Project Report
Advanced Methods in Optimization

A heuristic approach for the Share-a-ride problem

Ana Beatriz Fernandes Herthel

student number 11852994

Abstract

Some of the greatest challenges in urban environments nowadays concerns the mobility of people and fast delivery of parcels. With the surge of e-commerce and the increase in volume of goods to be delivered, we aim to explore logistic solutions by proposing algorithms to add parcel transportation services to ride-hailing systems.

Thus, we present a heuristic approach based on ILS for the share-a-ride problem (SARP). We analyze the quality of solutions, based on the amount of service revenues, by comparing these values to the ones previously obtained through solving a MILP formulation using a commercial solver. Moreover, we assess the performance of the method, in terms of computational efforts.

Our results show that, for 60% of instances, the heuristic method was able to obtain equivalent or better solution values when comparing to the ones previously obtained with an exact method. The average solution time was significantly decreased, and solutions could be achieved in hundredths of seconds.

Contents

1 Introduction

The growing urbanization, the desire to decrease the dependence on fossil energy and, therefore, reduce greenhouse gas emissions, are some of the reasons why new solutions in urban mobility are necessary. In this context, crowd-based transportation emerged as a meaningful alternative to share resources and reduce carbon footprint. It relies on members of the crowd to offer underused space in their vehicles to perform associated activities. Companies such as Uber, Lyft and Didi provide peer-to-peer ride-sharing services, where people who look for rides are paired up with drivers willing to take them. The presence of this business model facilitates access from suburban areas to city centers, as it is able reach areas that public transport might not. This, in turn, contributes to the growth of productivity in bustling economic centers, without increasing their population [5].

The rapid growth of e-commerce is also shifting the way last mile delivery is performed in large cities. Online stores are constantly gaining ground over brick-and-mortar shops, with the inherently comfortable, easy and fast purchasing process of the former. This causes a surge in the number of smaller parcels to be delivered. According to the [6], the handling of lightweight parcels represented the largest part of revenues for postal services in 2020.

The growth trend in shipment volume of parcels of smaller dimensions is also observed in large e-commerce driven companies like Amazon. The majority of their transported packages (around 86%) weigh under 2.3 kg [2] with 1.9 billion units delivered in the US alone, in 2019 [10]. High capacity vehicles, such as vans and trucks, become oversized for deliveries of such scale, which, in most cases, are also expected to quickly arrive to the consumer at lower costs [3].

Usually, in urban areas, the transportation of people and parcels is performed in separate systems dedicated to each activity, even though these systems generally use the same network. By incorporating additional parcel moving services to make use of these relocating trips, vehicles can be employed more efficiently, leading to smaller fleets dedicated to these services in the city centers.

The integration of parcel transportation services with ride-hailing systems can be attractive to both peer-to-peer ride-sharing companies as well as small scale retailers. For the former, it becomes a new source of revenue, and for the latter, it is a possibility of moving goods quickly and safely, without having to heavily invest in dedicated shipping solutions. Moreover, the growth of ride-hailing systems over taxis makes them promising candidates for frameworks in which parcels and passengers share trips. This is reinforced by the fact that ride-hailing systems already have a framework to supply information about parcel movements to customers, without reducing level of service for passengers.

The use of algorithms for solving the problem of joint mobility of parcels and people was introduced by [7]. This problem is called the share-a-ride-problem (SARP) and it arises when the shared transportation of parcels and passengers is performed in taxi trips. This problem is an extension of the well-known dial-a-ride problem (DARP) [4], which is NP-hard [1]. We use the SARP as a framework to model the parcel and passenger transport integration in ride-hailing systems.

Previous results of this research have shown a significant improvement in revenues when parcels and passengers share trips in vehicles. This improvement was even higher when considering situations in which parcels can be directly served without the demand for serving passengers before the parcel delivery. The problem, however, is very difficult to be solved and, for a number of requests as low as 15 in some instance groups tested, solving the MILP formulation did not yield the optimal solution in the established running time of two hours.

