as follows. If the current iteration finds a new best solution, the scores related to the used operators are increased by π_1; if it finds a solution better than the previous one, their scores are increased by π_2; if it finds a non-improving yet accepted solution, their scores are increased by π_3. Every 100 iterations, new weights are calculated using the scores obtained, and all scores are reset to zero.

4.5. Requests selection

At each iteration, u requests will be selected and added to a perturbation set C (set C initially includes the unserved requests). Ten selection operators are used, the first five are adapted from Ropke and Pisinger [8], the ninth operator is motivated by Cordeau and Laporte [12], while others are inspired by Demir et al. [1] and Parragh et al. [16].

Algorithm 2: Eight-step evaluation scheme

Input: Route s

1 Set $D_0 := e_0$, $TWT := 0$;

2 for each i in s: compute A_i, W_i, B_i, D_i, I_i (see Table 1)

if $I_i > \eta$ **then** $f(s) := M$, **goto Output**

if $q(s)$ or $w(s) > 0$ **then** for the $ALNS_F$, $f(s) := M$ and **goto Output**

if $d(s)$ or $t(s)$=0 **then goto STEP 8**

3 Compute F_0

4 $TWT := \sum_{0<p<\varpi} W_p$, set $D_0 := e_0 + \min\{F_0, TWT\}$

5 Update A_i, W_i, B_i, D_i for each vertex v_i in the route

6 Compute L_i for each request assigned to the route

7 For each j that needs checking the time slacks

(a) **if** $TWT = 0$ **then goto** STEP 8, **else** $TWT := TWT - W_j$

(b) Compute time slacks F_j for requests: for passengers (Equation (4)) and for parcels (Equation (5)).

(c) Set $B_j := B_j + \min\{F_j, TWT\}$; $D_j := B_j + d_j$, $TWT := TWT - \min\{F_j, TWT\}$

(d) Update A_i, W_i, B_i, D_i for each vertex v_i that comes after v_j in the route, if B_i did not change, **goto STEP 8**

8 Compute changes in violations, calculate $f(s)$ using Equation (3)

Output: $f(s)$

- **Random (R1)**: This operator randomly selects u requests, thus diversifying the search.

- **Close and Loose Package-First-Shaw (R2 and R3)**: The basic Shaw removal heuristic used in this paper is proposed by Ropke and Pisinger [8], the relatedness function of requests i and j is given by (6).

$$R(i,j) \quad = \quad d_{ij} + d_{i+\sigma,j+\sigma} + \lambda(|B_i - B_j| + |B_{i+\sigma} - B_{j+\sigma}|) \qquad (6)$$

  The first 5 steps in Algorithm 2 are used when calculating the value of the B_i for request i. For given j, if passenger i_1 and parcel i_2 have the same relationship value: $R(i_1, j) = R(i_2, j)$, we prefer to remove the parcel i_2 rather than the passenger i_1. Therefore, Equation 6 is used when calculating the relationship for a parcel, and the relationship value used for a passenger when implement operators **R2** and **R3** as follows:

$$\begin{aligned} R(i,j) \quad &= \quad \overline{p}(d_{ij} + d_{i+\sigma,j+\sigma} + \lambda(|B_i - B_j| + |B_{i+\sigma} - B_{j+\sigma}|)) \\ &\quad\quad 0 < \overline{p} < 1 \end{aligned} \qquad (7)$$

  The requests with small and large relationship values to the already removed requests are selected for operators **R2** and **R3**, respectively.

- **Forbidden random (R4)**: This operator keeps a record of the removal counts for the last 100 iterations, defined as N_i for a given request i. If N_i is bigger than a predefined number $(150/\sigma)$, then request i is forbidden for selection. We apply **R1** to the non-forbidden requests.

- **Forbidden Shaw removal (R5)**: This operator is similar to **R4**. If N_i is bigger than a predefined number $(150/\sigma)$, then request i is forbidden for selection. We apply **R3** to the non-forbidden requests.

- **Sequence (R6)**: The operator first concatenates all the routes in a sequence, then randomly selects $0.75u$ successive stops. Finally, the operator repairs the routes (for each request i either $|C \cap \{i, i+\sigma\}| = 0$ or $|C \cap \{i, i+\sigma\}| = 2$).

- **2-sequences (R7)**: The operator randomly chooses two routes, then randomly selects a sequence of $0.5u$ requests from each route to put into set C. If the route length is smaller than $0.5u$, the whole route is put into set C.

- **Historical (R8)**: This operator keeps a record of the related distance of the pickup and drop off points for every request i, defined as the sum of the distances to the preceding and following requests: $d_i = d_{i-1,i}+d_{i,i+1}+d_{i-1+\sigma,i+\sigma}+d_{i+\sigma,i+\sigma+1}$. At each iteration, the best position cost d_i^{best} is updated to be the minimum of all d_i values calculated until this iteration. The u stops with the largest deviation $(d_i - d_i^{best})$ are put into set C.

