2019 IEEE 6th International Conference on Industrial Engineering and Applications

An Adaptive Large Neighborhood Search Heuristic for Dynamic Ridesharing Problem

Shijia Hua
Department of Industrial Engineering
Tsinghua University
Beijing, China
e-mail: hsjhsjzju@hotmail.com

Mingyao Qi
Research Center on Modern Logistics
Graduate School at Shenzhen, Tsinghua University
Shenzhen, China
e-mail: qimy@sz.tsinghua.edu.cn

Abstract—**Ridesharing is getting more and more attention in the present society with several advantages in reducing congestion and minimizing the impact of transportation on the environment. In this paper, we consider a ridesharing problem on how to assign passengers to vehicles optimally. Unlike the others in the literature, this paper considers the ridesharing problem with previously scheduled routes and introduces rescheduling ratio into the problem. To solve the proposed problem, an adaptive large neighborhood search heuristic with a number of newly proposed operators is presented. Numerical experiments demonstrate its efficiency compared with commonly used heuristics. Furthermore, we discuss the effect of rescheduling ratio and present recommendations for the on-line taxi-dispatch systems.**

Keywords-component; dynamic; ridesharing; adaptive large neighborhood search

I. INTRODUCTION

Recently, the traffic jam problem is a major challenge encountered by cities all over the world, which has also brought about a series of social, environmental and economic problems. Ridesharing is a mode of transportation which allows people with similar itineraries and schedules to share a car and thus reduces the number of vehicles required. Ridesharing emerges as an efficient way to reduce traffic congestion and is becoming common in several cities around the world.

Several variants of multiple decision-making problems have arisen along with the development of ridesharing platforms, most of which are related to the development and optimization of matching algorithms between drivers and passengers. Agatza [1] undertook a review mainly on the optimization models of the ride-matching problem and concluded that the transportation area is far from mature. Furuhata [2] pointed out the existing challenges, most of which were about passengers-matching scenarios.

From the perspective of the methods adopted, the ridesharing problem can be modeled as the dynamic Dial-A-Ride Problem (DARP) of multiple cars. Research on the problems of DARP and PDPTW (pick-up and delivery problems with time windows) dates back to 1980. Psaraftis [3] proposed a dynamic programming method to solve the small-scale PDPTW problem of a single vehicle. Exact and heuristic algorithms have been further emphasized and extended from then on [4-7]. Hosni [8] formulated the shared-taxi problem as a general mixed integer programming where vehicles can be at different locations and can have passengers onboard. To solve the model, they proposed a Lagrangian decomposition approach and two heuristics. Ropke and Pisinger [9] presented an adaptive large neighborhood search (ALNS) heuristic for the traditional PDPTW.

In this paper, we investigate the dynamic ridesharing problem extended from the problem discussed in Hosni [8]. Vehicles with onboard and scheduled passengers together with previously planned routes are considered. When new requests come into the ridesharing system, algorithms are used to match the passengers and drivers with the goal of maximizing the total profit. An adaptive large neighborhood search heuristic with several new operators (revised Shaw removal, random insertion, and sequential best insertion heuristics) is proposed to solve the problem. We also introduce a rescheduling ratio and discuss the balance between the total profit and passengers' convenience when rearranging the already scheduled passengers. Numerical experiments show that the ALNS algorithm performs better than the commonly used heuristics. The revenue loss is less than 20% when not rearrange the scheduled requests.

The main contributions of this paper are: First, the dynamic ridesharing problem with previously scheduled routes of vehicles is carefully studied together with the rescheduling ratio introduced for the first time; Second, an adaptive large neighborhood search heuristic with several new operators is proposed to get feasible and high-quality solution; Thirdly, study the effect of rescheduling ratio and present recommendations for the on-line taxi-dispatch systems.

II. PROBLEM DESCRIPTION

We consider the ridesharing problem in a dynamic environment where new passengers and drivers continuously enter and leave the system. The algorithm is repeatedly called after a fixed interval of time. Therefore, the problem can be described as follows:

Three types of requests are introduced: (1) onboard requests: the passengers of the requests have already been picked up by vehicles, consequently the onboard passengers must be delivered by particular vehicles; (2) scheduled requests: the passengers have been assigned to vehicles in the last computation but not picked up yet. Thus they could be rescheduled for better revenue, but with a cost of reducing the comfort for both passengers and drivers. In this paper, a rescheduling ratio α is introduced to adjust the proportion of rescheduled requests among the total scheduled requests. (3) new requests: the passengers who enter the system during the current time interval.