In view of that, we decided to design and implement a heuristic approach to the SARP, such that good solutions could be obtained in tractable computational times. The heuristic is based on Iterated Local Search (ILS) [8], with two intra-route neighborhood operators and two inter-route neighborhood ones. We compared results to the ones obtained by exactly solving a MILP formulation for the SARP. We evaluate the algorithms performance both in terms of the solution values obtained as well as the computational efforts required for each run.

2 Share-a-ride problem (SARP)

2.1 Problem description

The SARP concerns the problem in which parcels and passengers share trips in taxis. In this work, we consider this joint transportation taking place in a ride-hailing system. Passenger service is considered as the main activity, therefore all requests of this type must be serviced and the associated

trips cannot have detours. Parcel deliveries, on the other hand, are optional, and therefore will be ignored if they have no economic benefits.

We define $G = (V, A)$ as a directed graph such that V represents the set of nodes and A represents the set of arcs. Set V is defined as $V = N \cup P \cup D \cup S$, such that $N = \{0, 1, ..., n-1\}$ is the set of customer service nodes, $P = \{n, n+1, ..., n+m-1\}$ and $D = \{n+m, n+m+1, ..., n+2m-1\}$ are respectively the sets of parcel pickup and delivery nodes. With these notations, n and m refer respectively to the number of passenger and parcel requests. Parcels picked up on $i \in P$ must be delivered to $i + m \in D$. Since passengers detours are not permitted, a passenger pickup is always followed by the passenger drop-off, and, therefore, these two activities can be represented as a single node

Ride-hailing vehicles are often privately owned and a driver can choose when to start the servicing of passengers and when to end it. Thus, we consider a multi-depot scenario, to account for vehicles starting journeys at different origins. Nonetheless, the driver does not need to return to his or her original location, after the last customer or parcel served. This is commonly known as an "open route".

A fleet of vehicles is represented by a set $K = \{0, 1, ..., \kappa\}$. Each vehicle $k \in K$ starts its trip at a starting location $s^k \in S$, such that $S = \{n + 2m, n + 2m + 1, ..., n + 2m + \kappa\}$. The vehicles also have a maximum driving time T. Vehicle capacities are taken into account through restrictions on the number of transported passengers and parcels imposed by our scenarios.

The set of arcs A is defined by trips that exist in any of the following occasions:

- From a vehicle origin location to a passenger request or parcel pickup location.
- Between pairs of different passenger requests.
- Between passenger requests and parcel pickup or delivery locations.
- From a parcel pickup or delivery location to any different pickup or delivery location.
- Between a passenger request or parcel delivery location to a dummy ending depot.

The arcs $(i, j) \in A$ have associated traveling times (t_{ij}), distances (d_{ij}) and costs (c_{ij}). The values of d_{ij} and t_{ij} have the following relationship: $d_{ij} = \nu t_{ij}$, in which ν is the average speed considered for the vehicles. The distance between pickup and drop-off locations in a passenger trip with no detours is represented by d_{ii} ($i \in N$). Distances d_{ii} follow the same relationship regarding traveling times and average speed as in the "non-collapsed" nodes.

Nodes $i \in V$ have a service time r_i, time windows $[e_i, l_i]$ and demand q_i associated to them. Service times for parcels represent the time taken to load and unload them from the vehicle.

For passengers, these times take into account not only time to board and leave the vehicle, but also the traveling time between their pickup and drop-off locations. For these types of requests, $q_i = 0$. Their time windows are considered as time points, such that $e_i = l_i$, and service should start at that time. For the remaining nodes, the time window is set to the complete planning horizon, meaning that service can occur at any time. We also define $r_i > 0$ ($i \in P \cup D$) and $q_i = -q_{i+m}$ ($i \in P$). For $i \in S$, $r_i = 0$, $q_i = 0$.

The objective in this problem is to maximize profit by transporting customers and parcels. To compute it, we use the parameters presented in Table 1.

Table 1: Parameters for profit calculation

Parameter	**Description**
γ_1	Initial fare charged for customer transportation
μ_1	Fare per km charged for customer transportation
γ_2	Initial fare charged for parcel transportation
μ_2	Fare per km charged for parcel transportation
μ_3	Average driving costs per km
ϕ_i	Revenues for transporting a customer $i \in N$
θ_i	Revenues for transporting a parcel $i \in P$.