- **Tabu based (R9)**: This operator keeps a record of the times that every request has been removed. Then, u requests with the smallest frequencies of removal are put into set C.

- **1-Route (R10)**: A randomly selected route is put into set C.

4.6. Requests perturbation

After the procedure of requests selection, seven perturbation heuristics have been implemented. Basically, the operators can be divided into two types: the first type is one-by-one removal of requests belonging to C from the current route and reinserting as many unserved requests as we can; the second type is removing all the requests in C at once and reinserting one by one as many unserved requests as possible. The first four operators are motivated by Cordeau and Laporte [12], while others are inspired by Ropke and Pisinger [8]. For the $ALNS_F$, if no feasible routing is found when inserting a request, this request is rejected. For the $ALNS_I$, if $f(s) > M$, this request is rejected.

- **1-by-1 (I1)**: The selected requests are sequentially removed one by one and reinserted into the best position.

- **Balanced 1-by-1 (I2)**: For every node, we choose a route with the best route objective value to insert. It tends to generate a relatively balanced solution.

- **Diversification 1-by-1 (I3)**: The operator is similar to **I1** except that there is a penalty to diversify the search. Penalty $p(s) = \lambda c(s)\sqrt{mn}\rho_{ik}$ is added to the objective function $f(s)$ when evaluating the cost of inserting node i into route k, where $\lambda = 0.015$, m is the number of vehicles, n is the number of requests, $c(s)$ is the route cost, and ρ_{ik} is the number of times node i has been inserted into route k in the previous 100 iterations. This type of scaling factor has been used in the tabu search (see, e.g., Cordeau and Laporte [12]).

- **Tabu 1-by-1 (I4)**: This operator implements a diversification strategy similar to the tabu search. Suppose that request i is removed from some route k, the request is then not allowed to be reinserted into route k. The ban can only be canceled if insertion into route k leads to a better routing profit compared to the best known routing profit of route k with i inside. For the request that has never been served before, skip the removal step and only do the insertion.

- **Global all-at-once (I5)**: The operator repeatedly inserts unserved requests in the best position of all the routes. The difference with **I1** is that all the requests are removed at once, then inserted again one by one.

- **Regret-3 all-at-once (I6)**:

  The regret-3 value is the difference in the profit of inserting the request in its best route and its second best route, plus the difference of inserting in best and third best route. Let f_i denote the regret-3 value of request i, in each step the operator chooses to insert the request i that maximizes:

  $i^* := \underset{i}{\operatorname{argmax}} f_i.$

  If some requests cannot be inserted in at least 3 routes, then operator **I5** is used.

- **Local all-at-once (I7)**: This operator is similar to **I5**, the difference is that **I7** only specific route is checked, and **I5** checks all routes. Suppose, request i is removed from some route k, it tries to insert the request i into the same route k again but in a better position.

In Section 5.3, we return to the issue of the efficiency of the selection and perturbation operators.

4.7. Stopping criteria

Dynamic stopping criteria are defined: let t count the hundreds of iterations, $g(t)$ be the value determining the stopping criterion (if $g(t) < 0.5$ or $100t = 25000$, stop the algorithm), the value of $g(t)$ updates every 100 iterations. $e(t)$ stands for the best objective value until $10000 + 100t$ iterations. The expressions used for updating $g(t)$ after 10000 iterations are as follows:

$$g(0) = 1, \quad g(t) = 0.99g(t-1) + |e(t-1) - e(t)| / \max(e(t-1), e(t)) \quad (8)$$

5. Computational Experiments

This section presents results of the computational experiments to assess the performance of the ALNS. The ALNS is implemented in Java and executed on an Intel Xeon E5-4610 2.4 GHz 6 core CPU 32 GB RAM computer. The parameters used in the ALNS are shown in Table 2. The parameter tuning strategy is the one proposed by Ropke and Pisinger [8]. Every time only one parameter is adjusted, while the rest are being fixed. The setting with the best average behavior (in terms of average deviation from the best known solutions) is chosen. This process iterates through all parameters once.

Table 2: Parameters used in the ALNS

Descriptions	SARP Values	DARP values
Number of selection stops in the first 15000 iterations	5%-25%	5%-25%
Number of selection stops after 15000 iterations	15%-45%	15%-35%
Roulette wheel parameter, ρ	0.70	0.70
Score of a global better solution, π_1	5.00	15.00
Score of a better solution, π_2	9.00	10.00
Score of a worse solution, π_3	4.00	5.00
Shaw parameter, λ	0.70	0.70
Close removal parameter, p_1	1.80	1.80
Loose removal parameter, p_2	0.56	0.56
Penalty adjustment parameter, δ	[0,0.25]	[0,0.25]
$P^0_{d_R}$ used for selection operators	0.10	0.10
$P^0_{d_I}$ used for perturbation operators	**I1** 0.25, others 0.125	**I5** 0.25, others 0.125

5.1. Test Instances

To analyze the behavior of the metaheuristics, we used three classes of instances, and the main features of different instance classes are described in Table 3. The first class (which includes 15 requests) is based on the San Francisco taxi trail data (http://cabspotting.org), aimed at comparing the performance of the ALNS with the solution given by an MIP solver. In total, 10 instances are generated (SF1–SF10). Each instance contains 15 requests and 2 vehicles. The number of passengers is two times that of the parcels. The capacity of each vehicle is 5. The weights of passengers and parcels are

set to 3 and 1, respectively. The time window width for passenger pickup stops is 1 hour, and a window of 1.5 hours is used for passenger drop off stops. Both the time window width for pickup and drop off stops of parcels are 2 hours (9:00–11:00). The working time limit of drivers is 2 hours.