Each request is associated with an earliest pickup time from the corresponding pickup location and a latest drop-off time from the corresponding drop-off location. Vehicles enter the system with different locations, capacities and can have associated onboard and scheduled passengers. Therefore, a certain percentage (α) of scheduled requests as well as the new requests need to be planned. The onboard and remaining scheduled requests stay the same.

The objective function is to maximize the total profit. Given a directed graph $G=(V,A)$ where V is the set of vertices and A is the set of arcs. Requests are partitioned into fixed requests set F and unfixed requests set S . Each request yields a revenue R_P which is based on the distance between pickup and delivery locations. A vehicle travels from node i to node j with a cost c_{ij} . We define two binary variables as follows:

x_{ij}=1 if arc(i , j) is traveled, and 0 otherwise.

d_p=1 if request p is arranged, and 0 otherwise.

The objective function is:

$$\max \sum_{p\in S} R_P d_p - \sum_{i\in V}\sum_{j\in V} c_{ij} x_{ij} \tag{1}$$

We also need to consider the following constraints: (1) vehicle capacity; (2) the time window of requests; (3) the fixed requests must be visited by associated vehicles.

III. Algorithm Design

Since the proposed ridesharing problem is NP-hard, ALNS heuristic is developed to deal with this problem in this section. This heuristic is composed of a number of competing sub-heuristics that are used with a frequency corresponding to their historic performance [9]. The pseudocode of the ALNS heuristic is shown in Table 1.

A. Initial Solution

We obtain the initial solution using the nearest vehicle dispatch [10] algorithm. The requests are processed individually in turn. Firstly, sort the vehicles in ascending order according to the distance between the location of an available vehicle and the pickup location of the request. Secondly, insert the request into the current path of the vehicle with its optimal insertion position in sequence. If there are no available routes for the request, then test the next vehicle until finding a feasible vehicle or reject the request. Once the request has been inserted, turn to the next request.

We also try another insertion heuristic [10] as the initial solution. It turns out that the ALNS algorithm has relatively little dependence on the initial solution, especially on a large scale problem. Therefore, we use the nearest vehicle dispatch as initial solution technique for its time-saving benefit.

TABLE I. ALNS Framework

```
Algorithm 1: ALNS Heuristic
Function ALNS( $s_{curr}$ ∈ {solutions},
        $parameters[N, M, \delta, p, p_{worst}, \psi, \kappa, \omega, \sigma_1, \sigma_2, \sigma_3, r, \phi, c]$ )
  Initialize: set all weights equal and all scores to 0
  solution $s_{best} \leftarrow s_{curr}$ , $iterationTime = 0$ , $notImprovedTime = 0$
  repeat
    using the roulette-wheel to select a removal operator
    using the roulette-wheel to select an insertion operator
    update the usage counter of operators
    apply the operator pair, obtain a new solution $s_{new}$
    if $F(s_{new}) > F(s_{best})$ then
      $s_{best} \leftarrow s_{curr} \leftarrow s_{new}$ , $notImprovedTime$=0
      Update the score of operator pair using $\sigma_1$
    else if $F(s_{new}) > F(s_{curr})$ then
      $s_{curr} \leftarrow s_{new}$ , $notImprovedTime++$
      Update the score of operator pair using $\sigma_2$
    else
      if $s_{new}$ is accepted by the simulated annealing criterion then
        $s_{curr} \leftarrow s_{new}$ , $notImprovedTime++$
        Update the score of operator pair using $\sigma_3$
    end if
    if $(iterationTime+1)\%100 == 0$ // update every 100 times
      Update the weight and score for all operators
    end if
    $iterationTime++$
  until $iterationTime >$ N || $notImprovedTime > M$ //Stop criteria
  Return $s_{best}$
```

B. Removal Heuristics

This section describes four removal heuristics. After we gain a solution, removal heuristics ruin the current solution by removing a number of planned requests of the current solution. We control the number of requests to be removed q by a parameter δ . We choose a random number between [0, max{ δn , 2}) (n is the number of planned requests in the current solution and if $n<1$ then turn to the insertion part directly) each iteration. The pseudocode of the generic removal heuristics is shown in Table 2.