For the calculation of traveling costs c_{ij}, we use $\mu_3 d_{ij}$. Costs of fuel, insurance and maintenance are included in μ_3, which is also used for calculating traveling costs within the "collapsed" customer nodes.

Moreover, the calculations for the revenue parameters are performed as follows: $\phi_i = \gamma_1 + (\mu_1 - \mu_3)d_{ii}$, $(i \in N)$, $\theta_i = \gamma_2 + \mu_2 d_{i,i+m}$, $(i \in P)$.

3 Mathematical formulation

We chose to model the problem as an open route. For that end, we created a set of "dummy" depots to be added to V and we define it as $F = \{n+2m+\kappa+1, n+2m+\kappa+2, ..., n+2m+2\kappa+1\}$, such that a vehicle $k \in K$ that starts its trip at point $s^k = n+2m+k$, ends it at $f^k = n+2m+\kappa+k+1$. Since these "dummy" depots exist only due to modelling decisions, the cost and time associated to arcs arriving at f^k, $k \in K$, is equal to zero. Furthermore, the service time and the demand associated to these nodes is also zero.

The variables in this model are the starting time of a service b_i, $i \in V$, the load of a vehicle after visiting $i \in V$, w_i, as well as the binary variables y_i^k and x_{ij}^k. The value of y_i is equal to 1 if $i \in V$ is serviced and 0, otherwise. Moreover, x_{ij}^k is 1 if arc $(i,j) \in A$ is traversed by vehicle k and 0, otherwise.

The model can be written as follows.

$$\max \sum_{i \in N} \sum_{j \in V} \sum_{k \in K} \phi_i x_{ij}^k + \sum_{i \in P} \sum_{j \in V} \sum_{k \in K} \theta_i x_{ij}^k - \sum_{(i,j) \in A} \sum_{k \in K} c_{ij} x_{ij}^k \tag{1}$$

$$\sum_{k \in K} \sum_{j \in V} x_{ij}^k = 1 \qquad i \in N \tag{2}$$

$$\sum_{k \in K} \sum_{j \in V} x_{ij}^k = y_i \qquad i \in P \tag{3}$$

$$\sum_{j \in V} x_{ij}^k = \sum_{j \in V} x_{(i+m,j)}^k \qquad i \in P, k \in K \tag{4}$$

$$\sum_{j \in V} x_{ij}^k = \sum_{j \in V} x_{ji}^k \qquad i \in N \cup P \cup D, k \in K \tag{5}$$

$$\sum_{j \in N \cup P} x_{s^k j}^k = \sum_{i \in N \cup D} x_{i f^k}^k = 1 \qquad k \in K \tag{6}$$

$$b_i \leq M y_i \qquad i \in V \tag{7}$$

$$b_i \leq b_{i+m} \qquad i \in P \tag{8}$$

$$b_j \geq b_i + r_i + t_{ij} - M \left(1 - \sum_{k \in K} x_{ij}^k\right) \qquad (i,j) \in A \tag{9}$$

$$e_i \leq b_i \leq l_i \qquad i \in V, k \in K \tag{10}$$

$$b_{f^k} - b_{s^k} \leq T \qquad k \in K \tag{11}$$

$$w_j \geq w_i + q_j - W \left(1 - \sum_{k \in K} x_{ij}^k\right) \qquad (i,j) \in A \tag{12}$$

$$x_{(i+m,i)}^k = 0 \qquad i \in P, k \in K \tag{13}$$

$$x_{ij}^k \in \{0,1\} \qquad (i,j) \in A, k \in K \tag{14}$$

$$y_i \in \{0,1\} \qquad i \in V \tag{15}$$

$$b_i \geq 0 \qquad i \in V \tag{16}$$

$$w_i \geq 0 \qquad i \in V. \tag{17}$$

The objective function is presented in (1). Constraints (2) state that all passenger nodes must be serviced. Constraints (3) connect the parcel selection and flow decisions, whereas Constraints (4) guarantee that a parcel that is picked up is delivered by the same vehicle. Flow conservation is ensured by Constraints (5). The origin and ending points of routes for each vehicle are determined by Constraints (6). Time constraints are specified in (7)–(11) while load constraints are described by

(12). Constraints (13) remove all arcs between a parcel delivery location and its pickup. Constraints (14)–(17), specify variables domains.