The second class consists of the DARP instances, proposed by Cordeau and Laporte [12]. We define one third of the requests as parcels, the others are passengers. The maximum travel time for a passenger is twice the direct travel time. A direct passenger service can be disturbed at most twice for serving other requests. The number of vehicles used in most of the instances (all except R1a and R1b in Table 5) is one less than in the original settings.

The third class with 9 instances (SM1–SM9) is similar to the second class (again, San Francisco taxi trail data), each instance contains 60–300 requests. Both the time window width for pickup and drop off stops of parcels are 8 hours (9:00–17:00). The working time limit of drivers is 10 hours. Other settings are the same as for the first class.

All the instances used can be found at the http://smartlogisticslab.nl/. In the first and third classes of instances, distances are calculated in Manhattan metrics $d(x; y) = |x_1 - y_1| + |x_2 - y_2|$, and travel times are calculated as a ratio between the distance traveled and the average speed 5.36 m/s. Euclidean metrics is used to compute the distance between the stops for the second group of instances (the DARP instances). The parameters used in the objective function are defined as follows: $\alpha = 3.50$, $\beta = 2.33$, $\gamma_1 = 2.70$, $\gamma_2 = 0.90$, $\gamma_3 = 0.60$, and $\gamma_4 = 3.50$.

The goal of this series of experiments is twofold: the ALNS should perform well in both the solution quality and CPU times for the SARP on the one hand, and should be applicable to the general DARP problem on the other hand.

Table 3: Main features of different instances classes

Class	# Inst.	# Requests	Pass.	Distance	Speed	WTL
1 SF taxi trails	10	15	2/3	Manhattan	5.36 m/s	2 hours
2 DARP [12]	20	24–144	2/3	Euclidean	1 unit	480 units
3 SF taxi trails	9	60–300	2/3	Manhattan	5.36 m/s	10 hours

Pass.: the fraction of passengers in the number of requests

WTL: working time limit of drivers

5.2. Computational Results

The performance of the ALNS for the benchmark instances is reported in this section. Every ALNS result is based on 10 runs and the best one is presented. An average CPU time for one run out of 10 is measured in minutes.

5.2.1. Computational Results for San Francisco instances

In this section, we focus on a set of 10 instances (SF1–SF10). Due to the small instance size, we use 5000 iterations when running the ALNS. The reason is that when we have small instances with few requests, the running time is not an issue, and we did not set extra limitations besides the fixed number of iterations. Table 4 shows the efficiency of the ALNS introduced in comparison to a MIP solver (GUROBI, time limit of 10 hours). The first column denotes the instances. The columns "Gap" refer to the gaps relative to the lower bound from GUROBI, for the objective value (upper bound) obtained from the GUROBI, the $ALNS_I$ and the $ALNS_F$, respectively. Furthermore, the CPU times of the $ALNS_I$ and the $ALNS_F$ are reported.

We observe that the $ALNS_F$ provides similar solutions to the $ALNS_I$, but the CPU times of the $ALNS_I$ are approximately eight times larger. The performance of the $ALNS_F$ and the $ALNS_I$ is superior in both the solution times and the quality of the obtained solutions in comparison to the results obtained by GUROBI.

Table 4: Performance of the ALNS (5000 iterations) on San Francisco instances

Instance	Gap (%)			CPU (min)	
	UB	$ALNS_I$	$ALNS_F$	$ALNS_I$	$ALNS_F$
SF1	24.94	37.75	25.80	0.93	0.10
SF2	14.44	14.44	14.44	1.01	0.10
SF3	32.91	16.09	16.09	0.85	0.07
SF4	25.46	26.41	25.99	0.72	0.07
SF5	32.83	21.59	21.59	0.95	0.10
SF6	37.52	31.89	41.46	0.86	0.10
SF7	35.23	24.48	24.48	0.63	0.10
SF8	97.64	61.48	61.47	0.63	0.10
SF9	47.63	28.96	28.96	0.33	0.10
SF10	49.07	39.95	44.33	0.67	0.07
Average	39.77	30.30	30.46	0.76	0.09

5.2.2. Computational Results for the DARP Instances

To assess the performance of the ALNS, we present computational results on the Cordeau and Laporte DARP instances [12]. By assuming that all requests are passengers and setting the maximum ride time equal to a fixed value and the number of vehicles equal to the number used in [12], by further requiring all requests to be served (every rejected request incurs a penalty of M), and minimizing the total distance as the objective function, the SARP is transformed to the DARP.