1) *Random removal*: There is no rank role and p is set to 1. The requests are randomly chosen and then removed from the current solution.
2) *Worst removal:* This operator aims to remove the request with expensive insertion cost. Therefore, in this paper, R is sorted by descending
$$c_m = P(r) - P_{-m}(r) \tag{2}$$
where r is the current route of request m , $P_{-m}(r)$ is the distance of the route without request m (but not removed from the solution). The parameter p in worst removal is set to p_{worst} .
3) *Shaw removal:* This operator expects to remove requests that are relatively similar to each other, so as to create a better solution [11, 12]. Different from the previous two removal operators, Shaw removal randomly choose a request i from D first and then sort R by the ascending $L(i,j), j\in R$. In this paper, we calculate the similarity level $L(i,j)$ as follows:

TABLE II. GENERIC REMOVAL HEURISTICS FRAMEWORK

Algorithm 2: Generic Removal Heuristics

```
1  Function Generic Removal ( $s \in$ {solutions}, $q \in [0, \max\{\delta n, 2\}), p$ )
2    Initialize: obtain all the planned requests $R$
           set of removal requests: $D$
3    while $|D| < q$ do
4      Sort $R$ according to some rule
5      choose a random number $a \in [0,1)$
6      select request $r = R[a^p |R|]$
7      $D = D \cup r$
8      remove $r$ from $R$
9    end while
10   remove the requests in $D$ from $s$
11   Return
```

$$L(i,j) = \psi(d_{P(i),P(j)} + d_{D(i),D(j)}) + \kappa(|T_{P(i)} - T_{P(j)}| + |T_{D(i)} - T_{D(j)}|) + \omega\left(1 - \frac{|K_1 \cap K_2|}{\min\{|K_1|, |K_2|\}}\right) \tag{3}$$

$P(i)$ and $D(i)$ denote the pickup and delivery nodes of request i . $K(i)$ is the set of vehicles that are able to serve request i .Thus the first two items measure the location and time window difference of two requests and the last one measures the vehicles that are able to serve both requests.

4) *Revised Shaw removal:* The overall logical of this operator is similar to the Shaw Removal. But we use a different way to calculate the similarity level of two requests. It uses the matching level concept in [13]:

$$O(i,j) = 0.5 \times \frac{\max\{d_i, d_j\}}{d(r_{ij})} + 0.5 \times \frac{\min\{d_i, d_j\}}{d(r_{ij})} \tag{4}$$

d_i denotes the distance between the pickup location and delivery location of request i . $d(r_{ij})$ denotes the shortest feasible route connects request i and j . Furthermore, $O(i,j) \in (0,1)$, the closer value to 1 means the closer relationship between request i and j .

C. Insertion Heuristics

This section describes four insertion heuristics which reinsert the requests into the current solution.

1) *Random insertion*: Passengers are randomly selected and inserted in this case. This heuristic expects to diversify the search and escape the local optima.
2) *Parallel greedy insertion*: It inserts one request in each iteration. We concurrently calculate the minimal insertion cost of all unplanned requests and choose the smallest one to insert at its minimum cost position until all requests have been inserted or no more requests can be inserted. The calculation method is shown below,

$$c_{m,r} = P_{+m}(r) - P(r) \qquad c_m = \min_{r \in allRoutes} c_{m,r} \tag{5}$$

$c_{m,r}$ denotes the insertion cost of request m into route r .

3) *Parallel regret insertion*: In this paper, we use the regret-2 heuristic. The regret heuristic selects the request based on the regret value which considers both the best and the second-best placement and chooses the request with the maximal difference value.

$$c_m = c_{m,r2} - c_{m,r1} \tag{6}$$

c_m denotes the regret value for request m , r_i denotes the i^{th} lowest insertion cost.

4) *Sequential best insertion*: This heuristic processes the requests in sequence and for each request chooses the vehicle with minimal insertion cost.

D. Choosing a Removal and an Insertion Heuristic

We use a roulette-wheel selection principle to select either the removal or the insertion heuristics which are assigned with different weights.

E. Adaptive Weight Adjustment

In this paper, we divide the entire search into several segments (100 iterations) and update the weights according to the scores at the end of segments. The score of a heuristic is increased with $\sigma_1, \sigma_2, \sigma_3$, depending on the situations as follows:

σ_1 , if the new solution results in a new global best solution;

σ_2 , if the new solution is better than the current one;

σ_3 , if the new solution is worse than the current one but still accepted;

We update the weight for heuristic i as follows:

$$w_i = w_i(1-r) + r\frac{\Pi_i}{\theta_i} \tag{7}$$

where Π_i is the score of heuristic i , and θ_i is the used times of heuristic i during the last segment.