As all passengers must be serviced and no detours for these requests are allowed, we can remove the first term of (1) as a constant to be added to the solution value later. Hence, the objective function can be rewritten as follows.

$$\max \sum_{i \in P} \sum_{j \in V} \sum_{k \in K} \theta_i x_{ij}^k - \sum_{(i,j) \in A} \sum_{k \in K} c_{ij} x_{ij}^k \tag{18}$$

We make sure Constraints (7) and (9) are valid by setting M to the value of the end of the time horizon considered. For Constraints (11), the validity is ensured by defining $W = m + 1$.

4 Proposed algorithm

In this section, we discuss the steps of the heuristic proposed for the SARP. Algorithm 1 describes the pseudocode of the procedure. ILS_{SARP} receives input parameter I_{ILS}, which refer to the maximum number of consecutive method iterations without improvements.

ILS_{SARP} starts with the initialization of the objective function and solution (lines 2 and 3). An initial solution is generated (line 4) and it is then assigned to the current solution (line 5). Next, the ILS counter ($Iter_{ILS}$) is initialized with value zero (line 6). In lines 7 to 16, the ILS procedure is executed. A local search procedure is applied over the current solution (line 8), changing positions of requests within the same route. Next, the current solution goes through another round of local search (line 9), but this time, the requests can change positions and routes as well. The updated objective function and solution are stored if there is an improvement on the current solution (lines 11 and 12). $Iter_{ILS}$ is set to 0 if that condition is met (line 13). The ILS continues with the application of a perturbation mechanism over the best solution of the current iteration and it increases the value of $Iter_{ILS}$ by 1 unit (lines 15 and 16). The best solution is stored (line 18) and returned by the algorithm (line 19).

Algorithm 1 ILS_{SARP}

```
1: procedure ILS_SARP(I_ILS)
2:     f* ← ∞
3:     s* ← ∅
4:     s ← GenerateInitialRoute()
5:     s' ← s
6:     Iter_ILS ← 0
7:     while Iter_ILS ≤ I_ILS do
8:         s ← RVNDIntra(s)
9:         s ← RVNDInter(s)
10:        if f(s) > f(s') then
11:            s' ← s
12:            f(s') ← f(s)
13:            Iter_ILS ← 0
14:        end if
15:        s ← Perturbation(s')
16:        Iter_ILS ← Iter_ILS + 1
17:    end while
18:    s* ← s'
19:    return s*
20: end procedure
```

In what follows, the steps of the algorithm are more thoroughly explained. We give details on how an initial solution is generated, as well as how the local search procedures and the perturbation mechanism work.

4.1 Construction procedure

The construction procedure in this work uses greedy and randomized components. The greediness guarantees starting routes with promising edges in terms of costs. With the use of randomness, we can make sure different starting solutions are generated every time the method is executed.

In the first part of the construction procedure, we randomly select a number of passenger requests to be added to the solution. We set a value of 10% of the solving instance's passenger requests to be added in this way. At first, the solution starts with only one route, but new routes are created if the addition of the next passenger request renders the solution infeasible.

After the randomly chosen passengers are added, the remaining requests of this type are added to the solution using a greedy procedure, namely, the Cheapest Insertion method. As in the random phase, new routes are added to the solution to guarantee its feasibility and the addition of all passenger requests.

To finalize this phase, the parcel requests are added to the solution, using the Cheapest Insertion procedure. We make sure pickup and delivery nodes for each parcel request are added to the same route, and the precedence is not violated.

In all stages of the construction phase, we make sure the time point for each passenger is met with new additions to the routes. We also guarantee these types of requests are placed in the route according to the order of their time points. Furthermore, we keep route duration under a previously established maximum of eight hours.