Table 5 summarizes the computational results. The first column denotes the instances, the second and third columns indicate the number of requests and cars. The fourth and fifth columns provide the percentage deviation of the $ALNS_I$ and the $ALNS_F$ compared to the best results from [12] and [16]. The last two columns report the CPU time of the $ALNS_I$ and the $ALNS_F$. The average gap of the $ALNS_I$ is 2.24% for the $ALNS_I$. We provide 2 better results, compared with all the literature using the same instances, we report two new benchmarks (R1a: 187.84, R1b: 164.38). The gap of the $ALNS_F$ is 22.73%. However, it is approximately two times faster than the $ALNS_I$.

5.2.3. Computational Results for the SARP

By testing both $ALNS_I$ and $ALNS_F$ on instances R1a, R2a, R3a, R7a and R8a, we find that the $ALNS_I$ provides similar results to those of the $ALNS_F$ for instances R1a, R3a, and R7a. Furthermore, the $ALNS_I$ gives slightly better result than the $ALNS_F$ on the other two instances (R2a and R8a). Nevertheless, the $ALNS_F$ is approximately 9 times faster than the $ALNS_I$. It turned out to be impossible to get final results in a reasonable amount of time for the $ALNS_I$ for some instances. Therefore, we mainly used the $ALNS_F$ for the computations. Furthermore, we run a approach called DARP-first-SARP-second, which means first running 25000 iterations to get a DARP solution (treating all requests as passengers), then repairing the solution by running 5000 iterations to get a SARP solution.

Table 6 presents results obtained by the $ALNS_F$ on the instances based on Cordeau and Laporte [12]. The first column denotes the instances, the second to fourth columns display the $ALNS_F$ result, the gap to the best value in percentage, and the CPU times. The following three columns show the corresponding results for the DARP-first-SARP-second approach. All the best values per instance are identified across all the testing during developing the ALNS heuristic, and presented in the last column. The average CPU time is 22 min. The gap of the DARP-first-SARP-second approach to the best

Table 5: Results of the ALNS on the benchmark DARP instances

Instance	#Requests	#Cars	Gap (%)		CPU (min)	
			$ALNS_I$	$ALNS_F$	$ALNS_I$	$ALNS_F$
R1a	24	3	-1.15	1.35	1.63	0.49
R2a	48	5	0.55	9.59	9.12	12.11
R3a	72	7	2.01	18.14	17.67	12.11
R4a	96	9	4.95	25.21	56.14	18.07
R5a	120	11	5.38	26.79	65.46	37.37
R6a	144	13	0.92	29.53	153.57	49.84
R7a	36	4	0.00	6.05	4.01	1.12
R8a	72	6	1.45	21.27	44.32	20.93
R9a	108	8	3.66	57.32	148.41	39.78
R10a	144	10	2.53	29.62	242.52	128.30
R1b	24	3	-0.05	2.70	3.59	0.41
R2b	48	5	0.18	15.05	12.35	3.76
R3b	72	7	2.19	24.21	29.94	28.88
R4b	96	9	4.10	29.66	48.76	23.85
R5b	120	11	3.11	33.12	124.12	33.55
R6b	144	13	3.49	41.94	145.33	48.90
R7b	36	4	0.01	8.63	3.65	4.43
R8b	72	6	3.68	10.95	31.49	51.79
R9b	108	8	3.28	25.40	134.27	75.87
R10b	144	10	4.48	38.08	242.64	115.17
Average			2.24	22.73	75.95	35.34

SARP solution is 6.40% but with approximately two times of the CPU for running the ALNS directly on the SARP (with a gap of 1.08%). These results underline the efficiency of the ALNS introduced in this paper.

Table 6: Results for the $ALNS_F$ on the SARP instances

Instance	Profit	Gap(%)	CPU (min)	Profit	Gap(%)	CPU (min)	Best profit
R1a	221.14	0.95	1.05	218.72	2.04	1.89	223.27
R2a	434.17	3.27	3.23	393.48	12.33	7.93	448.84
R3a	944.81	0.36	4.58	898.10	5.29	15.18	948.27
R4a	1145.65	0.73	6.39	1080.03	6.42	30.74	1154.17
R5a	1282.33	0.67	19.47	1199.87	7.05	43.19	1290.94
R6a	1737.64	1.16	30.88	1639.09	6.75	71.2	1757.81
R7a	391.42	0.01	1.36	370.83	5.27	3.46	391.46
R8a	901.32	0	7.82	821.13	8.90	25.69	901.32
R9a	1296.60	0	20.37	1208.61	6.79	51.50	1296.60
R10a	1642.40	3.27	89.02	1473.63	13.12	184.66	1696.08
R1b	243.66	0.38	0.99	241.55	1.24	3.48	244.59
R2b	500.4	0.51	3.26	485.51	3.47	11.06	502.96
R3b	974.37	0.55	8.07	934.21	4.64	30.01	979.68
R4b	1187.04	1.51	15.16	1136.68	5.67	36.8	1205.00
R5b	1335.36	1.45	50.78	1279.08	5.58	116.62	1354.70
R6b	1827.77	0.19	68.10	1739.62	5.00	159.93	1831.24
R7b	431.67	0.62	0.72	424.82	2.19	2.86	434.35
R8b	980.21	0.01	7.85	901.34	8.06	30.53	980.33
R9b	1373.91	0.65	19.83	1242.48	10.16	116.57	1382.91
R10b	1677.63	5.27	85.97	1628.71	8.03	124.7	1770.90
Average	1026.48	1.08	22.245	965.87	6.40	53.4	1039.77