F. Acceptance and Stopping Criteria

We use the simulated annealing as the acceptance criteria. The solution is accepted with probability:

$$p(s_{new}, s_{curr}) = \min\{e^{(s_{new} - s_{curr})/T_k}, 1\} \tag{8}$$

where $T_k = T_0 c^k$, denote the temperature that decreases in each iteration. And we reset the initial temperature each iteration as:

$$T_0 = -\phi |s_{curr}| / \ln 0.5 \tag{9}$$

The algorithm stops when a particular number of iterations have passed or the solution does not improve in a number of successive iterations.

IV. RESULTS AND ANALYSIS

This section briefly describes how the test instances were generated, followed by experiments results of the new proposed algorithm compared with two commonly used heuristics. All of the computational tests presented here are performed on an Intel(R) Core(TM) i5-6300 CPU @ 2.40GHz. Then we analyze the effect of rescheduling ratio on the modified objective function and raise some suggestion for the online-taxi platform.

A. Experiment Description and Parameter Settings

The instances are extended from [8]. We generated the sets consist of vehicles, onboard requests, scheduled requests, and new requests. The coordinates of the locations of vehicles as well as the pickup and delivery locations were randomly generated in the square $[0,20]\times[0,20]$ with a uniform distribution. The distance between two locations is assumed to be the Euclidean distance, and the vehicle speed is set to 1 for all the vehicles. The cost equals the distance and we define the revenue of a request on the basis of the pickup and delivery locations of a request as $R_p = 10 + 2.5 \times d(p,d)$, where $d(p,d)$ is the distance between pickup and delivery locations. The vehicle capacity is set to 2 for all vehicles. The instance are divided into two part: Sets R are instances with relaxed time windows while Sets T with tight time windows. Suppose the starting time is 0, and we randomly generated the earliest pickup time for onboard and scheduled requests from the uniform distribution $u(0,5)$ while the new requests from the uniform distribution $u(5,10)$. For instance with relaxed time window, the latest delivery time is set to be $latestDeliveryTime = earliestPickupTime + time(p,d) + u(5,30)$ where $time(p,d)$ denotes the time from pickup location to delivery location. For instance with tight time window, the uniform distribution turns into $u(0,15)$. We planned the scheduled requests randomly to the vehicles satisfying all the constraints. The name of instance $p\_n\_m\_s\_k$ denotes a problem with n vehicles, m onboard requests, s scheduled requests and k new requests.

For the generated instances, we tested the nearest vehicle dispatch(NVD) heuristic, the best insertion(BIS) heuristic and the ALNS heuristic. NVD is the most widely employed strategy for on-line taxi-dispatch systems [10]. The parameters for ALNS heuristic are set as follows:

$$[N, M, \delta, p, p_{worst}, \psi, \kappa, \omega, \sigma_1, \sigma_2, \sigma_3, r, \phi, c] = [2500,100,0.4,6,3,9,3,5,33,9,13,0.1,0.05,0.99975]$$

B. Comparison with Existing Approaches

To assess the performance of ALNS, we run the proposed algorithm using the generated instances compared with existing techniques NVD and BIS. The computational results are summarized in Table III. Row "Set" refers to the type of time window (R/T) and the figure denotes different vehicle-request ratio. The columns "OBJ" and "Time" respectively represent the objective function value and the CPU time running the algorithm in milliseconds. We set rescheduling ratio to 1 for all algorithms, which means all the scheduled requests will be rearranged during the process.

The results of the test instances show that ALNS has a better optimal value than NVD and BIS both on the instances with tight and relaxed time window. The advantage is more obvious under large instances. However, the ALNS need more time to get a feasible solution for the reason of more iterations and time-consuming operators like Shaw removal heuristic.