4.2 Local search

In this work, the local search procedure proposed is based on the well-known Variable Neighborhood Descent (VND) [9], with random neighborhood ordering. The procedure is separated into two phases: intra-route and inter-route.

In the intra-route phase, we aim to improve each route of the solution, independently. We use two neighborhood structures to search for new positions for the requests already present in the route under improvement procedure, such that the overall cost of this route is lowered. For the inter-route phase, the requests can change routes. We do not only look for ways to reduce route costs but we also try to increase revenues by adding parcel requests that could still be out of the solution up to that point. In both phases, we also make sure no passenger time point, parcel precedence or maximum route duration is violated.

The pseudocode of the local search procedures is presented in Algorithm 2. For both the intra-route and inter-route phases, the procedure framework is the same, with a difference only in the types of neighborhood operators used.

While the neighborhood list is not empty, one neighborhood structure from the list is selected at random (line 4). The search is then performed using the selected neighborhood (lines 5–9) employing the best improvement strategy. If there is an improvement to the solution, all neighborhood operators of the list become available for the next iteration (lines 10–12). Otherwise, the previously explored neighborhood structure is removed from the list (line 14). The procedure ends when the list of neighborhoods is empty and the solution is not further improved.

Algorithm 2 Local Search

```
 1: procedure LocalSearch(s)
 2:     Initialize Neighborhood List (NL);
 3:     while NL is not empty do
 4:         Select a neighborhood from NL at random;
 5:         if N^(1) is selected then
 6:             Find the best neighbor s' of s ∈ N^(1)
 7:         else if N^(2) is selected then
 8:             Find the best neighbor s' of s ∈ N^(2)
 9:         end if
10:         if f(s') > f(s) then
11:             s ← s'
12:             Repopulate NL with all neighborhood structures
13:         else
14:             Remove selected neighborhood from NL
15:         end if
16:     end while
17:     return s;
18: end procedure
```

4.3 Neighborhoods

In this section, we present the neighborhood operators used in the `RVNDIntra` and `RVNDInter` procedures. All neighborhoods are implemented using the best improvement strategy.

- Intra-route:
  - **Swap:** Two requests are selected and their positions are exchanged.
  - **Relocate-Intra:** One request is selected and inserted in another position in the same route.
- Inter-route:
  - **Relocate-Inter:** One request is selected and inserted in the cheapest position in a different route.
  - **AddUnserved:** One unserved parcel request is selected to be inserted in the cheapest position between all routes of the solution.

For all neighborhoods, first we select one candidate to be moved in the solution or added to it, and we obtain all possible locations for insertion in a certain route. This list of possible positions is generated by assessing time and precedence constraints for the candidate, to make sure the movement is feasible.

From the available possible new locations, cost reductions (or revenues) are analyzed efficiently by calculating a δ. This δ is obtained by subtracting unused arc values and adding new arc values that represent the movement tested. After all of the possible movements are tested, the one that yields the highest cost reduction (or highest revenue) (i.e. the best δ) is performed.

4.4 Perturbation mechanism

To escape local optima and improve the method's search breadth, we devised a perturbation mechanism to change part of the current solution, after the local search phases.

The perturbation starts by randomly selecting 20% of customer requests and 30% of parcels to be removed from the solution. Next, the passengers are added to the solution again, in a random route, in a random but feasible location. If no feasible location is found in this phase, a new route is created to accommodate the passenger request.

The removed parcels are stored in a list to be added to the solution later, using the **AddUnserved** neighborhood.

5 Computational experiments

The heuristic algorithm was implemented using C++ (g++ 9.3.0) and the computational experiments were executed in an an Intel ®Core™ i7-10510U CPU with 1.80GHz processor and 16GB of RAM running Ubuntu Linux 20.04.

To assess performance of the heuristic method, the results obtained were compared to previously obtained objective function values and solution times from solving a MILP model. The model was implemented using C++, g++ 9.3.0, compiled with flag -O3.