5.3. ALNS configuration

We use the $ALNS_F$ in further experiments due to performance considerations. We evaluate the $ALNS_F$ constructions in four aspects: effect of the initial solution on the final solution, how the different removal and insertion heuristics behave, the impact of decreasing the number of iterations, and the evaluation of the time slack strategy.

5.3.1. The effect of the initial solution on the final solution

To test the effect of the initial solution to the ALNS, we ran several instances 1000 times with random initial solutions. Overall, the ALNS per-

forms stably on different instances. We present typical results in Figure 3. The figure shows the relation between the profit of initial (X-axis) and final solutions (Y-axis). The profits of the final solutions are clustered within an interval [424,434], which indicates that the quality of the initial solution does not influence the final results.

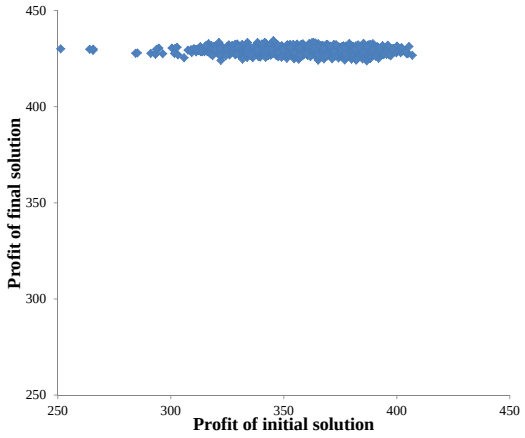


Figure 3: Impact of initial solution on final solution for instance R7b

5.3.2. Behavior of different operators

An entropy-based diversity measurement is used for the assessment of the behavior of different operators. Suppose, set S includes 100 solutions of the last 100 iterations, and let σ be the number of requests. We define coincidence matrix M as follows: $M_{ii} = 0$, $M_{ij} = \sum_{s \in S} T_s^{ij}$, $\forall i, j \in 1, 2, \ldots, \sigma$, where $T_s^{ij} = 1$ iff stops i and j are visited consecutively by some vehicle in solution $s \in S$. The diversity of S is defined as follows:

$D = -\sum_i \sum_j r_{ij} \ln r_{ij}$, where $r_{ij} = 1$ if $M_{ij} = 0$, otherwise $r_{ij} = M_{ij}/100$.

Figure 4 illustrates how the entropy changes for instance R7b. Two types of behavior can be seen: converging (diversity values with more than 90% of the links in the solutions are the same in last 100 iterations) and chaotic (diversity values with more than 40% of the links in the solutions are different in last 100 iterations). Generally, operators **I1**, **I5**, **I6**, **I7** incur converging behavior of the ALNS heuristic, others incur chaotic behavior. We expect that maintaining a higher diversity at the beginning and a lower diversity at the end will lead to better results, as shown in Figure 4. However, if the

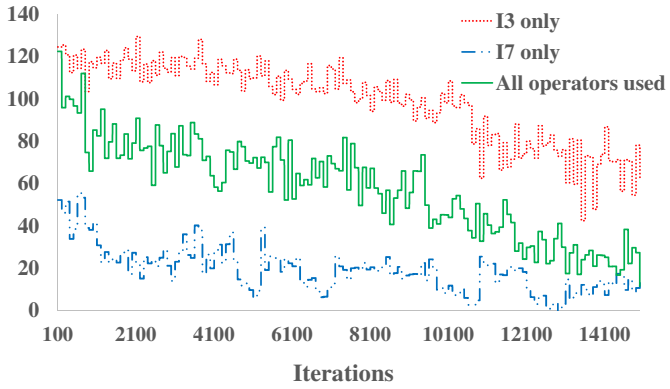


Figure 4: Diversity for different operators for instance R7b

diversity value is too low (such that the individuals are similar), the solution gets stuck in the local optimum.

The quality and running time used by different selection operators are also tested; all the results are based on 10 runs. The total running time of all selection operators is around one second, Figure 5 lists the call counts (the number of times the operator was added) of different selection operators. The figure indicates that the behavior of removal operators is similar. Thus, we mainly analyze the selection operators in the following.