TABLE III. COMPARISON BETWEEN ALNS WITH NVD AND BIS

Name	Set	NVD		BIS		ALNS		Set	NVD		BIS		ALNS	
		OBJ	Time	OBJ	Time	OBJ	Time		OBJ	Time	OBJ	Time	OBJ	Time
p_2_1_1_1		36.36	1	36.36	1	**36.36**	57		17.26	1	17.26	2	**17.26**	193
p_4_2_2_2		73.59	1	65.81	1	**73.59**	40		68.83	1	82.92	1	**82.92**	176
p_6_3_3_3		102.59	1	**107.06**	1	106.94	98		20.22	1	20.22	1	**25.89**	86
p_8_4_4_4		152.21	1	154.04	1	**157.65**	91		79.78	1	79.63	2	**79.78**	95
p_10_5_5_5	R1	120.64	1	151.45	1	**163.69**	113	T1	142.88	3	153.93	2	**154.93**	178
p_12_6_6_6		177.48	1	185.94	5	**199.06**	149		126.74	1	151.04	2	**159.12**	119
p_14_7_7_7		204.40	3	228.81	2	**247.48**	218		168.94	1	168.94	2	**185.42**	196
p_16_8_8_8		272.43	1	294.75	1	**308.81**	163		263.24	1	274.32	2	**285.96**	174
p_18_9_9_9		313.69	1	366.55	4	**368.50**	228		302.88	1	340.34	3	**352.10**	246
p_20_10_10_10		319.03	1	357.99	2	**381.66**	302		282.63	1	**322.18**	3	319.57	259
p_2_1_2_2		52.52	8	52.52	6	**52.52**	281		80.34	1	80.34	1	**80.34**	175
p_4_2_4_4		63.08	1	63.08	2	**63.08**	226		171.79	1	166.10	1	**171.79**	155
p_6_3_6_6		191.38	1	192.90	3	**212.82**	335		77.05	1	77.05	1	**87.52**	299
p_8_4_8_8		234.68	4	259.00	4	**354.65**	225		163.96	2	263.77	2	**282.91**	395
p_10_5_10_10	R2	355.78	2	375.26	7	**416.15**	305	T2	181.44	2	184.64	3	**230.91**	550
p_12_6_12_12		447.68	2	443.74	5	**540.25**	528		354.25	2	342.34	3	**447.62**	841
p_14_7_14_14		506.20	2	541.65	8	**595.28**	610		357.05	3	363.83	7	**415.66**	1556
p_16_8_16_16		551.87	2	597.59	10	**661.11**	1203		352.00	3	342.65	4	**472.97**	1533
p_18_9_18_18		718.97	2	789.21	14	**911.04**	1460		394.85	5	449.50	8	**594.85**	2938
p_20_10_20_20		828.01	3	886.79	17	**967.12**	2159		685.54	9	748.11	11	**801.95**	2260
p_2_1_2_4		67.52	1	67.52	2	**67.52**	219		66.36	2	66.36	3	**66.36**	237
p_4_2_4_8		180.14	2	189.34	2	**209.18**	454		109.40	1	99.74	1	**123.99**	362
p_6_3_6_12		259.79	2	246.83	4	**307.51**	660		148.04	1	181.28	3	**222.94**	794
p_8_4_8_16		374.71	3	393.95	5	**504.80**	717		155.16	2	177.69	4	**214.26**	1431
p_10_5_10_20	R3	523.19	4	479.99	6	**618.70**	1521	T3	290.06	5	388.78	15	**436.76**	1567
p_12_6_12_24		510.13	7	528.79	16	**693.21**	2051		457.99	4	432.85	7	**569.02**	3076
p_14_7_14_28		694.50	10	664.46	17	**828.30**	3183		491.14	10	415.44	11	**559.50**	5586
p_16_8_16_32		857.38	8	908.07	26	**1113.77**	4361		621.66	9	653.50	15	**742.39**	6853
p_18_9_18_36		1061.17	12	1164.03	31	**1288.07**	7218		592.29	13	717.56	21	**855.81**	11752
p_20_10_20_40		1111.91	13	1202.08	36	**1323.41**	8786		787.40	14	773.41	24	**939.62**	15688

C. The Effect of Rescheduling Ratio

Since we often receive information about the driver after calling a taxi in real life, changing drivers may cause frustration among passengers. Besides, rearranging the scheduled request will increase the size of the problem and thus need more time to obtain a solution.

In order to measure the effect of rescheduling ratio, we set the rescheduling ratio to be 0, 0.2, 0.4, 0.6, 0.8, and 1 for one instance. We modified the objective function for the comparability between different cases as follows:

$$\min \sum_{i\in V}\sum_{j\in V} c_{ij}x_{ij} + \sum_{p\in U} R_P d_p \tag{10}$$

where U is the unscheduled requests after the process.

We chose the small instance p_10_5_5_5 and the largest instance p_20_10_20_40 with relaxed and tight time window respectively. The results are shown in Figure 1 and Figure 2.

As rescheduling ratio increases, the modified objective function declines with fluctuation because of the instability of heuristic algorithms. But the decrease ratio between 0 and 1 is less than 20% steadily using heuristics. Thus the online taxi-dispatch platform can seek a balance between cost and the comfort of passengers. Since the revenue loss between 0.8 and 1 is small among all the tests, at least the passenger that the driver heads for cannot be rearranged for the convenience of both the passenger and the driver. Or the online taxi-dispatch systems can consider picking a portion of the orders rescheduled in chronological order.