The exact method experiments were conducted on a single thread of a server equipped with an Intel Xeon 2.0 GHz processor and 128 GB of RAM, running Ubuntu Linux 16.04. We used IBM CPLEX 12.7 to solve the model and established a limit of two hours for each run. The number of vehicles required for solving each instance is the minimum possible to achieve feasible solutions.

5.1 Instances used

We used a set of instances called MD that was previously proposed in our research. To create this set, we generated points in a grid in such a way that the obtained distances are consistent with a range of trip lengths previously calculated via Uber data. To pair these points as pickup and destination, we applied a method similar to the one proposed by [11]. At first, we took a random generated point as a pickup location for either parcel or customer and ranked the unassigned points

using proximity as a criterion. Following that, we chose the λ first points and select one of them at random to be the destination for this request. We used values of $\lambda = \{2, 5, \infty\}$. Four instances were generated for each size and pairing method used, which resulted in 12 instances for each combination of number of passenger and parcel requests.

The number of requests chosen for this set span from seven to nine passenger trips which are paired with five to seven parcels, but not exceeding a total number of 15 requests.

All instances are named according to the data set to where they belong, the number of customers and number of parcel requests, as well as its index, e.g. MD-7-4-1 is an instance of data set MD with 7 customers and 4 parcel requests, of index 1.

6 Results

To test the algorithm's performance, we selected 15 of the previously presented instances to solve and compare results with the exact method. We focused on instances which could not be solved to optimality with the established running time limit for the exact method.

For every instance, the heuristic method was executed 15 times and the best solutions were stored. Table 2 show the obtained results, separated by method. The column labeled **Instance** contains the names of the instances tested. For the exact method, we present the solution values obtained (**S. Val. (USD)**), the solving time required **Time(s)** and solution status (**Status**), which can either be optimal (O) or feasible (F).

For the heuristic method, we present the best found solution from all 15 trials (**BFS (USD)**), the solution values obtained and solving times, average between the 15 runs (**Avg. S. Val. (USD)**, **Avg. Time (s)**). And, lastly, we present the gap, in percentage, between the average solution values obtained and the best solution values found (either by the heuristic or the exact method).

Table 2: Solution values and solving times for exact and heuristic methods

Inst.	Exact			Heuristic			
	S. Val. (USD)	Time (s)	Status	BFS (USD)	Avg. S. Val. (USD)	Avg. Time (s)	Gap (%)
MD-7-5-1	**28.107**	283.71	O	**28.107**	24.274	0.0056	13.637
MD-7-5-4	91.514	7199.33	F	**98.057**	88.729	0.0051	2.839
MD-7-7-1	**148.322**	7199.13	F	**148.322**	138.685	0.0077	6.497
MD-7-7-2	**116.242**	7197.43	F	116.023	108.013	0.0072	7.079
MD-7-7-3	**133.430**	7196.47	F	**133.430**	128.090	0.0121	4.002
MD-7-7-4	94.904	7193.56	F	**97.028**	92.880	0.0071	2.086
MD-8-6-1	96.424	7197.95	F	**96.944**	90.188	0.0105	6.432
MD-8-6-2	**128.188**	7196.24	F	122.673	115.287	0.0086	10.064
MD-8-7-3	**136.693**	7193.93	F	135.520	125.849	0.0132	7.933
MD-8-7-4	**126.422**	7197.61	F	121.732	117.105	0.0101	7.370
MD-9-5-1	**122.453**	7196.68	F	**122.453**	112.810	0.0071	7.875
MD-9-5-4	110.449	7195.44	F	**113.534**	107.300	0.0084	2.774
MD-9-6-1	**157.065**	7195.54	F	154.681	148.633	0.0076	5.368
MD-9-6-2	**123.491**	7195.92	F	123.336	112.604	0.0096	8.816
MD-9-6-3	137.606	7194.38	F	**138.143**	134.505	0.0087	2.244

We can observe a meaningful reduction in solution times. For the exact method, only a feasible solution could be achieved for most instances, often arriving at the maximum running time of two hours. On the other hand, the heuristic method runs very fast, obtaining good solutions in under 0.02 seconds.