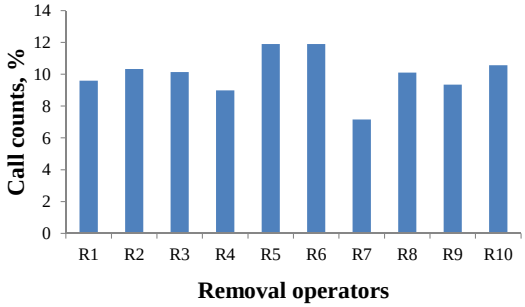


Figure 5: The call counts of removal operators for instance R7b

Figure 6 presents call counts (the number of times the operator was called) and running times (as a percentage of the total CPU times) used for different perturbation operators. Most of the operators show a similar behavior, except **I6** (below 5%). The influence of individual operators on the solution quality is presented in Table 7. All the results are worse than if 7 operators are used. Among all the individual operators, **I3** is the best (with a gap

9.18%) and **I7** is the worst (with a gap 36.79%). If operator **I1** is excluded, the gap increases to 8.95%. Furthermore, for other operators, the result does not change a lot (within a gap between 2.21. . . 5.84%). A conclusion is that the contribution of operator **I1** is higher than that of other operators. In addition, the result changes within a range of 10% if any operator is deleted.

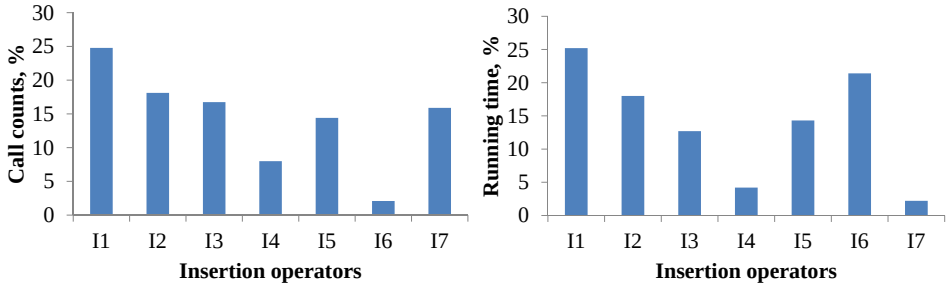


Figure 6: The call counts and running time of selection operators for instance R7b

Table 7: Impact of the perturbation operators on the solution quality for instance R2a (all the results are measured as a percentage deviation from the $ALNS_F$ with all 7 perturbation operators)

	I1	**I2**	**I3**	**I4**	**I5**	**I6**	**I7**
use only one operator	27.15	16.89	9.18	17.19	18.08	20.02	36.79
exclude one operator	8.95	5.84	2.60	2.98	3.34	3.17	2.21

5.3.3. The advantage of the stopping criteria

To check the impact of the stopping criteria, we ran several instances. Overall, our stopping criteria can be met before the fixed number of iterations is reached. We present the details in Figure 7. We can see for this instance, the ALNS stopped at around 16000 iterations, as $g(t)$ reached the threshold of 0.5. Note that the entropy values could be used instead of $g(t)$, leading to an even earlier stopping.

The advantage of the current stopping criterion is that it depends on the convergence rate, and avoids the extra running time if the convergence is fast. However, for large instances with much more requests, it is better to set a lower bound for the number of iterations, otherwise the results can be unstable.

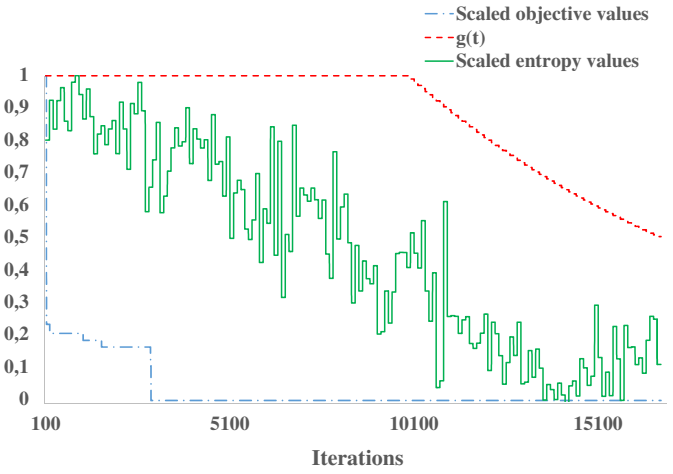


Figure 7: Comparisons of best objective value, g(t) and entropy value on instance R1a

5.3.4. The impact of decreasing the number of iterations

In Table 8, the deviations of the $ALNS_F$ being run for 5000, 10000 iterations, with respect to the $ALNS_F$ being run for 25000 iterations, are summarized. Columns "Average" and "Best" refer to the average and best profit, respectively. Comparing average values over ten runs, the deviation from the average solution for 5000 and 10000 are 3.52% and 1.75%, respectively. Furthermore, the deviation from the best known solution is 2.91% in the former case compared with 2.03% in the latter case. This indicates that when the number of iterations increases, the performance of the $ALNS_F$ improves. Note that the CPU times grow only linearly with the number of iterations.