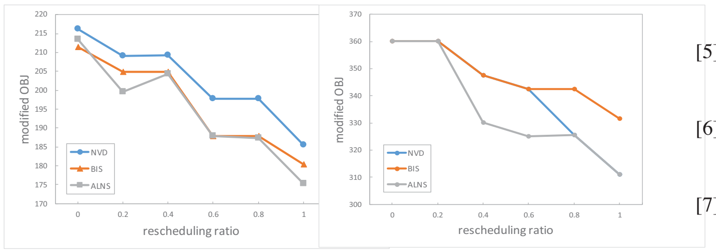

Figure 1. Results for instance p_10_5_5_5 with relaxed and tight time window respectively with different rescheduling ratio.

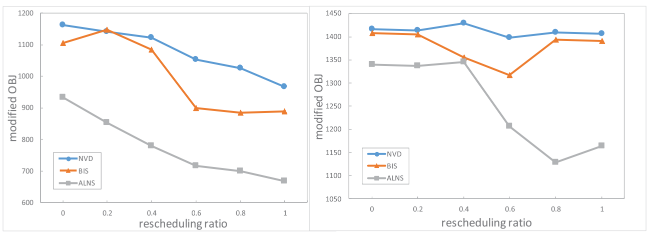

Figure 2. Results for instance p_20_10_20_40 with relaxed and tight time window respectively with different rescheduling ratio.

V. Conclusion

In this paper, a dynamic ridesharing problem with previously scheduled routes of vehicles is studied. What is more, rescheduling ratio is introduced to measure the rearranged ratio of previously scheduled requests. To solve this problem, we propose an adaptive large neighborhood search heuristic with several new operators (revised Shaw removal, random insertion, and sequential best insertion heuristics). Numerical experiments demonstrate the quality of the proposed ALNS which outperformed NVD and BIS. What is more, the effect of rescheduling ratio is discussed and the decrease ratio between 0 and 1 is less than 20% steadily using heuristics, which recommends the on-line taxi-dispatch systems consider picking a portion of the scheduled requests not rearranged.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (71772100), National Key Technologies Research and Development Program: (2016YFB0502601), and Shenzhen Science and Technology Project (JCYJ20170412171044606).

References

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

[2] Furuhata M, Dessouky M, Ordóñez F, et al. Ridesharing: The state-of-the-art and future directions[J]. Transportation Research Part B Methodological, 2013, 57(57):28-46.

[3] Psaraftis, H.N., 1980. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. Transp. Sci. 14 (2), 130–154

[4] Baldacci, R., Maniezzo, V., Mingozzi, A., 2004. An exact method for the car pooling problem based on Lagrangean column generation. Oper. Res. 52, 422–439.

[5] Ropke, S., Cordeau, J.F., 2009. Branch and cut and price for the pick-up and delivery problem with time windows. Transp. Sci. 43, 267–286.

[6] Nanry, W.P., Barnes, J.W., 2000. Solving the pick-up and delivery problem with time windows using reactive tabu search. Transp. Res. Part B: Methodol. 34 (2), 107–121.

[7] Bent, R., Van Hentenryck, P., 2006. A two-stage hybrid algorithm for pick-up and delivery vehicle routing problems with time windows. Comput. Oper. Res. 33 (4), 875–893.

[8] Hosni H, Naoum-Sawaya J, Artail H. The shared-taxi problem: Formulation and solution methods[J]. Transportation Research Part B, 2014, 70(C):303-318.

[9] Ropke, S., Pisinger, D., 2006. An adaptive large neighbourhood search heuristic for the pick-up and delivery problem with time windows. Transp. Sci. 40 (4), 455–472.

[10] Jung J, Jayakrishnan R, Ji Y P. Dynamic Shared-Taxi Dispatch Algorithm with Hybrid-Simulated Annealing[M]. John Wiley & Sons, Inc. 2016.

[11] Shaw, P. 1997. A new local search algorithm providing high quality solutions to vehicle routing problems. Technical report, Department of Computer Science, University of Strathclyde, Scotland.

[12] Shaw, P. 1998. Using constraint programming and local search methods to solve vehicle routing problems. Proc. CP-98 (Fourth Internat. Conf. Principles Practice Constraint Programming).

[13] Hou L, Li D, Zhang D. Ride-matching and routing optimisation: Models and a large neighbourhood search heuristic. Transportation Research Part E: Logistics and Transportation Review. 2018 Oct 1;118:143-62.