In terms of solution values, ILS_{SARP} was able to obtain the same or better results than the ones from solving MILP formulations, in 9 out of 15 instances. However, the heuristic does not seem to be consistent, which is visible when analyzing the average solution values obtained and the gap between them and the best solution values found so far.

7 Conclusion and Future Work

We proposed a heuristic algorithm for the SARP. We used ILS as the framework for this heuristic, with two intra-route neighborhood operators (Swap and Relocate-Intra) and two inter-route ones (Relocate-Inter and AddUnserved). The method is called ILS_{SARP}.

ILS_{SARP} starts with a construction procedure that has a random phase and a greedy phase. The local search phase uses a randomized version of the variable neighborhood descent (VND) as basis for choosing the next neighborhood structure to be tested.

We also presented a perturbation mechanism in which we remove passengers and parcels, adding back the passengers, but not the parcels at that stage.

We compared results to the exact method and we could observe significant gains in solving time, with each instance being solved in under 0.02 seconds. Solution value gaps were still high, with values ranging from 2.24% to 13%, even though the method was able to find equal or better solutions than the ones obtained with the exact method in 9 of the 15 instances.

As a preliminary analysis of this heuristic's performance in the context of the SARP, we can conclude it is a good strategy and, with further improvements, we can even improve scalability and solve much larger instances to obtain close to optimal values.

The performance of the method can be improved with the addition of new neighborhood operators and more efficient feasibility check functions. Another improvement to the heuristic can be the use of a memory structure, such that repeated solutions are avoided. Furthermore, we can add a multi-start component to the ILS_{SARP} procedure, and return only the best solution found after a certain number of runs.

References

[1] Baugh Jr., J. W., Kakivaya, G. K. R., and Stone, J. R. "Intractability of the dial-a-ride problem and a multiobjective solution using simulated annealing". In: *Engineering Optimization* 30.2 (1998), pp. 91–123.

[2] Columbia Broadcasting System. *Amazon's Jeff Bezos looks to the future*. 2013. URL: https://www.cbsnews.com/news/amazons-jeff-bezos-looks-to-the-future/ (visited on 08/09/2019).

[3] Convey. *Shoppers have a need for speed (not just free shipping) this holiday season*. 2019. URL: https://www.getconvey.com/resource/press-amazon-effect-shipping-peak-season/ (visited on 11/17/2020).

[4] Cordeau, J.-F. and Laporte, G. "The dial-a-ride problem (DARP): Variants, modeling issues and algorithms". In: *Quarterly Journal of the Belgian, French and Italian Operations Research Societies* 1.2 (2003), pp. 89–101.

[5] Graham, D. J. and Gibbons, S. "Quantifying wider economic impacts of agglomeration for transport appraisal: Existing evidence and future directions". In: *Economics of Transportation* 19 (2019), p. 100121. DOI: https://doi.org/10.1016/j.ecotra.2019.100121. URL: https://www.sciencedirect.com/science/article/pii/S2212012218300856.

[6] International Post Corporation. *Global post industry report 2020: Key findings*. 2021.

[7] Li, B. et al. "The share-a-ride problem: people and parcels sharing taxis". In: *European Journal of Operational Research* 238.1 (2014), pp. 31–40.

[8] Lourenço, H. R., Martin, O. C., and Stützle, T. "Iterated Local Search: Framework and Applications". In: *Handbook of Metaheuristics*. Boston, MA: Springer US, 2010. Chap. 12, pp. 363–397.

[9] Mladenović, N. and Hansen, P. "Variable neighborhood search". In: *Computers & Operations Research* 24.11 (1997), pp. 1097–1100.

[10] Pitney Bowes Inc. *Press release - Pitney Bowes parcel shipping index reports*. 2020. URL: http://news.pb.com/article_display.cfm?article_id=5958 (visited on 01/20/2021).

[11] Renaud, J., Boctor, F. F., and Ouenniche, J. "A heuristic for the pickup and delivery traveling salesman problem". In: *Computers & Operations Research* 27.9 (2000), pp. 905–916.