Table 8: Results of 5000, 10000 iterations, measured as a percentage deviation from 25000 iterations

Instance	Average		Best	
	5000	10000	5000	10000
R1a	1.33	0.05	0.28	0.02
R2a	5.52	3.17	5.99	5.05
R3a	3.57	1.63	2.95	1.57
R4a	3.69	2.03	2.45	1.98
R5a	3.50	1.88	2.88	1.53
Average	3.52	1.75	2.91	2.03

5.3.5. The evaluation of the time slack strategy

We randomly generate 1000 solutions (every solution is obtained by executing 500 iterations the $ALNS_F$) based on one instance (R7b). The objective value is computed using optimal time slack strategy and the Algorithm 2 with a simplified time slack strategy. For the optimal time slack strategy, we optimize the time slack using the MIP solver (GUROBI) by fixing all the routing variables in the constraints and letting the solver optimize the time-related variables. The average gap of simplified time slack strategy compared to optimal time slack strategy is 0.02% with 0.10% standard deviation. Moreover, we set the load of passenger to 3 and capacity of taxi to 5. Therefore, if we have two passengers in one taxi, the load is 6, which exceeds the taxi capacity. Finally, the average gap compared to the GUROBI solution is 0. From the results, we can see that the time slack optimization has a limited impact. Therefore, it seems acceptable to use the time slack strategy we proposed.

5.4. Medium real-life instances

In this section we test our ALNS on a set of 9 medium size instances (SM1–SM9). Table 9 presents the results obtained by executing the $ALNS_F$ 10 times with 3000 iterations. The first column denotes the instances, the second to third columns display the number of requests and cars. The following column presents the CPU times of the $ALNS_F$. Furthermore, columns #Pass and #Pack refer to the number of served passengers and packages, respectively. The objective values are given in the seventh column. Finally, we again make a comparison between simplified and optimal time slack calculation by fixing the routing variables and letting the MIP solver optimize the timing ones in the last column. From the table, we see that the running time increases as the number of requests grows. The CPU times for some instances (SM6–SM9) can excess one hour, but are still reasonable for a problem that needs to be solved daily. Moreover, most of the requests can be served. By comparing between simplified and optimal time slack calculation, it turns out that optimizing the time slacks only slightly improves the profit.

6. Conclusion

In this paper, we proposed an ALNS-based heuristic for solving the SARP. The proposed approach is flexible and can be adapted to handle the DARP

Table 9: The result of the $ALNS_F$ on medium size instances

Instance	#Requests	#Cars	CPU (min)	#Pass	#Pack	Profit	
						ALNS	GUROBI-SL
SM1	60	2	7.33	33	21	335.71	335.71
SM2	90	3	8.94	53	30	560.21	560.21
SM3	120	4	27.38	69	41	766.64	766.92
SM4	150	5	33.94	93	51	1059.98	1059.58
SM5	180	6	43.43	111	61	1163.18	1163.18
SM6	210	7	63.44	133	71	1498.41	1498.41
SM7	240	8	98.91	146	81	1543.81	1543.81
SM8	270	9	131.94	164	90	2066.10	2066.10
SM9	300	10	147.11	190	101	2064.97	2065.95

GUROBI-SL: optimal time slack calculation by fixing the routing variables and letting the MIP solver optimize the timing

(or related vehicle routing problems). The ALNS was mainly tested on instances generated from the Cordeau and Laporte [12] DARP instances. Our results are within 2.24% from the best results from the literature when considering best values over 10 runs.

The related neighborhood search heuristic papers (the DARP with same tested instances) all implement searching methods that accept infeasible solutions, the CPU time is much higher than if only feasible ones are accepted. The ALNS we designed for the SARP can get similar results for the following two approaches: 1) only feasible solutions are accepted ($ALNS_F$) during the search 2) infeasible solutions are also accepted but penalized for violation ($ALNS_I$) during the search. Furthermore, the $ALNS_F$ is much faster than the $ALNS_I$, because the $ALNS_F$ do not need to check all the constraints and stops evaluating a solution after the first violated constraint is found. Nevertheless, we should not expect the algorithm to be the universally best algorithm that can fit all the models.

Different operators for the DARP and for the SARP are required. The main differences between the SARP and the DARP is described in Section 3. Moreover, the difference not only lies in the operators, but also in how to handle the time slacks and the objective function. Even for the same operators, in the SARP passengers have higher priority for insertion and removal. The operators named 1-by-1, Tabu 1-by-1, regret-3 all-at-once operators are important for both DARP and SARP.

The results for the SARP are presented and can be used as a benchmark in the future. An entropy-based diversity measurement is used to assess the performance of the ALNS, the proposed measurement can be extended to include an adaptive weight adjustment procedure and dynamic stopping criteria in a new set up. A major insight that our work provides is that the ALNS-based heuristic can solve the medium size SARP instances, which indicates that it can be beneficial for the practical application of the SARP system.

We believe that extending the application of the ALNS heuristics to the online SARP or to a stochastic environment are valuable areas for future research.

Appendix A Proof of Theorem 4.1

Proof. Let B_{i_2} be the initial service start time of request i_2, $\overline{B_{i_2}}$ be the service start time after add the time slack. For a given subroute $(i, \ldots, i+\psi, \ldots, j)$, if i and $i+\psi$ have time slack F_i and $F_{i+\psi}$, respectively, then B_j update according to:

$$B_j := B_j + \max(0, F_i - \sum_{h=i+1}^{i+\psi} W_h) + \max(0, F_{i+\psi} - \sum_{h=i+\psi+1}^{j} W_h) \quad (9)$$

By using the Equation (9), if we add two forward time slacks F_1 and F_2 on i and i_2, respectively, the formula of L_{i_1} and L_{i_2} are as follows:

$$L_{i_1} = B_{j_1} + \max(0, F_1 - \sum_{h=i+1}^{j_1} W_h) - B_{i_1} \quad (10)$$

$$\begin{aligned} L_{i_2} &= B_{j_2} + \max(0, F_1 - \sum_{h=i_1+1}^{i_m} W_h) + \max(0, F_2 - \sum_{h=i_2+1}^{j_2} W_h) - \overline{B_{i_2}} \\ &= B_{j_2} + \max(0, F_1 - \sum_{h=i_1+1}^{i_m} W_h) + \max(0, F_2 - \sum_{h=i_2+1}^{j_2} W_h) - \\ &\quad (B_{i_2} + \max(0, F_1 - \sum_{h=i_1+1}^{i_m} W_h) + F_2) \\ &= B_{j_2} + \max(0, F_2 - \sum_{h=i_2+1}^{j_2} W_h) - (B_{i_2} + F_2) \end{aligned} \quad (11)$$

If we only add time slack F_2 on stop i_2, let L'_{i_1}, L'_{i_2} be the ride time of request i_2 and i_2, respectively:

$$L'_{i_1} = B_{j_1} - B_{i_1} \tag{12}$$

$$L'_{i_2} = B_{j_2} + \max(0, F_2 - \sum_{h=i_2+1}^{j_2} W_h) - (B_{i_2} + F_2) \tag{13}$$

Thus $L_{i_1} \geq L'_{i_1}$, $L_{i_2} = L'_{i_2}$. □

[1] Demir E, Bektaş T, Laporte G. An adaptive large neighborhood search heuristic for the pollution-routing problem. European Journal of Operational Research 2012;223(2):346–59.

[2] Toth P, Vigo D. The Vehicle Routing Problem; chap. VRP with Pickup and Delivery. Discrete Mathematics and Applications; Society for Industrial and Applied Mathematics; 2002, p. 225–42.

[3] Li H, Lim A. A metaheuristic for the pickup and delivery problem with time windows. In: The 13th IEEE Conference on Tools with Artificial Intelligence (ICTAI-2001). Dallas, USA; 2003, p. 160–70.

[4] Cordeau JF, Laporte G. The dial-a-ride problem (DARP): Variants, modeling issues and algorithms. 4OR 2003;1(2):89–101.

[5] Cordeau JF, Laporte G. The dial-a-ride problem: models and algorithms. Annals of Operations Research 2007;153(1):29–46.

[6] Schilde M, Doerner KF, Hartl RF. Metaheuristics for the dynamic stochastic dial-a-ride problem with expected return transports. Computers and Operations Research 2011;38(12):1719–30.

[7] Li B, Krushinsky D, Reijers HA, Van Woensel T. The share-a-ride problem: People and parcels sharing taxis. European Journal of Operational Research 2014;238(1):31–40.

[8] Ropke S, Pisinger D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science 2006;40(4):455–72.

[9] Lu Q, Dessouky M. A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows. European Journal of Operational Research 2006;175(2):672–87.

[10] Diana M, Dessouky M. A new regret insertion heuristic for solving large-scale dial-a-ride problems with time windows. Transportation Research Part B 2004;38(6):539–57.

[11] Häme L. An adaptive insertion algorithm for the single-vehicle dial-a-ride problem with narrow time windows. European Journal of Operational Research 2011;209(1):11–22.

[12] Cordeau JF, Laporte G. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. Transportation research Part B 2003;37(6):579–94.

[13] Savelsbergh MWP. The vehicle routing problem with time windows: Minimizing route duration. INFORMS Journal on Computing 1992;4(2):146–54.

[14] Baugh JW, Kakivaya GJR, Stone JR. Intractability of the dial-a-ride problem and a multiobjective solution using simulated annealing. Engineering Optimization 1998;30(2):91–123.

[15] Hernández-Pérez H, Salazar-González J. Heuristics for the one-commodity pickup-and delivery traveling salesman problem. Transportation Science 2004;38(2):245–55.

[16] Parragh SN, Doerner KF, Hartl RF. Variable neighborhood search for the dial-a-ride problem. Computers and Operations Research 2010;37(6):1129–38.